University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Marko Đurasević

AUTOMATED DESIGN OF DISPATCHING RULES IN UNRELATED MACHINES ENVIRONMENT

DOCTORAL THESIS

Zagreb, 2018

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Marko Đurasević

AUTOMATED DESIGN OF DISPATCHING RULES IN UNRELATED MACHINES ENVIRONMENT

DOCTORAL THESIS

Supervisor: Professor Domagoj Jakobović, Ph.D.

Zagreb, 2018

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Marko Đurasević

AUTOMATIZIRANO OBLIKOVANJE PRAVILA RASPOREĐIVANJA U OKOLINI NESRODNIH STROJEVA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Domagoj Jakobović

Zagreb, 2018.

Doctoral thesis has been made at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Electronics, Microelectronics, Computer and Intelligent Systems.

Mentor: Professor Domagoj Jakobović, Ph.D.

Doctoral thesis has: 467 pages

Number of doctoral thesis: ___________

About the Supervisor:

Domagoj Jakobović was born in Našice in 1973. He received B.Sc., M.Sc. and Ph.D. degrees in computer science from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1996, 2001 and 2005, respectively.

From April 1997 he is working at the Department of electronics, microelectronics, computer and intelligent systems at FER. In January 2018 he was promoted to Full Professor. He led six scientific projects and was included in several domestic and international projects. Currently he is a principal investigator of the project "EvoCrypt" financed by the Croatian science foundation. He published more than 80 papers in journals and conference proceedings in the area of application of stochastic optimization and machine learning in scheduling and cryptography, as well as development of parallel evolutionary algorithms.

Prof. Jakobović is a member of IEEE and ACM. He is member of a journal editorial board and he serves as a technical reviewer for various international journals.

O mentoru:

Domagoj Jakobović rođen je u Našicama 1973. godine. Diplomirao je, magistrirao i doktorirao u polju računarstva na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1996., 2001. odnosno 2005. godine.

Od travnja 1997. godine radi na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave FER-a. U siječnju 2018. godine izabran je u zvanje redovitog profesora. Vodio je šest znanstvenih projekata i bio suradnik na nekoliko domaćih i međunarodnih projekata. Trenutno je voditelj projekta "EvoCrypt" koji financira Hrvatska zaklada za znanost. Objavio je više od 80 radova u časopisima i zbornicima konferencija u području primjene stohastičke optimizacije i strojnog učenja u problemima raspoređivanja i kriptografiji te razvoja paralelnih evolucijskih algoritama.

Prof. Jakobović član je stručnih udruga IEEE i ACM. Član je uredničkog odbora znanstvenog časopisa te sudjeluje kao recenzent u desetak inozemnih časopisa.

Dedicated to my parents and grandparents for all their love and support.

Acknowledgements

I would like to take this opportunity to express my gratitude and thank all the people who helped me throughout my life, and without which I would not be able to reach this goal and complete my thesis. This thesis is not only my own personal success, but rather the success of all the people who supported me during all the years.

First of all I want to thank my parents, Jasna and Dubravko, who supported me during my entire life, and did everything they could in order to allow me to focus on my studies and finish my thesis. Thank you for all the love and care you gave me through life. Thank you for all the sacrifices you made for me. Thank you for all the support in hard and difficult times. Thank you for motivating me to pursue my dreams however hard it sometimes might have been. Thank you for everything you did for me. I also want to thank my grandparents, Stjepan and Pepica, for the wonderful childhood and for all the time they spent with me, I will forever treasure those moments. Thank you for all the talks and stories you told me, which helped me to take my mind of the many difficulties I faced. Thank you for all your love and support, and for always encouraging me to do my best and never give up. You always wished you could do more to help me, but you helped me more than you could ever imagine.

I would like to thank my mentor, Professor Domagoj Jakobović, who supported and mentored me throughout all my studies. I want to thank him for all the time and effort that he invested in me, all the talks and discussions we had, as well as all the help and understanding that he provided. He was always available to me whenever I had questions or problems, and would do his best to help me in any way he could. Without his guidance I would have never been able to complete my thesis in such a small period od time. I hope that one day I will be an equally good mentor as he is. In addition, I want to thank him for recommending me as a teaching assistant, and thus allowing me to work on what I love the most.

I also want to thank Professor Željka Mihajlović, who undertook a great effort in making it possible that I stay at the faculty as a teaching and research assistant. She was always available to me if I had some problems or questions, and always did her best to help me out. Her support was always valuable, especially in the first years of my employment at the faculty. With her help it was possible that I also continue collaborating with AVL-AST on interesting projects even after my full employment at the faculty, for which I am deeply thankful.

I want thank all the colleagues from the Department of Electronics, Microelectronics, Computer and Intelligent Systems, who were always supportive and helped out whenever needed. It was always an enjoyment to work and collaborate with you during these last several years.

I also want to thank all other professors who motivated me to pursue a career in teaching and research. I would especially like to thank Associate Professor Josip Knezović, with whom I have collaborated a lot during my studies. He had a big influence on me, since from the beginning of my studies he motivated me to read, learn new things, and work on projects outside the regular courses, which broadened my knowledge and made me a better researcher. I always regarded him as a second mentor, since he was always available for me when I needed help, especially when submitting the work for the Rector's award, where he spent several sleepless nights working with us on preparing the submission. Without his help I would never have had the confidence to pursue a PhD degree.

Furthermore, I want to extend my thanks to Goran Mirković, the director of AVL-AST. He made it possible that I stay at the faculty as a teaching assistant by financing a part of my salary. In addition, with his support it was also possible to extend the collaboration even after I became a full employee of the faculty, so that I can continue working on many interesting topics and projects. I also want to thank Srđan Katušić for accepting me in his department, and always providing support if needed. A big thanks also goes to Gerald Stieglbauer, with whom it was always an enjoyment to work. He always showed understanding when I needed time to work on my PhD, and motivated me when I ran into problems. Through our discussions and talks I learned a lot from him and gained valuable experience. Finally, a big thanks goes to all the other colleagues with whom I have worked over the years, especially to my former room mate Dean Gržanić, who motivated me to do my best and finish my thesis as soon as possible.

Finally, I also want to thank all my friends who always supported me during these last several years. Domagoj, for being a great friend all these years, for all the entertaining and interesting talks we had, and for always helping me out however he could, especially with my thesis, but also with many other things as well. Lidija, for all the fun times and endless discussions which took my mind of the work I had to do, but also for always being there for me when I needed support and for always listening to me when I needed someone to talk to. Matea, for all the lunches on which we shared our frustrations about various things. Tomislav, for always helping out when needed. Katja, Dino, Ivan, Damir, Dario, and Andrija, for their continuous support and motivation.

Abstract

Scheduling is a decision-making process in which a certain set of activities or tasks needs to be allocated on one of the available scarce resources, over a given time period. The objective of the scheduling process is to create a schedule which optimises certain user defined criteria. Scheduling problems appear in many real world situations, such as in manufacturing processes, airports, and computer clusters. Unfortunately, most scheduling problem instances belong to the category of NP-hard problems. Therefore, various heuristic methods are most often used in order to obtain solutions for different scheduling problems. One of the most commonly used methods for solving scheduling problems are dispatching rules. Unlike many other methods which iteratively improve the quality of schedules, dispatching rules create the schedule incrementally by selecting which job should be scheduled on which machine at each decision moment. This makes dispatching rules especially useful for scheduling under dynamic conditions, since they can quickly adapt to the changing conditions of the system. However, designing good dispatching rules is a difficult and tedious task. For that reason, genetic programming is often used in order to automatically design new dispatching rules.

The main objective of this thesis is to improve the performance of dispatching rules which are generated by genetic programming. In the first part of the thesis multi-objective and many-objective optimisation methods were used in order to generate dispatching rules for optimising several objectives simultaneously. The obtained results demonstrate that the methods generated new dispatching rules which perform well for various scheduling objectives. In the second part of the thesis different ensemble learning methods were applied with genetic programming to generate ensembles of dispatching rules, which can achieve better results than by using only a single dispatching rule. The third part of the thesis proposes a procedure for selecting the dispatching rule which is best suited for a concrete problem instance. The aforementioned procedure achieves a better performance than if only a single dispatching rule would be used to solve all problem instances. The final part of the thesis analyses the adaptation of dispatching rules for static scheduling, by using several different methods. The tested methods provide different trade-offs between the quality of the results and execution times of the methods, with several methods outperforming results achieved by a genetic algorithm.

Keywords: genetic programming, scheduling problems, unrelated machines environment, multi-objective optimisation, ensemble learning, dynamic scheduling conditions, static scheduling conditions, machine learning

Prošireni sažetak

Automatizirano oblikovanje pravila raspoređivanja u okolini nesrodnih strojeva

Uvod

Raspoređivanje se može definirati kao proces u kojem je kroz određeni vremenski period potrebno rasporediti zadani skup aktivnosti, odnosno poslova, na jedan od dostupnih resursa, odnosno strojeva. Cilj procesa raspoređivanje jest izraditi raspored koji optimira određene korisnički definirane kriterije. Razlog zbog kojeg su problemi raspoređivanja iznimno detaljno proučavani proizlazi iz činjenice da se problemi raspoređivanja pojavljuju u mnogim situacijama iz stvarnog svijeta, kao primjerice raspoređivanje u proizvodnim i montažnim linijama, raspoređivanje aviona po pistama, raspoređivanje u raznim proizvodnim pogonima, ili raspoređivanju pacijenata za radioterapiju. Zbog tog razloga, razvojem novih i boljih metoda za rješavanje problema raspoređivanja bilo bi moguće primijeniti ih u mnogim scenarijima iz stvarnog svijeta, kako bi se primjerice poboljšala proizvodnja u proizvodnim procesima ili povećalo zadovoljstvo korisnika.

Nažalost, većina problema raspoređivanja spada u kategoriju NP-teških problema. Posljedica toga jest da ne postoje efikasni algoritmi koji bi mogli pronaći optimalno rješenja za dani kriterij optimizacije. Zbog tog razloga, problemi raspoređivanja se najčešće rješavaju korištenjem različitih heurističkih algoritama, koji se uglavnom dijele u dvije kategorije: unapređivačke i konstruktivne heuristike. Unapređivačke heuristike započinju s već postojećim rasporedom kojeg iterativno nastoje poboljšati. Razne metode poput genetskih algoritama ili optimizacije rojem čestica pripadaju toj kategoriji. Nažalost, navedene metode mogu se primijeniti samo u statičnim uvjetima raspoređivanja, u kojima su sve informacije o problemu raspoređivanja dostupne prije početka rada sustava, pa se onda posljedično i sam raspored može izraditi prije početka rada sustava. Takav način rada ograničava raspon problema za koje se unapređivačke heuristike mogu primijeniti, zbog toga što se najčešće ne mogu primijeniti u dinamičnim uvjetima, u kojima postoji potreba za konstantnom adaptacijom promjenama koje se kontinuirano događaju tijekom rada sustava.

Kako bi se izradili rasporedi za probleme u dinamičnim uvjetima, u kojima informacije o problemu postaju dostupne tijekom rada sustava, razvijen je veliki broj različitih konstruktivnih heuristika. U većini slučajeva te heuristike su definirane u obliku pravila raspoređivanja koja ne pretražuju čitav prostor rješenja, već inkrementalno konstruiraju raspored za neki problem. Kada izrađuju raspored, pravila raspoređivanja započinju s praznim rasporedom kojeg onda iterativno izgrađuju. Svaki put kada je potrebno rasporediti neki posao, pravilo raspoređivanja na temelju različitih parametara poslova i sustava (trajanje izvođenja posla, vrijeme dolaska u sustav, važnost posla) odabire koji će se posao u danom trenutku rasporediti na odabrani stroj. Pravila raspoređivanja mogu se podijeliti na dvije cjeline, shemu za izradu rasporeda i prioritetnu funkciju. Shema za izradu rasporeda definira kako će se čitav raspored izgraditi, primjerice u kojim trenucima će se poslovi raspoređivati, ili na koji stroj će se pojedini posao rasporediti. S druge strane prioritetna funkcija se koristi kako bi se mogao odabrati posao koji će se idući rasporediti. Pomoću prioritetne funkcije odrede se prioriteti svih dostupnih poslova na temelju njihovih karakteristika te shema za izradu rasporeda odabere najprikladniji posao s obzirom na vrijednosti njihovih prioriteta te ga rasporedi na neki od strojeva. Korištenjem pravila raspoređivanja moguće je iznimno brzo reagirati na različite promjene koje se mogu dogoditi u sustavu. Dodatno, trajanje izvođenja pravila raspoređivanja može se u većini slučajeva smatrati zanemarivom u usporedbi s trajanjem izvođenja unapređivačkih heuristika. Međutim, pravila raspoređivanja također imaju određene nedostatke. Jedan od najbitnijih nedostataka jest da je ručna izrada pravila raspoređivanja iznimno težak i dugotrajan proces koji se najčešće obavlja od strane eksperata za pojedinu domenu. Dakle, ako za dani problem ne postoji prikladno pravilo raspoređivanja, bit će ga potrebno prethodno izraditi, što s obzirom na veliki broj kriterija i različitih ograničenja nije nimalo jednostavno. Dodatan problem kod pravila raspoređivanja jest da ona najčešće ne mogu pronaći rješenja jednake kvalitete kao unapređivačke heuristike koje se mogu primijeniti samo za probleme u statičnim uvjetima. Konačno, kako postoji iznimno velik broj razvijenih pravila raspoređivanja, teško je unaprijed znati koje od tih pravila raspoređivanja će postići najbolje rezultate za pojedini problem raspoređivanja.

Da bi se izbjegla potreba za ručnom izradom pravila raspoređivanja, velik dio istraživanja usmjeren je na primjenu različitih metoda strojnog učenja i evolucijskog računarstva za automatsku izradu novih pravila raspoređivanja. Prilikom automatske izrade pravila raspoređivanja ne izrađuje se čitavo pravilo raspoređivanja, već se shema za izradu rasporeda najčešće definira ručno, dok se same prioritetne funkcije izrađuju korištenjem nekog algoritma strojnog učenja. Razlog tome je što se ista shema za izradu rasporeda može koristiti za rješavanje različitih problema raspoređivanja uz odabir prikladne prioritetne funkcije. Prioritetne funkcije mogu se izraditi korištenjem različitih postupaka, primjerice genetskog programiranja, umjetnih neuronskih mreža, stabala odluke i mnogih drugih. U dosadašnjem istraživanju najčešće se je ipak koristilo genetsko programiranje za izradu novih pravila raspoređivanja. Razloga tome je što su korištenjem ovog postupka postignuti najbolji rezultati za dani problem, ali i zbog toga što se pomoću genetskog programiranja mogu razviti pravila raspoređivanja koje je lakše interpretirati od pravila raspoređivanja izrađenih korištenjem drugih postupaka. Genetsko programiranje je postupak evolucijskog računarstva koji se može koristiti za rješavanje različitih optimizacijskih problema, i koji se je zbog svoje sposobnosti da razvije kompleksne funkcije i izraze pokazao veoma prikladnim za izradu novih pravila raspoređivanja. Primjenom genetskog programiranja moguće je jednostavno i efikasno izraditi nova pravila raspoređivanja za rješavanje različitih problema raspoređivanja. Pokazano je također da automatski izrađena pravila raspoređivanja veoma često postižu rezultate koji su bolji od rezultata dobivenih korištenjem ručno izrađenih pravila raspoređivanja. Dakle, korištenjem genetskog programiranja nije moguće samo automatski izraditi nova pravila raspoređivanja, već razviti i pravila raspoređivanja koja postižu bolje rezultate od već postojećih pravila raspoređivanja.

Iz navedenih razloga tijekom zadnjih dvadeset godina velik dio istraživanja bio je fokusiran upravo na automatsku izradu pravila raspoređivanja za rješavanje različitih problema raspoređivanja. Tako su u prethodnom istraživanju bili rješavani problemi raspoređivanja u kojima je bilo potrebno optimirati različite kriterije i pod različitim ograničenjima. Dodatno, bili su isprobani i uspoređeni različiti algoritmi za izradu novih pravila raspoređivanja, kako bi se odredilo koji od tih algoritama postižu najbolje rezultate za izradu novih pravila raspoređivanja. Osim toga, isprobane su i različiti postupci za poboljšanje efikasnosti razvijenih pravila raspoređivanja, ali i njihove interpretabilnosti kako bi bilo lakše odrediti na koji način pravila raspoređivanja donose svoje odluke i izrađuju raspored. No iako je napravljeno već mnogo istraživanja u području automatske izrade pravila raspoređivanja, postoji još mnogo otvorenih područja i problema koji u dosadašnjim istraživanjima nisu adekvatno pokrivena.

Motivacija

Glavna motivacija disertacije jest ostvariti nove znanstvene doprinose u područjima automatske izrade pravila raspoređivanja koja do sada u postojećoj literaturi nisu bila adekvatno istražena. Područja koja su bila u fokusu ove disertacije su: izrada pravila raspoređivanja za optimizaciju više kriterija istodobno, izrada skupova pravila raspoređivanja, definiranje procedure za odabir automatski izrađenih pravila raspoređivanja s obzirom na karakteristike instance problema, i izrada pravila raspoređivanje za optimizaciju problema raspoređivanja u statičnim uvjetima.

Problemi višekriterijske optimizacije su često proučavani u sklopu evolucijskog računarstva. Osim primjene postojećih algoritama na različite probleme, velik dio istraživanja usmjeren je i na razvoj novih algoritama za višekriterijski i mnogokriterijsku optimizaciju. Razlog zašto su višekriterijski i mnogokriterijski problemi često proučavani leži u činjenici da je u problemima iz stvarnog svijeta najčešće potrebno optimirati nekoliko kriterija istodobno. Upravo iz tog razloga postoji potreba za razvojem pravila raspoređivanja koje će moći izraditi rasporede kojima se optimira ne samo jedan, već nekoliko kriterija istodobno. Kako je već iznimno zahtjevno ručno izraditi pravila raspoređivanja koja optimiraju samo jedan kriterij, razviti pravila raspoređivanja koja optimiraju nekoliko kriterija istodobno je svakako još teži problem. Zbog toga je bitno istražiti mogu li se korištenjem različitih algoritama evolucijskog računarstva izraditi pravila raspoređivanja koja su prikladna za rješavanje višekriterijskih problema raspoređivanja. Iako je već određeni dio istraživanja bio usmjeren na rješavanje i analizu ovog problema, nažalost većina istraživanja bila je fokusirana na optimizaciju dva ili tri višekriterijska problema.

Na temelju isključivo tih rezultata teško je dati ocjenu o tome mogu li se izraditi prikladna pravila raspoređivanja za različite višekriterijske probleme koji se sastoje ne samo od različitog broja kriterija, već i od različitih kombinacija kriterija koji se međusobno zajedno optimiraju. Osim toga, rijetko su rađene usporedbe između automatski izrađenih pravila raspoređivanja i ručno izrađenih pravila raspoređivanja za optimizaciju više kriterija istodobno, kako bi se pokazalo mogu li automatski izrađena pravila raspoređivanja uistinu ostvariti bolje rezultate od ručno izrađenih pravila raspoređivanja. Upravo iz prethodno navedenih razloga, jedan od ciljeva ove disertacije jest pobliže proučiti problem izrade pravila raspoređivanja prikladnih za optimizaciju više kriterija istodobno.

Jedan od najvećih izazova u izradi pravila raspoređivanja jest poboljšanje njihovih performansi, jer kao što je prethodno pokazano u literaturi, pravila raspoređivanja najčešće ne postižu jednako dobre rezultate kao složenije unapređivačke heuristike. Osim toga, poboljšanjem performansi automatski izrađenih pravila raspoređivanja ona postaju prikladnija za primjenu u stvarnim sustavima za raspoređivanje. Iako su se performanse pravila raspoređivanja izrađenih genetskim programiranjem nastojale poboljšati korištenjem različitih prikaza rješenja ili postupaka lokalne pretrage pri njihovoj izradi, ipak postoji ograničenje do kojeg je moguće poboljšavati performanse jednog pravila raspoređivanja prije nego dođe do njegove prenaučenosti na skupu za učenje. Zbog toga javlja ideja da se umjesto korištenja samo jednog pravila raspoređivanja koristi skup pravila raspoređivanja kako bi se donijele odluke o tome koji će se posao idući rasporediti. Kako je skupno učenje već pokazalo iznimno dobre rezultate prilikom rješavanja različitih problema iz strojnog učenja (primjerice za poboljšanje performansi različitih klasifikacijskih postupaka), veoma je vjerojatno da se pomoću skupnog učenja mogu postići bolji rezultati kod problema raspoređivanja. U prethodnih nekoliko istraživanja već su demonstrirane prednosti korištenja skupova pravila raspoređivanja, jer su njima postignuti bolji rezultati nego korištenjem individualnih pravila raspoređivanja. No korištenje skupova pravila raspoređivanja radi postizanja boljih performansi je započelo tek nedavno, zbog čega još nije obavljeno mnogo istraživanja u ovom području. Tako su u dosadašnjem istraživanju primijenjene samo dvije metode za izradu skupova pravila raspoređivanja, koji su uglavnom bili primijenjeni za rješavanje problema raspoređivanja pod statičnim uvjetima. Zbog navedenih razloga jedan od ciljeva ove disertacije jest izraditi skupove pravila raspoređivanja koji se mogu koristiti za rješavanje problema raspoređivanja u dinamičnim uvjetima. Za izradu skupova pravila raspoređivanja isprobat će se nekoliko različitih postupaka, neki od kojih su izrađeni na temelju popularnih postupaka iz strojnog učenja.

Iako je korištenjem genetskog programiranja riješen problem oko potrebe za ručnom izradom pravila raspoređivanja, još uvijek postoje određeni problemi koji time nisu riješeni. Jedan od takvih problema, prouzročen činjenicom da postoji mnogo pravila raspoređivanja razvijenih za različite situacije, jest da unaprijed nije poznato koje pravilo raspoređivanja bi bilo najprikladnije za rješavanje koje instance problema. Čak iako se pravila raspoređivanja izrade na temelju problema s različitim karakteristikama, ne postoji garancija da će razvijeno pravilo raspoređivanja ostvariti dobre rezultate na svim vrstama problema nad kojima je bilo učeno. Štoviše, nemoguće je razviti pravilo raspoređivanja koje postiže dobre rezultate na svim mogućim instancama problema. Upravo zbog toga, čak iako se genetskim programiranjem razvije kvalitetno pravilo raspoređivanja, uvijek će biti moguće izraditi instancu problema nad kojom će dobiveno pravilo raspoređivanja ostvariti loše rezultate. Zato i kvalitetna pravila raspoređivanja mogu ostvariti loše rezultate ako su primijenjena za rješavanje neprikladnih instanci problema. Zbog navedenih razloga postoji potreba za definiranjem procedure koja će na temelju karakteristika instanci problema moći odabrati koje bi bilo prikladno pravilo raspoređivanja upravo za njeno rješavanje. Iako je na ovu temu je već obavljen veliki broj istraživanja, ono je isključivo bilo fokusirano na odabir jednog od nekoliko jednostavnih ručno izrađenih pravila raspoređivanja. Zbog toga je jedan od ciljeva ove disertacije definirati proceduru koja na temelju karakteristika instanci problema može odabrati najprikladnije automatski izrađeno pravilo raspoređivanja. U sklopu ove teme želi se isprobati može li se korištenjem ovakve procedure dodatno poboljšati efikasnost automatski izrađenih pravila raspoređivanja te kako će se procedura ponašati kada je potrebno napraviti odabir između većeg broja složenih pravila raspoređivanja koja već i sama po sebi mogu ostvariti dobre rezultate.

Većina istraživanja koje je bilo do sada obavljeno u sklopu automatizirane izrade pravila raspoređivanja je bilo fokusirano na rješavanje problema u statičnim uvjetima. Iako su u dijelu istraživanja bili korišteni problemi pod statičnim uvjetima, veoma malo istraživanja je bilo fokusirano upravo na izradu novih metoda koje bi unaprijedile kvalitetu pravila raspoređivanja u statičnim uvjetima. Naime, kako su pravila raspoređivanja uglavnom izrađena za rješavanje problema u dinamičnim uvjetima, ona postižu lošije rezultate nego drugi postupci koji su prilagođeni rješavanju problema raspoređivanja u statičnim uvjetima (kao primjerice genetski algoritmi ili slične unapređivačke heuristike). Razlog tome je što pravila raspoređivanja ne koriste statične informacije o problemu, upravo kako bi bila primjenjiva u dinamičnim okruženjima. Naravno, time imaju uži pogled na problem i postižu lošije rezultate nego postupci koji koriste takve informacije. No pravila raspoređivanja imaju određene prednosti nad unapređivačkim heuristikama. Prva prednost jest ta da mogu izraditi raspored u mnogo kraćem vremenu od unapređivačkih heuristika. Osim toga, pravila raspoređivanja izrađuju raspored inkrementalno, što znači da se dio rasporeda koji je već izrađen može izvoditi dok se ostatak rasporeda izgrađuje. Zbog navedenih razloga pravila raspoređivanja bi se mogla pokazati korisnima i za rješavanje problema raspoređivanja u statičnim uvjetima, ako je brzina izrade rasporeda također od velike važnosti. No kako bi se postigli što bolji rezultati korištenjem pravila raspoređivanja, potrebno ih je prilagoditi na način da prilikom odabira idućeg posla kojeg će rasporediti, osim informacija koje bi im bile dostupne u dinamičnim uvjetima, koriste i dodatne statične informacije o problemu. Kao što je ranije spomenuto, do sada je iznimno malo istraživanja bilo napravljeno kako bi se pravila raspoređivanja prilagodila za rješavanje problema u statičnim uvjetima, zbog čega jedan od ciljeva ove disertacije pobliže proučiti postupke kojima se pravila raspoređivanja mogu prilagoditi za rješavanje problema pod statičnim uvjetima.

Pregled disertacije

Disertacija je podijeljena na devet poglavlja, pri čemu prva četiri poglavlja daju uvod i motivaciju za automatsku izradu pravila raspoređivanja, dok iduća četiri poglavlja opisuju dobivene rezultate i ostvarene znanstvene doprinose. U posljednjem poglavlju dan je kratak zaključak o obavljenom istraživanju te smjernice za buduća istraživanja u automatskoj izradi pravila raspoređivanja.

Prvo poglavlje daje kratak uvod u disertaciju. U navedenom poglavlju ukratko su opisani problemi raspoređivanja te je izložena motivacija za proučavanje tih problema. Nadalje, u poglavlju je također izložena motivacija za automatsku izradu pravila raspoređivanja. Kroz poglavlje je istaknuto nekoliko otvorenih pitanja u tom području koja su proučavana u sklopu disertacije. U poglavlju je također dan i pregled izvornih znanstvenih doprinosa koji su ostvareni u sklopu disertacije. Konačno, poglavlje je zaključeno kratkim pregledom disertacije.

U drugom poglavlju dana je formalna definicija problema raspoređivanja. Osim što je opisana notica za definiranje problema raspoređivanja, istaknuti su različiti kriteriji koji se mogu optimirati, kao i ograničenja te uvjeti pod kojima se proces raspoređivanja može obavljati. Također, u ovom poglavlju opisani su različiti postupci optimizacije koji se mogu primijeniti za rješavanje problema raspoređivanja te su istaknute prednosti i nedostaci svakog od tih postupaka. U poglavlju se posebice stavlja naglasak na problem raspoređivanja u okruženju nesrodnih strojeva, koji je centralni problem koji se je rješavao u disertaciji. Za dani problem raspoređivanja nabrojana su i opisana ručno izrađena pravila raspoređivanja dostupna u literaturi, koja su se u daljnjim poglavljima koristila za dobivanje referentnih rezultata s kojima su se uspoređivali rezultati dobiveni automatski izrađenim pravilima raspoređivanja.

Treće poglavlje detaljno opisuje algoritam genetskog programiranja, koje se je u disertaciji koristilo kao primarna metoda za automatsku izradu novih pravila raspoređivanja. U ovom poglavlju opisani su svi bitni dijelovi genetskog programiranja, kao što su inicijalizacija, selekcija, prikaz rješenja, parametri algoritma te genetski operatori križanja i mutacije. Osim standardnog genetskog programiranja, u poglavlju su opisane i dvije varijante, naime *gene expression programming* i *dimensionally aware genetic programming*, koje su također bile upotrijebljene za automatsku izradu pravila raspoređivanja.

U četvrtom poglavlju opisan je način na koji se genetsko programiranje može koristiti za izradu novih pravila raspoređivanja. Na početku poglavlja izloženi su detalji o tome kako se genetsko programiranje može prilagoditi za izradu prioritetnih funkcija koje će se koristiti u sklopu novih pravila raspoređivanja. Nadalje, u poglavlju je dan iscrpni pregled postojeće literature koja je fokusirana upravo na automatsku izradu pravila raspoređivanja korištenjem genetskog programiranja i sličnih postupaka. Također, u poglavlju su navedeni i optimalni parametri genetskog programiranja koji su korišteni za izradu novih pravila raspoređivanja, kao i postupak izrade problema koji je korišten za izradu novih pravila raspoređivanja, ali i za njihovu evaluaciju tijekom čitave disertacije. Poglavlje završava s pregledom rezultata koji su dobiveni automatski razvijenim pravilima raspoređivanja. Dobiveni rezultati su uspoređeni s rezultatima dobivenim korištenjem dostupnih ručno izrađenih pravila raspoređivanja, kao i jednog genetskog algoritma koji je odabran kao predstavnik unapređivačkih heuristika. Prikazani rezultati koristili su se kao referentni rezultati s kojima su uspoređivani rezultati ostvareni svim metodama korištenim u disertaciji, kako bi se pokazalo mogu li se korištenjem odabranih i predloženih metoda poboljšati performanse automatski generiranih pravila raspoređivanja.

Peto poglavlje bavi se problemom izrade pravila raspoređivanja koja se mogu primijeniti za optimizaciju više kriterija istodobno. U poglavlju su prvo izloženi temelji višekriterijske optimizacije i način na koji se genetsko programiranje može prilagoditi za rješavanje ovog problema. Kako bi se isprobala prikladnost genetskog programiranja za izradu pravila raspoređivanja koja optimiraju viče kriterija istodobno, isprobana su četiri algoritma za višekriterijsku optimizaciju. Sve četiri algoritma su primijenjena za rješavanje 14 različitih višekriterijskih problema gdje se je broj kriterija kretao između tri i devet. Za većinu testiranih višekriterijskih problema algoritmi su izradili pravila raspoređivanja koja su postigla bolje rezultate od dostupnih ručno izrađenih pravila raspoređivanja. Iako su najbolji rezultati postignuti prilikom optimizacije problema koji su se sastojali od šest ili manje kriterija, razvijena su kvalitetna pravila raspoređivanja i za probleme s većim brojem istodobno optimiranih kriterija. Jedan od problema koji je uočen prilikom optimizacije višekriterijskih problema raspoređivanja jest taj da su algoritmi pokazali iznimno veliku osjetljivost s obzirom na to koji se kriteriji međusobno zajedno optimiraju. Naime, ako se optimiraju konfliktni kriteriji, performanse algoritama su često bile lošije nego u slučajevima kada se optimiraju kriteriji koji nisu izrazito međusobno konfliktni. Osim toga, pomoću dobivenih rezultate bilo je moguće odrediti koreliranost različitih kriterija, što se može pokazati korisnim u budućim istraživanjima prilikom izrade novih višekriterijskih problema koji će se optimirati. Na temelju rezultata ostvarenih u ovom poglavlju pokazano je kako se pomoću genetskog programiranja mogu razviti pravila raspoređivanja koja su prikladna za rješavanje višekriterijskih problema raspoređivanja.

Šesto poglavlje fokusirano je na izradu skupova pravila raspoređivanja, kako bi se poboljšale performanse razvijenih individualnih pravila raspoređivanja. U poglavlju je prvo opisano pet metoda koje će se koristiti za izradu skupova pravila raspoređivanja. Od tih pet metoda, tri metode su preuzete iz literature (*BoostGP*, *BagGP* i *kooperativna koevolucija*), dok su dvije metode predložene u disertaciji (*Simple Ensemble Combination* (SEC) i *Ensemble subset search*

(ESS)). Osim navedenih metoda za izradu skupova pravila raspoređivanja, korištene su i dvije metode pomoću kojih se donosi odluka na temelju skupa pravila raspoređivanja: *sum* i *vote*. Kako je SEC metoda predložena u disertaciji, prvi dio poglavlja bio je fokusiran na testiranje te metode uz različite kombinacije parametara kako bi se ispitale njene performanse. Nakon što su određeni optimalni parametri za SEC metodu, ona i preostale četiri metode korištene su za izradu skupova pravila raspoređivanja, pri čemu su metode primijenjene za optimiranje četiri kriterija. Za sva četiri testirana kriterija korištenjem skupova pravila raspoređivanja postignuti su bolji rezultati nego korištenjem individualnih ručno ili automatski izrađenih pravila raspoređivanja. Najbolji rezultati za sva četiri kriterija postignuti su od strane skupova koji su bili generirani korištenjem metoda predloženih u disertaciji. Kroz detaljniju analizu generiranih skupova pravila raspoređivanja uočeno je kako skupovi pravila raspoređivanja za većinu instanci problema postižu jednako dobre ili čak i bolje rezultate od najboljeg individualnog pravila raspoređivanja, što zapravo dovodi do toga da onda i na ukupnom skupu problema skupovi pravila raspoređivanja postižu bolje rezultate od individualnih pravila raspoređivanja. Osim toga, najbolji rezultati su najčešće postignuti od strane skupova koji se sastoje od manjeg broja pravila raspoređivanja. Na temelju svih ostvarenih rezultata u sklopu ovog poglavlja može se zaključiti kako skupovi pravila raspoređivanja predstavljaju efikasan mehanizam za poboljšanje performansi pravila raspoređivanja.

U sedmom poglavlju analizirano je kako se različiti postupci strojnog učenja mogu primijeniti za definiranje procedure koja bi na temelju karakteristika pravila raspoređivanja mogla odabrati prikladno pravilo raspoređivanja za rješavanje konkretne instance problema. Na početku poglavlja dan je pregled literature koja se je bavila odabirom prikladnog ručno izrađenog pravila raspoređivanja. Nakon toga, detaljno je opisana predložena procedura za odabir automatski izrađenih pravila raspoređivanja, koja je primijenjena u statičnim i dinamičnim scenarijima za odabir prikladnog pravila raspoređivanja. U statičnim scenarijima radi se s pretpostavkom da su određene karakteristike instance problema dostupne prije početka rada sustava (iako se raspoređivanje obavlja u dinamičnim uvjetima) i da je korištenjem tih karakteristika moguće odrediti koje pravilo raspoređivanja bi bilo prikladno za njeno rješavanje. S druge strane, u dinamičnim uvjetima radi se s pretpostavkom da nikakve karakteristike o instance problema nisu dostupne niti poznate prije početka rada sustava, pa ih je posljedično potrebno aproksimirati tijekom rada sustava i tek onda donijeti odluku o tome koje bi pravilo raspoređivanja bilo najprikladnije. U nastavku poglavlja opisan je način provođenja eksperimenata, kao i način izrade instanci problema, pri čemu će kod nekih problema karakteristike biti konstante tijekom čitavog rada sustava, dok će se kod nekih karakteristike mijenjati tijekom rada sustava. Predložena procedura primijenjena je nad nekoliko različitih problema te su rezultati uspoređeni s jednim ručno odabranim pravilom raspoređivanja. Dobiveni rezultati pokazuju kako predložena procedura može postići bolje rezultate od ručno odabranog pravila raspoređivanja, zbog toga što uspijeva odrediti prikladna pravila raspoređivanja za pojedine trenutke. No ozbiljni nedostatak predložene procedure jest činjenica da je potrebno odrediti optimalne vrijednosti za veliki broj parametara, jer u suprotnom procedura postiže iznimno loše rezultate. Kroz ostvarene rezultate pokazano je kako predložena procedura može ostvariti bolje rezultate odabirom prikladnih pravila raspoređivanja za trenutnu situaciju, nego kad bi se koristilo samo jedno ručno odabrano pravilo raspoređivanja za rješavanje svih instanci.

Osmo poglavlje bavi se problemom prilagodbe automatski generiranih pravila raspoređivanja za probleme raspoređivanja u statičnim okruženjima. U tu svrhu bit će iskorištene četiri metode koje su opisane u početnom dijelu poglavlja: statični terminalni čvorovi, *look-ahead* metoda, iterativna pravila raspoređivanja i *rollout* algoritam. U nastavku poglavlja prvo se analizira kvaliteta rješenja dobivenih svakim od ovih postupaka i njihovim međusobnim kombinacijama. Osim ispitivanja same kvalitete izrađenih rasporeda, mjereno je i vrijeme potrebno za izradu rasporeda te su svi postupci ocijenjeni i s obzirom na taj kriterij. Također je napravljena i analiza ponašanja pojedinih postupaka prilikom rješavanja jedne instance problema, kako bi se odredile prednosti i nedostaci svakog od postupka. Dobiveni rezultati pokazuju da je look-ahead metoda najprikladnija ako su kvaliteta rasporeda i vrijeme potrebno za njegovu izradu jednako bitni. Naime, pri usporedbi s automatski izrađenim pravilima raspoređivanja za dinamične uvjete vidljivo je da pravila raspoređivanja s look-ahead metodom postižu značajno bolje rezultate, uz samo dvostruko dulje trajanje izvođenja. Također, u usporedbi s referentnim genetskim algoritmom navedena metoda pokazala je da može postići relativno dobre rezultate no u mnogo kraćem vremenu. S druge strane, rollout algoritam ostvaruje najbolje rezultate od svih testiranih postupaka. Naravno, posljedica toga je da je tom algoritmu potrebno i značajno više vremena da izgradi raspored nego primjerice automatski izrađenim pravilima raspoređivanja za dinamična okruženja. No u usporedbi s referentnim genetskim algoritmom pokazano je kako rollout algoritam postiže bolje rezultate i to najčešće u kraćem vremenu. Zbog toga se rollout algoritam nameće kao najprikladniji postupak ako je isključivo kvaliteta rasporeda bitna. Također je pokazano kako se korištenjem različitih kombinacija postupaka mogu postići još bolji rezultati, no uz dulje trajanje izrade rasporeda. Bez obzira na to, podešavanjem parametara testiranih postupaka moguće je napraviti kompromis između kvalitete izrađenog rasporeda i vremena potrebnog za njegovu izradu. S obzirom na rezultate ostvarene u ovom poglavlju može se zaključiti kako je korištenjem isprobanih postupaka za prilagodbu pravila raspoređivanja za probleme u statičnim uvjetima moguće postići iznimno dobre rezultate, koji nerijetko nadmašuju i rezultate dobivene korištenjem referentnog genetskog algoritma.

U devetom poglavlju dan je kratak zaključak disertacije te su istaknuti izvorni znanstveni doprinosi koji su ostvareni kroz disertaciju. Nadalje, dan je pregled tema za buduće istraživanje u automatskoj izradi pravila raspoređivanja, kao što su unapređenje interpretabilnosti pravila raspoređivanja, definiranje novih shema za izradu rasporeda, kombiniranje različitih postupaka

isprobanih u sklopu disertacije te mnoge druge.

Zaključak

Glavni cilj disertacije je poboljšati kvalitetu pravila raspoređivanja generiranih korištenjem genetskog programiranja te omogućiti izradu pravila raspoređivanja koja su prikladna za primjenu u različitim uvjetima i situacijama. U sklopu disertacije ostvarena su sljedeća četiri izvorna znanstvena doprinosa:

- Postupak oblikovanja prioritetnih funkcija za višekriterijske i mnogokriterijske probleme raspoređivanja temeljen na evolucijskim algoritmima
- Metoda skupnog učenja s ciljem poboljšanja kvalitete pravila raspoređivanja
- Postupak odabira pravila raspoređivanja prilagođenih svojstvima instance problema
- Postupak oblikovanja pravila raspoređivanja primjenjivih u statičnoj okolini raspoređivanja

Kroz postignute rezultate može se zaključiti kako je za sve testirane probleme, kriterije i uvjete raspoređivanja genetsko programiranje pokazalo veliku uspješnost u izradi novih pravila raspoređivanja, koja nerijetko postižu bolje rezultate od postojećih ručno izrađenih pravila raspoređivanja. Osim toga, korištenjem postupaka koji su bili testirani i predloženi u sklopu disertacije moguće je dodatno poboljšati kvalitetu automatski izrađenih pravila raspoređivanja u usporedbi s pravilima raspoređivanja koja su izrađena korištenjem standardnog genetskog programiranje, i time ostvariti još bolje rezultate u usporedbi s ručno izrađenim pravilima raspoređivanja. Upravo zbog svih izloženih zaključaka i ostvarenih rezultata moguće je zaključiti kako je genetsko programiranje pokazalo iznimno veliki potencijal za primjenu u automatskoj izradi pravila raspoređivanja u okruženju nesrodnih strojeva. Osim toga, ostvareni rezultati također pokazuju i veliki potencijal za provođenje daljnjeg istraživanja, kao i za daljnje poboljšanje postojećih postupaka u području automatske izrade pravila raspoređivanja.

Ključne riječi: genetsko programiranje, okolina nesrodnih strojeva, višekriterijska optimizacija, skupno učenje, dinamični uvjeti raspoređivanja, statični uvjeti raspoređivanja, strojno učenje

Contents

Chapter 1

Introduction

Scheduling is a decision-making process in which a certain set of activities or tasks needs to be allocated on one of the available scarce resources, over a given time period [1]. The objective of the scheduling process is to create a schedule which optimises certain user defined criteria. The reason why scheduling problems are widely researched originates from the fact that these problems appear in many real world situations, like scheduling planes on runways [2, 3], scheduling in manufacturing and assembly lines [4, 5], scheduling in wafer fabrication [6] and production plants [7], or scheduling for radiotherapy pre-treatment [8]. Therefore, by obtaining better methods for solving scheduling problems it would also be possible to apply them for many real world scenarios in order to, for example, increase user satisfaction or production in manufacturing environments.

Unfortunately, most scheduling problems belong to the category of NP-hard problems [1]. As a consequence, no efficient algorithms which could obtain the optimal solution for a given objective are available. Therefore, solutions are most often obtained by using various heuristic algorithms, which are usually divided into two groups: improvement heuristics and constructive heuristics. Improvement heuristics start with an initial schedule, which they try to iteratively improve. Various metaheuristic methods, like genetic algorithms or particle swarm optimisation, belong to this category [9]. However, these methods can only be applied for static scheduling environments, in which all the information about the scheduling problem is present before the system starts with its execution. This heavily limits the range of problems on which improvement heuristics can be applied, since they usually can not be used in dynamic scheduling environments, in which a constant adaptation to the changing conditions is needed.

In order to create schedules in dynamic scheduling environments, where the information about the system becomes available during its execution, a vast number of constructive heuristics have been developed. In most cases these heuristics are defined in the form of dispatching rules, which do not search the entire space of solutions, but rather build up the schedule incrementally [10, 11, 12, 13]. When creating schedules, the dispatching rules start with an empty schedule, and each time a scheduling decision needs to be performed they use the currently available information from the system to determine which job should be scheduled on which machine at the current moment in time. This allows dispatching rules to quickly react to the changing conditions in the scheduling environment. In addition, their time complexity is in most cases almost negligible when compared to the time complexity of improvement heuristics. However, dispatching rules also cope with a certain number of problems. The most serious problem is that good dispatching rules are quite hard to design, which means that in each case an adequate dispatching rule is not available, a new rule would need to be designed.

To deal with the problem of having to design new dispatching rules manually, a large number of research was conducted, in which various machine learning and evolutionary computation methods were used for automatic design of new dispatching rules [14, 15]. In most cases genetic programming [16] was used to automatically design new dispatching rules. Genetic programming is a metaheuristic optimisation procedure which is capable of evolving complex functions and expressions, and therefore it is more than suitable for creating new dispatching rules. With the use of genetic programming it is possible to automatically design new dispatching rules for various scheduling criteria and conditions. In addition to this, in most cases automatically designed dispatching rules achieve a significantly better performance than any of the manually designed dispatching rules from the literature. Therefore, by using genetic programming it is not only possible to automatically design new dispatching rules, but also to acquire rules which provide superior performance when compared to already existing dispatching rules.

Through the years a large amount of research was performed in the field of automatically designing new dispatching rules. This can best be seen from the two recent surveys which give an overview of the research which was performed in this area [14, 15]. In addition, three PhD theses by Jakobović [17], Nguyen [18], and Hunt [19] also demonstrate that the field of automatically designing new dispatching is actively researched. Although a great deal of research has already been performed in this area, there are still many open issues which are not yet adequately covered in the literature.

1.1 Research motivations

This section will give an overview of several currently open issues in the field of automatic design of dispatching rules. Dealing with these issues will be the main objective of this thesis.

Multi-objective and many-objective optimisation are currently one of the most investigated areas in evolutionary computation, with many new methods being constantly proposed [20]. Various scheduling problems, in which several criteria were optimised simultaneously, were examined by numerous researchers [21, 22, 23, 24, 25]. Therefore, designing dispatching rules for simultaneous optimisation of several scheduling criteria is also an interesting field of research which was investigated by several researchers [26, 27, 28, 29, 30, 31, 32]. The reason for this area being of interest is because in real world scenarios it is rarely the case that only a single objective needs to be optimised. Therefore it is very important to analyse the possibility of automatically designing dispatching rules suited for the simultaneous optimisation of several criteria, and to propose methods which can be used to automatically generate them. Some research has already been performed in this field, and the obtained results demonstrate that there is a high potential of automatically developing dispatching rules for optimising several criteria simultaneously. Unfortunately, most of the research which was performed on this topic focused on optimising only one or two combinations of criteria. Therefore, no study provides a notion of how the performance of methods for designing dispatching rules, which optimise multiple objectives simultaneously, depends on the number or composition of the criteria which make up the multi-objective problem. In addition, in only one study did the authors provide a comparison of dispatching rules generated for optimising multiple objectives with manually developed dispatching rules, in order to give an impression of the quality of automatically developed dispatching rules. The other studies provided only the values of the multi-objective metrics which, although are informative when comparing different multi-objective algorithms, do not provide any information about the quality of the obtained dispatching rules.

Since dispatching rules are unable to perform equally well as the more complex improvement heuristics, it is important to improve their performance as much as possible. For that purpose many methods for obtaining better dispatching rules were developed, including the introduction of new terminal nodes [33], using different representations [34], and using local search operators [29]. However, another possibility of improving the performance of dispatching rules would be to create ensembles of dispatching rules. This would mean that instead of using only one DR to perform the scheduling decision, a group of DRs which form the ensemble would jointly perform the scheduling decision. Since ensemble learning methods have shown to achieve good performance on certain machine learning problems, it is also highly probable that by applying them for automatic generation of dispatching rules could also lead to improved results. This was already demonstrated in several occasions in which ensembles of dispatching rules were constructed [35, 36, 37, 38]. However, in most occasions the ensemble learning methods were applied only in the static scheduling environment. In addition, ensemble learning approaches which are based on some popular methods from machine learning, like bagging and boosting, were also not applied yet. There is also a lot of possibility to define new ensemble learning methods, or procedures which could be used in order to improve the performance of already existing methods.

Although generating dispatching rules by genetic programming does solve some important problems associated with dispatching rules, there is still one serious problem remaining. The problem is that in dynamic environments it is not known which of the available dispatching rules is most suited to be used for the given problem instance. Although dispatching rules can be generated by using a training set consisting of problems with different characteristics, this still does not guarantee that the DRs generated by genetic programming will perform well on all problem instances on which they were trained. On the contrary, it is rather impossible to create a dispatching rule which will perform well on all possible problem instances. Therefore, even if a good dispatching rule is obtained by genetic programming, it will always be possible to define a problem for which that rule will perform poorly. Even if many good rules for different types of problems are obtained by genetic programming, they will still not be able to perform well if selected for inappropriate problem instances. For that reason a procedure for determining which dispatching rule should be applied at which moment would be required. A great deal of research was already conducted on this topic, however, the entire research focused solely on using simple manually defined dispatching rules [39, 40, 41]. Furthermore, most of the research focuses on using only a few dispatching rules out of which the appropriate one needs to be selected. Therefore, it is not known how these procedures would perform when used with a large number of complex automatically generated dispatching rules. However, if combining such a procedure with automatically generated dispatching rules would lead to improved results, this would resolve another important issue still associated with automatically generated dispatching rules, and certainly lead to an even greater automation of the entire process.

Even though much of the research on automatic design of dispatching rules was performed in the static scheduling environment, only a few studies focused on methods of adapting dispatching rules for the static scheduling environment [42, 43]. Although it might seem unnecessary to perform such research, since improvement heuristics will generally achieve much better results for static scheduling environments, dispatching rules still have several advantages over improvement heuristics. One of the most important advantages is their superior execution time. The second advantage is that dispatching rules create the schedule incrementally, which means that the part of the schedule which is already created by the rule can be executed, while the the rest of the schedule is being constructed incrementally by the dispatching rule. Because of these reasons, dispatching rules can also prove to be useful for scheduling in static environments if it is necessary to build the schedule and start executing the system as soon as possible. Unfortunately, dispatching rules are mostly suited for dynamic environments, and therefore they do not use static information about the problem. Thus it is important to adapt those dispatching rules to make them more suitable for scheduling in static environments, and to allow them to achieve a better performance. However, as previously outlined, very little research has been done in this area, meaning that there is still a lot of possibility for improving the results of automatically generated dispatching rules for the static scheduling environment.

1.2 Major contributions of the thesis

The overall objective of this thesis is to apply genetic programming to automatically design new dispatching rules for the unrelated machines environment. The main focus of this research is to improve the performance of dispatching rules generated by genetic programming, and make them more robust for solving different scheduling problem instances. In order to achieve this objective, the standard genetic programming algorithm and the schedule generation scheme will be extended with various methods that lead to better performance of the automatically designed dispatching rules. By using the methods proposed in this thesis, genetic programming should be able to automatically design dispatching rules which can perform well on a wide range of different problem instances, for both the static and dynamic scheduling environment, and for optimising one or multiple criteria.

The first objective of this thesis is the development of dispatching rules for simultaneous optimisation of multiple objectives. The goal here is to couple genetic programming with multi-objective and many-objective algorithms to design dispatching rules for simultaneous optimisation of several criteria. For that purpose, four prominent multi-objective and many-objective genetic algorithms will be selected and combined with genetic programming. Such a procedure will then be used to develop dispatching rules for optimising multiple scheduling criteria simultaneously. The benefit of such an approach is that it will not only provide a single dispatching rule as a result, but rather an array of dispatching rules where each of the obtained rules provides a different trade off between the objectives for which it was optimised. The main intention in this thesis is to test whether such a procedure can be used to generate new dispatching rules for multi-objective and many-objective problems of various sizes and combinations of different scheduling criteria. Therefore the automatically generated dispatching rules will be compared to several manually designed dispatching rules, to test whether they can outperform them when multiple criteria are optimised. In addition, the correlation between the different scheduling criteria will also be analysed to obtain the knowledge about which criteria are most appropriate for simultaneous optimisation.

The second objective of the thesis is to increase the performance of the automatically generated dispatching rules. This is an important goal, since improving the performance of the approach makes it more viable for application in real environments. To achieve this goal, several ensemble learning approaches are selected and combined with genetic programming to generate ensembles of dispatching rules. The benefit of ensembles is that they allow for several dispatching rules to have an influence on each scheduling decision. This reduces the probability of performing bad decisions, since it will rarely happen that all the rules in the ensemble will perform a bad scheduling decision. In order to create ensembles of dispatching rules, two ensemble learning methods will be proposed in this thesis, while three prominent ensemble learning methods will be selected from the literature. The thesis will investigate the performance of all the selected ensemble learning methods on several scheduling criteria. In addition, several ensembles will be analysed in more detail to extract certain knowledge from their structure, which could possibly provide useful in the process of creating new ensembles.

The third objective of this thesis is to deal with the problem of selecting the appropriate dispatching rule for solving the current problem instance. This issue is quite important since applying an inappropriate dispatching rule for a scheduling problem can lead to the creation of a schedule which performs poorly for the optimised objective. Therefore, the thesis proposes a procedure which can, based on certain features of the current scheduling problem instance, determine the dispatching rule which should be most appropriate for solving the current problem instance. The procedure is based on using a machine learning method that can learn the association between problem instances of different characteristics and dispatching rules. This knowledge can later be used for determining which dispatching rule should be executed for the current problem instance. The procedure will be tested on two scenarios, one in which the needed problem instance characteristics are available up front, and the other where the procedure needs to approximate them during the execution of the system. In addition, to gain insights about the procedure, the thesis also provides a detailed analysis of its behaviour on several different problem types.

The final objective with which the thesis deals is the adaptation of automatically designed dispatching rules for solving problems in static scheduling conditions. Four scheduling methods will be used to adapt dispatching rules for static scheduling conditions. In this thesis, a number of terminal nodes which provide static information to dispatching rules will be proposed. In addition, a method of combining the rollout algorithm with automatically generated dispatching rules is also proposed. The selected methods are tested on several problem instances in order to validate their performance. In addition, combinations of different methods are also tested to further improve the performance of individual methods. All tested methods are compared to the results achieved by dispatching rules generated for dynamic scheduling conditions, but also with a genetic algorithm, to analyse how the performance of dispatching rules compares to those of improvement heuristics. The goal of the thesis is also to provide an analysis of the execution time of the applied methods, to illustrate how the different methods balance between execution time and the quality of the obtained results.

Based on the previous descriptions, the major contributions of this thesis can be summarised through the following four points:

1. Priority function design procedure for multi-objective and many-objective scheduling problems based on evolutionary algorithms.
2. Ensemble learning method with the objective of improving the quality of dispatching rules.

3. Selection procedure of dispatching rules adapted to the features of the problem instance.
4. Design procedure of dispatching rules applicable in the static scheduling environment.

1.3 Outline of the thesis

The thesis is divided into to nine chapters. While Chapters 2, 3, and 4 give a detailed introduction of the problem and the methods which are applied for solving it, Chapters 5, 6, 7, and 8 present the contributions of this thesis.

Chapter 1 represents the introductory chapter of the thesis. This chapter first gives a short introduction of the problem and outlines the motivations for researching it. In addition, the chapter also gives a summary of the major contributions achieved in this thesis. The chapter concludes with an overview of the entire thesis.

Chapter 2 gives a formal description of scheduling problems. In this chapter, the notation of scheduling problems, and the different conditions under which scheduling can be performed are described. The various methods which are most often used for solving different kinds of scheduling problems are also outlined with their strengths and weaknesses. The chapter also provides a closer overview of the unrelated machines scheduling environment. This part includes a detailed overview of the most prominent dispatching rules collected from the literature, which are used for solving scheduling problems in the unrelated machines environment.

Chapter 3 gives a detailed description of the genetic programming algorithm. The chapter describes all the main parts of the algorithm, like solution representation, genetic operators and selection. Aside from the standard genetic programming algorithm, two of its variants, dimensionally aware genetic programming and gene expression programming, are also described since these algorithms are used in some parts of the thesis.

Chapter 4 describes how genetic programming can be used for automatic design of new dispatching rules for the unrelated machines environment. The chapter first describes the details of adapting genetic programming in order to create new priority functions. An extensive survey of the literature dealing with automatic design of dispatching rules is also presented in the chapter. Details about the parameters of genetic programming and the experimental design, which are used throughout the thesis, are described. The section concludes with an overview of the current state of the art results of automatically generated dispatching rules for the unrelated machines environment, as well as an overview of the performance of the selected manually designed dispatching rules. These results will be used in subsequent chapters of the thesis in order to demonstrate how the achieved results compare to those obtained by methods already proposed in the literature.

Chapter 5 deals with the problem of automatic design of dispatching rules for optimising several scheduling criteria simultaneously. The chapter first gives a short introduction of multi-objective optimisation, and describes the details of how genetic programming can be adapted to cope with this problem. The experimental design for multi-objective optimisation is also described, since there are small differences in evaluating the performance of solutions when compared to the case of optimising only a single objective. Four prominent multi-objective and many-objective genetic algorithms are combined with genetic programming for solving the problem, and their parameters are fine tuned to obtain the best possible results. The four selected methods are applied on several multi-objective and many-objective scheduling problems, and their performance is compared to that of dispatching rules generated for optimising a single objective. Several performance metrics are used to compare the results achieved by the individual multi-objective and many-objective algorithms. To gain further insights on how the generated dispatching rules perform, several rules which optimise various criteria combinations are selected and compared to five manually designed dispatching rules. Finally, the chapter also provides a correlation analysis of various scheduling criteria.

Chapter 6 proposes the application of different ensemble learning methods for the construction of ensembles of dispatching rules, which should lead to an improved performance over individual dispatching rules. The chapter first describes five ensemble learning methods which are used for designing ensembles of dispatching rules, two of which are proposed in this thesis, while the other three are selected from the literature. Furthermore, the adjusted experimental design for the ensemble learning methods is also described. Since the simple ensemble combination method is proposed in this thesis, its performance is analysed in more detail to determine how the different parameters influence the quality of the generated ensembles. After determining the optimal parameter values for the simple ensemble combination approach, all approaches are applied for solving different scheduling problem instances, in order to compare their performance. Based on the observed results, a discussion about the strengths and weaknesses of the applied approaches is presented. In the end of the chapter, several ensembles constructed by different approaches are collected and analysed in more detail, to gain a deeper insight on how the different approaches construct the ensembles.

Chapter 7 describes how dispatching rules generated by genetic programming can be combined with different machine learning methods to provide a procedure which can, based on the current characteristics of the problem instance, determine the appropriate dispatching rule that should be applied. First, a survey of the literature dealing with the automatic selection of dispatching rules, based on characteristics of problem instances, is given. After that, a procedure for selecting automatically generated dispatching rules is proposed. The proposed procedure is applied in a static and dynamic scenario. In the static scenario the characteristics of the problems are known in advance and can be used by the procedure, while in the dynamic scenario the characteristics are not known in advance and need to be approximated during the execution of the system. The results achieved by the proposed procedure are compared to the results which would be achieved if only a single manually selected dispatching rule would be used for all problem instances. In addition, the rule selection procedure is analysed in more detail on a few selected problem instances to gain a deeper insight into the details of the procedure. The chapter is concluded with a short discussion about the proposed procedure.

Chapter 8 deals with the adaptation of dispatching rules to make them more appropriate for solving scheduling problems in static conditions. Four methods for adapting dispatching rules to static scheduling conditions are described. An extensive set of terminal nodes, which provide information about the static characteristics of the problem to dispatching rules, are proposed. The application of the rollout algorithm with automatically designed dispatching rules is also proposed. All four methods are applied on several scheduling problem instances in order to measure their performance. In addition, combinations of different methods are also tested to analyse whether it is possible to obtain improved results by combining different methods for adapting dispatching rules for static conditions. An analysis of the execution times of the applied methods is also performed to obtain a notion of how the time needed to create the solution depends on the selected methods. Finally, the chapter concludes with an additional analysis of the different methods, which provides further insights on how the different methods construct the schedule, and outlines the reasons because of which some methods perform better.

Chapter 9 gives the conclusion of the thesis, and provides an overview of the achieved scientific contributions. Furthermore, the chapter also provides an overview of possible future research directions in the field of automatic design of dispatching rules.

Chapter 2

Scheduling problems

Scheduling is a decision-making process in which a set of jobs needs to be scheduled on a finite set of machines, to optimise one or more user defined criteria [1]. Scheduling problems appear in many real world situations, like scheduling planes on runways [2, 3], scheduling in manufacturing and assembly lines [4], scheduling in wafer fabrication [6, 44, 45] and production plants [7], scheduling resources in clouds [46, 47], staff scheduling [48], multiprocessor scheduling [49], or scheduling for radiotherapy pre-treatment [8]. However, most scheduling problem instances belong to the category of NP-hard problems, meaning that heuristic algorithms are most often used to solve scheduling problems.

The rest of this chapter will give a short overview of various scheduling problems, environments, conditions, and also methods for solving scheduling problems. The unrelated machines environment will be described in more detail, especially with regards to metaheuristic methods and dispatching rules used for solving scheduling problems associated with the aforementioned environment.

2.1 Notation of scheduling problems

The number of jobs in a scheduling problem is usually denoted with n, while on the other hand the number of machines is denoted with m. The index j is usually used to denote a concrete job, while the index i is used in order to denote a machine. In most theoretic scheduling problems it is assumed that the number of jobs and machines are finite. Although there are many different scheduling problems in existence, in most of them the following job characteristics are used [1, 50]:

- **Processing time** p_{ij} - defines the time needed for job j to be executed on machine i.
- **Release time** r_j - defines the point in time at which job j becomes available and is released into the system. Before its release time a job can not start with its execution.
- **Due date** d_j - defines the point in time until which job j should finish with its execution,

otherwise a certain penalty will be incurred.

- **Deadline** $\bar{d}_j$ - defines the point in time until which job j must finish with its execution.
- **Weight** w_j - defines the weight (importance) of the job with index j. A job is considered to be more important if it has a larger weight value. The weight can have a different value for each of the scheduling criteria. In this thesis different weights will be used for tardiness (w_{Tj}), earliness (w_{Ej}) and the completion time (w_{Cj}) criteria.

After the schedule is constructed, for each job it is possible to calculate several metrics which will in turn be used to determine the values of the scheduling criteria. The following metrics are most commonly used [1, 50]:

- **Completion time** (C_j) - the moment in time at which job j finishes with its execution and exists the system.
- **Flowtime** (F_j) - the amount of time that job j spent in the system:

$$F_j = C_j - r_j. \tag{2.1}$$

- **Tardiness of a job** (T_j) - the amount of time that job j spent executing after its due date:

$$T_j = \max(C_j - d_j, 0). \tag{2.2}$$

- **Earliness** (E_j) - the amount of time that job j finished prior to its due date:

$$E_j = \max\left(-(C_j - d_j), 0\right). \tag{2.3}$$

- **Unit penalty** (U_j) - a flag denoting whether a job is tardy or not:

$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}. \tag{2.4}$$

Since there are many different scheduling problems, they are most commonly described by using a triplet $\alpha|\beta|\gamma$. The first field of this triplet denotes the machine environment of the scheduling problem and always contains just one entry. The second field contains details about the different characteristics and constraints present in the scheduling environment. This field can contain zero, one or several entries. The final field denotes the scheduling objectives which are minimised and contains one or more entries.

The α field can represent one of the following machine environments [1, 50]:

- **Single machine** (1) - consists of one machine on which all jobs are executed.
- **Identical machines in parallel** (Pm) - consists of m machines in parallel which have the same execution speed.

- **Uniform machines in parallel** (Qm) - consists of m machines in parallel which have different execution speeds v_i. If the duration of job j is p_j, then the processing time of the job can be calculated as $p_{ij} = p_j / v_i$.
- **Unrelated machines in parallel** (Rm) - consists of m machines in parallel. In this environment the execution speed of a machine does not only depend on the machine itself, but also on the job it is executing. Thus, the processing time can be calculated as $p_{ij} = p_j / v_{ij}$, where v_{ij} is the execution speed of machine i for job j.
- **Flow shop** (Fm) - consists of m machines in series and each job needs to be processed by each of the machines. All jobs follow the same predetermined route by which they are processed on the machines.
- **Flexible flow shop** (FFc) - an extension of the flow shop environment, which consists of c work centres, and each centre consists of a certain number of parallel machines. All jobs need to be processed on each of the work centres following the same route, however, any of the parallel machines in the work centre can be used in order to process the job.
- **Job shop** (Jm) - consists of m machines in series and n jobs, where each job j is divided into o_j operations. Each operation of a job needs to be processed by one of the machines in an order which is predefined for each job. The number of operations can be less than the number of machines, meaning that jobs do not need to visit each of the machines.
- **Flexible job shop** (FJc) - an extension of the job shop environment which consists of c work centres, and each centre consists of a certain number of parallel machines. Each job j is divided into o_j operations, and each operation needs to be processed by one of the work centres following a route predefined for that job. However, any of the parallel machines in the work centre can be used in order to process the job.
- **Open shop** (Om) - consists of m machines in series. Each job j can be divided into o_j operations, where each operation needs to be processed by one of the machines. The order by which the operations are processed by the machines can be determined freely for each job, it is only important that all of them are processed.

These machine environments are additionally grouped into two categories: single stage and multi-stage. Single stage environments are those in which each job needs to be processed only on a single machine in order to be completed. The single machine, identical machines, uniform machines, and unrelated machines environments belong to this category. On the other hand, in multi-stage environments jobs need to be processed on several machines in order to be completed. The flow shop, flexible flow shop, job shop, flexible job shop, and open shop environments belong to this category.

Possible entries in the β field, which denotes additional scheduling constraints and characteristics, are [1, 50]:

- **Release dates** (r_j) - denotes that release times are used and that jobs can not be scheduled before their release time r_j. Otherwise, if this entry is not present, all jobs are available from the start and can be scheduled at any time.

- **Preemptions** (*prmp*) - denotes that, while executing on a machine, a job can be interrupted and another job can be scheduled on that machine. The work done on the interrupted job is not lost, so when the job is scheduled again it executes only for its remaining processing time.
- **Precedence constraints** (*prec*) - denotes that certain jobs need to be completed in order for other jobs to start executing.
- **Sequence dependent setup times** (s_{ijk}) - denotes an additional cost which is incurred when job k is executed on machine i after job j has finished executing. This constraint represents additional time needed to prepare the machine for the execution of a new job.
- **Job families** ($fmls$) - denotes that there exist F families of jobs. Jobs of the same family can execute one after another, without the need for any setup. On the other hand, when jobs from different families execute one after another, a certain setup cost is incurred.
- **Batch processing** ($batch(b)$) - denotes that machines can process a batch of b jobs simultaneously, and the execution time of the entire batch depends on the longest executing job.
- **Breakdowns** ($brkdwn$) - denotes that some machines can be unavailable in certain periods of time.
- **Machine eligibility restrictions** (M_j) - denotes that not all machines are able to process all jobs, but rather some machines can only process certain subsets of jobs.
- **Permutation** (*prmu*) - denotes that the order in which the jobs are scheduled on the first machine needs to be maintained throughout all other machines. This constraint is only applicable in the flow shop environment.
- **Blocking** ($block$) - denotes that buffers exist between different machines, and if a buffer is full the machine filling the buffer may not release the job when it is completed until there is available space in the buffer. This constraint is only applicable in the flow shop environment.
- **No-wait** (*nwt*) - denotes that a job is not allowed to wait between the execution on two successive machines. As a consequence, the starting time of a job needs to be delayed to ensure that the job can be processed by all machines, without having to wait for any of the machines. This constraint is only applicable in the flow shop environment.
- **Recirculation** (*rcrc*) - denotes that a job may visit the same machine or work centre more than once. This constraint is applicable in the job shop and flexible job shop environments.
- **Restrictions on the number of jobs** (*nbr*) - denotes the restriction on the maximum number of jobs.

- **Restrictions on the number of operations in jobs** (n_j) - denotes the restriction on the maximum number of operations of a job.
- **Restrictions on the processing times** (p_j) - denotes the restriction on the values of the processing times.
- **Deadlines** ($\bar{d}_j$) - denotes that each job needs to be completed before its deadline.

Lastly, some of the scheduling objectives which can be present in the γ field include [1, 50, 51, 52, 53]:

- **Makespan** (C_{max}) - denotes the completion time of the last job that leaves the system:

$$C_{max} = \max_j(C_j). \tag{2.5}$$

- **Maximum flowtime** (F_{max}) - denotes the maximum flowtime achieved by any of the jobs:

$$F_{max} = \max_j(F_j). \tag{2.6}$$

- **Maximum tardiness** (T_{max}) - denotes the maximum tardiness achieved by any of the jobs:

$$T_{max} = \max_j(T_j). \tag{2.7}$$

- **Total weighted completion time** (Cw) - denotes the weighted sum of all completion times:

$$Cw = \sum_j w_{Cj} C_j. \tag{2.8}$$

- **Total weighted tardiness** (Twt) - denotes the weighted sum of tardiness values of all jobs:

$$Twt = \sum_j w_{Tj} T_j. \tag{2.9}$$

- **Total flowtime** (Ft) - denotes the sum of flowtimes of all jobs:

$$Ft = \sum_j F_j. \tag{2.10}$$

- **Weighted number of tardy jobs** (Nwt) - denotes the weighted sum of all tardy jobs:

$$Nwt = \sum_j w_{Tj} U_j. \tag{2.11}$$

- **Weighted earliness and weighted tardiness** ($Etwt$) - denotes the sum of the total weighted tardiness and the total weighted earliness:

$$Etwt = \sum_j (w_{Ej} E_j + w_{Tj} T_j). \tag{2.12}$$

- **Machine utilisation** (M_{ut}) - denotes the difference between the maximum utilisation and minimum utilisation of all machines:

$$M_{ut} = \max_i \left(\frac{P_i}{C_{max}} \right) - \min_i \left(\frac{P_i}{C_{max}} \right), \tag{2.13}$$

  where P_i is defined as the sum of processing times of all jobs which were executed on machine with index i.

The previously enumerated criteria denote only those which will be considered in this thesis. The M_{ut} criterion is a special criterion defined for this thesis. The goal of this criterion is to evenly distribute the load across all machines, to avoid the situation in which some machines would do little to no processing, while others would be overloaded. Although this criterion would rarely be used as the main scheduling criterion, it can nevertheless act as a secondary criterion in scenarios where load balancing is likewise essential.

2.2 Scheduling conditions

Aside from the different environments and constraints described in the previous section, scheduling can also be performed under various conditions depending on the availability of the parameters, the reliability of parameters, and the manner in which the schedule is constructed.

Based on the reliability of the parameters, scheduling can be divided into two groups:

- **Deterministic scheduling** - in which it is presumed that the values of all parameters are known with a satisfactory precision, regardless of the point in time when they become available.
- **Stochastic scheduling** - in which exact values for some parameters are not known until a certain moment in time (for example, the real execution time of a job will only be known after the job finishes with its execution). However, the parameter values are not completely unknown, but are defined through certain stochastic functions.

Depending on the availability of the parameters, scheduling is divided into:

- **Offline scheduling** - in which it is presumed that all parameters and their values are known and available before the start of the execution of the system. For example, the total number of jobs with their release times and processing times is known before the system starts executing.
- **Online scheduling** - in which it is presumed that not all parameter values are known from the start, but rather become available during the execution of the system. For example, it is not known when the next job will be released into the system, and the parameters of the job, like processing times and the due date, become available only when the job is released into the system.

Finally, depending on the manner in which the schedules are created, scheduling procedures are divided into:

- **Static scheduling** - in which the entire schedule is constructed before the system starts with its execution. This kind of scheduling process requires that all parameters are known in advance, meaning that it can be only applied under offline scheduling conditions.
- **Dynamic scheduling** - in which the schedule is constructed incrementally in parallel with the execution of the system. Procedures which belong to this category usually work in a way that they only determine the next state of the system, and that they are invoked each time a change occurs in the system (for example the release of a new job or completion of the execution of a job). Dynamic scheduling can be applied under both, offline and online scheduling conditions, meaning that dynamic scheduling procedures may use certain information about the future of the system, if it is available.

2.3 Methods for solving scheduling problems

The methods for solving scheduling problems are usually divided into three groups [1, 54, 55]:

- **Exact algorithms** - represent procedures that can find the optimal solution of a scheduling problem. Some notable procedures which belong to this group are dynamic programming, the branch and bound algorithm, and different mathematical programming techniques (linear and integer programming) [1, 56, 57, 58, 59]. Unfortunately, all these procedures are quite time consuming, and are therefore not applicable for larger problem instances. In addition, since these procedures extensively search the entire solution space of the problem, they belong to the category of static scheduling procedures and can therefore be applied only under offline scheduling conditions.
- **Approximation algorithms** - represent procedures that can produce solutions in polynomial time and the solutions are guaranteed to be within a fixed percentage of the actual optimum [57, 59, 60, 61].
- **Heuristic algorithms** - represent procedures that give no guarantee on the quality of the obtained solutions. This group is additionally divided into two subgroups depending on how the solutions are generated:
  - **Improvement heuristics** - start with a complete schedule or schedules and try to iteratively improve them by performing certain modifications on the schedule. This category includes procedures like genetic algorithms [62, 63, 64], particle swarm optimisation [65, 66], ant colony optimisation [66, 67], tabu search [68, 69, 70], simulated annealing [71, 72, 73], and many others. Methods which belong to this category have been extensively used for solving various scheduling problems [9, 74, 75, 76, 77, 78, 79].

– **Constructive heuristics** - start from an empty schedule and incrementally construct it by adding one job at a time. The most prominent representative of this group are dispatching rules [10, 11, 12, 13, 80, 81, 82].

2.4 The unrelated machines environment

As described previously, the unrelated machines environment consists of n jobs which need to be scheduled on one of the m available machines. The characteristic of this environment is that each machine has a different processing speed for each of the jobs, and therefore the processing speeds can vary freely across all machines. Scheduling in the unrelated machines environment can be found in many practical real world examples, such as in multiprocessor computers, landing lanes in airports, operating rooms in hospitals, circuit board manufacturing, semiconductor manufacturing, group technology cells, painting and plastic industries, injection moulding process and remanufacturing [83, 84, 85]. The rest of this section will describe several methods for solving scheduling problems in the unrelated machines environment.

2.4.1 Improvement heuristics

As mentioned previously, improvement heuristics start with a complete schedule, which can be generated randomly or by some other heuristic, and improve it iteratively by applying various modifications on the schedule. The advantages of improvement heuristics is that they usually achieve good results in a reasonable amount of time. Besides that, they are very flexible and can be applied to scheduling problems with various criteria and constraints. The greatest disadvantage of improvement algorithms is that they can usually be applied only in offline scheduling, since they need to start with an already complete schedule which they try to improve. Although different metaheuristic approaches are the most common representative of this group of algorithms, there are many additional heuristics specifically designed for various scheduling problems in the unrelated machines environment [86, 87, 88, 89].

Metaheuristic algorithms have commonly been used for solving different unrelated machines scheduling problems. Although many different metaheuristic algorithms, like tabu search [84, 90], simulated annealing [91] and ant colony optimisation [92, 93] are used for solving scheduling problems in the unrelated machines environment, genetic algorithms are still the most widely used approach for solving this problem. Genetic algorithms (GAs) have been applied to solve numerous scheduling problems, be it by themselves or in combination with other approaches [94, 95, 96, 97, 98, 99, 100, 101].

One of the most important steps in metaheuristic algorithms is to define the solution representation. Since the unrelated machines scheduling problem belongs to the category of combinatorial problems, permutation representations are appropriate for representing the solutions. Figure 2.1 represents an example of a solution encoded in the permutation representation. In this figure a solution is encoded for the scheduling problem with three machines and ten jobs. The first array in the solution represents a permutation of the jobs, which determines the sequence in which they will be executed. In this example, the job with the index 9 will be executed first. However, this information alone is not enough to create a schedule, since the mapping of jobs to machines is not specified. Therefore, a second array of integers is used for specifying which job is executed on which machine. The example in figure 2.1 represents a schedule in which the jobs with the indices 9, 2, 0 and 1 will be executed (in that order) on the machine with the index 0, jobs with the indices 7, 8 and 6 will be executed on the machine with the index 1, while jobs with indices 4, 3 and 5 will be executed on the machine with index 2.

Figure 2.1: The permutation solution representation

9	4	3	7	2	0	8	1	5	6
0	2	2	1	0	0	1	0	2	1

Naturally, other solution representations can also be used, given that a corresponding decoding scheme is defined. Figure 2.2 represents a floating point encoding scheme of the same solution which was previously encoded with the permutation encoding scheme. The floating point encoding scheme consists of only one array of n floating point numbers from the interval $[0,1]$. Each floating point number in the array represents a priority value associated with the job of the corresponding index. This means that the priority value on the index 0 in the array represents the priority value of the job with the index 0. Based on the priority values, it is possible to construct the sequence in which the jobs need to be executed. This is performed in a way that jobs with a smaller priority value need to be executed first. Since the job with the index 9 has the smallest priority value (0.03), it will be executed first. The sequence for the other jobs is determined in the same way. In order to determine the mapping between machines and jobs the interval $[0,1]$ is divided into m subintervals. Depending to which subinterval the priority value belongs, it will be scheduled on the corresponding machine. For the considered example which consists of three machines the interval $[0,1]$ will be divided into three subintervals: $[0,0.33>$ for the machine with index 0, $[0.33,0.66>$ for the machine with index 1, and $[0.66,1]$ for the machine with index 2. Therefore, jobs with indices 0, 1, 2 and 9 will be mapped to machine with the index 0, jobs with the indices 6, 7 and 8 will be mapped to the machine with the index 1, while jobs with the indices 3, 4, 5 will be mapped to machine with the index 2.

Figure 2.2: The floating point solution representation

0.27	0.31	0.15	0.77	0.70	0.89	0.62	0.43	0.47	0.03

2.4.2 Dispatching rules

Dispatching rules (DRs) are simple constructive heuristics which incrementally construct the schedule. This is achieved by assigning priority values to jobs and machines, and then whenever a machine is free the job with the highest priority is selected and scheduled. DRs usually consist of two parts, a priority function, which is used to determine the priorities of jobs and machines, and a schedule generation scheme, which uses the priority values in order to construct the schedule. The advantages of DRs are their fast execution speed, which can almost be considered negligible when compared to some improvement algorithms, their applicability in dynamic environments, and the possibility to rapidly adapt to the changing conditions of the scheduling environment. However, DRs also have certain disadvantages. One of them is that since they create the schedule incrementally, they usually achieve the worst results among all the aforementioned methods for solving scheduling problems. In addition, the design of good DRs is usually a lengthy trial and error process which needs to be performed by domain experts. This is especially problematic since there exist many different scheduling objectives and conditions for which such DRs would need to be designed. Finally, the performance of a DR also depends on the problem instance that it is applied on. However, it is impossible to know up front which DR is best suited for solving the given problem instance. Because of that reason, it is possible that a DR, which achieves poor results on the given problem instance, is selected.

DRs which were designed for solving problems in the unrelated machines environment include (in all cases it is presumed that jobs with a higher priority value need to be scheduled sooner):

- **Minimum completion time** (MCT) [13, 102] - jobs are selected in provisional order and the priorities of the selected job on all machines are calculated as

$$\pi_{i,j} = \frac{1}{\max(mr_i, time) + p_{ij}},$$

  where mr_i represents the time when machine i becomes available, and *time* represents the current time of the system. In this way jobs will be scheduled on the machine on which they will be completed the soonest.

- **Minimum execution time** (MET) [13, 102] - determines the priorities of jobs as

$$\pi_{i,j} = \frac{1}{p_{i,j}}.$$

Therefore, jobs will be scheduled based only on their processing times, so that each job is scheduled on the machine on which it achieves its minimum processing time. This can naturally lead to situations in which a great amount of jobs is waiting to be processed on a single machine, while the other machines remain free. In order to avoid this, jobs can be selected by their processing time, but executed on the machine on which they achieve their minimum completion time.

- **Earliest release date** (ERD) [1] - determines the priorities of jobs as

$$\pi_j = \frac{1}{r_j}.$$

  This means that jobs will be scheduled in order by which they became available. The job with the highest priority will be scheduled on the machine on which it achieves its minimum completion time.

- **Longest processing time** (LPT) [1] - determines the priorities of jobs as

$$\pi_{i,j} = p_{i,j}.$$

  Jobs with the longest processing time will therefore be selected first and scheduled on the machine on which they achieve their minimum completion time.

- **Weighted shortest processing time** (WSPT) [103] - calculates the priorities as

$$\pi_{i,j} = \frac{w_{C_j}}{p_{i,j}}.$$

  This rule functions similarly as the MET rule, however it additionally considers weights which can be defined for jobs.

- **Maximum standard deviation** (Maxstd) [104] - calculates the standard deviations of processing times for each job, and schedules the one with the highest standard deviation. The selected job is scheduled on the machine on which it achieves its minimum completion time. The intuition behind this rule is to prioritise those jobs which have a high variation in their processing times, since they will have a larger influence on the makespan if scheduled on an inappropriate machine.

- **Switching algorithm** (SA) [13] - uses both the MET and MCT rules in a cyclic fashion depending on the load distribution of the system. The motivation behind this heuristic lies in the fact that the MET rule can create imbalance in the load of the machines by assigning most of the jobs to only a small subset of machines. The MCT rule, on the other hand, tries to even out the load balance across all the machines. Therefore, the SA heuristic uses both rules in order to keep a good balance across all machines, but also to assign jobs to those machines on which they have the smallest processing times. The heuristic uses the *load balance index* to determine when the algorithm should switch from one rule to the other. The index is calculated as

$$\nabla = \frac{r_{min}}{r_{max}},$$

where r_{min} denotes the earliest ready time, and r_{max} the largest ready time of all machines. In addition, two threshold values are also defined: ∇_l and ∇_h. The SA heuristic starts to schedule jobs by using the MCT rule until the load balance index reaches a value of at least ∇_h, when it switches to the MET rule. This will cause the load balance index to decrease over time until it decreases to a value of ∇_l or less, when the SA heuristic switches back to the MCT rule.

- **k-percent best** (KPB) [13] - considers only a certain subset of machines when scheduling a job. The subset of machines is constructed by selecting the $m * \frac{k}{100}$ machines on which the job j achieves the shortest processing times. The job is assigned to a machine from the selected subset on which it achieves the minimum completion time. The purpose of this heuristic is to schedule jobs on machines for which they have the smallest processing times, to prevent them from being scheduled on other machines which could prove to be more suitable for some other jobs which arrive into the system.
- **Ordered minimum completion time** (OMCT) [105] - represents an extension of the MCT rule in which the priorities of the jobs are calculated as

$$\pi_j = \alpha * \sigma + (1 - \alpha) * S,$$

where σ represents the standard deviation of all processing times of job j, $\alpha \in [0, 1]$ a control parameter, and S the *sufferage* value which is defined as the difference between the second smallest completion time and the smallest completion time of job j. The job with the highest priority will be scheduled on the machine on which it achieves its smallest completion time. By using the standard deviation and sufferage values, this rule tries to determine which jobs would execute longer if they are not scheduled on their preferred machine, and gives them a larger priority value.

- **Work queue** (WQ) [106] - selects the machine which has the least workload, i.e. the machine which spent the least time processing jobs. After the machine is selected, the job which achieves its minimum completion time on this machine is scheduled on it. The motivation behind this rule is to evenly distribute the work over all machines.
- **Opportunistic load balancing** (OLB) [102] - schedules a job on the next available machine, regardless of the expected execution time or completion time of that job. The intuition behind this rule is to evenly distribute the load on all machines. Unfortunately, since this rule does not consider the execution times of jobs, it can create schedules with

poor results for the makespan criterion. This can be improved to a certain degree so that if several machines are free at the same time, the job is scheduled on the machine on which it achieves its minimum execution time.

- **Just in time** (JIT) - tries to schedule the jobs as closely to their due dates as possible. For each job the priority value is calculated as

$$\pi_{i,j} = (d_j - p_{i,j} - time)^2,$$

where $time$ denotes the current time of the system. The priority is additionally multiplied with w_{T_j} if the job is late, or w_{E_j} if the job is early. The job with the smallest priority value is selected and scheduled on the machine on which it achieved that priority value.

- **Earliest due date** (EDD) [1, 107] - calculates the priories of jobs as

$$\pi_j = \frac{1}{d_j}.$$

The reasoning behind this rule is to schedule the job with the earliest due date, in order to minimise the tardiness of jobs. The job with the largest priority is scheduled on the machine on which it achieves its minimum completion time.

- **Minimum slack** (MS) [1] - calculates the priorities of jobs as

$$\pi_{i,j} = \max\left(d_j - p_{i,j} - time, 0\right),$$

where $time$ represents the current time of the system. In this rule the job with the smallest priority is selected and scheduled on the machine on which it achieves its minimum completion time. The rule priorities those jobs which are already late or close to being late.

- **Montagne's heuristic** (MON) [108] - calculates the priorities of jobs as

$$\pi_{ij} = \frac{w_{T_j}}{p_{ij}} * \left(1 - \frac{d_j}{\hat{p}}\right),$$

where $\hat{p}$ represents the sum of processing times of all available jobs for machine i. The job with the highest priority is scheduled on the machine on which it achieves its minimum completion time. This rule tries to scale the WSPT rule with an additional slack factor to give priority to jobs which have an earlier due date. A disadvantage of this rule is that the slack factor is not dynamic, but rather constant during the system execution.

- **Weigthed critical ratio** (CR) [108] - calculates the priorities of jobs as

$$\pi_{i,j} = \frac{w_{T_j}}{p_{ij}} \left(\frac{1}{1 + \frac{(d_j - p_{i,j} - time)}{\bar{p}}} \right),$$

  where $time$ denotes the current time of the system, $\bar{p}$ the average processing time of all jobs waiting to be scheduled. The job with the highest priority is scheduled on the machine on which it achieves its minimum completion time. This rule extends the WSPT rule with a dynamic slack factor, by which it gives more priority to jobs which are close to their due dates. The disadvantage of this rule is that if the job is late, the priority continues to grow. In this thesis the CR rule will be used without the weight, since this variant achieved better results.

- **Cost over time** (COVERT) [108, 109] - calculates the priorities of jobs as

$$\pi_{i,j} = \frac{w_{T_j}}{p_{ij}} \max\left(\left(1 - \frac{\max(d_j - p_{i,j} - time, 0)}{k\bar{p}}\right), 0 \right),$$

  where $time$ denotes the current time of the system, $\bar{p}$ the average processing time of all jobs waiting to be scheduled, and k a scaling parameter. The job with the highest priority is scheduled on the machine on which it achieves its minimum completion time. This rule is similar to the CR rule, however it does not allow that the priority of jobs increases the more they are late.

- **Apparent tardiness cost** (ATC) [103, 107, 110, 111] - calculates the priorities of jobs as

$$\pi_{i,j} = \frac{w_{T_j}}{p_{ij}} \exp\left(-\frac{\max(d_j - p_{i,j} - time, 0)}{k\bar{p}} \right),$$

  where $time$ denotes the current time of the system, $\bar{p}$ the average processing time of all jobs waiting to be scheduled, and k a scaling parameter. The rule can be considered a combination of the WSPT and MS rules, and the scaling factor is used to determine which of these rules will have more influence in the ATC rule.

- **Min-min** [12, 13, 102] - calculates the completion time of each available job on all the machines. After that, for each job the machine for which the job achieves its minimum completion time is determined. The job with the overall smallest completion time is selected and scheduled on the machine on which it achieves its minimum completion time. Algorithm 2.1 represents the min-min rule.

- **Max-min** [13, 102] - for each job the rule determines the machine for which the corresponding job achieves its minimum completion time. However, unlike the min-min rule, the max-min rule selects the job with the largest minimum completion time. In that way the max-min rule will prioritise jobs with longer execution times. Algorithm 2.2

Algorithm 2.1 The min-min rule

```
1: while unscheduled jobs are available do
2:     for each released and unscheduled job $j$ do
3:         for each machine $i$ do
4:             Calculate the completion time $ct_{ij}$ for job $j$ and machine $i$
5:         end for
6:     end for
7:     For each job determine the machine on which it achieves its minimum completion time
8:     Select the job which achieves the overall smallest minimum completion time
9:     Schedule the selected job on the machine for which it achieves its minimum completion
       time
10: end while
```

represents the max-min rule.

Algorithm 2.2 The max-min rule

```
1: while unscheduled jobs are available do
2:     for each released and unscheduled job $j$ do
3:         for each machine $i$ do
4:             Calculate the completion time $ct_{ij}$ for job $j$ and machine $i$
5:         end for
6:     end for
7:     For each job determine the machine on which it achieves its minimum completion time
8:     Select the job which achieves the largest minimum completion time
9:     Schedule the selected job on the machine for which it achieves its minimum completion
       time
10: end while
```

- **Min-max** [106] - for each job the rule determines the machine for which the corresponding job achieves its minimum completion time. The job whose minimum processing time divided by the processing time on the selected machine in the previous step has the maximum value will be scheduled on the selected machine. The intuition behind this rule is to schedule the job whose processing time on the selected machine is the closest to the shortest processing time of that job. Algorithm 2.3 represents the min-max rule.
- **Sufferage** [13] - for each job the rule determines the machine for which the corresponding job achieves its minimum completion time. The rule then determines the sufferage value for each job, which is calculated as the difference between the second earliest completion time and the earliest completion time. The job with the largest sufferage value is scheduled on the machine for which it achieves its minimum completion time. The intuition behind this heuristic is to schedule the job which would "suffer" the most if not scheduled on the machine with its minimum execution time. Algorithm 2.4 represents the sufferage rule.

Algorithm 2.3 The min-max rule

```
1: while unscheduled jobs are available do
2:     for each released and unscheduled job $j$ do
3:         for each machine $i$ do
4:             Calculate the completion time $ct_{ij}$ for job $j$ and machine $i$
5:         end for
6:         Determine the machine $i_{mct}$ on which job $j$ achieves its minimum completion time
7:         Calculate the ratio $\frac{\min_i(p_{ij})}{p_{i_{mct}j}}$
8:     end for
9:     Select the job with the largest ratio value and schedule it on the machine for which it
       achieves its minimum completion time
10: end while
```

Algorithm 2.4 The sufferage rule

```
1: while unscheduled jobs are available do
2:     for each released and unscheduled job $j$ do
3:         for each machine $i$ do
4:             Calculate the completion time $ct_{ij}$ for job $j$ and machine $i$
5:         end for
6:         Determine the earliest completion time $mct_1$ of job $j$
7:         Determine the second earliest completion time $mct_2$ of job $j$
8:         Calculate the *sufferage* value of job $j$ as $mct_2 - mct_1$
9:     end for
10:    Select the job which has the largest sufferage value
11:    Schedule the selected job on the machine for which it achieves its minimum completion
       time
12: end while
```

- **Sufferage2** [112] - for each job the rule determines the machine for which the corresponding job achieves its minimum completion time. The rule calculates the sufferage value for each job, but additionally scales this value with the following factor

$$\frac{\min_i(p_{ij})}{\min_i(ct_{ij})},$$

  where ct_{ij} denotes the completion time of job j on machine i. With this scaling factor the rule also incorporates the information about the processing and completion times when selecting the job to be scheduled. The job with the largest scaled sufferage value is selected and scheduled on the machine on the machine for which it achieves its minimum completion time. Algorithm 2.5 represents the sufferage2 rule.
- **Relative cost** (RC) [113] - for each job this rule determines the machine on which the job achieves its minimum completion time. Then for each job it calculates two parameters, namely the *static relative cost* and *dynamic relative cost*. The static relative cost for job j

Algorithm 2.5 The sufferage2 rule

```
1: while unscheduled jobs are available do
2:     for each released and unscheduled job $j$ do
3:         for each machine $i$ do
4:             Calculate the completion time $ct_{ij}$ for job $j$ and machine $i$
5:         end for
6:         Determine the earliest completion time $mct_1$ of job $j$
7:         Determine the second earliest completion time $mct_2$ of job $j$
8:         Calculate the priority value of job $j$ as $\frac{\min_i(p_{ij})}{\min_i(ct_{ij})} * (mct_2 - mct_1)$
9:     end for
10:    Select the job which has the largest scaled sufferage value
11:    Schedule the selected job on the machine for which it achieves its minimum completion
       time
12: end while
```

and machine i is calculated as

$$\gamma_{ij}^{s} = \frac{p_{ij}}{\frac{\sum_{k \in machines} p_{kj}}{m}},$$

while the dynamic relative cost is calculated as

$$\gamma_{ij}^{d} = \frac{ct_{ij}}{\frac{\sum_{k \in machines} ct_{kj}}{m}},$$

where ct_{ij} denotes the completion time of job j on machine i. The total priority of a job is calculated as

$$\pi_{i,j} = \frac{1}{(\gamma_{ij}^{s})^{\alpha} * \gamma_{ij}^{d}},$$

where α represents a scaling factor. When selecting which job should be scheduled on the machine for which it achieves its minimum completion time, this rule tries to balance between the jobs minimum processing time and minimum completion time, and selects the one which has smaller values for both.

- **Longest job to shortest resource - shortest job on fastest resource** (LJFR-SJFR) [106] - for each job the rule determines the machine for which the corresponding job achieves its minimum completion time. In the first step this rule schedules m jobs with the longest minimum completion times to the fastest machines. After this first step the rule alternatively schedules the job with the shortest minimum execution time to the fastest machine, and then the job with the longest minimum execution time to the fastest machine.
- **Minimum execution completion time** (MECT) [114] - represents a combination between the MET and MCT dispatching rules. Algorithm 2.6 represents the outline of MECT. The DR first determines the maximum ready time of all machines mr_{max}. Afterwards, the rule determines the machines on which job j can finish with its execution prior

to mr_{max}. If such machines exist, the one for which job j achieves the minimum execution time is selected. However, if such machines do not exist, the machine on which job j achieves its minimum completion time is selected. Out of all unscheduled jobs, the job which achieves the minimum completion time on the selected machine will be scheduled. The intuition behind MECT is to alternatively use the minimum execution and completion times for performing the scheduling decision. The rule will use the minimum execution time to select the machine on which job j should be executed, if this will not lead to the increase of the makespan. However, if there is no decision which does not increase the makespan, then the machine for which job j achieves its minimum completion time is selected.

Algorithm 2.6 The MECT rule

```
1:  while unscheduled jobs are available do
2:      Let $mr_i$ denote the ready time of machine $i$
3:      $mr_{max} = \max_i(mr_i)$
4:      Let $ct_{ij}$ represent the completion time of job $j$ on machine $i$
5:      for each unscheduled job $j$ do
6:          Let $M'$ represent all machines for which $p_{ij} + mr_i < mr_{max}$
7:          Let $sm_j$ represent the selected machine for job $j$
8:          if $|M'| > 0$ then
9:              $sm_j = \arg\min_{i \in M'} p_{ij}$
10:         else
11:             $sm_j = \arg\min_{i \in M} ct_{ij}$
12:         end if
13:     end for
14:     Schedule the job with the smallest value of $ct_{sm_j j}$
15: end while
```

Chapter 3

Genetic programming

Genetic programming (GP) is a metaheuristic algorithm which simulates natural evolution in order to find solutions for various optimisation problems [115, 116, 117, 118, 119, 120]. It belongs to the category of *evolutionary algorithms* together with genetic algorithms, evolutionary strategy, evolutionary programming, differential evolution and many others. In 1990 John R. Koza has laid down many of the fundamentals of GP which are still used today [16, 121]. Since then, GP was used to solve a vast number of optimisation and classification problems [122], and in many cases it obtained results which are competitive to those achieved by human experts [123]. Because of its ability of representing and evolving complex expressions, GP has often been used as a *hyper-heuristic* [124, 125, 126, 127] to evolve new heuristic procedures for different problems like bin packing [128, 129, 130], vehicle routing problems [131, 132, 133], timetabling [134, 135] and project scheduling [136].

This chapter will give a short overview of the GP algorithm, describing all of its main parts like selection, genetic operators, solution representation, etc. In addition to the standard GP algorithm, two other GP variants which will also be applied in the thesis, namely dimensionally aware GP and gene expression programming, will also be shortly described.

3.1 Standard genetic programming

The GP variant which is used in this thesis is given in Algorithm 3.1. In the first step of the algorithm the initial population is initialised and every individual is evaluated. The evolutionary process is repeated until a certain termination criterion is met. In each step of the evolutionary process k individuals are selected randomly. The best two individuals are used as parents in the crossover operator, to generate a new individual. The newly created individual is then further mutated with a certain probability, and used to replace the worst of the k selected individuals. This type of algorithm is usually called the *steady state* algorithm.

Another very popular GP variant is the *generational* GP algorithm. In this algorithm, after

Algorithm 3.1 Standard steady state GP algorithm

```
1: Initialise the population $P$ and evaluate all individuals in it
2: do
3:     Randomly select $k$ individuals from the population $P$
4:     Perform the crossover operator on the best two of the $k$ selected individuals to create a
       new individual
5:     Perform the mutation operator on the new individual with a certain probability
6:     Replace the worst of the $k$ selected individuals with the newly created individual
7: while termination criterion is not met
```

the mutation is performed on the new individual, it does not replace the worst individual, but the newly created individual is rather placed in a new population. Individuals are created until the new population reaches the same size as the current population, at which point the old population is replaced by the new population, and the process is repeated until the termination criterion is satisfied. The steps of this procedure are given in Algorithm 3.2. An important problem with this type of GP is that by deleting the old population all good individuals which were previously generated are lost. In order to prevent this, usually one or more of the best individuals are directly transferred into the new population, in order to retain the best found solutions thus far. The property of preserving the best individual during the entire run of the algorithm is called *elitism*.

Algorithm 3.2 Generational GP algorithm

```
1: Initialise the population $P$ and evaluate all individuals in it
2: do
3:     $newP = \emptyset$
4:     while $|newP| < |P|$ do
5:         Select $k$ individuals from the population $P$
6:         Perform the crossover operator on the best two of the $k$ selected individuals in order
           to create a new individual
7:         Perform the mutation operator on the new individual with a certain probability
8:         Place the new individual in $newP$
9:     end while
10:    $P = newP$
11: while termination criterion is not met
```

3.1.1 Solution representation

Since GP is used for evolving programs, expressions, or mathematical functions it needs to use a solution representation with which it is simple to represent such structures, and on which it is easy to perform structural changes through different genetic operators. The most commonly used representation is the *tree* representation, in which the solutions are encoded in a tree structure. Figure 3.1 demonstrates a sample tree representation of an individual which can be decoded into the following expression: $x - 3 + \frac{x^2}{2}$.

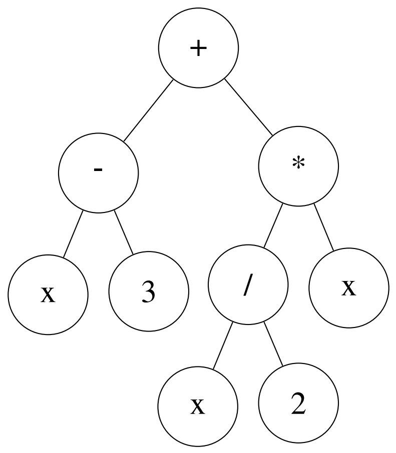


Figure 3.1: Tree representation of an individual

When specifying the solution representation for a certain problem, one of the most important steps is to define which nodes will be used to represent the solution. The set of nodes used by GP is called the *primitive set*. Selecting nodes for the primitive set is quite important since a poor choice of nodes can cause not only slow convergence of GP, but can also prevent GP from obtaining good solutions. Therefore, the nodes need to be carefully selected to introduce the *sufficiency property*, and allow GP to represent good solutions. Furthermore, the primitive set should also be kept minimal, since the inclusion of a large number of nodes in the algorithm can cause slow convergence, since the search space drastically increases with the number of nodes in the primitive set. The nodes in the primitive set can be divided into who groups:

- **Terminal nodes** - represent variables or constants in the expression. These nodes need to be carefully modelled and selected so that they represent all the essential characteristics of the problem. Terminal nodes can only appear in the leaves of the tree.
- **Function nodes** - represent various operations that can be performed on one or more nodes. Through function nodes it is possible to model different mathematical operations (addition, subtraction, trigonometric operations), logical operators (and, or, xor), or even certain control structures (*if* statement). It is important to ensure that the behaviour of function nodes is well defined for all possible inputs, so that the tree can be evaluated. For example, the division operation is not defined when the divisor is zero. This problem is usually solved by introducing *protected* operators, which define a default return value for such exceptional cases. Another way of solving this problem would be to use *interval arithmetic* for determining which individuals are invalid on the domain so that they can be removed from the population [137].

In addition to the primitive set, an additional parameter which determines the maximum size of the tree is usually defined. This parameter is most often defined as either the maximum number of nodes which the tree can consist of, or as the maximum depth of the tree. The objective of this parameter is to restrict the growth of individuals beyond a certain size. This parameter is used to deal with a common problem which appears in GP, called *bloat* [120, 138, 139, 140, 141]. Bloat is defined as the growth of individuals without a significant improvement in terms of their quality. This phenomenon needs to be avoided since larger individuals are more difficult to interpret and evaluate.

Apart from the tree representation of individuals, many alternative solution representations were proposed for GP. Some of the more popular alternative GP representations are graph based GP [142], linear GP [143], grammar based GP [144, 145], and Cartesian GP [146].

3.1.2 Initialisation

The first step in the execution of GP is usually the initialisation of the initial population. Individuals which form the initial population are usually generated randomly, however it is also possible to generate the initial population by using some specific heuristic procedures. Usually, one of the following three individual initialisation methods are used: *full*, *grow*, and *ramped half-and-half*.

The *full* initialisation method creates individuals in which all leaf nodes will have the depth that is equal to the maximum allowed depth of the individual [120]. The generation process is quite simple, if the current node which needs to be generated has the depth which is smaller than the maximum allowed depth, then a random function node will be generated. Otherwise, if the depth of the node is equal to the maximum allowed depth of the tree, then a random terminal will be generated. An important drawback of this generation method is that it will generate trees which will all have very similar shapes, and therefore the algorithm will need to rely on the genetic operators to introduce more diversity into the shapes of the individuals. Figure 3.2 represents an example of creating an individual with the maximum allowed depth of 2, by using the full generation method.

The *grow* initialisation method does not impose the constraint that all terminal nodes need to have the same depth, as was the case with the full method [120]. In this initialisation method, if the current node which is to be generated has a depth smaller than the maximum depth, then either a function or a terminal node can be generated. However, if the current node which needs to be generated has the maximum depth, then only terminal nodes can be generated to ensure that the method does not create a larger tree than it is allowed to. The grow method will therefore be able to create individuals with much more variability in their shapes. Figure 3.3 represents an example of how an individual with the maximum allowed depth of 2 is generated using the grow method.

To further increase the diversity of the population, the *ramped half-and-half* initialisation approach is commonly used [120]. This method initialises half of the population by using the

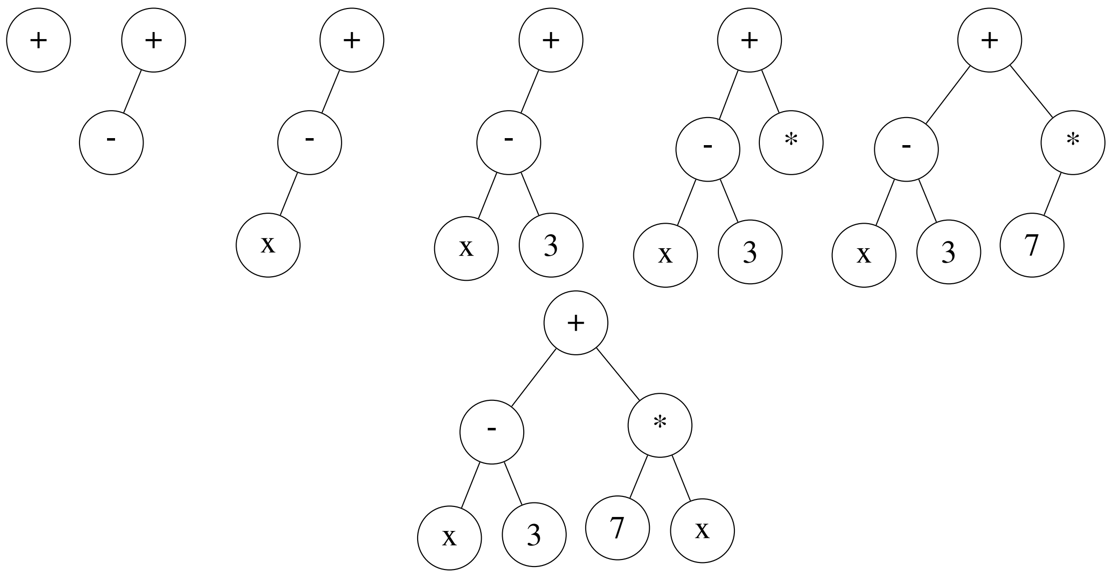


Figure 3.2: Example of the full generation method

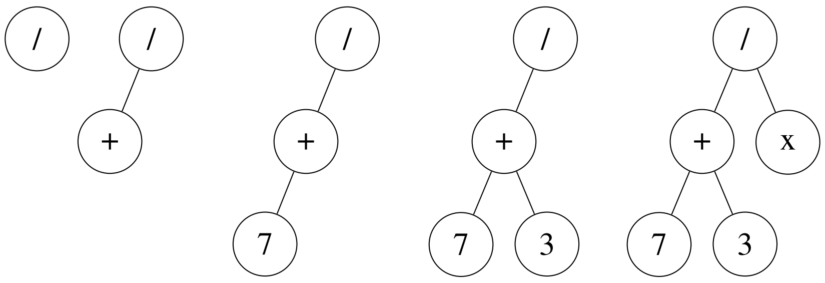


Figure 3.3: Example of the grow generation method

grow method, and the other half by using the full method. In addition, to improve the diversity of the sizes and shapes of individuals, the method does not generate all individuals with the same depth limit, but rather a random depth limit, which is smaller or equal to the maximum allowed depth of individuals, is used for each individual independently.

3.1.3 Evaluation

In order to guide the search process of GP, it is mandatory to define a measure of quality for each solution. For that purpose a *fitness function* is defined, which returns a numeric value for each individual. This value denotes the fitness or quality of a solution, and is used to determine which individuals represent "good" solutions, and which individuals represent "bad" solutions. The fitness function needs to be designed carefully, since it has a major effect on the convergence of the algorithm. This is especially true for GP, since individuals in GP do not represent a solution for only one concrete problem instance, but can represent a solution which can be applied on a variety of problem instances. Therefore, when the quality of a solution is determined by the fitness function, the solution is usually evaluated on several problem instances, and the value returned by the fitness function represents the measure of quality obtained on all the tested problem instances. Thus, it is important that the fitness function uses a set of problem instances with different properties for evaluating solutions. Since individuals are tested on several problem instances, the evaluation of the fitness function can become quite time consuming for certain problems.

3.1.4 Selection

The selection mechanism represents an essential part of GP, since a good selection mechanism should ensure that better individuals have a higher probability of surviving and producing offsprings, while worse individuals should have a higher probability of being eliminated from the population. Two popular selection mechanisms are the tournament selection and roulette wheel selection.

In *tournament selection*, k individuals are randomly selected from the population. Out of these k selected individuals, the best two individuals are selected as parents for the crossover operator. After the new individual is created, it will be placed in the new population in the generational GP, while in the steady state GP it will be used to replace the worst individual in the tournament. The size of the tournament is usually set to three individuals. The benefits of this selection are that it is simple to implement and can be performed very fast. Figure 3.4 represents an example of performing the tournament selection for the tournament of size three.

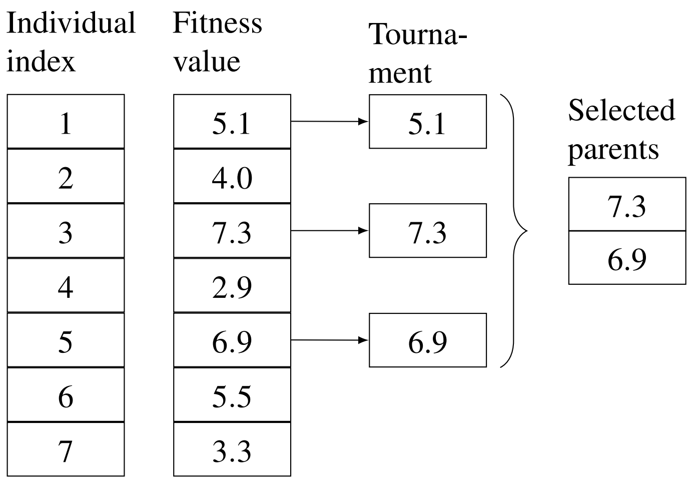
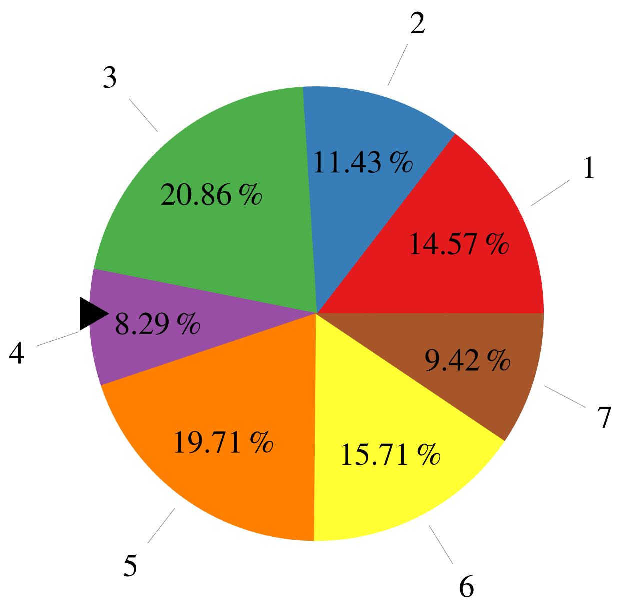


Figure 3.4: Example of the tournament selection

In *roulette wheel selection* a probability of being selected is defined for each individual. The value of this probability is proportional to the fitness of the individuals. Therefore, individuals with a better fitness value will have a higher probability of being selected as parents, while individuals with a lower fitness value will have a smaller probability of being selected. The selection probability of individuals is usually calculated as $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, where f_i is the fitness of individual i in the population, and N is the size of the population. Unfortunately, this method has some drawbacks which can largely influence the execution of the algorithm. First of all, this selection method can be quite computationally intensive, since the selection probabilities of individuals need to be recalculated every time there is a change in the population. Another, even more serious problem, is that if several solutions have a much higher fitness value than the rest of the population, those solutions will be selected as parents for the crossover operator in most of the cases. This can lead to the situation in which the population consists of individuals which are descendants of only a few individuals. If this happens, there is a high probability that the algorithm will converge to a local optimum, since the entire search process will be guided by only a few individuals. Figure 3.5 represents an example of the roulette wheel selection. The entire selection process can be visualised in the form of a wheel where each individual takes up a certain part of the wheel, and the size of the part is proportional to the fitness of the individual. A fixed point is placed on the wheel (denoted with a triangle), and every time an individual needs to be selected, the wheel is spun and the individual on which the marker lands is selected. In this example the individual with the index 4 would be selected.



Index	Fitness
1	5.1
2	4.0
3	7.3
4	2.9
5	6.9
6	5.5
7	3.3

Figure 3.5: Example of the roulette wheel selection

3.1.5 Genetic operators

To be able to achieve good solutions, GP needs the possibility to search the solution space by performing different operations on individual solutions. In GP these operations are called *genetic operators*. Their purpose is to introduce changes in the individuals in order to generate better solutions. The two most prominent genetic operators in GP are *crossover* and *mutation*.

Crossover is usually performed on two individuals which are called *parents*, from which it produces a new individual called a *child*. The motivation behind this operator is to combine properties from two good individuals to obtain an even better individual as a result. Therefore, crossover usually takes one part of the solution tree from one parent, and another part from the other parent. These two parts are then combined in a certain way to form a new individual. Since crossover is performed by combining nodes from both parents, the child individual will only consist of the genetic material which was already present in its parents. A wide variety of crossover operators are defined for GP: subtree crossover, one-point crossover, uniform crossover, context-preserving crossover, size-fair crossover, and many others.

Subtree crossover is one of the simplest crossover operators [120]. In each tree a node is randomly chosen as the crossover point. The child is formed by replacing the subtree rooted at the crossover point in the first parent, by the subtree rooted at the crossover point from the second parent. In this variant of crossover, not all nodes have an equal probability of being selected as the crossover point. The reason for this is that the tree consists largely of terminal nodes, and therefore this crossover would mostly select those nodes as crossover points, which would cause only a small amount of genetic material to be exchanged. Therefore, an additional constraint is introduced which ensures that function nodes are selected in 90% of cases, while in the other 10% terminal nodes are selected. Figure 3.6 represents an example of the subtree crossover. The red nodes denote the selected crossover points.

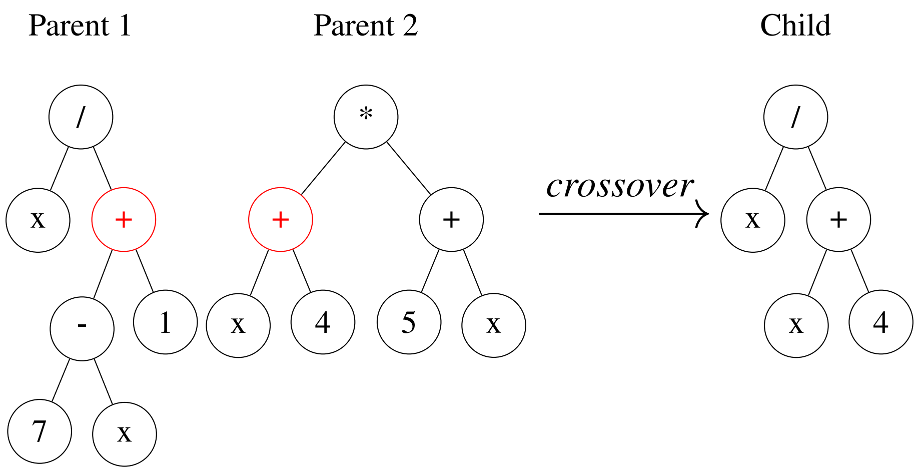


Figure 3.6: Subtree crossover

One-point crossover defines a *common region*, in which both parents have the same shape [120]. In this crossover, only nodes which belong to the common region can be selected as the crossover point. Furthermore, the crossover point is on the same position in both parents. Therefore, this crossover will replace a subtree from one parent with a subtree located at the same depth and position from the other parent. Figure 3.7 represents an example of the one-point crossover operator. The nodes which belong to the common region are denoted in blue, while the red nodes denote the selected crossover points.

Uniform crossover, in the first step, determines the common region between the selected parents [120]. The child is created so that for each node in the common region it is randomly

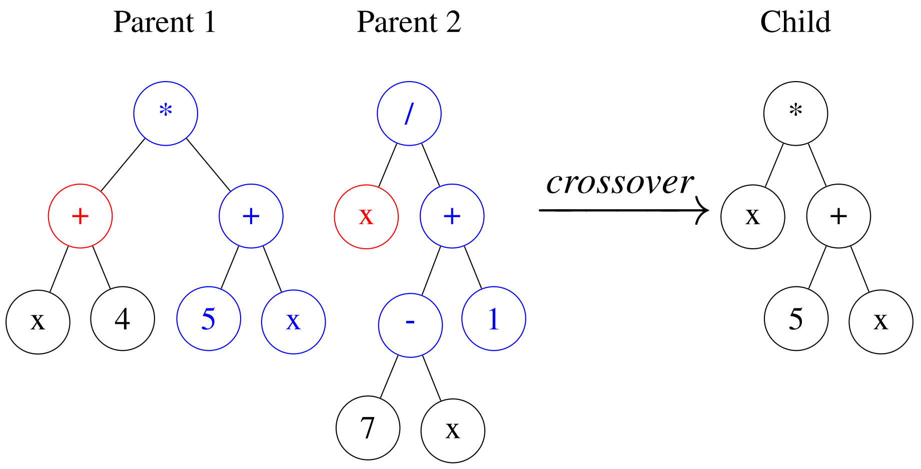


Figure 3.7: One-point crossover

decided whether the node at that position will be inherited from the first or the second parent. If a node lies at the border of the common region, and is a function node, then the entire subtree rooted in that node is also inherited by the child. This crossover is similar as the one-point crossover, but it allows for a greater variability in the resulting child individual. Figure 3.8 represents an example of how the uniform crossover is performed. The blue nodes in the parent individuals denote nodes which belong to the common region. In the child individual, nodes which are inherited from the first parent are denoted in red, while the nodes which are inherited from the second parent are denoted in green.

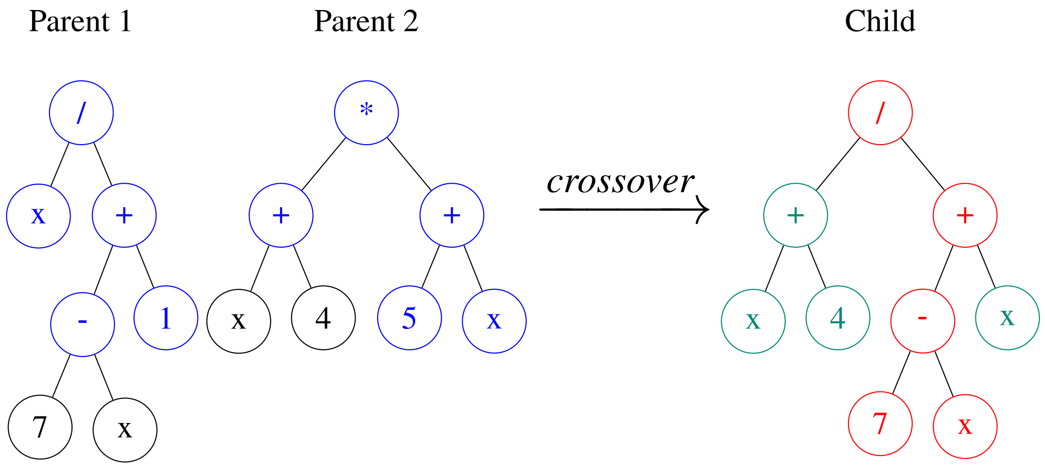


Figure 3.8: Uniform crossover

Context-preserving crossover defines coordinates for each node in the tree [147]. The coordinates of a certain node are defined as a tuple $T = (b_1, b_2, \ldots, b_n)$, where n represents the depth of the node, and b_i determines which branch was chosen at the depth i. The branches of nodes are usually enumerated from left to right, starting with the value 1, while the root node of the tree is denoted with an empty tuple. When choosing the crossover points, only the nodes which have the same coordinates can be selected. Figure 3.9 represents an example of this type of crossover. The coordinates of all nodes are denoted in the figure, while the nodes selected as the crossover points are denoted in red. The chosen node has the coordinates $(2,1)$, since in order to reach the node from the root of the tree, first the second branch needs to be taken at depth 1, and then again the first branch at depth 2.

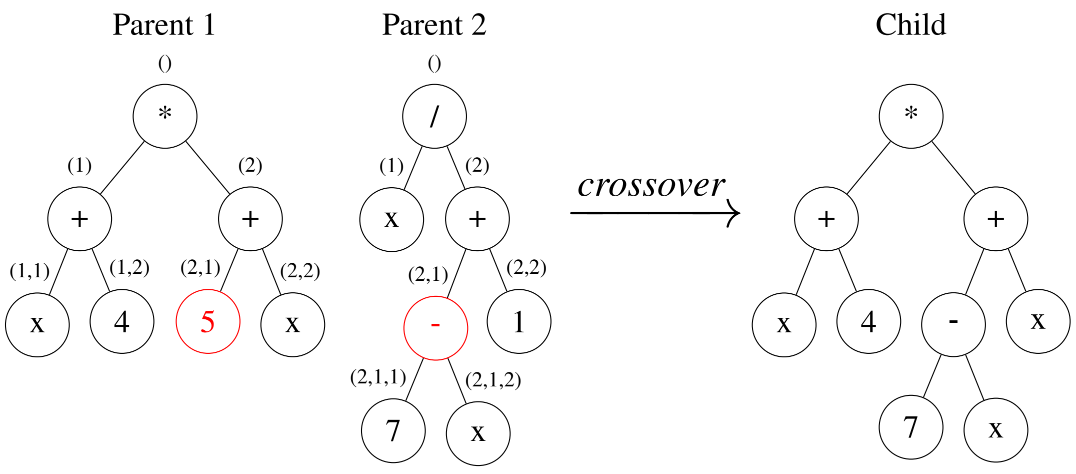


Figure 3.9: Context-preserving crossover

Size-fair crossover randomly selects a node in the first parent which will represent the crossover point [148]. The crossover then calculates the maximum subtree size, which can be selected from the second parent, as $n = 1 + 2 * s$, where s denotes the size of the subtree which will be removed from the first parent. Therefore, only nodes for which the subtree rooted in them does not contain more than n nodes, can be selected as the crossover point in the second parent. The reasoning for such a procedure is to ensure that the subtree which will be copied from the second parent will not be too large when compared to the subtree which was removed. Figure 3.10 represents an example of the subtree crossover. The crossover points are denoted in red. In the example the subtree which will be removed from the first parent is of size 1, therefore only nodes with the subtree of sizes smaller or equal to 3 can be chosen as the crossover point in the second parent.

Mutation performs random changes in the individual, with which it tries to introduce new genetic material. The mutation operator is usually performed on the child individuals generated by crossover. However, mutation is not always performed, but rather with a certain probability. This is due to the fact that there is no guarantee that mutation will increase the fitness of the individual, therefore it is suggested that mutation is performed sparingly in order to keep good solutions which were achieved by crossover. Nevertheless, mutation is important since by using only crossover the entire population will slowly converge to several better solutions, and could thus get stuck in a local optimum. The mutation prevents this, and allows the algorithm to escape local optima, and to reintroduce genetic material which has been lost during the evolution process. Many mutation operators are defined in the literature, some of which are: subtree mutation, Gauss mutation, hoist mutation, node replacement mutation, node complement mutation,

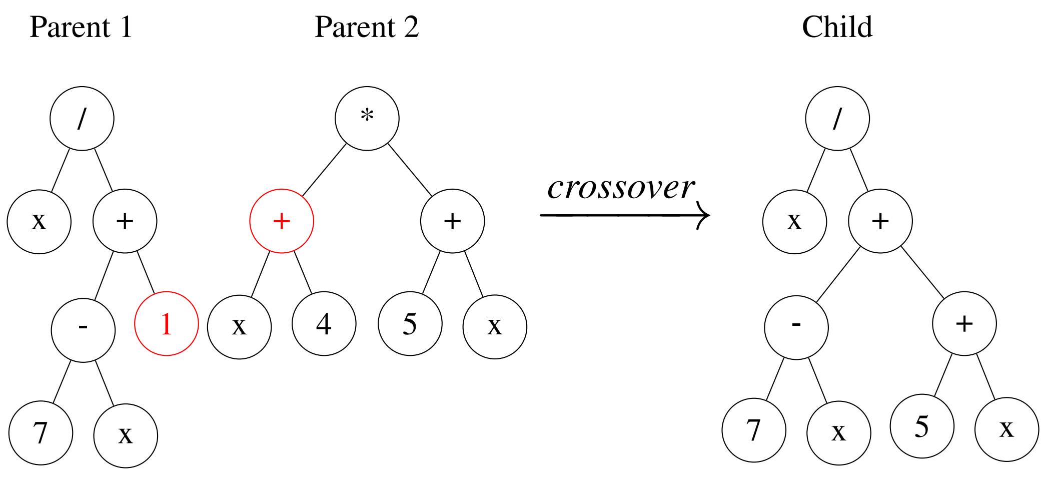


Figure 3.10: Size-fair crossover

permutation mutation, shrink mutation, and many others.

Subtree mutation selects a random node in the individual and replaces the subtree rooted in that node with a randomly generated subtree [120]. This mutation can be performed by using the subtree crossover between the selected individual and a new randomly generated individual. When the subtree mutation is performed in this way it is usually called the "headless chicken" crossover. This mutation can be extended so that it prevents that after the mutation the depth of the individual grows by more than 15% when compared to the individual before mutation. Figure 3.11 represents an example of the subtree mutation. The red node denotes the root of the subtree which is replaced by a randomly generated subtree.

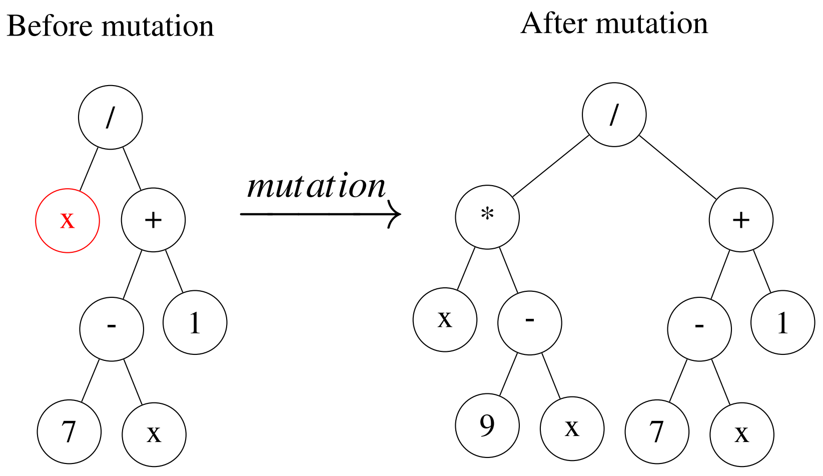


Figure 3.11: Subtree mutation

Gauss mutation can only be applied on terminal nodes which represent a numerical constant [120]. This mutation selects one of such nodes and adds a value which is randomly generated by using the Gaussian distribution. Figure 3.12 represents an example of Gauss mutation where the node which will be mutated is denoted in red.

Hoist mutation randomly selects a node from the individual, and replaces the entire individ-

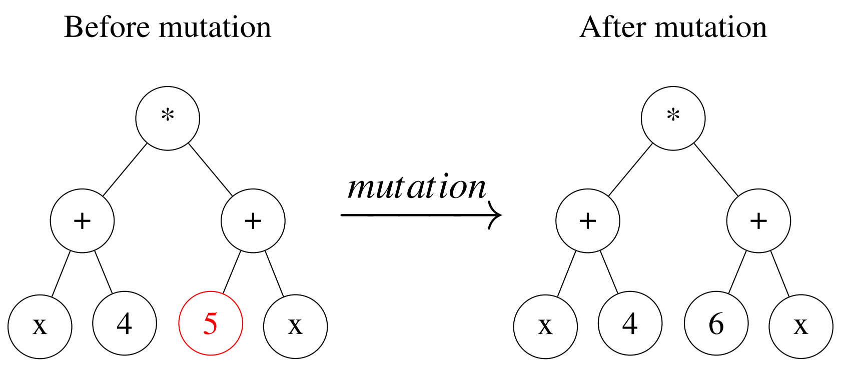


Figure 3.12: Gauss mutation

ual with the subtree rooted at the selected node [120]. Although this mutation does not introduce new genetic material in the individual, it is useful since it reduces the size of the individual, thus creating simpler and more interpretable individuals. In addition, by reducing the size of individuals, this operator also tries to diminish the effects of bloat. Figure 3.13 represents an example of hoist mutation. The red node denotes the root node of the subtree which will replace the original individual.

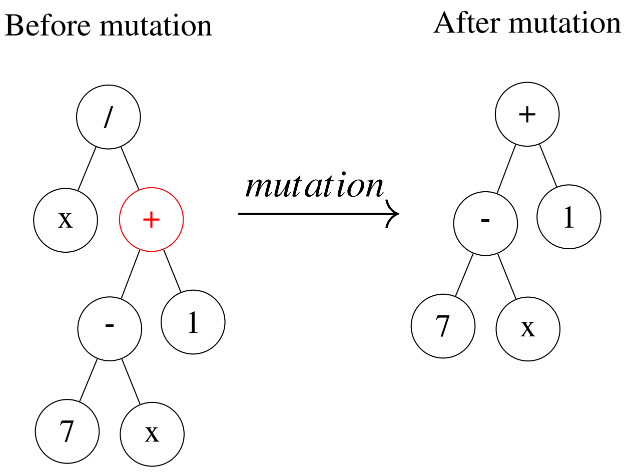


Figure 3.13: Hoist mutation

Node replacement mutation, also known as *point mutation*, randomly selects a node from the individual and replaces it with another randomly generated node [120]. However, the selected node cannot be replaced by just any other node, but only with nodes which will not cause any illegal situations. This means that a terminal node can only be replaced by another terminal node, and a function node with another function node with the same number of arguments. Figure 3.14 represents an example of the replacement mutation, where the red node denotes the node ($-$) which is chosen to be replaced by a new node ($*$).

Node complement mutation functions similarly as the node replacement mutation, however it places additional constraints on the nodes which can be mutated. This mutation can only be applied to nodes for which a complementary node is defined. Although complementary nodes

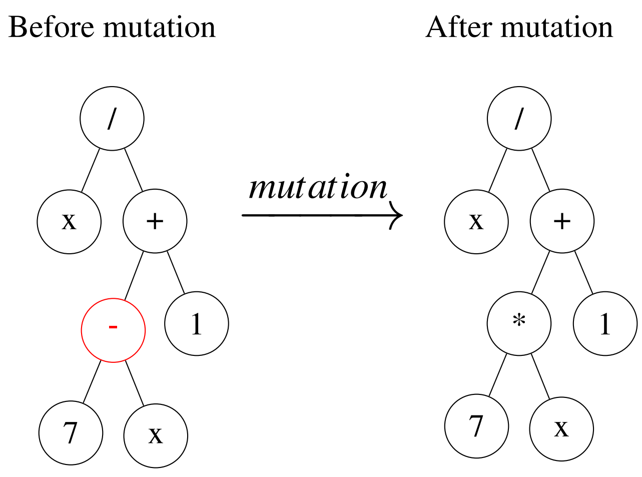


Figure 3.14: Node replacement mutation

can be freely defined by the user, they usually consist of nodes which represent complementary operations like addition and subtraction, or multiplication and division. Naturally, complementary nodes need to have the same number of arguments to ensure that by exchanging them the structure of the tree will remain syntactically correct. Figure 3.15 represents an example of the node complement mutation. The red node denotes the node selected for mutation. In this case the node which represents the subtraction operation is selected, for which a complementary node that performs the addition operation is defined. The complementary node is then used in order to replace the selected node in the individual.

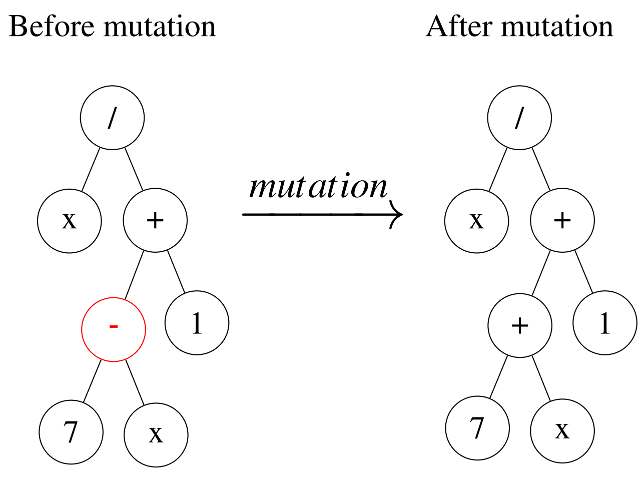


Figure 3.15: Node complement mutation

Permutation mutation randomly selects a function node in the individual and randomly permutes its subtrees [120]. The problem with this operator is that if it is applied on nodes which represent commutative operators it will have no effect, since the result of the operation will stay the same. Therefore, an extension to this operator exists in the form of the *swap mutation*, which restricts the selection only to nodes which represent non-commutative binary operators. Figure 3.16 represents an example of the permutation mutation. The selected node, for which the positions of its two subtrees are interchanged, is denoted in red.

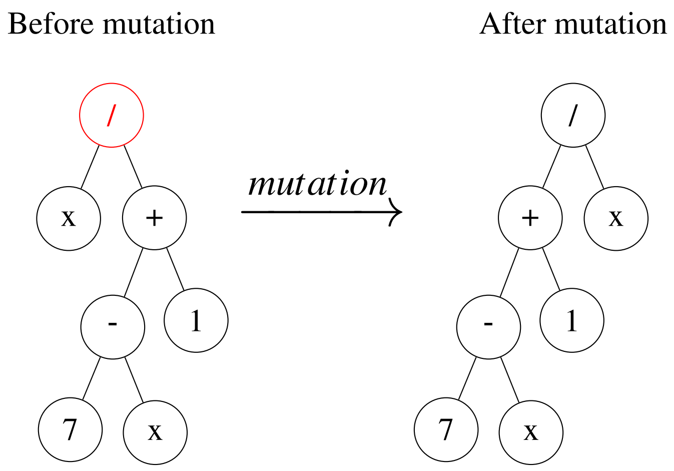


Figure 3.16: Permutation mutation

Shrink mutation selects a random node from the individual and replaces the subtree rooted in that node with a randomly chosen terminal node [120]. The motivation for this mutation operator is to reduce the sizes of the trees in the population, and therefore diminish the effects of bloat. Figure 3.17 represents an example of the shrink mutation. The red node denotes the root node of the subtree which is replaced by a random terminal node. As it can be seen from this example, shrink mutation can have very destructive effects if a node which is close to the root of the individual is chosen.

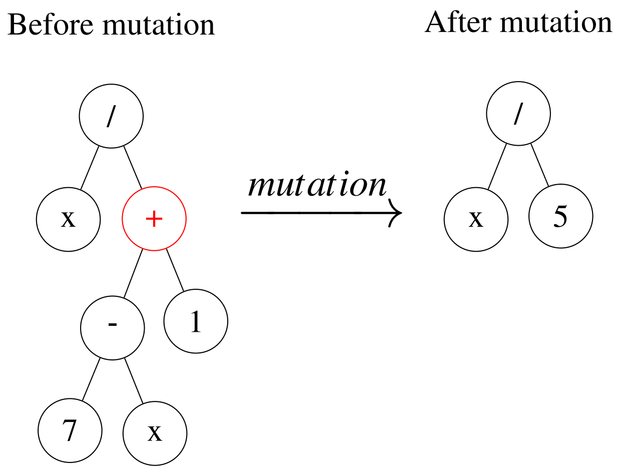


Figure 3.17: Shrink mutation

3.1.6 Termination criteria

To determine when GP should finish with its execution, it is mandatory to define one or more termination criteria. The most commonly used termination criteria in GP are:

- **Maximum number of iterations** - defines that the algorithm should cease its execution when a certain number of iterations is performed. This termination criterion is often used by elimination GP variants.
- **Maximum number of generations** - defines that the algorithm should cease its execution when a certain number of generations are created. This termination criterion is used by generational GP.
- **Maximum number of function evaluations** - defines that the algorithm should cease its execution when a certain number of fitness function evaluations is performed. Since evaluating fitness functions is usually quite a costly operation, this termination criterion is usually used when comparing the performance of different algorithms to make the comparisons objective.
- **Execution time** - defines the maximum amount of time that the algorithm is allowed to execute. This criterion is useful when a certain time limit for obtaining a solution is imposed.
- **Stagnation** - defines that the algorithm should cease with its execution if in a certain number of iterations or generations it does not achieve any improvement in the quality of the best solution.
- **Threshold value** - defines that the algorithm should cease its execution when the fitness value of the best solution has reached a certain threshold value. This criterion can be used when either the optimal value is known in advance, or it is satisfactory to obtain a solution of a specific fitness.

The values for the termination criteria must be chosen very carefully, since GP is sensitive to their values. For example, if GP terminates too early, then solutions of poor quality will be obtained. On the other hand, if GP programming is allowed to run for a too long time it can start to overfit on the problem instances which are used for the evolution process. Therefore, the obtained solutions will achieve good results for instances which were used for learning, but will loose their ability to generalise over unseen problem instances and will perform poorly on them.

3.2 Dimensionally aware genetic programming

In its basic form GP has no notion of any semantic information available through the input variables. For example, terminal nodes could have different units assigned to them, however GP would combine the nodes without considering that information. This can lead to the situation that GP generates expressions which make no sense since they perform illegal operations on the input variables.

In order to deal with this problem *dimensionally aware GP* (DAGP) was proposed [149].

This type of GP introduces semantic information into all nodes, and defines how functions can be applied depending on the semantic information of the child nodes. For example, each node can have a unit associated to it, and the summation function node can be applied only if both children have the same semantic information, which in this case would mean that they have the same unit. By using that information DAGP can evolve rules which adhere to all the semantic constraints defined by the user. However, introducing semantic information alone is not enough to ensure that DAGP will evolve only semantically correct solutions.

To ensure that DAGP evolves only semantically correct solutions, additional modifications need to be performed on the algorithm. First of all it needs to be ensured that all individuals which are created are semantically correct, which is done by modifying the initialisation procedures of individuals. Therefore, during the creation of individuals, the semantic information is used to ensure that only nodes which lead to semantically correct individuals are selected. In this way it is ensured that all individuals in the initial population are semantically correct. However, applying the crossover and mutation operators will lead to the appearance of semantically incorrect solutions during the evolution process. Therefore, all genetic operators need to be extended in a way that they perform modifications which will not violate any of the given semantic rules. An obvious disadvantage of this procedure is that it is much more computationally expensive than standard GP, since it needs to ensure that all individuals it creates adhere to all semantic rules.

Figure 3.18 represents an example of a semantically correct and incorrect solution. Suppose that the *dd*, *PT*, and *SL* terminal nodes represent values measured in seconds, whereas the *w* terminal represents an dimensionless value. In addition, suppose that the addition and subtraction functions can be applied only on nodes which have the same unit, whereas the the division operator can be applied on nodes regardless of their units. Therefore, the left expression in the figure represents a semantically incorrect solution, since it subtracts the *w* and *pt* nodes, which do not represent values of the same unit. On the other hand, the second expression represents a semantically correct solution, since all functions are applied on nodes with compatible units.

3.3 Gene expression programming

Gene expression programming (GEP) is an evolutionary algorithm which evolves expression trees much like GP, but encodes them in a linear string similar to genetic algorithms [150]. In this way GEP tries to combine the simplicity of the representation and operators from genetic algorithms, with the possibility to evolve expressions from GP.

The individuals in GEP are of constant size, however, the part of the individual which is used to form the expression tree depends on the individual. Each GEP individual consists of one or more *genes*, where each gene consists of a number of primitive nodes and represents an

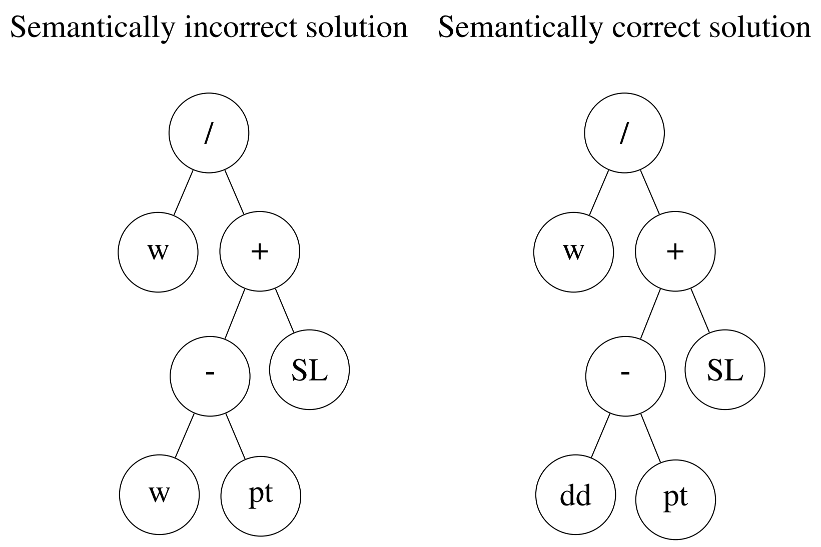

Semantically incorrect solution
Semantically correct solution
/
w
+
-
SL
w
pt
/
w
+
-
SL
dd
pt

Figure 3.18: Example of a semantically incorrect and correct solution

independent expression tree. In order to ensure that each gene always represents a syntactically correct expression, some constraints are placed upon the contents of each gene. First of all, each gene can be divided into two parts: the head of the gene and the tail of the gene. The head of the gene represents the starting h nodes of the gene, where h is usually a user specified parameter. This part of the gene can consist of any function and terminal nodes from the primitive set. The rest of the nodes belong to the tail of the gene, whose size depends on the size of the head, and is calculated as $t = h * (n_{max} - 1) + 1$, where t is the size of the tail, and n_{max} the maximum number of arguments from all nodes in the function set. The tail of the gene, however, can consist only of terminal nodes. The reason for this is to ensure that the gene will consist of enough terminal nodes for it to be decoded into a legal syntactically correct expression. Each gene can be decoded into an expression tree so that the first node in the gene represents the root node of the expression tree. After that, nodes are taken from the gene and placed into the expression tree in a depth first fashion, until all leaves of the tree consist of terminal nodes. The part of the gene which is used to form the expression tree is called the *coding region*. The final thing which needs to be defined is how the genes of an individual can be combined into a single expression. This is done by using *linking nodes*, additional function nodes which are defined for combining genes into a single expression. These nodes can either be manually defined, or also encoded into the individual. Figure 3.19 represents an example of a GEP individual. The individual consists of three genes with the head size of three nodes. The "|" marker is used to separate the nodes from different genes, while the underlined nodes belong to the coding region of the gene. Figure 3.20 represents the expression to which the GEP individual from Figure 3.19 is decoded. The nodes from the first, second and third gene are denoted in blue, green, and purple, respectively. The nodes denoted in red denote the linking nodes which are defined for this individual.

$$\underbrace{\underline{+\ -\ \mathrm{pt}\ \mathrm{w}\ \mathrm{MR}}\ \mathrm{age}\ \mathrm{w}}_{\text{Gene 1}}\ |\ \underbrace{\underline{*\ \mathrm{PAT}\ \mathrm{w}}\ \mathrm{w}\ \mathrm{dd}\ \mathrm{SL}\ \mathrm{pt}}_{\text{Gene 2}}\ |\ \underbrace{\underline{\mathrm{w}}\ +\ /\ \mathrm{pt}\ \mathrm{SL}\ \mathrm{dd}\ \mathrm{age}}_{\text{Gene 3}}$$

Figure 3.19: Example of a GEP individual

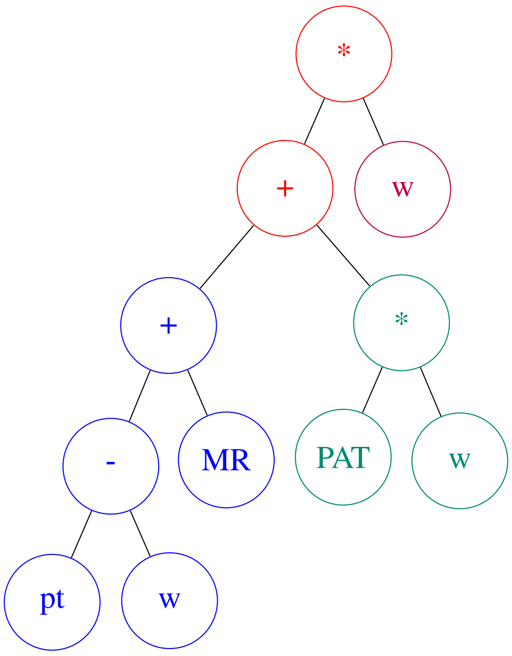


Figure 3.20: Expression tree representation of a GEP individual

Similar to other evolutionary algorithms, GEP also utilises different genetic operators to perform modifications on individuals. However, the operators which GEP uses are more similar to the ones which are used by genetic algorithms. For example, the one point crossover from genetic algorithms can easily be applied to GEP individuals. In this crossover a random crossover point is chosen, and the child individual is created so that all the nodes before the crossover point are taken over from one parent, while all the nodes after the crossover point are taken over from the other parent. Figure 3.21 gives an example of the one point crossover in GEP. As for the mutation operator, the point mutation is commonly used. In this mutation a random element in the individual is selected and replaced with another randomly generated element. If the selected element belongs to the head of the gene, then it can be replaced by any element from the primitive set. On the other hand, if the selected element belongs to the tail of the gene, then it can be replaced only by elements from the terminal set. The mutation is not restricted only to elements in the coding region, which means that the mutation can have a neutral effect if it is applied to elements outside that region. Figure 3.22 represents an example of point mutation in GEP, where the element which was selected for mutation is denoted in red.

In addition to the mutation and crossover operators, GEP also uses a special kind of operators called the *transposition operators*. These operators perform different modifications on the genes of an individual. Usually three different transposition operators are used in GEP: IS, RIS,

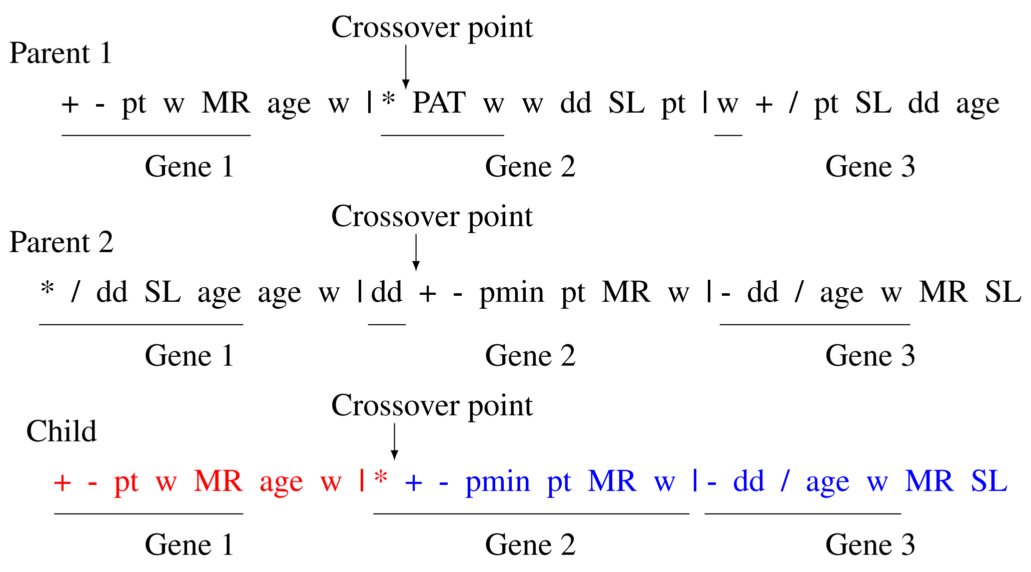

Parent 1
Crossover point
+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
Gene 1
Gene 2
Gene 3
Parent 2
Crossover point
* / dd SL age age w | dd + - pmin pt MR w | - dd / age w MR SL
Gene 1
Gene 2
Gene 3
Child
Crossover point
+ - pt w MR age w | * + - pmin pt MR w | - dd / age w MR SL
Gene 1
Gene 2
Gene 3

Figure 3.21: Example of the one point crossover in GEP

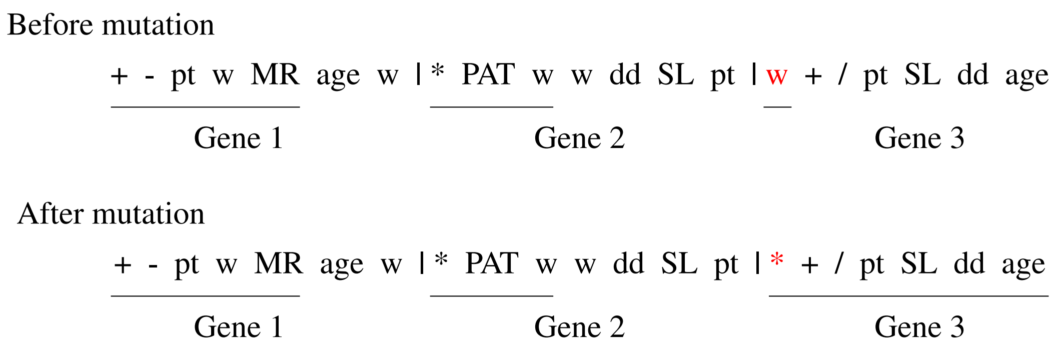

Before mutation
+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
Gene 1
Gene 2
Gene 3
After mutation
+ - pt w MR age w | * PAT w w dd SL pt | * + / pt SL dd age
Gene 1
Gene 2
Gene 3

Figure 3.22: Example of the replacement mutation in GEP

and gene transposition. The IS transposition operator randomly selects a fragment of a gene and places it in the head of a random gene. The only restriction in this operator is that the fragment cannot be placed at the very beginning of the gene. Figure 3.23 represents an example of the IS transposition operator, where the transposed sequence is denoted in red. The RIS transposition functions similarly to the IS transposition. A random sequence of elements is chosen, but unlike in the IS transposition, here it is mandatory that it starts with a function element. After that, the chosen segment must be transposed to very beginning of the gene. Figure 3.24 represents an example of the RIS transposition. Finally, the gene transposition operator randomly selects one gene and sets it as the first gene in the individual, while the other genes are shifted to the right. Figure 3.25 represents an example of gene transposition, where the gene which was selected for transposition is denoted in red.

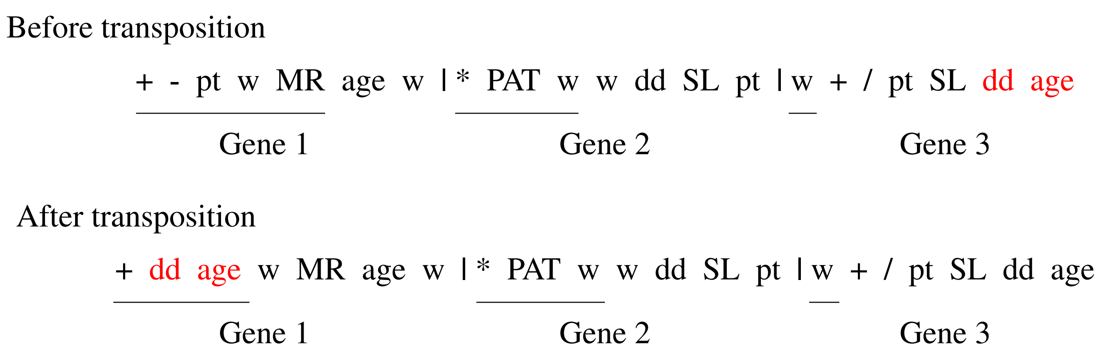

Before transposition
+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
Gene 1
Gene 2
Gene 3
After transposition
+ dd age w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
Gene 1
Gene 2
Gene 3

Figure 3.23: Example of the IS transposition

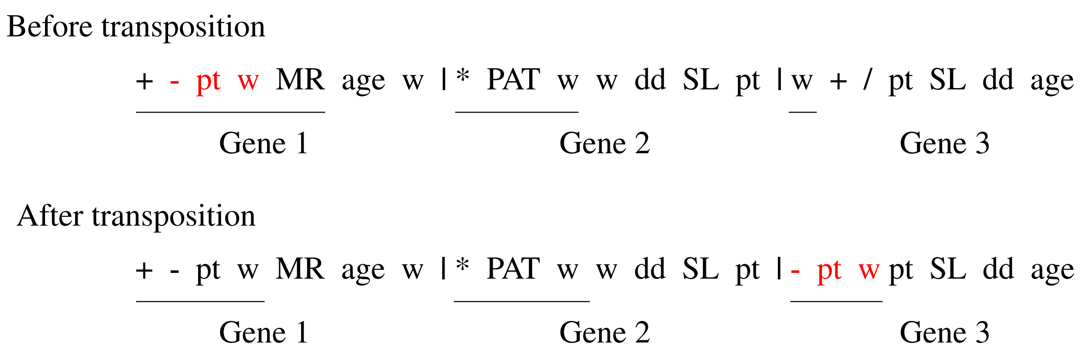

Before transposition
+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
Gene 1
Gene 2
Gene 3
After transposition
+ - pt w MR age w | * PAT w w dd SL pt | - pt w pt SL dd age
Gene 1
Gene 2
Gene 3

Figure 3.24: Example of the RIS transposition

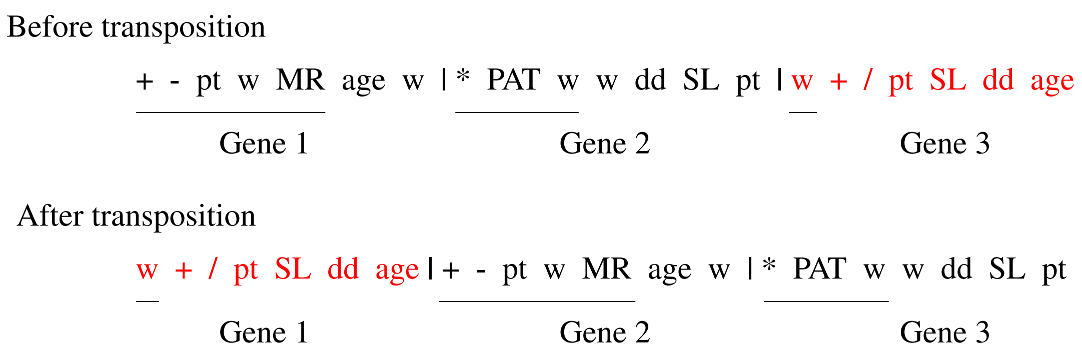

Before transposition
+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
Gene 1
Gene 2
Gene 3
After transposition
w + / pt SL dd age | + - pt w MR age w | * PAT w w dd SL pt
Gene 1
Gene 2
Gene 3

Figure 3.25: Example of gene transposition in GEP

Chapter 4

Design of dispatching rules by genetic programming for the unrelated machines environment

Manual creation of new DRs is usually a lengthy trial and error process, therefore there is a growing need that such a process is automated by using various procedures. For that purpose, many different machine learning and optimisation methods are used in order to automatically generate new DRs for a variety of scheduling problems. However, the most commonly used method for the creation of new DRs is GP. This is due to the fact that not only can GP generate good and interpretable DRs, but it can also be modified in different ways to improve its performance or to create DRs for various scheduling conditions.

This chapter will describe how the GP approach can be used for automatic design of new DRs for the unrelated machines environment. The approach described in this chapter will be used as a basis for all extensions in the rest of the thesis. In addition to that, this chapter will also give an overview of the literature dealing with the automatic design of DRs for various scheduling problems. Finally, this chapter will also present the current results achieved by automatically designed DRs for the unrelated machines environment.

4.1 Designing dispatching rules with genetic programming

DRs which are evolved by GP usually consist of two independent parts: the schedule generation scheme and the priority function. The priority function determines a priority value for a certain job-machine pair, which is calculated based on certain properties of jobs and machines. These priority values are then used by the schedule generation scheme to determine which job should be scheduled on which machine and in what order. Algorithm 4.1 represents the schedule generation scheme which is used to generate schedules for the unrelated machines environment.

In the first step, the procedure waits until at least one job and one machine are available. After that, the priority values are calculated for scheduling each available job (those which are released, but not yet scheduled) on each of the machines (even those which are still executing another job). By using the calculated priority values, the best machine (the one for which the concrete job achieves the best priority value) is determined for each job. Out of all jobs whose best machine is available, the one with the best priority value is chosen and scheduled on the appropriate machine. This part is repeated until there are no more jobs whose best machine is available. If however, there is no job whose best machine is available, then the scheduling decision is postponed to a later moment in time, when another job or machine becomes available. The entire procedure is repeated until there are no more jobs to be scheduled, or until the system stops with its execution.

Algorithm 4.1 Schedule generation scheme used by DRs generated by GP

```
1:  while unscheduled jobs are available do
2:      Wait until at least one job and one machine are available
3:      for all available jobs and all machines do
4:          Obtain the priority $\pi_{ij}$ of scheduling job $j$ on machine $i$
5:      end for
6:      for each job $j$ from the set of available jobs do
7:          Determine the best machine (the one for which the best value of priority $\pi_{ij}$
8:          is achieved)
9:      end for
10:     while jobs whose best machine is available exist do
11:         Determine the best priority of all such jobs
12:         Schedule the job with the best priority on the corresponding machine
13:     end while
14: end while
```

Unlike the scheduling generation scheme, which is defined manually, the priority functions are generated automatically by using GP. The objective of GP is to evolve a priority function which is appropriate for optimising a certain scheduling criterion, and which can be used by the aforementioned schedule generation scheme. For that purpose the primitive set of nodes, which will be used by GP, needs to be defined. The terminal set which is used by GP is given in Table 4.1. The *time* variable, which is used in some terminal node definitions, represents the current time of the system when the value of the nodes is calculated. Since all the nodes do not provide useful information when generating DRs for optimising each of the tested criteria, the set of nodes which will be used by GP depends on the criterion which is optimised. Therefore the dd, SL and w_{T_j} are used only for the due date related criteria (total weighted tardiness, number of tardy jobs, maximum tardiness, and weighted earliness and tardiness), the w_{C_j} only with the total weighted completion time criterion, and w_{E_j} only with the weighted earliness and weighted tardiness criterion.

Table 4.1: Terminal nodes used by GP

Node name	Description
pt	processing time of job j on the machine i (p_{ij})
pmin	the minimal job processing time on all machines: $\min_i(p_{ij})$
pavg	the average processing time of a job on all machines
PAT	patience - the amount of time until the machine with the minimal processing time for the current job will be available
MR	machine ready - the amount of time until the current machine becomes available
age	the time that the job spent in the system: $time - r_j$
	Used when optimising the due date related criteria
dd	due date of a job (d_j)
SL	positive slack of a job: $\max(d_j - p_{ij} - time, 0)$
w_t	tardiness weight of a job (w_{Tj})
	Used when optimising the total weighted completion criterion
w_c	completion time weight of a job (w_{Cj})
	Used when optimising the weighted earliness and weighted tardiness criterion
w_e	earliness weight of a job (w_{Ej})

Aside from the terminal nodes, the set of function nodes also needs to be defined for GP. Table 4.2 denotes the set of function nodes which will be used by GP. Although many other function nodes can be used by GP to evolve the priority function, it was shown that no significant improvements can be achieved by using them [151]. Therefore, the minimal function set which achieves the best results was chosen to evolve priority functions.

By using the aforementioned sets of terminal and function nodes, GP can evolve new priority functions which can be used by the schedule generation scheme. Figure 4.1 illustrates the tree representation of an example priority function evolved by GP for optimising the Twt, Nwt, and T_{max} criteria. This tree can be translated into the following expression:

$$pmin + PAT + \frac{pmin}{w_t} - (pt + MR + pavg * w_t) + \frac{SL}{MR * w_t^2} + dd + w_t + pmin - \frac{w_t}{age} - w_t.$$

After a short analysis of the presented priority function, several important building blocks of the function can be outlined. Before performing the analysis it should be stressed out that the

Table 4.2: Function nodes used by GP

Node name	Description
+	binary addition operator
-	binary subtraction operator
*	binary multiplication operator
/	secure binary division: $/(a,b) = \begin{cases} 1, \text{ if } \lvert b \rvert < 0.000001 \\ \frac{a}{b}, \text{ else} \end{cases}$
POS	unary operator: $POS(a) = \max(a,0)$

priority function is developed so that the most appropriate jobs for scheduling should obtain the smallest priority value. Since the priority function is optimised for due date criteria, one of the most important parts of the rule is $\frac{SL}{MR*w_t^2} + dd$. The dd part of the expression denotes that jobs which have a larger due date will also have a larger priority value. On the other hand, the $\frac{SL}{MR*w_t^2}$ expression will have a larger value for jobs which have larger slack values (amount of time until they would be late), and have a smaller weight. From the rest of the priority function it was shown that the expression $PAT + \frac{pmin}{w_t} - pt - MR$ is also important. The sub-expression $\frac{pmin}{w_t}$ will prioritise jobs which have a smaller minimum execution time and larger priority, while the PAT terminal gives more importance to jobs whose best machine will be free soon. The other part of the expression, $-pt - MR$ prefers jobs which have a larger processing time and machines which will be free in a much latter time. However, it seems that this part of the priority function is important since it will allow for the rule to schedule jobs on machines other than the one on which the jobs achieve the fastest execution time. The other expressions have shown to be redundant in the rule and seem to represent noise which was accumulated during the evolution process, since with their removals the fitness of the rule can even be slightly improved. Therefore, the considered priority function can be reduced to the following expression, without any deterioration in performance:

$$PAT + \frac{pmin}{w_t} - pt - MR + \frac{SL}{MR * w_t^2} + dd.$$

This shows that the evolved priority functions can consist of a significant number of unnecessary elements. However, many methods were proposed to generate simpler expressions or to remove the redundant parts [137, 152, 153, 154, 155].

The time complexity of generating schedules by automatically generated DRs can be divided into two parts: the complexity of evolving the priority function by GP, and the complexity of generating the schedule by the schedule generation scheme. The evolution of the priority

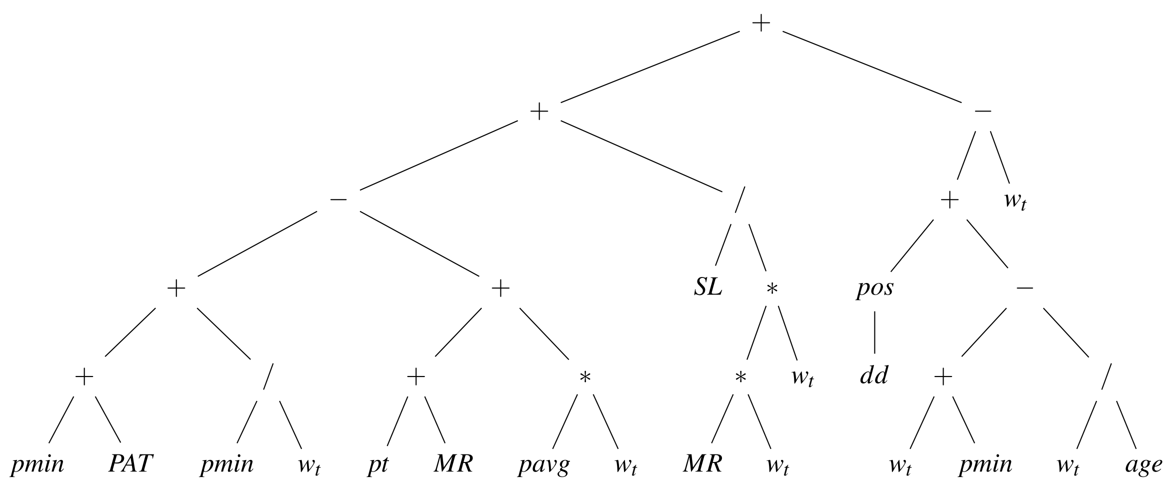


Figure 4.1: Example of a priority function evolved by GP

function by GP is a very time consuming procedure, with its complexity comparable to that of improvement heuristics. However, the evolution of the priority function needs to be done only once, and can be performed offline, before the start of the system execution. The evolved priority function is then used by the schedule generation scheme to incrementally build the schedule. The complexity of the schedule generation scheme is comparable to the complexity of manually designed DRs, meaning that it can be used to create schedules in dynamic online conditions. Therefore, a significant amount of time is invested only for generating the priority function, but each problem instance can be solved very fast.

The benefit of such an approach is that, unlike the improvement heuristics which generate a solution to only one concrete problem, GP creates a priority function which can be used for solving numerous problem instances. In addition, since the priority function and the schedule generation scheme are loosely coupled, the schedule generation scheme can be used with different priority functions for optimising various scheduling criteria.

4.2 Literature overview

This section will give an overview of methods for automatically generating DRs which are based on GP and similar evolutionary methods. In addition, the section will also include a very brief overview of other methods which were also applied for the generation of new DRs, like neural networks or other machine learning methods.

Dimopoulos and Zalzala [156] were the first to apply GP for solving a scheduling problem. They focused on solving the classic single machine scheduling problem. In their report they proposed two ways of applying GP to solve the considered scheduling problem. The first way was to use GP to determine a sequence in which certain manually designed DRs should be applied on the problem [157]. Therefore, GP is not used to create a new DR, but rather is used more like a standard GA which determines the sequence in which already existing DRs should be applied to obtain optimal solutions. However, the second way of applying GP for solving scheduling problems was to actually define a terminal set which consisted of job and system properties, and that should be used by GP to evolve new DRs [158]. Through several experiments GP demonstrated the ability to generate new DRs which could outperform several manually designed DRs. Cheng et al. [2] were the first to apply GP to generate a DR for scheduling air planes on different runways at the airport. Their approach evolved an algebraic metric, which uses several system properties to produce a numeric value based on which the next flight which should be scheduled is selected. In their study they additionally outlined several benefits of using GP to evolve new DRs over standard improvement heuristics.

Miyashita [159] was the first to apply GP to generate DRs for the job shop scheduling environment. In his work, Miyashita considered the scheduling environment as a multi agent system where each machine represents an individual agent. Based on that he proposed three different models: the homogeneous model, the distinct agent model, and the mixed agent model. The homogeneous model generates a single DR for all machines in the scheduling environment. On the other hand, the distinct agent model generates an independent DR for each machine in the scheduling environment. Finally, the mixed agent model combines the two aforementioned models in a way that two DRs are evolved, first of which will be used by *bottleneck machines* (machines which have a limited capacity and consequentially reduce the capacity of the entire system), while the second will be used by all other machines. Although the mixed agent model achieved the best results among the three multi-agent models, it comes with an obvious disadvantage, which is that prior to the execution of the system it must be known which machines represent bottleneck resources. In his study, Adams [160] also applied GP to solve scheduling problems, however the proposed representation is more complicated than the previously proposed representations, and no comparison with manually designed DRs was performed to validate the effectiveness of the approach.

Yin et al. [161] applied GP for solving scheduling problems subject to machine breakdowns. The authors propose the use of a bi-tree structured chromosome, where one expression is used to calculate the priority values of jobs, while the other is used to calculate the amount of idle time that should be inserted before processing the currently considered job. The objective of inserting this additional idle time is to act as a buffer against possible disruptions in the system. The proposed method has shown to outperform a manually designed DR. Ho and Tay [162] use GP to evolve DRs for the flexible job shop environment. In their study the authors propose the terminal set which should be used by GP, and apply it to generate DRs for minimising the the total tardiness criterion. The method was used to generate five DRs which were compared with standard DRs by using extensive simulations. The obtained results demonstrate that the generated DRs were easily able to outperform manually designed DRs from the literature.

Geiger et al. [163] considered evolving DRs for the single machine environment. In their study they evolved new DRs for optimising three criteria (makespan, maximum lateness and total tardiness), both in static and dynamic environments, and compared the obtained results with several selected standard DRs. The generated DRs achieved similar or even better performance than the selected standard DRs. In [164], Geiger et al. extend their approach to apply it for the single batch processor scheduling environment. The proposed approach is compared with several manually designed DRs for this problem, and demonstrates that it can generate DRs which perform slightly better for the test scenarios. Jakobović and Budin [165] use GP to create new DRs for the dynamic single machine scheduling problem, for optimising the *Twt* and *Nwt* criteria. Through experiments it was shown that GP was able to achieve better performance than several standard DRs. In addition, GP was also applied for job shop scheduling on four different scheduling criteria, where it was shown that it could mostly outperform standard DRs. The authors also propose a GP method, called GP-3, which extends the mixed agent model of Miyashita. Their approach generates three expressions, first of which represents the DR used for bottleneck machines, the second represents the DR used non bottleneck machines, while the third expression represents a decision function used to determine whether a machine is a bottleneck resource or not. Therefore this approach does not need any prior knowledge about the scheduling environment, since it uses the decision function to adapt to the changing conditions during the execution of the system. Although the GP-3 method does not guarantee that the expression it evolves will truly be able to identify which machines are bottlenecks, it was nevertheless able to achieve better results when compared to the approach which evolved only one DR for all machines.

Jakobović et al. [166] were the first to apply genetic programming for creating new DRs for the parallel uniform scheduling machine environment. The authors proposed both, the terminal nodes which should be applied for the considered scheduling environment, as well as the schedule generation scheme for creating the schedule by using the evolved priority functions. In the study both, static and dynamic scheduling conditions were considered, and the method was tested for optimising several scheduling criteria. The automatically generated DRs have shown to consistently outperform several manually designed DRs. Tay and Ho [167] applied GP to design DRs for minimising the total tardiness criterion in the flexible job shop environment. The method was applied to generate DRs for several test samples with different problem characteristics. The obtained results demonstrated that the evolved DRs performed significantly better than several standard DRs. Tay and Ho [26] were also the first to apply GP to create DRs for the multi-objective flexible job shop problem. In their study they focused on simultaneously optimising the makespan, total flowtime, and total tardiness criteria. In order to simultaneously optimise the three aforementioned criteria, they defined the fitness function as a linear combination of all three criteria. The results have shown that no single DR is able to perform well on all the tested criteria, and that GP was able to evolve DRs which can outperform the simple manually designed DRs. Beham et al. [168] use a simulation model and multi-population based GP to evolve new DRs. Although the approach has shown to obtain human competitive results, it has also shown to be computationally very expensive and to produce quite complex expressions. Yang et al. [169] propose a multi-objective GP rule generator for creating new DRs for a lot scheduling problem. The automatically developed DRs are able to outperform five simple DRs from the literature for two scheduling objectives.

Baykasoglu and Gocken [170] propose the use of GEP to generate due date assignment rules (DDARs). Due date assignment rules are used to approximate the due dates of jobs that enter the system, which should ideally be equal to the completion time of the considered jobs. The authors develop several simulation models upon which they test the rules generated by GEP. The obtained results show that in most of the test scenarios the rules generated by GEP dominate due date assignment rules from the literature. In [171], Baykasoglu et al. propose a GP based data mining approach for the selection of DRs which perform well under certain system parameters. GP is trained on a set of problem instances to learn which of the simple DRs is best suited for the current conditions. The obtained results show that the approach is able to select DRs which are appropriate for the current system conditions. Furoholmen et al. [172] consider the application of GEP to the distributors pallet packing problem. Although this in itself is not a scheduling problem, the main problem consists of two sub-problems, where one problem is concerned with pre-scheduling all items on the production line, while the second is concerned with packing the item onto the pallet. In order to solve this problem the authors used the cooperative coevolution method together with GEP, which independently evolved a heuristic for each of the two sub-problems. The best heuristic obtained by the proposed method achieved significant improvements over a standard manually designed heuristic.

Kofler et al. [7] propose an abstract model for the estimation of sequence dependant setup costs, and use GP to generate new DRs for the production planning problem. The DRs generated by GP were compared to the results obtained by several simple DRs, but also by results obtained by a GA. The generated DRs outperformed all of the simple DRs, however, they were unable to outperform the results achieved by the GA. Nevertheless, the execution times of the generated DRs were severely smaller than the execution time of the GA, which represents an advantage of the generated DRs over the GA. In [173], Kofler et al. compared three optimisation approaches for a real world scheduling problem. In the first approach a GA was used to determine the best best assignments of simple DRs on groups of machines. In the second approach the GA was used to determine the best assignment of simple DRs to each individual machine, while in the third approach GP was used to evolve a new DR used by all machines. Although the best results where obtained when assigning DRs to each individual machine, the generated assignments were shown to have a limited reusability and stability. Pickardt et al.

[174] apply GP for automatic creation of DRs to minimise weighted tardiness in semiconductor manufacturing. In order to make the results obtained by GP comparable to the results obtained by manually designed heuristics, the authors defined a terminal set which consists only of information that is also available to the manually designed DRs. Through long running simulations it was demonstrated that the DRs obtained by GP were able to significantly outperform any of the tested manually designed DRs. Thus, this study shows the potential of applying GP to complex manufacturing problems. Vazquez-Rodriguez and Ochoa [175] use GP to obtain variants of the NEH heuristic which is used for the flow shop problem. In this study GP is used to discover new ranking functions which are used to rank the jobs before they are scheduled. The proposed method has been tested on several simulation scenarios for optimising five scheduling criteria. The experiments demonstrate that the proposed approach is capable of achieving good results, which can outperform the standard NEH heuristic in several cases.

Hildebrandt et al. [42] propose that GP is coupled with stochastic manufacturing simulations to evaluate the individuals. The approach was tested on the dynamic job-shop problems for minimising the mean flowtime criterion. The results indicate that not only was GP able to evolve DRs which outperform the best manually designed rules, but also that GP was able to find rules which are able to generalise well on changing conditions of the system. The authors also analysed the influence of using look-ahead with DRs, which has not shown to be beneficial since it leads to results with a high variance. Nie et al. [176] were the first to apply GEP to create DRs for the dynamic single machine scheduling problem. GEP was applied to generate new DRs for optimising several different scheduling criteria, and the obtained results were compared to those obtained by GP and several standard DRs. Both GP and GEP have shown to consistently perform better than any of the standard DRs. Between those two methods, GEP has in several occasions shown to achieve slightly better results than GP. Nie et al. [177] also apply GEP to create DRs for the job shop scheduling problem. The results have shown that DRs generated by GEP were able to outperform several manually designed DRs. In [178], Nie et al. also apply GEP to generate DRs for the dynamic flexible job shop environment. GEP was applied for optimising the makespan, mean flowtime, and mean tardiness criteria. The results demonstrated that DRs generated by GEP achieved a better overall performance when compared with several manually designed DRs. Wang et al. [179] proposed a method which incorporates the the particle swarm optimisation (PSO) algorithm into GEP. The experimental results demonstrate that the proposed method achieved better performance when compared to previous approaches. Pitzer et al. [180] use a simulation environment to evolve large archives of DRs which provide new insights of DR optimisation. By using this archive of DRs, the authors have analysed the trade off between the size of the generated DRs and their performance. The results show that the performance of DRs improves with the increase of their size. However, after a certain size the DRs start to exhibit signs of overfitting.

In [27], Nie et al. extend their GEP approach for the single machine environment to adapt if for solving a multi-objective scheduling problem consisting of four criteria. To adapt the proposed approach for multi-objective scheduling, it is extended by using a fitness assignment scheme which combines Pareto dominance and density information to guide the search, a diversity maintaining strategy to adjust the non-dominated set of each generation, and an elitist strategy to guarantee the convergence of the search. The proposed method generated DRs which optimise several criteria simultaneously. Nie et al. [181] also use GEP to develop DRs for the dynamic flexible job shop problem. In this study the authors propose a new encoding and decoding scheme which introduces automatically defined functions (ADFs) into the GEP representation. The GEP representation is extended with an additional region which defines the interaction between ADFs, while the individual genes represent the individual building blocks of the ADF. In addition to that, each individual in GEP consists of two chromosomes, one of which represents the expression used to determine on which machine the current job should be scheduled, while the second expression is used to determine in which sequence the jobs will be executed on the given machine. In [182], Nie et al. consider the dynamic job shop scheduling problem with release dates. In the study the authors propose a heuristic for decomposing the original problem into a number of sub-problems, where each sub-problem represents a single machine scheduling problem with release dates. GEP is used to design DRs for solving such a problem, and through experimental results it was shown that it outperformed several manually designed DRs, and performed equally well as DRs designed by GP.

Unlike in several of the previous studies where GP was used to generate new DRs, Nguyen et al. [183] applied GP to generate an expression for selecting heuristics for solving a concrete scheduling problem. The experimental results prove that the proposed heuristic selection method can obtain good performance on a wide variety of problems. In [184, 185], Nguyen et al. propose two GP methods to evolve due date assignment models. The first proposed model uses the aggregate information from jobs, machines and the shop to predict the due dates of jobs, while the second model indirectly estimates the due dates by predicting the flowtime of each operation. The experimental results show that the automatically generated due date assignment models perform better estimates than several manually designed models. Out of the two proposed methods, the one which estimates the due dates based on the flowtimes of individual operations achieved better results. Nguyen et al. [186] propose an automatic programming method which uses iterated local search to generate new DRs for the job shop scheduling problem. The idea of this approach is to create an initial DR in a similar way which is used by GP to create individuals in the starting population. The solution is then iteratively improved by using different local search operators until a maximum number of iterations is reached. The proposed method is compared with GP and GEP, with the experimental results showing that the proposed method is not only able to obtain results which are better, but are also shorter than those obtained by the other methods. Therefore, the proposed method has shown to be a viable alternative to GP and GEP for creating new DRs.

Jakobović and Marasović [187] used GP to evolve DRs for the dynamic single machine scheduling problem with precedence constraints and sequence dependant setup times. Even by including these additional constraints GP again outperformed manually designed DRs. In the study, GP was also used to evolve DRs for the job shop scheduling environment, where DRs evolved by GP achieved better performance than several standard DRs. In addition, the study also includes an analysis of how different GP parameter values influence the performance of the evolved DRs. The paper outlines the importance of choosing the right value for the termination criterion to avoid overfitting on the training set. Abednego and Hendratmo [188] apply GP to create DRs for several different problem instances, and show that the evolved DRs achieve superior performance over several manually designed DRs. Nguyen et al. [189, 190] consider the problem of automatic design of scheduling policies which consist of DRs and DDARs. The authors propose a cooperative coevolution method for simultaneous development of both the DR and the DDAR for the dynamic job shop scheduling problem. The effectiveness of the proposed approach was tested on several problem sets and compared not only to several manually designed scheduling policies, but also to policies designed by several multi-objective algorithms where a single individual consisted of two chromosomes, one which is used as the DR, while the other is used as the DDAR. The proposed cooperative coevolution approach has outperformed both of those two methods, thus demonstrating the ability to generate scheduling policies of good quality.

Nguyen et al. [43] propose the use of GP to evolve iterative DRs. The idea behind these DRs is that they create the schedule several times, each time extracting knowledge from previously created schedules to improve the performance of newly generated schedules. The process is repeated as long as the quality of the newly generated schedules continues to improve. Since the schedule needs to be created several times, IDRs can be applied only in static scheduling conditions. Through experiments it was shown that the evolved iterative DRs achieved better performance than DRs evolved by GP and standard manually designed DRs. The method was additionally combined with variable neighbourhood search [191, 192] and look-ahead to further improve its performance. Hunt et al. [33] identify that a major problem of using DR approaches is their lack of a global perspective of the current and potential future state of the system. In order to remedy this problem, the authors propose that several new nodes are used, which provide the DR with more information about the system state. Although the search space increases by including these additional terminals, GP was nevertheless able to evolve DRs which achieved a better performance for minimising the *Twt* criterion in the dynamic job shop environment. Hunt et al. [193] also investigate the potential of GP for evolving optimal DRs. They evolve DRs for the problem of optimising the makespan criterion in the static two machine job shop environment, for which an optimal algorithm exists. Through the performed experiments it was proven that GP is expressive enough to create a DR which was equivalent to the optimal algorithm. In addition, GP was also applied for optimising the *Twt* criterion in the dynamic two machine job shop environment. The purpose here was to analyse whether there is a difference between using a single DR for all machines, or using machine specific DRs. In addition, the performance of GP was also analysed when the same problem instances are used throughout the entire evolution process, or when the problem instances are changed every few generations. Unfortunately, a conclusive answer about which methods were better could not be given, since the results demonstrated that the performance of these methods depends heavily on the problem instances used for testing.

Nguyen et al. [34] analyse the influence of different solution representations in GP on the quality of the obtained DRs. In their study they propose three different representations: the decision-tree like representation, the arithmetic representation, and the mixed representation. The idea of the decision-tree like representation is to design DRs which have the ability to select a DR from a set of available rules, based on the properties of the system. The arithmetic representation is used to create expressions which are used for calculating the priorities of jobs, based on which it is determined which job should be scheduled. This representation was most often used in the rest of the literature. The final representation is similar to the first representation, however, instead of using only predefined DRs, GP now also has the ability to create new DRs that can be applied. Although all three representations achieved a quite good performance when compared to standard DRs, the mixed representation achieved the most promising results out of the three tested representations. Branke et al. [194] analyse different hyper-heuristic methods and their representations for evolving DRs. In their study the authors test three different solution representations of DRs: a linear combination of attributes, a neural network based representation, and a tree based representation evolved by GP. All three methods were tested on dynamic stochastic job shop scenarios to measure their performance. The results have shown that the tree representation was able to achieve solutions of best quality, followed closely by neural networks. The linear representation achieved the least competitive results, however, for a smaller number of evaluations this method managed to achieve the best results. Therefore, the choice of the representation can also depend on the available computational budget. Đurasević et al. [151] have analysed how the choice of the selected GP method influences the the quality of the obtained solutions. In their study they compared several methods: GP, GP without optimised parameters, GEP, DAGP, and iterative DRs. The reason for including GP without optimised parameters was to test how large of an influence parameter optimisation can have on the results, since this is usually a quite time consuming process. In the end, GP without parameter optimisation achieved the worst and most dispersed results among all the methods, thus outlining the importance of performing parameter optimisation. GP, DAGP, and GEP achieved mostly similar results, without major differences, thus the choice of the method depends solely on the user. The iterative DRs have, as expected, significantly outperformed any of the other methods, which is expected since they are applied under static conditions. All the methods were compared to several manually designed DRs, but also to the best results achieved by a GA. Although they were able to perform equally well as the manual DRs, even outperforming them in most cases, they were still unable to match the results obtained by a GA.

Park et al. [195] applied GP to create DRs for the order and acceptance (OAS) problem. In the OAS problem, in addition to scheduling the jobs, the system also needs to select which jobs should be accepted into the system to maximise the profit. The study proposes two methods of evolving DRs for this problem, the first in which GP evolves a single expression which is used both for scheduling and determining whether the job should be accepted, while in the second GP is used to evolve two independent expressions, one which is used for scheduling and other which is used for determining whether the job should be accepted. The experiments have shown that the proposed methods perform better than several manually designed heuristics for the OAS problem. In addition, it was shown that the separation of the acceptance and scheduling decisions into independent expressions did not lead to any improvement in the results. Nguyen et al. [196] proposed a two-stage learning/optimising system for solving multi-objective OAS problems. In the first stage of the proposed system, GP is used to generate effective DRs for the OAS problem. These rules are then used to create the initial population of an evolutionary multi-objective algorithm which performs further optimisation. The obtained results demonstrated that the proposed method is effective when compared to other methods from the literature, thus outlining the potential of using methods like GP to improve the performance of improvement heuristics. Park et al. [197] use GP to tackle the OAS problem, and incorporate stochastic behaviour into the DRs to allow them to effectively explore multiple potential solutions, which additionally improves the performance of the evolved DRs. This is implemented in a way that the priority value acts as a probability value which is used to select one of the several jobs which obtained the best priority values. Although the introduction of randomness into the algorithm might not seem beneficial, it was shown that such an approach can outperform the standard GP approach which was designed for creating DRs for the OAS problem. Park et al. [198] propose a combination of DRs generated for the OAS problem with a particle swarm optimisation method and tabu search to improve te performances of the approaches. In the tabu search the DR generated by GP is used to generate the initial solution, upon which the local search operators will be applied. On the other hand, for the particle swarm optimisation method the DR will be used to generate the initial best position to which all the particles could converge to. The experiments have demonstrated that the tabu search method has benefited form such an initialisation procedure, while the particle swarm optimisation method did not show improvements when using the generated DR for initialisation.

Nguyen et al. [199] propose a new method called sequential genetic programming, to create DRs for the OAS problem. The general idea of this approach is that the priority functions are trained directly from optimal scheduling decisions at different decision moments throughout the execution of the system. A branch and bound method is also developed, which is used to create optimal solutions that are used by GP for learning. The experiments have shown that the proposed method does not only achieve better performance than several manually designed OAS heuristics, but that it also outperforms rules generated by the standard GP method. In [200], Nguyen proposes a system for optimising the OAS problem which tries to combine the advantages of hyper-heuristic and metaheuristic methods. The objective of the hyper-heuristic method is to extract knowledge about the problem in the form of DRs. The metaheuristic approach then uses the solutions obtained by the generated DRs to improve them further and to obtain near optimal solutions. Using the solutions obtained by the generated DRs as initial solutions in the metaheuristic method should also lead to an improved execution time of the metaheuristic approach. The performed experiments demonstrate that by using the solutions obtained by the generated rules the performance of the metaheuristic can also be improved. In [201], the authors apply GP in combination with the branch and bound algorithm to find optimal solutions for the OAS problem. GP is used to design effective branching strategies for the branch and bound algorithm. The results demonstrate that the enhanced branch and bound algorithm can solve the OAS problem more effectively than the standard branch and bound algorithm.

Han et al. [202] apply GP to generate DRs for the hybrid job shop environments with a multi-level bill of materials (BOM), which represents a list of parts, components, or materials that are needed in the manufacturing process. The performance of the method was tested using a simulation environment, which demonstrated that GP can achieve better performance than several standard DRs. Hildebrandt et al. [203] apply GP to generate DRs which optimise the mean cycle time of lots in semiconductor manufacturing. In order to validate the effectiveness of the approach, this study used a simulation environment which consisted of over 200 machines and 3000 jobs per simulation run. The proposed method achieved better performance than several selected benchmark heuristics. However, the authors outlined that further investigations must be performed, since it has proven to be difficult to compare DRs generated by GP and manually designed DRs. Mascia et al. [204] define a grammar which can be used to generate iterated greedy algorithms which minimise the weighted tardiness criterion for the permutation flow shop problem. The authors propose a parametric representation of the grammar, which has shown to obtain better results than when using the representation of grammatical evolution.

Pickardt et al. [205] propose a two-stage approach which combines the benefits of GP and an evolutionary algorithm. In the proposed approach, GP is first used to generate new DRs, which are then, together with several manually designed DRs, used by the evolutionary algorithm to determine their assignment to different work centres. The proposed approach was tested on a scenario from semiconductor manufacturing, in which the goal was to optimise the mean weighted tardiness criterion. The proposed approach was compared with both, several manually designed DRs, DRs evolved by GP, and the evolutionary algorithm on its own. It must be stressed out that the evolutionary algorithm was used only to determine the placement of manually designed DRs on work centres, and not to generate the entire schedule on its own. The proposed method has shown to obtain solutions which are superior to any of the methods it was compared to. The reason for achieving such a good performance lies in the fact that GP and the evolutionary algorithm optimise two different heuristic search spaces, thus allowing the procedure to obtain better solutions. Qin et al. [206] apply GP to create DRs for the interbay automated material handling system. In this study the authors use GP to generate DRs which should simultaneously optimise three criteria, however they did not apply any multi-objective GP method, but rather linearly combined all three criteria into a single value, which was then used as the fitness measure during the evolution process. Through several simulation scenarios it was shown that the proposed method performed better than other DRs for all of the tested scenarios.

Nguyen et al. [28] used GP for evolving multi-objective DRs which can simultaneously optimise five scheduling criteria in the dynamic job shop environment. Unlike Tay and Ho [26], the authors in this study used a multi-objective GP algorithm to evolve a Pareto front of solutions, thus giving the possibility of choosing the DR which provides the needed trade off between the various scheduling criteria. The proposed multi-objective algorithm which was used in the study is based on the HaD-MOEA algorithm. The experiments have demonstrated that the proposed approach achieved better performance than several standard DRs, both for a single objective and multiple objectives. In addition, the study also provided an analysis of the evolved Pareto fronts and an analysis of the correlation between several scheduling criteria. In [29], Nguyen et al. extended their multi-objective algorithm with additional local search operators, which leads to improved results. In addition, in the study the authors also analyse the effect of using the same problem instances throughout the entire evolution process, and changing the instances used in each generation. For this test the experiments have shown that changing the problem instances in each generation does not lead to improved results. Masood et al. [31] consider using GP for designing DRs for the many objective job shop scheduling problem. In their study they used a many objective GP algorithm based on the SPEA2, NSGA-II, and NSGA-III algorithms. They used the aforementioned methods to solve two many objective problems, one consisting of four scheduling criteria, and the other consisting of five scheduling criteria. Through the experiments it was demonstrated that the GP method based on the NSGA-III algorithm achieved superior results when compared to the other two methods. Masood et al. [207] also propose a particle swarm optimisation GP method for the adaptation of the reference points used by NSGA-III when optimising many objective job shop problems. The motivation for this approach originates from the fact that NSGA-III was initially designed to work with uniformly distributed reference points, which do not match well with the non uniform Pareto fronts which are found in job shop scheduling problems. For that reason the particle swarm optimisation algorithm is applied to update the reference points every time a new population is formed. The performed experiments have demonstrated that the proposed extension of the approach can lead to better performance of the NSGA-III algorithm.

Sim and Hart [208] propose a new heuristic generation framework, called NELLI, which can be used to evolve new DRs. The method uses previously designed DRs as terminals, and tries to combine them into a tree like representation of the rule, by using mathematical operators. The rules are initialised in a ramped half-and-half fashion, similarly as in GP. In each iteration a new heuristic is created and added into the system, either by random initialisation, or by mutating an already existing DR. The experiments show that combining DRs by the proposed procedure can lead to the development of better DRs than those obtained previously in the literature. Riley et al. [209] analyse the importance of selecting the appropriate terminal set for the GP algorithm. First, through simple frequency analysis they try to determine the importance of different nodes available in the terminal set. Then, the authors also propose a terminal weighting scheme which can be used during the evolution process to determine the weights of different nodes in the terminal set. This information is also used by a proposed adaptive mutation operator, which uses the terminal weights as probabilities when creating new individuals. Although the proposed method has shown potential, it comes with several drawbacks. The most important one is that the behaviour of the procedure is unstable, which is caused by a large number of used attributes, but also by the fact that the frequencies are used for calculating the terminal weights, which can be misleading since it is possible that a certain terminal appears often in the DR, but does not deliver any useful information. Shi et al. [210] propose the use of scatter programming [211] to design new DRs for the hybrid flow shop problem. The authors also propose an additional local search operator to enhance the performance of scatter programming. The results show that DRs which were generated by their proposed approach achieve better results when compared to DRs generated by standard GP and scatter programming without the additional local search operator.

Park et al. [35] analyse the possibility of evolving ensembles of DRs for the static job shop environment. In order to evolve ensembles of DRs the authors have used the cooperative coevolution approach. The experimental results have shown that the ensembles of DRs achieve a better performance than the individual DRs which were evolved by GP. In addition, the influence of the ensemble size on the results was also analysed, and it was shown that the proposed method achieves similar performance for different ensemble sizes. Park et al. [212] propose an ensemble learning method which uses only one population of individuals. The individuals from the population are grouped randomly with other individuals from the population, to form the ensembles. The initial investigation on this topic did not show that the proposed approach leads to improved results, therefore further investigation is required. Hart and Sim [36] also propose a method for evolving ensembles of DRs for the static job shop scheduling problem, called NELLI-GP. The aim of this method is to create an ensemble of DRs where each of the DRs contained in the ensemble is generated to optimise only a certain subset of the training instances. Therefore, each DR in the ensemble will focus on a different subset of problem instances, making this method similar to boosting and bagging. The performed experiments have shown that the proposed method is able to achieve significantly better results than other GP methods from the literature, including those obtained by Park et al. [35].

Branke et al. [213] apply GP for the dual-constrained flow shop scheduling problem. Unlike in the previous problems where only the machine capacity was considered, in this type of problem operators are needed for loading and unloading the machines. In this case, the operators will use the developed DRs to determine which of the jobs should be loaded on, or unloaded from a machine. The experimental results show that GP generated DRs which performed considerably better than standard manually designed DRs. Chen et al. [214] apply GP to create DRs for a k-stage hybrid flow shop in which one stage is composed of non-identical batch processing machines, and the others of non-identical single processing machines. After the DRs are generated by GP, ant colony optimisation is used to select the best three DRs, one of which will be used for the assignment, the second for sequencing, and the third for batch formation. The selected three rules are then used to build the schedule. The proposed method has shown improved performance for the *Twt* criterion over other heuristic methods. Freitag and Hildebrandt [30] consider applying GP to create DRs for a complex manufacturing system. In the study a multi-objective GP algorithm is applied to automatically generate DRs which optimise both the earliness and tardiness criteria. The experiments were performed on a complex manufacturing system which consisted of more than 200 machines. However, even when applied for such a complex system, GP was able to generate DRs which outperform several manually designed DRs. Therefore, the study demonstrated the potential of applying GP even for large and complex manufacturing systems.

Hunt et al. [215] incorporate local search into the evaluation of DRs to encourage the DRs to make good local decisions, which should in the end lead to an improved overall performance. The obtained results show that the inclusion of local search leads to the development of DRs which make better decisions and consequentially attain a lower value for the *Twt* criterion. In [216], Hunt et al. study the interpretability of DRs. The authors define a grammar which is implemented though strongly typed GP [217], to restrict the GP to evolve DRs which are composed of meaningful expressions. However, the obtained results demonstrated that the DRs which were generated by using the defined grammar were unable to perform equally well as

DRs which were generated by standard GP, although they were easier to interpret. In addition, several new terminal and functional nodes were introduced in the study, which have lead to easier interpretation of the evolved DRs. Karunakaran et al. [32] study a multi-objective scheduling problem in which they want to simultaneously optimise three scheduling criteria. However, to solve this problem they propose a new GP method for generating DRs, which is based on an island model algorithm. Through the study the authors have tested three different island topologies, and have shown that the proposed method can obtain better Pareto fronts than other widely used multi-objective optimisation algorithms, such as SPEA2 and NSGA-II. An additional advantage of the proposed method is that its execution time can easily be improved, since it is suited for parallel execution. Karunakaran et al. [218] consider using GP for creating DRs for the dynamic job shop scheduling environment with uncertainty in the processing times. This means that the processing times of jobs are not entirely known in advance, but are rather subject to certain changes, since they can also depend on other factors in the system. To adapt GP to uncertainties in the system, the authors propose a new terminal node which tries to capture the information about the uncertainty in the system. The approach has been tested on several problem sets to optimise the Ft criterion. The experiments have shown that up to a certain level of uncertainty in the system, GP with the additional terminal node is able to generate new DRs which can cope with such a problem. However, after a certain level of uncertainty is exceeded, the additional terminal has not shown to provide any useful information.

Li et al. [219] study the problem of minimising the Twt criterion in intercell scheduling considering transportation capacity. They apply GP to develop new DRs based on attributes of parts, machines and vehicles. An additional GA is also applied to select appropriate rules for machines and vehicles out of the rule set which was generated by GP. The computational results show that the rules which are developed by GP exhibit a better performance when compared to several manually designed DRs. Mei and Zhang [220] investigate the influence of the size of the learning set instances on the reuseability of DRs on unseen problem instances. The motivation for this investigation is that it is desirable to use smaller problem instances to evolve DRs which could then be applied on larger unseen problem instances. The reason why a smaller learning set is preferred is to save time during the evolution process. Through an extensive experimental analysis it was shown that the best results were obtained in cases where the learning set contained problem instances with the same ratio between jobs and machines as was used in the test set. In addition, even better performance can be achieved if the training instances contain a number of jobs and machines which is similar to that used in the test set. Mei et al. [221] outline the importance of providing an appropriate terminal set for GP to perform its search. The terminal set can be designed to include a wide range of job, machine and system parameters, however not all of these terminal nodes will prove to be useful to GP when creating new DRs. Unfortunately, a large terminal set increases the search space, meaning that it will be much harder for GP to obtain good DRs. Because of that reason the authors propose a feature selection based GP to determine which features from the terminal set provide the most useful information to GP when creating new DRs. The experimental results demonstrate that using only the filtered set of terminals allows GP to create DRs which achieve significantly better performance on both the training and test set. Therefore, the proposed method is more than capable of determining which terminals present the most useful information for designing new DRs.

Nguyen et al. [222, 223] propose a new surrogate assisted GP to evolve more efficient DRs without significant computational costs. The motivation of using a surrogate model is to decrease the evaluation time of individuals, which usually represents the most time consuming part of the GP algorithm. The proposed surrogate approach creates simplified simulation models which it then uses to evaluate the effectiveness of DRs during the evolution process. Through experimental results it was shown that the proposed surrogate model is capable of outperforming other algorithms in terms of rule performance. In addition, in several cases the GP with the surrogate model was also able to improve the execution time of the algorithm, as well as to reduce the sizes of the evolved rules. Park et al. [37] analyse the application of fitness sharing when evolving ensembles of DRs. The intention of this extension is to improve the diversity of the members of the ensemble, which is important to increase the generalisation ability of the ensembles. The initial experiments demonstrated that with fitness sharing it was possible to reduce the size of the ensembles. In [38], Park et al. consider optimising the *Twt* criterion in the job shop scheduling environment. In this study the authors propose the use of a multilevel GP approach to evolve ensembles of DRs. In this approach groups denote sets of individuals which cooperate with each other. In addition to this approach, the approach of Park et al. [35] is extended with additional terminals and applied for the dynamic job shop problem. The results demonstrate that the multilevel GP approach evolved rules with insignificant improvements in the results, while the second approach was able to significantly improve the obtained results. Ingimundardottir and Runarsson applied an evolutionary algorithm, namely CMA-ES, to determine the weights of different features in linear composite dispatching rules [224].

Park et al. [225] study the application of GP for generating DRs to optimise the dynamic job shop scheduling problem which is subject to breakdowns. The authors first develop a data set which is used for evaluating the GP approach, after which the standard GP approach is applied to generate new DRs for the scheduling problem with machine breakdowns. The obtained results show that the approach is quite sensitive to the level of breakdowns which are used to generate the problem instances. The authors also performed an analysis of the distribution of terminal nodes contained in the evolved rules, and noticed that the use of training instances with high levels of machine breakdowns will lead to different distributions of terminal nodes in the evolved DRs. Mei et al. [226] propose the use of GP to create *time-invariant* DRs. A DR is time-invariant if it generates the same schedule for the same pattern of jobs and machines, regardless of the time when such a pattern occurs. A terminal set which ensures that GP evolves time-invariant DRs is proposed and used by GP to develop new DRs. The experiments demonstrated that the proposed time-invariant GP generated DRs which achieve significantly better performance than standard DRs. In addition, it was also shown that the evolved DRs are of smaller sizes than DRs evolved by standard GP. Even though in some occasions the standard GP managed to outperform the time-invariant GP, the DRs generated by the standard GP can be transformed into time-invariant DRs with similar performance.

In [227], Karunakaran et al. further investigate the use of GP for evolving DRs for dynamic job shop scheduling problem with uncertain processing times, which means that the processing time of an operation is not known exactly until it finishes with its execution. In this study the authors propose two ways of generating new DRs which can cope with the uncertainties in the scheduling problem. In the first approach DRs are evolved using the expected processing times, however in the testing phase the estimated processing times of jobs are used, which are calculated based on the realised uncertainty on the training set. On the other hand, in the second method the are directly trained by using the estimates of processing times. Both methods show to outperform the standard GP approach and the approach proposed for dealing with uncertainty in [218]. Out of the two proposed methods, the second one has achieved better generalisation characteristics. Karunakaran et al. [228] propose another method of dealing with uncertainties in job shop scheduling problems. The authors notice that with the introduction of uncertainties, more and more machines become bottleneck resources in the system. Therefore, a method for classification of the bottleneck level of machines in the shop is proposed. Furthermore, a new method for developing pairs of DRs for different bottlenecks, which is based on the cooperative coevolution method, is proposed. The obtained results show that the proposed method achieves better performance than the standard GP method, and also the GP-3 method proposed in [165].

Aside from the previous studies which mostly applied GP and similar evolutionary methods to generate new DRs, a great deal of research also focused on using other machine learning methods for the creation of DRs. Lee et al. [229] use the C4.5 algorithm to generate decision rules which, based on certain system properties, determine which of the available simple manually defined DRs should be applied for scheduling. To further improve the performance of the system, the authors propose the use of a GA to select the best rules for individual machines. The proposed approach leads to a significant improvement in the obtained results. Koonce and Tsai [230] used data mining to extract knowledge from solutions generated from a GA. This knowledge was used to define a set of DRs which could imitate the performance of the GA on the test instances, and perform well on other unseen problem instances. Li and Olafsson [231] propose a novel approach for generating DRs by using a data driven approach. In this approach the C4.5 algorithm is used to discover new DRs from production data. The proposed approach was applied on a single machine scheduling problem, where it was shown that the approach could closely mimic the behaviour of the heuristic on whose data it was trained on. In [232], Olafsson and Li propose a novel two-phase learning approach, where in the first phase it is determined which decisions correspond to the best scheduling practices. The obtained knowledge is then used in the second phase to learn a new DR. The C4.5 algorithm is used to generate new DRs which show to perform well on the test scenarios. Ingimundardottir and Runarsson [233] propose a framework for discovering new DRs by analysing characteristics of optimal solutions. A logistic regression classification method is applied to learn good scheduling choices in each dispatching step. The experimental results demonstrate that the proposed method outperforms standard DRs. The same authors also used evolutionary algorithms to determine the weights of different features in linear composite dispatching rules [224].

El-Bouri et al. [234] present an approach for single machine job sequencing based on neural networks. The network is trained on a set of problem instances to learn how to produce a job sequence which best satisfies the optimised objective. The authors outline that the benefit of using such an approach is for situations in which no suitable DRs exists. Weckman et al. [235] develop a neural network scheduler for the job shop environment. First, a GA is used to generate solutions to known benchmark problems, which are then used to train the network to capture the predictive knowledge regarding the position of an operation in a sequence. The neural network approach is compared with some common DRs, and it is shown to achieve better performance. Eguchi et al. [236] also train a neural network to act as a DR for scheduling jobs in the dynamic job shop environment. The proposed approach was tested on several problem instances, and it was shown that it outperforms the best DRs available from the literature. Petrović et al. [237] use a fuzzy rule-based system to determine lot sizes in the job shop environment, using several system parameters. Kapanoglu and Alikalfa [238] used GP to develop new if-then priority rules for the dynamic job shop environment, which considerably outperform several standard DRs. In [239, 240] a literature overview of several applications of different machine learning methods in scheduling problems is given.

Branke et al. [14] give a comprehensive literature survey of automated generation of DRs for various scheduling problems. They also outline several areas in which further research could be conducted. In [241], Nguyen et al. outline several key challenges in the application of evolutionary computation to combinatorial problems. They propose that the future studies should focus more on practical requirements and problem environments to which the obtained solutions could be applied. In [15], Nguyen et al. describe a unified framework for automated design of DRs in different scheduling environments. This survey provides a detailed overview of the different applications of GP for generating new DRs.

4.3 Parameters and experimental design

This section will describe the design of experiments, as well as the choice of parameters for GP. The parameter values and the experimental design described in this section will also be used in the latter chapters of the thesis, if not stated differently.

4.3.1 GP parameters

The effectiveness of GP depends heavily on the parameters of the algorithm. Therefore, it is crucial to optimise the parameter values of the algorithm, to achieve solutions of good quality. For that reason an extensive parameter optimisation procedure was performed, and the values of the parameters which produce the best results were selected. The best algorithm parameter values were determined in a way that the algorithm was executed for different values of a concrete parameter, while the other parameters were fixed to certain values. For each tested parameter value the algorithm was executed 50 times, and the parameter value for which the algorithm achieves the best average value on those 50 executions is selected. The criterion which was optimised when determining the optimal parameter values was the *Twt* criterion. Table 4.3 represents the values selected for the various GP parameters. It is interesting to illustrate how the performance of the algorithm depends on the number of iterations which are used as the termination criterion. Figure 4.2 depicts the influence of the iteration count on the performance of the obtained dispatching rules, both on the training and test set. The figure shows that the longer the algorithm executes, the better performance it achieves on the training set. However, on the test set it can be observed that the value for the *Twt* criterion improves until the algorithm reaches around 80000 iterations, after which the value of the criterion starts to deteriorate. This means that after the algorithm performs a certain number of iterations it starts to overfit on the training set, and produces DRs which do not generalise well on unseen problem instances.

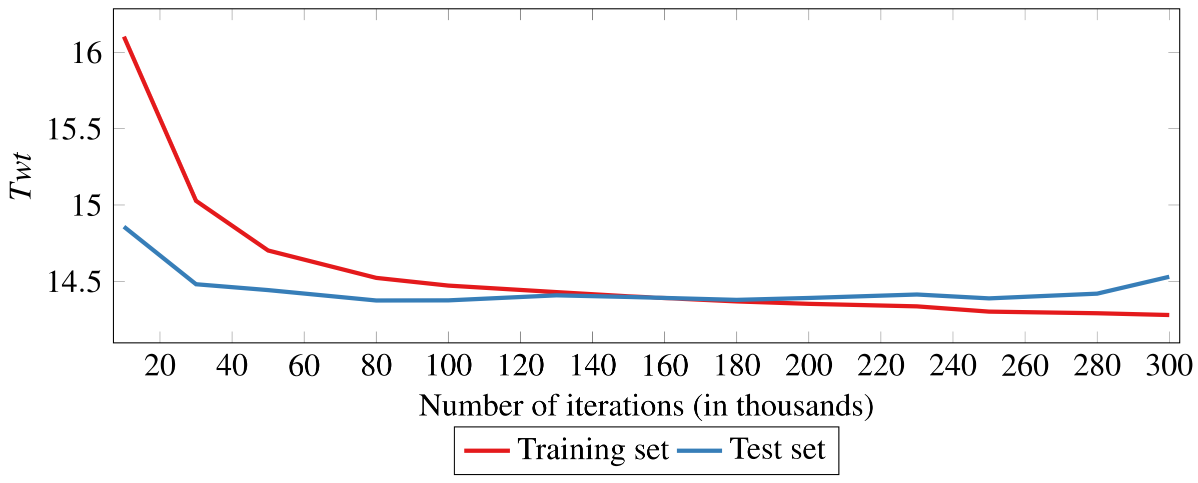


Figure 4.2: Influence of the number of iterations on the performance of GP

Table 4.3: Parameter values used by GP

Parameter name	Parameter value
Population size	1000 individuals
Termination criterion	80000 iterations
Selection	steady state tournament GP
Tournament size	three individuals
Initialisation	ramped half-and-half
Maximum tree depth	5
Crossover operators	subtree, uniform, context-preserving, size-fair
Mutation operators	subtree, Gauss, hoist, node complement, node replacement, permutation, shrink

DAGP uses the exact same parameters as the standard GP, except for the genetic operators. In DAGP only the one-point crossover and the node replacement mutation are used. In addition, to ensure semantic correctness of the solutions, it is required that all nodes carry the information about the physical unit (e.g. seconds, meters), as well as the exponent to which the unit is raised (e.g. s^2, s^{-1}). Since all terminal nodes in this application have the same unit - time (e.g. in seconds), it is sufficient to carry only the information about their exponent, since the unit will be the same for all nodes. The only exception is the weight node, which in itself has no unit. However, this can be treated as if the unit is raised to the zeroth exponent. In addition to the semantic information defined for terminal nodes, it is also mandatory to define the semantic rules for function nodes. Table 4.4 denotes the semantic rules defined for the function nodes when designing DRs.

GEP also uses the same parameters as the standard GP, except for those associated with the individual size and genetic operators. Through optimisation it was determined that GEP achieved the best results when using three genes with the head size of six nodes. As for the genetic operators, one crossover (one point crossover), one mutation (node replacement mutation) and three transposition operators (IS, RIS and gene transposition) are used.

4.3.2 Experimental design

GP will be used to evolve DRs for solving the unrelated machines scheduling problem, with the addition of job release times. The scheduling procedure will be performed online and in dynamic conditions. In order to evolve priority functions by GP it is required to define a certain set of problem instances, called the *training set*, which will be used during the evolution process

Table 4.4: Semantic rules defined for DAGP

Node	Constraint	New exponent value
+	the left and right child must have the same exponent	the same as the exponent of the child nodes
-	the left and right child must have the same exponent	the same as the exponent of the child nodes
*	none	the sum of the exponents of the left and right child
/	none	the difference between the exponents of the left and right child
POS	none	the same as the exponent of the child node

to determine the fitness of different solutions. However, the training set can not be used to evaluate the effectiveness of the evolved priority functions. This is due to the fact that since the priority functions were evolved using the training set, they could have adapted for solving only problem instances available in the training set, and would therefore have a poor generalisation ability. Therefore, to objectively measure the quality of the evolved priority functions, a new set of problem instances, called the *test set*, needs to be used to measure the quality of the evolved priority functions on unseen problem instances. Therefore, priority functions will be evolved using the training set, but their performance will be evaluated on the test set.

Since the priority functions, which are evolved by GP, should be appropriate for solving problems of different characteristics, both the training and test set should also contain problem instances of various characteristics. Therefore, both the training and test set contain 60 problem instances. Depending on the problem instance, the number of jobs can be 12, 25, 50, or 100, while the number of machines can be 3, 6, or 10. Both sets will have five problem instances for each combination of the number of jobs and machines, and the characteristics for each job are generated randomly. The processing times of jobs are generated from the interval

$$p_{ij} \in [0, 100],$$

by using one of the following three probabilistic distributions: uniform, normal (Gaussian), and quasi-bimodal. Which of the aforementioned three distributions will be used for generating the processing times is chosen randomly for each job (with all three distributions having the same probability of being chosen). The motivation behind the use of three distributions for generating processing times is to make the evolved priority functions more resilient, since in real conditions jobs could be received from different sources. All job weights are generated uniformly from the interval

$$w_{T_j}, w_{C_j}, w_{E_j} \in < 0, 1].$$

The release times of jobs are generated by a uniform distribution from the interval

$$r_j \in \left[0, \frac{\hat{p}}{2}\right],$$

where $\hat{p}$ is defined as

$$\hat{p} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}}{m^2},$$

and p_{ij} denotes the processing time of job j on machine i, while m denotes the total number of machines. The due dates of jobs are also defined using a uniform distribution from the interval

$$d_j \in \left[r_j + (\hat{p} - r_j) * \left(1 - T - \frac{R}{2}\right), r_j + (\hat{p} - r_j) * \left(1 - T + \frac{R}{2}\right)\right],$$

where T represents the due date tightness parameter, while R represents the due date range parameter. The due date range parameter defines the dispersion of the due date values, while the due date tightness adjusts the amount of jobs that will be late. While generating the problem set, both of those parameters assumed values of 0.2, 0.4, 0.6, 0.8, and 1 in various combinations.

The total fitness of an individual on one of the problem instance sets is calculated by creating the schedule using the priority function represented by the individual, and determining the value of the optimised objective. This is done for each problem instance contained in the set. The fitness function could then be defined as the sum of objective values for each of the problem instances in the set. However, since each problem instance has different characteristics (number of jobs, number of machines, due date range and tightness), this means that problem instances could have vastly different objective values. This could cause GP to evolve priority functions which focus more on problem instances that have a larger influence on the fitness value. Therefore, for each problem instance to have an equal influence on the fitness value, the objective values are additionally normalised for each problem instance. The normalisation procedures for all the scheduling criteria are defined as follows:

- Normalisation of the makespan criterion

$$f_i = \frac{C_{max}}{n\bar{p}}.$$

- Normalisation of the maximum flowtime criterion

$$f_i = \frac{F_{max}}{n\bar{p}}.$$

- Normalisation of the maximum tardiness criterion

$$f_i = \frac{T_{max}}{n\bar{p}}.$$

- Normalisation of the total weighted completion time criterion

$$f_i = \frac{Cw}{n\overline{w_{C_j}}\bar{p}}.$$

- Normalisation of the weighted tardiness criterion

$$f_i = \frac{Twt}{n\overline{w_{T_j}}\bar{p}}.$$

- Normalisation of the flowtime criterion

$$f_i = \frac{Ft}{n\bar{p}}.$$

- Normalisation of the weighted number of tardy jobs criterion

$$f_i = \frac{Nwt}{n\overline{w_{T_j}}}.$$

- Normalisation of the weighted earliness and weighted tardiness criterion

$$f_i = \sum_j \left(\frac{w_{Ej}E_j}{n\overline{w_{T_j}}\bar{p}} + \frac{w_{Tj}T_j}{n\overline{w_{E_j}}\bar{p}} \right).$$

- Normalisation of the machine utilisation (no additional normalisation is required)

$$f_i = M_{ut}.$$

In the previous expressions, $\bar{p}$ represents the average of processing times of all jobs, $\overline{w_{T_j}}$ represents the average of the tardiness weights of all jobs, $\overline{w_{E_j}}$ represents the average of the earliness weights of all jobs, $\overline{w_{C_j}}$ represents the average of the completion time weights of all jobs. The total fitness for the problem instance set is calculated as the sum of the normalised criteria values for each problem instance in the set

$$F = \sum f_i.$$

Since GP is a stochastic optimisation procedure, the behaviour of the algorithm will be different each time it is executed. This also means that it is highly possible that each time the algorithm is run, that a different solution will be obtained. Therefore, by executing the algorithm only once, it is not possible to measure the effectiveness of the algorithm. In order for the benchmark results to be statistically significant, each experiment was executed at least 30 times, while preserving the best solution from each run. The resulting best solutions from these runs were used to calculate quantitative information such as the median of the obtained fitness values, as well as the minimum and the maximum fitness value. In addition, the Mann-Whitney [242] statistical test will be used to determine if there is a statistically significant difference between the results obtained by two different experiments. The difference between two results will be considered significant if a p value smaller than 0.05 was achieved by the statistical test.

The results will also be presented by using a *Tukey box plot* [243, 244]. In these plots the solutions which represent outliers will be denoted as black points. In addition, the plots will also include the average values, which will be presented as a black rhombus.

4.4 Results

This section will give an overview of the results which can be achieved by the manually defined DRs and the DRs automatically designed by the GP methods described in Chapter 3. The results presented in this section will be used throughout the rest of the thesis to validate the performance of the proposed methods.

4.4.1 Performance of automatically generated DRs

The approach of automatically designing DRs by GP was applied for the nine scheduling criteria described in Section 2.1. To validate the performance of this approach, it is compared to several manually designed DRs described in Section 2.4.2. Table 4.5 represents the results achieved by the standard DRs and DRs designed by GP. The first three rows represent the results achieved by a genetic algorithm, which was executed 30 times for each problem instance. The row denoted as "GA - med" denotes the median values of the results achieved by all 30 runs, while the row "GA - min" denotes the very best result achieved by those 30 runs. In addition to those two values, the third value denoted as "GA - best" represents the results which are achieved in a way that for each problem instance the best value achieved by any of the 30 runs of the genetic algorithm is selected. This means that the value is not achieved by only one genetic algorithm run, but with all 30 runs together. The next 26 rows represent the results achieved by various standard manually designed DRs. The best result for each criteria is denoted in bold, while the best five values of all the DRs are denoted with grey cells. GP was executed 50 times for optimising each criterion, and the minimum and median values of those runs are denoted as the last two rows in the table.

The results denoted in the table demonstrate that no single standard DR is able to achieve good results for all the considered criteria. The ATC rule was executed with $k = 0.05$, the RC rule with $\alpha = 0.2$, the SA rule with $\nabla_l = 0.1$ and $\nabla_h = 0.8$, the OMCT rule with $\alpha = 0.9$, COVERT with $k = 0.2$, while the other rules do not use any parameters. The parameter values for each rule were fine tuned to obtain better results. The standard DRs usually achieve good results on only a few criteria for which they were designed, while for the rest of the criteria the results are usually not as good. For example, the WSPT rule achieved the best result for the Cw criterion out of all the tested DRs, however, it achieved quite bad results for rest of the criteria. Similarly, the ATC rule achieves good results for the due date related criteria, while for the other criteria it is not able to achieve similarly good results. On the other hand, the COVERT and RC rules achieved good results for several different criteria, although they usually do not achieve the best result for even one of those criteria. This just proves that if a DR is to optimise several criteria well, it will usually not achieve the best results for any of the criteria, but will compromise over all the criteria to achieve good results for most of them.

By comparing the results achieved by the automatically designed DRs with the standard DRs, it can be seen that GP is able to generate DRs which perform similarly or better than the standard DRs. The only two criteria for which DRs designed by GP were unable to outperform the standard DRs are the completion time related criteria C_{max} and Cw. For the other criteria the automatically designed DRs outperform the standard DRs. This is especially true for the M_{ut} and $Etwt$ criteria, for which only a few standard DRs achieve good results. These results demonstrate that GP truly has the ability to design DRs which are better than the standard DRs. However, it is possible to improve the performance of automatically generated DRs even further, to make them more competitive to standard DRs.

Apart from comparing the results of the automatically generated DRs with the standard DRs, it is also interesting to analyse how the DRs, which are designed to optimise one criterion, perform on the other criteria for which they were not designed. In Table 4.6 each row denotes the criterion for which the DRs were optimised, and in each column the generated DRs were evaluated for a different criterion. Several interesting phenomena can be observed from the table. The first is that DRs designed by GP usually perform well for the criterion for which they were optimised, while for the other criteria they perform quite bad. However, this is not really unexpected, since the applied GP algorithm only optimises one criterion at a time. Therefore it tries to achieve good performance on only one criterion, while ignoring the others. Consequentially, standard DRs will usually perform reasonably well over several criteria, while automatically designed DRs will perform well mostly on the criterion for which they were evolved. A more interesting phenomenon which occurs is that for some criteria better DRs are designed when optimising other objectives. This is evident for the T_{max} and Nwt criteria, since the DRs which were generated when optimising the Twt criterion achieve better results on those two criteria, rather than those which were generating DRs by directly optimising those two criteria. The reason for this phenomenon seems to lie in the fact that all three criteria can be optimised si-

Table 4.5: Comparison of automatically designed DRs with manually designed DRs

Method	Criterion								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
GA - best	36.79	844.4	80.55	12.74	140.8	0.006	5.340	1.897	9.533
GA - min	37.48	849.2	98.32	13.71	146.4	0.011	5.373	1.900	9.917
GA - med	37.72	851.3	103.0	13.97	149.9	0.012	5.455	1.941	10.27
MCT	38.57	902.3	977.2	**14.03**	181.1	0.130	8.007	2.891	18.88
MET	38.44	878.9	996.3	16.01	157.9	0.131	7.003	3.130	16.15
ERD	38.57	902.3	977.2	**14.03**	181.1	0.130	8.007	2.891	18.88
LPT	38.08	924.3	975.9	17.29	200.7	0.123	8.233	4.014	27.22
WSPT	38.68	**873.4**	991.7	17.14	169.2	0.129	7.305	3.468	19.45
SA	38.47	882.8	994.0	15.68	161.2	0.129	7.430	3.117	17.02
KBP	38.37	875.3	998.5	15.70	154.5	0.133	7.013	3.092	15.93
Maxstd	38.27	885.7	993.6	15.93	165.0	0.127	7.144	3.195	17.57
OMCT	38.31	886.8	994.2	15.71	165.6	0.127	7.147	3.208	18.11
OLB	46.84	981.6	940.2	25.81	258.1	0.104	10.75	7.345	38.85
WQ	73.44	1733	844.4	57.24	1018	**0.072**	31.61	26.00	364.1
JIT	60.36	1881	**378.9**	51.00	1156	0.101	29.33	17.09	189.3
EDD	38.72	910.2	960.8	16.72	189.4	0.129	6.976	2.521	14.50
MS	38.59	911.4	962.1	16.44	190.3	0.130	7.319	2.678	16.05
MON	38.39	888.4	989.6	16.19	168.3	0.128	6.711	2.668	14.97
CR	38.47	901.7	978.7	14.04	180.5	0.130	7.775	2.921	19.23
COVERT	38.26	903.0	967.1	14.57	182.3	0.126	6.755	2.442	13.50
ATC	38.26	901.5	968.5	16.31	180.8	0.125	**6.686**	**2.418**	**13.30**
Min-min	38.41	875.1	998.7	15.54	154.5	0.131	6.950	3.003	15.80
Max-min	38.62	908.8	974.7	14.08	187.9	0.127	8.008	3.072	20.94
Min-max	38.14	885.7	989.2	14.43	165.3	0.132	7.425	3.111	17.18
Sufferage	37.88	881.7	993.7	15.15	160.1	0.128	6.986	2.854	15.94
Sufferage2	**37.85**	917.6	975.2	16.57	196.4	0.123	8.096	3.566	24.07
RC	38.11	874.9	998.3	14.91	**154.1**	0.128	6.786	2.864	15.12
LJFR-SJFR	38.41	877.0	995.9	15.58	157.3	0.132	7.090	2.953	16.21
MECT	38.48	876.7	998.6	15.65	156.0	0.133	7.011	3.141	16.33
GP - min	38.02	873.8	236.9	13.60	154.0	0.046	6.384	2.376	12.96
GP - med	38.26	874.9	369.3	13.96	155.0	0.054	7.005	2.653	13.60

multaneously, but the *Twt* criterion seems to be easier for GP to optimise than the other two. Therefore, when optimising the *Twt* criterion, GP will also generate DRs which optimise the other two criteria quite well. As a consequence, it could be possible that even better results could be achieved if all three criteria were optimised simultaneously. The conclusion which can be drawn from all these findings is that instead of optimising only one criterion, several criteria should be optimised to design DRs which perform well on various criteria simultaneously. In addition, if several similar criteria are optimised simultaneously, it could be possible to generate DRs which achieve a better performance than that of DRs generated for optimising only one of those criteria.

4.4.2 Comparison of GP approaches

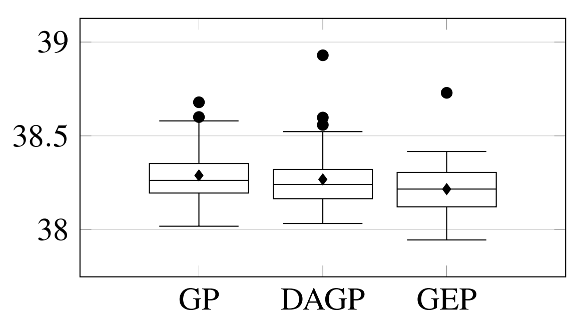
In this section the performance of the different GP approaches described in Chapter 3 will be outlined. Their performance was evaluated on four scheduling criteria: total weighted tardiness, number of tardy jobs, flowtime, and makespan. All the approaches were executed 50 times, and the minimum, median, and maximum values achieved from those 50 executions are denoted in Table 4.7. The best values for each criterion are denoted in bold. Figure 4.3 additionally gives a box plot representation of the results. The results demonstrate that all three approaches perform very similarly, with there being no significant difference between them, except for the Ft criterion where GEP achieved statistically better results than the other two approaches. Since neither of the approaches achieved generally better results, it was decided that the standard GP approach will mainly be used to develop new DRs for the rest of the thesis.

Table 4.6: Results for the automatically designed DRs across all the criteria

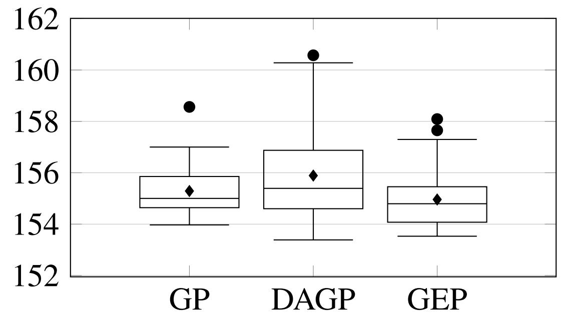
Optimised criterion		Evaluation criterion								
		C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
C_{max}	Min	**38.02**	878.4	971.1	14.03	158.4	0.123	7.228	2.951	16.57
	Med	**38.26**	894.9	987.3	16.06	172.8	0.131	7.703	3.345	18.99
	Max	**38.68**	924.4	995.9	19.77	203.1	0.142	8.239	4.370	26.82
Cw	Min	38.58	**873.8**	999.0	15.72	**153.7**	0.127	6.876	3.193	16.50
	Med	38.58	**874.9**	1000	16.13	**154.8**	0.137	7.170	3.339	17.02
	Max	39.03	**877.3**	1002	17.32	**156.8**	0.145	7.368	3.830	18.06
$Etwt$	Min	53.88	1626	**236.9**	40.88	904.1	0.088	15.53	4.866	38.03
	Med	56.92	1714	**274.4**	44.24	986.7	0.107	27.70	6.826	65.83
	Max	60.84	1848	**303.0**	50.81	1122	0.139	33.86	9.859	95.00
F_{max}	Min	38.06	893.5	975.6	**13.60**	173.6	0.122	7.219	2.728	17.31
	Med	38.37	898.2	981.4	**13.96**	177.6	0.133	7.612	2.996	18.61
	Max	38.83	904.5	984.5	**14.48**	183.9	0.139	8.099	3.212	20.25
Ft	Min	38.32	874.1	998.5	15.70	154.0	0.132	6.763	3.132	16.41
	Med	38.65	875.3	1000	16.19	155.0	0.139	7.140	3.385	17.03
	Max	39.16	879.3	1002	17.29	158.6	0.147	7.369	3.929	18.67
M_{ut}	Min	49.76	1443	733.1	37.85	724.5	**0.046**	26.86	15.33	191.0
	Med	77.45	2253	969.7	63.76	1521	**0.054**	42.24	35.27	687.1
	Max	102.8	2792	1438	90.07	2070	**0.058**	46.03	57.59	1196
Nwt	Min	38.54	879.7	812.6	16.49	158.7	0.145	6.384	2.562	13.57
	Med	40.01	906.3	977.2	19.91	184.2	0.145	7.005	3.392	17.39
	Max	43.40	1058	1003	28.74	337.0	0.157	7.939	7.395	50.45
T_{max}	Min	38.28	887.5	820.7	15.56	167.0	0.123	6.808	2.376	13.53
	Med	39.25	912.3	962.2	17.99	190.6	0.138	7.264	2.653	15.01
	Max	42.32	1059	984.9	27.22	333.7	0.153	7.989	4.051	26.59
Twt	Min	38.69	890.6	795.7	16.95	170.5	0.137	**6.198**	**2.361**	**12.96**
	Med	39.58	911.2	961.5	18.72	188.8	0.146	**6.687**	**2.558**	**13.60**
	Max	42.24	1078	981.8	27.09	353.2	0.152	**7.233**	**2.966**	**14.62**

Table 4.7: Comparison of the GP approaches

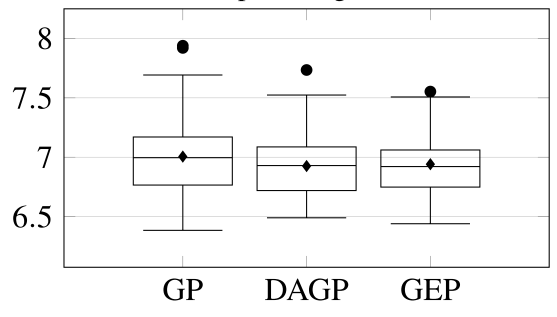
Criterion		Approach		
		GP	DAGP	GEP
C_{max}	Min	38.02	38.03	**37.95**
	Med	38.26	38.24	**38.22**
	Max	**38.68**	38.93	38.73
Ft	Min	154.0	**153.4**	153.5
	Med	155.0	155.4	**154.8**
	Max	158.6	160.6	**158.1**
Nwt	Min	**6.384**	6.490	6.440
	Med	7.005	6.934	**6.925**
	Max	7.939	7.734	**7.553**
Twt	Min	**12.96**	13.13	13.06
	Med	**13.60**	13.64	13.68
	Max	**14.62**	14.81	15.14



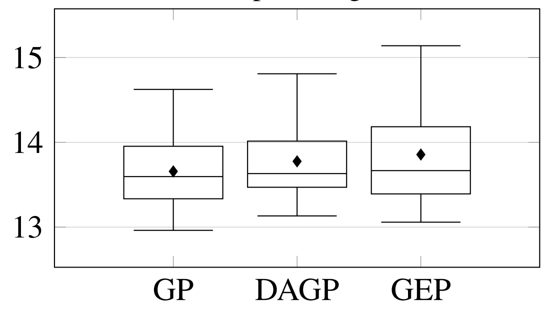
(a) Results for optimising the C_{max} criterion



(b) Results for optimising the Ft criterion



(c) Results for optimising the Nwt criterion



(d) Results for optimising the Twt criterion

Figure 4.3: Comparison between GP, DAGP, and GEP

Chapter 5

Automatic development of dispatching rules for multi-objective and many-objective problems

Multi-objective optimisation is becoming an increasingly important field of study, since in many real world problems it is necessary to optimise several objectives simultaneously. Even in scheduling problems it is common that several objectives need to be optimised together. However, manually designing DRs, which optimise several objectives simultaneously, is a hard trial and error process which requires good knowledge of the underlying problem. Therefore, automatically designing DRs by using multi-objective evolutionary algorithms is becoming an increasingly interesting topic. Unfortunately, very little research has been done in the area of automatically designing DRs for optimising multiple objectives simultaneously, or even on optimising multiple objectives for the unrelated machines environment.

Tay and Ho [26] were the first to consider evolving DRs which optimise several criteria simultaneously. In their study they considered the simultaneous optimisation of three scheduling objectives. However, they did not apply any multi-objective algorithm to solve this problem, but rather transformed the multi-objective problem into a single objective problem. The transformation was performed in a way that all the objectives were combined by using a weighted linear sum which was then optimised. Unfortunately, for such a method it is necessary to have good knowledge about the search space to determine the optimal values for the weights of each objective. Since this knowledge is usually unavailable, this method is in most cases not applicable. Nie et al. [27] propose a multi-objective evolutionary algorithm for optimising four criteria in the single machine environment. Although the authors demonstrate the performance of several evolved DRs, they are unfortunately not compared to any existing DRs measure their effectiveness. Nguyen et al. [28] were the first to apply a multi-objective evolutionary algorithm to generate DRs for optimising five scheduling criteria simultaneously. The authors have shown that the DRs evolved for optimising several objectives simultaneously can perform well when compared to several manually designed DRs. In addition, they also demonstrated that the multi-objective algorithm can evolve DRs which also achieve good results for only a single objective. Further analysis of multi-objective optimisation was performed in [29]. In that study, the authors have applied local search operators with the multi-objective algorithms. With the use of these local search operators it was possible to increase the performance of the multi-objective algorithms. Freitag and Hildebrandt [30] apply NSGA-II to generate DRs for simultaneous optimisation of both the tardiness and earliness criteria. They illustrate that the multi-objective algorithm obtains a wide range of solutions which perform better than some standard DRs. In [31] three multi-objective and many-objective algorithms, including SPEA2, NSGA-II, and NSGA-III, were used to evolve DRs for optimising four and five scheduling objectives simultaneously. The algorithms were compared only by using multi-objective performance measures, however no detailed analysis of the evolved DRs was given. Karunakaran et al. [32] apply an island model GP to evolve DRs for the optimisation of two and three criteria simultaneously. The proposed method has in some cases shown to achieve better performance than some standard multi-objective optimisation methods. Unfortunately, neither of the previous works did an extensive analysis on how the number and combination of optimised scheduling criteria influences the optimisation process and the performance of the evolved DRs.

Regarding multi-objective and many-objective optimisation in the unrelated machines environment, not much research has been done in this area. Fowler et al. [245] applied different approaches to solve the problem of scheduling a printed wiring board manufacturer's drilling operation subject to five optimisation criteria. In [246] the authors have optimised a scheduling problem of a printed wiring board manufacturing line, where six scheduling criteria were optimised simultaneously. The simulated annealing approach was used by Kolahan and Kayvanfar [247], to solve a scheduling problem consisting of the makespan, earliness and tardiness cost, and matching cost objectives. In [248] a scheduling problem consisting of two and three scheduling criteria was optimised by the use of two proposed heuristics and a genetic algorithm. A short overview of some other multi-objective problems in the unrelated machines environment can be found in [249]. Although the research on this topic is quite sparse, the references show that multi-objective and many-objective optimisation problems in the unrelated machines environment appear in many real world situations, therefore outlining the need to develop DRs which are suitable for optimising several criteria simultaneously.

In this chapter different multi-objective GP (MOGP) methods will be applied to develop DRs for optimising various combinations of scheduling criteria. In order to analyse the influence of the number of optimised objectives, scheduling problems consisting of three, five, six, seven, and nine objectives will be optimised. First, a short introduction about the multi-objective optimisation will be presented, after which the description of the experimental design will be given. The parameter optimisation process for the MOGP algorithms will also be described. After that the performance of the MOGP algorithms will be compared with the single objective GP (SOGP). In addition, the different MOGP algorithms will also be compared with each other by using different multi-objective metrics. Several generated DRs will be selected and compared to several standard DRs to analyse whether it is possible to evolve DRs which outperform standard DRs for several criteria. Finally, the mutual correlation between the different scheduling criteria will also be analysed. The chapter is concluded with an overview of the achieved results, and guidelines for further research.

5.1 Multi-objective optimisation

Multi-objective optimisation deals with problems in which two or more criteria need to be optimised simultaneously. However, the term multi-objective is most commonly used to denote only problems in which two or three objectives are optimised simultaneously. In order to denote problems in which more than three objectives are optimised the term *many-objective optimisation* is often used.

For most multi-objective problems a single solution which achieves good results across all criteria does not exist. Therefore the goal is to find solutions which optimise all the criteria well enough. Instead of finding only one best solution as in single objective optimisation, in multi-objective optimisation it is desired to obtain a set of good solutions. This set will contain various solutions, some of which will focus only on optimising one or two criteria (but will not perform well on the remaining criteria), and other solutions will focus on optimising all criteria reasonably well. In the end the user can select the solution which provides the desired trade off between all objectives. However, solutions in multi-objective optimisation can not be compared as easily as in the single objective case, since the quality of solutions is determined by several objective values. In order to deal with this problem two ways of comparing solutions are usually applied in multi-objective optimisation.

The first method for comparing solutions is to define a new measure of quality for each individual as $F(x) = \sum_i w_i f_i(x)$, where x is a sample solution, f_i the objective function, w_i the weight factor of the corresponding objective function, and F the aggregated objective function. In this way, all objective functions are aggregated into a single value, which can then be used for ranking solutions [120]. The advantage of this approach is that standard single objective optimisation methods can be used to solve the multi-objective optimisation problem. However, this approach also has serious disadvantages, which make it rarely used in the literature. The first disadvantage is that different criteria can be of different magnitudes. Therefore the weight factors need to be determined carefully so that the desired influence of all criteria, in the aggregated objective function, is achieved. The second problem is that by using the aggregated

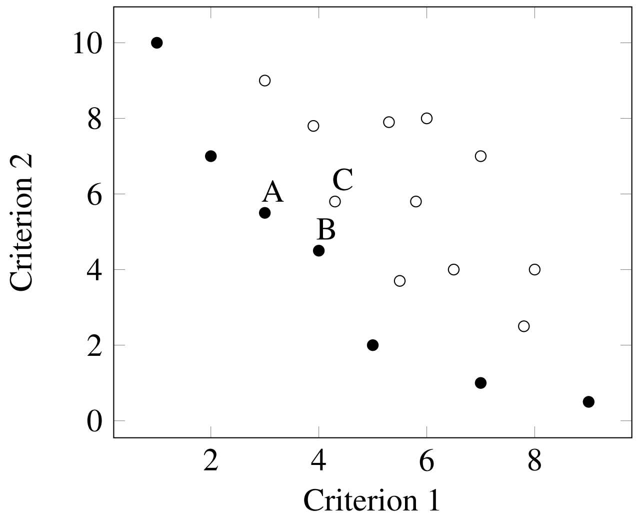


Figure 5.1: Example of a Pareto front

objective function, the search methods will usually converge only to a single solution, or a small number of solutions. Thus, the users will not have the possibility of selecting the solution which optimises all the objectives to a desired degree.

In order to deal with the problems of the aggregated approach, solutions are usually compared by using the dominance relation. In multi-objective minimisation with k objectives, solution x^1 is said to *dominate* solution x^2 if both of the following hold:

1. $f_i(x^1) \leq f_i(x^2)$ for all $i \in \{1,\ldots,k\}$
2. $f_j(x^1) < f_j(x^2)$ for at least one $j \in \{1,\ldots,k\}$.

This means that solution x^1 will dominate solution x^2 if it has a strictly better value for at least one objective, and it does not achieve a worse value than x^2 for all other objectives. A solution which is not dominated by any other solution is said to be *nondominated* or *Pareto optimal*. The set of all nondominated solutions is called the *Pareto front*. Figure 5.1 represents an example of a Pareto front. Solutions which are denoted with black circles represent Pareto optimal solutions, while the white circles represent dominated solutions. For example, the point denoted as C from the figure does not belong to the set of nondominated solutions, since points A and B achieve better values for both criteria, and therefore dominate point C.

5.1.1 Multi-objective GP

Since the basic evolutionary algorithms are intended for optimising a single objective, many adaptations and new algorithms were proposed in the literature, to adapt evolutionary algorithms for solving multi-objective problems. Although these adaptations are usually performed on genetic algorithms, they are not dependent on the problem representation, and can therefore also be used by GP. Various multi-objective and many-objective algorithms were proposed in the literature [20, 250], among which one of the most popular is the *nondominated sorting* *genetic algorithm II* (NSGA-II) [251].

NSGA-II is an optimisation algorithm which extends the standard genetic algorithms to support solving multi-objective problems. Algorithm 5.1 represents the steps which are performed by the NSGA-II algorithm. In the first step the algorithm initialises and evaluates the starting population P. Then in each iteration the algorithm creates a new offspring population C of the same size, by performing crossover and mutation on the individuals from population P. Both populations are combined and *nondominated sorting* is performed on the joint population. The result of the nondominated sort is a classification of all individuals into different fronts. The first front will consist of individuals which are not dominated by any other individual. Each subsequent front will consist of individuals which are not dominated by any other individuals, except for those which were previously assigned to a certain front. After that, a new population is created by iterating through all the fronts, starting with the first front, and adding individuals to the new population. However, before adding the individuals to the new population, the *crowding distance* of all individuals is calculated. The crowding distance of an individual is a measure which provides the information on the density of other individuals which surround that individual. An individual will have a higher crowding distance value the more isolated it is. This measure is useful in the selection process, since selecting solutions which are more isolated leads to a better coverage of the search space. After the crowding distance has been calculated, the individuals in the current front are sorted by it in descending order, and added to the population until it contains the same number of individuals as the old population P. When this happens, the old population P is replaced by the new population N, and a new iteration of the algorithm is started if the termination condition was not fulfilled. The advantage of this algorithm is that it obtains not only one solution, but a set of nondominated solutions, each of which makes a trade off between optimising the different criteria, thus allowing the user to select the solution most befitting of his needs.

Depending on the selected optimisation algorithm, the procedure can mostly be similar to that of NSGA-II, but can also vary significantly. However, all such algorithms have in common that they usually do not obtain only a single solution, but a Pareto optimal set of solutions. Therefore, out of this Pareto set it is possible to select the solution which achieves the requested trade off between the different optimisation criteria.

Unfortunately, unlike in the single objective case, where it is easy to determine which procedure obtains the better solution, the situation is much more difficult for the multi-objective case. This is due to the fact that most multi-objective procedures will return a set of solutions, and therefore it is hard to determine which of the obtained sets is better. Therefore, to be able to compare the solutions obtained by different multi-objective methods, several performance metrics have been defined for multi-objective methods [252, 253]. Some of the more popular metrics used to measure the performance of the obtained Pareto fronts include: hypervolume,

Algorithm 5.1 The NSGA-II algorithm

```
 1: Initialise the population $P$ and evaluate all individuals in it
 2: do
 3:     Generate the child population $C$ by performing crossover and mutation on $P$
 4:     Perform nondominated sorting on $P \cup C$ and save the fronts in the $F$ list
 5:     Let $N = \emptyset$ denote the new population
 6:     Let $i = 0$ denote the index of the front
 7:     while $|N| + |F[i]| < |P|$ do
 8:         Calculate the crowding distance of all individuals in $F[i]$
 9:         Sort all individuals in $F[i]$ by their crowding distance
10:         $N = N \cup F[i]$
11:         $i = i + 1$
12:     end while
13:     Calculate the crowding distance of all individuals in $F[i]$
14:     Sort all individuals in $F[i]$ by their crowding distance
15:     Put individuals from $F[i]$ into $N$ until $|N| = |P|$
16:     $P = N$
17: while termination criterion is not met
```

generational distance, inverted generational distance, the number of nondominated solutions, the epsilon indicator and spread. Most of the performance measures use a reference Pareto front to measure how close the Pareto front obtained by a certain procedure is to the reference Pareto front (convergence), or how good the obtained Pareto front covers the reference Pareto front (diversity). In the metric definitions, S will denote the Pareto front obtained by some multi-objective method whose quality needs to be determined, while P will denote the true Pareto front of the problem, or an approximation of it.

Hypervolume (HV) is a diversity-convergence metric which measures the amount of volume in the objective space that is covered by a given Pareto front [254]. The metric is defined as

$$HV(S,R) = volume\left(\bigcup_{i=1}^{|S|} v_i\right),$$

where v_i represents the hypercube constructed between the point with index i and the reference point R. When minimisation is considered, the reference point is required to be numerically larger or equal than all points in S, for each of the objectives. Therefore, this metric measures the total volume which is enclosed between the solutions in S and the reference point. A larger hypervolume value denotes that a larger volume is enclosed between S and the reference point, which means that the obtained Pareto front S is closer to the reference Pareto front P. However, a larger hypervolume also indicates that the solutions in the obtained Pareto front S are scattered more evenly. Therefore, hypervolume measures both the convergence and diversity of the considered Pareto front S.

Generational distance (GD) represents a convergence metric which measures the distance between the obtained Pareto front S and the reference Pareto front P [255]. The metric is defined as

$$GD(S,P) = \frac{\left(\sum_{i=1}^{|S|} d_i^q\right)^{1/q}}{|S|},$$

where $d_i = min_{p \in P} \|F(s_i) - F(p_i)\|, s_i \in S$, $q = 2$ and $\|\cdot\|$ is defined as the Euclidean distance measure. Therefore, for each point in the Pareto set S obtained by a certain MOGP algorithm, GD selects the point in P which is the closest to it, calculates the distance between those two points, and sums up all the values to get the distance for the entire front.

The inverted generational distance (IGD) [253] represents a diversity-convergence metric which measures the distance between the the approximated Pareto front P, and Pareto front S obtained by some algorithm. This metric is defined as

$$IGD(P,S) = \frac{\left(\sum_{i=1}^{|P|} d_i^q\right)^{1/q}}{|P|},$$

where $d_i = min_{s \in S} \|F(p_i) - F(s_i)\|, p_i \in P$, $q = 2$ and $\|\cdot\|$ is defined as the Euclidean distance measure. As it can be seen, the definition of this metric is very similar to the definition of the GD metric. For each point in the real Pareto set, IGD selects the closest point from the obtained Pareto set S, and calculates the distance between those two points. Because of this modification the IGD also able measures the diversity of the obtained Pareto front S.

The percentage of nondominated solutions (ND) metric denotes the percentage of solutions which are nondominated in the final population of the algorithm. The metric is defined as:

$$ND = \frac{|P|}{|POP|} * 100,$$

where $|P|$ denotes the size of the obtained Pareto front, while $|POP|$ denotes population size used by the algorithm. This metric can also give a notion of the complexity of the optimised multi-objective problem. Larger values of this metric denote that a large number of solutions is required to approximate the real Pareto front, which denotes that the considered multi-objective problem is difficult to optimise.

The epsilon indicator (E) represents a convergence metric [256, 257]. The metric is defined as

$$I_{\varepsilon+}^1(S,P) = \inf_{\varepsilon}\{\forall \vec{p} \in P | \exists \vec{s} \in S : \vec{s} \preceq \vec{p} + \varepsilon\},$$

where $p \preceq s$ denotes that the point s weakly dominates the point p, which means that the solution s is not worse than p in any of the objectives. This metric therefore defines the ε value which denotes the value by which it is required to translate all the solutions in the obtained Pareto front

S, so that all the solutions in it weakly dominate all the solutions in the reference Pareto front P. Thus, the smaller the ε value, the closer the considered Pareto front S is to the reference Pareto front P.

The spread (S) metric represents a diversity metric which calculates the non-uniformity of the obtained Pareto front S [251]. The metric is defined as

$$S(S,P) = \frac{d_f + d_l + \sum_{i=1}^{|S|-1} |d_i - \bar{d}|}{d_f + d_l + (|S|-1) * \bar{d})},$$

where d_f and d_l denote the minimum Euclidean distances between the solutions in S and the extreme (bounding) solutions in the reference Pareto front P, d_i represents the Euclidean distance between consecutive solutions, and $\bar{d}$ represents the average of all distances. A solution set which is well spread out and diverse will have a smaller value of this metric.

5.2 Experimental design

In order to generate DRs for optimising multiple objectives simultaneously, four multi-objective and many-objective algorithms will be used: NSGA-II [251], HaD-MOEA [258], MOEA/D [259], and NSGA-III [260]. NSGA-II was chosen since it represents one of the most commonly used multi-objective algorithms. The HaD-MOEA algorithm was chosen since it was previously used in several studies to generate DRs for optimising several objectives simultaneously. Finally, NSGA-III and MOEA/D were chosen since they represent two popular algorithms for optimising many-objective problems. The reason for choosing four different multi-objective and many-objective algorithms lies in the fact that no single method can achieve the best performance on all different problems. Therefore, all four methods were used obtain insights of the possibility to evolve good DRs for optimising several objectives simultaneously.

For each of the multi-objective and many-objective scheduling problems that will be solved, each of the four MOGP methods is executed 30 times to obtain statistically significant results. The DRs are evolved using the training set consisting of 60 problem instances, and their performance is evaluated on the test set. Through each algorithm run, a set of Pareto optimal solutions for the training set will be obtained. However, not all solutions which belong to the obtained Pareto set are also Pareto optimal on the test set. Therefore, the obtained Pareto front needs to be evaluated on the test set to select only those solutions which belong to the Pareto front for the test set. After the Pareto front on the test set is obtained for each run, these Pareto fronts are evaluated using the six multi-objective metrics described previously. Since the optimal Pareto front is not known for any of the considered scheduling problems, an approximation of the Pareto front will be constructed and used as the reference Pareto front. The approximated Pareto front will be constructed in a way that Pareto fronts obtained by all algorithms in all runs are collected in a single set of solutions, out of which all nondominated solutions are selected and form the approximated Pareto front. Therefore, the approximated Pareto front will consist of the very best solutions which were obtained by all of the algorithms.

5.3 Parameter tuning

In order for the MOGP methods to obtain the best possible results, several parameters of those methods were optimised. For determining the optimal parameter values the many-objective scheduling problem consisting of six criteria (Twt, Ft, Nwt, C_{max}, T_{max}, F_{max}) was selected. The optimisation process is performed so the parameters are optimised one by one, and independently of each other. For each parameter several values are tested, and the one for which the best average value of the hypervolume metric on all the 30 runs is obtained, will be selected as the optimal parameter value. All parameter values which were not optimised are set to the values denoted in Table 4.3, except for the set of mutation operators. For multi-objective and many-objective optimisation only the subtree mutation was used, since this mutation operator leads to the most diverse populations. This is especially important in multi-objective optimisation, where it is crucial to obtain Pareto fronts of diverse solutions. For the initial parameter values the following were chosen: 200000 function evaluations as the termination condition, 100 individuals for the population size, and the mutation probability of 0.5. Regarding the algorithm specific parameters, the following initial values were chosen: 56 reference points for NSGA-III, five neighbours in HaD-MOEA, 10 neighbours and the Tchebycheff decomposition method for MOEA/D. The initial parameters were chosen as a rule of thumb based on some previously conducted experiments.

5.3.1 Optimisation of the number of function evaluations

The first optimised parameter is the number of function evaluations, which acts as the termination criterion for all procedures. It is crucial to find a good value for this parameter, since choosing a too small value can lead to a premature termination of algorithms and therefore the inability of obtaining good solutions. On the other hand, a too large number of evaluations is not only computationally expensive, but can also cause the algorithms to overfit on the training set.

Figure 5.2 represents the dependency of the multi-objective metrics on the number of evaluations which are used as the termination condition. The figure demonstrates that no single algorithm achieves the best performance on all the considered metrics, which justifies the need of using several metrics to objectively evaluate the performance of different multi-objective and many-objective algorithms.

The values of the hypervolume metric steadily grow until the value of 60000 function evaluations is reached, after which the values of the metric increase only slightly. The best overall values for this metric are achieved around 100000 evaluations, after which the values for this metric begin to stagnate for most of the algorithms. The figure denotes that NSGA-II and NSGA-III achieve mostly the same values for the hypervolume metric, except for the smaller number of evaluations where NSGA-II achieves better values. On the other hand the MOEA/D and HaD-MOEA algorithms have mostly achieved worse results for this metric than both NSGA-II and NSGA-III.

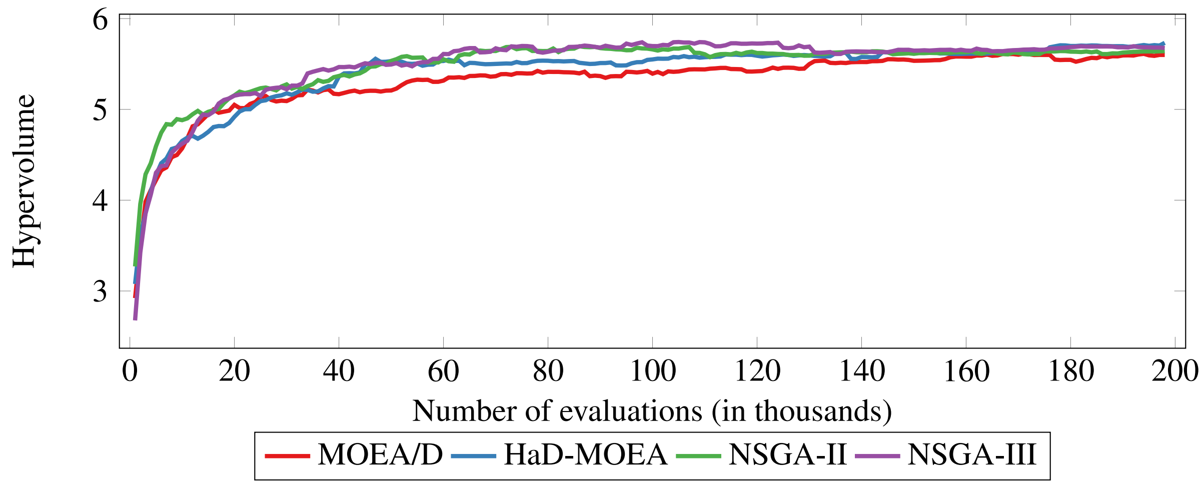
For the GD, IGD, and epsilon metrics the behaviour of the algorithms is mostly the same. The values of all three metrics decrease rapidly until around 40000 function evaluations are performed, after which the decrease in values is quite slow. For the GD metric, all algorithms achieve mostly the same values, with only small differences. However, for the IGD and epsilon metrics the MOEA/D algorithm achieved the overall worst results, while the NSGA-II algorithm usually achieved the best results for those two metrics.

Regarding the other two metrics it is not as evident after what number of evaluations their values start to stagnate. However, it is evident that after around 100000 function evaluations their values do not change as drastically. The NSGA-III algorithm has consistently achieved the best results for the number of nondominated solutions metric. It is also interesting to note that the NSGA-III algorithm was the first to reach the value of 25% of nondominated solutions in the population. Out of the other three algorithms none has consistently achieved the overall worst results, since it was shown that the metric value depends heavily on the number of function evaluations. Regarding the spread metric, MOEA/D achieved the overall worst results. However, neither of the algorithms achieved consistently the best results for this metric, but HaD-MOEA and NSGA-III have mostly performed better than NSGA-II for this metric.

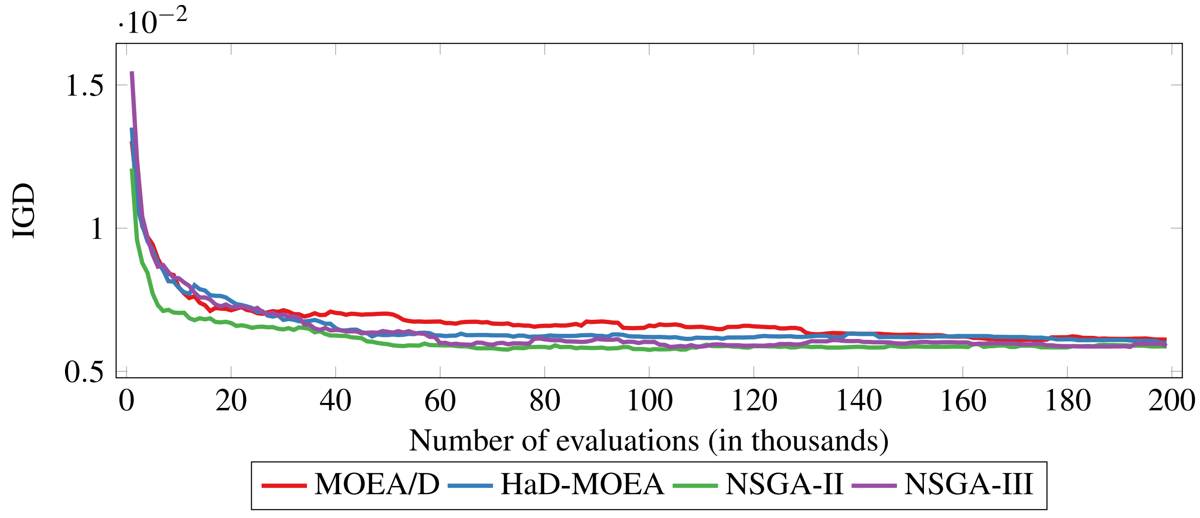
Based on all the metric values it was decided that a number of 100000 function evaluations will be used as the termination criterion, since for that number of evaluations all algorithms achieve good results and the increase in the number of evaluations does not lead to significant improvements in the considered metrics.

5.3.2 Population size optimisation

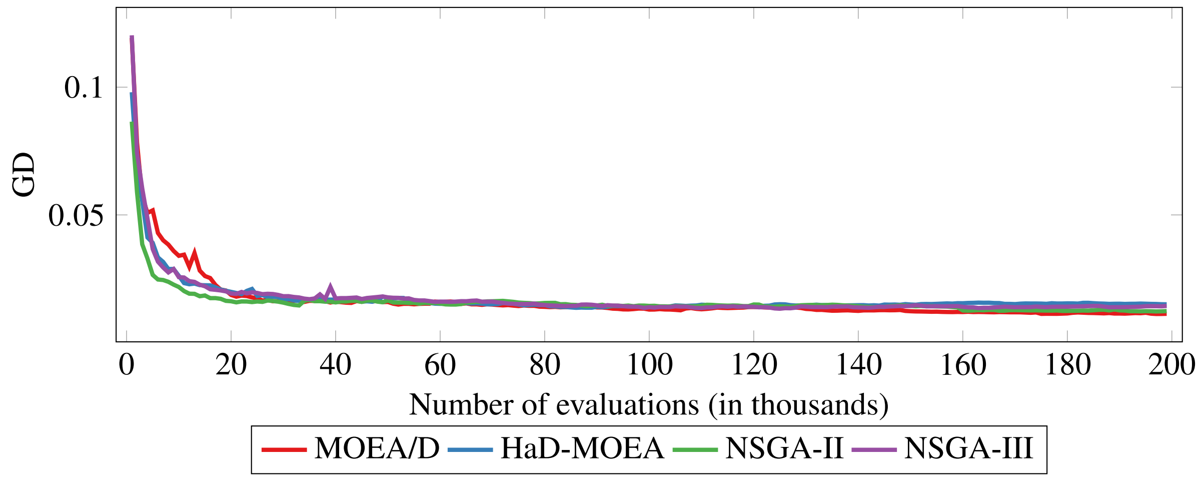
In this section the results obtained for optimising the population size will be presented. Five population sizes were be tested, namely population sizes of 50, 100, 200, 500 and 1000 individuals. For the NSGA-III algorithm different population sizes are used, since it is suggested that the population size is equal to the number of reference points used by the algorithm. Therefore, except for the initial population of 100 individuals, all other population sizes used with NSGA-III will be equal to the number of reference points, which will be kept as close as possible to the population sizes used by the other algorithms.



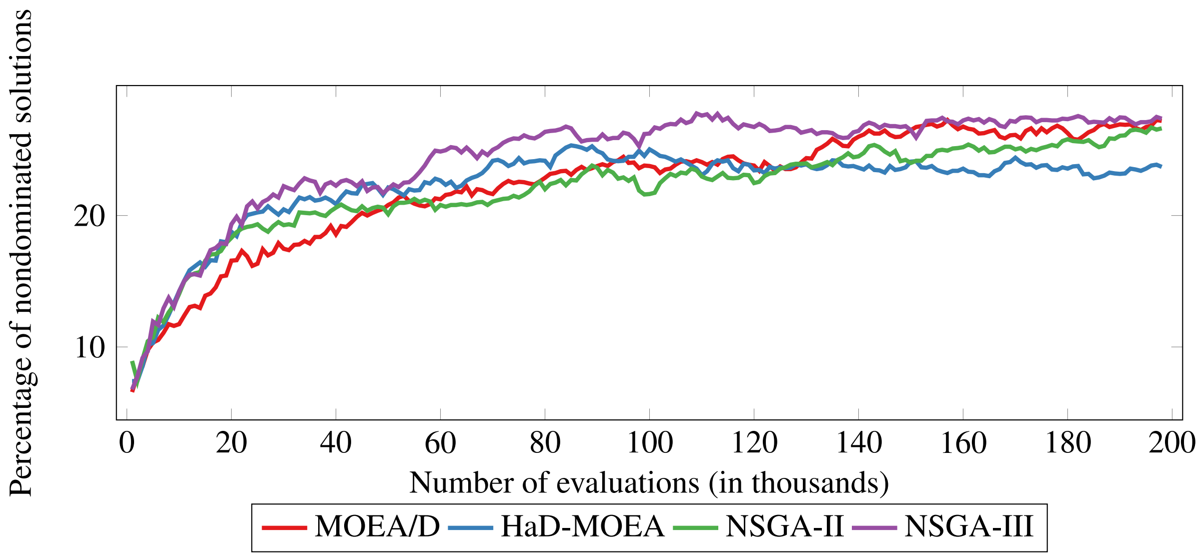
(a) Influence on the hypervolume metric



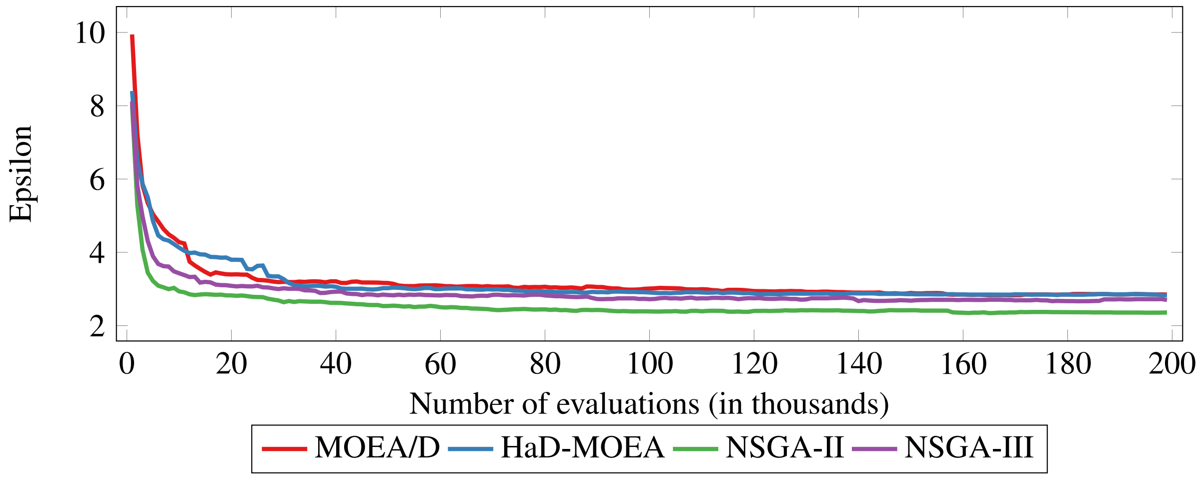
(b) Influence on the IGD metric



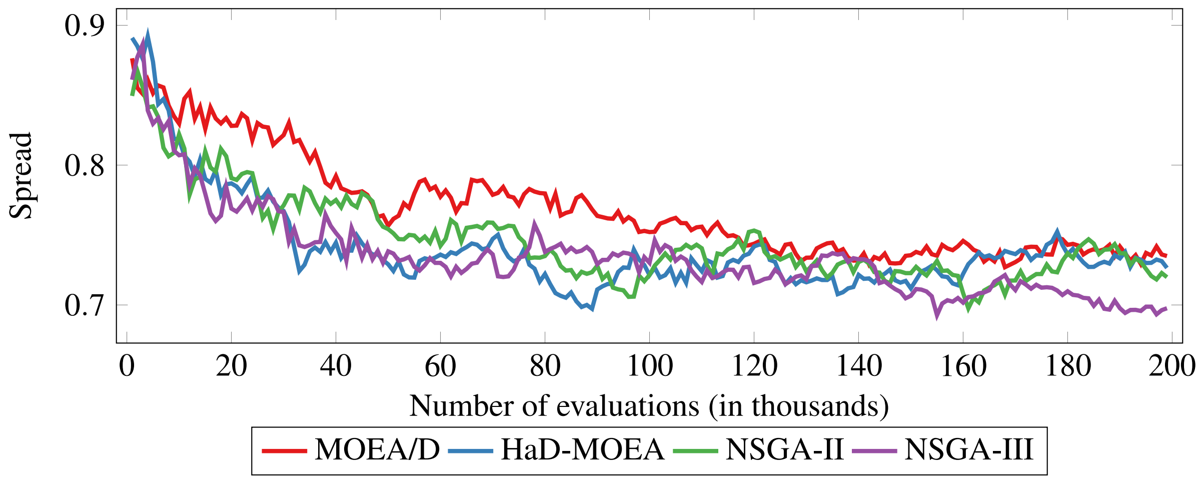
(c) Influence on the GD metric



(d) Influence on the percentage of nondominated solutions metric



(e) Influence on the epsilon metric



(f) Influence on the spread metric

Figure 5.2: Influence of the number of evaluations on the multi-objective metrics

Table 5.1 denotes the influence of the multi-objective metrics on the selected population size. The results show that for all algorithms most of the metrics achieve the best values for the largest population size. However, for the MOEA/D algorithm for half of the metrics the best values were achieved for the smallest population size, while for the other half of the metrics the best values were achieved for the largest population size. Since the hypervolume metric is used to select the optimal parameter value, the population size of 50 individuals will be selected for MOEA/D. For the HaD-MOEA and NSGA-II algorithms the population size of 1000 individuals will be used. The NSGA-III algorithm will also use the largest population size. However, since the number of reference points in NSGA-III depends on the number of criteria which are optimised, the population size will thus vary for the different optimisation problems. The population sizes for NSGA-III will therefore be 1275 for three objectives, 1365 for five objectives, 1287 for six objectives, 924 for seven objectives, and 1287 for nine objectives.

5.3.3 Mutation probability optimisation

The second parameter which also has a large influence on the performance of the algorithms is the mutation probability. It is important to select a good value of this parameter to ensure a good balance between convergence of the algorithm and the diversity of the solutions. For the mutation probability the following five values were used: 0.1, 0.3, 0.5, 0.7 and 0.9. Table 5.2 represents the results achieved for the different multi-objective metrics when using the different values for the mutation probability. The results show that each of the algorithms prefers a different value for the mutation probability, which is most probably the consequence of the different selection mechanisms that each of the algorithms use. The results demonstrate that the multi-objective metrics sometimes achieve their optimal values for different mutation probabilities. As previously when the population size was optimised, the mutation probability which achieved the best value for the hypervolume metric will be selected.

5.3.4 Algorithm specific parameter optimisation

Since the MOEA/D and HaD-MOEA algorithms use additional parameters, the values of those parameters will also be optimised. Both algorithms use a parameter which denotes the neighbourhood size. For this parameter five different values were tried out: 2, 3, 5, 10 and 20. The MOEA/D algorithm uses an additional parameter which determines how the decomposition of the multi-objective problem is performed. For this algorithm three different decomposition methods are compared: the Tchebycheff approach, the normalised Tchebycheff approach, and the boundary intersection (BI) approach.

Table 5.3 represents the influence of the neighbourhood size on the multi-objective metrics. The results show that both algorithms prefer a smaller neighbourhood size, MOEA/D a neigh-

Table 5.1: Influence of the population size on the multi-objective metrics

Algorithm	Metric	Population size				
		50	100	200	500	1000
MOEA/D	HV	**5.692**	5.393	5.681	5.536	5.576
	IGD	**5.759**	6.601	5.776	6.097	6.043
	GD	1.539	1.295	1.173	0.798	**0.745**
	ND	**39.8**	23.8	15.2	8.7	5.0
	E	2.438	3.012	2.428	2.365	**2.253**
	S	0.725	0.752	0.712	0.704	**0.681**
HaD-MOEA	HV	4.956	5.55	5.999	7.0377	**7.272**
	IGD	6.897	6.200	5.490	4.060	**3.808**
	GD	2.987	1.401	1.695	1.153	**0.998**
	ND	20.7	**25.1**	16.1	10.7	7.3
	E	3.509	2.904	3.857	2.000	**1.829**
	S	0.821	0.721	0.665	0.601	**0.597**
NSGA-II	HV	7.107	5.665	8.217	8.474	**8.839**
	IGD	4.782	5.749	3.306	2.862	**2.626**
	GD	3.938	1.439	1.587	0.729	**0.477**
	ND	**40.5**	21.7	32.5	25.6	18.5
	E	3.445	2.388	2.332	1.737	**1.511**
	S	0.548	0.721	0.517	0.497	**0.490**
		NSGA-III population size				
		56	100	252	462	1287
NSGA-III	HV	6.386	5.703	7.634	8.308	**8.381**
	IGD	5.132	6.024	3.593	2.959	**2.790**
	GD	2.037	1.389	0.902	0.601	**0.422**
	ND	**41.5**	26.4	30.3	24.2	16
	E	2.752	2.722	2.061	1.579	**1.394**
	S	0.692	0.730	0.565	0.531	**0.505**

Table 5.2: Influence of the mutation probability on the multi-objective metrics

Algorithm	Metric	Mutation probability				
		0.1	0.3	0.5	0.7	0.9
MOEA/D	HV	5.564	5.553	**5.692**	5.679	5.687
	IGD	6.250	6.157	5.759	**5.564**	5.790
	GD	1.833	1.645	1.539	**1.485**	1.628
	ND	33.9	38.5	39.8	**40.9**	36.4
	E	3.119	2.590	2.438	**2.327**	2.425
	S	0.767	0.737	**0.725**	0.757	0.776
HaD-MOEA	HV	7.396	7.492	7.272	**7.579**	7.486
	IGD	3.851	3.634	3.808	**3.601**	3.721
	GD	1.258	**0.848**	0.998	1.050	1.130
	ND	7.7	**8.1**	7.3	6.4	6.1
	E	2.289	**1.804**	1.829	1.925	2.067
	S	0.585	0.578	0.597	**0.565**	0.600
NSGA-II	HV	**8.945**	8.929	8.839	8.536	8.833
	IGD	2.611	**2.544**	2.626	2.725	2.626
	GD	**0.444**	0.446	0.477	0.522	0.547
	ND	**20.7**	19.2	18.5	18.7	17.7
	E	**1.448**	1.467	1.511	1.477	1.472
	S	0.499	0.498	**0.490**	0.509	0.507
NSGA-III	HV	8.287	8.367	8.381	8.718	**8.915**
	IGD	2.821	2.743	2.790	2.577	**2.264**
	GD	0.410	0.424	0.422	0.431	**0.409**
	ND	14	14.8	16	16.6	**17.8**
	E	1.436	1.429	1.394	**1.378**	1.387
	S	0.509	0.515	0.505	0.513	**0.495**

Table 5.3: Influence of the neighbourhood size on the multi-objective metrics

Algorithm	Metric	Neighbourhood size				
		2	3	5	10	20
MOEA/D	HV	**5.860**	5.848	5.708	5.692	5.617
	IGD	5.692	**5.654**	5.846	5.759	5.896
	GD	2.349	1.760	1.865	1.539	**1.464**
	ND	29.5	37.8	37.1	39.8	**40**
	E	2.343	**2.234**	2.410	2.438	2.331
	S	**0.495**	0.738	0.741	0.725	0.703
HaD-MOEA	HV	7.534	7.477	**7.579**	7.447	7.404
	IGD	3.622	3.716	**3.601**	3.811	3.796
	GD	**0.962**	1.112	1.050	1.066	1.019
	ND	**6.9**	6.3	6.4	6.6	6.8
	E	1.996	2.005	1.925	1.904	**1.895**
	S	0.591	0.607	**0.565**	0.586	0.606

bourhood of size two and HaD-MOEA a neighbourhood of size five. In addition, the results also denote that for no single parameter value were the two MOGP algorithms able to achieve the best performance for all multi-objective metrics.

Table 5.4 represents the results obtained for the different decomposition methods used by the MOEA/D algorithm. The results show that all metrics achieved the best values when the normalised Tchebycheff decomposition method is used. Therefore, this method will be used by MOEA/D for all further experiments.

5.4 Results

This section will present the results achieved by the selected algorithms for different multi-objective and many-objective scheduling problems. In the first part the results achieved by the MOGP algorithms will be compared to those achieved by SOGP to analyse whether MOGP algorithms can match the performance of SOGP for individual criteria. The second part of this section will use the multi-objective metrics to compare the performance of the MOGP algorithms between themselves. Since most tables in this section will include results for several criteria combinations, the results achieved for each combination will be separated by a row

Table 5.4: Influence of the decomposition methods on the multi-objective metrics

Algorithm	Metric	Decomposition method		
		Tchebycheff	Norm. Tchebycheff	BI
MOEA/D	HV	5.86	**6.449**	4.318
	IGD	5.692	**4.839**	8.580
	GD	2.349	**2.081**	2.493
	ND	29.5	**35.5**	27
	E	2.343	**2.330**	2.828
	S	0.695	**0.661**	0.864

denoting the optimised criteria combination for which the subsequent results were obtained.

5.4.1 Comparison of results achieved by MOGP and SOGP

In this section the results obtained by the MOGP algorithms will be compared to the results obtained by SOGP, for each of the optimised multi-objective and many-objective scheduling problems. For each scheduling criterion the individual which achieved the best value for that criterion will be selected from each algorithm run. This means that different individuals can be selected for the various scheduling criteria. The tables will denote the minimum, median, and maximum values for each of the optimised objectives, achieved by the best individuals obtained by all performed algorithm runs. Those experiments for which MOGP achieved significantly better results than SOGP will be denoted in dark grey, while the experiments for which there was no significant difference between the results obtained by MOGP and SOGP will be denoted in light grey.

Table 5.5 represents the results achieved by the different MOGP methods for the simultaneous optimisation of three objectives. The table shows some quite interesting and unexpected results. The MOGP methods achieved results which are not only equal to those of SOGP, but in most cases even better. The NSGA-III algorithm achieved the best results among all the MOGP algorithms, and has, except for one occasion, consistently obtained significantly better results than SOGP. Although the performance of the other MOGP algorithms is not as good as that of NSGA-III, they can, for most criteria, still achieve significantly better or equally good results as SOGP. Out of the remaining three MOGP algorithms, NSGA-II achieved the best performance, followed closely by HaD-MOEA, while the MOEA/D algorithm achieved the worst performance, rarely being able to obtain significantly better results than SOGP. For most criteria the improvements achieved by MOGP algorithms over SOGP are not large, like for example for the Cw criterion where a maximum improvement of 0.7% for the median value was achieved. However, in certain occasions the improvements have shown to be quite significant. The most evident example of this is when optimising the (Nwt, T_{max}, Twt) criteria combination, where NSGA-III evolved DRs that outperform the median values of SOGP by 8.1%, 8.5%, and 3% for the Nwt, T_{max} and Twt criterion, respectively.

Figure 5.3 represents the box plot representation of results for all the tested criteria combinations. The figure denotes how for several criteria the MOGP algorithms achieve much better solution distributions than SOGP. This is especially evident for the Ft and Cw criteria. The magnitude of the improvements largely depends on the combination of criteria which is optimised. When the Twt criterion was paired with the Nwt, and T_{max} criteria, significant improvements were obtained for that criterion when compared to SOGP. On the other hand, if the Twt criterion is combined with the Cw and Ft criteria it is evident that the MOGP algorithms do not achieve any significant improvements, but rather achieve results which are even worse in some cases. Similarly, the best results for the C_{max} criterion are achieved if it is not paired up with any due date related criteria like Twt, T_{max}, or Nwt, but rather if it is optimised together with the F_{max} and Ft criteria. Therefore, by carefully grouping different criteria together, the results achieved by using MOGP algorithms can be improved, and can even outperform the results obtained by SOGP in most cases.

Table 5.6 represents the results achieved for optimising five scheduling criteria simultaneously. The MOGP algorithms can, in most of the cases, once again achieve equally good results as the SOGP algorithm. For the (Cw, $Etwt$, Ft, Nwt, Twt) criteria combination the MOGP algorithms achieve good performance, even outperforming the minimum value of SOGP by 11.6% for the $Etwt$ criterion. For the (F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination the algorithms outperform the median values of SOGP by 3.6% for the Twt criterion, 9.6% for the Nwt criterion, and performing equally well as SOGP for the other three criteria. However, for the (C_{max}, $Etwt$, Ft, M_{ut}, Twt) criteria combination, the improvements which the MOGP algorithms achieved for the optimised criteria are not as prominent as for the other criteria combinations.

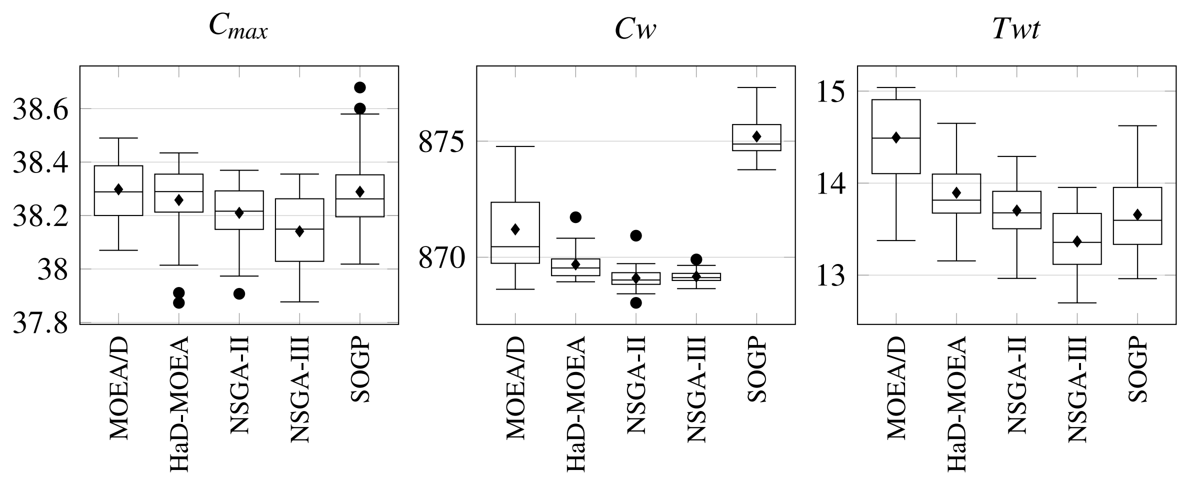
The NSGA-III algorithm achieves dominant results only for the (F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination. For this criteria combination the other MOGP algorithms were also achieved results which are mostly significantly better or equal to those of SOGP. For the (C_{max}, Cw, M_{ut}, F_{max}, Ft) criteria combination the NSGA-III algorithm achieves the best results as well, however, it did not achieve significantly better results than SOGP for all optimised criteria. For the other two criteria combinations the NSGA-II algorithm achieved the best results among all the MOGP algorithms, and for most of the objectives it significantly outperformed the results obtained by SOGP. For this number of criteria it is also evident that the MOGP algorithms have difficulties in outperforming the results obtained by SOGP. This can best be observed when optimising the (C_{max}, $Etwt$, Ft, M_{ut}, Twt) criteria combination, for which only

Table 5.5: Results obtained when optimising three objectives simultaneously

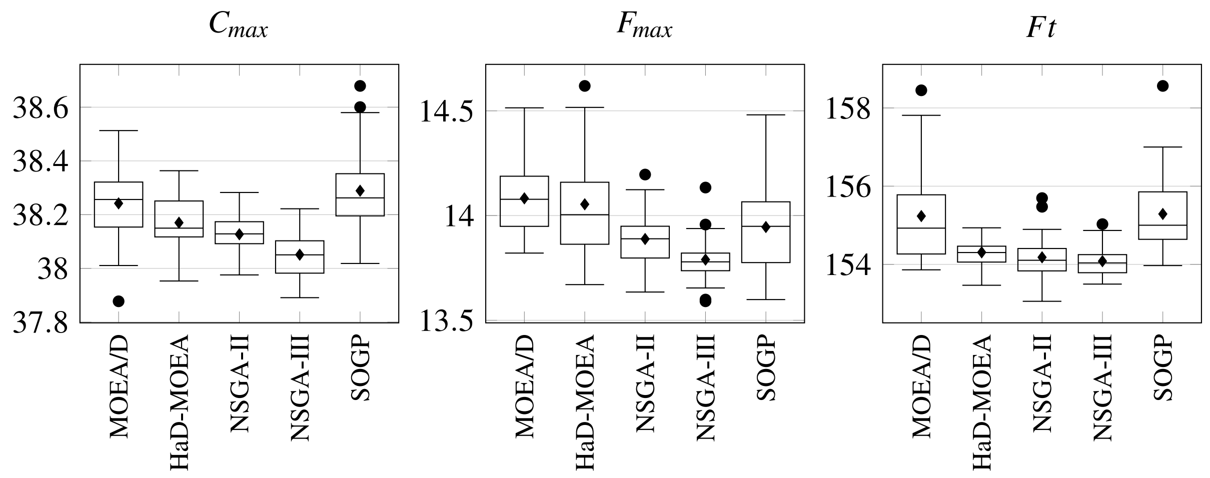
Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
		C_{max}, Cw, Twt				
C_{max}	Min	38.07	**37.87**	37.91	37.88	38.02
	Med	38.29	38.29	38.22	**38.15**	38.26
	Max	38.49	38.43	38.37	**38.36**	38.68
Cw	Min	868.6	868.9	**868.0**	868.6	873.8
	Med	870.6	869.5	**869.0**	869.1	874.9
	Max	874.8	871.7	870.9	**869.9**	877.3
Twt	Min	13.38	13.15	12.96	**12.70**	12.96
	Med	14.50	13.82	13.69	**13.40**	13.60
	Max	15.04	14.65	14.29	**13.95**	14.62
		C_{max}, F_{max}, Ft				
C_{max}	Min	**37.88**	37.95	37.98	37.89	38.02
	Med	38.26	38.16	38.13	**38.05**	38.26
	Max	38.51	38.36	38.28	**38.22**	38.68
F_{max}	Min	13.82	13.67	13.63	**13.59**	13.60
	Med	14.08	14.02	13.89	**13.78**	13.96
	Max	14.51	14.62	14.20	**14.13**	14.48
Ft	Min	153.9	153.5	**153.1**	153.5	154.0
	Med	155.0	154.3	154.1	**154.0**	155.0
	Max	158.5	**154.9**	155.7	155.0	158.6
		C_{max}, F_{max}, T_{max}				
C_{max}	Min	37.99	37.93	**37.83**	37.87	38.02
	Med	38.26	38.21	38.11	**38.09**	38.26
	Max	38.47	38.46	38.29	**38.27**	38.68
F_{max}	Min	13.72	13.81	13.63	**13.59**	13.60
	Med	14.14	14.09	13.86	**13.80**	13.96
	Max	14.71	14.53	14.13	**13.99**	14.48
T_{max}	Min	2.372	2.383	**2.371**	**2.371**	2.376
	Med	2.572	2.478	2.475	**2.444**	2.653
	Max	2.891	2.690	2.671	**2.552**	4.051

Table 5.5: Results obtained when optimising three objectives simultaneously

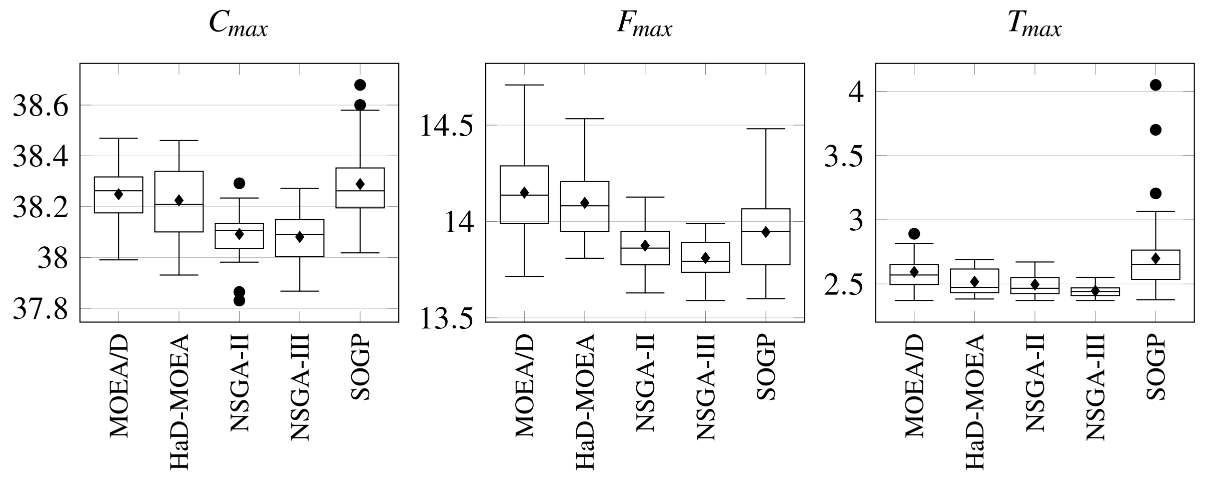
Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
				Cw, Ft, Twt		
Cw	Min	869.3	869.0	**867.9**	**867.9**	873.8
	Med	870.7	849.5	869.2	**869.1**	873.8
	Max	874.8	873.0	870.7	**869.5**	874.9
Ft	Min	153.4	153.3	153.4	**153.1**	154.0
	Med	154.7	154.4	154.0	**153.9**	155.0
	Max	156.3	155.9	155.4	**155.2**	158.6
Twt	Min	13.19	13.02	13.03	**12.79**	12.96
	Med	14.38	14.23	13.87	**13.65**	13.60
	Max	15.76	15.04	15.32	**14.47**	14.62
				Nwt, T_{max}, Twt		
Nwt	Min	6.423	6.235	6.275	**6.164**	6.384
	Med	6.650	6.567	6.456	**6.439**	7.005
	Max	7.138	7.153	**6.810**	6.857	7.939
T_{max}	Min	2.391	2.359	2.327	**2.249**	2.376
	Med	2.538	2.521	2.477	**2.428**	2.653
	Max	2.747	2.789	2.729	**2.560**	4.051
Twt	Min	13.21	12.94	12.72	**12.28**	12.96
	Med	13.74	13.56	13.25	**13.19**	13.60
	Max	14.72	15.72	14.28	**14.12**	14.62



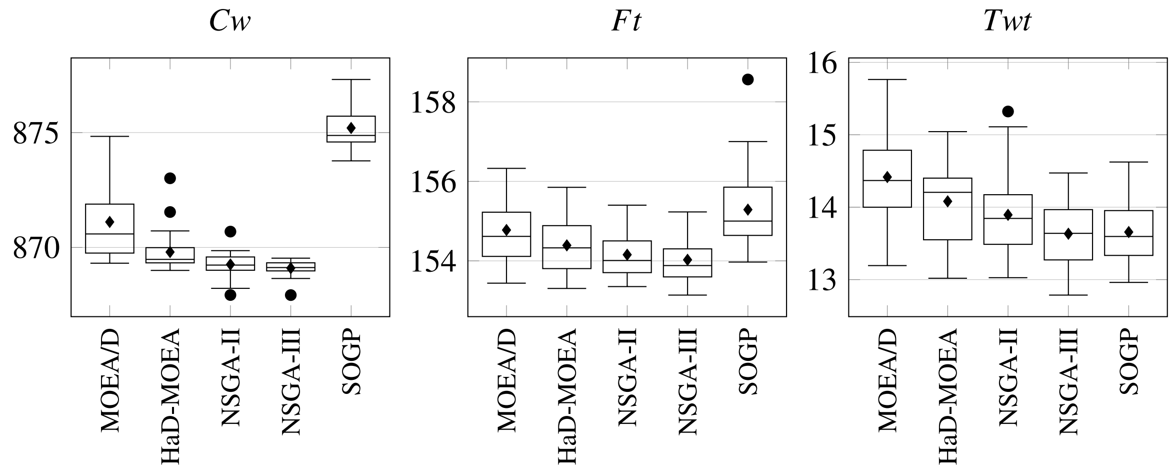
(a) Results obtained when optimising the (C_{max}, Cw, Twt) criteria combination



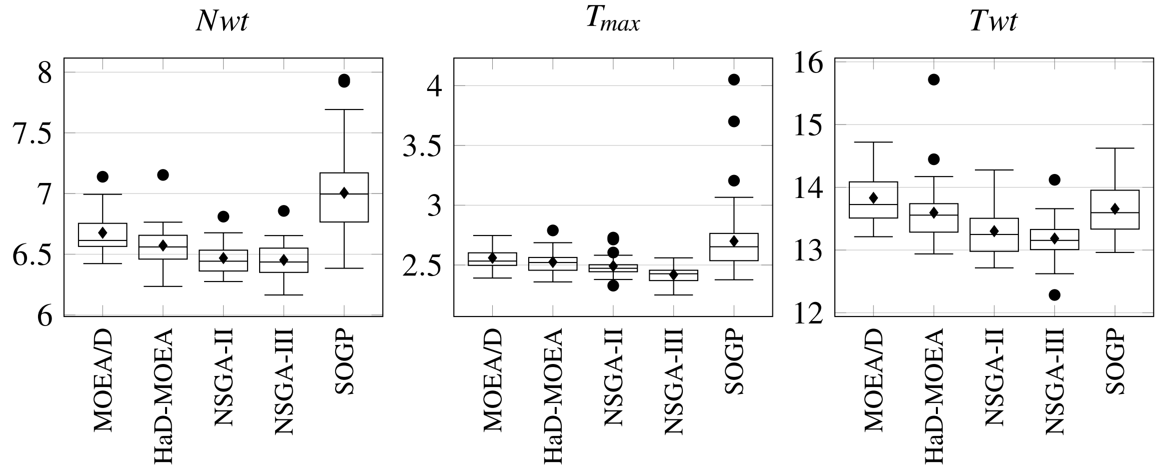
(b) Results obtained when optimising the (C_{max}, F_{max}, Ft) criteria combination



(c) Results obtained when optimising the $(C_{max}, F_{max}, T_{max})$ criteria combination



(d) Results obtained when optimising the (Cw, Ft, Twt) criteria combination



(e) Results obtained when optimising the (Nwt, T_{max}, Twt) criteria combination

Figure 5.3: Box plot representation of the results obtained when optimising three objectives simultaneously

Table 5.6: Results obtained when optimising five objectives simultaneously

Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
		$C_{max}, Cw, M_{ut}, F_{max}, Ft$				
C_{max}	Min	38.06	37.90	**37.83**	37.85	38.02
	Med	38.31	38.16	38.06	**38.04**	38.26
	Max	38.52	38.32	38.25	**38.21**	38.68
Cw	Min	868.3	869.4	868.5	**868.0**	873.8
	Med	871.9	870.1	869.6	**869.2**	874.9
	Max	880.2	874.9	874.0	**870.2**	877.3
M_{ut}	Min	0.048	0.049	0.047	**0.046**	**0.046**
	Med	0.057	0.057	**0.054**	**0.054**	**0.054**
	Max	0.076	0.064	**0.057**	0.059	0.058
F_{max}	Min	13.90	13.76	13.74	13.85	**13.60**
	Med	14.17	14.03	**13.91**	13.92	13.96
	Max	14.73	14.56	14.29	**14.08**	14.48
Ft	Min	153.8	153.4	153.7	**153.2**	154.0
	Med	155.7	154.4	154.1	**153.7**	155.0
	Max	167.3	157.0	155.2	**154.5**	158.6
		$C_{max}, Etwt, Ft, M_{ut}, Twt$				
C_{max}	Min	38.09	37.90	**37.87**	38.01	38.02
	Med	38.49	38.17	**38.16**	38.28	38.26
	Max	39.64	**38.34**	38.42	38.53	38.68
$Etwt$	Min	267.7	269.1	**200.0**	239.9	236.9
	Med	300.3	315.1	**265.5**	273.8	274.4
	Max	394.4	360.6	305.8	**291.1**	303.0
Ft	Min	153.4	**153.3**	**153.3**	155.7	154.0
	Med	157.0	155.7	**154.5**	163.3	155.0
	Max	171.6	159.3	**156.8**	178.2	158.6
M_{ut}	Min	0.051	0.052	**0.046**	0.048	**0.46**
	Med	0.063	0.061	**0.051**	0.052	0.54
	Max	0.077	0.070	**0.056**	**0.056**	0.58
Twt	Min	13.67	13.01	13.16	13.42	**12.96**
	Med	15.25	13.83	13.75	14.02	**13.60**
	Max	18.48	14.71	14.26	15.16	**14.62**

Table 5.6: Results obtained when optimising five objectives simultaneously

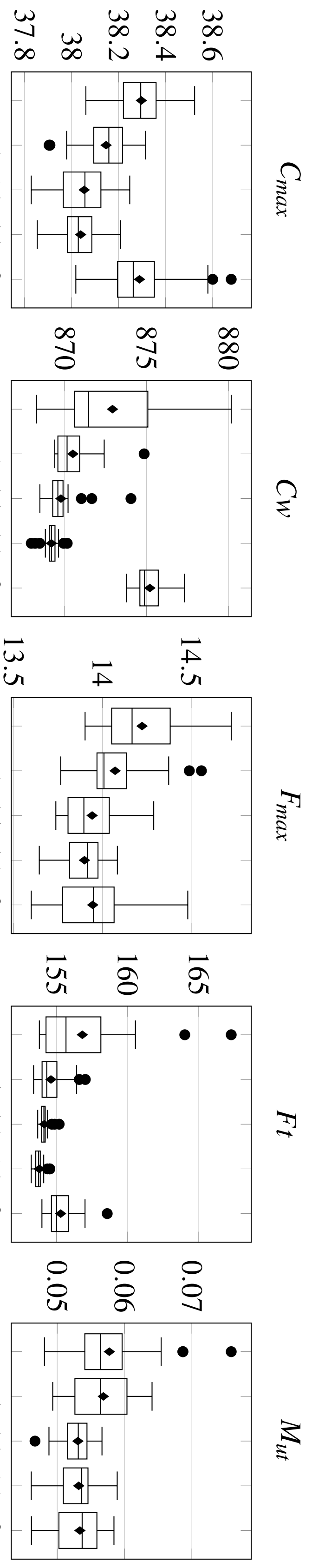
Metrics		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
		$Cw, Etwt, Ft, Nwt, Twt$				
Cw	Min	869.3	869.2	869.3	**867.0**	873.8
	Med	875.2	871.8	**871.0**	871.3	874.9
	Max	888.5	877.6	**874.5**	875.4	877.3
$Etwt$	Min	261.0	279.8	**209.4**	242.7	236.9
	Med	302.9	320.5	**260.6**	273.8	274.4
	Max	393.0	420.2	**284.6**	298.3	303.0
Ft	Min	153.5	153.7	**153.4**	153.9	154.0
	Med	156.9	155.7	**154.6**	155.3	155.0
	Max	171.4	160.6	**156.8**	157.2	158.6
Nwt	Min	6.454	6.457	6.329	**6.270**	6.384
	Med	6.885	6.600	6.546	**6.476**	7.005
	Max	7.395	7.036	6.946	**6.694**	7.939
Twt	Min	13.12	13.19	12.86	**12.66**	12.96
	Med	14.28	13.64	13.36	**13.34**	13.60
	Max	15.60	14.70	14.17	**13.67**	14.62
		$F_{max}, Ft, Nwt, T_{max}, Twt$				
F_{max}	Min	13.90	13.70	**13.60**	13.66	**13.60**
	Med	14.48	14.11	13.90	**13.81**	13.96
	Max	15.42	15.55	14.16	**14.10**	14.48
Ft	Min	153.9	153.4	**153.3**	**153.3**	154.0
	Med	155.3	154.3	**153.7**	153.8	155.0
	Max	157.4	155.9	154.7	**154.6**	158.6
Nwt	Min	6.342	6.365	6.174	**6.121**	6.384
	Med	6.742	6.632	**6.391**	6.414	7.005
	Max	6.979	6.932	6.731	**6.584**	7.939
T_{max}	Min	2.361	2.385	**2.330**	2.350	2.376
	Med	2.593	2.457	2.424	**2.398**	2.653
	Max	2.800	2.635	2.547	**2.478**	4.051
Twt	Min	13.32	12.80	**12.68**	12.72	12.96
	Med	14.01	13.65	13.13	**13.11**	13.60
	Max	14.82	14.79	13.86	**13.47**	14.62

the NSGA-II algorithm achieved a good performance for all the optimised criteria. The other MOGP algorithms did not perform well for this criteria combination, since they achieved significantly worse results than SOGP for some of the criteria. Therefore, it is evident that for a small increase in the number of optimised criteria a single algorithm is unable to achieve the best results for all the considered optimisation problems.

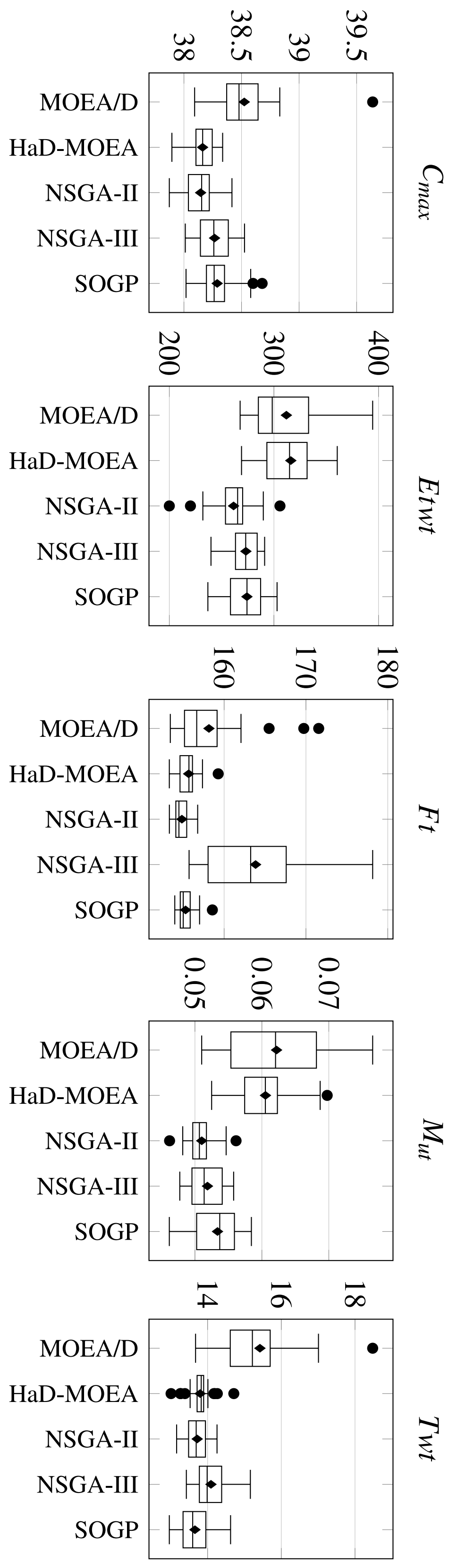
Figure 5.4 shows the box plot representation of the results for optimising five scheduling criteria simultaneously. Once again it can be seen that the choice of the criteria which are optimised together heavily influences the performance of the MOGP algorithms. For example, when the flowtime and due date related criteria are paired together as in the (F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination, the MOGP algorithms can achieve good performance for all the criteria. However, if the Twt criterion is paired with the C_{max} criterion, the performance of the MOGP algorithms on both criteria is similar to that of SOGP. Pairing the Ft criterion with the $Etwt$ criterion also causes the MOGP algorithms to achieve bad performance for the Ft criterion. The MOGP methods achieved the worst results when optimising the (C_{max}, $Etwt$, Ft, M_{ut}, Twt) criteria combinations, which is probably due to the fact that it consists of several different criteria types, and therefore it is hard for the MOGP algorithms to obtain good values for all the criteria.

Table 5.7 represents the results achieved for optimising six criteria simultaneously. For this number of criteria, only the combination which was used for optimising the algorithm parameters will be considered. The NSGA-III algorithm achieved significantly better results than SOGP for all of the optimised criteria. The other algorithms were able to achieve good results as well, performing significantly better or equally well as SOGP for most of the optimised criteria. Out of the other algorithms, NSGA-II can even perform better for certain criteria than NSGA-III. For the due date related criteria the MOGP algorithms achieved improvements over the median values obtained by SOGP of approximately 9.5% for the Nwt criterion, 10.1% for the T_{max} criterion, and 4.8% for the Twt criterion. For the other optimised criteria the obtained improvements are not as prominent. Even though scheduling criteria of different types were optimised together, it did not have any negative influence on the performance of the MOGP methods. The reason for this could be that the $Etwt$ and M_{ut} criteria were not included in the optimised combination of criteria.

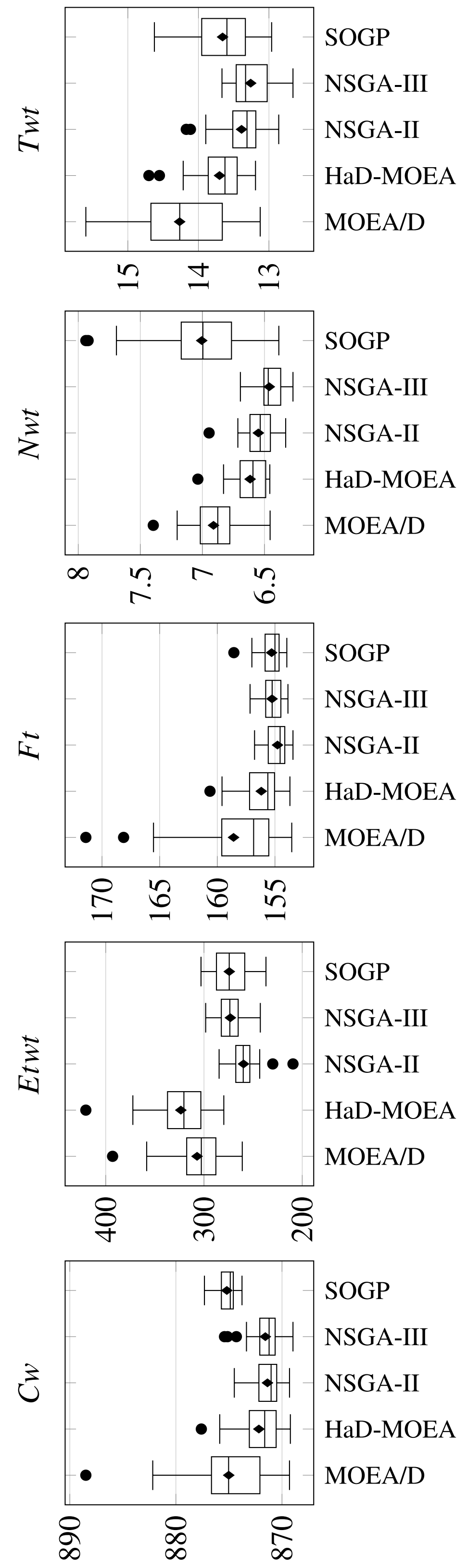
Figure 5.5 shows the box plot representation of the results for optimising six criteria simultaneously. The figure denotes that the NSGA-III algorithm obtains the best solution distributions out of all MOGP algorithms for all six criteria. The NSGA-II algorithm also achieved good results on all of the tested criteria, while the other two algorithms struggle on certain criteria. Out of the optimised criteria, the NSGA-III algorithm achieved much better solution distributions than SOGP for the C_{max}, Ft, Nwt, T_{max} and Twt criteria, while on the other hand the method struggled mostly for the F_{max} criterion, but still managed to achieve significantly better results



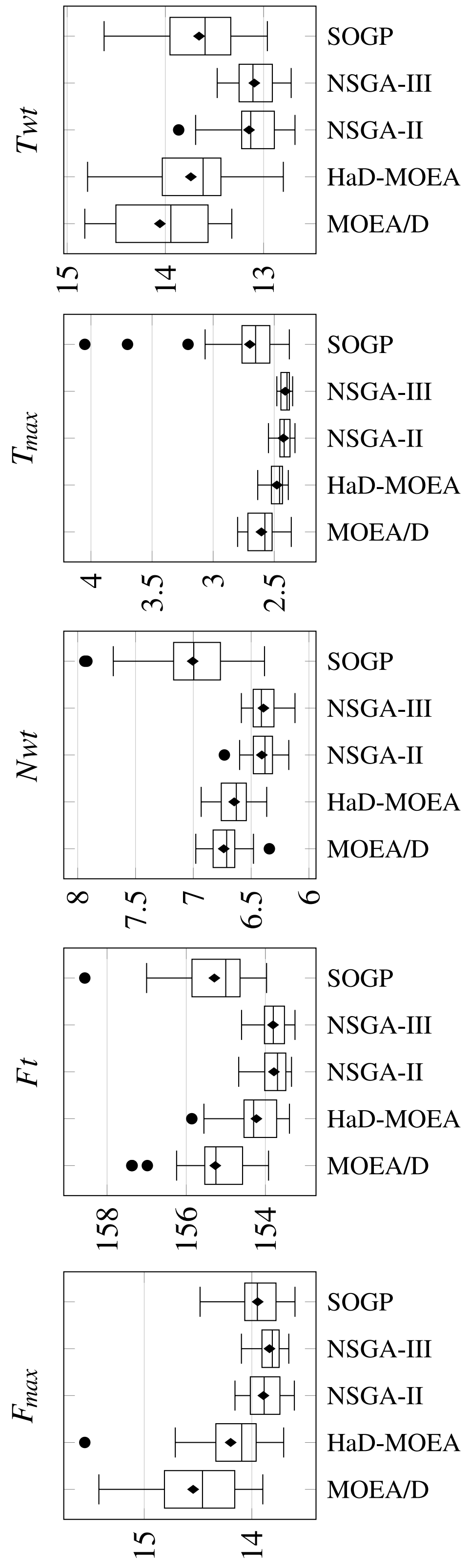
(a) Results obtained when optimising the (C_{max}, C_W, F_{max}, Ft, M_{ut}) criteria combination



(b) Results obtained when optimising the (C_{max}, $Etwt$, Ft, M_{ut}, Twt) criteria combination



(c) Results obtained when optimising the (Cw, $Etwt$, Ft, Nwt, Twt) criteria combination



(d) Results obtained when optimising the (F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination

Figure 5.4: Box plot representation of the results obtained when optimising five objectives simultaneously

Table 5.7: Results obtained when optimising six objectives simultaneously

Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
C_{max}	Min	38.10	38.04	37.89	**37.87**	38.02
	Med	38.34	38.23	38.11	**38.07**	38.26
	Max	38.47	38.38	38.25	**38.24**	38.68
F_{max}	Min	13.97	13.69	13.64	**13.55**	13.60
	Med	14.25	14.06	13.87	**13.82**	13.96
	Max	15.14	14.39	**14.05**	14.08	14.48
Ft	Min	153.7	153.4	**153.1**	153.2	154.0
	Med	154.9	154.1	153.7	**153.5**	155.0
	Max	157.2	155.7	154.7	**154.4**	158.6
Nwt	Min	6.480	6.272	6.188	**6.183**	6.384
	Med	6.730	6.606	**6.338**	6.380	7.005
	Max	6.993	6.915	6.624	**6.567**	7.939
T_{max}	Min	2.423	2.337	**2.301**	2.347	2.376
	Med	2.611	2.428	2.398	**2.384**	2.653
	Max	2.800	2.665	**2.494**	2.516	4.051
Twt	Min	13.13	12.85	**12.59**	12.70	12.96
	Med	14.20	13.49	13.09	**12.95**	13.60
	Max	14.94	14.44	13.85	**13.60**	14.62

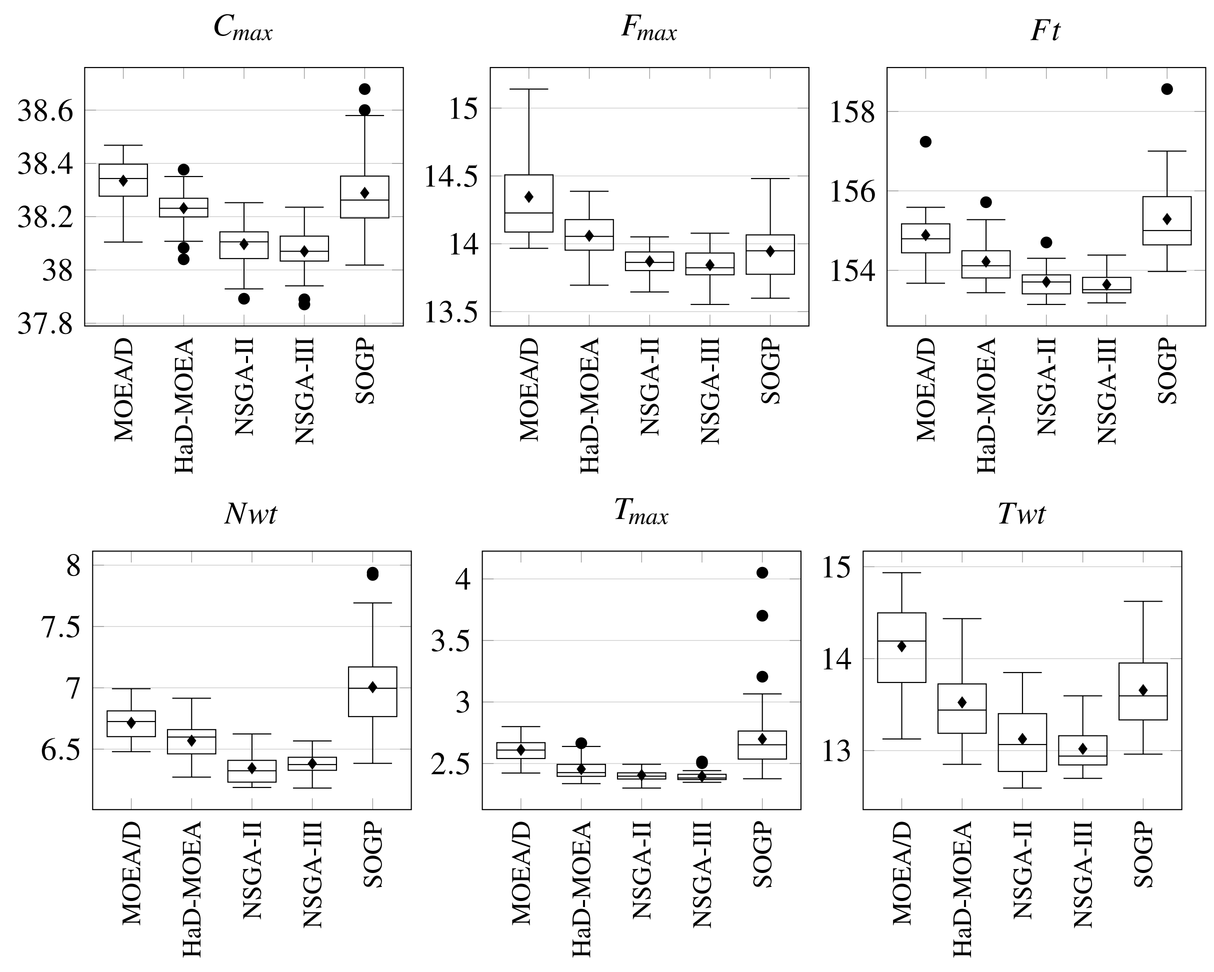


Figure 5.5: Box plot representation of the results obtained when optimising six objectives simultaneously

than SOGP even for that criterion.

Table 5.8 represents the results which are achieved by the MOGP algorithms when optimising seven criteria simultaneously. For the (C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination the NSGA-II and NSGA-III algorithms both achieved the best results, with NSGA-II achieving slightly better results for most of the criteria. For the given criteria combination both algorithms achieve significantly better results than SOGP for six criteria, and equally good results on the remaining criterion. The other MOGP algorithms are unable to perform equally well, but can still outperform SOGP for several optimised criteria. The MOGP algorithms again achieve good performance for the due date related criteria, resulting in improvements of 8.4%, 9.5%, and 4.0% over the median values of SOGP for the Nwt, T_{max}, and Twt criteria, respectively. This proves that even for larger criteria combinations the MOGP algorithms can significantly outperform the results of SOGP. However, on the other two criteria combinations the NSGA-II algorithm achieves the best performance, with NSGA-III and HaD-MOEA performing worse to a small extent. For those two criteria combinations one can observe that for certain criteria the multi-objective methods are unable to achieve significantly better results than

SOGP.

Figure 5.6 shows the box plot representation of the results for optimising seven criteria simultaneously. The figure denotes that the best solutions for all criteria are mostly achieved when optimising the $(C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt)$ criteria combination. For the other two criteria combinations the NSGA-II algorithm is usually the only method which achieved good solution distributions for most of the criteria. The other three algorithms exhibit more problems on these two criteria combinations. By analysing the criteria combinations on which the multi-objective algorithms do not perform well, it is can be observed that in those two criteria combinations the M_{ut} or $Etwt$ criteria are optimised. Based on the fact that the multi-objective methods achieved inferior results when these criteria were also included in the five objective optimisation problems, it can be concluded that the algorithms perform well if the $Etwt$ and M_{ut} criteria are not included in the optimisation set. On the other hand, the inclusion of those two criteria in the optimisation set has a negative effect on the performance of the MOGP algorithms, especially on NSGA-III which with the inclusion of those criteria becomes unable to match the performance of NSGA-II.

Table 5.9 represents the results for optimising all nine scheduling criteria simultaneously. For this criteria combination the MOGP algorithms exhibit further deterioration in their performance. The NSGA-II algorithm again achieved the best results out of all the considered MOGP algorithms. When compared to the results obtained by SOGP, NSGA-II achieved significantly better results in five occasions. The other three algorithms achieved significantly better results than SOGP for at most three criteria. Therefore, for this criteria combination the NSGA-II algorithm is the most dominant out of all the MOGP algorithms. It should be noted that the NSGA-III algorithm achieved a significant deterioration in the results, achieving even the worst results out of all methods for certain criteria. Therefore, NSGA-III seems to perform poorly when many different criteria types are optimised together. For this criteria combination the algorithms are unable to achieve equally good improvements over SOGP as previously.

Figure 5.7 shows the box plot representation of the results for optimising all nine criteria simultaneously. The figure shows that although the differences in the results between MOGP algorithms and SOGP are not as large as for smaller criteria combinations, the MOGP algorithms can still outperform SOGP for certain criteria. The figure also demonstrates how for several criteria the NSGA-III algorithm achieves quite bad solution distributions, in many cases even worse than that of SOGP. The worst solution distributions by the MOGP algorithms are achieved for the $Etwt$, F_{max}, Ft and Twt criteria.

The results have demonstrated that the MOGP algorithms can outperform, or perform at least equally well as SOGP for most of the tested criteria combinations. Although MOGP algorithms achieved better performance on smaller criteria combinations than on larger ones, it nevertheless seems that the combination of criteria which are optimised has a larger influence

Table 5.8: Results obtained when optimising seven objectives simultaneously

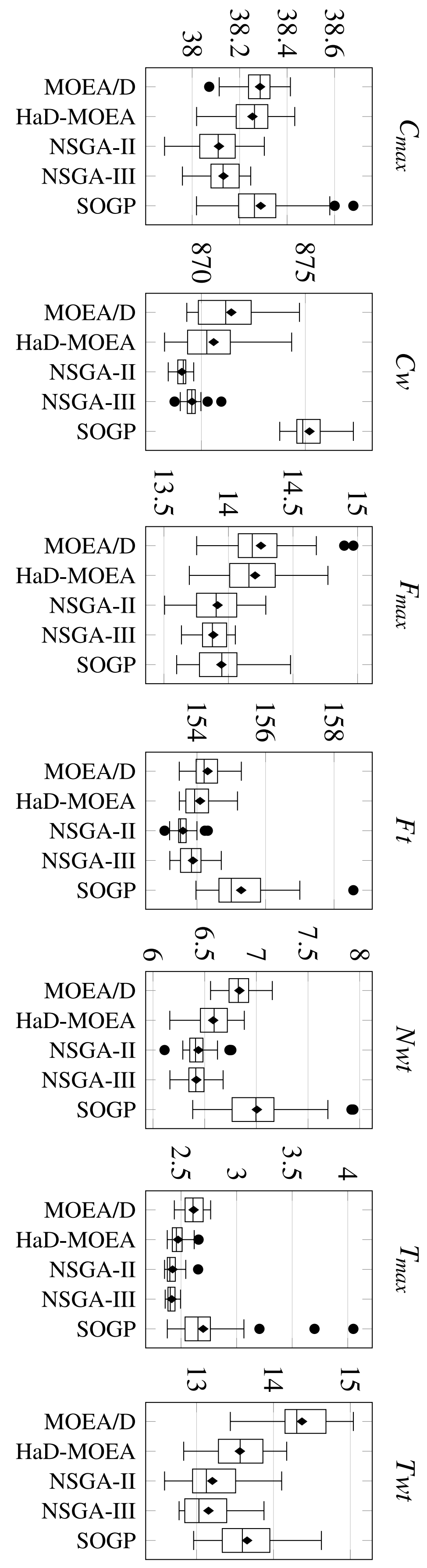
Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
		$C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt$				
C_{max}	Min	38.07	38.02	**37.88**	37.96	38.02
	Med	38.29	38.28	**38.11**	38.14	38.26
	Max	38.41	38.43	38.30	**38.25**	38.68
Cw	Min	869.3	**868.2**	868.4	868.7	873.8
	Med	871.2	870.3	**869.1**	869.5	874.9
	Max	874.7	874.3	**869.6**	871.0	877.3
F_{max}	Min	13.75	13.70	**13.50**	13.64	13.60
	Med	14.21	14.17	13.91	**13.88**	13.96
	Max	14.97	14.77	14.29	**14.05**	14.48
Ft	Min	153.5	153.5	**153.1**	153.2	154.0
	Med	154.2	154.0	**153.5**	153.9	155.0
	Max	155.3	155.2	**154.3**	154.7	158.6
Nwt	Min	6.557	6.163	**6.112**	6.164	6.384
	Med	6.837	6.593	**6.416**	6.428	7.005
	Max	7.154	6.884	6.758	**6.678**	7.939
T_{max}	Min	2.440	2.376	**2.352**	2.357	2.376
	Med	2.609	2.459	**2.402**	2.404	2.653
	Max	2.766	2.658	2.654	**2.496**	4.051
Twt	Min	13.44	12.83	**12.58**	12.77	12.96
	Med	14.31	13.59	13.13	**13.06**	13.60
	Max	15.04	14.17	14.11	**13.88**	14.62

Table 5.8: Results obtained when optimising seven objectives simultaneously

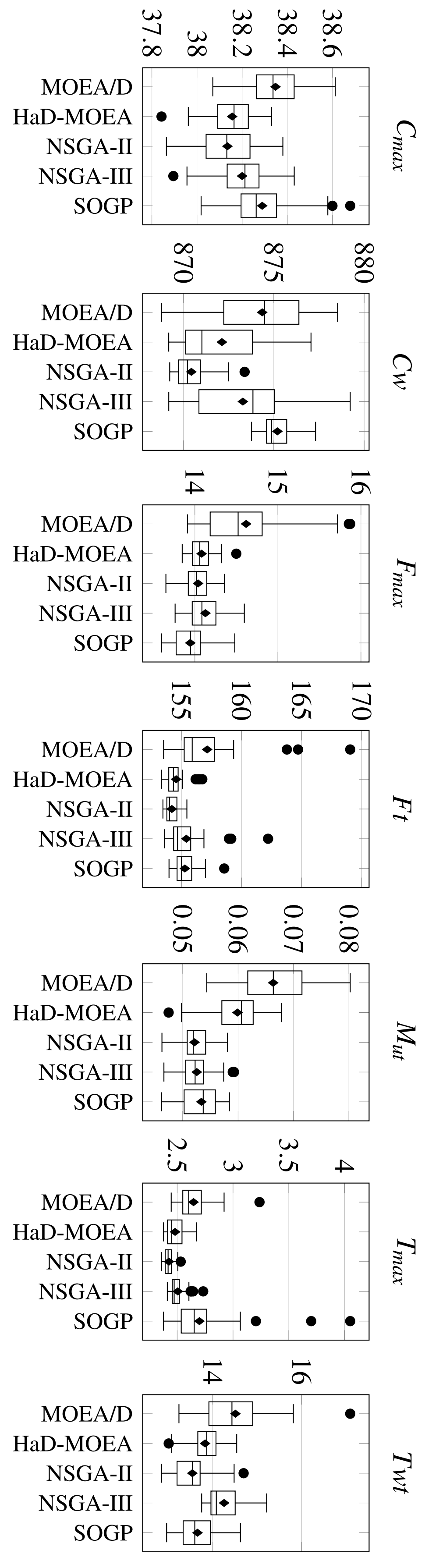
Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
$C_{max}, Cw, F_{max}, Ft, M_{ut}, T_{max}, Twt$						
C_{max}	Min	38.07	**37.84**	37.86	37.90	38.02
	Med	38.35	38.17	**38.14**	38.21	38.26
	Max	38.61	**38.33**	38.38	38.43	38.68
Cw	Min	868.8	**869.2**	**869.2**	**869.2**	873.8
	Med	874.6	871.6	**870.3**	873.9	874.9
	Max	878.5	877.1	**873.4**	879.2	877.3
F_{max}	Min	13.91	13.85	13.65	13.76	**13.60**
	Med	14.52	14.06	14.03	14.09	**13.96**
	Max	15.87	14.50	**14.36**	14.60	14.48
Ft	Min	153.5	**153.3**	153.5	153.6	154.0
	Med	156.2	154.3	**154.0**	154.8	155.0
	Max	169.0	156.7	**155.5**	162.2	158.6
M_{ut}	Min	0.054	0.047	**0.046**	0.047	**0.046**
	Med	0.066	0.061	**0.052**	**0.052**	0.054
	Max	0.080	0.068	**0.058**	0.059	**0.058**
T_{max}	Min	2.446	2.377	**2.359**	2.412	2.376
	Med	2.605	2.454	**2.420**	2.478	2.653
	Max	3.238	2.672	**2.532**	2.733	4.051
Twt	Min	13.24	13.01	**12.85**	13.75	12.96
	Med	14.44	13.88	**13.57**	14.11	13.60
	Max	17.09	14.54	14.69	15.21	**14.62**

Table 5.8: Results obtained when optimising seven objectives simultaneously

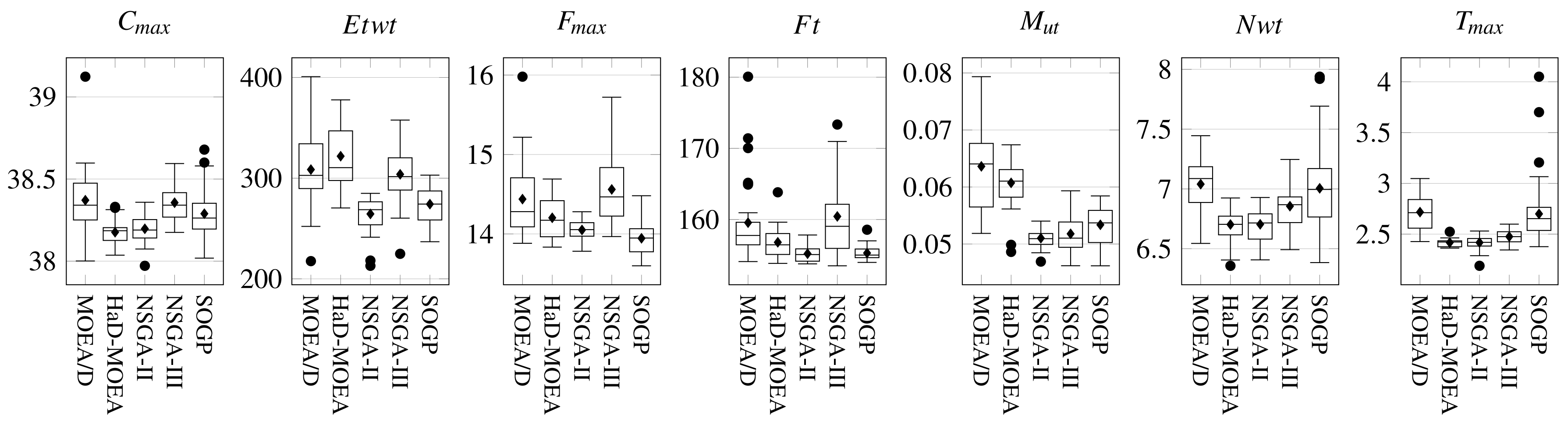
Metrics		Algorithm				
		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
		C_{max}, $Etwt$, F_{max}, Ft, M_{ut}, Nwt, T_{max}				
C_{max}	Min	38.00	38.04	**37.97**	38.18	38.02
	Med	38.34	**38.19**	**38.19**	38.34	38.26
	Max	39.12	**38.33**	38.36	38.59	38.68
$Etwt$	Min	217.6	270.4	**213.0**	224.9	236.9
	Med	303.7	311.3	**269.8**	301.7	274.4
	Max	400.8	377.8	**284.9**	357.7	303.0
F_{max}	Min	13.88	13.83	13.78	13.97	**13.60**
	Med	14.30	14.17	14.07	14.48	**13.96**
	Max	15.98	14.69	**14.28**	15.72	14.48
Ft	Min	154.1	153.8	153.8	**153.5**	154.0
	Med	157.9	156.5	155.2	159.2	**155.0**
	Max	180.1	163.8	**157.8**	173.3	158.6
M_{ut}	Min	0.052	0.049	0.047	**0.046**	**0.046**
	Med	0.065	0.061	**0.051**	**0.051**	0.054
	Max	0.080	0.067	**0.054**	0.059	0.058
Nwt	Min	6.545	**6.357**	6.407	6.492	6.384
	Med	7.087	**6.707**	6.716	6.870	7.005
	Max	7.445	**6.925**	6.929	7.247	7.939
T_{max}	Min	2.427	2.362	**2.188**	2.344	2.376
	Med	2.714	**2.423**	**2.423**	2.478	2.653
	Max	3.047	**2.523**	2.531	2.598	4.051



(a) Results obtained when optimising the (C_{max}, C_w, F_{max}, F_t, N_{wt}, T_{max}, T_{wt} criteria combination



(b) Results obtained when optimising the (C_{max}, C_w, F_{max}, F_t, M_{ut}, T_{max}, T_{wt} criteria combination



(c) Results obtained when optimising the (C_{max}, $Etwt$, F_{max}, Ft, M_{ut}, Nwt, T_{max} criteria combination

Figure 5.6: Box plot representation of the results obtained when optimising seven objectives simultaneously

Table 5.9: Results obtained when optimising nine objectives simultaneously

Metrics		MOEA/D	HaD-MOEA	NSGA-II	NSGA-III	SOGP
C_{max}	Min	**37.89**	38.00	37.98	38.09	38.02
	Med	38.36	38.21	**38.19**	38.30	38.26
	Max	38.67	**38.35**	38.43	38.55	38.68
Cw	Min	870.5	870.4	**869.8**	874.2	873.8
	Med	874.8	874.5	**872.5**	878.4	874.9
	Max	885.4	885.8	**877.0**	887.1	877.3
$Etwt$	Min	275.9	282.5	253.7	270.3	**236.9**
	Med	313.1	327.0	278.6	288.9	**274.4**
	Max	368.9	382.2	320.8	328.8	**303.0**
F_{max}	Min	14.03	13.89	13.76	13.90	**13.60**
	Med	14.39	14.21	14.04	14.49	**13.96**
	Max	16.64	15.36	14.56	15.98	**14.48**
Ft	Min	154.3	154.1	**153.7**	154.4	154.0
	Med	157.6	156.3	155.7	160.7	**155.0**
	Max	168.8	166.9	**157.6**	171.6	158.6
M_{ut}	Min	0.052	0.048	**0.046**	0.048	**0.046**
	Med	0.064	0.062	**0.050**	0.052	0.054
	Max	0.078	0.069	**0.055**	0.062	0.058
Nwt	Min	6.536	6.414	6.434	6.531	**6.384**
	Med	6.978	6.700	**6.685**	6.802	7.005
	Max	7.268	6.981	**6.906**	7.128	7.939
T_{max}	Min	2.389	2.279	**2.269**	2.325	2.376
	Med	2.649	**2.400**	2.431	2.453	2.653
	Max	3.034	2.588	2.581	**2.546**	4.051
Twt	Min	13.39	12.97	13.08	13.48	**12.96**
	Med	14.59	**13.58**	13.73	14.13	13.60
	Max	17.52	**14.53**	**14.53**	15.08	14.62

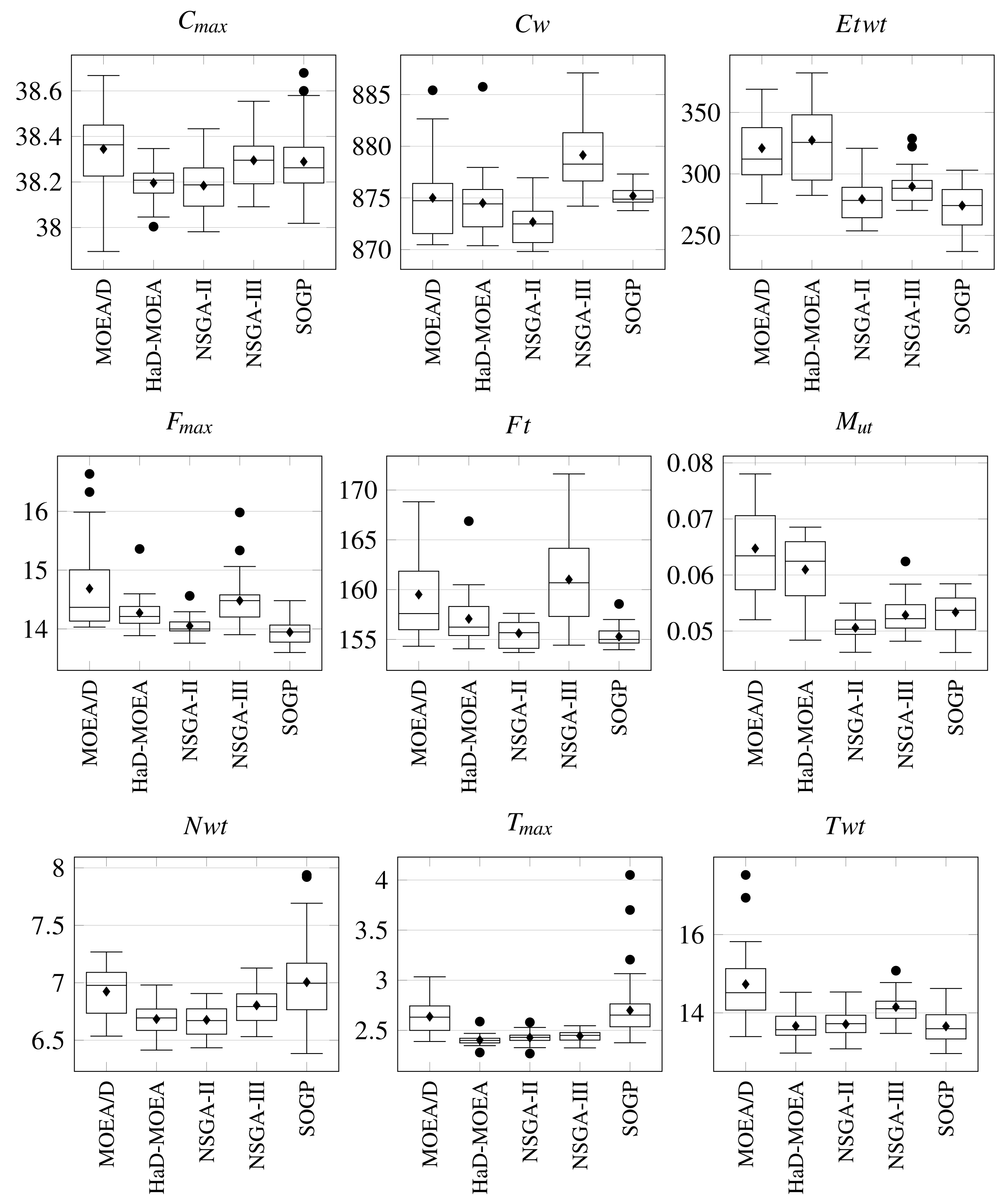


Figure 5.7: Box plot representation of the results obtained when optimising nine objectives simultaneously

on the possibility of the MOGP algorithms to outperform SOGP. By grouping together similar criteria, like in the (Nwt, T_{max}, Twt) criteria combination, MOGP algorithms can significantly outperform SOGP for each of the optimised criteria. It seems that by grouping similar criteria together the MOGP algorithms have a "wider" look on the problem, which allows them to achieve much better performance on each of the optimised criteria. However, when different criteria types are mixed with each other, the performance of the MOGP algorithms will depend on which types of criteria were combined. For example, combining flowtime and completion time criteria did not have an influence on the effectiveness of the MOGP algorithms, but when combining the due date related criteria with the completion time criteria as in the (C_{max}, Cw, Twt) criteria combination, or when combining different types of criteria as in the (Cw, Ft, Twt) criteria combination, the performance of the MOGP algorithms decreases, and some are even unable to outperform SOGP for certain criteria. It is interesting to note that as the number of criteria increases, but they still contain different criteria types as in the (F_{max}, Ft, Nwt, T_{max}, Twt) and (C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt) combinations, the MOGP algorithms are able to outperform SOGP in most of the cases. It is also interesting to note that the performance of some MOGP algorithms can even improve when optimising criteria combinations consisting of five and six criteria. Therefore it seems to be beneficial to include several criteria of the same type in the combination, since this can lead to better performance instead of using only a single criteria of each type. On the other hand, combining criteria of the same definitions (like maximum values, or weighted sums) did not show any significant influence on the results, therefore it seems that grouping criteria in this way is not beneficial.

It is interesting to analyse how the inclusion of the $Etwt$ and M_{ut} criteria influences the performance of MOGP algorithms. When optimising five criteria, by including either the $Etwt$ or M_{ut} criterion, the MOGP algorithm are still able to outperform SOGP. However, the MOEA/D and HaD-MOEA algorithms start to have more problems when these criteria are included. When both of the criteria are included at the same time, then even the performance of NSGA-III declines, and only NSGA-II can outperform the results of SOGP almost consistently. A similar thing can be observed when optimising seven criteria simultaneously. NSGA-II usually outperforms results of SOGP for almost all objectives. On the other hand, the other three algorithms struggle much more to outperform SOGP, especially if both the $Etwt$ and M_{ut} criteria are optimised simultaneously. When optimising nine criteria, even NSGA-II starts to struggle to outperform SOGP. Based on the previous observations, it is evident that for a smaller number of criteria including either $Etwt$ or M_{ut} will not be problematic, and most algorithms will still be able to outperform SOGP. However, by increasing the number of criteria or including both criteria in the combination, the results of all algorithms start to deteriorate, and the algorithms become unable to outperform SOGP for an increasing number of criteria.

Based on the results shown in this section, it can be concluded that MOGP algorithms can perform better or equally well as SOGP, for most criteria combinations. If criteria of the same type are grouped together and optimised, the MOGP algorithms can achieve significantly better performance than SOGP. As for the dependency on the criteria combinations, it was shown that the performance of MOGP algorithms depends more on the combination of criteria which are optimised, than on the number of optimised criteria. MOGP algorithms perform well for optimising all criteria combinations which do not include the $Etwt$ and M_{ut} criteria, since in most cases optimising those two criteria leads to a poor performance on all other optimised criteria. The root of this issue seems to originate from the fact that the Pareto front becomes quite elongated by optimising those two criteria, and therefore the algorithms tend to focus more on finding the solutions along the Pareto front, and focus much less on finding solutions at the extreme points of each criterion. Thus, the search space increases drastically with the inclusion of those two criteria, which negatively influences the algorithms.

5.4.2 Performance comparison of MOGP algorithms

In order to analyse the performance of the selected MOGP algorithms with regards to the quality of the Pareto fronts they obtain, the values of the multi-objective metrics will be calculated for the tested multi-objective and many-objective problems. The tables will represent the average values of the metrics based on 30 executions for each criteria combination. The best values achieved for the different criteria combinations will be denoted in bold. Since the HV and IGD metrics are the two most commonly used multi-objective metrics, which take into consideration both the convergence and diversity of the Pareto front, statistical tests will be applied for those two metrics to determine which algorithms achieve the best results. In addition, the values for all metric, except for the number of nondominated solutions, will be represented with box plots. The number of nondominated solutions will not be presented since it does not provide any significant information about the quality of the obtained Pareto fronts.

Table 5.10 represents the results achieved for the multi-objective metrics when optimising three criteria simultaneously. The NSGA-III algorithm consistently achieved the best values for the HV and IGD metrics across all the criteria combinations. The MOEA/D algorithm achieved the largest number of nondominated solution, which is expected since it was applied with a small population size. The results also denote that all the algorithms achieved a very small value for that metric, which means that they obtained Pareto fronts consisting of a small number of solutions. For the other three criteria no single algorithm achieved the best values across all criteria combinations. NSGA-II has mostly achieved the best results for the GD metric, while NSGA-III achieved the best values for the E and S metrics.

Figure 5.8 shows the box plot representation of the multi-objective metrics when three criteria are optimised simultaneously. The figure denotes that the NSGA-II and NSGA-III algorithms achieve the best results for all the metrics, whereas the other two algorithms achieve

Table 5.10: Multi-objective metric values obtained when optimising three objectives simultaneously

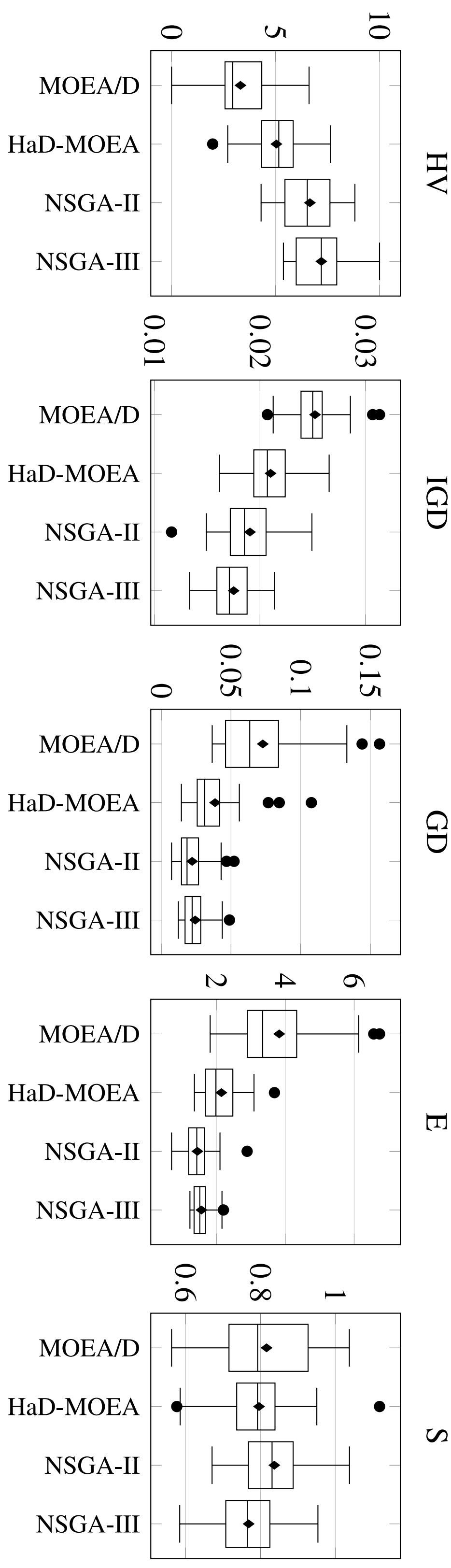
Metrics	Algorithm			
	MOEA/D	HaD-MOEA	NSGA-II	NSGA-III
		C_{max}, Cw, Twt		
HV	3.312	5.040	6.651	**7.201**
IGD	0.025	0.021	0.019	**0.018**
GD	0.073	0.039	**0.022**	0.024
ND	**15.4**	2.1	3.8	2.7
E	3.826	2.147	**1.451**	1.569
S	0.816	0.796	0.837	**0.768**
		C_{max}, F_{max}, Ft		
HV	3.840	5.031	6.661	**7.341**
IGD	0.033	0.028	0.021	**0.020**
GD	0.64	0.041	**0.020**	0.024
ND	**15**	1.7	3.1	2.2
E	2.234	1.274	1.157	**1.031**
S	0.781	0.737	0.779	**0.731**
		$C_{max}, F_{max}, T_{max}$		
HV	3.036	4.006	6.649	**7.309**
IGD	0.052	0.050	0.040	**0.037**
GD	0.336	0.116	**0.060**	0.064
ND	**4**	0.2	0.3	0.23
E	0.625	0.551	0.375	**0.349**
S	**0.751**	0.795	0.816	0.770
		Cw, Ft, Twt		
HV	3.523	5.539	7.400	**7.578**
IGD	0.030	0.025	0.022	**0.019**
GD	0.058	0.043	**0.021**	0.025
ND	**15**	1.5	3.25	2.1
E	4.215	2.985	1.888	**1.871**
S	0.847	0.884	0.874	**0.800**
		Nwt, T_{max}, Twt		
HV	4.698	5.706	6.872	**7.25**
IGD	0.708	0.589	0.443	**0.382**
GD	1.221	1.187	0.512	**0.494**
ND	**5**	0.35	0.45	0.31
E	1.548	1.312	1.018	**0.909**
S	0.899	0.859	0.794	**0.777**

inferior results. The differences are usually the smallest for the S metric, which means that all algorithms evolve Pareto fronts with good a dispersion. However, the convergence of the obtained Pareto fronts by MOEA/D and HaD-MOEA are worse than that of the other two algorithms. For the HV metric, NSGA-II and NSGA-III achieve significantly better results for all criteria combinations when compared to the other two algorithms. Between those two algorithms, NSGA-III is significantly better when optimising the (C_{max}, F_{max}, T_{max}) and (C_{max}, F_{max}, Ft) criteria combinations, while for the other three criteria combinations there is no significant difference. As for the IGD metric, NSGA-III and NSGA-II have again proven to be significantly better than the other two algorithms, with NSGA-III achieving significantly better results than NSGA-II for all criteria combinations except the (Nwt, T_{max}, Twt) criteria combination. Therefore, when optimising three criteria simultaneously, the NSGA-III algorithm obtained Pareto fronts of the best quality among all the tested algorithms.

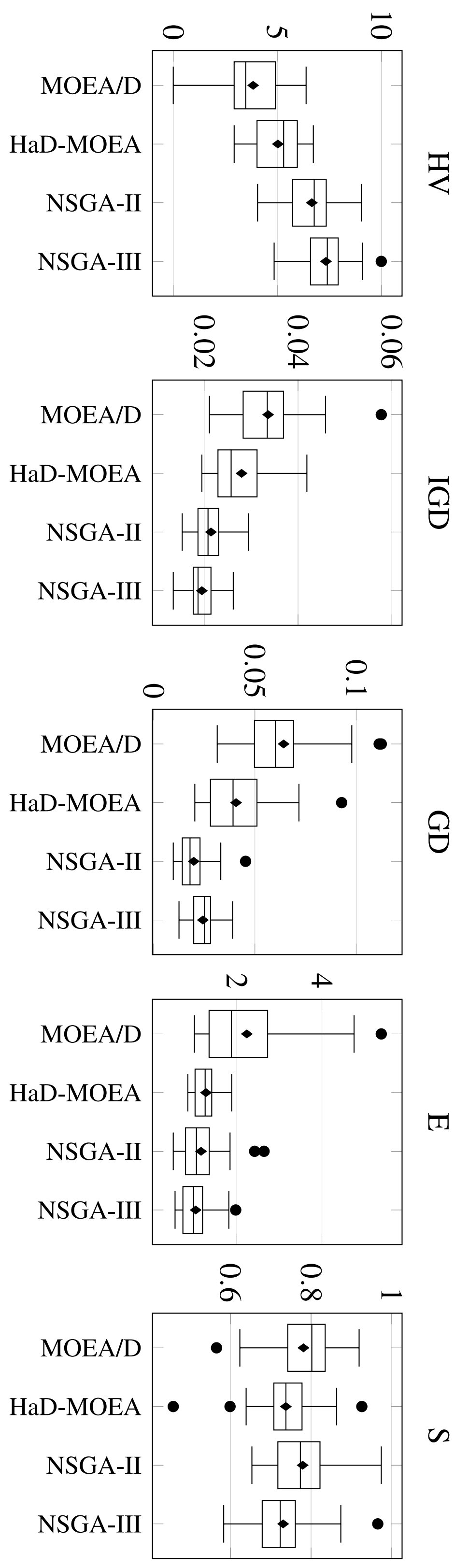
Table 5.11 represents the results for the multi-objective metrics achieved for optimising five criteria simultaneously. Since small values were achieved for the IGD and GD metrics, they were both multiplied by 100 to more easily denote their values in the table. The NSGA-III algorithm achieved the best performance on most of the criteria combinations, however NSGA-II also achieved good performance, even outperforming NSGA-III for certain metrics. As for the percentage of nondominated solutions, the value for this metric is larger than it was when only three criteria were optimised. Therefore, it seems that the value of this metric depends on the number of criteria which are optimised. A larger percentage of nondominated solutions usually signalises that certain criteria are negatively correlated and therefore many solutions which balance between those criteria can be obtained. This is best evident when optimising the (C_{max}, $Etwt$, Ft,M_{ut}, Twt) and (Cw, $Etwt$, Ft, Nwt, Twt) criteria combinations, which both include the $Etwt$ criterion. For both of those combinations the percentage of nondominated solutions was at least two times larger than for the other optimised criteria combinations.

Figure 5.9 shows the box plot representation of the metric values for the simultaneous optimisation of five criteria. The figure denotes that once again the NSGA-II and NSGA-III algorithms achieved better performance than the other two algorithms. On the other hand the MOEA/D algorithm usually achieved the worst results for all the metrics. For the HV metric, NSGA-II achieves significantly better results than all other algorithms for all criteria combinations, except for the (C_{max}, Cw, F_{max}, Ft, M_{ut}) and (F_{max}, Ft, Nwt, T_{max}, Twt) criteria combinations where there is no significant difference between it and NSGA-III. For the IGD criteria, the NSGA-III algorithm achieved significantly better results for all criteria combinations, except for (C_{max}, $Etwt$, Ft, M_{ut}, Twt) and (F_{max}, Ft, Nwt, T_{max}, Twt), where there was no significant difference between it and NSGA-II, and for (Cw, $Etwt$, Ft, Nwt, Twt) where NSGA-II achieved better results.

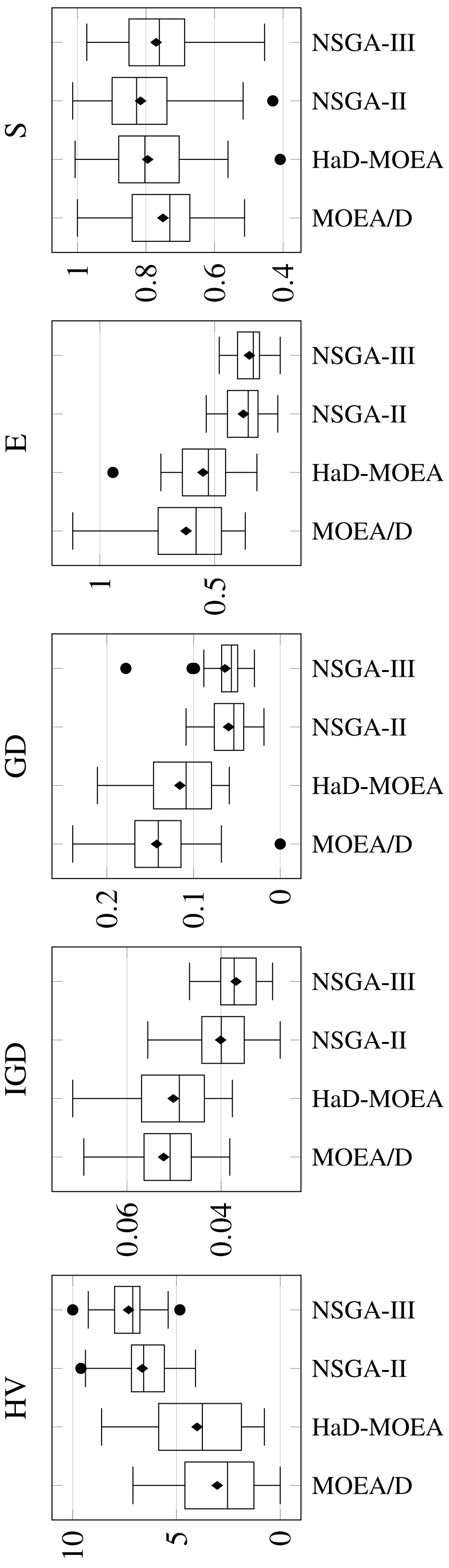
Based on the results achieved by the metrics for the five criteria optimisation problems,



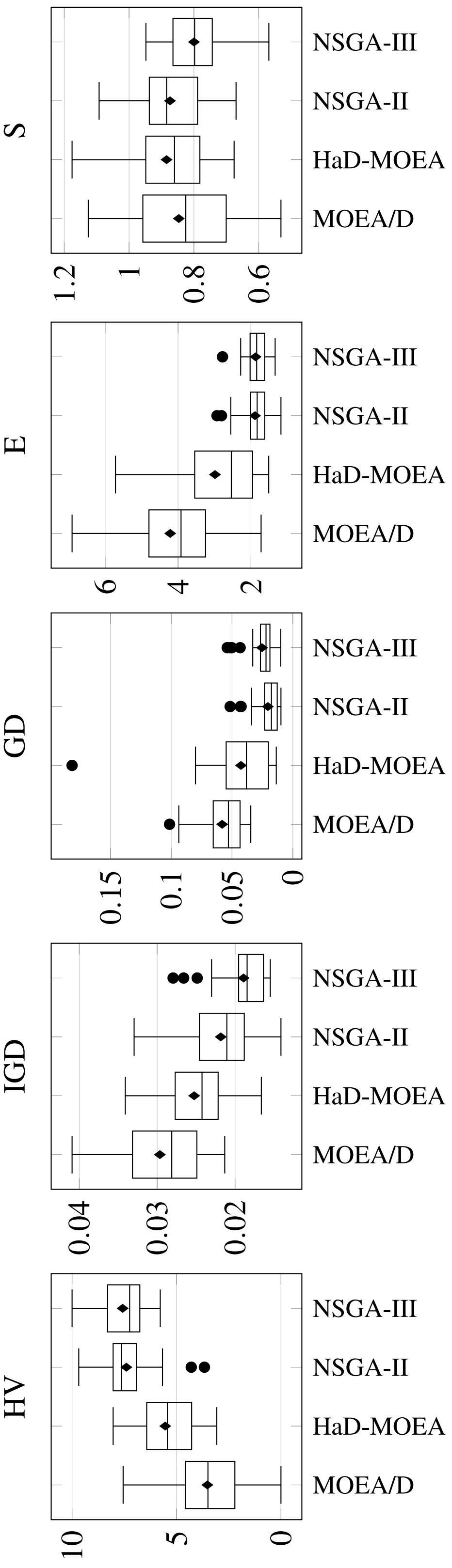
(a) Results obtained when optimising the (C_{max}, Cw, Twt) criteria combination



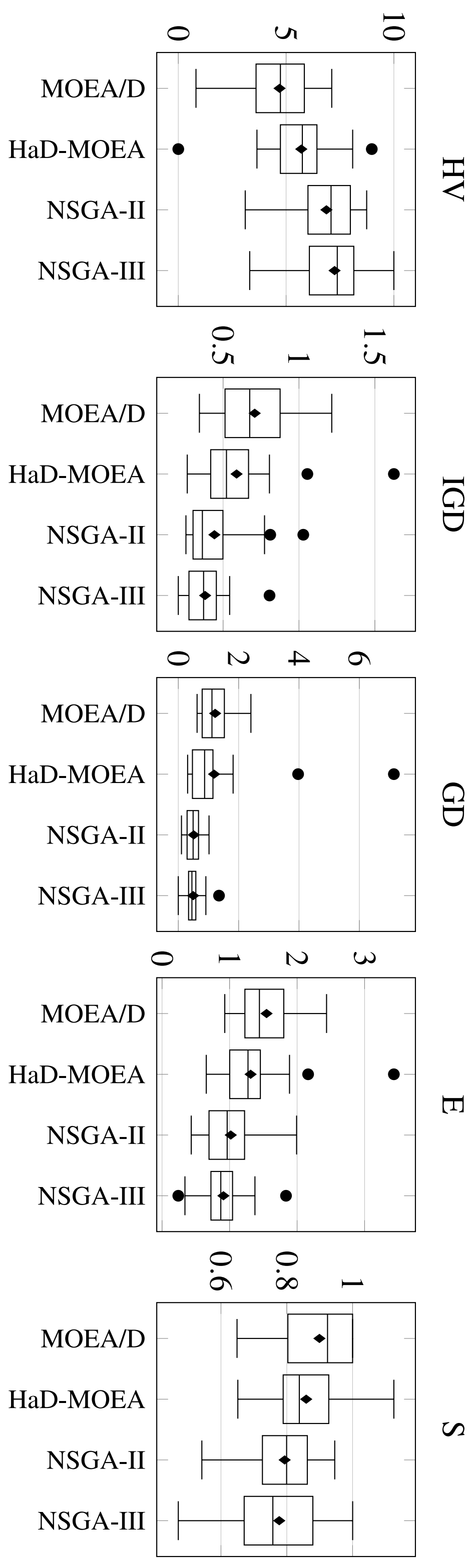
(b) Results obtained when optimising the (C_{max}, F_{max}, Ft) criteria combination



(c) Results obtained when optimising the (C_{max}, F_{max}, T_{max}) criteria combination



(d) Results obtained when optimising the (Cw, Ft, Twt) criteria combination

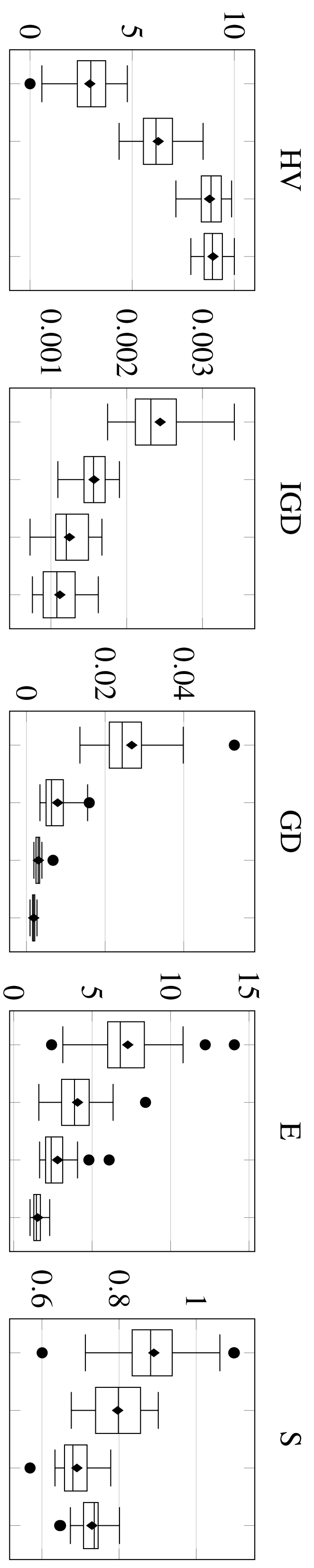


(e) Results obtained when optimising the (Nwt, T_{max}, Twt) criteria combination

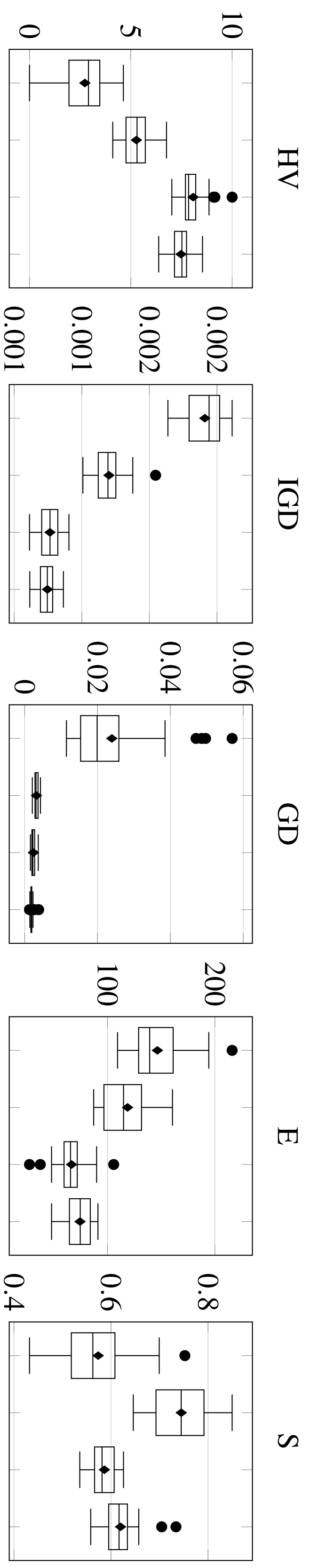
Figure 5.8: Box plot representation of multi-objective metrics obtained when optimising three objectives simultaneously

Table 5.11: Multi-objective metric values obtained when optimising five objectives simultaneously

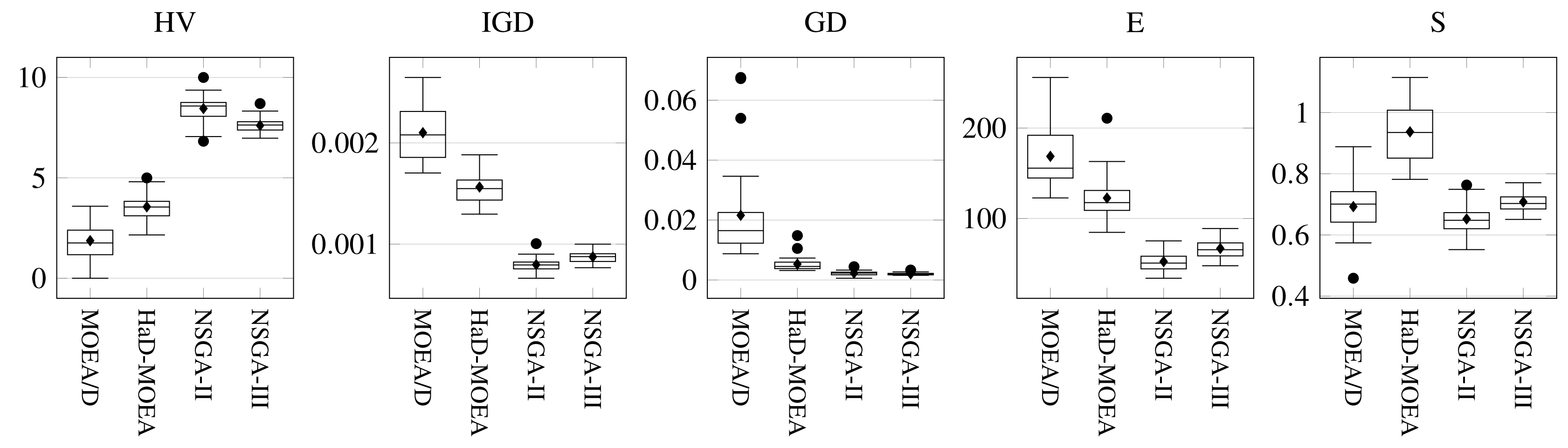
Metrics	Algorithm			
	MOEA/D	HaD-MOEA	NSGA-II	NSGA-III
		$C_{max}, C_w, F_{max}, Ft, M_{ut}$		
HV	2.926	6.274	8.787	**8.958**
IGD	0.244	0.157	0.124	**0.112**
GD	2.677	0.794	0.304	**0.186**
ND	**36.7**	8.1	18.8	20.2
E	7.274	4.073	2.797	**1.529**
S	0.890	0.796	**0.691**	0.730
		$C_{max}, Etwt, Ft, M_{ut}, Twt$		
HV	2.733	5.276	**8.078**	7.478
IGD	0.191	0.120	0.077	**0.075**
GD	2.394	0.325	0.240	**0.191**
ND	**72.6**	35.7	40.9	47.5
E	146.5	118.7	**66.53**	74.40
S	**0.574**	0.745	0.586	0.620
		$Cw, Etwt, Ft, Nwt, Twt$		
HV	1.871	3.558	**8.453**	7.61
IGD	0.210	0.156	**0.080**	0.087
GD	2.159	0.531	0.228	**0.204**
ND	**65.6**	24	52.3	52.2
E	168.7	122.6	**52.29**	66.79
S	0.693	0.937	**0.652**	0.708
		$F_{max}, Ft, Nwt, T_{max}, Twt$		
HV	2.54	4.599	7.582	**7.735**
IGD	1.018	0.790	0.524	**0.495**
GD	6.113	2.612	1.055	**0.959**
ND	**27.9**	4.4	17.5	11.7
E	2.342	1.686	1.122	**1.056**
S	0.654	0.572	**0.510**	0.547



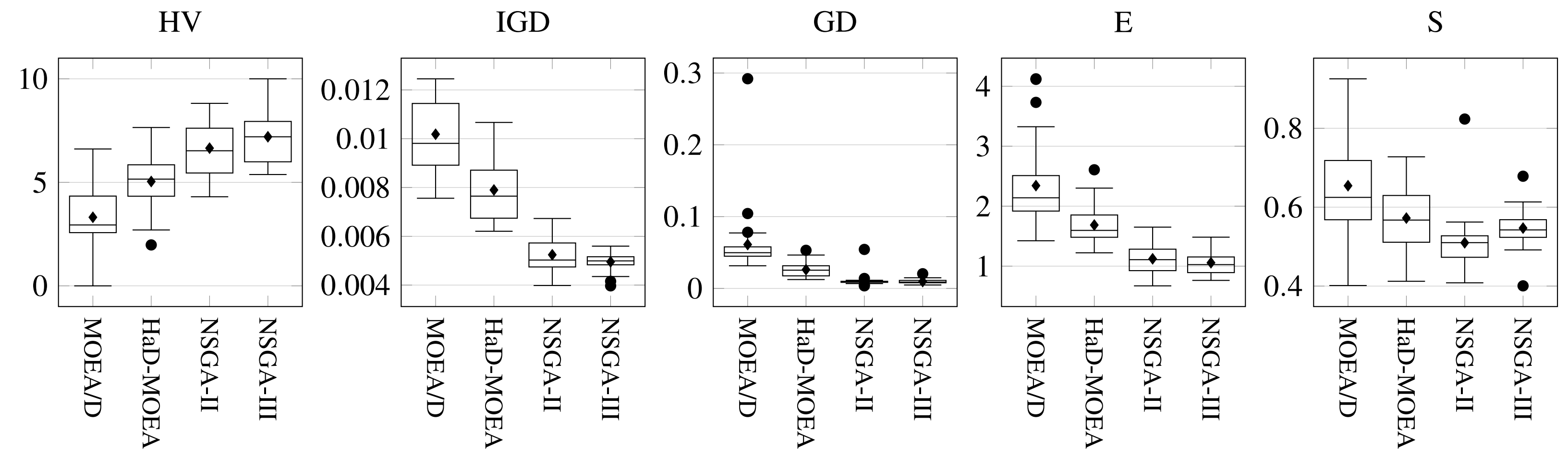
(a) Results obtained when optimising the (C_{max}, C_w, F_{max}, Ft, M_{ut}) criteria combination



(b) Results obtained when optimising the (C_{max}, $Etwt$, Ft, M_{ut}, Twt) criteria combination



(c) Results obtained when optimising the (Cw, $Etwt$, Ft, Nwt, Twt) criteria combination



(d) Results obtained when optimising the (F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination

Figure 5.9: Box plot representation of multi-objective metrics obtained when optimising five objectives simultaneously

Table 5.12: Multi-objective metric values obtained when optimising six objectives simultaneously

Metrics	Algorithm			
	MOEA/D	HaD-MOEA	NSGA-II	NSGA-III
	$C_{max}, F_{max}, Ft, Nwt, T_{max}, Twt$			
HV	6.449	7.579	**8.945**	8.915
IGD	4.839	3.601	2.611	**2.264**
GD	2.081	1.050	0.444	**0.409**
ND	**35.5**	6.4	20.7	17.8
E	2.330	1.925	1.448	**1.387**
S	0.661	0.565	0.499	**0.495**

it can be seen that the NSGA-III algorithm performs well on problems which do not include criteria with a high negative correlation. Therefore NSGA-III performs quite well on the (F_{max}, Ft, Nwt, T_{max}, Twt) and (C_{max}, C_w, F_{max}, Ft, M_{ut}) criteria combinations. However, for the other two criteria combinations which include a criterion that is negatively correlated with the others, the performance of NSGA-III is inferior to that of NSGA-II. It seems that NSGA-III performs better on problems where the Pareto front is not highly dispersed and where the algorithm can focus more on convergence than on diversity.

Table 5.12 represents the metric values achieved when simultaneously optimising six criteria. Since the values for the IGD and GD metrics were quite small, for an easier representation in the table their values were multiplied by 1000 and 100, respectively. The results show that NSGA-III and NSGA-II perform similarly for all the metrics, but NSGA-III achieves better results for all metrics, except for the HV and ND metrics. Since the value of the ND metric is not too large for all algorithms, it can be presumed that the optimised criteria are not correlated negatively to a great extent.

Figure 5.10 represents the box plot representation of the multi-objective metrics when six criteria are optimised simultaneously. The figure shows that NSGA-II and NSGA-III consistently achieve superior results for all multi-objective metrics when compared to the other two MOGP algorithms. By comparing these two algorithms between themselves, it is evident that they achieve a similar performance. This is also backed up by the statistical tests performed for the HV and IGD metrics. The statistical test show that for both metrics the NSGA-II and NSGA-III algorithms perform significantly better than the other two algorithms, however, no significant difference exists between those two algorithms.

Table 5.13 represents the results for the multi-objective metrics which are achieved by op-

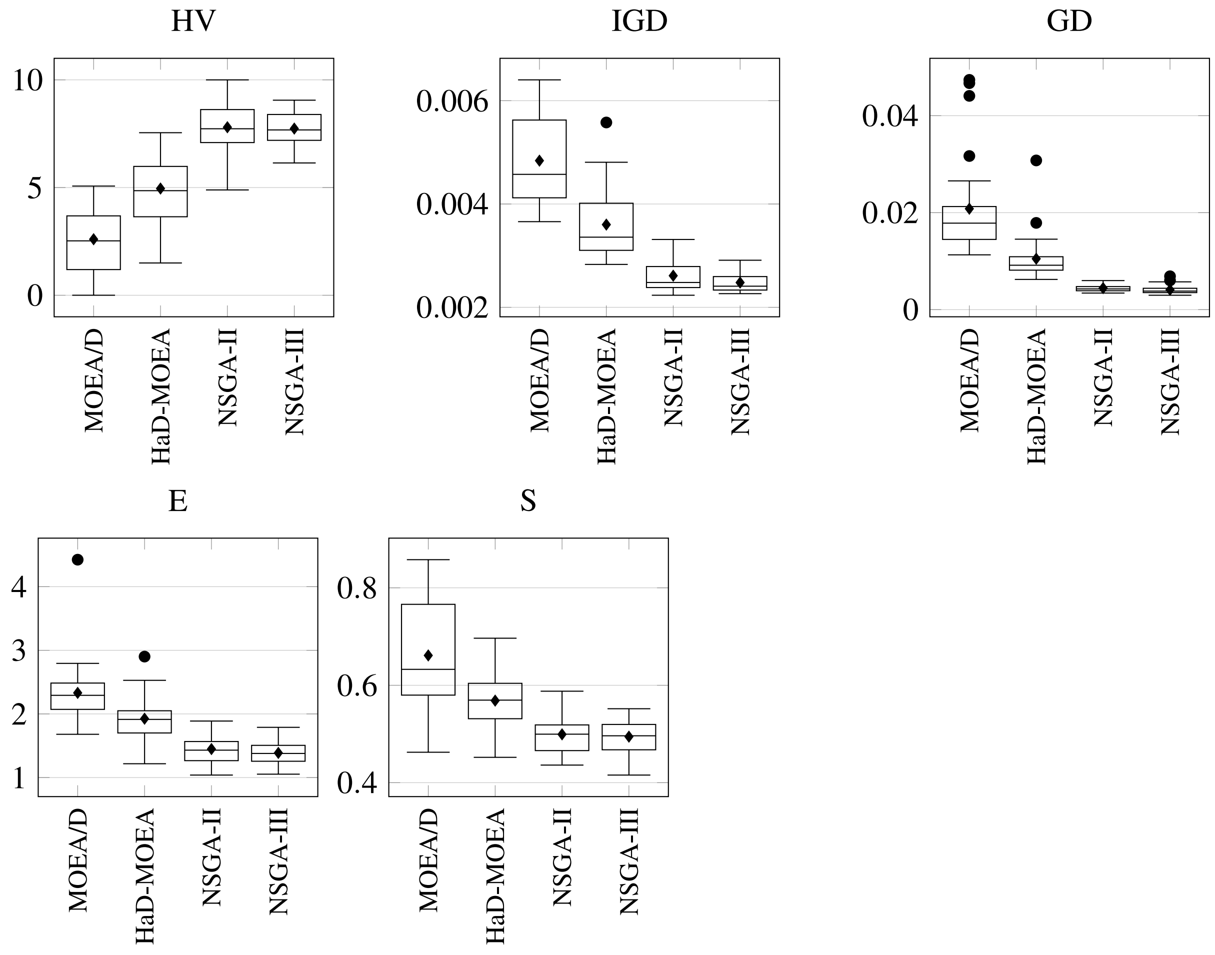


Figure 5.10: Box plot representation of multi-objective metrics obtained when optimising six objectives simultaneously

timising seven criteria simultaneously. Once again the GD and IGD values were multiplied by 100 to represent them easier in the table. The results denote that for this number of criteria the NSGA-II algorithm outperforms the results achieved by NSGA-III. Only for the (C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination the NSGA-III algorithm achieved better HV and GD values than NSGA-II. However, NSGA-II still achieved better performance for the other three metrics. When optimising the (C_{max}, $Etwt$, F_{max}, Ft, M_{ut}, Nwt, T_{max}) criteria combination the algorithms achieve a much higher value for the percentage of nondominated solutions than for the other criteria, which means that more solutions are needed to approximate the Pareto front. The reason for this is due to the fact that both the $Etwt$ and M_{ut} criteria are included, which drastically increases the size of the Pareto front.

Figure 5.11 shows the box plot representation of the multi-objective metrics when optimising seven criteria simultaneously. Even for these criteria combinations NSGA-II and NSGA-III perform better than the other two MOGP algorithms. However, when optimising seven criteria the NSGA-II algorithm outperforms NSGA-III for most of the metrics. For the HV metric the NSGA-II algorithm achieved significantly better results than any other algorithm for the (C_{max}, Cw, F_{max}, Ft, M_{ut}, T_{max}, Twt) and (C_{max}, $Etwt$, F_{max}, Ft, M_{ut}, Nwt, T_{max}) criteria combinations, while for the (C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt) combination there was no significant difference between it and NSGA-III. For the IGD metric, NSGA-II achieved significantly better results than any other algorithm, except for the (C_{max}, Cw, F_{max}, Ft, M_{ut}, T_{max}, Twt) criteria combination, where it did not achieve significantly better results than NSGA-III. Therefore, the NSGA-II algorithm performs better on problems which include criteria that are highly negatively correlated, whereas if those criteria are not optimised, the NSGA-III algorithm performs equally well as NSGA-II.

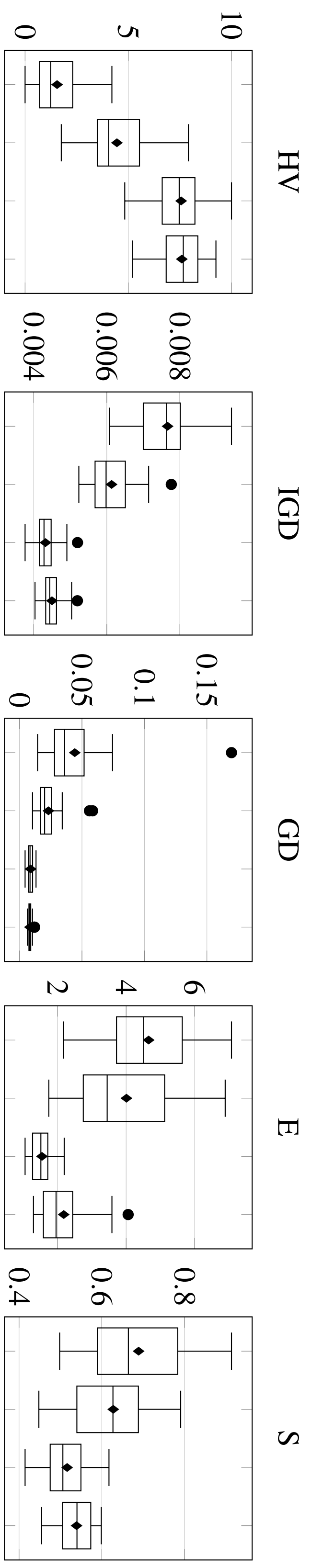
Table 5.14 represents the results achieved for the multi-objective metrics when optimising all nine scheduling criteria simultaneously. The results show that the NSGA-III algorithm outperforms the NSGA-II algorithm for all metrics except the HV and ND metrics. It seems that this number of criteria was simply too large for the NSGA-II algorithm to perform well on it, whereas the NSGA-III algorithm performs well even when optimising such a large number of criteria. For this criteria combination the percentage of nondominated solutions obtained by all algorithms is usually above 50%, which means that a large number of solutions was required to obtain a good Pareto front.

Figure 5.12 shows the box plot representation of the achieved multi-objective metric values for optimising nine criteria simultaneously. For this criteria combination the NSGA-III algorithm achieves significantly better results than the other algorithms, for both the HV and IGD metrics. Therefore, NSGA-III obtains the most diverse Pareto front out of all the algorithms for this criteria combination.

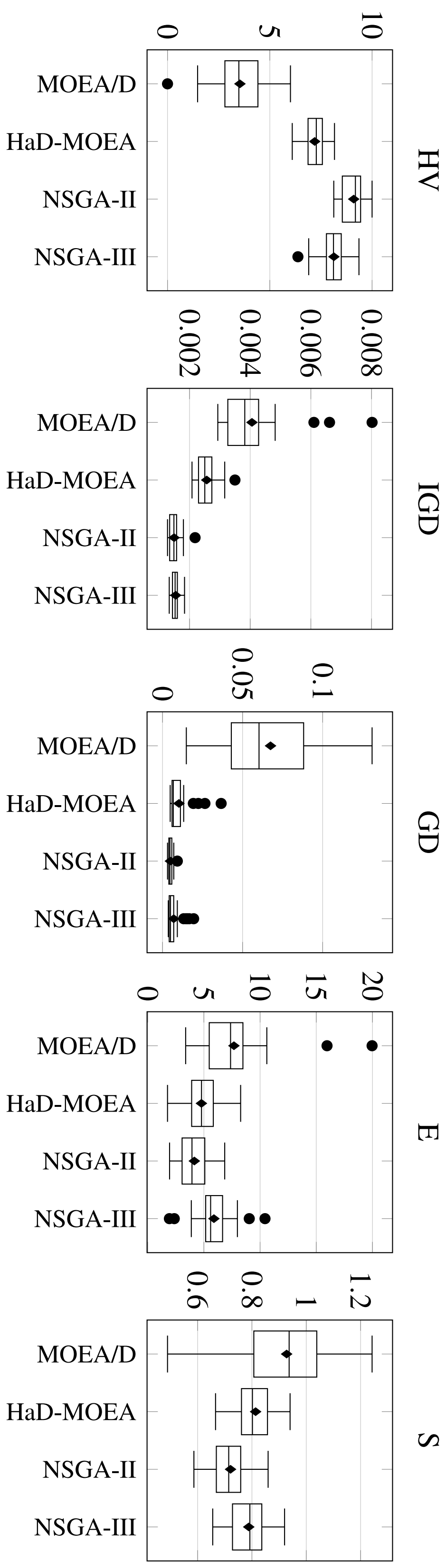
Through the experiments it was shown that no single algorithm achieved the best results

Table 5.13: Multi-objective metric values obtained when optimising seven objectives simultaneously

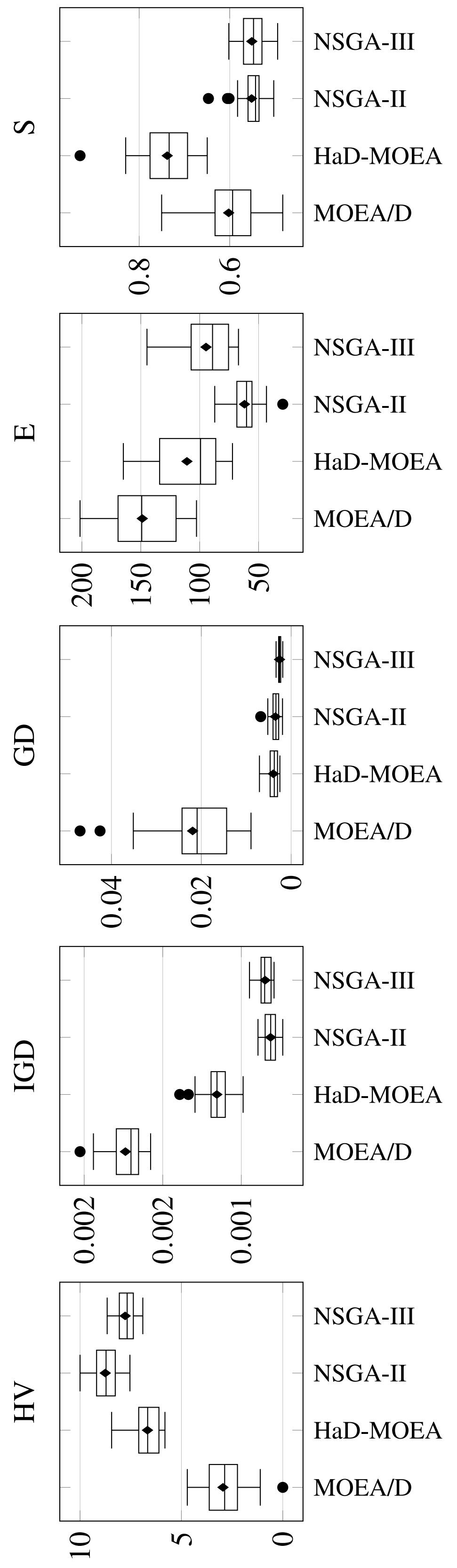
Metrics	Algorithm			
	MOEA/D	HaD-MOEA	NSGA-II	NSGA-III
	$C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt$			
HV	1.55	4.449	7.561	**7.592**
IGD	0.767	0.613	**0.432**	0.450
GD	3.612	2.316	0.890	**0.841**
ND	**43.1**	7.7	23.7	24.3
E	4.656	4.009	**1.543**	2.178
S	0.689	0.628	**0.516**	0.539
	$C_{max}, Cw, F_{max}, Ft, M_{ut}, T_{max}, Twt$			
HV	3.541	7.196	**9.1**	8.1
IGD	0.406	0.257	**0.149**	0.154
GD	6.744	1.026	**0.491**	0.700
ND	**39.5**	13.2	24.9	24.8
E	7.674	4.780	**4.151**	5.893
S	0.928	0.814	**0.721**	0.788
	$C_{max}, Etwt, F_{max}, Ft, M_{ut}, Nwt, T_{max}$			
HV	2.949	6.674	**8.756**	7.773
IGD	0.174	0.116	**0.081**	0.085
GD	2.193	0.397	0.350	**0.258**
ND	**77.9**	44.2	59.5	66.8
E	148.7	110.8	**61.98**	94.64
S	0.602	0.738	0.552	**0.551**



(a) Results obtained when optimising the (C_{max}, Cw, F_{max}, Ft, Nwt, T_{max}, Twt) criteria combination



(b) Results obtained when optimising the (C_{max}, Cw, F_{max}, Ft, M_{ut}, T_{max}, Twt) criteria combination



(c) Results obtained when optimising the (C_{max}, $Etwt$, F_{max}, Ft, M_{ut}, Nwt, T_{max}) criteria combination

Figure 5.11: Box plot representation of multi-objective metrics obtained when optimising seven objectives simultaneously

Table 5.14: Multi-objective metric values obtained when optimising nine objectives simultaneously

Metrics	Algorithm			
	MOEA/D	HaD-MOEA	NSGA-II	NSGA-III
HV	2.565	6.497	7.872	**8.179**
IGD	0.157	0.104	0.077	**0.068**
GD	1.492	0.365	0.431	**0.196**
ND	**81.3**	44.9	58.2	63.3
E	157.2	97.73	**73.65**	76.79
S	0.616	0.736	0.643	**0.564**

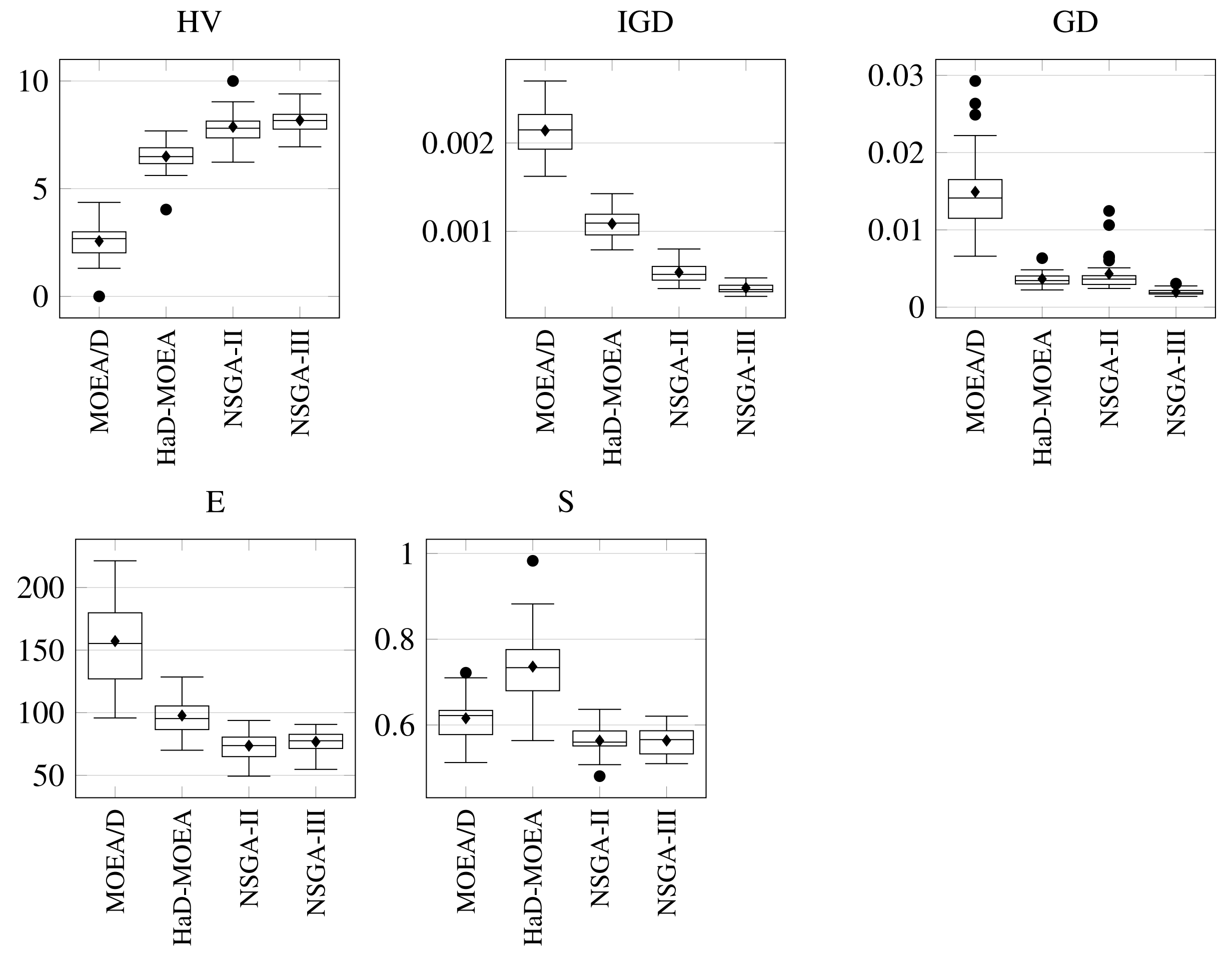


Figure 5.12: Box plot representation of multi-objective metrics obtained when optimising nine objectives simultaneously

over all of the tested criteria combinations. For the smallest combination sizes of three criteria, the best results were usually achieved by the NSGA-III algorithm, which can be seen not only from the HV and IGD values, but also through the fact that it achieved the best values for most of the optimised criteria. However, as the number of criteria grew, it is possible to observe that the NSGA-III starts to encounter difficulties, as it was sometimes outperformed by NSGA-II. For optimising five criteria, NSGA-III still achieves better results than NSGA-II for most of the criteria (except for the combination where both the $Etwt$ and M_{ut} criteria are included). However, based on the HV and IGD metrics, there was mostly no difference between the two algorithms. When optimising the six criteria combination, where the $Etwt$ and M_{ut} criteria were not included, the NSGA-III algorithm achieved the best values for the most of the multi-objective metrics. NSGA-II also achieved good performance, without there being a statistically significant difference between the results achieved by it and NSGA-III for the HV and IGD metrics. Similar behaviour as when optimising five criteria is also evident when seven criteria are optimised, since NSGA-III achieved worse results than NSGA-II for most of the objectives, especially those which include the $Etwt$ and M_{ut} criteria. For the HV and IGD metrics the NSGA-II algorithm even outperformed NSGA-III for two out of the three tested criteria combinations. In the case of nine criteria, NSGA-III once again outperformed NSGA-II.

Such behaviour is quite surprising, since it would be expected that the NSGA-III algorithm performs better than NSGA-II for larger criteria combinations. The reason for this seems to originate from the way in which the solutions are selected into the next generation by the two algorithms. The reference point selection mechanism present in NSGA-III seems to favour convergence more than it does diversification. This behaviour is probably caused by the fact that more reference points will be present at the centre of the Pareto front, and therefore the algorithm will simply select more solutions which are close to the centre. On the other hand, NSGA-II will select solutions in areas that are less populated, which will most likely be on the edges of the Pareto fronts. As a consequence, NSGA-III will perform well on smaller criteria combinations, where the Pareto front is not large, and therefore the algorithm will easily converge to good solutions. In addition, NSGA-III also handles larger criteria combinations well, as long as the criteria which are included in those combinations are not negatively correlated, where optimising one criterion would lead to bad values for other criteria (such as with the inclusion of $Etwt$ and M_{ut}), since in such a situation the Pareto front becomes large, and there is more need for diversification. The solution distributions also back-up these observations, since for several criteria NSGA-III obtains less distributed solutions than NSGA-II, like for (C_{max}, C_w, M_{ut}, F_{max}, Ft), (Cw, $Etwt$, Ft, Nwt, Twt), (C_{max}, $Etwt$, F_{max}, Ft, M_{ut}, Nwt, T_{max}), and when optimising all nine criteria simultaneously. However, this is not the first time that NSGA-II outperforms NSGA-III for a large number of criteria. Similar observations were also noticed in papers which dealt with comparison of many-objective algorithms on different types of problems [261, 262]. Those papers show that NSGA-III performed quite poor for several different types of problems, and that NSGA-II outperformed NSGA-III when optimising larger sets of criteria. The papers also suggest that the recently proposed many-objective algorithms (such as MOEA/D and NSGA-III) use selection mechanisms which are well suited for standard problems that are used to test the performance of many-objective algorithms. However, when tested on other types of problems the algorithms can exhibit performance which is inferior to that of NSGA-II.

HaD-MOEA achieved results which are mostly inferior to the results of NSGA-II and NSGA-III. This algorithm works in a similar fashion as NSGA-II, with the only difference being the way in which the crowding distance is calculated. In HaD-MOEA the crowding distance is calculated as the harmonic distance of the n closest neighbours of the current solution. By using such a crowding distance measure, HaD-MOEA achieved a very small percentage of nondominated solutions in each run. In addition, when optimising criteria combinations that include the $Etwt$ and M_{ut} criteria, it is obvious, from the solution distributions, that HaD-MOEA tends to focus mostly on the other criteria, and thus achieved quite bad solution distributions for the $Etwt$ and M_{ut} criteria. The cause for this problem is probably due to the fact that the closest neighbours are used to calculate the crowding distance. This crowding distance can perform poorly if small groups of solutions exist, which are far apart from each other in the search space. Since the crowding distance is calculated only on the nearest neighbours of a solution, it is possible that all solutions will have a similar value of the crowding distance, and therefore all the solutions could be deleted even if they are far away from all other solutions. This can have a negative effect on diversity, which is especially important when optimising criteria combinations with large Pareto fronts.

The MOEA/D algorithm achieved quite a poor performance for most of the optimised criteria. The reasons for such behaviour are probably twofold. The first reason is the same as for NSGA-III, meaning that the way the algorithm was designed is not appropriate for optimising the tested criteria combinations. The second reason is connected to the way in which the parameters were optimised, since the generic parameters were optimised first, after which the algorithm specific parameters were optimised. It is possible that the combination method which is used in MOEA/D has a great influence on all other generic parameters, and therefore the optimal parameters for that combination method could largely differ from the ones which were determined in the first optimisation steps. Since MOEA/D also uses a quite small population compared to the other algorithms, it will be much more difficult for it to obtain a good set of solutions, especially for problems with large Pareto fronts.

Based on the results it can be concluded that there is much difference in the performance of the different MOGP algorithms. Because of its strong convergence ability, NSGA-III performed well when a smaller number of criteria were optimised simultaneously, and for larger combinations in which there are no criteria which have a strong negative correlation with other criteria (like $Etwt$ and M_{ut}). Since out of all the tested algorithms NSGA-II produced the most diverse solutions and is therefore better able to cover the Pareto front, it was the most appropriate method for handling combinations which consisted out of negatively correlated criteria. Although the other two MOGP algorithms perform well on certain objective combinations, they were in most cases inferior to NSGA-II and NSGA-III.

5.5 Comparison with standard DRs

In the last section it was shown that the MOGP algorithms generated DRs which can in most cases outperform those obtained by SOGP. However, based on those results all alone it is not possible to determine the performance of individual DRs on several criteria simultaneously. For that purpose, several automatically generated DRs by using MOGP algorithms will be selected and compared with manually designed DRs. In this way it will be possible to determine if the automatically designed DRs can outperform manually designed DRs not only for one criterion, but rather for several different criteria simultaneously. The following five manual DRs will be used for the comparison: MCT, ATC, RC, COVERT and sufferage. MCT was selected since it represents one of the simplest DRs, but nevertheless achieves the best result for the F_{max} criterion. On the other hand, ATC was selected since it represents the rule which can achieve the best results for the due date related criteria. The COVERT and RC rules were selected because they achieve good results on several criteria simultaneously, and therefore represent the best manually designed multi-objective DRs. Finally, the sufferage rule was selected since it achieves a good value for the makespan and flowtime criteria.

The results will be presented in tables where the values achieved by the selected standard DR will be presented at the top, while the rest of the table will include results achieved by automatically designed DRs evolved for different criteria combinations. For each criteria combination, automatically generated DRs that achieved better results than the considered standard DR for most of the criteria were selected. If more such DRs exist, then one of them is selected randomly. For automatically designed DRs the table includes results only for those criteria for which the considered DR was optimised, while the values for the other criteria are denoted with "-". Each criterion value for which the automatically generated DRs achieved better results than standard DRs, is denoted in bold.

5.5.1 Comparison of automatically generated DRs with MCT

Table 5.15 represents the results of automatically generated DRs when compared to the MCT rule. The results denote that when optimising six criteria or less, the automatically generated rules achieve better values than the MCT rule on all the tested criteria. When optimising seven criteria simultaneously, only for one out of the three tested criteria combination the automatically generated DRs did not achieve a better performance than the MCT rule for each individual criterion. For that one criteria combination the automatically generated DRs were unable to achieve a better result only for the *Etwt* criterion. Even the DR which was evolved for optimising nine criteria simultaneously outperformed the MCT rule for eight criteria, only failing to surpass the MCT rule for the M_{ut} criterion. The strength of the automatically generated DRs is even further signified by the fact that for each criteria combination which included the F_{max} criterion, it was possible to achieve better values for that criterion than by using the MCT rule. Therefore, such simple manual DRs do not seem to be competitive to automatically generated DRs, which consistently outperformed the MCT rule, even for the criterion for which the MCT rule achieved the best result.

5.5.2 Comparison of automatically generated DRs with ATC

Table 5.16 represents the comparison of the results achieved by the automatically generated DRs with the ATC rule. When three criteria are optimised the evolved DRs usually outperform the ATC rule for two or three criteria. Although the rules *R1* and *R2* were evolved for the same criteria combination, rule *R1* could not outperform ATC for the *Twt* criterion, while rule *R2* could not outperform ATC for the C_{max} criterion. Therefore, even though the MOGP algorithms were unable to generate a rule which can perform better than the ATC rule for all three criteria, it is possible to select a DR which achieves the required trade off between the various optimised criteria, since the MOGP algorithms evolve a Pareto front of solutions. The results obtained for rule *R6* are especially interesting, since this rule was evolved only on the three due date related criteria. For all three due date related criteria the rule achieves a better performance than the ATC rule. This rule achieves an increase in performance over the ATC rule of 1.7%, 7%, and 7.7% for the *Nwt*, T_{max}, and *Twt* criteria, respectively. Therefore, by using the MOGP algorithms it was possible to develop a DR which performs better than the best manually designed DR for the due date related criteria.

When five and six criteria are optimised simultaneously, the generated DRs usually outperform the ATC rule for four or five criteria. Rules like *R7* were easily able to outperform the ATC rule, since they did not optimise any due date related criteria. However, it is interesting to observe that for the (C_{max}, *Etwt*, *Ft*, M_{ut}, *Twt*) criteria combination neither of the MOGP algorithms generated a DR which would achieve a good performance for the *Twt* criterion, while also achieving good results for the other criteria as well. The *R8* rule achieved poor results for the *Twt* criterion, although it performed well on all the other criteria it was optimised for. The other rules achieved better performance than the ATC rule for all criteria, except for the F_{max} criterion. It seems that when more due date related criteria are included in the set of optimised

Table 5.15: Comparison of automatically generated multi-objective DRs with the MCT rule

Method	Criteria								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
MCT	38.57	902.3	977.2	14.03	181.1	0.130	8.007	2.891	18.88
Evolved DRs - three objectives									
R1	**38.16**	**875.3**	-	-	-	-	-	-	**14.50**
R2	**37.91**	-	-	**13.59**	**177.2**	-	-	-	-
R3	**38.05**	-	-	**13.59**	-	-	-	**2.760**	-
R4	-	**874.0**	-	-	**154.0**	-	-	-	**14.73**
R5	-	-	-	-	-	-	**6.566**	**2.249**	**12.28**
Evolved DRs - five objectives									
R6	**38.15**	**897.9**	-	**13.74**	**178.3**	**0.126**	-	-	-
R7	**38.07**	-	**976.0**	-	**175.3**	**0.125**	-	-	**14.65**
R8	-	**879.1**	**973.8**	-	**170.6**	-	**6.968**	-	**14.89**
R9	-	-	-	**13.61**	**175.0**	-	**7.248**	**2.690**	**17.17**
Evolved DRs - six objectives									
R10	**37.89**	-	-	**13.55**	**170.1**	-	**7.265**	**2.656**	**17.32**
Evolved DRs - seven objectives									
R11	**38.08**	**891.3**	-	**13.62**	**170.7**	-	**7.047**	**2.787**	**17.19**
R12	**38.33**	**897.7**	-	**13.79**	**177.4**	**0.123**	-	**2.844**	**18.53**
R13	**38.15**	-	980.5	**13.86**	**178.5**	**0.124**	**7.322**	**2.793**	-
Evolved DRs - nine objectives									
R14	**38.47**	**900.3**	**976.6**	**13.89**	**180.4**	0.139	**7.701**	**2.843**	**18.54**

Table 5.16: Comparison of automatically generated multi-objective DRs with the ATC rule

Method	Criteria								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
ATC	38.26	901.5	968.5	14.27	195.9	0.127	6.686	2.418	13.30
Evolved DRs - three objectives									
R1	**38.21**	**887.2**	-	-	-	-	-	-	13.36
R2	38.62	**869.9**	-	-	-	-	-	-	**12.70**
R3	**37.89**	-	-	**13.61**	**175.5**	-	-	-	-
R4	**38.17**	-	-	15.41	-	-	-	**2.383**	-
R5	-	**893.4**	-	-	**173.5**	-	-	-	**12.79**
R6	-	-	-	-	-	-	**6.566**	**2.249**	**12.28**
Evolved DRs - five objectives									
R7	**38.11**	**882.5**	-	**14.17**	**161.9**	**0.126**	-	-	-
R8	**38.10**	-	**963.0**	-	**179.3**	**0.122**	-	-	19.43
R9	-	**900.5**	**968.5**	-	**179.3**	-	**6.333**	-	**13.00**
R10	-	-	-	17.00	**173.5**	-	**6.440**	**2.401**	**12.68**
Evolved DRs - six objectives									
R11	**38.22**	-	-	16.45	**178.1**	-	**6.306**	**2.410**	**12.85**
Evolved DRs - seven objectives									
R12	**38.08**	903.1	-	15.22	**182.9**	-	**6.650**	**2.390**	**13.22**
R13	39.19	**894.3**	-	17.49	**173.0**	0.145	-	**2.409**	**13.08**
R14	38.41	-	**968.0**	16.22	**177.0**	**0.125**	**6.651**	2.646	-
Evolved DRs - nine objectives									
R15	39.31	904.0	**967.9**	17.65	**182.5**	0.145	**6.566**	2.477	**13.08**

criteria, it will be easier for the MOGP algorithms to evolve DRs that perform well on them.

For the case when seven and nine criteria are optimised, the DRs usually outperform the ATC rule only on four or five criteria. Rules *R12* and *R13* outperform the ATC rule for all the due date related criteria they were optimised on, which demonstrates the effectiveness for generating good DRs even when larger criteria combinations are optimised. The remaining two rules were unable to outperform ATC for all due date related criteria, and generally outperform the ATC rule only for four scheduling criteria.

5.5.3 Comparison of automatically generated DRs with RC

Table 5.17 represents the results of the comparison between the automatically generated DRs and the RC rule. The RC rule is an example of a rule which achieves the best results for the makespan and flowtime related criteria, but also performs well on other criteria as well. In addition, this rule obtains the best result for the Ft criterion out of all the manually designed DRs.

When three criteria are optimised, the automatically designed DRs usually outperform the RC rule for two or three criteria. It can be observed that the problems mostly arise if the Cw or Ft criteria are optimised with either the C_{max} or F_{max} criteria, in which case the evolved DRs were able to perform better than the RC rule on only one of those two groups of criteria, but not on both. Rules *R1* and *R2* were optimised on such criteria combinations, and therefore outperform the RC rule for at most two criteria. However, rules *R3* and *R4* demonstrate that if only one of those groups of criteria is optimised, the evolved DRs can outperform the RC rule for all optimised criteria. The *R5* rule also shows that it is easy to obtain a DR which outperforms the RC rule when optimising only the due date related criteria.

A similar situation can be also observed when optimising five criteria simultaneously. Rules like *R6*, *R7*, and *R10* focus more on optimising the C_{max} and F_{max} criteria. When neither of those two criteria are included in the optimisation set, like for rules *R8* and *R9* the DRs achieved better results for the Cw and Ft criteria. However, for neither of these two rules was it possible to achieve a better value for the Ft criterion than the one achieved by the RC rule. Therefore it seems difficult for the MOGP algorithms to generate a rule which can not only achieve a good performance for the Ft criterion, but that can perform well on other criteria as well. When six criteria are optimised simultaneously the MOGP algorithms evolved DRs which performed better than the RC rule for all criteria except the Ft criterion.

The same behaviour is observed even for the larger criteria combinations. The DRs usually performed better for the C_{max} and F_{max} criteria and for most of the due date related criteria. Rules *R12*, *R13* and *R15* are examples of such DRs. The generated DRs usually achieved better values than the RC rule for five criteria. The MOGP algorithms evolved DRs which perform better than the RC rule for the Ft and Cw criteria even when a larger number of objectives were

Table 5.17: Comparison of automatically generated multi-objective DRs with the RC rule

Method	Criteria								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
RC	38.11	874.9	998.3	14.91	154.1	0.128	6.786	2.863	15.40
Evolved DRs - three objectives									
R1	38.47	**874.0**	-	-	-	-	-	-	**14.30**
R2	**37.89**	-	-	**13.61**	175.5	-	-	-	-
R3	**37.97**	-	-	**13.62**	-	-	-	**2.754**	-
R4	-	**872.4**	-	-	**153.8**	-	-	-	**14.93**
R5	-	-	-	-	-	-	**6.566**	**2.249**	**12.28**
Evolved DRs - five objectives									
R6	**38.06**	899.4	-	**13.96**	177.7	**0.127**	-	-	-
R7	**38.07**		**976.0**	-	175.3	**0.125**	-	-	**14.65**
R8	-	**874.1**	**988.4**	-	163.9	-	7.361	-	**15.23**
R9	-	877.5	**992.5**	-	157.1	-	**6.734**	-	**14.46**
R10	-	-	-	**14.68**	159.0	-	**6.743**	**2.683**	**14.53**
Evolved DRs - six objectives									
R11	**38.04**	-	-	**14.64**	172.1	-	**6.667**	**2.379**	**13.51**
Evolved DRs - seven objectives									
R12	**37.98**	890.0	-	**14.20**	168.0	-	**6.762**	**2.562**	**14.63**
R13	**38.03**	898.5	-	**14.90**	175.8	**0.128**	-	**2.638**	**14.95**
R14	38.41	-	**968.0**	16.22	177.0	**0.125**	**6.651**	**2.646**	-
Evolved DRs - nine objectives									
R15	**38.08**	900.1	**972.2**	**14.74**	179.4	0.142	6.984	**2.511**	**14.59**

optimised, however those rules achieved inferior performance than the RC rule for all the other criteria.

5.5.4 Comparison of automatically generated DRs with COVERT

Table 5.18 compares the results achieved by the generated DRs with the COVERT DR. COVERT is another rule which achieves good performance on several criteria. This rule performs well on the due date related criteria, as well as for the C_{max} and F_{max} criteria. However, unlike the RC rule, the COVERT rule does not obtain an overall best solution out of all the tested standard DRs on either of the scheduling criteria. When three criteria are optimised, the MOGP algorithms generated DRs which, for all the tested criteria combinations, outperform the COVERT rule on all the optimised criteria. The achieved improvements over the COVERT rule depend on the criteria combination which was optimised. For example, rule *R1* achieves small improvements over the COVERT rule for all three optimised criteria, while the *R2* rule significantly outperforms the COVERT rule for the F_{max} and Ft criteria. Therefore, when a small number of criteria is considered, the MOGP algorithms can easily generate DRs which outperform the COVERT rule for various criteria combinations.

The good performance of the automatically generated DRs can be observed even when a larger number of criteria are optimised. Rules *R6* and *R8* can even outperform the COVERT rule for all of the optimised objectives, even though different criteria types are optimised together. On the other hand, rules *R7* and *R9* outperform the COVERT rule for four out of the five optimised criteria. Even when optimising six criteria, the evolved DRs can outperform the COVERT rule for five criteria. Thus, the generated DRs can be deemed superior than the COVERT rule even when a larger number of criteria is optimised simultaneously.

However, when seven criteria are optimised simultaneously, the performance of the generated DRs depends more on the combination of criteria which is optimised. Thus, rule *R11* achieved better results than the COVERT rule for all criteria except the F_{max} and Nwt criteria. As the M_{ut} and $Etwt$ criteria are introduced in the optimisation set, the number of criteria for which the generated DRs achieve better results than the COVERT rule decreases. When all nine criteria are optimised simultaneously, the generated rule outperforms the COVERT rule for five objectives. Therefore, it seems that when a larger number of criteria is optimised simultaneously, better results over the COVERT rule can be achieved when the M_{ut} and $Etwt$ are not optimised.

5.5.5 Comparison of automatically generated DRs with the sufferage rule

Table 5.19 shows the comparison of the generated DRs with the sufferage rule. This rule was selected since it achieves the second best performance for the C_{max} criterion, while still achiev-

Table 5.18: Comparison of automatically generated multi-objective DRs with the COVERT rule

Method	Criteria								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
COVERT	38.26	903.1	967.1	14.57	182.3	0.126	6.755	2.442	13.50
Evolved DRs - three objectives									
R1	**38.21**	**887.2**	-	-	-	-	-	-	**13.49**
R2	**37.89**	-	-	**13.61**	**175.5**	-	-	-	-
R3	**38.02**	-	-	**14.37**	-	-	-	**2.427**	-
R4	-	**893.4**	-	-	**173.5**	-	-	-	**12.79**
R5	-	-	-	-	-	-	**6.566**	**2.249**	**12.28**
Evolved DRs - five objectives									
R6	**38.06**	**900.0**	-	**14.01**	**177.7**	**0.126**	-	-	-
R7	**38.10**	-	**963.0**	-	**179.3**	**0.122**	-	-	19.43
R8	-	**899.3**	**967.1**	-	**179.0**	-	**6.686**	-	**13.26**
R9	-	-	-	17.00	**173.5**	-	**6.440**	**2.401**	**12.68**
Evolved DRs - six objectives									
R10	**38.22**	-	-	16.45	**178.1**	-	**6.306**	**2.410**	**12.85**
Evolved DRs - seven objectives									
R11	**38.12**	**886.4**	-	15.71	**164.4**	-	6.776	**2.426**	**13.49**
R12	**38.14**	**896.2**	-	**14.49**	**175.2**	**0.126**	-	2.665	15.15
R13	**38.15**	-	980.5	**13.86**	**178.5**	**0.124**	7.322	2.793	-
Evolved DRs - nine objectives									
R14	38.87	**894.4**	977.0	16.81	**173.3**	0.142	**6.566**	**2.420**	**12.97**

ing good performance on the other criteria as well. When three criteria are optimised, the results demonstrate that the generated DRs have the most problems in outperforming the sufferage rule for the C_{max} criterion. This is expected, since the sufferage rule achieves an extremely good value for that criterion. The *R3* rule managed to slightly outperform the sufferage rule for the C_{max} criterion, however at the expense of achieving slightly worse results than the sufferage rule for the F_{max} criterion. When the C_{max} criterion is not included in the optimisation set, then the MOGP algorithms have no problem of evolving DRs which outperform the sufferage rule for all of the criteria.

A similar situation can be observed even when five and six criteria are optimised simultaneously. For the C_{max} criterion the selected rules are unable to outperform the sufferage rule, however for the other criteria the generated DRs achieved consistently better results. Nevertheless, some rules still achieve good performance for the C_{max} criterion. For example, the *R11* rule, which is evolved for optimising six criteria, achieves only 0.3% worse results than the sufferage rule for the C_{max} criterion, but for the other criteria it can achieve significant improvements over the sufferage rule, like 3.4% for F_{max}, and 12.4% for Twt. Therefore, even if the generated DRs can not outperform the sufferage rule for the C_{max} criterion, in many cases they do not achieve significantly worse results.

When seven and nine criteria are optimised simultaneously, the generated DRs outperform the sufferage rule for five or six criteria. Rule *R12* can be seen to outperform the sufferage rule for six out of seven criteria, and it achieves inferior results only for the C_{max} criterion by only 0.6%. For other criteria, especially the due date related ones, this rules can achieve large improvements over the sufferage rule. Therefore, this rule can be considered as a good alternative to the sufferage rule, since it achieves only slightly worse result for the C_{max} criterion than the sufferage rule. Rules *R13*, *R14*, and *R15* also achieve good results and outperform the sufferage rule for most of the optimised criteria, however, the values which these rules achieve for the C_{max} criterion are worse than that of the *R12* rule.

5.6 Analysis of the correlation between the scheduling criteria

Through the optimisation of different criteria combinations it was shown that for certain combinations the MOGP algorithms achieved better performance than on others. Usually, if criteria which are positively correlated are optimised together, the MOGP algorithms achieved better performance. For that reason it is important to analyse how the different criteria are correlated with each other, so that if possible a good combination of criteria can be selected. The analysis of the correlation of the different criteria was performed in a way that the Pareto fronts for all MOGP algorithms, which were obtained by optimising nine criteria simultaneously, were

Table 5.19: Comparison of automatically generated multi-objective DRs with the sufferage rule

Method	Criteria								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
Sufferage	37.88	881.6	989.6	14.50	165.7	0.128	6.986	2.854	15.94
Evolved DRs - three objectives									
R1	38.16	**875.3**	-	-	-	-	-	-	**14.50**
R2	38.06	-	-	**14.07**	**161.9**	-	-	-	-
R3	**37.86**	-	-	14.55	-	-	-	**2.599**	-
R4	37.97	-	-	**13.62**	-	-	-	**2.754**	-
R5	-	**874.0**	-	-	**154.0**	-	-	-	**14.73**
R6	-	-	-	-	-	-	**6.566**	**2.249**	**12.28**
Evolved DRs - five objectives									
R7	38.14	**879.1**	-	**14.26**	**162.3**	**0.126**	-	-	-
R8	38.61	-	**986.3**	-	**163.9**	**0.128**	-	-	**14.96**
R9	-	**881.5**	**984.7**	-	**163.0**	-	**6.761**	-	**13.92**
R10	-	-	-	**14.25**	**163.9**	-	**6.810**	**2.710**	**15.12**
Evolved DRs - six objectives									
R11	38.01	-	-	**14.01**	**162.3**	-	**6.837**	**2.486**	**13.96**
Evolved DRs - seven objectives									
R12	38.12	**881.6**	-	**14.32**	**160.5**	-	**6.816**	**2.534**	**14.49**
R13	38.32	**879.3**	-	**14.41**	**159.0**	0.130	-	**2.774**	**15.11**
R14	38.46	-	**988.5**	15.77	**161.6**	**0.127**	**6.876**	**2.826**	-
Evolved DRs - nine objectives									
R15	38.62	**881.0**	**988.9**	16.28	**161.0**	0.141	**6.796**	**2.734**	**14.17**

combined into a single Pareto front of nondominated solutions. The combined Pareto front was used to plot all pairwise criteria combinations. In that way it is possible to obtain a notion of how two different criteria influence each other.

Figure 5.13 represents the the correlation of the C_{max} criterion with the other criteria. The figure shows that this criterion correlates the best with the F_{max} criterion. Good correlations are also achieved with both the Ft and Cw criteria. On the other hand, the worst correlations are achieved with the $Etwt$ and M_{ut} criterion, which can be seen from the Pareto fronts, since with the minimisation of C_{max} the values for the other two criteria increase significantly. The C_{max} criterion also does not correlate positively with the due date related criteria, since the increase in the value of C_{max} will usually deteriorate their values. However, they are not as nearly negatively correlated as was the case with the $Etwt$ criterion, where the increase in the value of the C_{max} criterion leads to a great deterioration in the values of the $Etwt$ criterion. Therefore, the increase in the value of the C_{max} criterion will not lead to a severe decline in the values for the due date related criteria.

Figure 5.14 represents the influence between the Cw criterion and the other eight tested criteria. The Cw criterion achieves the best correlation with the Ft criterion, which can be seen from the fact that the Pareto front for that pair of criteria forms almost a straight line. However, this is expected since the definition for those two criteria are quite similar. Good correlation is also achieved with the C_{max} and F_{max} criteria, but not to such a great extent. The correlation between the Cw criterion and the due date related criteria is not as good as for the previous three criteria. Nevertheless, the improvement in the Cw criterion will not cause extremely bad results for the due date related criteria. In addition, the Cw criterion correlates much better with Twt than the other two due date related criteria, which makes it is more beneficial to simultaneously optimise the Cw criterion with Twt. Finally, the $Etwt$ and M_{ut} criteria are also negatively correlated with the Cw criterion to an extremely large extent.

The correlation of the $Etwt$ criterion with the other considered criteria is shown in Figure 5.15. The figure denotes that the $Etwt$ criterion is negatively correlated with all other scheduling criteria. The largest negative correlation can be observed with the completion time and flowtime related criteria. This can be seen by the fact that if a good value is achieved for the $Etwt$ criterion this necessarily leads to a very bad result in any of the flowtime or completion time related criteria. The negative correlation between the $Etwt$ criterion and the Nwt and M_{ut} criteria is less severe, which makes it easier to achieve relatively good values for both criteria. For the other two due date related criteria the negative correlation is the smallest. This is expected since one of the goals of the $Etwt$ criterion is to reduce the tardiness of all jobs, and therefore this also has an effect that better values can be achieved for the T_{max} and Twt criteria.

Figure 5.16 represents the correlation between the F_{max} criterion and the other tested scheduling criteria. The figure denotes that F_{max} is strongly correlated with the C_{max} criterion, but also

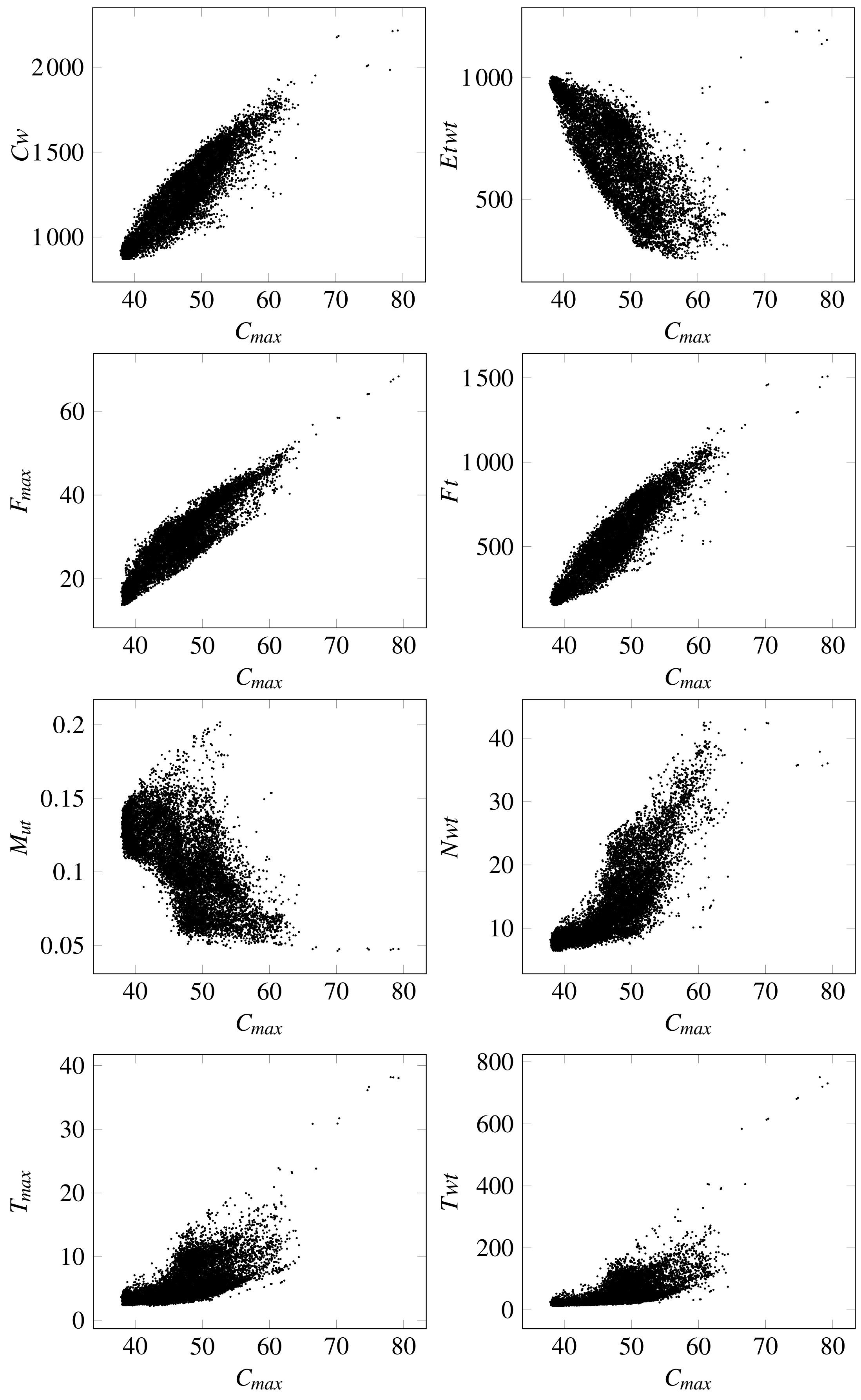


Figure 5.13: Correlation of the C_{max} criterion with the other criteria

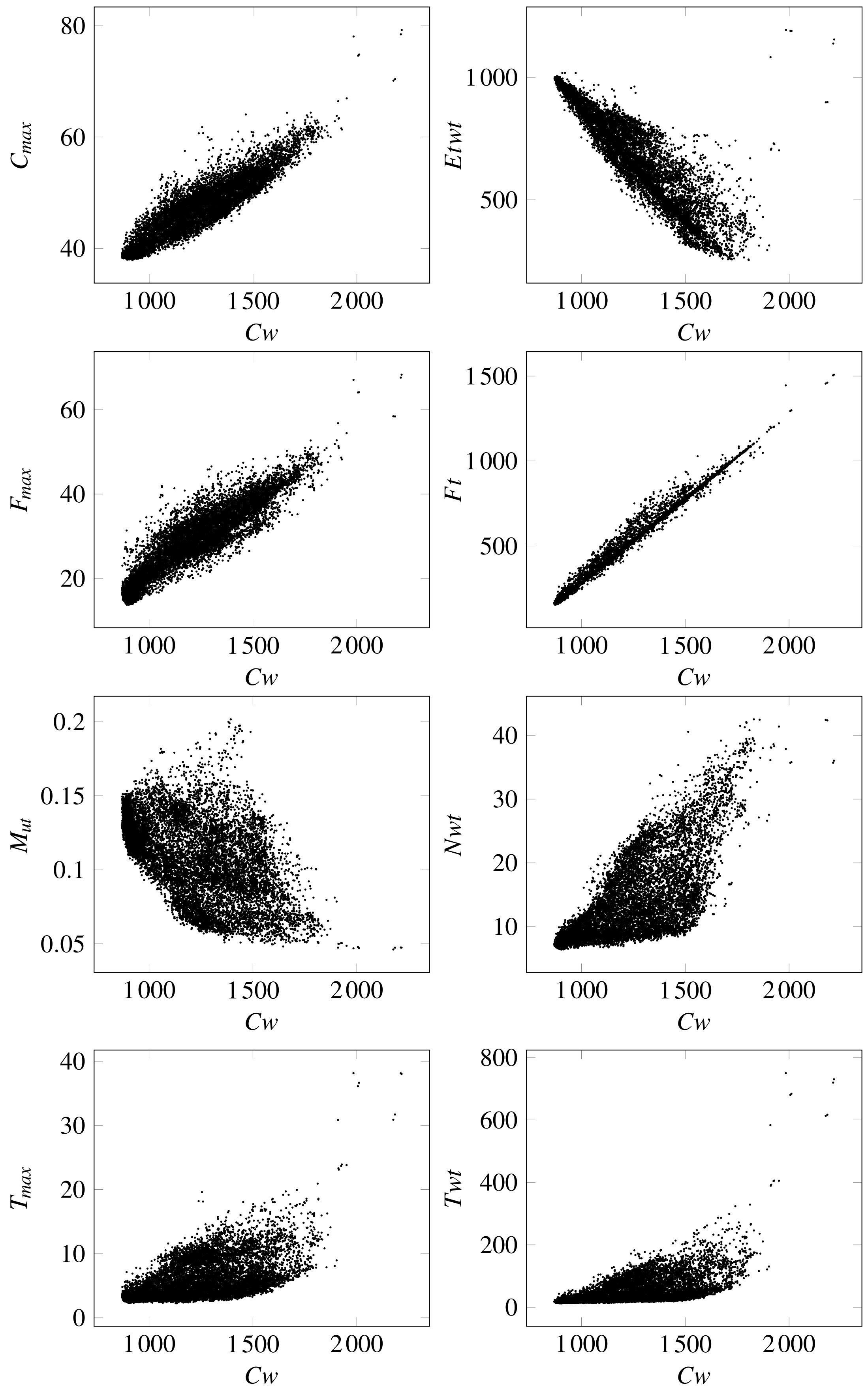


Figure 5.14: Correlation of the *Cw* criterion with the other criteria

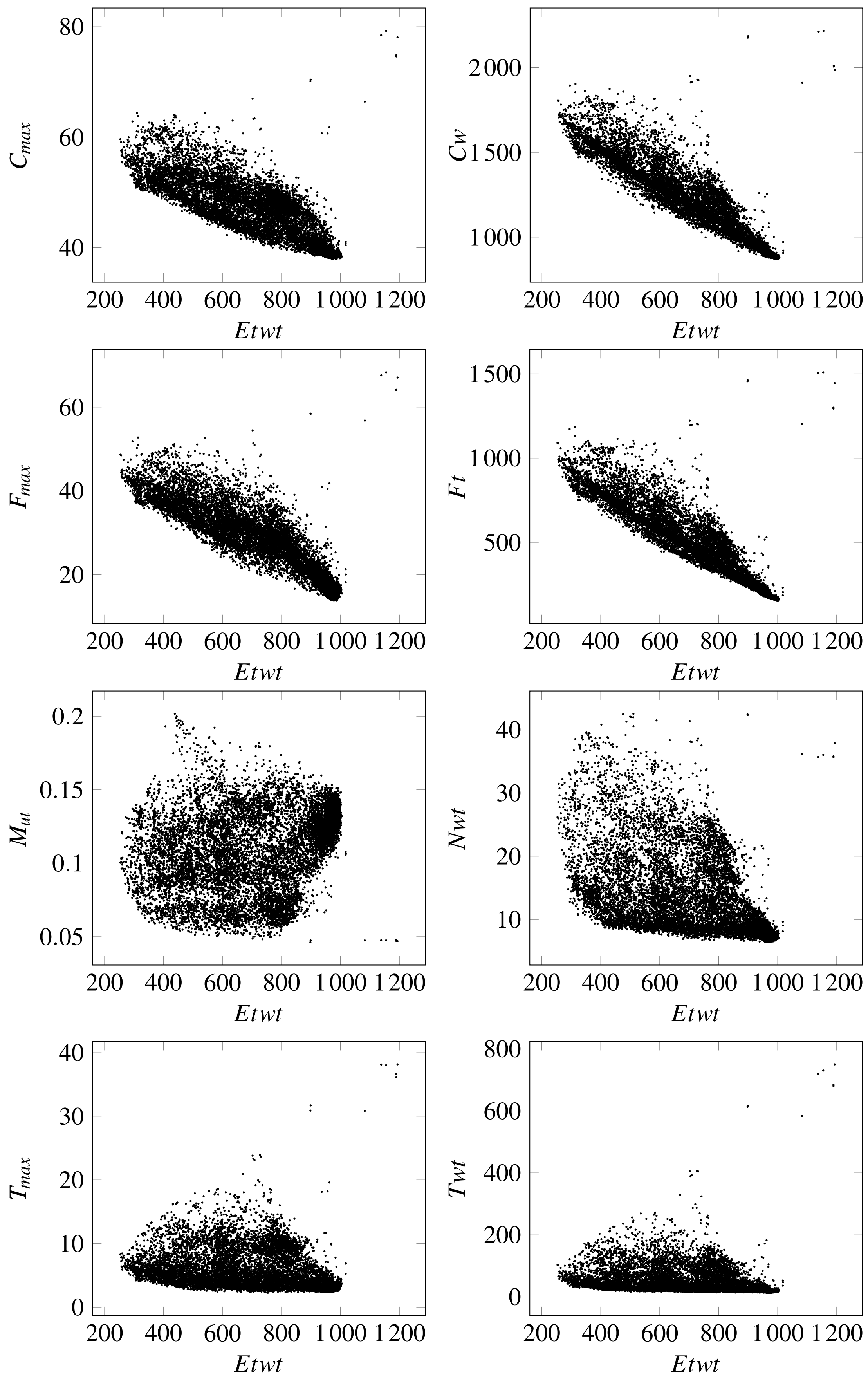


Figure 5.15: Correlation of the *Etwt* criterion with the other criteria

with the other two completion time and flowtime related criteria. The worst correlation of the F_{max} criterion is achieved with the M_{ut} and $Etwt$ criteria. On the other hand, the F_{max} criterion also correlates well with the T_{max} and Twt criteria. The reason for this is due to the fact that if the maximum flowtime is reduced it is also very likely that the maximum tardiness will also be reduced to a certain extent, since the jobs will be in the system for a shorter amount of time, and therefore the jobs will have less chance of being late.

Figure 5.17 represents the correlation between the Ft criterion and the other eight tested criteria. The best correlation is achieved for the Cw criterion. A good correlation can be observed for the Twt criterion as well. Therefore, the optimisation of the Ft criterion will not lead to overly bad results for the Twt criterion. As previously described, the C_{max} and F_{max} criteria are also positively correlated with the Ft criterion, at least to a certain extent. The other two due date related criteria have a worse correlation with Ft, but again not as extremely negative as in the case when the Ft criterion was optimised with the $Etwt$ and M_{ut} criteria.

The correlation between the M_{ut} criterion and the other scheduling criteria are represented in Figure 5.18. The figure shows that M_{ut} correlates negatively with all the other scheduling criteria. This is expected since the M_{ut} criterion tries to disperse the load on all machines equally, and thus can cause the jobs to wait until they are scheduled on a certain machine. However, this criterion is not as negatively correlated with the other criteria, as was the case with the $Etwt$ criterion. It shows the most negative correlation with the flowtime and completion time related criteria, but it is less negatively correlated with the due date related criteria. The reason for this is that the delay of the jobs has less effect on the due date related criteria, since the job can be freely delayed until its due date, without any increase in the value of the criteria.

Figure 5.19 represents the correlation of the Nwt criterion with the other scheduling criteria. The most negative correlation of the Nwt criterion can be observed with the M_{ut} and $Etwt$ criteria. The Nwt criterion also negatively correlates with the flowtime and completion time related criteria, but only to a smaller extent. Positive correlation with the Nwt criterion is achieved by the other two due date related criteria. Out of those two criteria, better correlation is achieved with the Twt criterion, since the decrease in the number of tardy jobs will usually lead to a smaller value in the total tardiness.

Figure 5.20 represents the correlation between the T_{max} criterion with the other scheduling criteria. This criterion achieves the most positive correlation with the Twt criterion. The reason for this is that decreasing the maximum tardiness will also, to a certain degree, decrease the total tardiness. A good correlation can be observed between the T_{max} criterion and the Nwt and F_{max} criteria. The reason why T_{max} is less correlated with Nwt is because reducing the T_{max} value will not necessarily lead to less jobs being tardy. With the other criteria the correlation is worse to a certain extent, especially for $Etwt$ and M_{ut} where the correlation is extremely negative.

The correlation between the Twt criterion with the other scheduling criteria is shown in Fig-

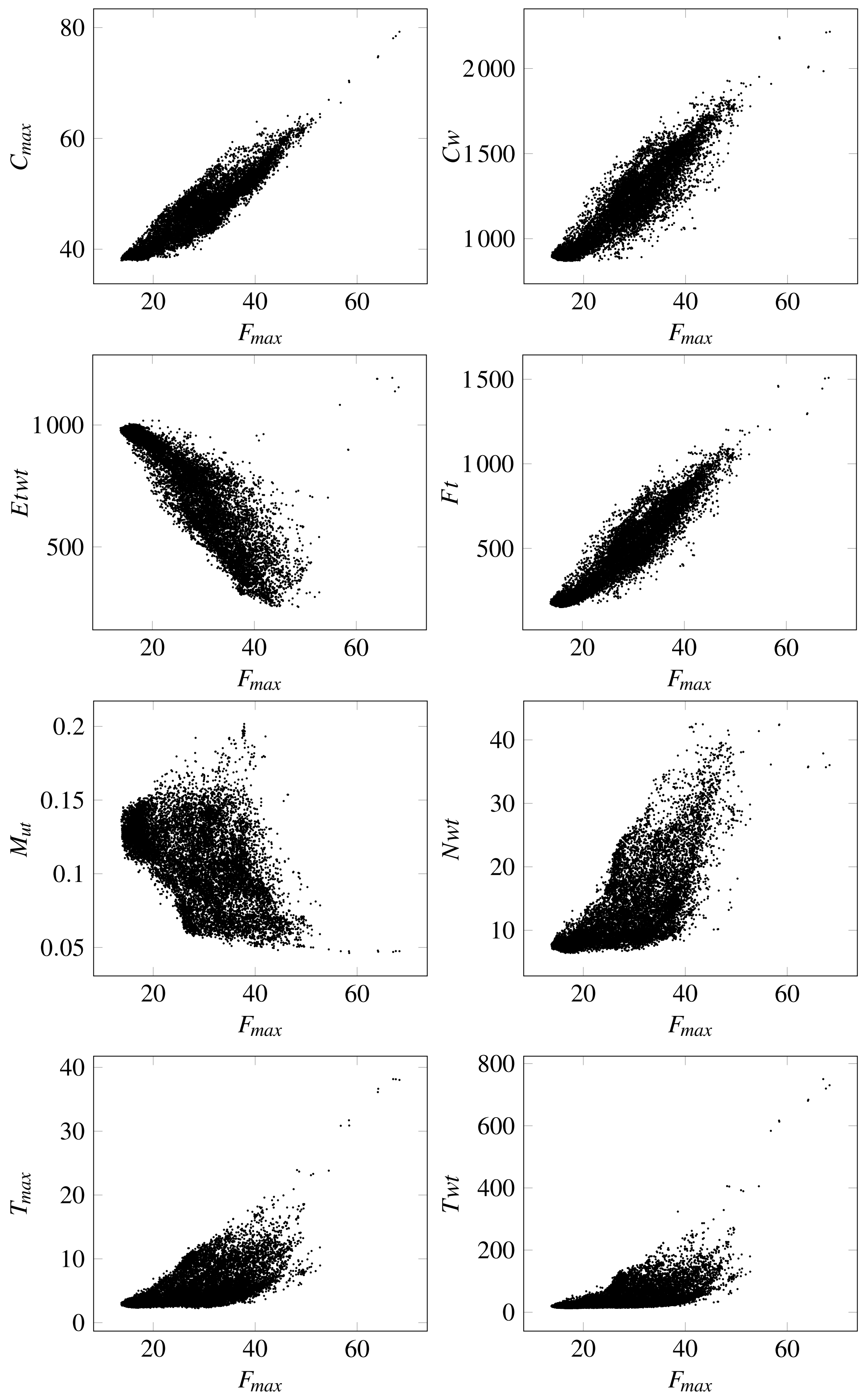


Figure 5.16: Correlation of the F_{max} criterion with the other criteria

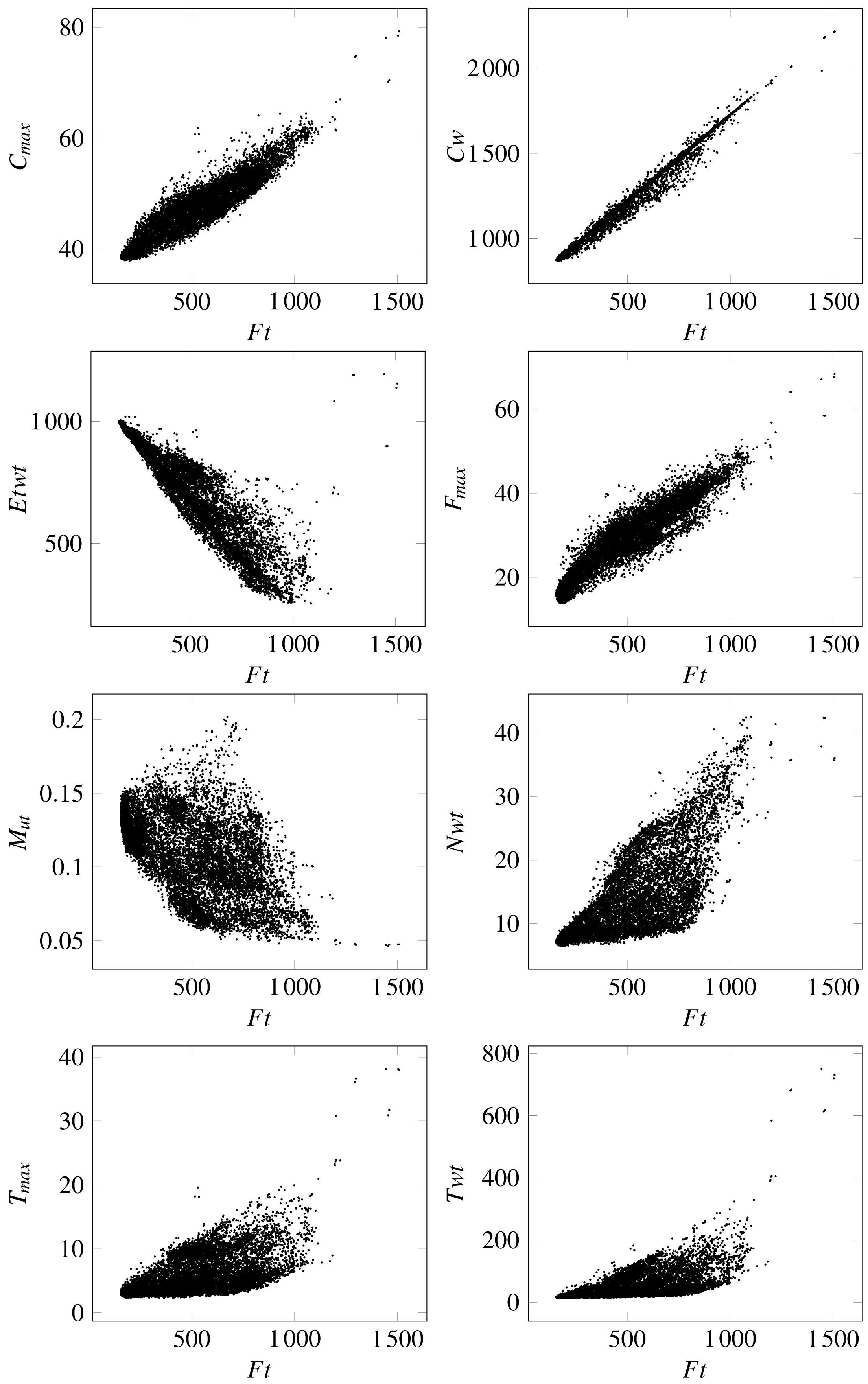


Figure 5.17: Correlation of the Ft criterion with the other criteria

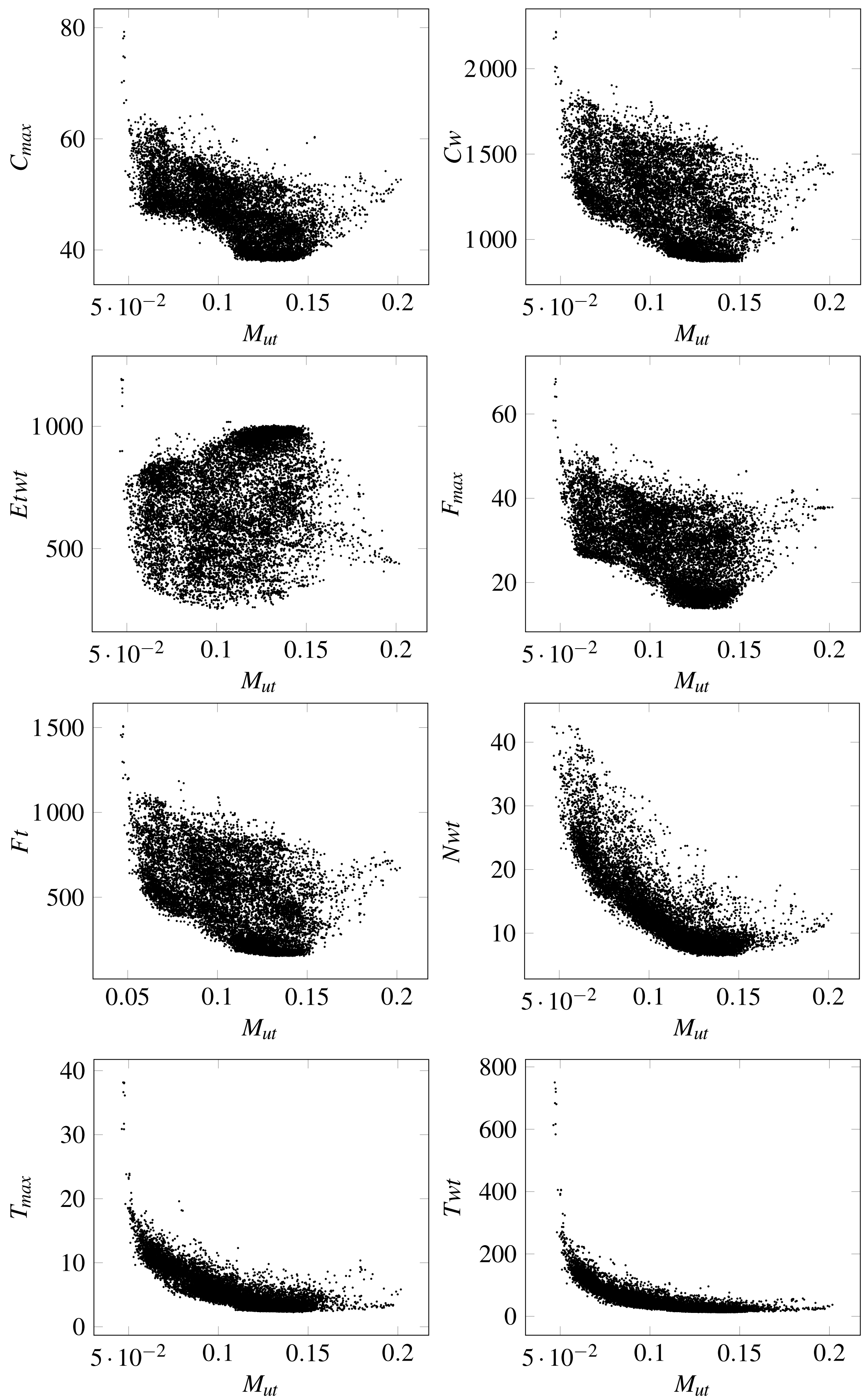


Figure 5.18: Correlation of the M_{ut} criterion with the other criteria

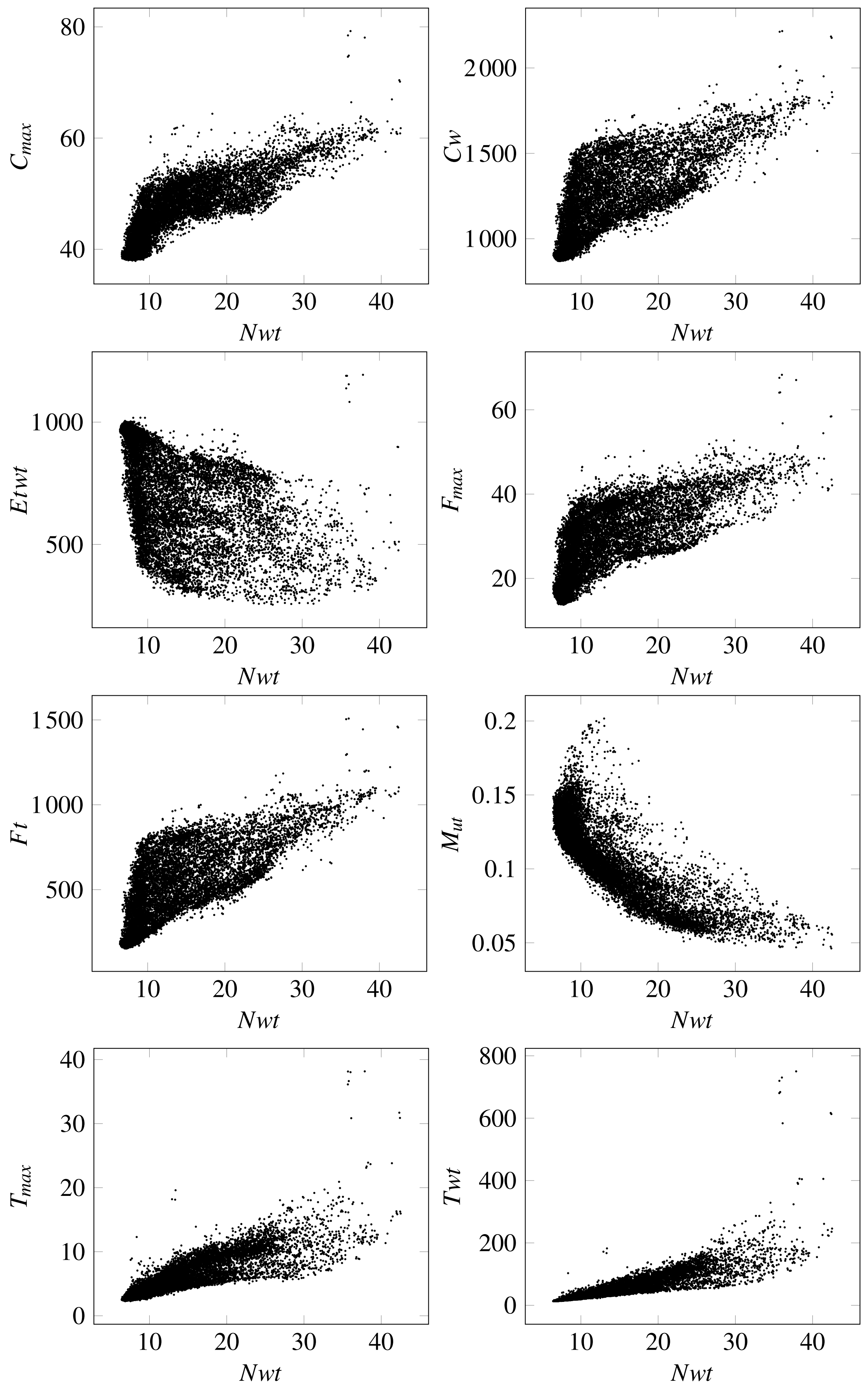


Figure 5.19: Correlation of the *Nwt* criterion with the other criteria

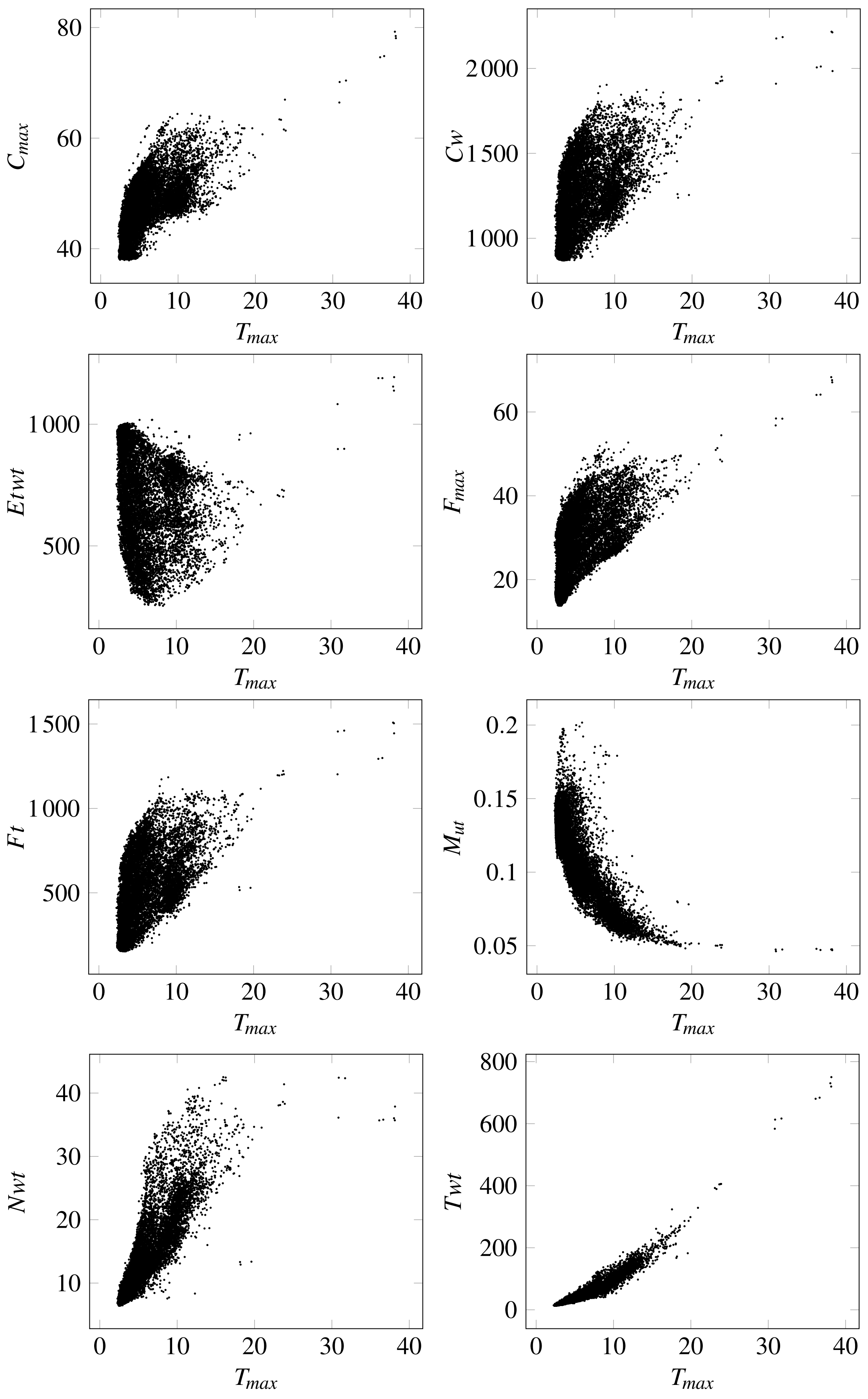


Figure 5.20: Correlation of the T_{max} criterion with the other criteria

ure 5.21. The best correlation is achieved with the Nwt and T_{max} criteria. This is expected since by decreasing the total tardiness, the number of tardy jobs and maximum tardiness will also implicitly be reduced. Good correlation can also be observed between the Twt criterion and the Ft criterion, since if the jobs spend less time in the system, they will have a smaller probability of being late. For the other flowtime and completion time related criteria the correlation is negative, but only to a small degree. Therefore, better Twt values will worsen the values of those criteria to a smaller extent. The worst correlation is again achieved with the $Etwt$ and the M_{ut} criteria.

5.7 Conclusion

Multi-objective and many-objective optimisation represents an increasingly growing field of research. The main reason for this is the fact that in many real world problems it is required to optimise several criteria simultaneously. Therefore, it is important to analyse how the different MOGP algorithms perform for various problems, and if they can find solutions competitive to those obtained by other methods.

In this chapter four MOGP algorithms were applied on several multi-objective and many-objective scheduling problems. The main aim of this chapter was to analyse whether DRs evolved by MOGP algorithms can outperform standard DRs and DRs evolved by SOGP, for various scheduling criteria. Since four MOGP algorithms were applied, a short analysis on the performance of all algorithms was also performed. Finally, the chapter also gives an analysis about the correlation of the different scheduling criteria.

The results obtained in this chapter show that the multi-objective DRs evolved by the MOGP algorithms were able to perform better than several standard manually designed DRs for most of the optimisation problems which were considered. The MOGP algorithms have also proven that they can even outmatch the performance of DRs which were evolved only for a single criterion. This demonstrates that although the algorithms are designed for optimising several criteria simultaneously, they are nevertheless powerful enough to obtain extreme solutions which in many cases outperform DRs evolved by SOGP. Out of the tested MOGP algorithms, NSGA-II and NSGA-III achieved the best results. However, the performance of the MOGP methods heavily depends on the combination of criteria which is optimised. Since NSGA-III has good convergence, it performed better for problems where the Pareto front is compact. On the other hand, the NSGA-II algorithm performs better on scheduling problems which include criteria that have a strong negative correlation with other criteria, and for which the diversification of solutions seems to be more important. In addition, the performance of the algorithms depends on the combination of criteria which is optimised. This also outlines the importance of selecting the appropriate set of criteria that should be optimised. The algorithms achieved the best per-

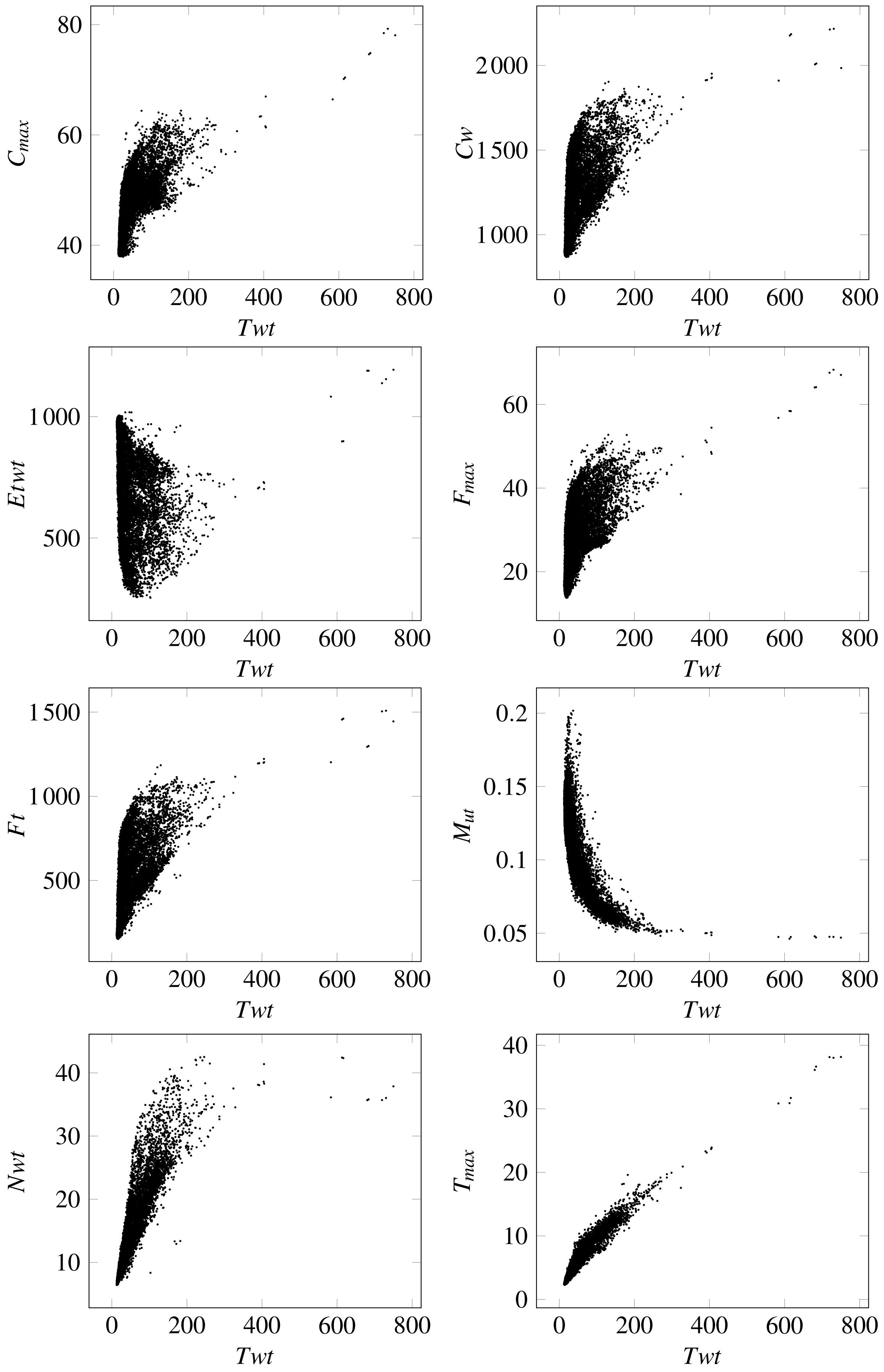


Figure 5.21: Correlation of the *Twt* criterion with the other criteria

formance if they are used to optimise criteria combinations which contain criteria of the same type (for example only due date related criteria), or criteria which are positively correlated. Even if the criteria are negatively correlated, but only to a smaller extent (like the due date and makespan related criteria), the algorithms still obtain good results. On the other hand, when optimising a criteria combination which includes criteria (like M_{ut} or $Etwt$) that are negatively correlated with all the other criteria, the performance of the algorithms deteriorates heavily. Therefore, in order for the MOGP algorithms to evolve better DRs, a set of criteria, which does not contain heavily conflicting objectives, should be used.

Based on the obtained results, it can be concluded that the MOGP algorithms are extremely well suited for evolving DRs which are capable for optimising several scheduling criteria simultaneously. However, there are still many open research topics in this area. Since there is a variety of MOGP algorithms, the first area of interest would be to test additional algorithms like: strength Pareto evolutionary algorithm 2 (SPEA2) [263], many-objective metaheuristic based on the $R2$ indicator (MOMBI) [264], and adaptive NSGA-III (A-NSGA-III) [265]. Another open topic would also be to include more non-standard scheduling criteria in the optimisation sets, to analyse how the algorithms would perform in those cases. Finally, it would be useful to perform a deeper analysis of the evolved multi-objective rules to extract knowledge on their behaviour.

Chapter 6

Designing ensembles of dispatching rules

Ensemble learning is often used to improve the performance of classifier systems in machine learning [266]. Although ensemble learning approaches like bagging [267] or boosting [268] are commonly used in the machine learning community, ensemble learning approaches have not been as extensively used together with GP to improve its performance. Some notable examples of applying ensembles to GP include classification with unbalanced data [269, 270], pattern classification [271] and intrusion detection [272].

Although DRs do not perform classification, they nevertheless need to perform a decision on which job should be scheduled on which machine at the current moment in time. Creating a single DR which will always perform good decisions is quite a difficult task. For that reason, it could prove more beneficial to use an ensemble of DRs to perform the decisions, because the ensemble could include DRs which perform well on different situations and for different kinds of problems. Ensemble learning techniques have rarely been used for creating ensembles of dispatching rules. In [35] an ensemble learning method was proposed, in which the ensemble is constructed by using the cooperative coevolution algorithm. The ensemble learning GP procedure generally produced more robust rules than the single rule GP. Unfortunately, this approach was applied only for the static scheduling problem. A second approach, named NELLI-GP, was proposed in [36]. This method creates the ensemble in a way that each DR, which is contained in the ensemble, is created to optimise only a certain subset of training instances. Therefore, the ensemble will consist of DRs where each rule will focus on solving a different subset of problem instances. Although this approach achieves good results, it was also applied only on the static job shop scheduling problem. In [37], the authors shortly investigate fitness sharing for evolving ensembles of DRs. The initial experiments show that with the proposed method it is possible to reduce the sizes of the ensembles, while retaining a relatively similar performance. Park et al. [38] apply the approach proposed in [35] to dynamic job shop scheduling problems, and also propose the use of a multilevel GP to create ensembles of DRs. However, ensembles generated by the proposed multilevel GP method achieve inferior results to the ensembles generated by the cooperative coevolution approach.

This chapter will analyse whether using ensembles of DRs can lead to better performance than when using individual DRs. The ensembles will be generated by five ensemble learning methods, two of which are proposed in this thesis, while the other three are taken from the literature. In the first section the applied ensemble learning methods will be shortly described. Following that, the design of the experiments will be explained, after which the results for all of the tested ensemble learning methods are presented. A short discussion about the different observations which were made during the experiments is also given. Lastly, an analysis of the best ensembles achieved by the different ensemble learning methods is performed to obtain a deeper insight on how the different ensemble learning approaches construct the ensembles. The chapter is concluded with a short overview and directions for further research.

6.1 GP ensemble learning methods

6.1.1 Simple ensemble combination

Simple ensemble combination (SEC) represents an ensemble learning approach which, unlike the other tested approaches, creates ensembles out of already existing DRs. Therefore, the approach has a smaller computational complexity, since it does not need to evolve new DRs which would constitute the ensemble. This allows for a greater flexibility in the choice of the DRs which can be used for the creation of ensembles.

The SEC approach consists of two independent steps which need to be specified. The first is the way in which the DRs in the ensemble will be combined to perform a joint decision, while the second is the procedure that selects which DRs will form the ensemble. The benefit of the SEC approach is that there is much freedom in defining how each of those steps will work, thus allowing for the design of different variants of the approach. The rest of this section will describe both of these steps in detail.

Combination of DRs into an ensemble

One of the most important things which need to be specified for the SEC method is the way in which the ensemble will perform its decisions based on the DRs which are contained in the ensemble. In order for the DRs which are contained in the ensemble to perform a joint decision, two combination methods, which are based on similar procedures in the machine learning community [266], are used: sum and vote. Both combination methods will also be used by other ensemble learning approaches in order for the entire ensemble to perform its decision.

The sum combination method is defined as a sum of the priority values of all DRs contained in the ensemble. The result of that sum represents the priority value which will be assigned to a job-machine pair. This value is used by the schedule generation scheme to schedule jobs on machines in a way that the job-machine pair which received the best priority value will be chosen for scheduling. The obvious advantage of this approach is its inherent simplicity, since the values of all DRs in the ensemble only need to be summed up. There is however one open issue with this combination method which needs to be addressed and explained. The issue is that if the ensemble consists of DRs which are independently generated, then there is no guarantee that the priority values obtained from the different DRs will be of the same order of magnitude. Because of that reason it is possible that some DRs in the ensemble would not have any influence on the final decision of the ensemble, since the contribution of their priority values would be insignificant when compared to the priority values of other DRs. This issue could be solved by normalising the priority values obtained by each DR. Unfortunately, this can not be easily achieved, since the range of priority values which can be obtained by a DR is not known in dynamic environments. In static environments it would, for example, be possible to determine the range of priority values for each DR by using *interval arithmetic* [137]. However, this issue does not influence the ensemble decisions as much as it would be expected, since it was shown that the priority values of different DRs were usually of the same order of magnitude, and thus each DR in the ensemble had a similar influence on the decision of the entire ensemble.

On the other hand, the vote combination method functions a bit differently. In the vote method, instead of summing the priority values of all DRs in the ensemble, each DR casts a vote for the element which received the best priority value by that rule, and the element which received the most votes is the one that should be selected for scheduling. An obvious problem with this procedure is that ties can appear much more often than with the sum method. For example, if there are many elements for which the DR can cast a vote, then it is possible that each element receives at most one vote, and thus it would be problematic to determine which element should be selected. In order to alleviate this problem, the voting is not performed for all job-machine pairs immediately, but will be split up in several parts. In the first part, for the currently considered job, each DR will cast a vote for the machine that received the best priority value for that job. The machine which received the most votes is assigned to the currently considered job. If this was the first job for which a machine was assigned to, the procedure is repeated for the next unscheduled job which is already released into the system. However, if there exists already an association of a machine to a job, then the vote method is used again to determine which of those two associations is better. In this second step each DR casts a vote for either job-machine association, and the one which receives the most votes is kept, while the other is deleted. This procedure is repeated for all unscheduled, but released jobs. In the end of the procedure, one job-machine association will remain, and if the machine in that association is free, the job which was associated to the machine will be scheduled on it.

Algorithm 6.1 represents a pseudo-code of the vote combination method. If at any time of the vote combination method several job-machine associations receive the same number of votes, then the one whose job has the earlier release time will be chosen. Naturally, a better way could have been used to deal with ties, for example using the sum combination method, but in this thesis it was decided to focus only on the two simple methods, and to leave the possibilities for their improvements for further research. An obvious advantage of the vote method over the sum combination method is that the magnitudes of priority values are not important, since each DR casts a vote of the same weight. Therefore, the vote combination method is more stable, since it will not allow a single DR to have a large influence on the decisions of the ensemble, which could happen for the sum combination method. However, unlike in the sum combination method, ties between jobs are more probable to occur in this method, especially for problems with a larger number of jobs and machines.

Algorithm 6.1 The vote combination method

```
1: Let bestPair represent the best selected job-machine association (empty at the beginning)
2: for each unscheduled job which is already released into the system do
3:     for each DR in the ensemble do
4:         Calculate the priority value by using the selected DR for all machines
5:         Determine the machine for which the DR achieved the best value and vote for it
6:     end for
7:     Select the machine with the most votes
8:     Let currentPair denote the job-machine association chosen in this iteration
9:     if bestPair is not empty then
10:        for each DR in the ensemble do
11:            Make a vote between currentPair and bestPair
12:        end for
13:        if currentPair received more votes than bestPair then
14:            bestPair ← currentPair
15:        end if
16:    else
17:        bestPair ← currentPair
18:    end if
19: end for
20: Schedule the job in the bestPair on the machine in the bestPair
```

Creation of ensembles of DRs

The second and more difficult part of SEC is to define the procedure for choosing which DRs will form the ensemble. Since scheduling is performed in dynamic conditions, the ensemble needs to be constructed in advance, preferably on an independent problem set from the one which was used to generate the DRs, to reduce the probability of overfitting. Five ensemble construction methods will be proposed for constructing the ensembles: random selection method, probabilistic selection method, grow method, grow-destroy method, and instance based method.

The most simple of the proposed ensemble construction methods is the *random selection method*. This method constructs the ensemble in a way that it randomly selects which of the available DRs should form the ensemble. In this method, all DRs will have the same probability of being selected for the ensemble. Unfortunately, the probability of obtaining a good ensemble by trying out only one combination of DRs is quite small. For that reason, instead of constructing a single ensemble, this method constructs a certain number of ensembles, evaluates them on the problem instance set, and selects the one which achieved the best value for the optimised objective. The number of ensembles which will be created and tested is a parameter of the random selection method. Naturally, the complexity and execution time of this approach will increase as the number of the tested ensembles increases. The pseudo-code of this approach is presented in Algorithm 6.2. The approach will not keep track if duplicate ensembles appear, so it is possible that the approach creates the same ensemble several times. However, because of the sheer number of combinations this does not pose a serious problem. In addition, if the number of possible combinations is smaller than the number of ensembles which should be constructed, then an exhaustive search can applied rather than a random search. The main disadvantage of this approach is that the ensemble construction is performed completely randomly, without using any information about the quality of the DRs or generated ensembles.

Algorithm 6.2 The random selection method

```
1:  Let $R$ represent the set of available DRs
2:  $bestE \leftarrow \emptyset$
3:  while the number of created ensembles is less than the maximum allowed value do
4:      $E \leftarrow \emptyset$
5:      while Size of $E$ is smaller than the given ensemble size do
6:          Select a random DR from $R \setminus E$, and add it to $E$
7:      end while
8:      if $bestE$ is empty then
9:          $bestE \leftarrow$ E
10:     else if $E$ achieves a better fitness than $bestE$ then
11:         $bestE \leftarrow$ E
12:     end if
13: end while
```

An extension of the previous approach is the *probabilistic selection method*. In this method each DR has a different probability of being selected, which depends on the quality of the obtained DRs. Therefore, DRs which perform better will have a higher probability of being selected into the ensemble. Algorithm 6.3 represents the pseudo-code of the probabilistic selection method. In the proposed method the selection probability of a DR was defined to be proportional to the value of its fitness. An additional ε value is used to scale all fitness values to prevent the worst individual of having the probability of being selected equal to 0. Naturally, other more sophisticated ways of calculating the selection probabilities of DRs can also be used, like for example *windowing* [64]. The motivation behind this method is to guide the search towards those DRs which individually have a better fitness, in hope that better ensembles will consist of DRs which individually achieve good fitness values.

Algorithm 6.3 The probabilistic selection method

```
1: Let $R$ represent the set of available DRs
2: $bestE \leftarrow \emptyset$
3: Let $worstFit$ represent the best fitness achieved by any of the DRs in $R$
4: Let $\varepsilon$ represent an arbitrary constant
5: for each $rule$ in $R$ do
6:     Let $fit$ represent the fitness of $rule$
7:     Calculate the probability of $rule$ as $worstFit - fit + \varepsilon$
8: end for
9: Normalise the probabilities of all rules
10: while the number of created ensembles is less than the maximum allowed value do
11:     $E \leftarrow \emptyset$
12:     while Size of $E$ is smaller than the given ensemble size do
13:         Select a random DR from $R \setminus E$ based on the assigned probabilities and add it to $E$
14:     end while
15:     if $bestE$ is empty then
16:         $bestE \leftarrow$ E
17:     else if $E$ achieves a better fitness than $bestE$ then
18:         $bestE \leftarrow$ E
19:     end if
20: end while
```

The *grow method* is a greedy heuristic which functions differently than the previous two methods. Unlike the aforementioned two methods, which randomly select DRs that should form the ensemble, and evaluate only the final ensemble, the grow method creates the ensemble incrementally. The method starts with a single DR which can either be a randomly selected DR, the DR with the best fitness value, or a DR selected by certain other criteria. In each iteration of the method, the DR which leads to the highest increase in the quality of the ensemble is added into it. This is done until the ensemble reaches its specified size. The pseudo-code of the procedure is given in Algorithm 6.4. Unlike the previous two construction methods, this one behaves deterministically if the same initial DR is used. It can happen that in certain iterations the growth of the ensemble can lead to a worse result than that achieved in the previous iteration. Since such situations can quite often represent only local minima, the procedure is continued until the ensemble of the desired size is obtained. The motivation behind this method is that the DRs which increase the overall quality of the ensemble should be selected to form the ensemble.

The *grow-destroy method* represents an extension of the previous method. The first part of the method is the same as in the grow method, however, instead of building up an ensemble of

Algorithm 6.4 The grow method

```
1: Let $R$ represent the set of available DRs
2: $E \leftarrow \emptyset$
3: Select a random DR from $R$ and add it to $E$
4: while the size of $E$ is smaller than the given ensemble size do
5:     Select the DR from $R \setminus E$ whose addition to $E$ would lead to the best results, and add it to $E$
6: end while
```

the specified size, this method creates an ensemble twice the size. After this step is done, the method iteratively removes DRs from the ensemble, until it reaches the specified size. In each iteration the method chooses to remove the DR whose removal would result in the best fitness of the smaller ensemble. Algorithm 6.5 represents the pseudo-code of the described method. As the previous method, this one is also deterministic given the same starting DR. The motivation for this method is to allow for the ensemble to grow over the specified size and to collect more good DRs in the ensemble. Then in the next step the methods removes those DRs which least contribute to the quality of the ensemble.

Algorithm 6.5 The grow-destroy method

```
1: Let $R$ represent the set of available DRs
2: $E \leftarrow \emptyset$
3: Select a random DR from $R$ and add it to $E$
4: while the size of $E$ is smaller than twice of the specified ensemble size do
5:     Select the DR from $R \setminus E$ whose addition to $E$ would lead to the best results, and add it to $E$
6: end while
7: while the size of $E$ is larger than the specified ensemble size do
8:     Select the DR from $E$ whose removal would lead to the best results, and remove it from $E$
9: end while
```

Finally, the *instance based method* works similarly as the grow method, but it uses a completely different way of selecting the DRs which will form the ensemble. To select the DRs which should be added to the ensemble, the decision of this method will be based on the problem instances for which the ensemble does not perform well. In the first step for each problem instance the minimum objective value, achieved by any of the available DRs, is determined. The first DR of the ensemble can be chosen either randomly, by its fitness, or by the number of problem instances for which it achieves the best result. The method then iteratively adds DRs to the ensemble. However, the method selects those DRs which achieve the best performance on those problem instances on which the ensemble does not achieve a value at least as good as the best value achieved by any of the DRs on their own. Therefore, this approach does not necessarily guide the search towards ensembles which have a better overall fitness, but rather to those ensembles which solve the most problem instances as best as possible. Algorithm 6.6 represents the pseudo-code of the instance based method. The motivation for such a selection method is to create an ensemble which achieves good performance on as many problem instances as possible. Such a selection method should hopefully produce an ensemble suited for solving a variety of problem instances well, that should also produce good results on unseen problem instances. Another benefit of this approach is that at each iteration it evaluates only one ensemble, since DRs are selected based on their individual performance, and the ensemble needs to be evaluated only at the end of each iteration to determine on which problem instances it does not perform well.

Algorithm 6.6 The instance based method

```
1:  Let $R$ represent the set of available DRs
2:  $E \leftarrow \emptyset$
3:  Let $P$ represent the set of problem instances in the validation set
4:  Select a random DR from $R$ and add it to $E$
5:  for each problem instance $p$ in $P$ do
6:      determine the best objective value achieved by any DR in $R$ and denote it as $opt[p]$
7:  end for
8:  while Size of $E$ is smaller than the specified ensemble size do
9:      Let $NP$ denote the set of problem instances for which the ensemble $E$ can not achieve
        better or equal values than those in $opt$
10:     for each rule $r$ in $R \setminus E$ do
11:         $iop[r] = 0$
12:         for each problem instance $p$ in $NP$ do
13:             Let $fit$ denote the fitness of rule $r$ on instance $p$
14:             if $fit > opt[p]$ then
15:                 $iop[r] += fit - opt[p]$
16:             end if
17:         end for
18:     end for
19:     Select rule $r$ from $R \setminus E$ which has the smallest value of $iop[r]$ and add it to $E$
20: end while
```

6.1.2 BagGP

BagGP is an ensemble learning approach which applies the bagging procedure to GP [273]. This approach evolves DRs in a way that each DR is evolved on a different training set, which is constructed by sampling with repetition from the original training set. The evolved DRs are then combined to form an ensemble. The motivation for this approach is that by evolving DRs on different training sets it is more likely that the evolved DRs will specialise for solving instances with different characteristics. This should in turn also allow the ensemble to perform better on a wider set of problems. The pseudo-code of the BagGP approach is given in Algorithm

6.7. This approach, in addition to the ensemble size and combination method, has an additional parameter, called *bag size*, which determines the size of the sampled training set used to evolve the DRs. The size of the newly sampled training set can be set to an almost arbitrary value, which can be smaller, larger or equal to the original training set size.

Algorithm 6.7 The BagGP approach

```
1: Let $P$ represent the validation set of problem instances
2: Let $n$ represent the size of the ensemble which should be constructed
3: $E \leftarrow \emptyset$
4: while $|E| < n$ do
5:     Create a problem instance set $S$ by sampling with repetition from $P$
6:     Use GP on $S$ to evolve a new DR
7:     Add the evolved DR into the ensemble $E$
8: end while
```

6.1.3 BoostGP

BoostGP is an approach which applies the AdaBoost [268] algorithm to GP [274, 275]. This ensemble learning approach evolves several DRs so that it weights the training set instances in a way that problem instances which were solved poorly in previous GP runs get a higher importance in the following GP runs, so that newly evolved DRs focus more on solving such problematic instances. Algorithm 6.8 denotes the BoostGP approach. The approach is mostly the same as the ones denoted in the literature with one notable difference. Since this algorithm is adapted from the regression problem in which the fitness is usually calculated as the difference between the value which was achieved by the evolved individual and the expected value $|f_i - y_i|$, there is a need to adapt it to the case of evolving DRs where there does not exist an explicit expected value which needs to be achieved, but rather a certain criterion is minimised. Since neither of the criteria which will be used for testing can have a value lower than zero, the approach is adjusted to treat zero as the expected values for each criterion.

In order to combine the DRs into a single ensemble, four different combination methods are used. The first two methods are the sum and vote methods described previously. The other two methods represent the weighted sum and vote methods which use the confidences obtained for each DR as the weights which will be used to multiply the vote of the DR in the vote method, and the priority value of the DR in the sum method. In addition to the combination method, the second parameter of this approach is the size of the ensemble which needs to be generated.

Algorithm 6.8 The BoostGP approach

```
1: Let $P$ be the number of problem instances in the training set and $T$ the ensemble size
2: Initialise the weights for each problem instance as $D_1(k) = 1/P$
3: for t=1..T do
4:     Run GP with the following fitness function: $fitness_t = \sum_{k=1}^{P}(f(individual, p_k) * D_t(k)) * P$ where $f$ represents the original fitness function, $p_k$ the problem instance on which the fitness is calculated, and $individual$ the individual for which the fitness is calculated
5:     Let the best individual in the GP run be denoted as $ind$
6:     Compute the loss for each problem instance: $L_k = \frac{|f_t(ind,p_k)|}{max_{k=1..P}(f_t(ind,p_k))}$
7:     Compute average loss: $\bar{L} = \sum_{k=1}^{P} L_k * D(k)$
8:     Compute the confidence for the DR: $\beta_t = \frac{\bar{L}}{1-\bar{L}}$
9:     Update the distribution $D_{t+1}(k) = \frac{D_t(k)*\beta_t^{1-L_i}}{Z_t}$, where $Z_t$ represents the normalisation factor
10:    Add the DR represented by $ind$ into the ensemble $E$
11: end for
```

6.1.4 Cooperative coevolution

The cooperative coevolution approach is an evolutionary algorithm which divides the optimisation problem into several sub-problems, which are solved independently in order to solve the original problem [276]. Each sub-problem is solved by one sub-population in the evolutionary algorithm, and the only interaction between individuals from different sub-populations is when they are combined for evaluation. Naturally, it is not possible to combine one individual with all individuals from the other sub-populations and calculate its fitness for all the combinations, since this would be too time consuming. For that reason, there usually exists a list which contains a representative solution from each sub-population. An individual is then evaluated in combination with the representative individuals from other sub-populations. Each time an individual which improves the fitness of the representative list is found, it is included in the representative list and replaces the previous representative from its sub-population. Algorithm 6.9 represents the pseudo-code of the cooperative coevolution algorithm. This approach can also easily be used to evolve ensembles of DRs in a way that each sub-population evolves a single DR which are then combined with DRs from other sub-populations to form an ensemble. The ensemble size and ensemble combination method are the only parameters which need to be defined for this approach.

6.1.5 Ensemble subset search

In order to additionally improve the performance of the ensemble learning approaches, an additional search will be performed on ensembles after the learning process. The objective of

Algorithm 6.9 The cooperative coevolution approach

```
1:  Initialise a number of populations $P$ equal to the size of the ensemble which needs to be
    generated
2:  Let $R$ represent the list of representatives from each population, and $R[p]$ the representative
    of population $p$
3:  Randomly select a single individual from each population and add it to $R$
4:  Evaluate $R$ on the problem instances
5:  for each population $p$ in $P$ do
6:      for each individual $ind$ in $p$ do
7:          Let $C$ denote a copy of $R$
8:          $C[p] \leftarrow ind$
9:          Evaluate the ensemble $C$ on the problem instances
10:         if $C$ achieved a better fitness than $R$ then
11:             $R \leftarrow C$
12:         end if
13:     end for
14: end for
15: do
16:     for each population $p$ in $P$ do
17:         Select $k$ individuals from the population $p$
18:         Perform the crossover operator on the best two of the $k$ selected individuals to create
            a new individual $ind$
19:         Perform the mutation operator on $ind$ with a certain probability
20:         Replace the worst of the $k$ selected individuals with the newly created individual
21:         Let $C$ denote a copy of $R$
22:         $C[p] \leftarrow ind$
23:         Evaluate the ensemble $C$ on the problem instances
24:         if $C$ achieved a better fitness than $R$ then
25:             $R \leftarrow C$
26:         end if
27:     end for
28: while termination criterion is not met
```

this procedure is to determine the optimal subset of DRs contained in the ensemble. The intuition behind this approach is that the ensembles which are evolved by the ensemble learning approaches do not have to be optimal, and that it could be possible to construct a better ensemble by using only a subset of the DRs contained in the original ensemble. This is especially possible when using approaches where the DRs of the ensemble are evolved independently of each other, like in BagGP or BoostGP. In those approaches the DRs forming the ensemble are evolved in independent GP runs, after which they are collected to form the ensemble. Therefore it is possible that the ensemble contains DRs that do not contribute to the quality of the ensemble. To remedy this, the original ensemble can be modified by removing the unnecessary DRs from the ensemble and consequentially improve the overall performance. By reducing the size of the ensemble, the execution speed and interpretability of ensembles can also be improved.

This approach takes the ensemble evolved by one of the ensemble learning approaches and uses the DRs that form the original ensemble to build ensembles of smaller sizes. If the size of the original ensemble is not too large, it is possible to try out all ensemble combinations of smaller sizes in a reasonable amount of time, and therefore determine the optimal ensemble subset. Then either the best overall ensemble subset, or the best ensemble subset of a concrete size can be selected. Otherwise, if the original ensemble is too large to perform an exhaustive search, the ensemble subsets need to be generated randomly, or by using a certain heuristic approach. Since the ensembles tested in this thesis will not contain more than ten DRs, it is possible to perform an exhaustive search with ensemble subset search (ESS). The description shows that ESS is similar to the SEC approach, with the differences being that it is applied on an existing ensemble of DRs, and that it constructs ensembles of different sizes (smaller than the original ensemble), unlike the SEC approach which creates ensembles of a predefined size only.

The benefit of this approach is that it can not only decrease the ensemble size, but also improve its performance. However, for ESS it is usually good to use an additional problem instance set to determine the optimal combination of the DRs that will form the ensemble subset. The additional problem instance set is used to reduce the probability of overfitting on the training set. After the optimal ensemble combination is determined, it can be used to solve unseen scheduling problems.

6.2 Experimental design

A set of 180 problem instances will be used to create and evaluate the ensembles. This set of problem instances is dived into three disjunct problem sets, each of which consists of a third of the problem instances. The first set represents the training set which is used by the different approaches to evolve the DRs. Since the SEC approach uses independently evolved DRs, which were evolved on the training set, it needs to use an additional problem instance set to determine the combination of DRs that should form the ensemble. ESS, which is applied on already existing ensembles, also needs to use an additional set of problem instances to determine the optimal ensemble subset. Therefore, these two approaches use a second problem instance set, called the validation set, to determine the combination of DRs which will form the ensemble. The final problem instance set, called the test set, is used to evaluate all the DRs and ensembles.

The parameters which are used by GP for evolving DRs are the same as those denoted in table 4.3. The SEC approach will simply use DRs which were previously evolved by GP, and therefore it is not required to generate new DRs. On the other hand, the other three ensemble learning methods need to generate new DRs. The BagGP and BoostGP approaches execute the GP algorithm for each DR they need to generate for the ensemble, since each DR is evolved independently in a separate GP run. This means that for the creation of one ensemble they will perform more iterations than when GP is used to evolve a single DR. It might seem that this leads to an unfair comparison between these approaches and the individual DRs evolved by GP. However, GP achieved no improvement with the increase of the number of iterations beyond the one which was used, but rather the performance of the generated DRs started to deteriorate, which means that the method began to overfit on the training set. Since cooperative coevolution evolves the DRs of an ensemble simultaneously, it will perform the same number of iterations as the GP which evolves individual DRs. All experiments were executed 30 times to obtain statistically significant results.

6.3 Performance analysis of the SEC approach

In this section it will be analysed how the different parts of the SEC approach influence its performance. For all experiments it will be tested whether the ensembles constructed by the SEC approach achieve significantly better results than the DRs out of which they were constructed. All experiments for which the SEC approach achieves significantly better results than the individual DRs will be denoted in grey. In addition, the best results achieved for different ensemble sizes will be denoted in bold.

6.3.1 Influence of different ensemble construction methods

This section will analyse the influence of the different ensemble construction methods on the performance of the ensembles. The five ensemble construction methods described in Section 6.1.1 will be compared. The random selection method and probabilistic selection method will use several values for the number of ensembles they create to determine the influence of that parameter on the performance of the construction methods. The number of ensembles which were constructed in one run of the method will be denoted in the tables alongside the name of the ensemble construction procedure. In addition, when the random selection method creates 20000 ensembles, the method will perform an exhaustive search when creating ensembles of size three, since for this ensemble size the number of distinct ensembles which can be constructed is less than 20000. Even if the approach would perform a random search it would still usually converge to the same value.

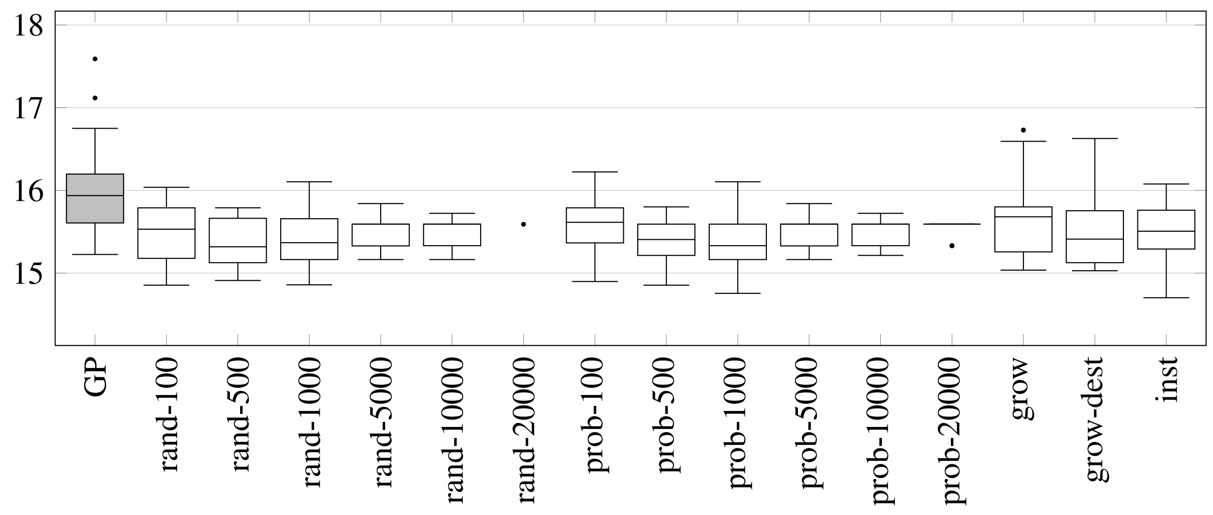
Table 6.1 represents the results achieved for the different ensemble construction methods when the sum combination method is applied. The results show that for all experiments the SEC approach achieved significantly better results than the DRs generated by GP. The largest improvement of SEC over GP amounts to 3.7% for the minimum value and 5.1% for the median value. Both those improvements were obtained when using ensembles of five DRs. An additional benefit of the SEC approach over GP is that it generates much less dispersed results, which can be seen by the maximum values which are in all cases smaller than the maximum value achieved by GP. This means that the SEC approach is more stable and will thus have a greater probability of obtaining good solutions than GP.

Although all the ensemble construction methods achieve significantly better results than GP, the quality of the ensembles depends heavily on both the size and the ensemble construction method. This can best be seen from Figure 6.1 which shows the box plot representation of the results. When creating ensembles of size three, the best results for the median value were achieved by the random selection method with 500 created ensembles, followed by the probabilistic selection method with 1000 created ensembles. The performance of the random selection method and probabilistic selection method depends heavily on the number of ensembles the methods construct. If a too large number of ensembles is constructed, then the methods tend to converge to the solution which is optimal on the validation set. This behaviour should be avoided, since the approaches start to overfit on the validation set and therefore do not perform well on unseen problem instances. Nevertheless, even in such occasions do the ensemble construction methods achieve significantly better results than DRs generated by GP. The grow-destroy and instance based methods also achieve quite good results, especially the instance based method which achieves the overall best solution when constructing ensembles of size three.

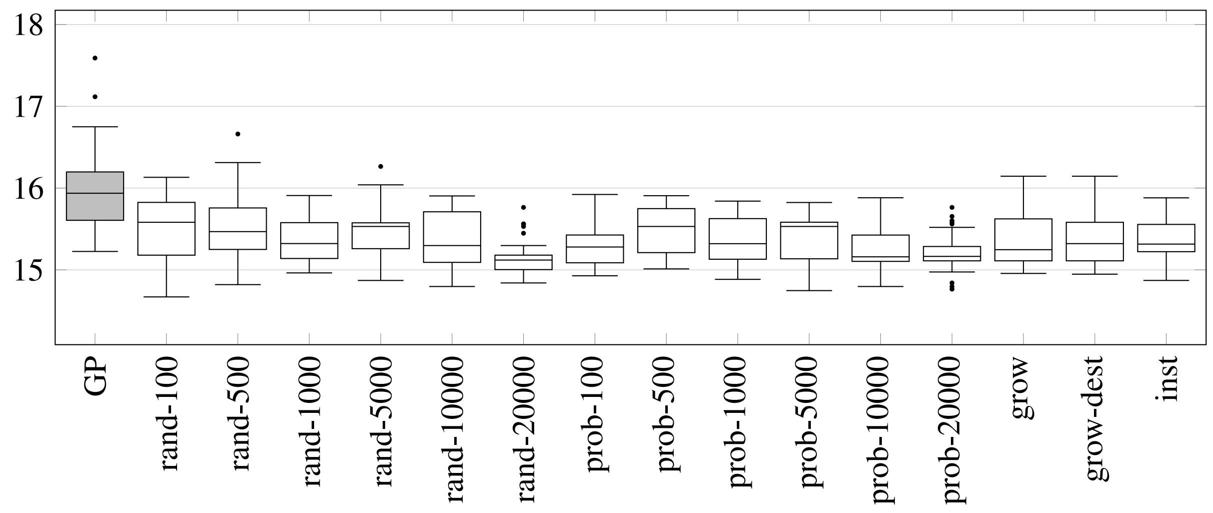
For larger ensemble sizes the behaviour of the approaches is different. First of all, the random and probabilistic selection methods achieve better results when creating a larger number of ensembles. As the number of constructed ensembles decreases, the results tend to deteriorate. For the ensemble size of five DRs, the best results for the median value were achieved by the random selection method when creating 20000 ensembles, followed closely by the probabilistic selection method when creating 20000 and 10000 ensembles. The grow, grow-destroy and instance based methods also perform rather well for the ensembles of size five. Those methods

Table 6.1: Influence of the ensemble construction methods on the results obtained by SEC with the sum combination method

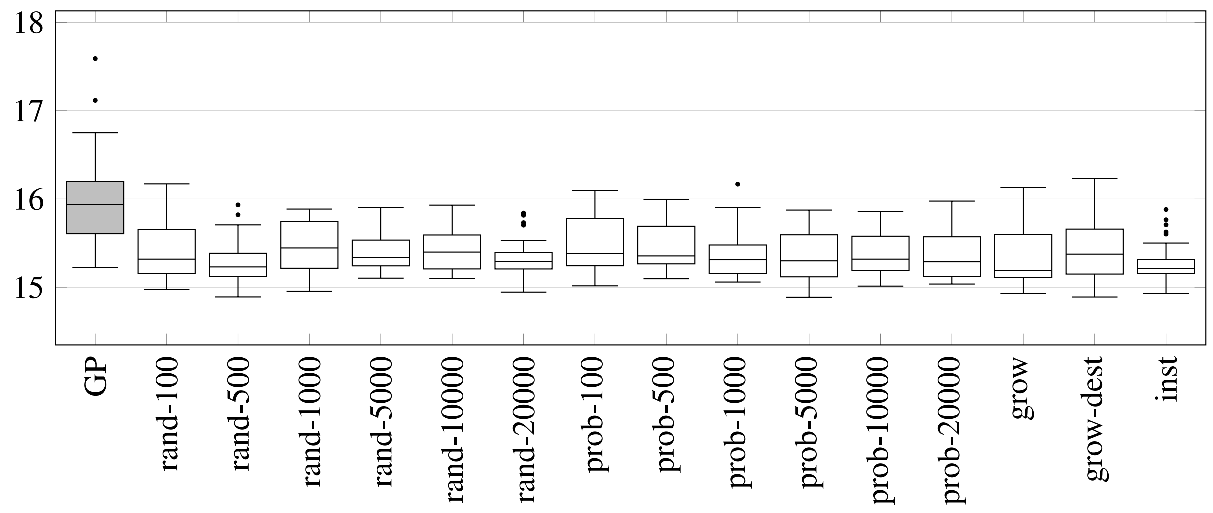
Procedure	Ensemble size								
	3			5			7		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
Rand 100	14.85	15.53	16.04	**14.67**	15.62	16.13	14.97	15.36	16.17
Rand 500	14.91	**15.32**	15.79	14.81	15.47	16.66	**14.89**	15.25	15.93
Rand 1000	14.86	15.38	16.10	14.96	15.36	15.91	14.95	15.52	15.86
Rand 5000	15.16	15.59	15.84	14.87	15.53	16.26	15.10	15.34	15.90
Rand 10000	15.16	15.59	15.72	14.80	15.37	15.90	15.10	15.40	15.93
Rand 20000	15.59	-	-	14.84	**15.12**	**15.66**	14.94	**15.14**	**15.61**
Prob 100	14.90	15.61	16.22	14.93	15.29	15.92	15.02	15.40	16.10
Prob 500	14.85	15.44	15.80	15.01	15.54	15.91	15.10	15.36	15.99
Prob 1000	14.76	15.33	16.10	14.88	15.39	15.84	15.06	15.31	16.17
Prob 5000	15.16	15.59	15.84	14.75	15.53	15.82	**14.89**	15.31	15.87
Prob 10000	15.21	15.59	15.72	14.80	15.16	15.88	15.01	15.32	15.86
Prob 20000	15.33	15.59	**15.59**	14.76	15.17	15.76	15.04	15.30	15.78
Grow	15.04	15.70	16.73	14.96	15.27	16.15	14.93	15.20	16.13
Grow-dest	15.03	15.45	16.63	14.95	15.32	16.15	**14.89**	15.38	16.23
Inst	**14.70**	15.51	16.08	14.87	15.33	15.88	14.93	15.22	15.88
GP	15.23	15.94	17.59	15.23	15.94	17.59	15.23	15.94	17.59



(a) Comparison of the ensemble construction methods for ensembles of size three



(b) Comparison of the ensemble construction methods for ensembles of size five



(c) Comparison of the ensemble construction methods for ensembles of size seven

Figure 6.1: Box plot representation of the results obtained by SEC with the sum combination method, when using different ensemble construction methods

did not perform equally well as the random and probabilistic selection methods for a larger number of generated ensembles. For most of the other cases they achieved equally good or even better results. Out of those three methods the instance based method seems to be the most stable and, and also obtains the best result.

For the ensembles of seven DRs, the best result is achieved by the random selection method which generates 20000 ensembles. The grow and instance based methods also obtain excellent results, achieving better median values than the random and probabilistic selection methods for almost all experiments. Also, for this ensemble size the performance of the random and probabilistic selection methods does not change much with the increase of the number of ensembles they create.

Table 6.2 represents the results for the different ensemble construction methods when the vote combination method is used. The SEC method achieved significantly better results than GP for all except two of the experiments. The maximum improvements of the SEC approach over DRs generated by GP amount to 2.7% for the minimum and median values. As with the sum combination method, the SEC approach has also shown to be more stable than GP when the vote combination method is used.

Figure 6.2 represents the box plot representation of the results for the vote combination method. For the ensemble size of three DRs, the best median value was achieved by the random selection method when creating 1000 ensembles. The worst median values are achieved by the grow, grow-destroy, instance based, and the probabilistic selection method when creating larger numbers of ensembles. Even for the vote combination method the random and probabilistic selection methods achieve the best results for a medium number of created ensembles. For a larger number of created ensembles, the two ensemble creation methods once again start to overfit on the validation set, and achieve bad results.

When creating ensembles of size five, mostly the same results were achieved as when creating ensembles of size three. However, for this ensemble size the best median value was achieved by the instance based method, followed by the probabilistic selection method when creating 500 ensembles. Unlike for the sum combination method, where the best results for the random selection method and probabilistic selection method were achieved for a larger number of created ensembles, here this is not the case, since the methods perform better when using a smaller or medium sized number of created ensembles. Also, the probabilistic selection method mostly achieved better results than the random selection method, but in most cases without any significant difference. The other two methods, grow and grow-destroy, achieved quite bad results for this ensemble size.

Finally, for the ensemble size of seven DRs the best median value is achieved by the random selection method with 20000 created ensembles, followed by the probabilistic selection method, with the number of created ensembles equalling to 5000. It is interesting to note that, except

Table 6.2: Influence of the ensemble construction methods on the results obtained by SEC with the vote combination method

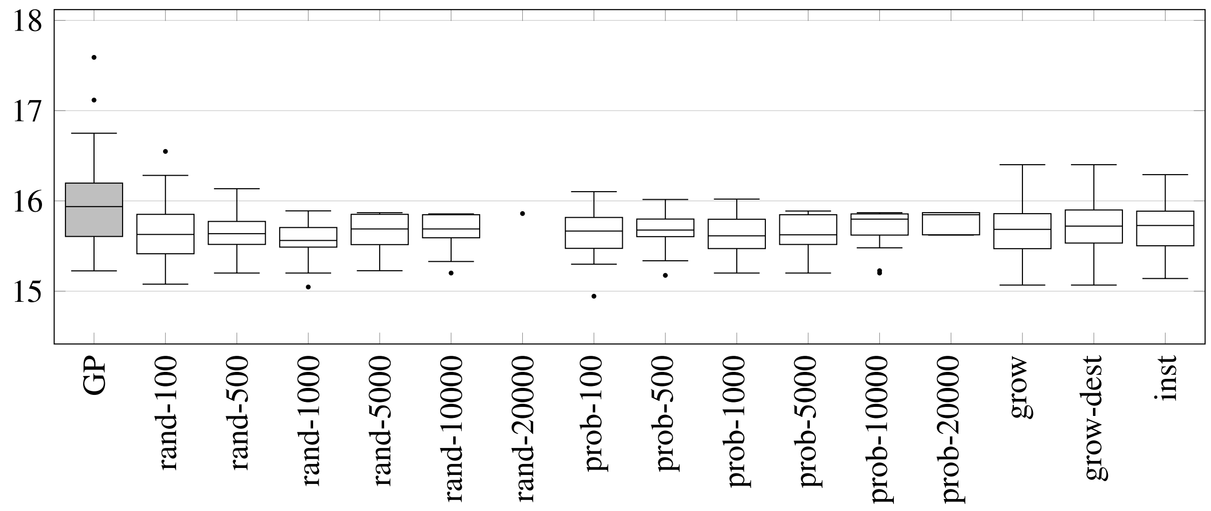
Procedure	Ensemble size								
	3			5			7		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
Rand 100	15.08	15.68	16.55	15.09	15.65	16.51	14.96	15.67	16.19
Rand 500	15.20	15.64	16.14	15.17	15.74	16.40	15.34	16.65	16.13
Rand 1000	15.05	**15.59**	15.89	14.98	14.69	16.32	15.18	15.66	16.22
Rand 5000	15.23	16.69	15.87	15.16	15.72	16.47	15.19	15.64	16.35
Rand 10000	15.20	15.69	**15.86**	15.08	15.66	16.39	15.07	15.67	16.38
Rand 20000	15.86	-	-	**14.91**	15.81	16.32	15.14	**15.55**	**15.92**
Prob 100	**14.94**	15.67	16.10	15.18	15.57	16.17	15.04	15.60	16.42
Prob 500	15.18	15.69	16.02	15.06	15.58	16.14	15.02	15.62	16.06
Prob 1000	15.20	15.63	16.02	15.15	15.71	**16.05**	15.17	15.68	16.35
Prob 5000	15.20	15.66	15.89	14.93	15.69	16.17	15.01	15.57	16.11
Prob 10000	15.20	15.82	15.87	14.95	15.70	16.25	15.10	15.64	16.14
Prob 20000	15.62	15.85	15.87	14.93	15.71	16.32	14.94	15.64	16.30
Grow	15.07	15.69	16.40	15.19	15.70	16.40	**14.82**	15.70	16.22
Grow-dest	15.07	15.73	16.40	15.12	15.69	16.40	15.08	15.65	16.22
Inst	15.14	15.73	16.29	14.99	**15.53**	16.46	15.07	15.66	16.02
GP	15.23	15.96	17.59	15.23	15.96	17.59	15.23	15.96	17.59

for the random selection method with 20000 created ensembles, the increase in the number of created ensembles does not necessarily lead to better results, like it was the case for the sum combination method. However, for this ensemble size, all the tested ensemble construction methods achieved very similar results.

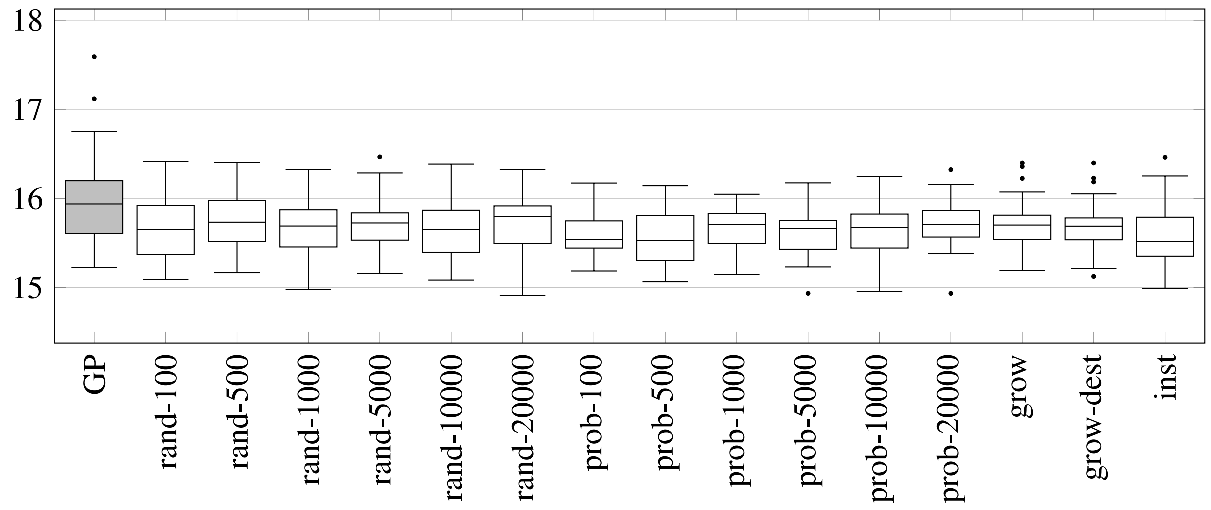
By comparing the results of the vote and sum combination methods, it can be seen that the sum combination method consistently achieves better results than the vote combination method for all ensemble sizes and ensemble creation methods. Even the best median values achieved by the vote combination method were worse than most of the median values achieved by the sum combination method. For example, the best median value achieved by the vote combination method was inferior to the worst median values achieved by the sum combination method for ensemble sizes of five and seven. This clearly denotes that the sum method is better suited for creating ensembles of better performance.

Based on the previously analysed results, it can be concluded that the ensemble construction methods have a significant influence on the quality of the generated ensembles. These differences are more prominent for the sum combination method, while with the vote combination method the results achieved by the various approaches are mostly similar. However, apart from the quality of the evolved ensembles, the different construction methods also differ in the number of the DRs that they create during their execution.

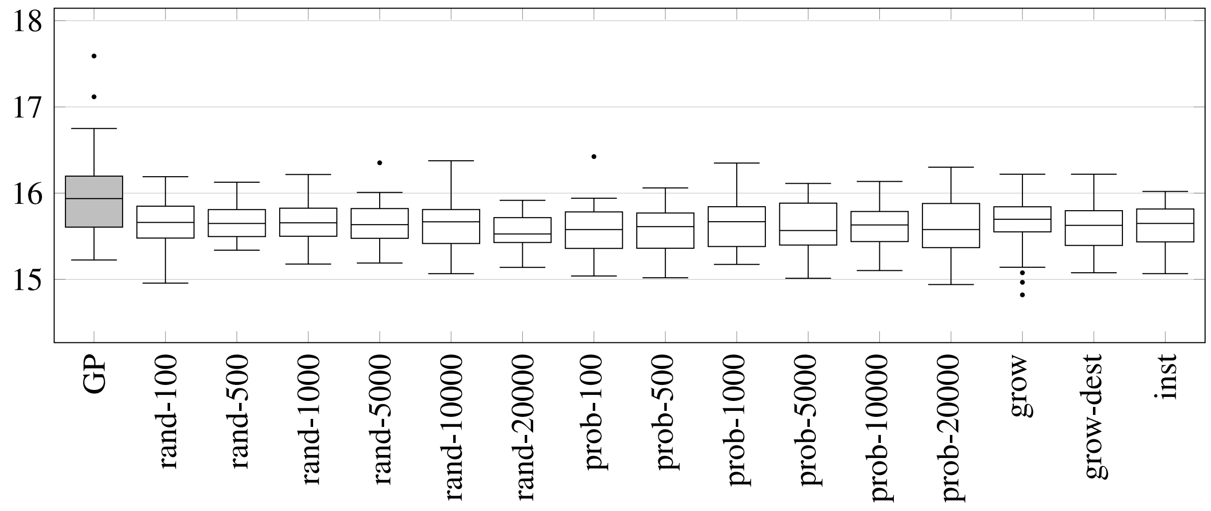
In most cases the best median values of results were achieved by the random selection method. However, for which number of constructed ensembles the random selection procedure will obtain the best results depends largely on the size of the ensemble which needs to be constructed. In most cases for smaller ensemble sizes it is preferred to create a smaller or medium number of ensembles, while as the ensemble size grows the number of constructed ensembles should also increase. For the smaller ensemble sizes it was demonstrated that if a too big number of ensembles are constructed the approach will start to overfit on the validation set. This is expected, since for the ensemble size of three DRs there is not an extensive number of combinations which can be tried out, therefore it is highly probable that the construction method will find the ensemble which is optimal on the validation set. This behaviour should be avoided, since in most cases such an ensemble does not perform well on the test set. The same is also true for the probabilistic selection method. Although the probabilistic selection method tries to enhance the random selection method in a way that the DRs which are selected to form the ensemble are not selected completely by random, but rather depending on their fitness, it did not manage to achieve results which are better than those of the random selection method. The cause for this seems to originate from the fact that ensembles with the best quality usually do not necessarily contain DRs which individually perform quite differently. Therefore, it seems as if the individual quality of the DRs does not represent relevant information which should be used for selecting DRs that should form the ensemble. Because of that reason the probabilistic



(a) Comparison of the ensemble construction methods for ensembles of size three



(b) Comparison of the ensemble construction methods for ensembles of size five



(c) Comparison of the ensemble construction methods for ensembles of size seven

Figure 6.2: Box plot representation of the results obtained by SEC with the vote combination method, when using different ensemble construction methods

selection method does not offer any advantage over the random selection method.

The motivation for the other three proposed methods is to guide the search towards good ensembles, but instead of using the individual fitness values of DRs, the influence of the selected DR on the entire ensemble will be used for guiding the search. Except for the ensemble size of three DRs, these three methods were shown to perform quite well, and were able to outperform the random and probabilistic selection methods in several occasions. The reason why the methods do not perform well on the smaller ensembles is because their search space is very limited, and therefore they will not be able to locate good solutions. Among the three tested methods, the best results were achieved by the instance based method, which even obtains the best results out of all the tested methods for certain parameter values. This is an important indicator which demonstrates that increasing the fitness of the ensemble is not necessarily the most important criteria which should be used to select the DRs which constitute the ensemble. Rather, it is more advantageous to use a procedure which guides the search towards those DRs which perform well on problem instances on which the entire ensemble performs poorly. Thus, the grow and grow-destroy construction methods are more likely to overfit on the validation set, while the instance based method is more resistant to overfitting, since the fitness value is not the main driving force of the search.

In addition, the grow, grow-destroy and instance based methods also evaluate a smaller number of ensembles, which means that they will usually execute faster. For example the grow method will evaluate around 300 ensembles when constructing an ensemble of size seven, while the grow-destroy method will evaluate around 600 ensembles. The instance based method will evaluate even less ensembles, concretely only 7, therefore making it the least expensive out of the tested methods. An additional important thing is that since these methods build the ensembles incrementally, in earlier iterations the ensembles which are constructed will be smaller and therefore evaluated faster. Therefore the execution time of the grow and grow-destroy methods is comparable to the execution time of random selection methods when creating 100 and 500 ensembles, depending on the size of the constructed ensemble. For that reason, if it is crucial that the ensembles are obtained in the smallest possible time, then these three methods should be preferred, since they can obtain good ensembles in a much shorter amount of time.

Since the best results were achieved by the random selection method, the rest of the experiments will use this construction method to generate the ensembles. The random selection method will generate 20000 ensembles, since for that parameter value the best median results were achieved in several cases.

6.3.2 Influence of the method used for the generation of DRs

This section will analyse how the method which is used to generate the DRs influences the performance of the constructed ensembles. The motivation behind this analysis is to determine whether DRs generated by certain methods are more suitable for creating ensembles, or if there is no such correlation. In the results an additional GP method denoted as UGP (unoptimised GP) will be included. This method represents the standard GP procedure which, instead of the optimised parameters, uses unoptimised parameter values which were determined as a rule of thumb. The parameter values for the UGP procedure are the same as those shown in table 4.3, except for the maximum tree depth which was set to 7, and the set of crossover operators which additionally included the one-point crossover operator. The motivation behind testing UGP is to analyse whether generating DRs with a highly optimised GP leads to significant improvements in the results achieved by the ensembles, since parameter optimisation is usually a quite time consuming process. Therefore, if there would be no significant difference between the results of the two methods, then there would be no need to perform the parameter optimisation process.

Table 6.3 represents the results achieved for ensembles of DRs generated by different GP methods. The table shows that the ensembles constructed by the SEC approach in most cases achieved a significantly better performance than the individual DRs. When using DRs which were generated by GP and UGP for the creation of ensembles, then for all experiments the constructed ensembles achieved a significantly better performance than the individual DRs. On the other hand, when DRs generated by GEP are used, then the SEC approach is unable to generate ensembles which outperform those DRs in two occasions, while for DRs generated by DAGP this happens in six occasions. It is also interesting to note that for the Twt and Ft criteria the ensembles created from all DRs outperformed the individual DRs, while on the other hand for the C_{max} criterion the ensembles were in most cases unable to achieve significantly better results than individual DRs. The results also show that the extent of the improvements largely depend on the criterion which is optimised. The smallest improvements in the median values are achieved for the Ft and C_{max} criteria, which amount to around 1%. However, for the Twt and Nwt criteria the SEC approach outperforms the median value of individual DRs up to 6.2% for the Twt criterion, and up 5.4% to for the Nwt criterion.

Figure 6.3 additionally shows the box plot representation of the results. The box plots which represent the results of the individual DRs will be denoted in grey. The figure shows how in many cases the ensembles obtain results which are much less dispersed than those of the individual DRs. This shows that the stability of the approach does not depend on the GP method which was used to generate the DRs. It is also evident that most of the generated ensembles achieve a better performance when compared to the median value of the results obtained by the individual DRs. This shows that the ensemble learning methods are likely to construct ensembles which perform better than most of the individual DRs.

Table 6.3: Influence of the different GP methods on the results obtained by SEC

Approach	Ensemble		*Twt*			*Nwt*			*Ft*			C_{max}		
	Combination	Size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
GP	-	-	15.23	15.96	17.59	7.674	8.107	8.669	158.1	159.3	161.6	38.29	38.70	39.45
	Sum	3	15.59	-	-	7.792	-	-	158.2	-	-	38.70	-	-
		5	**14.84**	**15.12**	**15.76**	7.556	8.064	8.276	157.6	158.7	160.3	38.34	38.51	38.73
		7	14.94	15.30	15.84	7.670	8.091	8.300	**157.5**	158.7	160.4	38.39	38.48	**38.63**
	Vote	3	15.86	-	-	7.586	-	-	157.8	-	-	38.74	-	-
		5	14.91	15.81	16.32	7.699	7.946	**8.212**	157.6	**158.5**	**159.4**	**38.28**	**38.37**	38.71
		7	15.14	15.55	15.92	**7.511**	**7.906**	8.321	**157.5**	158.6	159.6	38.34	38.48	38.79
UGP	-	-	15.25	16.12	20.99	7.532	8.123	8.834	158.4	160.3	163.2	38.46	38.76	39.77
	Sum	3	15.73	-	-	8.107	-	-	158.0	-	-	38.51	-	-
		5	14.86	**15.12**	15.66	7.531	7.837	8.094	**157.3**	158.5	160.5	38.34	38.52	38.63
		7	**14.77**	15.14	**15.61**	7.461	7.693	8.210	157.4	**158.0**	160.8	37.38	38.50	38.66
	Vote	3	15.15	-	-	7.890	-	-	158.6	-	-	38.56	-	-
		5	15.08	15.53	16.05	7.424	7.734	8.151	157.9	158.5	**159.4**	**38.28**	38.46	38.82
		7	15.01	15.36	15.74	**7.399**	**7.687**	**7.934**	157.8	158.7	159.5	38.30	**38.42**	**38.65**

Table 6.3: Influence of the different GP methods on the results obtained by SEC

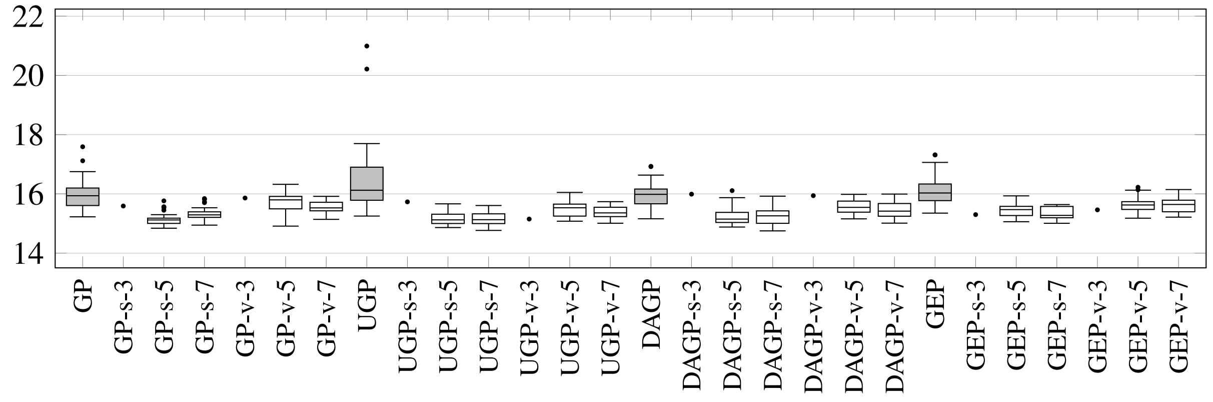
Approach			*Twt*			*Nwt*			*Ft*			C_{max}		
	Ensemble Combination	Size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
DAGP	-	-	15.16	15.99	16.93	7.799	8.027	8.367	157.5	159.9	164.9	38.38	38.68	39.28
	Sum	3	15.99	-	-	7.772	-	-	158.2	-	-	38.79	-	-
		5	14.88	**15.16**	16.11	7.927	8.033	8.222	**156.9**	**158.3**	160.0	38.60	**38.71**	**38.93**
		7	**14.75**	15.26	**15.92**	7.817	8.005	8.216	**156.9**	**158.3**	**159.4**	38.68	38.73	38.99
	Vote	3	15.94	-	-	8.253	-	-	159.0	-	-	38.77	-	-
		5	15.16	15.55	15.98	**7.755**	7.925	8.206	157.1	158.7	159.9	38.44	38.84	38.97
		7	15.01	15.45	15.99	7.765	**7.911**	**8.100**	157.9	158.7	**159.4**	**38.37**	38.77	**38.93**
GEP	-	-	15.35	16.03	17.32	7.631	8.082	8.919	157.8	160.0	162.9	38.26	38.58	38.94
	Sum	3	15.30	-	-	8.012	-	-	159.8	-	-	38.81	-	-
		5	15.06	15.51	15.93	7.840	7.970	**8.118**	157.6	158.9	160.7	38.45	38.66	38.81
		7	**15.01**	**15.28**	**15.64**	7.653	7.867	8.176	**157.4**	**158.3**	159.9	38.44	38.52	38.71
	Vote	3	15.46	-	-	7.760	-	-	159.4	-	-	38.33	-	-
		5	15.18	15.63	16.22	**7.516**	**7.857**	8.156	158.1	159.0	160.0	**38.24**	**38.32**	**38.38**
		7	15.21	15.65	16.15	7.681	7.935	8.123	157.7	158.6	**159.7**	38.25	38.30	38.40

The results show that the SEC method did not achieve the overall best results for all criteria by using DRs generated by only one of the tested GP methods. For the *Twt* criterion the SEC approach achieves the best results when using DRs evolved by standard GP and UGP. For the *Nwt* criterion the best results are achieved by using DRs generated by UGP. On the other hand for the *Ft* criterion the best results are achieved by using DRs generated by DAGP and UGP, while for the C_{max} criterion the best results are achieved when using DRs generated by GEP. However, the differences between the results achieved by the various approaches are usually not significant. The reason why the SEC method performs better for different criteria by using DRs which are generated by different GP methods is probably connected to the complexity of the generated DRs. When optimising the *Twt* and *Nwt* criteria, it seems to be more beneficial to construct ensembles by using more complex DRs, which are usually generated by GP and UGP. On the other hand, for the other two criteria ensembles which are created from simpler DRs, like those generated by GEP and DAGP, usually performed better.

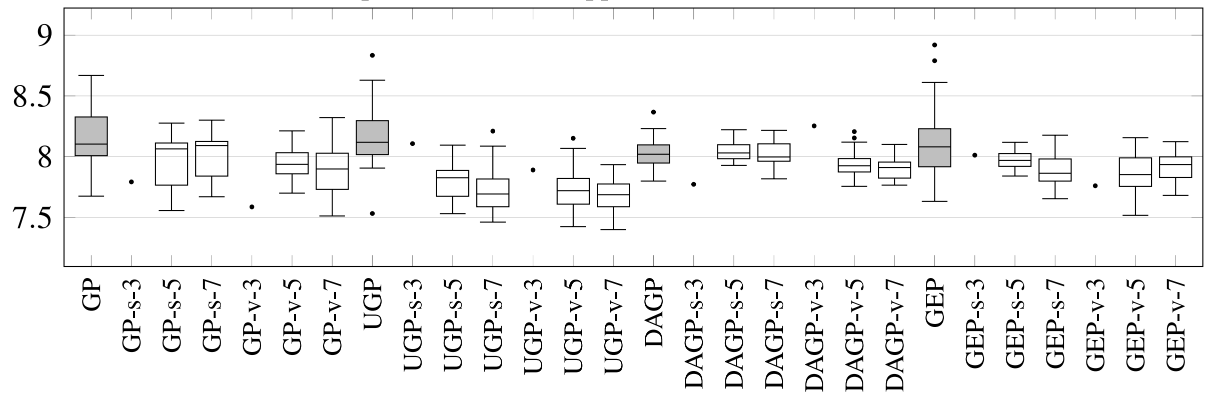
The performance of the SEC approach also depends on the ensemble combination method which is used. However, the results demonstrate that the criterion which is optimised will influence which of the two combination methods will achieve a better performance. For example, the sum combination method performs better on the *Twt* and *Ft* criteria, while the vote combination method performs better when applied for optimising the *Nwt* and C_{max} criteria. The size of the ensemble also has a severe influence on the performance. Usually the best results are achieved by ensembles either of size five or seven. However, the results do not give enough information to conclude which of those two ensemble sizes is preferable for which situations.

Another very interesting thing which the results show is that the ensembles of DRs generated by GP and UGP perform quite similarly. In some occasions the ensembles of DRs generated by UGP perform even better. Thus, the performance of the SEC approach does not depend too much on whether the DRs were created by a highly optimised procedure or not. This is an important fact to consider, since the parameters used for GP were carefully optimised to obtain DRs which achieve the best performance. With this parameter optimisation, GP achieved results with not only a better median value, but which were also less dispersed than the results achieved by UGP. However, the parameter optimisation process which was performed to obtain those results was in itself extremely time consuming. Therefore, even though parameter optimisation is an important step for GP to achieve good results, it seems that this step could be skipped or largely reduced, since SEC will be able to create good ensembles even from unoptimised DRs.

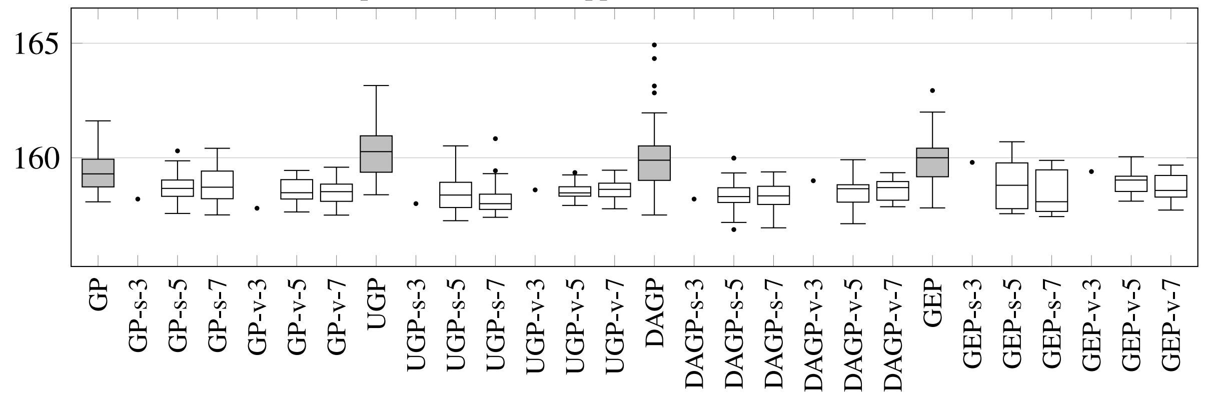
When all things are considered, it can be concluded that the SEC approach has shown to consistently increase the performance over individual DRs. The approach has proven to be more efficient when using DRs which are not constructed by methods which limit the diversity of DRs they create (like DAGP), or methods which generate less complex DRs (like GEP). Therefore, for the construction method of the SEC approach it is better to use a set of DRs



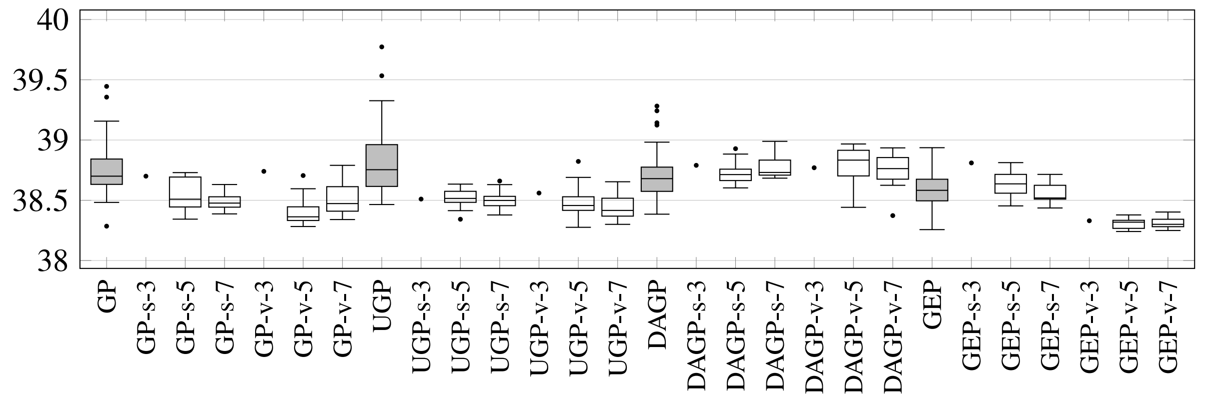
(a) Comparison of the GP approaches for the *Twt* criterion



(b) Comparison of the GP approaches for the *Nwt* criterion



(c) Comparison of the GP approaches for the *Ft* criterion



(d) Comparison of the GP approaches for the C_{max} criterion

Figure 6.3: Box plot representation of the results obtained by SEC, when using DRs evolved by different GP approaches for constructing the ensembles

which consists out of DRs with a lot of variety between them. SEC has also shown to be a very stable and reliable approach, which is evident from the dispersion of the solutions it obtains. This means that the ensembles constructed by SEC have a much higher chance of obtaining better solutions than those obtained by individual DRs.

Since it was not shown that the SEC method performs consistently better by using DRs generated by a single procedure, the DRs generated by GP will be used for creating the ensembles in further experiments.

6.3.3 Influence of the size of the generated DRs

This section will analyse how different sizes of the generated DRs have an influence on the performance of the ensembles. The motivation for this analysis is to determine whether there is a preferred DR size which should be used by the ensembles. In the experiments the SEC approach will use DRs generated by using three different maximum tree depths: three, five and seven. The DRs generated when using the maximum tree depths of three, five and seven will be denoted as GP-3, GP-5, and GP-7, respectively. Since no function node in the primitive set has more than two arguments, the maximum sizes of the DRs which can be generated by these three GP methods consist of 15, 63, and 255 nodes respectively for GP-3, GP-5, and GP-7.

Table 6.4 represents the results achieved by the SEC approach when constructing ensembles from DRs of different sizes. The results show that when using DRs generated by GP-3 the constructed ensembles did not achieve significantly better results in three occasions, while the ensembles constructed from DRs generated by GP-7 did not achieve significantly better results in only one occasion. For the Nwt and C_{max} criteria the ensembles were always able to significantly outperform individual DRs regardless of the size of the generated DRs. The improvements achieved by the ensembles of DRs for the median values of the Ft and C_{max} criteria were again quite small and amount up to around 1%. For the other two criteria the ensembles achieved a maximum improvement of 5.3% for the Twt criterion, and 4.7% for the Nwt criterion.

Figure 6.4 shows the box plot representation of the results achieved by the SEC approach when DRs of different sizes are used. The figure denotes that the SEC approach is more stable than GP, which can be seen from the fact that the results of the ensembles are usually less dispersed than those of the individual DRs. The SEC approach delivers the least dispersed results when DRs of the smallest sizes are used.

Table 6.4: Influence of the maximum tree depth used to generate DRs on the results obtained by SEC

Approach			*Twt*			*Nwt*			*Ft*			C_{max}		
	Ensemble Combination	Size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
GP-3	-	-	15.26	16.23	17.23	7.780	8.185	8.663	158.7	160.0	161.9	38.31	38.84	39.18
	Sum	3	15.37	-	-	7.878	-	-	159.9	-	-	38.34	-	-
		5	**15.16**	**15.39**	15.97	7.730	7.976	**8.033**	159.7	160.0	160.5	38.34	38.39	38.47
		7	15.23	15.50	**15.66**	7.962	8.001	8.098	**159.0**	**159.2**	159.9	38.31	38.38	**38.45**
	Vote	3	16.21	-	-	7.638	-	-	159.8	-	-	38.69	-	-
		5	16.03	16.23	16.26	7.645	**7.806**	8.235	**159.0**	**159.2**	**159.7**	38.27	38.37	38.47
		7	15.75	16.23	16.46	**7.572**	7.914	8.302	**159.0**	159.4	159.9	**38.24**	**38.34**	38.48
GP-5	-	-	15.23	15.96	17.59	7.674	8.107	8.669	158.1	159.3	161.6	38.29	38.70	39.45
	Sum	3	15.59	-	-	7.792	-	-	158.2	-	-	38.70	-	-
		5	**14.84**	**15.12**	**15.76**	7.556	8.064	8.276	157.6	158.7	160.3	38.34	38.51	38.73
		7	14.94	15.30	15.84	7.670	8.091	8.300	157.5	158.7	160.4	38.39	38.48	**38.63**
	Vote	3	15.86	-	-	7.586	-	-	157.8	-	-	38.74	-	-
		5	14.91	15.81	16.32	7.699	7.946	**8.212**	157.6	**158.5**	**159.4**	**38.28**	**38.37**	38.71
		7	15.14	15.55	15.92	**7.511**	**7.906**	8.321	**157.5**	158.6	159.6	38.34	38.48	38.79

Table 6.4: Influence of the maximum tree depth used to generate DRs on the results obtained by SEC

Approach			*Twt*			*Nwt*			*Ft*			C_{max}		
	Ensemble Combination	Size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
GP-7	-	-	15.28	16.18	20.51	7.727	8.131	8.906	158.6	160.6	163.0	38.38	38.85	39.41
	Sum	3	15.77	-	-	7.795	-	-	159.7	-	-	38.37	-	-
		5	14.96	15.58	16.03	**7.499**	7.765	7.884	**158.2**	159.6	161.3	38.37	38.61	**38.63**
		7	**14.91**	15.58	16.04	7.557	7.751	**7.978**	**158.2**	160.5	161.4	38.37	38.62	38.73
	Vote	3	15.51	-	-	7.669	-	-	159.4	-	-	38.43	-	-
		5	15.00	15.45	16.06	7.540	**7.750**	8.007	158.5	159.4	160.5	**38.32**	38.62	38.80
		7	15.03	**15.33**	**15.91**	7.508	7.774	8.165	158.3	**159.3**	**160.2**	38.35	**38.57**	38.89

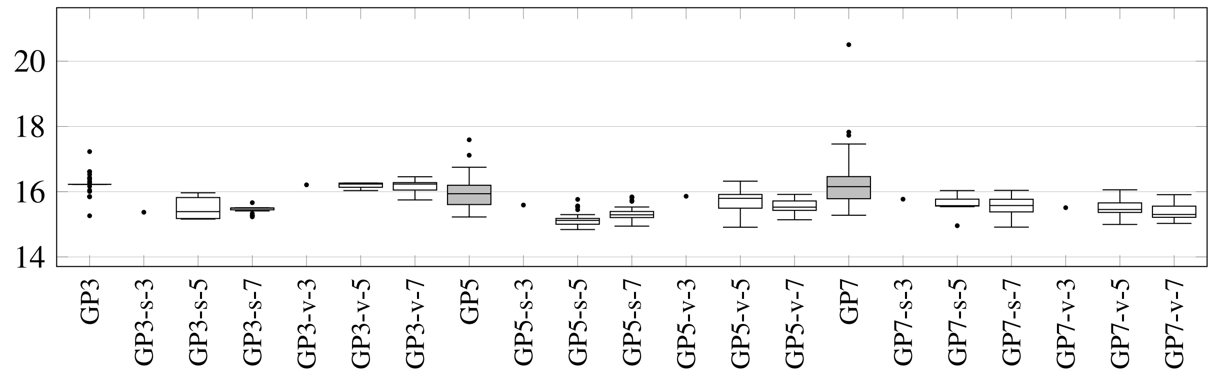
The results show that for different criteria DRs of different sizes are more appropriate. Therefore, for the C_{max} criterion the best results were achieved by using DRs of the smallest size, which were generated by the GP-3 method. This again shows that for this criterion less complex DRs are more suitable and can achieve better performance. The best results for the Twt and Ft criteria were achieved by using the GP-5 method, while the GP-7 method achieved the best results for the Nwt criterion. An interesting thing which can also be noticed is how the performance of the ensemble combination methods also depends on the maximum depth of the DRs. For the Twt criterion the performance of the vote combination method increases with the increase of the depth used to generate the DRs, however for the sum method this is not the case, since GP-5 achieves the best results. For the Nwt criterion it is evident that for both combination methods the results improve as the maximum depth used for generating the DRs increases. For the Ft criterion the best results for both combination methods are achieved for ensembles of DRs generated by GP-5, which suggests that DRs of medium complexity are preferred for this criterion. Finally, for the C_{max} criterion the performance of both combination methods deteriorates with the increase of the DR size.

6.4 Results obtained by different ensemble learning methods

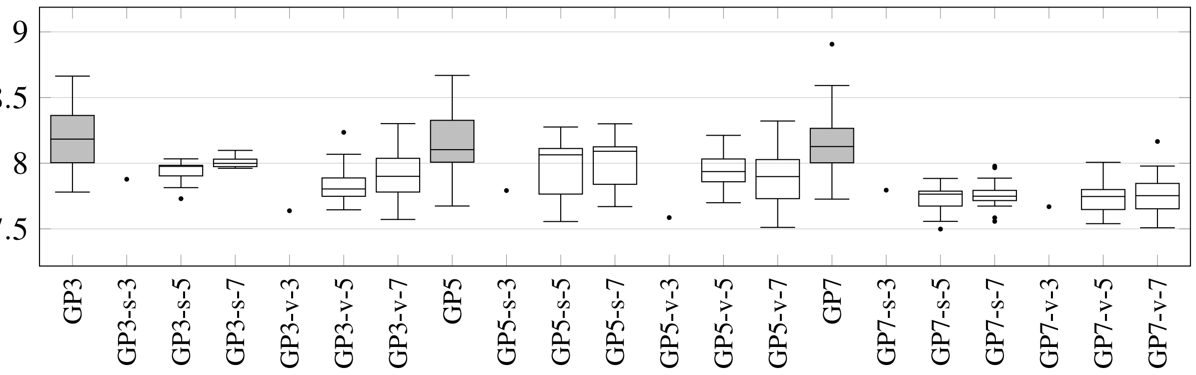
In this section the results achieved by ensembles of DRs, created by different ensemble learning approaches, will be presented. First the results for each individual approach on the Twt criterion will be presented, and the influence of the ensemble size and ensemble combination method on the performance of the approaches will be analysed. In the last section the methods will be compared with each other on four different scheduling criteria.

ESS will be applied on ensembles obtained by all approaches to determine whether by using this method it is possible to further improve the performance of the obtained ensembles. In the tables which represent the results obtained by ESS, the underlined experiments denote those for which ESS performs significantly better than the ensembles of the same size which was obtained without ESS. This means that, for example, the ensembles of size five constructed by ESS from ensembles generated by BagGP, will be compared to ensembles of size five constructed by BagGP. This comparison will show whether for a concrete ensemble size ESS can construct better ensembles.

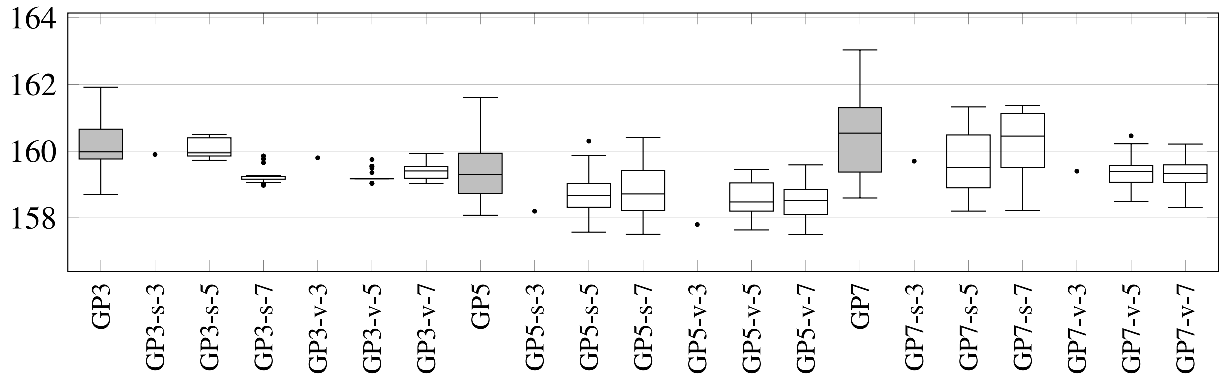
In addition, where possible, it was tested if there is a statistical difference between ensembles of sizes two, five, and ten when not using ESS, and sizes of two, five, and nine when using ESS. In such a way it is possible to obtain a rough estimation of the performance of smaller, medium, and larger ensembles, since testing the significant difference between all sizes would be too time consuming.



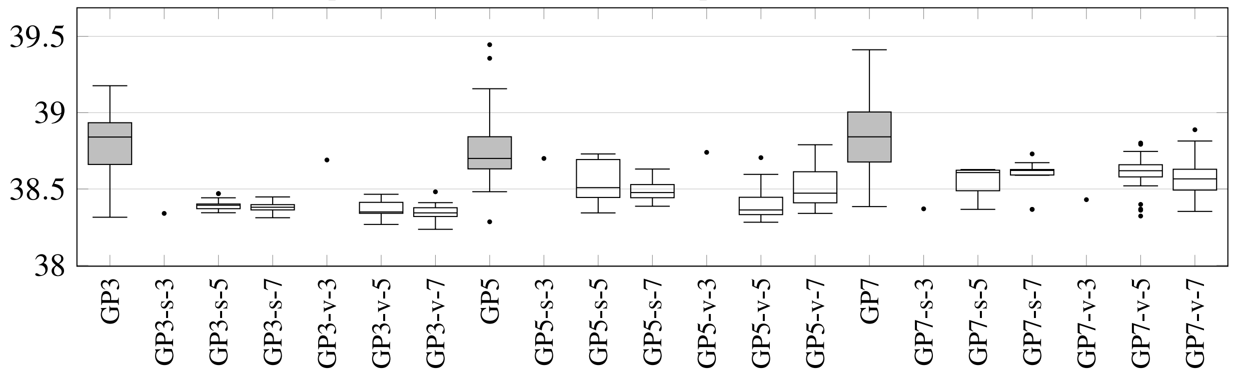
(a) Comparison of the maximum tree depths for the *Twt* criterion



(b) Comparison of the maximum tree depths for the *Nwt* criterion



(c) Comparison of the maximum tree depths for the *Ft* criterion



(d) Comparison of the maximum tree depths for the C_{max} criterion

Figure 6.4: Box plot representation of the results obtained by SEC, when using DRs evolved by varius maximum tree depths to construct the ensembles

6.4.1 Results obtained by the SEC approach

Table 6.5 represents the results obtained by the SEC approach for different ensemble sizes. For the experiments, DRs generated by the standard GP procedure with the optimised parameters were used. The random selection method which generates 20000 ensembles was applied for the ensembles construction. In order to eliminate the need for another problem instance set which would be used by ESS, the same validation set which was used by SEC to create the ensemble is also used by ESS to construct the ensemble subsets.

The experiments show that by using both ensemble combination methods the SEC approach achieved significantly better results than individual DRs. The sum combination method achieved significantly better results in all cases, whereas the vote combination method was unable to do so for the ensemble sizes of four and five DRs. With the sum combination method the maximum improvement of 5.1% for the median value was achieved over individual DRs. On the other hand, the vote combination method achieved a maximum improvement of only 2.5% for the median value. It is interesting to observe the results which are achieved by SEC for the two smallest ensemble sizes, for which an exhaustive search on the validation set was performed. The ensemble which was generated for the size two, by using the vote combination method, is shown to perform equally well as the best DR evolved by GP, while for the sum combination method the obtained ensemble performed even better than the best evolved DR. However, when the ensemble size increases to three DRs, the results deteriorate for both combination methods, although the obtained results are still better than the median value achieved by all evolved individual DRs. Therefore, the ensemble of size three seems to overfit more easily on the validation set than the ensemble of size two, due it its larger expressiveness. Based on these results it can be concluded that the SEC approach can easily construct ensembles which achieve superior performance over the individual DRs.

Figure 6.5 shows the box plot representation of the results. The figure shows that the sum method consistently achieved better median values for all the tested ensemble sizes. Both methods achieve much better solution distributions than the individual DRs. This is not only evident from the fact that the ensembles achieve much smaller maximum values than the individual DRs, but also from the fact that for many experiments the maximum values achieved by the ensembles was smaller or equal to the median value of the individual DRs. Furthermore, when creating ensembles of four and five DRs by using the sum combination method, the median values achieved by the SEC approach were smaller than the minimum value of the solutions achieved by GP. This means that for these parameters at least 50% of the solutions obtained by SEC will outperform the best individual DR. This demonstrates that the SEC approach has a high probability of obtaining better solutions than GP. Between the two combination methods, the sum method achieves less dispersed results than the vote combination method in most of the experiments. The sum combination method achieves better results when using medium sized

Table 6.5: Results obtained by the SEC approach

	Sum			Vote		
Ensemble size	Min	Med	Max	Min	Med	Max
2	15.17	-	-	15.23	-	-
3	15.59	-	-	15.86	-	-
4	14.92	15.18	15.93	15.20	15.87	16.23
5	**14.84**	**15.12**	15.76	**14.91**	15.81	16.32
6	14.89	15.55	15.89	15.17	15.70	16.04
7	14.94	15.30	15.84	15.14	15.55	**15.92**
8	14.88	15.37	15.86	15.02	15.81	16.43
9	15.18	15.32	**15.70**	15.20	**15.54**	16.06
10	15.10	15.29	16.07	15.15	15.65	16.35
GP	15.23	15.96	17.59	15.23	15.96	17.59

ensembles, while on the other hand the vote combination method usually performs better for ensembles consisting out of a large number of DRs. Since in the sum combination method each DR has an influence on the priorities of all job-machine pairs, it is harder to find a larger ensemble of DRs which perform well under all situations. On the other hand, in the vote combination method, each DR casts only a vote for the best element, therefore it is more beneficial to have a larger ensemble of DRs for the approach to be more stable and reduce the number of ties. When comparing ensembles of sizes five and ten for the sum combination method, statistically better results were achieved when using ensembles of size five.

The ensembles created by the SEC approach will be used by ESS to determine if it is possible to create ensemble subsets which have perform better than the entire ensemble. For that

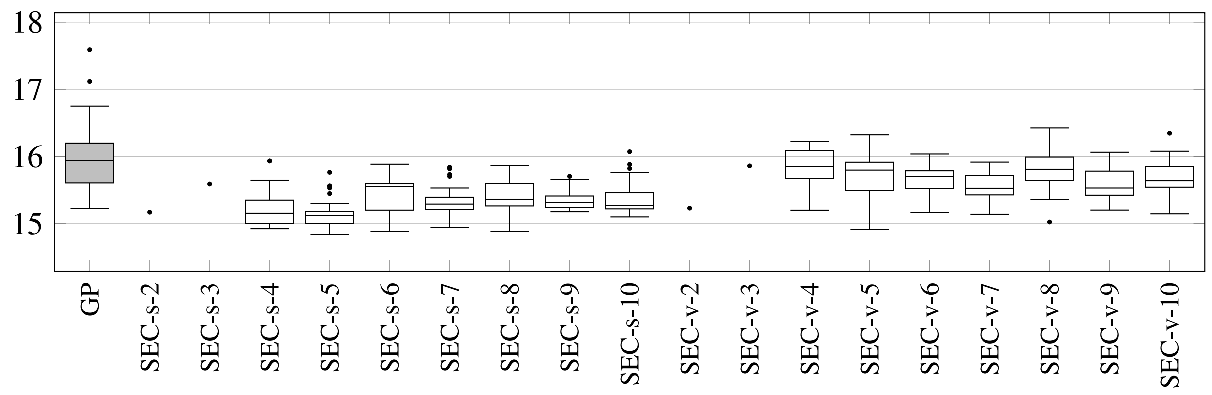


Figure 6.5: Box plot representation of the results obtained by the SEC approach

Table 6.6: Results obtained by ESS when applied on ensembles of size five generated by SEC

	Sum			Vote		
Ensemble subset size	Min	Med	Max	Min	Med	Max
2	15.00	15.25	16.57	15.42	16.20	17.31
3	14.88	**15.21**	15.99	15.33	15.69	**16.10**
4	**14.49**	15.24	**15.76**	**15.17**	**15.65**	16.25
GP	15.23	15.96	17.59	15.23	15.96	17.59

purpose the ensembles of size five and ten will be used. Ensembles of size five are used since they achieved the best median value, while on the other hand the ensembles of size ten are used since by using that ensemble size the most ensemble subsets can be constructed by ESS.

Table 6.6 represents the results which were achieved by ESS when using ensembles of size five generated by SEC. The results show that ESS achieved significantly better results for all experiments, except when the vote combination method is used with the ensemble subset size of two DRs. Unfortunately, the results show that ESS was unable to obtain significantly better results than ensembles of the same size which were obtained by SEC. Nevertheless, ESS still performs quite well when compared to individual DRs, since it achieved better median values by at most 4.5% for the sum combination method, and 1.9% for the vote combination method. ESS also obtained a single solution which performs better than any of the solutions obtained by the SEC method, and which outperforms the best DR generated by GP by 4.9%. Although this solution represents an outlier, it nevertheless illustrates the extent to which the ensembles can improve the performance of DRs.

Figure 6.6 shows the box plot representation of the results for ESS. The results for the ensembles of size five created by SEC are also included in the figure to enable an easier comparison between the results obtained by ESS and SEC. For the sum combination method ESS achieves similar results as ensembles generated by SEC for the ensemble size of five DRs, with the exception of the ensemble subset consisting of two DRs. For the vote combination method ESS achieved better median values for ensemble sizes of three and four DRs. The median values for those two experiments were even better than that achieved by ensembles of size five created by SEC, although their minimum values were worse.

Table 6.7 represents the results of ESS when applied on ensembles of size ten created by SEC. For the sum combination method, all results are significantly better than those of the individual DRs. On the other hand, for the vote combination method all results are significantly better except for ensemble sizes of two and seven. The maximum improvements for the median values amount to around 4.2% for the sum combination method, and 2.3% for the vote combi-

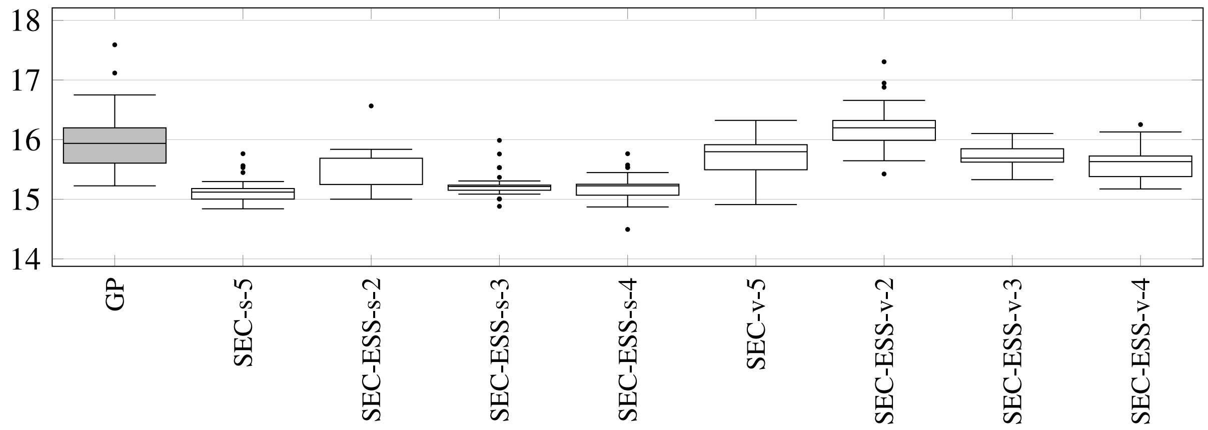


Figure 6.6: Box plot representation of the results obtained by ESS when applied on ensembles of size five generated by SEC

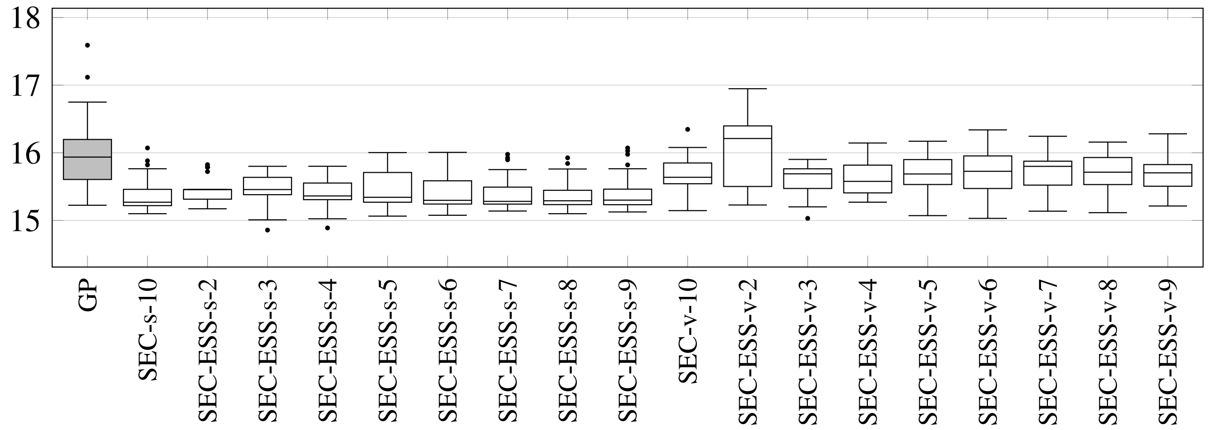


Figure 6.7: Box plot representation of the results obtained by ESS when applied on ensembles of size ten generated by SEC

nation method. However, ESS did not achieve significantly better results than SEC of the same ensemble sizes, except for one occasion (underlined in the table). For the sum combination method there is no statistical difference of the results obtained by ensembles of sizes two, five and nine. On the other hand, for the vote combination method, ensembles consisting of five and nine DRs achieve significantly better results than ensembles of size two.

Figure 6.7 shows the box plot representation of the results when ESS is applied on ensembles of size ten. The figure denotes that the sum combination method consistently achieves better results than the vote combination method. However, the difference in results between the ensemble subsets generated by ESS and ensembles generated by SEC is not large, but rather marginal for most ensemble sizes.

Table 6.7: Results obtained by ESS when applied on ensembles of size ten generated by SEC

	Sum			Vote		
Ensemble subset size	Min	Med	Max	Min	Med	Max
2	15.17	15.46	15.83	15.23	16.22	16.95
3	**14.86**	15.51	**15.80**	**15.03**	15.69	**15.90**
4	14.89	15.36	**15.80**	15.27	**15.59**	16.15
5	15.06	15.35	16.00	15.07	15.71	16.17
6	15.08	15.31	16.01	**15.03**	15.74	16.34
7	15.14	**15.29**	15.98	15.14	15.80	16.24
8	15.10	15.30	15.93	15.12	15.74	16.16
9	15.13	15.31	16.07	15.21	15.71	16.28
GP	15.23	15.96	17.59	15.23	15.96	17.59

6.4.2 Results obtained by the BagGP approach

In this section the results for the BagGP approach will be presented. In addition to testing the influence of the size of the ensemble and ensemble combination method, the influence of the sampled data set size will also be analysed. For that purpose, six bag sizes will be used: 30, 40, 50, 60, 70, and 80.

Table 6.8 represents the results achieved by BagGP for various bag sizes. The first thing which can be noticed is that for the sum combination method the ensembles generated by BagGP did not achieve significantly better results than the individual DRs evolved by GP. On the other hand, for the vote combination method, BagGP achieved significantly better results, usually when ensembles of larger sizes were used. The sum combination method rarely achieved a better median value than the DRs generated by GP, while on the other hand the vote combination method usually achieved better median values. The maximum improvement achieved by BagGP over the individual DRs is 3.6% for the median value.

Regarding the influence of the applied bag sizes, the results show that this parameter also has an influence on the performance of the generated ensembles. The quality of the ensembles generally improves as the bag size increases. Both ensemble combination methods achieved the best results when the largest bag size is used to generate the training set. This behaviour is expected since each DR which forms the ensemble will be evolved by using a larger and probably more diverse set of problems instances, which should cause that the individual DRs perform better even on their own. Since the DRs which form the ensemble are evolved independently from each other, it is more probable that a good ensemble will be constructed if the individual

Table 6.8: Results obtained by the BagGP approach

	Bag size																	
	30			40			50			60			70			80		
	Sum ensemble construction																	
Ensemble size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
1	15.42	16.62	18.91	15.39	16.64	20.25	15.12	16.48	18.05	15.25	16.42	22.05	15.16	16.45	21.89	15.42	16.33	18.95
2	15.18	16.45	18.71	15.10	16.15	18.97	15.29	16.26	18.05	15.12	16.07	17.09	**14.96**	16.28	20.82	15.02	16.15	18.44
3	15.18	16.35	19.43	14.91	16.38	21.14	15.19	**16.08**	20.31	15.40	15.99	17.09	15.04	16.11	18.85	15.00	16.04	18.23
4	15.59	16.33	18.12	15.27	16.20	21.39	**15.03**	16.31	**17.90**	15.35	**15.89**	**16.84**	15.13	16.05	20.15	15.00	15.99	17.31
5	15.18	16.29	**17.88**	15.30	16.32	19.70	**15.03**	16.32	**17.90**	15.38	15.90	18.48	15.21	16.04	19.09	14.93	15.99	17.97
6	15.26	16.27	18.90	15.01	16.16	22.72	15.23	16.29	18.66	15.03	16.15	18.48	15.34	16.05	19.33	**14.89**	15.86	17.31
7	15.23	16.31	18.56	15.31	**16.10**	21.38	15.24	16.41	21.17	15.03	16.12	18.48	15.16	**16.00**	**17.96**	**14.89**	15.93	17.31
8	**15.01**	16.22	18.16	15.31	**16.10**	**17.77**	15.24	16.30	23.70	**15.00**	16.06	19.04	15.17	**16.00**	18.02	14.91	15.85	**17.24**
9	15.17	16.21	18.20	**14.77**	16.20	17.84	15.20	16.32	23.70	15.06	16.10	19.04	15.36	16.02	22.95	14.91	**15.77**	17.26
10	15.16	**16.13**	19.95	14.78	16.14	17.84	15.12	16.11	23.70	15.07	16.13	19.06	15.11	16.07	21.41	15.01	15.88	17.26
	Vote ensemble construction																	
Ensemble size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
1	15.42	16.62	18.91	15.39	16.64	20.25	15.12	16.48	18.15	15.25	16.42	22.05	15.16	16.45	21.89	15.42	16.33	18.95
2	15.31	16.42	19.17	15.13	16.79	21.15	15.56	16.22	17.70	15.28	16.19	24.96	15.13	16.38	21.55	15.34	16.31	20.41
3	15.17	15.85	17.58	14.97	15.82	17.30	15.06	15.70	16.94	**15.05**	**15.55**	16.62	15.31	15.81	16.86	15.09	15.60	16.52
4	15.18	16.04	17.15	15.12	16.01	16.71	15.16	15.90	16.93	15.16	15.83	16.39	15.07	15.82	16.73	15.26	15.74	17.03
5	15.33	15.83	16.94	15.27	15.81	16.62	15.18	15.75	16.54	15.22	15.79	16.61	14.95	15.70	16.38	15.23	15.72	16.50
6	15.02	15.81	16.89	15.19	15.82	17.18	15.08	15.72	16.69	15.24	15.74	17.00	15.05	15.68	16.38	15.18	15.57	16.43
7	15.09	15.63	16.67	15.08	15.74	16.54	15.26	15.59	16.31	15.13	15.63	16.54	15.10	15.58	**16.05**	**15.08**	15.49	16.36
8	15.08	15.85	**16.17**	15.04	**15.69**	**16.31**	**15.01**	15.63	16.47	15.24	15.70	16.47	15.15	15.65	16.14	15.13	15.46	16.45
9	15.20	**15.64**	16.73	15.27	15.79	16.51	15.12	15.56	16.58	15.09	15.62	**16.46**	**14.88**	15.58	16.24	15.14	15.51	**16.16**
10	**15.01**	15.66	16.56	**14.95**	**15.69**	16.42	15.04	**15.47**	**16.15**	15.15	15.66	16.47	14.93	**15.57**	16.21	15.11	**15.39**	16.32

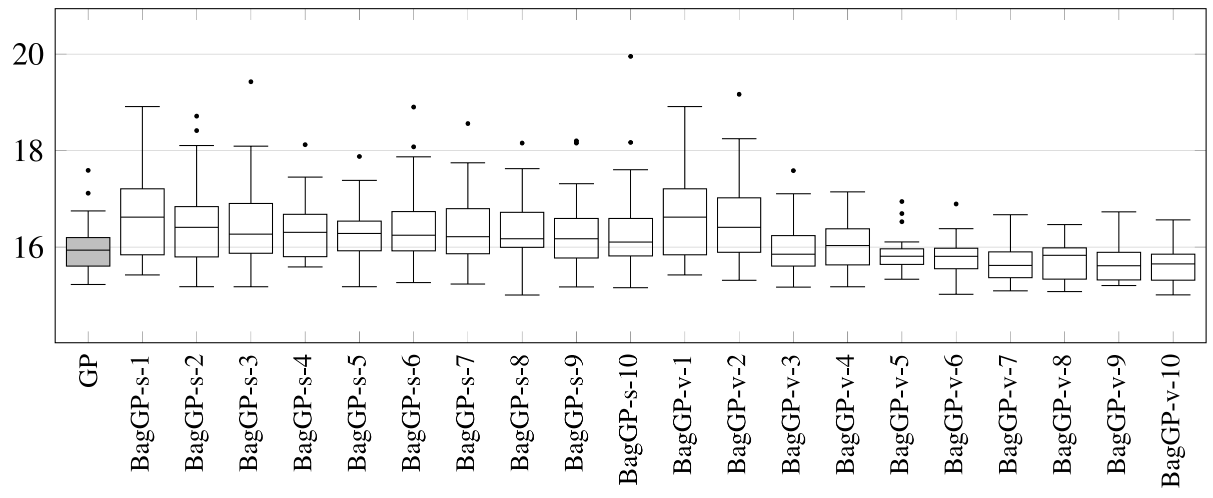
DRs perform well on most of the problem instances. However, increasing the bag size also has a serious drawback, which is that the time needed for evolving the ensembles by BagGP also increases drastically.

Figure 6.8 shows the box plot representation of the results. From the figure it can be observed that the vote combination method consistently achieves better results than the sum combination method. The vote combination method usually does not perform well when an ensemble of size two is used, but this is expected since in this case ties in the vote method will occur very often. In addition, the vote combination method also achieved results which are less dispersed than the results obtained by the sum combination method, which means that the vote combination method is more stable in this case. Regarding the ensemble size for which BagGP achieves the best results, this very much depends on the ensemble combination method which is applied. When the sum combination method is applied, then the best overall median value was achieved for the ensemble of size three. However, for different bag sizes the sum combination method achieved the best results for various sizes. The results which are achieved by the sum method for the different ensemble sizes are usually quite similar. This is also backed up by the fact that for the sum method there was no statistical difference between ensembles of sizes two, five and ten for any of the bag sizes which were used.

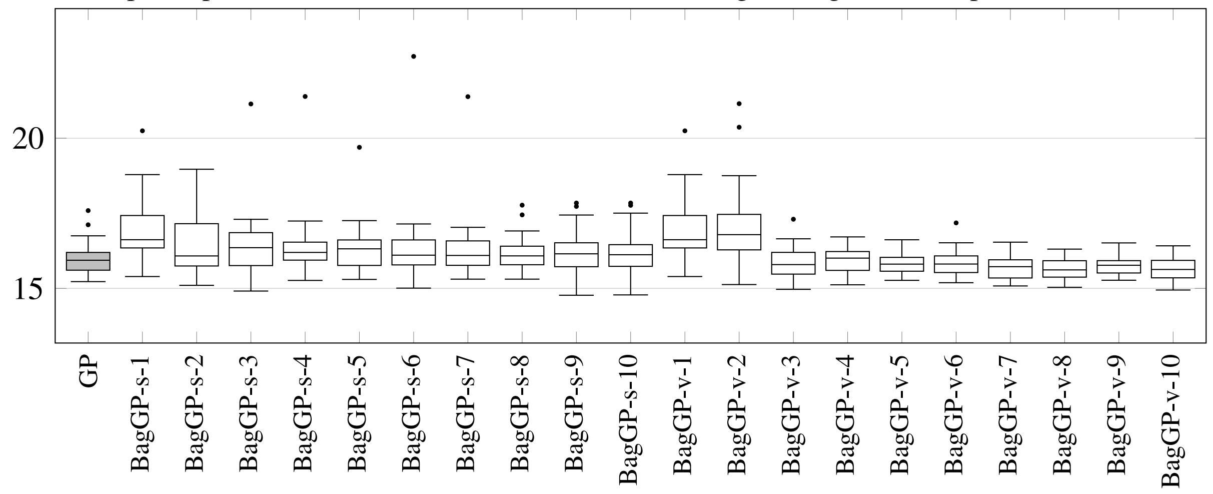
On the other hand, the vote method performs more consistently for different bag sizes, since it always achieved the best results when creating ensembles of larger sizes. The best median value achieved by the vote combination method is obtained for an ensemble of size ten. For the vote combination method ensembles of sizes five and ten always achieve significantly better results than ensembles of size two. When comparing ensembles of sizes five and ten, ensembles of size ten achieved significantly better results for bag sizes of 30, 50, and 80.

The standard BagGP approach will additionally be enhanced with ESS, which will be applied only to ensembles of size 10. Since the number of subsets of ten DRs is not large, an exhaustive search was performed to find the optimal ensemble subset for each ensemble obtained from the 30 runs of the BagGP approach. The results obtained for ESS are shown in Table 6.9. The sum combination method achieves significantly better results for larger bag sizes and smaller to medium ensemble sizes, when compared to the results achieved by the standard GP. In addition to achieving significantly better results for larger ensemble sizes, the vote combination method now also achieves significantly better results even for smaller and medium sized ensembles when using larger bag sizes. The improvements in the median values over the standard GP for the vote method are mostly the same as without using ESS. On the other hand the sum method achieved improvements up to 2.3% for the median value. The best median values for both combination methods were achieved when using the bag size of 80 instances.

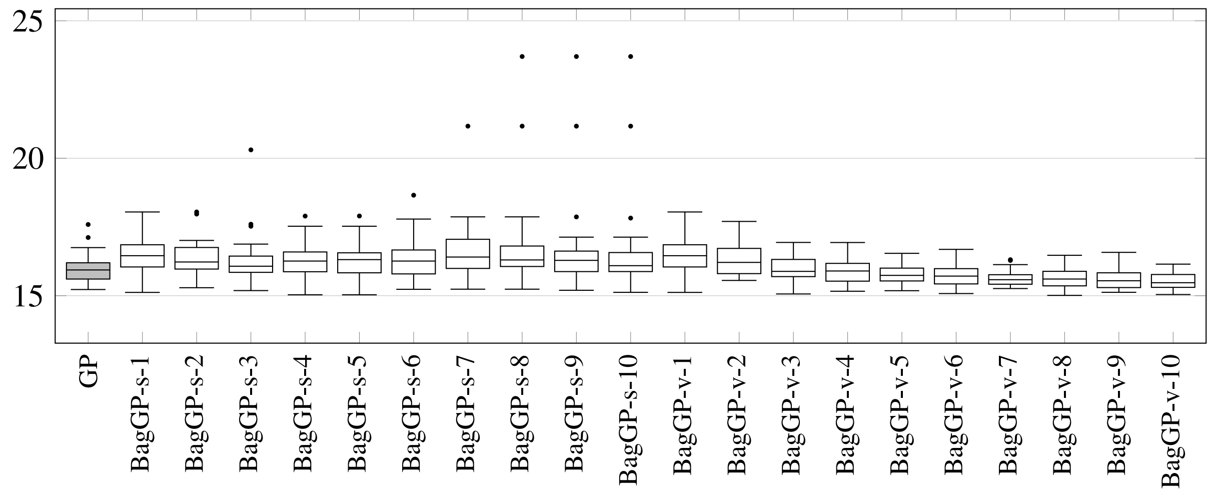
Figure 6.9 shows the box plot representation of the results achieved by ESS. For the sum



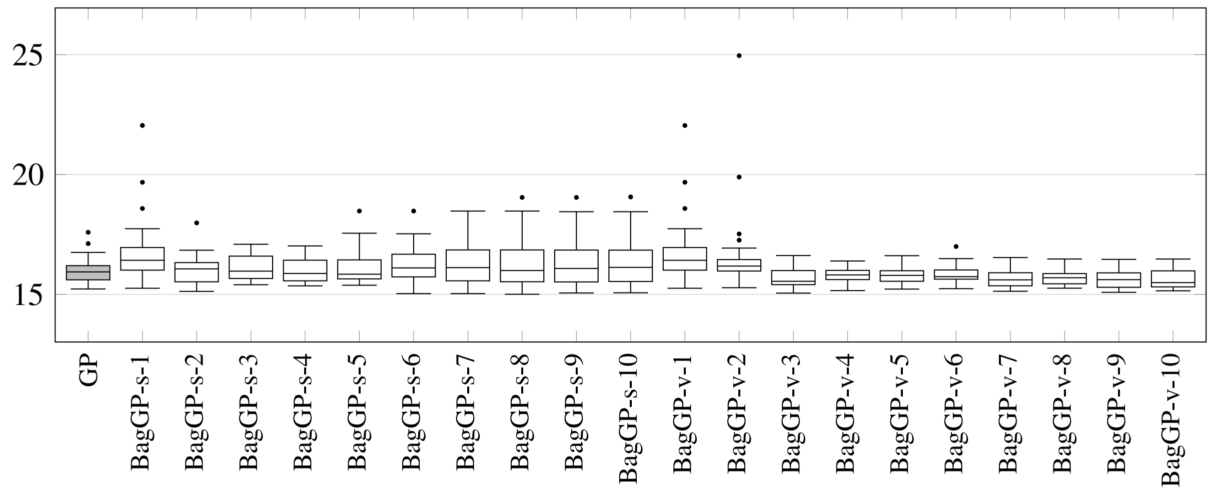
(a) Box plot representation of the results obtained when using the bag size of 30 problem instances



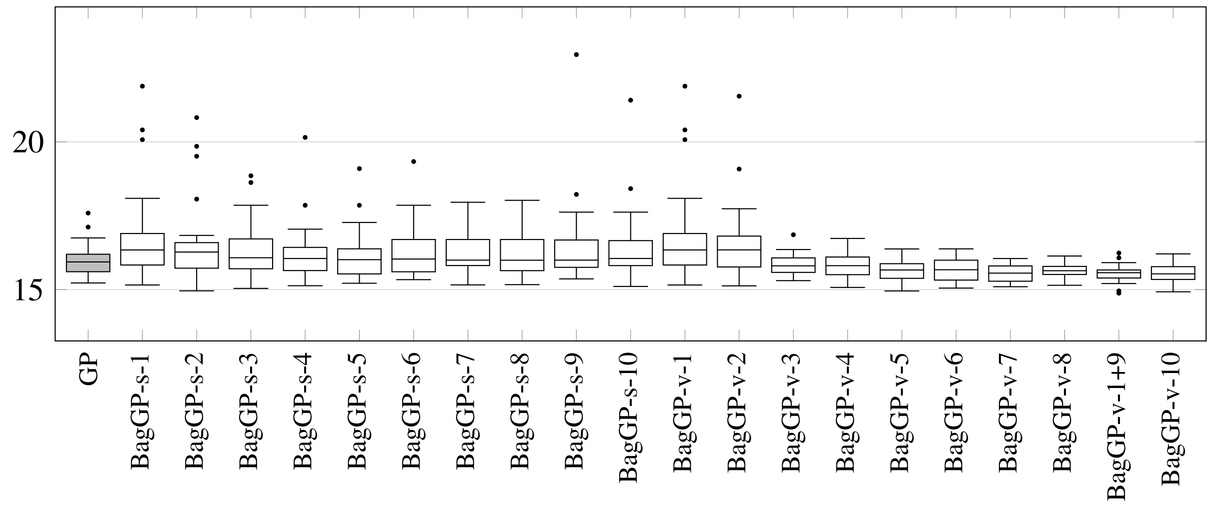
(b) Box plot representation of the results obtained when using the bag size of 40 problem instances



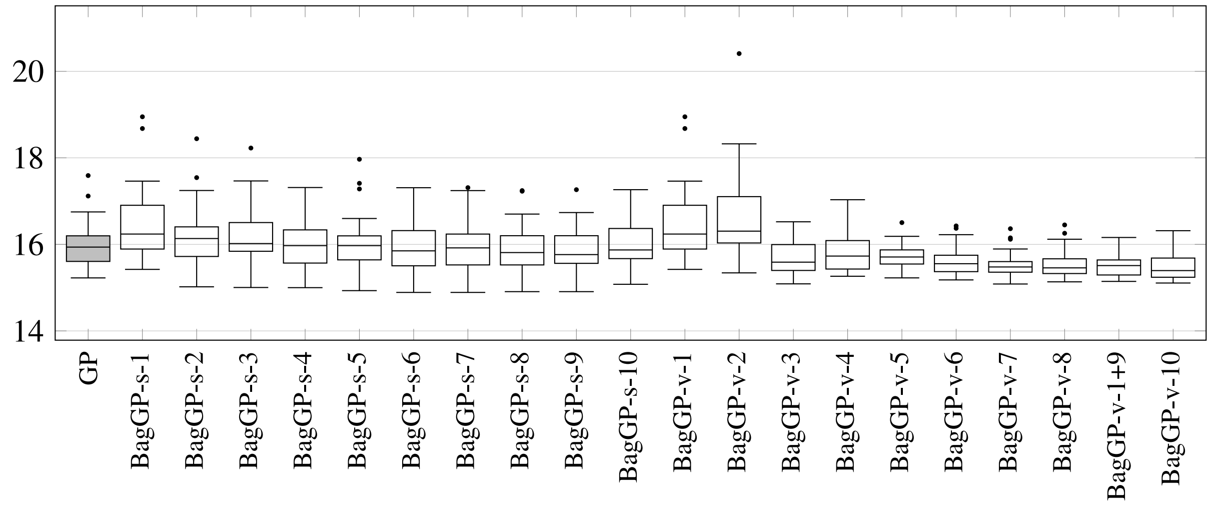
(c) Box plot representation of the results obtained when using the bag size of 50 problem instances



(d) Box plot representation of the results obtained when using the bag size of 60 problem instances



(e) Box plot representation of the results obtained when using the bag size of 70 problem instances



(f) Box plot representation of the results obtained when using the bag size of 80 problem instances

Figure 6.8: Box plot representation of the results obtained by the BagGP approach

Table 6.9: Results obtained by ESS when applied on ensembles of size ten generated by the BagGP approach

	Bag size																	
	30			40			50			60			70			80		
	Sum ensemble construction																	
Ensemble subset size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
2	15.20	**15.90**	17.35	15.10	15.91	18.58	15.33	16.03	17.23	15.30	15.75	17.01	15.12	15.83	**16.64**	15.09	15.84	17.19
3	14.86	**15.90**	**17.02**	**14.94**	**15.74**	**16.90**	15.41	16.22	17.10	**15.02**	15.64	16.73	15.35	15.80	16.74	15.03	15.67	**16.27**
4	**14.41**	16.03	17.54	15.41	15.92	17.79	**14.66**	15.97	**16.88**	15.06	15.70	16.64	15.15	15.65	16.67	**14.81**	**15.59**	17.07
5	14.53	**15.90**	17.61	15.31	15.89	17.81	14.87	**15.75**	**16.88**	15.06	15.63	**16.48**	**14.98**	15.70	16.97	15.05	15.76	17.15
6	15.19	16.99	17.41	14.96	15.97	17.89	15.09	15.95	17.69	15.06	**15.61**	16.52	15.03	**15.64**	18.11	14.89	15.80	17.15
7	14.97	16.10	17.18	15.07	16.01	17.87	14.84	15.93	17.53	15.06	15.70	17.18	15.25	15.66	18.34	14.92	15.75	17.19
8	15.09	16.08	17.54	15.19	16.12	17.65	15.09	16.06	17.01	15.16	15.81	16.82	15.25	15.83	17.59	15.02	15.77	17.30
9	15.04	16.09	19.66	15.17	16.12	17.73	15.12	15.90	17.04	15.05	16.00	17.25	15.33	15.88	18.22	15.16	15.79	17.26
	Vote ensemble construction																	
Ensemble subset size	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
2	15.33	16.10	17.83	14.99	16.16	17.60	15.12	16.14	17.86	15.02	15.94	16.98	15.12	15.92	17.29	15.04	15.88	17.73
3	15.10	15.81	16.98	15.10	15.88	16.61	15.04	15.66	16.87	**14.96**	15.77	16.71	15.01	15.80	16.40	15.14	15.59	16.56
4	15.26	15.93	16.69	**14.85**	15.84	16.73	15.01	15.73	16.78	15.12	15.80	16.65	15.04	**15.52**	16.42	15.07	15.64	16.31
5	15.08	**15.66**	16.81	15.03	15.71	16.72	14.90	15.54	16.57	15.09	15.69	16.37	15.01	15.64	16.29	15.05	**15.41**	16.36
6	15.11	15.83	16.56	15.05	15.72	16.76	15.00	15.64	16.59	15.10	15.68	16.42	15.02	15.58	16.14	**14.94**	15.52	16.50
7	**14.99**	15.73	16.64	15.05	**15.61**	16.72	**14.86**	15.59	**16.38**	15.09	**15.57**	16.49	14.93	15.61	**16.03**	14.99	15.51	**16.10**
8	15.19	15.70	**16.47**	15.17	15.73	16.68	15.07	15.61	16.45	15.07	15.58	16.47	15.03	15.64	16.14	15.03	15.46	16.21
9	15.21	15.67	16.73	15.00	15.67	**16.65**	15.00	**15.53**	16.51	15.06	15.61	**16.31**	**14.85**	15.56	16.10	15.15	15.44	16.16

combination method, ESS usually obtained better results for ensembles of smaller sizes. This means that ESS removed the unnecessary DRs from the ensemble, and consequentially improved its performance. For the vote combination method, the best median values were again achieved for larger and medium sized ensembles. The vote combination method achieved better results in most of the cases, although not as dominantly as when BagGP was used, since now the sum combination method achieves much better results.

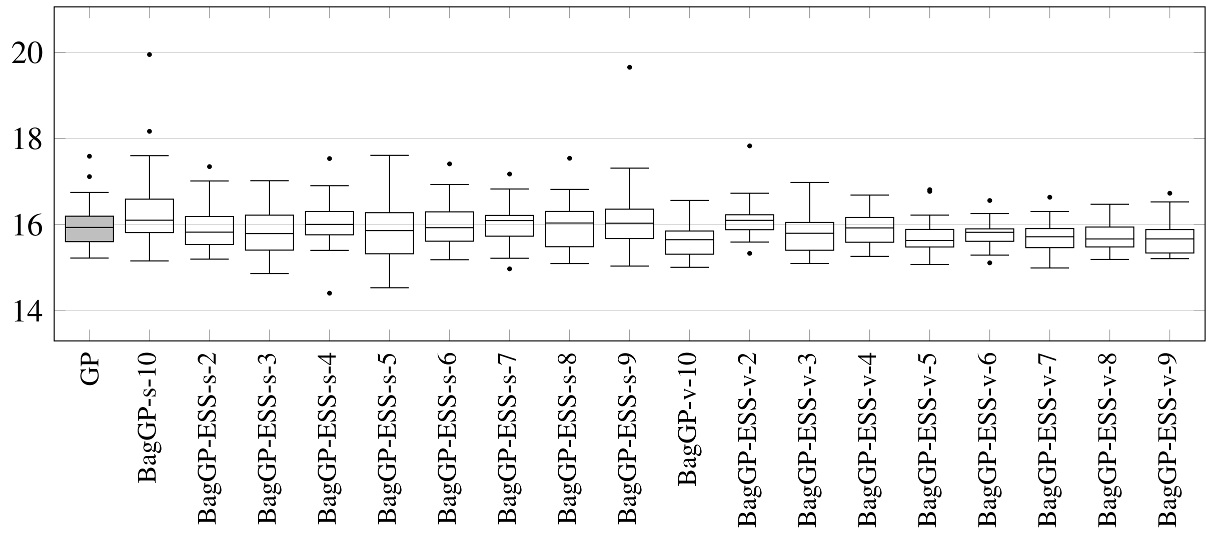
By comparing the results of ESS and the results achieved only by BagGP, it was shown that when using the sum combination method, in 70% of experiments ESS achieved significantly better results than the original ensembles which were generated by BagGP. On the other hand, for the vote combination method, the number of significantly better results achieved by ESS was only 24%. Therefore, ESS should be used with the sum combination method since it can lead to significantly better results, but it can also be used with the vote method since it also leads to certain improvements in the results for this combination method.

6.4.3 Results obtained by the BoostGP approach

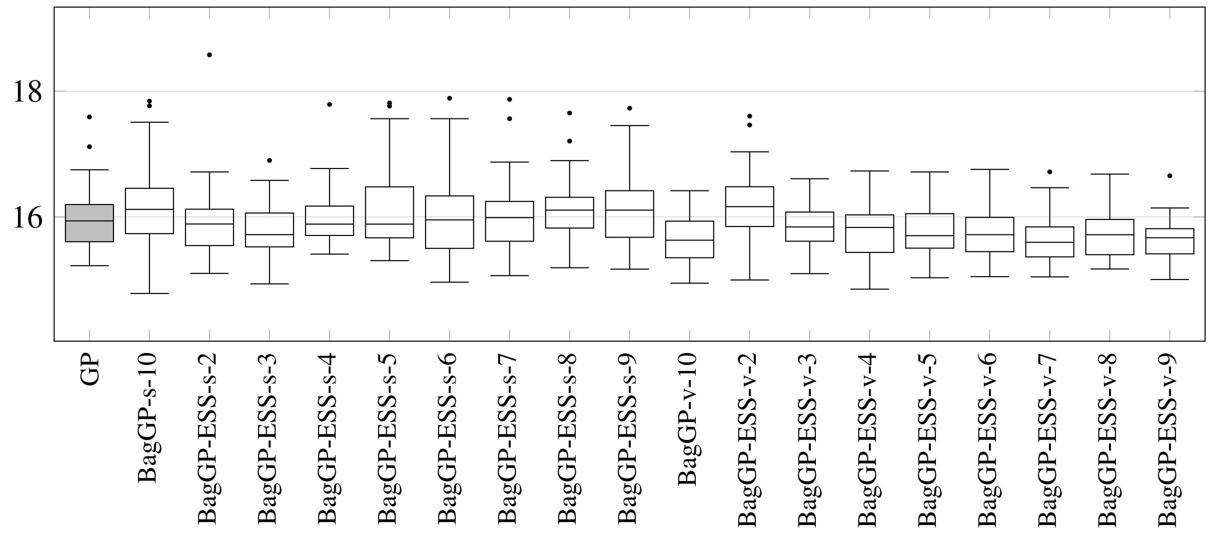
This section will present the results which were achieved by the BoostGP approach. The ensembles will be constructed in two ways, without using the confidences obtained for each DR as weights, and by using those values to additionally denote the weight of each DR. Ensembles evolved by both variants will also be used by ESS to determine the best ensemble subsets.

Table 6.10 represents the results achieved by the unweighted BoostGP approach. The results show that when the sum combination method is used, the BoostGP method is unable to achieve results which are significantly better than those of DRs generated by GP. Nevertheless, BoostGP still achieves better median values. On the other hand, when the vote combination method is used, BoostGP can achieve significantly better results than DRs generated by GP for all ensemble sizes larger than two. The maximum improvements for the median values which can be achieved amount to around 1.3% for the sum combination method, and 2.9% for the vote combination method. In addition, BoostGP achieves less dispersed results when the vote combination method is used.

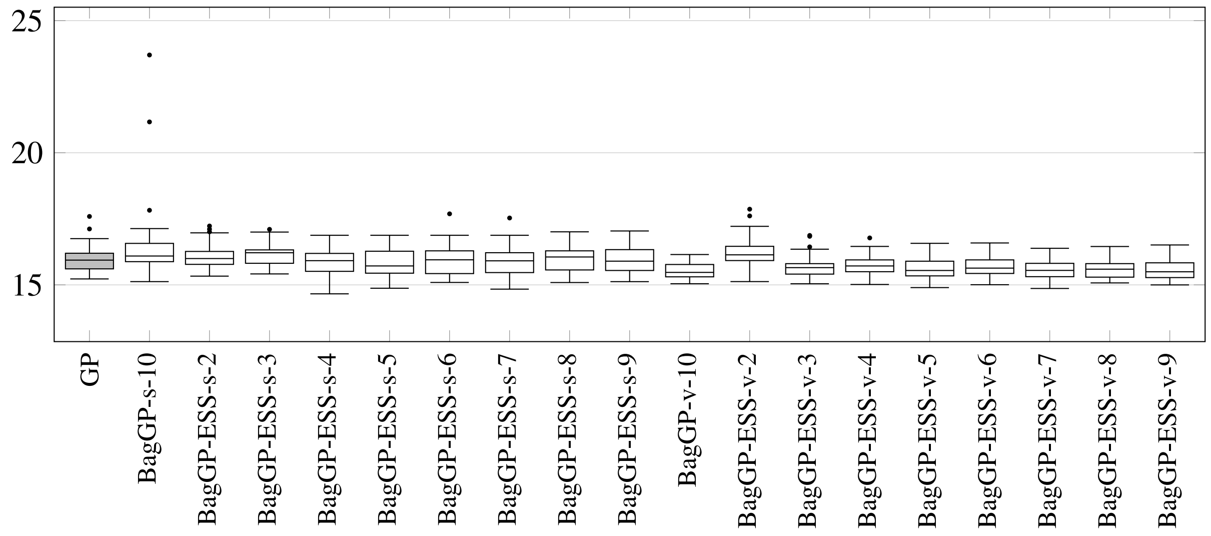
Figure 6.10 shows the box plot representation of the results for the unweighted BoostGP approach. The figure clearly shows that BoostGP consistently performs better when the vote combination method is used. Furthermore, with the vote combination method the achieved results are even less dispersed than when using the sum combination method. Although the best median value for the sum combination method is achieved by evolving ensembles of three DRs, the size of the ensemble has not shown to have a significant impact on the results, since there was no significant difference for the results achieved between ensembles of sizes two, five and ten. On the other hand, for the vote combination method ensembles of sizes five and ten achieve significantly better results than the ensembles of size two.



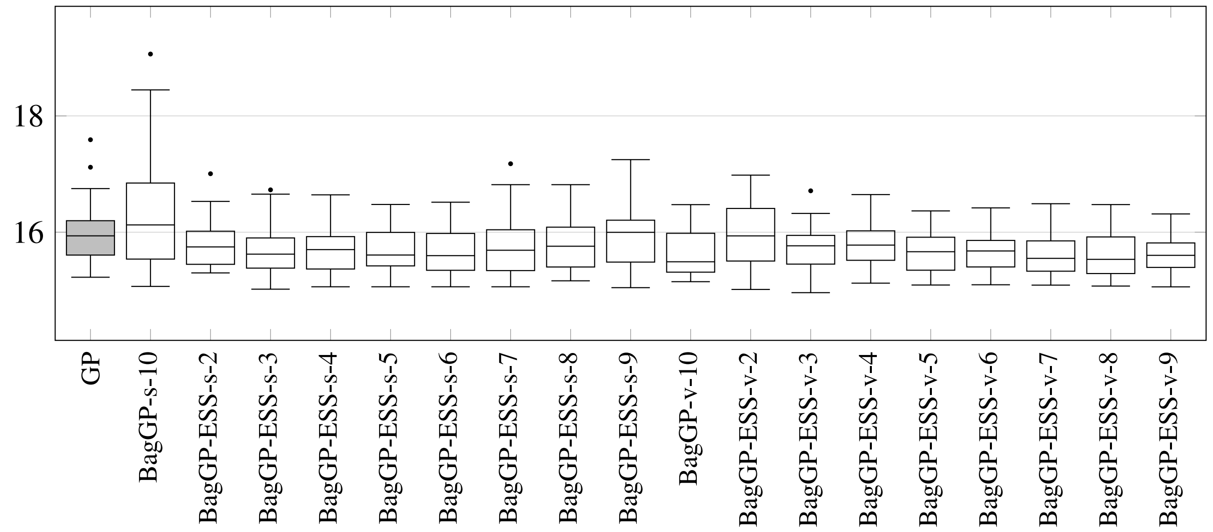
(a) Box plot representation of the results obtained when using the bag size of 30 problem instances



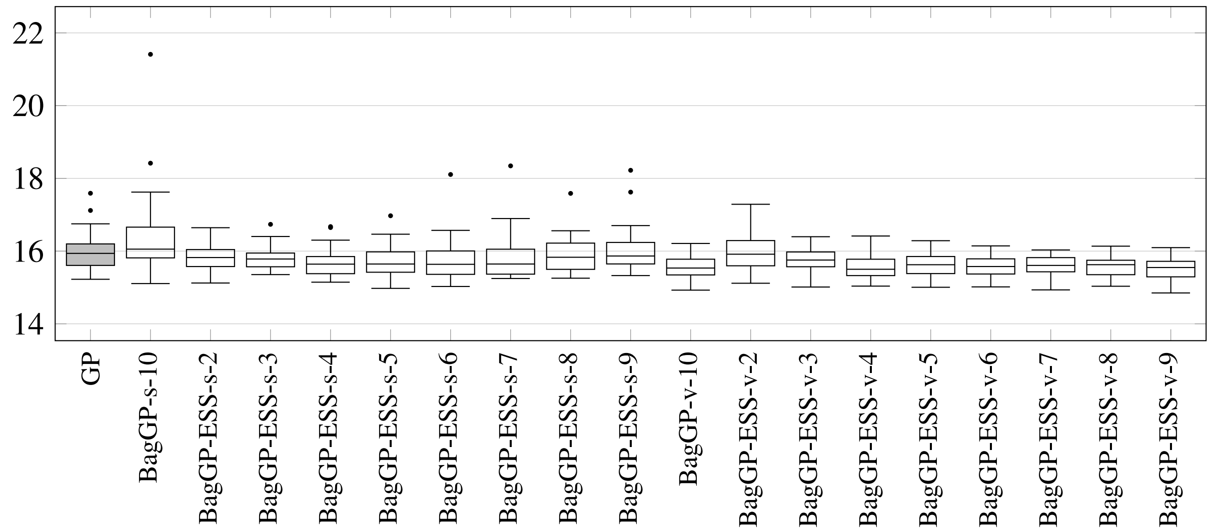
(b) Box plot representation of the results obtained when using the bag size of 40 problem instances



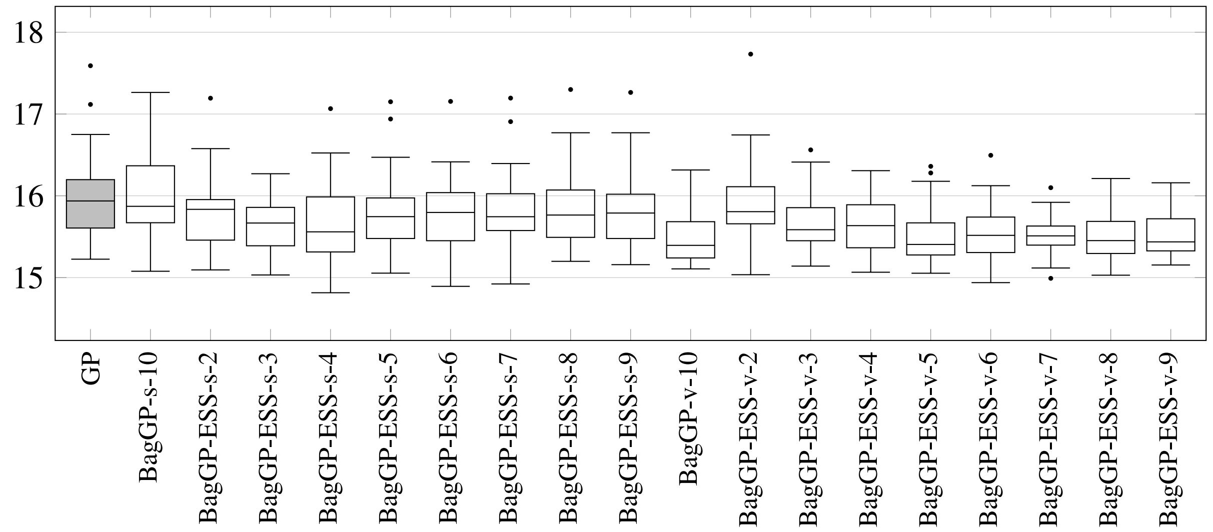
(c) Box plot representation of the results obtained when using the bag size of 50 problem instances



(d) Box plot representation of the results obtained when using the bag size of 60 problem instances



(e) Box plot representation of the results obtained when using the bag size of 70 problem instances



(f) Box plot representation of the results obtained when using the bag size of 80 problem instances

Figure 6.9: Box plot representation of the results obtained by ESS when using ensembles of size ten generated by the BagGP approach

Table 6.10: Results obtained by the unweighted BoostGP approach

	Sum			Vote		
Ensemble size	Min	Med	Max	Min	Med	Max
1	15.12	15.97	18.13	15.12	15.97	18.13
2	15.26	15.82	17.33	15.27	16.02	18.01
3	15.28	**15.76**	17.45	15.02	15.72	16.42
4	15.22	15.87	17.45	15.09	15.67	16.52
5	15.37	15.89	17.52	15.08	**15.50**	16.11
6	15.16	15.89	17.81	15.08	15.64	16.14
7	15.14	15.85	**16.42**	**14.99**	15.56	**16.00**
8	15.14	15.86	**16.42**	15.15	15.68	16.05
9	**15.09**	15.86	16.76	15.10	15.53	16.49
10	15.14	15.82	16.78	15.08	15.55	16.24
GP	15.23	15.96	17.59	15.23	15.96	17.59

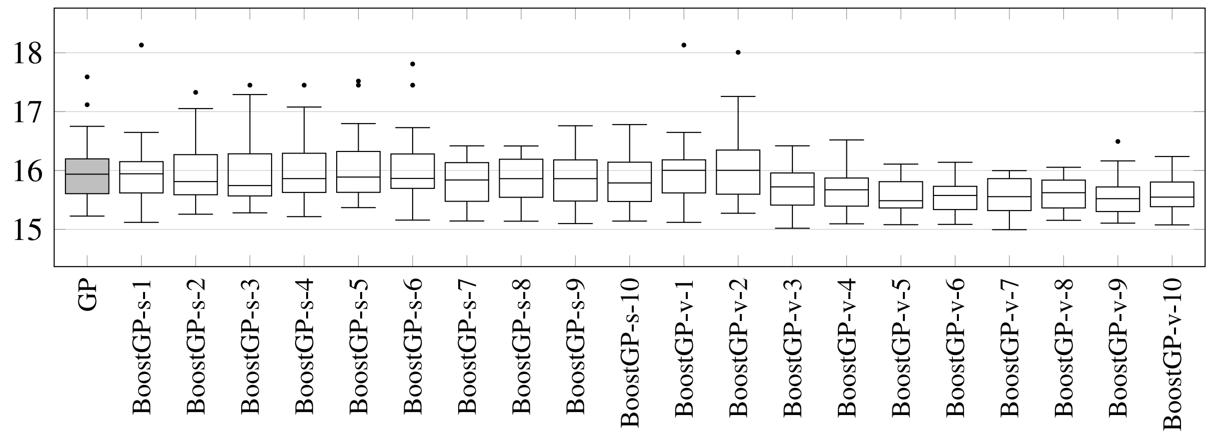


Figure 6.10: Box plot representation of the results obtained by the unweighted BoostGP approach

Table 6.11: Results obtained by ESS when using ensembles of size ten generated by the unweighted BoostGP approach

	Sum			Vote		
Ensemble subset size	Min	Med	Max	Min	Med	Max
2	**14.77**	15.71	16.59	15.10	16.04	16.82
3	**14.77**	15.70	**16.94**	15.02	15.64	16.63
4	14.85	15.66	16.86	15.15	15.63	16.19
5	14.85	**15.61**	16.47	**14.78**	15.60	16.13
6	14.89	15.67	16.39	14.93	15.65	16.25
7	14.79	15.80	16.31	14.98	15.59	16.34
8	14.95	15.79	16.38	15.03	15.62	16.49
9	15.25	15.80	16.42	14.99	**15.54**	**16.11**
GP	15.23	15.96	17.59	15.23	15.96	17.59

Table 6.11 represents the results achieved by ESS when using ensembles of size ten evolved by the BoostGP approach. The results show that the vote combination method can achieve significantly better results than DRs evolved by GP, for all ensemble sizes larger than two. In addition, BoostGP now also achieves significantly better results than DRs generated by GP when using the sum combination method, but only for medium sized ensembles. The maximum improvements in the median values over those achieved by GP are 2.2% for the sum combination method, and 2.6% for the vote combination method. Therefore, ESS was unable to achieve a greater improvement over GP for the median value, when compared to the best improvement achieved by BagGP.

Figure 6.11 shows the box plot representation of the results for ESS when using ensembles generated by the unweighted BoostGP approach. For most of the ensemble sizes the vote combination method once again achieves better median values than the sum combination method. In addition, the results achieved by using the vote combination method are also usually less dispersed. The results show that there is mostly no significant difference between the results achieved for the various ensemble sizes. Between ensemble sizes of two, five and nine DRs there was no significant difference for the sum combination method. However, when the vote combination method is applied, ensembles of sizes five and nine achieved significantly better results than ensembles of size two.

If the results obtained by the ensembles generated by ESS are compared to the ones achieved by BoostGP, it can be observed that although the results for the sum combination method have

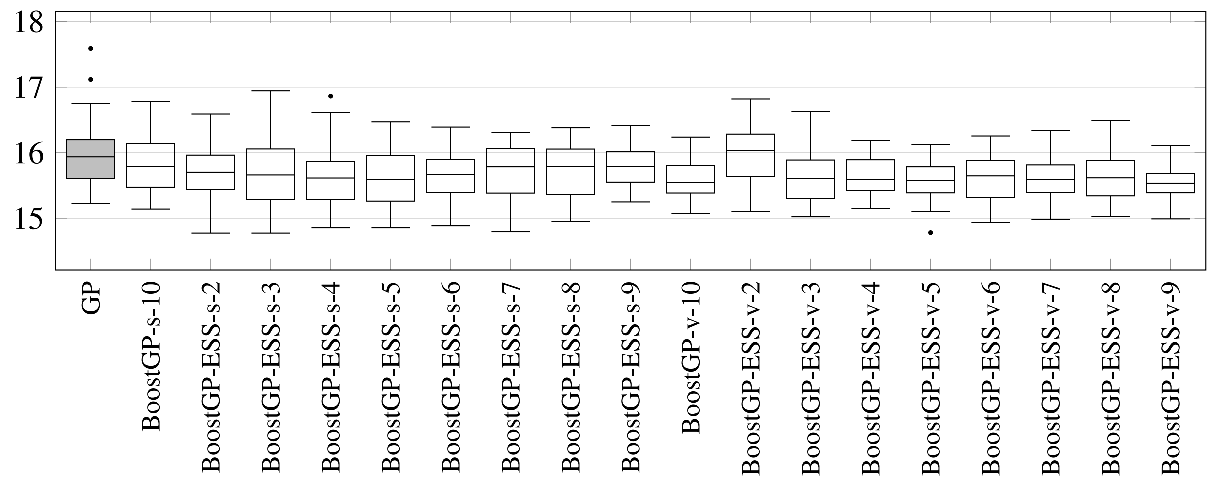


Figure 6.11: Box plot representation of the results obtained by ESS when using ensembles of size ten generated by the unweighted BoostGP approach

slightly improved, for the vote combination method there was no significant improvement in the results when additionally applying ESS. Moreover, a better overall median value for the vote combination method was achieved when using only BoostGP, without the additional application of ESS. Therefore, applying ESS makes sense for ensembles which use the sum combination method to improve their performance and reduce the size of the ensemble. However for the vote combination method applying ESS is useless since it does not lead to improvements in the results.

Table 6.12 represents the results achieved by the weighted BoostGP approach. When the sum combination method was used, BoostGP did not achieve significantly better results than GP, similarly as when the unweighted BoostGP approach was used. On the other hand, when using the vote combination method, significantly better results were achieved for all ensemble sizes larger than three. The maximum improvements which were achieved over the median values of the DRs evolved by GP amount to 1% for the sum combination method, and 2.7% for the vote combination method. By comparing the weighted BoostGP approach with the unweighted variant, it is evident that there is no great difference in the performance of the two variants. Therefore it is better to simply use the unweighted approach since it does not incur any additional complexity by using weights.

Figure 6.12 denotes the box plot representation of the results achieved by the weighted BoostGP approach. The plot shows that BoostGP achieves better median values of the results when the vote combination method is used. In addition, except for the smaller ensemble sizes, the vote combination method achieved less dispersed results than the sum combination method. For the sum combination method there was no significant difference between the results achieved by ensembles of sizes two, five, and ten. On the other hand, for the vote combination method, ensembles of size ten achieved significantly better results than ensembles of

Table 6.12: Results obtained by the weighted BoostGP approach

	Sum			Vote		
Ensemble size	Min	Med	Max	Min	Med	Max
1	15.12	15.97	18.13	15.12	15.97	18.13
2	15.39	15.81	17.01	15.16	15.93	18.28
3	15.30	15.79	17.91	15.10	15.91	17.93
4	15.16	15.90	17.45	15.11	15.64	16.47
5	15.05	15.87	17.61	15.06	15.66	16.48
6	15.05	15.93	17.63	15.17	15.71	**16.30**
7	**14.95**	**15.78**	**16.42**	**14.96**	15.56	16.35
8	15.19	15.81	16.54	15.09	15.57	16.66
9	15.05	15.84	16.54	15.02	**15.53**	16.50
10	15.15	15.82	16.54	**14.96**	15.54	16.45
GP	15.23	15.96	17.59	15.23	15.96	17.59

size two.

Table 6.13 represents the results achieved by ESS when using ensembles evolved by the weighted BoostGP approach. The table shown that in this occasion ESS achieved significantly better results than GP for both ensemble combination methods. The maximum improvements for the median value which can be achieved by ESS over DRs generated by GP amount to around 1.9% for the sum combination method, and 3.3% for the vote combination method. The table also shows that for certain smaller ensemble sizes ESS achieves significantly better results than the weighted BoostGP approach for the same ensemble sizes.

Figure 6.13 denotes the box plot representation of the results achieved by ESS when applied on ensembles generated by the weighted BoostGP approach. Better results are again achieved when the vote combination method is applied. For the sum combination method there was no significant difference between the results achieved by ensembles of sizes two, five and nine. On the other hand, for the vote method, ensembles of size five, and nine achieved significantly better results than ensembles of size two.

By comparing the results obtained by ESS when using ensembles generated by the weighted BoostGP approach, and those obtained without using ESS, it can be concluded that by using ESS it was again possible to achieve significantly better results for the sum combination method. Although for the vote combination method ESS could not obtain significantly better results, it still resulted in a slightly better maximum improvement over the median value of GP. The results

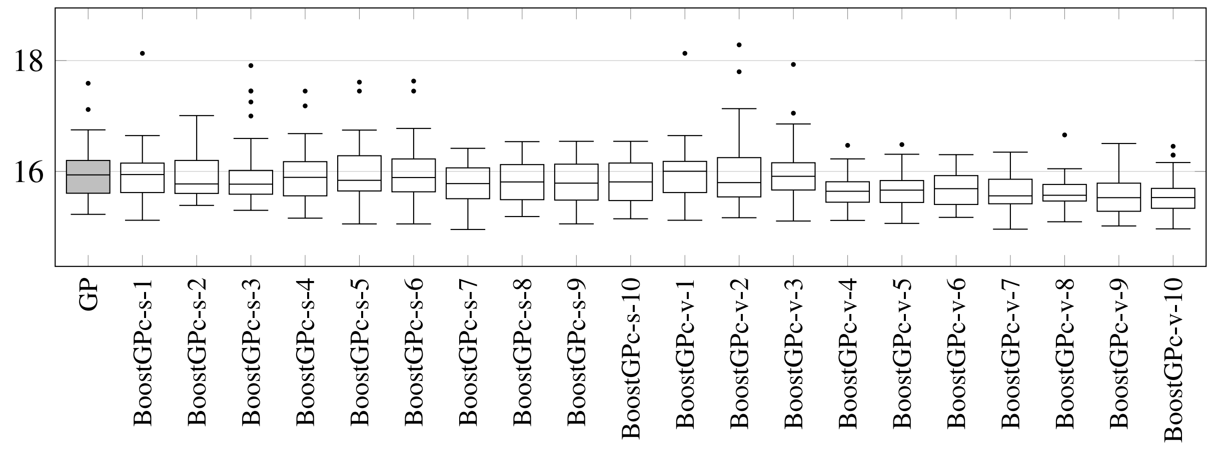


Figure 6.12: Box plot representation of the results obtained by the weighted BoostGP approach

Table 6.13: Results obtained by ESS when using ensembles generated by the weighted BoostGP approach

	Sum			Vote		
Ensemble subset size	Min	Med	Max	Min	Med	Max
2	**14.77**	15.75	16.66	15.10	16.03	16.63
3	15.13	15.71	17.00	15.11	15.72	16.14
4	14.92	**15.64**	16.43	15.09	15.77	16.71
5	14.94	15.72	16.80	14.99	15.62	**16.01**
6	15.05	15.70	16.59	14.97	15.51	16.38
7	15.02	15.70	16.59	15.10	15.58	16.61
8	15.02	15.72	17.31	15.02	**15.42**	16.29
9	14.97	15.74	**16.42**	**14.94**	15.59	16.50
GP	15.23	15.96	17.59	15.23	15.96	17.59

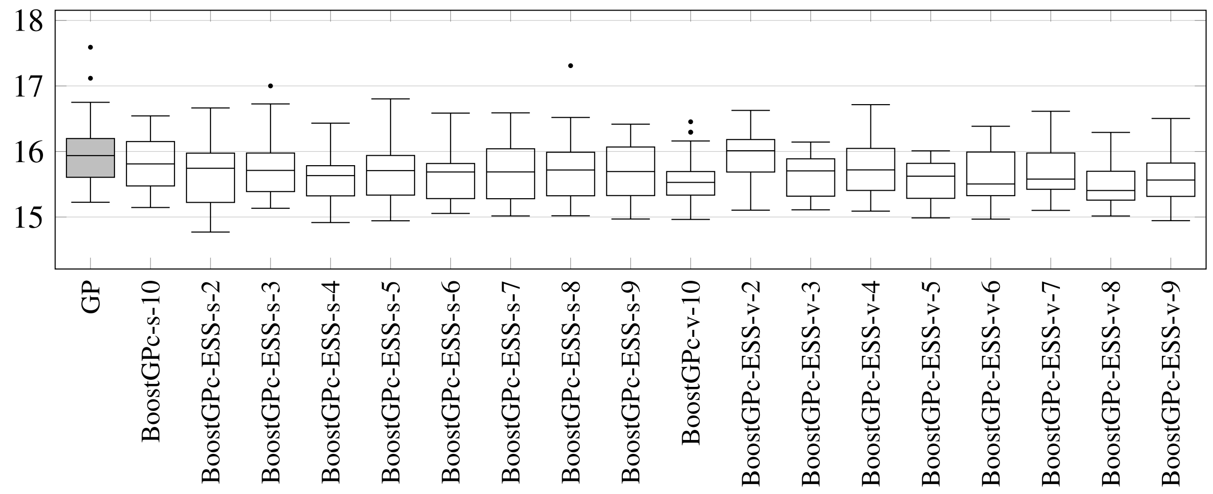


Figure 6.13: Box plot representation of the results obtained by ESS when using ensembles generated by the weighted BoostGP approach

also demonstrate that ESS achieves slightly better results when applied on ensembles generated by the weighted BoostGP approach rather than on the unweighted BoostGP approach. This is evident from the fact that ESS achieves a better overall median value, and also outperforms the results of GP for almost all ensemble sizes.

6.4.4 Results obtained by the cooperative coevolution approach

This section will present the results which were achieved by using the cooperative coevolution approach. The results obtained for this approach are presented in Table 6.14. The results show that the cooperative coevolution approach is unable to significantly outperform the results achieved by GP, not even in one occasion. For smaller ensemble sizes the method achieves results which are close to those achieved by GP, but as the size of the ensemble increases, the results achieved by cooperative coevolution deteriorate, and become even significantly worse than the results achieved by GP.

Figure 6.14 shows the box plot representation of the results for the cooperative coevolution approach. The figure clearly demonstrates that the cooperative coevolution approach achieves more dispersed solutions than GP. In many cases the approach achieved outlier solutions with quite bad performance. Although the approach is more stable when the vote combination method is applied, neither of the two combination methods has shown to consistently outperform the other. The cooperative coevolution approach performs better for both ensemble combination methods when smaller ensemble sizes are used. For both ensemble combination methods, the cooperative coevolution approach achieves significantly better results when using ensembles of size two, than when using ensembles of sizes five and ten.

Table 6.15 represents the results for ESS when applied on ensembles evolved by the cooperative coevolution approach. The results which were obtained by ESS using the ensembles

Table 6.14: Results obtained by the cooperative coevolution approach

	Sum			Vote		
Ensemble size	Min	Med	Max	Min	Med	Max
2	15.21	**16.15**	23.39	15.33	16.33	17.53
3	**15.18**	16.47	**18.80**	**15.14**	**16.01**	**16.96**
4	15.49	16.77	22.57	15.23	16.34	18.50
5	15.83	17.45	23.39	15.76	16.51	18.79
6	15.67	16.79	23.68	15.30	16.90	19.26
7	15.37	17.20	22.55	15.70	16.64	19.79
8	15.83	17.15	25.74	15.62	17.39	18.93
9	15.50	17.48	26.23	16.24	17.90	20.18
10	16.13	17.45	19.66	15.81	16.86	17.86
GP	15.23	15.96	17.59	15.23	15.96	17.59

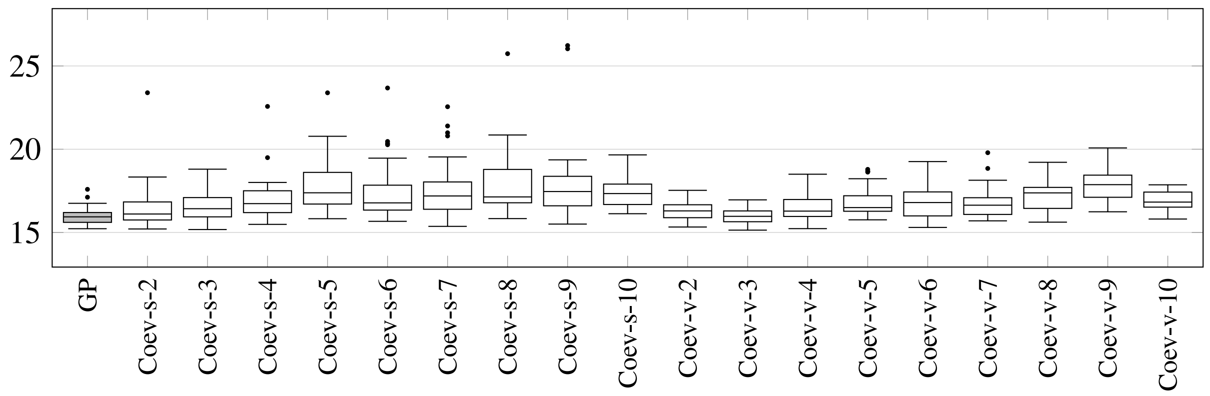


Figure 6.14: Box plot representation of the results obtained by the cooperative coevolution approach

Table 6.15: Results obtained by ESS when using ensembles of size ten generated by the cooperative coevolution approach

	Sum			Vote		
Ensemble subset size	Min	Med	Max	Min	Med	Max
2	**15.80**	19.74	34.80	15.60	18.15	22.80
3	15.86	18.15	28.44	16.28	17.22	19.62
4	15.86	17.76	21.31	15.76	17.32	20.34
5	15.86	**17.30**	20.54	15.50	16.89	19.64
6	15.81	17.57	20.54	**15.39**	**16.73**	**18.55**
7	16.10	17.60	20.28	15.75	16.82	19.11
8	16.10	17.73	**19.52**	15.43	16.76	19.46
9	16.10	17.56	20.71	15.50	16.79	19.29
GP	15.23	15.96	17.59	15.23	15.96	17.59

generated by cooperative coevolution are consistently worse than the results achieved by DRs evolved by GP. The obtained results were also quite dispersed when compared to the results obtained by DRs evolved by GP.

Figure 6.15 represents the box plot representation of the results for ESS when using ensembles generated by cooperative coevolution. The figure shows that ESS mostly achieves poor results, which tend to improve as the number of DRs in the ensemble increases. Such a behaviour is expected, since the individual DRs that form the ensemble have been evolved simultaneously, and therefore each DR was evolved so that it performs well when it is applied together with the other DRs in the ensemble. Therefore, the removal of DRs from the ensemble can disturb its balance, and lead to poor performance of the ensemble. In these experiments ESS has shown to perform significantly better if ensembles of sizes of five and nine DRs are used instead the ensemble size of two DRs.

Since the initial results achieved by the cooperative coevolution approach were extremely bad, another configuration of the cooperative coevolution approach was tried out. In this configuration the number of iterations that will be used to evolve the ensembles will depend on the size of the ensemble which needs to be evolved. Therefore, when an ensemble of size two is evolved, the cooperative coevolution approach will use a number of iterations which is two times larger than normal, therefore it will use 160000 iterations. When ensembles of three DRs are evolved, then the number of iterations will be three times larger, and so on. The motivation behind this configuration is to give the approach more time to evolve ensembles, since it is pos-

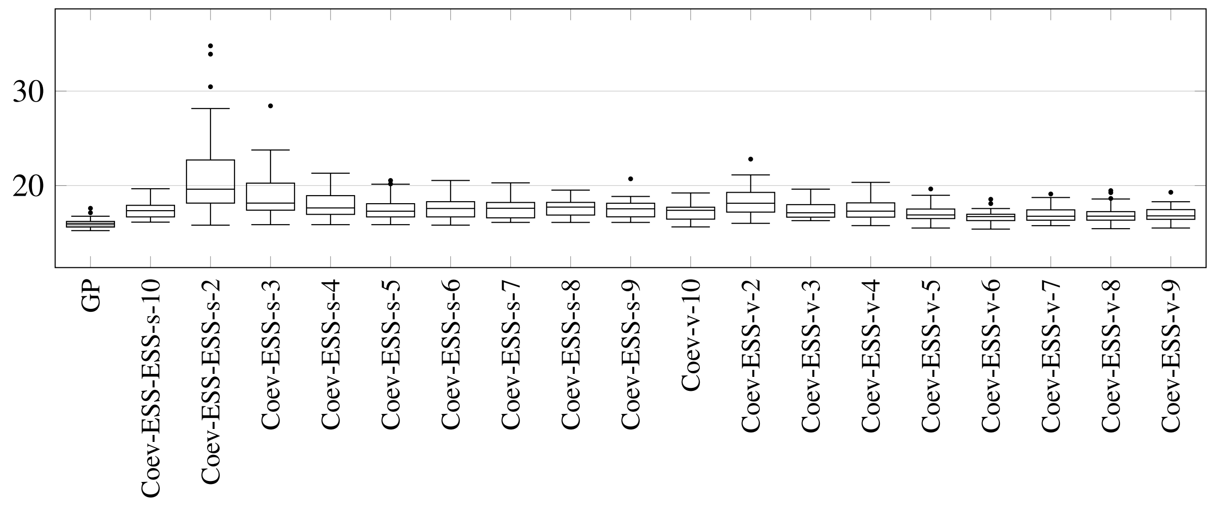


Figure 6.15: Box plot representation of the results obtained by ESS when using ensembles of size ten evolved by the cooperative coevolution approach

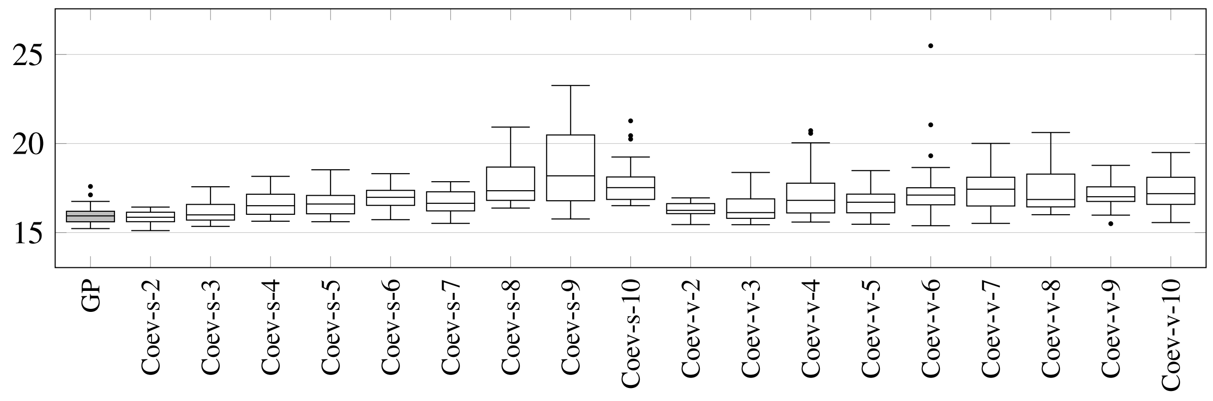


Figure 6.16: Box plot representation of the results obtained by the cooperative coevolution approach with a larger number of iterations

sible that in the first round of experiments the approach was interrupted before it could obtain good solutions.

Table 6.16 represents the results achieved by cooperative coevolution with the larger number of iterations. Once again was the cooperative coevolution approach unable to achieve significantly better results than GP for any of the experiments. The statistical tests show that when using the sum combination method with ensembles of sizes two and three, there was no significant difference between the standard GP and the cooperative coevolution approach. However, in all other experiments the cooperative coevolution achieved significantly worse results than the standard GP.

Figure 6.16 represents the box plot representation of results achieved by the cooperative coevolution approach when the larger number of iterations is used. The figure shows that the best results are achieved when smaller ensemble sizes are used, however as the size of the ensembles increases the results quickly deteriorate. For both ensemble combination methods, ensembles of size two achieved significantly better results than ensemble sizes of five and ten.

Table 6.16: Results obtained by the cooperative coevolution approach with a larger number of iterations

	Sum			Vote		
Ensemble Size	Min	Med	Max	Min	Med	Max
2	**15.11**	**15.92**	**16.43**	15.45	**16.33**	**16.95**
3	15.35	16.00	16.95	15.44	16.25	18.37
4	15.64	16.52	18.16	15.59	16.84	20.57
5	15.61	16.61	18.53	15.47	16.70	18.48
6	15.72	17.04	18.31	**15.39**	17.12	25.49
7	15.52	16.66	17.86	15.52	17.49	20.01
8	16.38	17.47	20.92	16.00	16.87	20.62
9	15.77	18.26	23.26	15.50	17.01	18.77
10	16.51	17.58	21.27	15.56	17.22	19.49
GP	15.23	15.96	17.59	15.23	15.96	17.59

Table 6.17 represents the results achieved by ESS when using ensembles generated by the cooperative coevolution approach with the larger number of iterations. However, neither for this case did ESS achieve significantly better results than GP, not even for one of the experiments. The obtained results were quite dispersed and the achieved median values were much worse than those obtained by GP.

Figure 6.17 shows the box plot representation of the results for ESS when using the ensembles evolved by the second configuration of the cooperative coevolution approach. ESS achieved better results when it was used to create ensembles of sizes five and nine than when creating ensembles of size two. Unfortunately, for all ensemble sizes ESS achieved significantly worse results than GP.

By comparing the two configurations with each other, it is possible to conclude that by increasing the number of iterations of the cooperative coevolution approach does not lead to improvements in the results. Therefore, there is no benefit in using the second configuration with the larger number of iterations, since it substantially increases the execution time, but does not have any major effect on the performance of the approach.

6.4.5 Performance comparison of ensemble learning approaches

This section will compare the performances between the different ensemble learning approaches on four selected scheduling criteria. For the *Twt* criterion the results which achieved the best

Table 6.17: Results obtained by ESS when using ensembles of size ten generated by the cooperative coevolution approach with a larger number of iterations

	Sum			Vote		
Ensemble subset size	Min	Med	Max	Min	Med	Max
2	16.52	20.31	33.34	16.62	18.79	24.15
3	15.77	18.11	22.66	16.24	17.79	23.39
4	15.75	17.57	19.91	**15.23**	17.66	22.18
5	15.78	17.52	21.04	16.14	17.23	21.20
6	15.74	17.58	**19.90**	15.51	17.14	22.46
7	15.75	**16.79**	20.69	15.80	**16.94**	**19.23**
8	15.75	17.49	20.68	15.57	17.02	19.60
9	**14.98**	17.32	21.74	15.58	17.13	21.49
GP	15.23	15.96	17.59	15.23	15.96	17.59

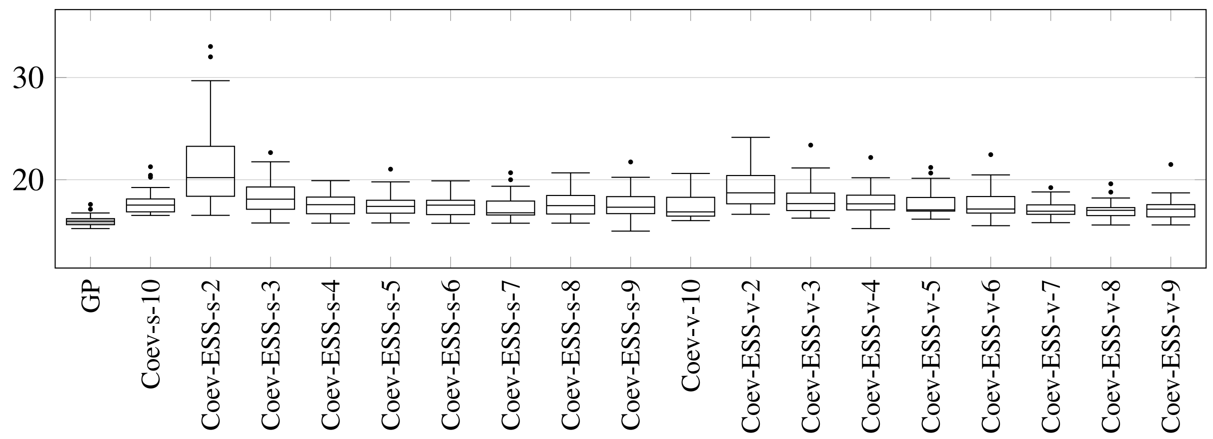


Figure 6.17: Box plot representation of the results obtained by ESS when using ensembles of size ten generated by the cooperative coevolution approach with a larger number of iterations

median values in the last four subsections will be aggregated. On the other hand, the parameters for the other three criteria were not as fine tuned as for the *Twt* criterion, but were rather chosen as a rule of thumb. Therefore better results could very likely be achieved if the parameters were further optimised for each given criterion individually.

Before analysing the results, the nomenclature of the approaches must first be described. Alongside the name of each approach, the *s* flag will denote that the sum combination method is used, whereas the *v* flag denotes that the vote combination method is used. The number alongside each approach will denote the size of the ensemble which was used. The *ESS* flag denotes that ESS was used to find a subset of ensembles and the size of the subset is denoted alongside the flag. The *B* flag in the BagGP approach denotes the bag size of problem instances which was used for evolving the ensembles. The *C* flag denotes that the confidences are used as weights in the BoostGP approach. Finally, the *con1* and *con2* flags denote that the first or the second configuration is used with the cooperative coevolution approach. In addition, all experiments which achieve significantly better results than DRs generated by GP will be denoted in grey. Each table will also include three standard DRs which achieved the very best performance for the given criterion.

Table 6.18 represents the results achieved by the different ensemble learning approaches for the *Twt* criterion. The first thing which can be noticed from the results is that all ensemble learning approaches achieved much better results than any of the standard DRs. For example, the best median value achieved by the SEC approach is by 9.1% better than the value achieved by the ATC rule. The overall best ensemble, which achieved the value of 14.41, outperforms the value of the ATC rule by 13.3%. Therefore the ensemble learning methods clearly outperform any of the standard DRs for this criterion.

By comparing the results of ensemble learning methods with GP, several interesting things can be observed. In most cases the ensemble learning approaches evolved ensembles with better performance than that of DRs evolved by GP. The BagGP, weighted BoostGP, and unweighted BoostGP approaches did not achieve achieve significantly better results than GP when they used the sum combination method, but were able to outperform GP if they used the vote combination method. By additionally using ESS it is possible to achieve significantly better results for all three aforementioned approaches. The reason for this is because the sum method is more sensitive to the composition of the ensemble, and since the DRs that form the ensembles are evolved independently in those approaches, it can easily happen that a DRs which has a negative influence on the entire ensemble is evolved and included in the ensemble. However, with ESS it is possible to filter out such DRs and thus improve the performance of the ensemble. The vote combination method is implicitly more robust since if the majority of rules in the ensemble perform good decisions, the addition of an extremely bad DR will not have an influence on the performance of the ensemble. Figure 6.18 shows the box plot representation of the results. The

Table 6.18: Performance comparison of the ensemble learning approaches when optimising the *Twt* criterion

Approach	Min	Med	Max
ATC	16.63	-	-
COVERT	16.86	-	-
EDD	17.31	-	-
GP	15.23	15.94	17.59
Sum ensemble combination			
SEC-5	14.84	**15.12**	**15.76**
SEC-5 ESS-3	14.88	15.21	15.99
BagGP-9 B80	14.91	15.77	17.26
BagGP-10 ESS-4 B80	**14.81**	15.59	17.07
BoostGP-3	15.28	15.76	17.45
BoostGP-10 ESS-5	14.85	15.61	16.47
BoostGP-7 C	14.95	15.78	16.42
BoostGP-10 C ESS-4	14.92	15.64	16.43
Coevolution-2 con2	15.11	15.92	16.43
Vote ensemble combination			
SEC-9	15.20	15.54	**16.06**
SEC-10 ESS-4	15.17	15.65	16.25
BagGP-9 B80	15.11	**15.39**	16.32
BagGP-10 ESS-5 B80	15.05	15.41	16.36
BoostGP-5	15.08	15.50	16.11
BoostGP-10 ESS-9	**14.99**	15.54	16.11
BoostGP-9 C	15.02	15.52	16.50
BoostGP-10 C ESS-8	15.02	15.42	16.29
Coevolution-3 con1	15.14	16.01	16.96

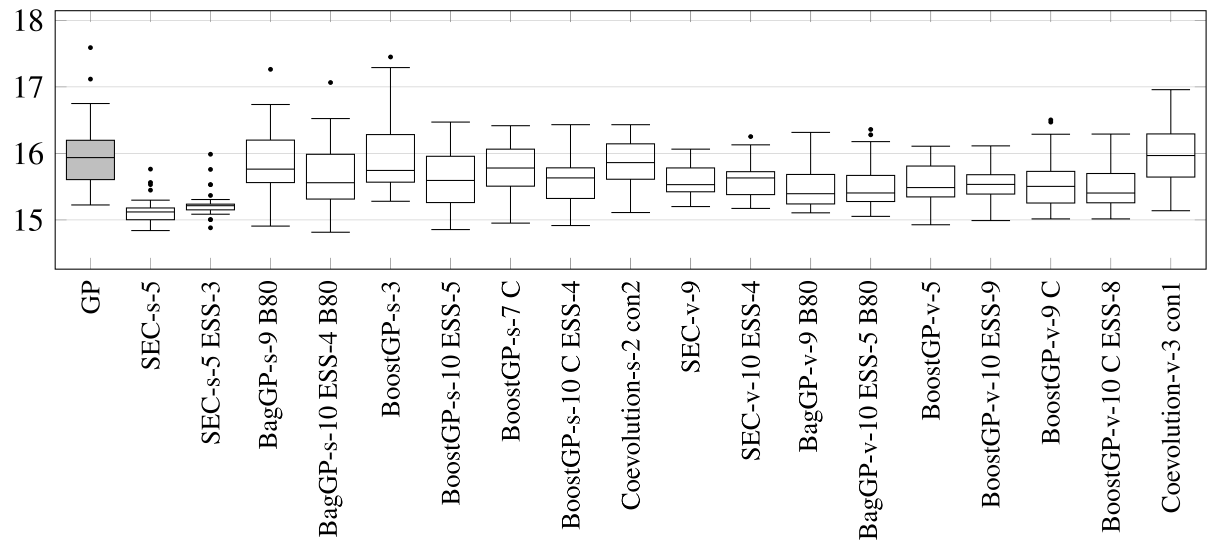


Figure 6.18: Box plot representation of the results obtained by the ensemble learning approaches when optimising the *Twt* criterion

figure shows that most ensemble learning methods achieve a much better solution distribution than DRs evolved by GP. This is especially true for the ensembles achieved by SEC with the sum combination method, since more than 75% of the obtained ensembles achieved a better performance than the best DR evolved by GP. By comparing the two ensemble combination methods, it can be concluded that the sum combination method achieved better results with the SEC and cooperative coevolution approaches, while the other approaches achieved better median values when the vote combination method was used.

Out of the tested ensemble learning approaches the worst results for both ensemble combination methods were achieved by the cooperative coevolution approach. When the sum combination method is used, the best results are achieved by the SEC approach, both with and without ESS. This method clearly outperforms any of the other tested ensemble learning approaches. The results achieved by BoostGP and BagGP are mostly the same. If ESS is used, their performances increase, but are still more or less equal. On the other hand, for the vote combination method, SEC, BoostGP and BagGP approaches achieve similar results, even with the application of ESS. The overall best median value was achieved by the BagGP approach. It is interesting to note that for the vote combination method ESS was in most cases unable to find an ensemble subset which achieves better performance than the original ensemble.

Table 6.19 represents the results achieved by the ensemble learning approaches for the *Nwt* criterion. The ensemble learning approaches once again perform much better than any of the three tested standard DRs. The best median value, which is achieved by the SEC approach with ESS, outperforms the Sufferage rule by 3.9%. On the other hand the best DR achieved by any of the ensemble learning approaches outperforms the Sufferage rule by 8.2%. Therefore, even for this criterion the ensemble learning methods prove to be better than any of the tested standard

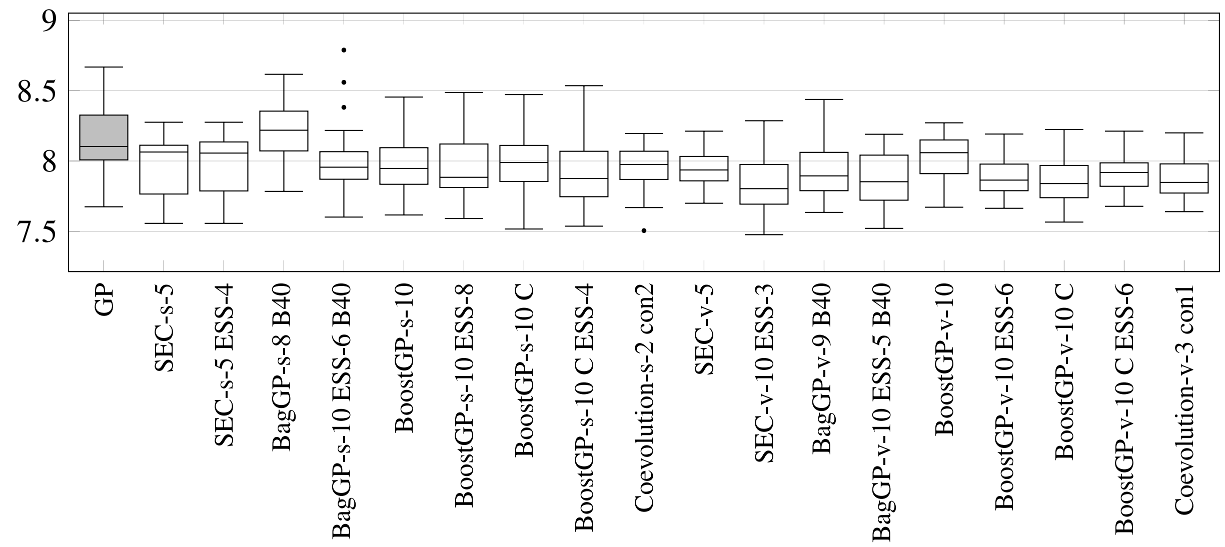


Figure 6.19: Box plot representation of the results obtained by the ensemble learning approaches when optimising the *Nwt* criterion

DRs.

For this criterion almost all ensemble learning approaches achieved significantly better results than DRs generated by GP. Only the BagGP method was unable to achieve significantly better results when using the sum combination method. Even the cooperative coevolution approach, which achieved the worst results when the *Twt* criterion was optimised, now easily outperforms results achieved by GP. The largest improvement that an ensemble learning method can achieve for the median value when compared to GP, amounts to around 3.4%. Figure 6.19 additionally shows the box plot representation of the results. The figure illustrates that the evolved ensembles achieve much better distributions of solutions, especially when the vote combination method is applied. Therefore, based on all previous observations, it can be concluded that the generated ensembles easily outperform the results obtained by the individual DRs.

For the sum combination method the best results for the *Nwt* criterion are achieved when using ESS on the ensembles evolved by the weighted BoostGP approach. However, by applying ESS on the unweighted BoostGP approach similar results can also be achieved. Therefore the BoostGP approach seems to be best suited out of all the tested ensemble learning approaches. The cooperative coevolution approach also achieved good results, obtaining the best minimum and maximum values, although for the median value it was unable to achieve the best performance. Nevertheless, the cooperative coevolution approach achieved similar median values as BoostGP without ESS. The SEC method achieved worse results than any of the aforementioned approaches, which is surprising since for the *Twt* criterion it was unrivalled in its performance. BagGP achieved the worst performance, but the results can significantly be improved by using ESS. On the other hand, for the vote combination method the worst results are achieved

Table 6.19: Performance comparison of the ensemble learning approaches when optimising the *Nwt* criterion

Approach	Min	Med	Max
Sufferage	8.148	-	-
ATC	8.190	-	-
COVERT	8.190	-	-
GP	7.674	8.107	8.669
Sum ensemble combination			
SEC-5	7.556	8.064	8.276
SEC-5 ESS-4	7.556	8.064	8.276
BagGP-8 B40	7.784	8.224	8.617
BagGP-10 ESS-6 B40	7.601	7.975	8.789
BoostGP-10	7.616	7.949	8.487
BoostGP-10 ESS-8	7.591	7.888	8.487
BoostGP-10 C	7.516	7.995	8.473
BoostGP-10 C ESS-4	7.536	**7.886**	8.536
Coevolution-2 con2	**7.505**	7.980	**8.196**
Vote ensemble combination			
SEC-5	7.699	7.946	8.212
SEC-10 ESS-3	**7.476**	**7.834**	8.287
BagGP-9 B40	7.634	7.901	8.438
BagGP-10 ESS-5 B40	7.520	7.865	**8.190**
BoostGP-10	7.663	7.873	8.192
BoostGP-10 ESS-6	7.565	7.848	8.224
BoostGP-10 C	7.677	7.920	8.212
BoostGP-10 C ESS-6	7.640	7.868	8.200
Coevolution-3 con1	7.671	8.062	8.272

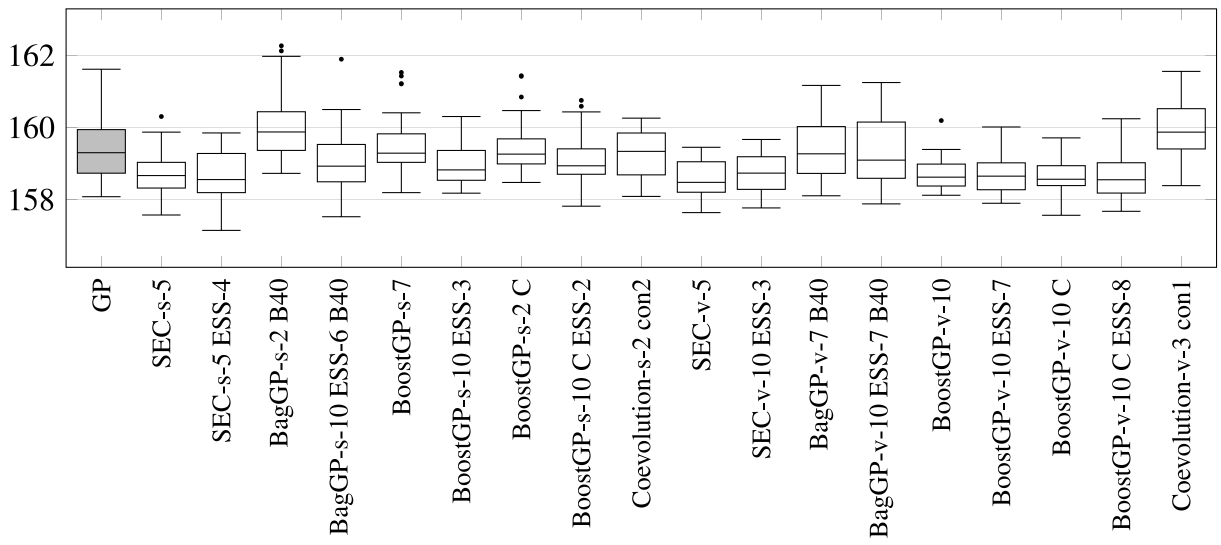


Figure 6.20: Box plot representation of the results obtained by the ensemble learning methods when optimising the Ft criterion

by the cooperative coevolution approach. The other three approaches all achieve very similar results, which are usually improved to a certain extent by additionally applying ESS. The best minimum and median values were achieved by ESS when using ensembles evolved by SEC. All approaches, except for the cooperative coevolution approach, achieved better performance if they are used with the vote combination method. In addition, the vote combination method has achieved solutions which are less dispersed.

Table 6.20 represents the results achieved by the ensemble learning methods for the Ft criterion. Even for this criterion the ensembles generated by the ensemble learning approaches can in most cases achieve better performance than the standard DRs. However, the improvements which can be achieved are this time not as extensive. The best achieved median value outperforms the KPB rule by 0.7%, while the best minimum value outperforms the KPB rule by 1.6%.

Unfortunately, for this criterion the ensemble learning approaches have shown to struggle to achieve significantly better results than DRs evolved by GP. When the sum combination method is used, only the SEC approach and ESS with SEC were able to achieve significantly better results than the DRs generated by GP. On the other hand, the vote combination method can achieve significantly better results than GP for the SEC and BoostGP approaches. Although the ensemble learning approaches achieve significantly better results than GP in several occasion, unfortunately they outperform the GP method by only 0.4% for the median value. Figure 6.20 shows the box plot representation of the results when the Ft criterion is optimised. The figure shows that most of the evolved ensembles achieve similar solution distributions as GP, however, those few ensemble learning methods which manage to achieve significantly better results achieve good solution distributions, and less dispersed results than GP.

Table 6.20: Performance comparison of the ensemble learning approaches when optimising the Ft criterion

Approach	Min	Med	Max
KPB	159.6	-	-
RC	159.8	-	-
Min-min	159.9	-	-
GP	158.1	159.3	161.6
Sum ensemble combination			
SEC-5	157.6	158.7	160.3
SEC-5 ESS-4	**157.1**	**158.6**	**159.8**
BagGP-2 B40	158.7	159.9	162.3
BagGP-10 ESS-6 B40	157.5	158.9	161.9
BoostGP-7	158.2	159.3	161.5
BoostGP-10 ESS-3	158.2	158.9	160.3
BoostGP-2 C	158.5	159.3	161.4
BoostGP-10 C ESS-2	157.8	158.9	160.8
Coevolution-2 con2	158.1	159.4	160.3
Vote ensemble combination			
SEC-5	**157.6**	**158.5**	**159.4**
SEC-10 ESS-3	157.8	158.8	159.7
BagGP-7 B40	158.1	159.3	161.2
BagGP-10 ESS-7 B40	157.9	159.1	161.2
BoostGP-10	158.1	158.7	160.2
BoostGP-10 ESS-7	157.9	158.7	160.0
BoostGP-10 C	**157.6**	158.6	159.7
BoostGP-10 C ESS-8	157.7	158.6	160.2
Coevolution-3 con1	158.4	160.0	161.6

For the sum combination method the best results were achieved by the SEC approach. BagGP, BoostGP, and cooperative coevolution all achieved similar results. The results of BagGP and BoostGP can be further increased by applying ESS. On the other hand, for the vote combination method the best results are again achieved by the SEC approach, followed closely by BoostGP. For this combination method the worst results were achieved by cooperative coevolution. The SEC approach achieved similar results for both combination methods. BoostGP and BagGP performed better when the vote combination method is used, while cooperative coevolution performs better when the sum combination method is used. The vote combination method again seems to achieve somewhat better solution distributions for most experiments.

Table 6.21 represents the results achieved by the ensemble learning approaches for the C_{max} criterion. Although all ensemble learning approaches are able to evolve an ensemble which outperforms the Sufferage2 rule, in most cases the approaches can not achieve a better median value. The best median value outperforms the Sufferage2 rule by only 0.2%, while the best overall ensemble outperforms the Sufferage2 rule by 0.6%. Although for this criterion the ensemble learning approaches achieved the smallest improvements, they nevertheless present a viable alternative to the standard DRs.

The ensemble learning approaches have in most cases outperformed the results achieved by GP. Only the cooperative coevolution approach, for both combination methods, and BagGP, for the sum combination method, were unable to achieve significantly better results than GP. For both combination methods the best median values were achieved by the SEC approach. The BagGP and BoostGP approaches obtained similar results when the vote combination method was used, while for the sum combination method the BoostGP approach achieved better results. Although most of the ensembles achieve significantly better results than GP, unfortunately they can outperform the results achieved by GP by at most 0.9% for the median value. Figure 6.21 shows the box plot representation of the results. The figure shows that in most cases the ensemble learning approaches achieve less dispersed results than GP. This is especially true for SEC and ESS with the vote combination method, where most of the obtained solutions outperform the solutions obtained by GP. For the sum combination method, ESS with SEC achieved the best solution distribution.

For this criterion, all ensemble learning approaches achieve better median values when used with the vote combination method. The SEC approach performs well with ensemble combinations methods, but for the vote combination method it achieves a better median value. On the other hand, the cooperative coevolution approach achieves the worst performance by using the sum combination method, but the results for the approach improve only slightly if the vote combination method is used. BoostGP and BagGP achieve good results for the vote combination method, while for the sum combination method they achieve slightly worse results.

Table 6.21: Performance comparison of the ensemble learning methods when optimising the C_{max} criterion

Approach	Min	Med	Max
Sufferage2	38.44	-	-
Sufferage	38.48	-	-
RC	38.56	-	-
GP	38.29	38.70	39.45
Sum ensemble combination			
SEC-5	38.34	38.51	**38.73**
SEC-5 ESS-2	**38.31**	**38.45**	38.77
BagGP-10 B40	38.42	38.77	39.44
BagGP-10 ESS-2 B40	38.35	38.61	39.08
BoostGP-9	38.38	38.62	39.11
BoostGP-10 ESS-2	38.38	38.58	38.95
BoostGP-5 C	38.33	38.62	38.99
BoostGP-10 C ESS-5	38.34	38.57	39.01
Coevolution-2 con2	38.34	38.76	38.99
Vote ensemble combination			
SEC-5	38.28	**38.37**	**38.71**
SEC-10 ESS-3	38.32	38.40	38.83
BagGP-5 B40	38.37	38.56	38.98
BagGP-10 ESS-9 B40	38.36	38.52	38.90
BoostGP-7	38.23	38.59	39.04
BoostGP-10 ESS-5	38.31	38.55	38.83
BoostGP-7 C	**38.20**	38.61	39.02
BoostGP-10 C ESS-5	38.36	38.52	38.89
Coevolution-3 con1	38.33	38.71	39.10

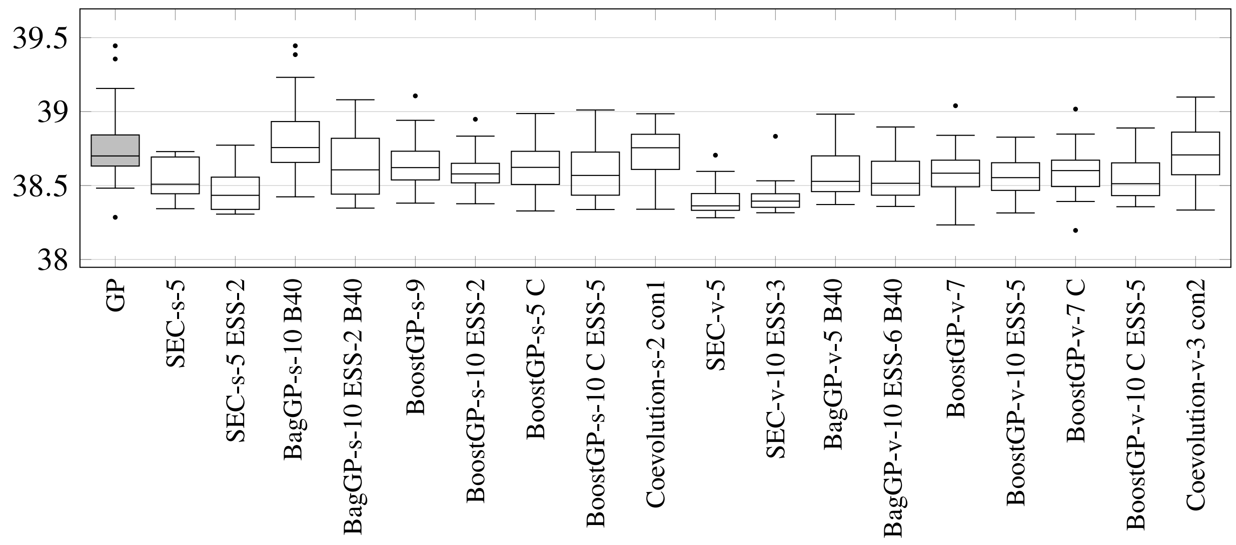


Figure 6.21: Box plot representation of the results obtained by the ensemble learning approaches when optimising the C_{max} criterion

The ESS method improves the performance of most ensemble learning methods. Based on the previous results it can clearly be concluded that the vote combination method is much more suitable when evolving ensembles for optimising the C_{max} criterion, since all ensemble learning methods achieve a better performance for the vote combination method.

6.5 Discussion

In this section a short discussion about the ensemble learning methods, which is based on the results and observations obtained in the previous section, will be provided.

6.5.1 SEC

Although being a quite simple approach, SEC achieved good results on all four tested scheduling criteria. While for the *Twt* criterion the approach achieved the best results by using the sum combination method, for the other three criteria better results were achieved by the vote combination method. For the sum combination method better results were achieved when medium sized ensembles were used. On the other hand, when the vote combination method was used, better results were achieved by using medium sized and larger ensembles. In certain occasions it was possible to further increase the performance by applying ESS on the generated ensembles.

The most obvious benefit of this approach is that it can be used with already existing DRs, thus eliminating the need of evolving new rules. But even if new rules need to be evolved, the time needed for their generation is significantly smaller than for the other methods, since they can be evolved completely independently from reach other. In addition, this approach offers much freedom in the choice of how the ensembles should be constructed. Thus, the time needed to obtain the ensembles can be adjusted by using methods of different complexities. An additional benefit of this approach, which became evident after analysing the results, is that it is usually more stable and achieves less dispersed results than the other tested ensemble learning approaches. The main drawback of this approach is that it needs an additional problem instance set on which the ensemble will be constructed. Nevertheless, based on all the aforementioned points, it is evident that this approach is not only competitive with other ensemble learning approaches from the literature, but rather that it is even superior n certain cases.

6.5.2 BagGP

The performance of the BagGP approach largely depends on the combination method which is used. The approach usually achieved better results when the vote combination method was used. The reason for this is the fact that the ensembles are evolved independently from each other, and therefore there can be a lot variability between the evolved DRs, which can have a negative effect on the sum combination method. However, with the use of ESS these results can generally be improved, but the vote combination method still achieved results with better median values for most cases. For the *Nwt* criterion the BagGP approach achieved the overall best median values, when using the vote combination method.

In addition to the ensemble size and the ensemble combination method, BagGP introduces an additional parameter, which represents the bag size. The experiments have shown that usually better results are achieved for larger bag sizes. However, with the bag size the computation cost of the approach also increases. Therefore, the value for this parameter needs to be carefully chosen so that good solutions can still be achieved, but that also the execution time of the algorithm is not too extensive.

The advantage of this approach is that DRs which form the ensemble are evolved independently and can therefore be evolved in parallel, which can improve its execution time. However, since the DRs that form the ensemble are evolved completely independently from each other, this causes approach to achieve results which can be quite dispersed. In addition to that, the bag size also needs to be optimised to select the one which leads to the best results. If the sum combination method is used, then ESS should also be applied on the final ensemble to achieve improved results. Regardless of all problems BagGP achieved a good performance, especially when used with ESS. However, in most cases it did not outperform the results from the SEC approach.

6.5.3 BoostGP

For each evolved DR in the ensemble, BoostGP additionally provides the confidence value for it. Therefore, one of the main objectives in the tests was also to determine whether using the confidence values as weights for individual DRs can lead to improved performance. The experiments have shown that additionally using the confidences as weights does not significantly increase the performance of the method. In certain occasions better median values can be achieved by using confidences as weights, however, there is no significant difference between the results obtain with and without using confidences as weights. Since the DRs that form the ensemble are mostly evolved independently, the vote combination method performs better when used with this approach. With the use of ESS it is possible to again improve the results achieved by BoostGP, especially when the sum combination method is used.

This approach achieves mostly similar results as the BagGP approach, but unlike the BagGP approach it does not introduce any extra parameters. The only situation where BoostGP is more preferable than BagGP is when optimising the Ft criterion, where BoostGP managed to outperform GP, while BagGP was unable to do so. On the other hand, the DRs in BoostGP can not be evolved independently as in BagGP, since for each DR in the ensemble the weights of the training samples need to be calculated based on the previous DRs. Therefore BoostGP represents an alternative to BagGP, and depending on the requirements either one of those two can be chosen and will perform similarly.

6.5.4 Cooperative coevolution

Since the cooperative coevolution approach simultaneously evolves DRs that form the ensemble, it was expected that this approach would achieve the most competitive results out of all the approaches. Unfortunately this approach has in most cases achieved results which were the worst out of all the tested approaches. The most evident reason why this happens is that the method overfits on the training set. This assumption is backed up by the fact that the cooperative coevolution approach achieves better results on the training set than GP. Even trying out different termination criteria did lead to significant improvements of the results. Thus in future work some other methods of preventing overfitting should be tried out to determine if this could improve the performance of the approach. Since the few good results obtained by this approach were achieved mostly when using smaller ensemble sizes, it is likely that the procedure struggles in evolving good DRs which complement the other DRs in the ensemble. This was especially evident for the vote combination method, where in certain situations the ensemble consisted of several rules which together made suboptimal choices. However, when one of those rules would be replaced, the effectiveness of the ensemble would deteriorate even further, therefore the algorithm would be stuck in a local optimum. Because of that reason, the procedure should be extended with mechanisms that could prevent such occurrences or correct them (for example by reinitialising the ensemble with random DRs).

The cooperative coevolution approach copes with another important problem, and that is its execution time. Namely, the execution time of this procedure heavily depends on the number of ensembles it evolves, but to a much greater extent than any of the aforementioned procedures. This is a consequence of the fact that in each iteration the cooperative coevolution approach has to evaluate an ensemble of DRs, thus prolonging the evaluation process, whereas the other procedures only evaluate individuals by themselves. This results in slower execution times, especially for larger ensemble sizes.

Since the cooperative coevolution approach was not able to evolve bigger ensembles of good quality, the best subsets found by ESS were usually not better than the best solution found by the cooperative coevolution approach. However, applying ESS on ensembles generated by cooperative coevolution might not even work well since the DRs which form the ensemble are evolved simultaneously and the ensemble is therefore much more sensitive to the changes of the DRs which constitute it.

Although the cooperative coevolution approach did achieve good results for the *Nwt* criterion, on all other criteria it did not perform well. In many cases this approach even achieved results which were significantly worse than those of GP. Therefore this approach has shown to be the least useful out of the tested ensemble learning approaches.

6.5.5 ESS

ESS has demonstrated to be very promising in improving the results of the ensemble learning approaches. Naturally, ESS did not obtain subsets of a better quality than the original ensemble in every single occasion, but in many cases it obtained ensemble subsets which significantly improved the results when compared to the original ensemble. ESS has especially proven useful when being used with ensemble learning approaches which independently evolve the DRs which form the ensemble, like BagGP and BoostGP. In most occasions ESS did not significantly improve the results for SEC, since that approach already functions similarly as ESS. For cooperative coevolution ESS also did not achieve any improvements. The reason for this could be due to the fact that in this approach DRs are much more interdependent than in other approaches, since the DRs are all evolved simultaneously. The best minimum values for all criteria, except for the C_{max} criterion, have been achieved by using ESS.

There are several benefits of using ESS. First of all it tries not only to find a better ensemble, but also to find an ensemble of a smaller size. As the experiments have shown, ESS was in many occasions able to find a better subset which significantly reduced the size of the original ensemble. Secondly, this approach is applicable to any of the tested ensemble learning approaches, and for some approaches (BagGP and BoostGP) it will additionally improve their performances. Lastly, the execution time of this approach is fast even when performing an exhaustive search for ensembles of size ten. Naturally, with bigger ensembles the execution time of ESS would grow drastically. However, the execution time of ESS can be improved by not using an extensive search for the subsets of ensembles, but rather a random search or a search guided by some heuristic method.

On the other hand, ESS also has certain disadvantages. In order to perform ESS, another problem instance set is required. In addition, ESS provides no guarantee that it will obtain ensembles which will perform better than the original ensemble on which ESS was applied. Therefore, in rare occasions it is possible that for certain ensembles all ensemble subsets constructed by ESS do not perform at least equally well as the original ensemble. Finally, using ESS also increases the time needed to obtain an ensemble. Although for smaller ensembles this time is almost negligible, it increases with the number of DRs in the ensemble.

Based on all the previous outlined characteristics, it is safe to conclude that ESS represents a good addition to ensemble learning approaches, to improve their performance and decrease the ensemble size.

6.5.6 Influence of the ensemble combination methods

From the previously obtained results, many conclusions can also be drawn about the two ensemble combination methods which were used. The first thing which was observed is that the vote combination method was more suitable for ensemble learning approaches which independently evolved the DRs that form the ensemble, like BoostGP and BagGP. In these situations the vote method is more robust, since even if one DR performs poorly, it will not have a large influence on the decision of the ensemble if all other rules perform well. On the other hand, the sum combination method leads to better results with the cooperative coevolution approach, in which the DRs in the ensemble are evolved simultaneously. The situation for the SEC approach is much more interesting, since that approach achieved better results for the Twt criterion when using the sum combination method, while on the other hand for the Nwt and C_{max} criterion it achieved better results when using the vote combination method. For the Ft criterion both combination methods achieved similar results.

ESS has proven to be more effective on ensembles which use the sum combination method, since that method is more sensitive to the composition of the ensemble. Nevertheless, ESS also achieved improvements for ensembles which use the vote combination method as well. Therefore, ESS achieved improvements regardless of the ensemble combination method, which is used by the ensemble to perform the decision.

6.5.7 Influence of the ensemble size

The size of the ensembles has also a significant influence on the results achieved by the ensemble learning approaches. For example, the sum combination method usually achieved better results when ensembles of smaller or medium sizes were used. This is especially true when ESS is applied on ensembles which use the sum combination method, since ESS quite often generated ensembles consisting only out of two or three DRs. This is not surprising since the sum combination method is more sensitive to the composition of the ensemble, since it is easier to find a smaller number of DRs which will work well together if their priority values are summed up.

On the other hand, the vote combination method preferred ensembles of medium and larger sizes. The reason for this is that in smaller ensembles ties in the decisions will usually occur more often. In addition, the vote combination method is more resilient if bad DRs appear in the ensemble, since if the other DRs perform good decisions the ensemble will also perform good decisions based on the majority of DRs. With ESS the ensemble sizes are usually reduced by a small number of ensembles, but they still consist of at least four DRs in most of the cases.

6.6 Analysis of the generated ensembles

In this section the best ensembles which were constructed by the different approaches will be analysed, to gain further insights into the structure of the generated ensembles. For each approach, the individual which achieved the best results on the test set will be selected and analysed. In order to make the analysis more concise, it will be performed only on a selected number of problem instances, which illustrate some interesting behaviours of the ensemble. For the SEC approach, the analysis will be performed on ensembles constructed by the random selection method with 20000 generated ensembles, since this construction method achieved the best median value out of all the tested construction methods, and is thus more likely to obtain good ensembles. It must be stressed out that the problem instances on which the analysis will be performed are selected from the problem set which was used to create the ensemble. This means that for SEC and ESS the analysis will be performed on problem instances from the validation set, while for BagGP, BoostGP, and cooperative coevolution the analysis will be performed on the training set. In addition, the analysis will be performed only on the rules which were evolved for optimising the *Twt* criterion. An additional analysis will also be performed for the SEC approach, to examine which DRs most often constitute the ensembles with the best performance. This analysis should provide a deeper insight if there is any regularity in the DRs which are chosen to form the ensembles, and if this information could somehow be used in the ensemble construction process.

6.6.1 Analysis of the frequency of DRs in the ensembles generated by SEC

Table 6.22 gives an overview of thirteen DRs which most commonly appeared in the best ensemble constructed in each algorithm run. For the analysis, only ensembles consisting out of five DRS combined with the sum method were used. For the random and probabilistic selec-

Table 6.22: Most commonly contained DRs in the best ensembles constructed by SEC and combined by the sum combination method

Method	DR indices												
	0	1	4	6	9	11	13	15	19	30	39	41	46
Random	5	1	10	7	0	17	3	4	15	19	0	9	14
Probabilistic	6	0	9	11	2	19	1	4	13	25	0	4	9
Grow	27	7	12	5	4	17	18	19	6	11	14	13	10
Grow-destroy	13	7	4	6	3	14	18	14	8	10	14	15	13
Instance based	1	20	1	23	19	25	1	43	8	22	1	27	1
Fitness	14.28	13.31	13.93	13.17	13.15	13.11	13.49	12.96	13.17	13.12	14.36	13.07	14.31

tion methods 30 runs were performed, while 50 runs were performed for the remaining three methods. Each cell in the table represents the number of times that, for a concrete construction method, the DR with the given index appeared in the best constructed ensemble. For each ensemble construction method, the values of five DRs which most often appeared in the best ensembles are denoted with a grey background. The table shows that the random and probabilistic selection methods mostly use the same DRs to create ensembles. These two methods construct ensembles which in around 60% of cases contain DRs 11 and 30, while approximately 50% of the ensembles contain DR 19. From the fitness values of the three aforementioned DRs it is evident that all of them achieve good individual performance. The other DRs which are most often used by these ensemble construction methods (rules 4 and 46), do not achieve an equally good performance. The results show that the grow and grow-destroy methods mostly use the same DRs to construct the ensembles. However, these DRs are in most cases different than those used by the random and probabilistic selection methods. The grow and grow-destroy methods also have a preference towards the inclusion of certain DRs in the ensemble, like DRs 11, 13, 15, 39 and 41, which were usually contained in around 25% to 40% of ensembles with the best performance. Most of these DRs also achieve a good individual performance. The instance based method uses DRs which are partially the same as those used by the random and probabilistic selection methods, and partially the same as those used by the grow and grow-destroy methods. In addition to those DRs, the instance based method also uses some DRs which are rarely used by any of the other ensemble construction methods, like DRs 1 and 9. This method is very biased towards using DR 15, since it is used in over 80% of the constructed ensembles. Furthermore, it is also biased towards DRs 6, 11, and 41, which also achieve good individual performance, and are used in around 50% of the best constructed ensembles.

By analysing the use of DRs over several construction methods, only DR 11 has constantly been among the five most commonly used DRs by all of the tested ensemble construction methods. The instance based method has created ensembles which almost always consisted out of

DRs which individually achieve a good performance. The other construction methods also create ensembles which consist of DRs which individually achieve good performance, however, in many occasions the ensembles also contain DRs which do not perform well on their own. Another interesting fact is that DR 15, which achieves the best individual performance, is most commonly used by the instance based method. However, it is less commonly used by the grow and grow-destroy methods, and rarely by the random and probabilistic selection methods. This demonstrates that the best individual DR will not necessarily be contained in the best ensembles. Based on the results, it is evident that the instance based method is mostly biased towards using DRs which individually perform well, whereas the other four methods usually use a mixture of DRs, where most achieve good individual performance, but still several DRs, which individually do not perform well, are also included in the ensembles. All the previous observations demonstrate that the applied construction mechanism has a large influence on the choice of DRs which form the ensemble.

Table 6.23 represents the prevalence of DRs in the best ensembles constructed with all the construction methods, and by using the vote combination method. The random and probabilistic selection methods once again construct ensembles which consist of similar DRs. These methods most commonly create ensembles which consist of DRs 32 (used in all ensembles constructed by the random selection method), 39 (used in more than 80% of the ensembles), and 19 (used in around 50% of the ensembles). The grow and grow-destroy methods construct ensembles which most often consist of DRs 28, 32, 33, and 39. Unlike for the sum combination method, several DRs are quite often contained in the best DRs constructed by the previous four construction methods, such as DRs 32 and 39. The instance based method constructs the ensembles by using DRs which most often differ from those used by the other construction methods. This construction method uses DR 15 in almost all ensembles, and rule 30 in more than 80% of the ensembles. The instance based method is again biased towards using the best individual DR, however the grow and grow-destroy methods are now less biased towards using this DR than they were when the sum combination method was used.

By analysing the occurrence of DRs over several construction methods, it is evident that no single DR is used predominantly by all ensemble construction methods. Rules 32 and 39 were used in many occasions by all construction methods, except for the instance based method. The random and probabilistic selection methods mostly create ensembles which consist out of DRs that individually achieve a good performance. However, these methods also often include rule 39 in the ensembles, which does not achieve a good performance on its own. The grow and grow-destroy methods use a wide range of DRs to construct the ensembles, however, they mostly use DRs which do not achieve the overall best performance, but rather rules whose performance is closer to the median performance of all DRs. Finally, the instance based method uses almost exclusively DRs which achieve a good individual performance.

Table 6.23: Most commonly contained DRs in the best ensembles constructed by SEC and combined by the vote combination method

Method	DR indices												
	6	11	15	19	27	28	30	32	33	34	39	41	45
Random	3	2	2	15	4	6	2	30	4	10	26	0	1
Probabilistic	0	6	5	17	6	4	1	27	3	5	25	1	3
Grow	10	11	6	8	5	16	7	27	11	7	13	4	9
Grow-destroy	10	8	9	11	5	13	10	29	18	11	12	3	9
Instance based	20	36	49	5	1	1	42	1	1	1	1	28	20
Fitness	13.17	13.11	12.96	13.17	13.61	13.69	13.12	13.20	13.72	13.49	14.36	13.07	13.14

The previous results show that, although there are several DRs, most notably rules 11, 19, and 39, which are commonly used by ensembles combined by both ensemble combination methods, most of the rules which are used by those two combination methods will be different. The only exception occurs for the instance based method, which mostly uses the same DRs to construct the ensembles, regardless of the applied combination method. It should also be noted that the vote combination method uses DRs with better individual performance in more occasions than the sum combination method.

It is also interesting to observe whether the information about the frequency of DR usage can be used to create good DRs. In order to try this out, the five DRs which appeared most often in the best ensembles were combined into an ensemble, and evaluated on the test set. This was done for all combination and creation methods. If there are more DRs with the same number of occurrences which can be selected as the final DR in the ensemble, the one for which the ensemble provides a better result will be selected. The results achieved when using the sum combination method were 14.94 for the random selection method, 15.18 for the probabilistic selection method, 15.21 for the grow method, 15.71 for the grow-destroy method, and 15.16 for the instance based method. All ensembles, except the one constructed for the grow-destroy method, obtain a better value than the best individual DR generated by GP. On the other hand, the vote method achieved a value of of 16.03 for the random selection method, 15.73 for the probabilistic selection method, 15.83 for the grow method, 15.76 for the grow-destroy method, and 15.18 for the instance based method. For the vote combination method, only the the ensemble constructed for the instance base method performed better than the best individual DR constructed by GP. All other ensembles, except for the one constructed by the random selection method, still achieved a better value than the median value of all the DRs generated by GP. The instance based construction method was the only one for which the ensemble constructed by using the most frequently occurring DRs performs well by using both construction methods. The obtained results prove that for the sum combination method the information about the frequency of DR occurrences in better ensembles could be used for constructing ensembles, whereas for the vote method this information has not proven to be too beneficial.

6.6.2 Analysis of ensembles generated by SEC

Table 6.24 represents the performance of the best constructed ensemble by SEC, with the sum combination method. The table also denotes the performance of individual DRs which form the ensemble. For the analysis, the best ensemble of size five, constructed by the random selection method, was used. The analysis will be performed on the validation set, since this set is used for constructing the ensembles, and therefore has the most influence on the choice of which DRs will form the ensemble. The best results of the individual DRs for each problem instance are denoted with a grey cell, while the results for which the ensemble performs at least as well as the best individual DR are denoted in bold. The results show that the ensemble is able to pick up the good behaviour of the individual DRs and to achieve the same performance as the best DR, like for problem instances 1, 22, and 28. If more DRs perform the same, it is more likely that the entire ensemble will also perform well. This behaviour can be observed on problem instances 22 and 28. In several occasions the ensemble also outperforms all of the DRs which it consists of, like for problem instances 17, 39, and 40. This is an important observation, which shows that the ensemble is not only able to perform equally well as the best DR it consists of, but rather with a good combination of DRs the ensemble can even outperform the best rule contained in it. However, it is also possible that the ensemble is unable to perform better or equal as the best DR it consists of. In most of these cases, the ensemble will achieve a fitness value which is between the best and the worst values obtained by the DRs contained in the ensemble. Such a behaviour can be observed for problem instances 13, 26, and 29. In rare occasions it can also happen that the ensemble performs worse than any of the DRs it contains, as it happens for problem instance 41. Figure 6.22 shows the performance of the ensemble on the entire validation set. The numbers in the figure denote how many instances in the problem set belong to each of the categories. From the figure it is evident that the ensemble performs well on most of the problem instances, and is thus able to achieve better performance than individual DRs on their own.

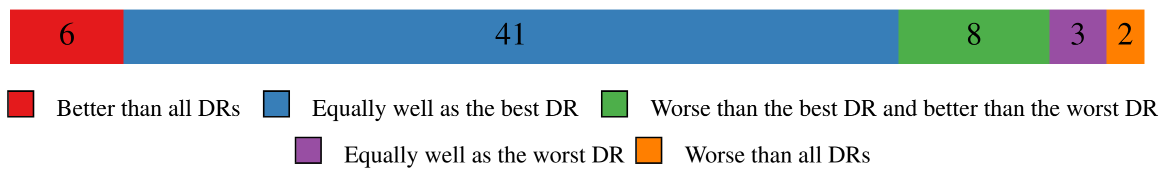


Figure 6.22: Performance of the best ensemble constructed by SEC and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

By comparing the performance of the ensemble with that of the individual DRs out of which it is constructed, it is evident that the ensemble achieves better performance than any of the DRs

Table 6.24: Performance analysis of the best ensemble generated by SEC and combined with the sum combination method

Problem instance index	Individual DR					Ensemble
	0	11	28	30	46	
1	0.620	0.494	0.295	0.264	0.301	**0.264**
13	1.131	0.737	0.767	0.870	1.126	0.836
17	0.116	0.489	0.170	0.204	0.259	**0.102**
22	1.894	1.860	2.137	1.889	1.860	**1.860**
26	0.936	1.010	1.123	0.978	1.142	0.978
28	0.032	0.091	0.091	0.032	0.032	**0.032**
29	0.506	0.524	0.506	0.506	0.430	0.506
39	0.996	0.960	1.034	0.955	0.999	**0.917**
40	1.399	1.233	1.399	1.233	1.386	**1.177**
41	0.053	0.091	0.091	0.116	0.058	0.163
Total fitness on all instances	14.28	13.11	13.69	13.12	14.31	12.45
Fitness on the test set	16.39	15.72	16.20	15.67	16.02	14.84

on their own. On the validation set, the ensemble can outperform the best DR in the ensemble by 5%, while on the test set the improvement over the best DR is approximately 5.3%. Therefore, on both problem sets the ensemble achieves a similar improvement over the individual DRs. Moreover, the individual DRs that form the ensemble do not all perform well on both problem instance sets. While two DRs achieve good results on both problem sets, the other three DRs perform worse to a certain extent. This proves that for the ensemble to perform well, it does not need to be constructed out of DRs which all individually perform well.

Table 6.25 represents the performance of a selected ensemble when the vote combination method is used. The performance of the selected ensemble is compared to the performance of individual DRs out of which it is constructed. The comparison will be performed on several selected problem instances that were also used for analysing the sum combination method. On seven problem instances denoted in the table, the ensemble achieved better or equal results as the best individual DR of the ensemble. Out of these seven instances, a better value by the ensemble was achieved in two occasions, namely for problem instances 13 and 39. It is interesting to note how even though on some problem instances only one DR achieved the best value, the entire ensemble nevertheless achieved the same performance as the best DR. On the other three instances from the table, the ensemble was unable to achieve equally good

Table 6.25: Performance analysis of the best ensemble generated by SEC and combined with the vote combination method

Problem instance index	Individual DR					Ensemble
	1	13	19	32	39	
1	0.295	0.264	0.356	0.264	0.351	**0.264**
13	0.869	1.086	0.812	0.859	0.982	**0.768**
17	0.119	0.172	0.194	0.264	0.129	0.154
22	1.685	1.685	1.644	1.855	1.965	**1.644**
26	0.978	0.978	0.858	1.128	0.842	**0.842**
28	0.032	0.032	0.032	0.032	0.032	**0.032**
29	0.506	0.506	0.506	0.602	0.506	**0.506**
39	1.081	0.986	1.080	1.007	1.058	**0.985**
40	1.399	1.386	1.386	1.233	1.532	1.386
41	0.212	0.180	0.163	0.075	0.053	0.137
Total fitness on all instances	13.31	13.49	13.17	13.20	14.36	12.41
Fitness on the test set	15.92	15.48	16.08	16.44	15.68	14.91

results as the best DR. However, on neither of the selected problem instances it did not achieve inferior results when compared to the worst DR in the ensemble. Figure 6.23 represents the performance of the ensemble on the entire validation set, with regards to the individual DRs which are contained in the ensemble.

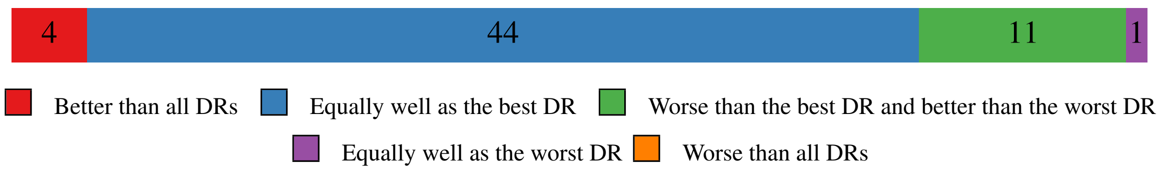


Figure 6.23: Performance of the best ensemble constructed by SEC and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

The ensemble constructed by the vote combination method achieved an improvement of 5.8% over the best DR in the ensemble on the validation set, and an improvement of 3.7% on the test set. Although most DRs which are contained in the ensemble achieve good results on the validation set, their performance is significantly worse when they are applied on the test set. However, this deterioration in their individual performance did not have an influence on the performance of the entire ensemble. This again proves that the individual performance of DRs that form the ensemble is not an indicator of the quality of the entire ensemble. The ensembles which were analysed for the two ensemble construction methods both achieve a similar performance on the two problem instance sets used for testing. However, the results demonstrate that the vote combination method creates ensembles which produce results that will be more in the range of the results achieved by the DRs that form the ensemble, whereas the sum combination method creates ensembles which have a greater possibility of achieving better results than the individual DRs, but can also achieve inferior results in certain occasions.

An additional analysis will also be performed for ESS when it is applied on the ensembles generated by SEC. Table 6.26 represents the analysis of the best ensemble obtained by ESS, for the sum combination method. The table shows that for five problem instances the obtained ensemble achieved results which are at least as good as those obtained by the best individual result of all DRs contained in the ensemble, while for three of those instances the ensemble outperformed the best results achieved by individual DRs. In several occasions, the ensemble can perform well even if only one of the DRs in the ensemble achieves good performance. On four out of five other problem instances, the ensemble did not achieve equally good results as the best DR, but still managed to perform better than the worst rule in the ensemble. On the problem instance 41 the ensemble achieved worse performance than that achieved by the worst DR in the ensemble, even though two DRs in the ensemble individually performed well on that problem instance. Figure 6.24 shows the performance of the ensemble when compared to the individual DRs that form the ensemble. The results show that the ensemble constructed by ESS did not perform well on as many problem instances as the ensemble constructed by SEC. Although the ensemble created by ESS achieved an inferior performance on the validation set, it obtained a much better performance on the test set than the ensemble constructed by SEC.

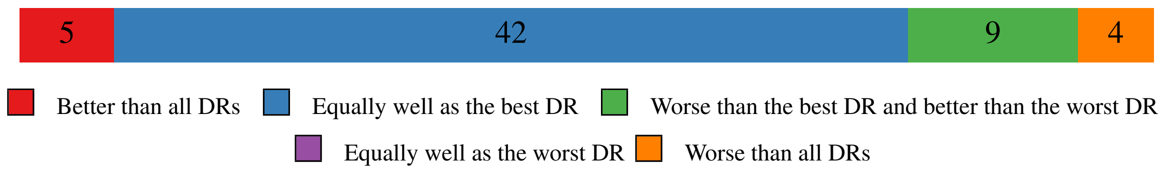


Figure 6.24: Performance of the best ensemble constructed by SEC with ESS and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

The DRs out of which the ensemble is constructed usually do not achieve a good individual performance. On the validation set, only DR 30 achieved a good performance, while rules 46 and 47 achieve an extremely bad performance. Nevertheless, the ensemble can achieve an improvement of 2.7% over the best individual. The situation is similar for the test set as well, since most rules again achieve a quite bad performance. However, the ensemble constructed from these DRs achieve an improvement of 7.5% over the best DR in the ensemble. Therefore, the poor performance of individual DRs did not have an influence on the performance of the

Table 6.26: Performance analysis of the best ensemble generated by SEC with ESS and combined by using the sum combination method

Problem instance index	Individual DRs				Ensemble
	25	30	46	47	
1	0.494	0.264	0.301	0.376	**0.264**
13	0.838	0.870	1.126	1.025	**0.755**
17	0.168	0.204	0.259	0.326	**0.136**
22	2.046	1.889	1.860	1.855	1.860
26	0.978	0.978	1.142	1.131	**0.857**
28	0.091	0.032	0.032	0.032	**0.032**
29	0.506	0.506	0.430	0.554	0.506
39	1.009	0.955	0.999	0.985	1.009
40	1.399	1.233	1.386	1.386	1.386
41	0.091	0.116	0.058	0.058	0.163
Total fitness on all instances	13.87	13.12	14.31	14.45	12.77
Fitness on the test set	16.75	15.67	16.02	16.51	14.49

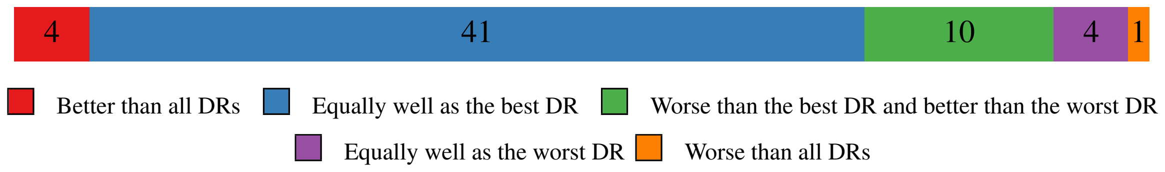


Figure 6.25: Performance of the best ensemble constructed by SEC with ESS and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

ensemble, on neither of the problem instance sets.

Table 6.27 represents the results achieved by the best ensemble created by ESS when using the vote combination method. For five problem instances in the table, the ensemble performs equally well or better than all of the DRs in the ensemble. On the other problem instances, the ensemble also shows to perform very similar to the best DR, like for problem instances 1, 13, and 17. Although some DRs perform poorly on certain problem instances, the better DRs are still able to guide the ensemble towards better solutions. On the entire validation set, the ensemble achieved better values than any of the DRs in the ensemble for four problem instances, while for 41 instances it achieved equal results as the best DR. On the other instances, the ensemble achieved results worse than the best DR, but only in one occasion it achieved a worse result than all DRs in the ensemble. Therefore, the ensemble obtained by ESS does not achieve a good performance on as many instances as the ensemble created by SEC, which also leads the ensemble generated by ESS to achieve worse results on both problem sets.

The generated ensemble outperforms the best DR it consists of by 2.4% on the validation set, and by 4.4% on the test set. The improvements on the validation set were not large since two DRs in the ensemble achieve a good individual performance. However, the improvement is much more evident on the test set where neither of the DRs in the ensemble achieved a good result. Nevertheless, the results are still inferior than that of the ensemble generated by SEC, therefore demonstrating that in this case ESS was unable to improve the performance of the ensembles. Furthermore, the results achieved by the vote combination method, when using ESS, were inferior to those when the sum combination method was applied. Therefore, ESS seems to be more useful for improving the results of ensembles using the sum combination method.

6.6.3 Analysis of ensembles generated by BagGP

Table 6.28 represents the best ensemble obtained by the BagGP method. This result was achieved for the bag size of 40 problem instances, and an ensemble of size nine. The results show that the entire ensemble performs worse than most of the DRs out of which it is constructed. However, this is not surprising since the DRs are evolved completely independently

Table 6.27: Performance analysis of the best ensemble generated by SEC with ESS and combined by using the vote combination method

Problem instance index	Individual DRs			Ensemble
	6	11	46	
1	0.297	0.494	0.301	0.301
13	0.842	0.737	1.126	0.871
17	0.114	0.489	0.259	0.170
22	1.865	1.860	1.860	**1.860**
26	0.857	1.010	1.142	**0.857**
28	0.032	0.091	0.032	**0.032**
29	0.506	0.524	0.430	0.566
39	0.962	0.960	0.999	**0.938**
40	1.233	1.233	1.386	**1.233**
41	0.109	0.091	0.058	0.109
Total fitness on all instances	13.17	13.11	14.31	12.79
Fitness on the test set	15.98	15.72	16.02	15.03

from each other, and therefore it can occur that the entire ensemble does not perform well. Although the ensemble consists of nine DRs, it rarely happens that several DRs in the ensemble achieve the same performance. This is due to the bagging procedure, in which DRs are evolved on subsets of problem instances. Therefore the DRs specialise only on solving the problem instances which were used for its training. Because of this, the performance of individual DRs for certain problem instances can be vastly different, like for problem instances 4, 6, 7, 13, 24, and 47, on which the worst DR in the ensemble mostly achieved a fitness value two times larger than the one obtained by the best DR. The ensemble can match the performance of the best individual DR in several cases (for problem instances 4 and 32), however, on the other instances it usually achieved results which are between those of the best and worst results achieved by DRs in the ensemble. Figure 6.26 shows the performance of the ensemble on the entire training set when compared to the individual DRs that form the ensemble. Because the ensemble achieves a poor performance on many of the problem instances, it did not achieve good results for the training set set.

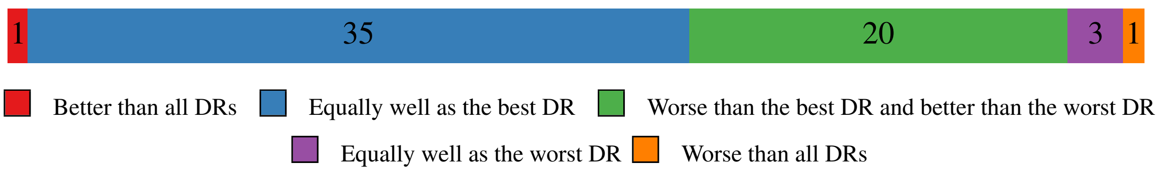


Figure 6.26: Performance of the best ensemble constructed by BagGP and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

As previously mentioned, the ensemble did not achieve a better performance than the best DR contained in it, but rather achieved a result worse by even 9.1% on the training set. Most of the DRs which form the ensemble achieved a better individual performance than the ensemble. On the test set, the constructed ensemble performs quite well, achieving an improvement of 3.2% over the best DR in the ensemble. For the test set, many of the DRs in the ensemble achieve a quite poor performance. Although the selected ensemble achieves a good performance on the test set, the fact that it achieved a bad performance on the training set makes it questionable if this ensemble would perform well on other problem instances.

Table 6.29 represents the analysis of the best ensemble which was evolved by BagGP when the vote combination method and the bag size of 70 instances were used. The first thing which is evident from the selected problem instances is that the ensemble outperforms the best result achieved by any of the DRs on only one problem instance, while on another problem instance it performs equally well. Figure 6.27 denotes the performance of the ensemble when compared to the individual DRs in the ensemble. Although the obtained numbers are similar to those achieved by BagGP with the sum combination method, in this case the individual DRs perform better, which also has a positive effect on the ensemble. This is important for problem instances

Table 6.28: Performance analysis of the best ensemble generated by BagGP and combined by using the sum combination method

Problem instance index	Individual DRs									Ensemble
	0	1	2	3	4	5	6	7	8	
4	0.476	0.826	0.462	0.462	0.674	0.632	0.376	0.698	0.476	**0.376**
6	0.461	0.169	0.325	0.141	0.411	0.191	0.167	0.165	0.474	0.503
7	0.975	1.140	1.150	1.563	1.328	1.314	1.099	0.989	1.251	1.298
13	0.203	0.185	0.201	0.433	0.291	0.359	0.181	0.166	0.186	0.201
22	1.589	1.589	1.606	1.589	1.438	1.438	1.645	1.589	1.589	1.515
24	1.474	1.381	1.381	1.381	2.147	1.493	1.563	1.467	1.474	2.147
32	0.684	0.661	0.727	0.646	0.664	0.662	0.801	0.653	0.719	**0.644**
39	0.784	0.657	0.752	0.828	0.842	0.965	0.692	0.898	0.660	0.744
47	0.492	0.572	0.548	0.503	0.619	0.453	0.585	0.627	0.740	0.607
59	0.256	0.285	0.234	0.235	0.246	0.236	0.243	0.243	0.243	0.247
Total fitness on all instances	15.43	15.70	15.16	15.65	17.13	15.69	15.85	15.62	15.81	16.54
Fitness on the test set	15.39	17.44	15.26	17.50	16.88	17.84	17.20	15.68	15.54	14.77

for which the ensemble does not perform equally well or better than the best individual DR, since in those cases the ensemble has mostly achieved a value similar to the median of the values achieved by the DRs which form the ensemble. Such a behaviour can be observed for problem instances 4, 22, 32, 39, and 49. Therefore, if individual DRs perform better, it should also lead to better performance of the entire ensemble for that problem instance. The results also demonstrate that as more rules in the ensemble perform equally well for a certain problem instance, the higher is the probability that the entire ensemble will also perform in the same way. This can be seen for problem instance 22, where four DRs achieve the same performance, and consequentially the entire ensemble also performs the same. The DRs evolved by the approach specialise for solving different problems, which is evident from the fact that two DRs rarely achieve an equal performance on a given problem instance.

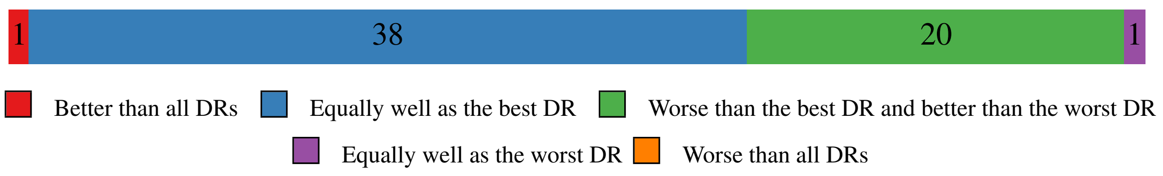


Figure 6.27: Performance of the best ensemble constructed by BagGP and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

Although this ensemble is also unable to outperform the best individual on the training set, it performs worse only by 0.7%, meaning that it achieves an almost equal result as the best DR

Table 6.29: Performance analysis of the best ensemble generated by BagGP and combined by using the vote combination method

Problem instance index	Individual DRs									Ensemble
	0	1	2	3	4	5	6	7	8	
4	0.675	0.675	0.476	0.675	0.476	0.376	0.476	0.476	0.632	0.476
6	0.184	0.275	0.391	0.165	0.224	0.301	0.100	0.363	0.100	0.301
7	1.083	1.070	1.126	2.054	0.914	1.320	0.926	0.994	1.413	**0.914**
13	0.240	0.187	0.194	0.223	0.185	0.213	0.277	0.198	0.185	0.239
22	1.600	1.600	1.526	1.589	1.589	1.589	1.589	1.602	1.515	1.589
24	1.580	1.359	1.381	1.580	1.326	1.474	1.472	1.472	1.474	1.474
32	0.685	0.535	0.701	0.640	0.685	0.670	0.691	0.685	0.606	0.685
39	0.781	0.745	0.732	0.730	0.659	0.779	0.827	0.795	0.770	0.795
47	0.476	0.492	0.498	0.492	0.492	0.643	0.492	0.579	0.572	0.492
59	0.273	0.227	0.243	0.238	0.242	0.256	0.288	0.243	0.232	**0.112**
Total fitness on all instances	15.01	15.12	15.29	16.03	14.73	15.55	15.12	15.78	15.33	14.83
Fitness on the test set	16.14	16.69	16.95	16.94	16.03	16.00	15.86	15.50	17.12	14.88

in the ensemble. The ensemble achieves a much better performance on the training set than the previous ensemble which used the sum combination method. On the test set, the ensemble outperforms the best DR of the ensemble by 4%. Therefore, the ensemble achieved a good performance on both problem sets. In the end, the ensemble with the vote combination method achieved a better performance on the training set, and a similar performance on the test set, when compared to the ensemble with the sum combination method.

Table 6.30 represents the results achieved by the best ensemble created by ESS, when using the sum combination method. This ensemble was evolved with the bag size of 30 problem instances. The results in the table demonstrate that ESS was able to select those DRs which lead to good results of the ensemble on both the validation set and the test set. For six problem instances in the table, the ensemble achieved equally good or better results than any of the DRs contained the ensemble. However, the other four problem instances show much more interesting things about the behaviour of the ensemble. On those problem instances, usually two or three DRs perform rather well, while the other DRs perform poorly for the problem instance. However, when the performance of the ensemble is observed on these problem instances, it tends to achieve performance which leans more towards the values achieved by better DRs in the ensemble. Therefore, the ensemble as a whole also achieved good results on most of the problem instances. Figure 6.28 represents the performance of the ensemble on all problem instances in the validation set, when compared to the individual DRs.

Table 6.30: Performance analysis of the best ensemble generated by BagGP with ESS and combined by using the sum combination method

Problem instance index	Individual DRs				Ensemble
	1	6	7	8	
1	0.301	0.376	0.711	0.711	**0.294**
2	0.798	0.771	0.914	0.788	**0.596**
13	0.839	0.996	0.981	1.015	**0.839**
17	0.335	0.172	0.105	0.109	0.110
22	2.149	1.705	1.840	1.698	1.741
26	0.857	0.978	1.142	1.238	0.978
29	0.506	0.506	0.506	0.806	**0.506**
40	1.386	1.223	1.348	1.751	1.329
41	0.170	0.109	0.075	0.075	**0.075**
57	0.067	0.080	0.070	0.067	**0.066**
Total fitness on all instances	14.06	13.18	14.26	15.36	12.55
Fitness on the test set	15.75	15.48	15.67	17.27	14.41

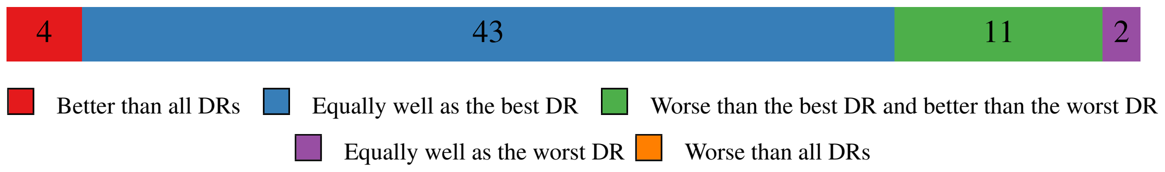


Figure 6.28: Performance of the best ensemble constructed by BagGP with ESS and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

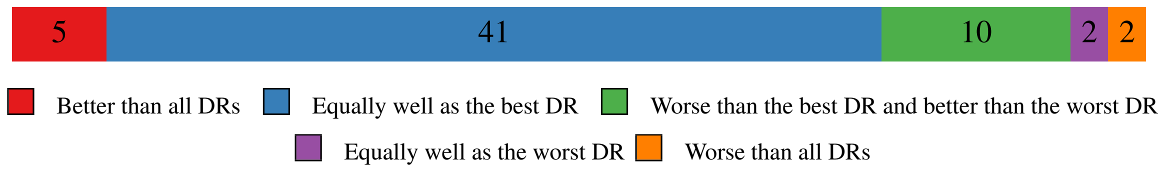


Figure 6.29: Performance of the best ensemble constructed by BagGP with ESS and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

The obtained ensemble achieved a good performance on both problem sets. For the validation set, the ensemble outperformed the best individual DR in the ensemble by 4.8%. On the other hand, on the test set the ensemble outperformed the best DR by 6.9%. Out of all the ensembles and DRs tested in this chapter, this ensemble achieved the overall best performance on the test set. Therefore, ESS obtained an ensemble which not only improved the performance of BagGP, but also reduced the size of the best ensemble to only four DRs.

The results for ESS with the vote combination method are presented in table 6.31. The ensemble was evolved by using the bag size of 40 problem instances. The results demonstrate that even for the vote combination method ESS creates an ensemble which achieves good performance for both the validation and test set. For five of the presented problem instances the ensemble performs at least equally well as the DR which achieves the best result for the problem instance. For most other problem instances the ensemble usually achieved values which lean towards the values obtained by better DRs, like for problem instances 13, 17, and 57. This is especially important for problem instances in which certain DRs achieve quite bad values, but the entire ensemble performs well since most of the other DRs in the ensemble obtain a good performance. Instance 17 is an example of such a situation, where rule 1 achieves a quite bad performance, but in the end it does not have a large effect on the performance of the entire ensemble for this problem instance. However, in cases where the majority of DRs in the ensemble perform poorly, it is more likely that the entire ensemble will not achieve a good performance (like for problem instance 37). Figure 6.29 shows the performance of the ensemble on the entire validation set, when compared to the individual DRs in the ensemble.

The considered ensemble achieved a quite good performance on both problem sets. For the validation problem set, the ensemble achieved an improvement of 8.9% over the best DR

Table 6.31: Performance analysis of the best ensemble generated by BagGP with ESS and combined by using the vote combination method

Problem instance index	Individual DRs				Ensemble
	1	6	7	8	
1	0.356	0.351	0.462	0.599	**0.264**
2	0.596	0.596	1.331	0.775	**0.596**
13	0.924	0.763	0.907	0.759	0.763
17	0.684	0.240	0.093	0.151	0.121
22	1.860	1.860	1.685	1.860	1.860
26	1.215	0.858	0.936	0.842	**0.834**
37	0.120	0.117	0.151	0.126	0.185
40	1.192	1.386	1.141	1.233	**1.141**
41	0.109	0.163	0.091	0.163	**0.091**
57	0.065	0.079	0.097	0.079	0.076
Total fitness on all instances	13.81	13.68	13.55	14.10	12.35
Fitness on the test set	16.31	15.71	16.05	15.62	14.85

in the ensemble. On the other hand, for the test set the ensemble achieved an improvement of approximately 4.9%. The ensemble also achieved a better performance on the validation set when compared to the ensemble obtained by using the sum combination method. However, on the test set the ensemble achieved inferior results to those obtained by the ensemble with the sum combination method. This demonstrates that it is hard to obtain an ensemble which achieves the best results for both of the problem sets. When compared to the ensemble obtained by BagGP, the ensemble constructed by ESS achieved a similar performance on the test set. However, the ensemble constructed by ESS consisted of only four DRs, whereas the ensemble constructed by BagGP consisted out of nine DRs. This demonstrates that ESS obtains ensembles with a much smaller sizes, but equal or better performance.

6.6.4 Analysis of ensembles generated by BoostGP

Table 6.32 represents the best ensemble obtained by BoostGP for the sum combination method, without the use of weights. The table shows that DRs which are evolved in later iterations usually perform better on those instances which were not solved well by DRs evolved in previous iterations. This behaviour is expected since the problem instances which are not solved optimally will gain more importance in each further iteration. On four problem instances in the table, the ensemble achieved better or equal values as all individual DRs. Out of these four problem instances, only for instance 5 did the ensemble achieved a better performance than the best DR in the ensemble. In cases where the ensemble was not able to achieve an equally good result as the best DR, the ensemble still performed better than the worst DR for that problem instance. The results achieved by the ensemble for those problem instances converge more to the values achieved by worse DRs. This behaviour can be observed for problem instances 22, 24, and 38. Figure 6.30 represents the performance of the ensemble on the entire test set, when compared to the individual DRs in the ensemble.

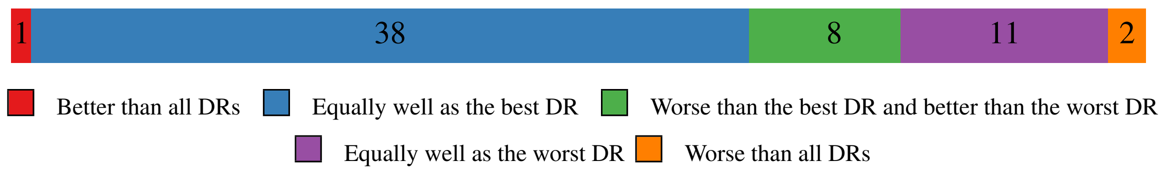


Figure 6.30: Performance of the best ensemble constructed by unweighted BoostGP and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

Because the ensemble did not achieve a good performance for several problem instances, it was unable to outperform the result on the training set achieved by the best individual DR in the ensemble, but rather the ensemble achieves an inferior result by 3.1%. On the test set, the ensemble achieved an improvement of approximately 1.9%. The ensemble obtained by

Table 6.32: Performance analysis of the best ensemble generated by unweighted BoostGP and combined by using the sum combination method

Problem instance index	Individual DRs									Ensemble
	0	1	2	3	4	5	6	7	8	
4	0.476	0.476	0.476	0.485	0.476	0.462	0.476	0.476	0.632	**0.462**
5	0.753	0.753	0.802	0.786	0.731	0.786	0.731	0.731	0.731	**0.692**
19	0.368	0.366	0.374	0.324	0.309	0.261	0.344	0.347	0.403	0.364
21	0.326	0.262	0.231	0.231	0.231	0.262	0.231	0.262	0.293	0.293
22	1.589	1.589	1.589	1.613	1.589	1.589	1.589	1.589	1.438	1.600
24	1.326	1.381	1.474	1.474	1.402	1.474	1.381	1.381	1.431	1.472
33	0.112	0.131	0.112	0.123	0.123	0.112	0.123	0.112	0.112	**0.112**
38	0.484	0.555	0.573	0.547	0.581	0.545	0.545	0.562	0.582	0.557
53	0.322	0.312	0.308	0.312	0.318	0.308	0.317	0.308	0.308	0.326
59	0.265	0.256	0.256	0.243	0.256	0.243	0.260	0.256	0.260	**0.243**
Total fitness on all instances	14.49	14.63	14.89	15.00	14.46	14.75	14.52	14.48	14.69	14.91
Fitness on the test set	15.47	15.72	15.46	15.50	16.06	15.40	15.83	16.20	16.16	15.10

BoostGP did not achieve an equally good performance on the test set as the previously described approaches.

Table 6.33 shows the analysis of the best ensemble evolved by the unweighted BoostGP approach, with the vote combination method. The evolved DRs which form the ensemble again perform well on the training set. However, the entire ensemble performs worse than most of the DRs contained in it, which is expected since the DRs are again evolved almost independently from each other. On only one problem instance denoted in the table, the ensemble achieved an equally good result as the best DR in the ensemble. This happened for the problem instance 24, where five DRs achieved the best result, and therefore the entire ensemble also obtained the same performance. On the remaining nine problem instances, the ensemble always achieved a value which is not inferior to the result obtained by the worst DR in the ensemble. Figure 6.31 represents the performance of the ensemble on the entire test set, when compared to the individual DRs in the ensemble. On the problem instances on which the ensemble performed better than the worst DR and worse than the best DR, the ensemble achieved a result which often represents the median value of the performance of DRs contained in the ensemble, thus allowing the ensemble to achieve a good performance over a large number of instances. The results obtained by this ensemble once again demonstrate that the ensembles which use the vote combination method are more inclined to achieve results which are in the range of the best and worst value obtained by the individual DRs.

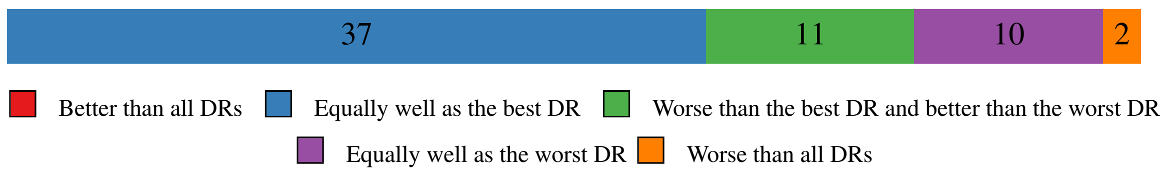


Figure 6.31: Performance of the best ensemble constructed by unweighted BoostGP and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

Table 6.33: Performance analysis of the best ensemble generated by unweighted BoostGP and combined by using the vote combination method

Problem instance index	Individual DRs							Ensemble
	0	1	2	3	4	5	6	
4	0.476	0.476	0.462	0.476	0.476	0.476	0.503	0.476
5	0.753	0.731	0.798	0.688	0.731	0.741	0.688	0.731
19	0.371	0.357	0.263	0.387	0.353	0.256	0.296	0.377
21	0.293	0.231	0.231	0.231	0.262	0.326	0.293	0.262
22	1.589	1.600	1.515	1.589	1.589	1.589	1.589	1.589
24	1.381	1.381	1.474	1.474	1.381	1.381	1.381	**1.381**
33	0.112	0.112	0.112	0.123	0.123	0.090	0.128	0.123
38	0.582	0.571	0.465	0.546	0.558	0.483	0.533	0.544
53	0.308	0.308	0.308	0.308	0.312	0.307	0.308	0.308
59	0.259	0.256	0.238	0.243	0.256	0.226	0.271	0.256
Total fitness on all instances	14.80	14.71	14.44	14.73	14.51	14.38	14.49	14.77
Fitness on the test set	16.00	15.76	15.97	16.19	16.05	15.77	15.13	14.99

The constructed ensemble is again unable to outperform the best DR out of which it is constructed. In this case, the constructed ensemble achieves an inferior performance of 2.7% when compared to the best DR in the ensemble. The improvement on the test set which the ensemble achieves over the best DR contained in it is approximately 1%. The improvement is relatively small because a DR in ensemble achieves a good individual performance on the test set. Nevertheless, this ensemble achieved better results for both problem sets than the previously created ensemble which used the sum combination method. Therefore, the vote combination method seems to be slightly more appropriate for the BoostGP approach.

Table 6.34 represents the results achieved by ESS for the sum combination method. The best ensemble achieved by ESS consists of only two DRs. Both DRs which form the ensemble achieve a good performance by themselves, but combined into an ensemble their performance is increased even further. For six problem instances, the ensemble achieves equally good or better results than the individual DRs. For instances 18 and 22, both DRs achieve a similar value, however, the ensemble performs better than both of these rules. In certain cases it can happen that the ensemble performs worse than any of the DRs in the ensemble, like for problem instance 10. Figure 6.32 demonstrates the performance of the ensemble on the validation set, when compared to the individual DRs in the ensemble.

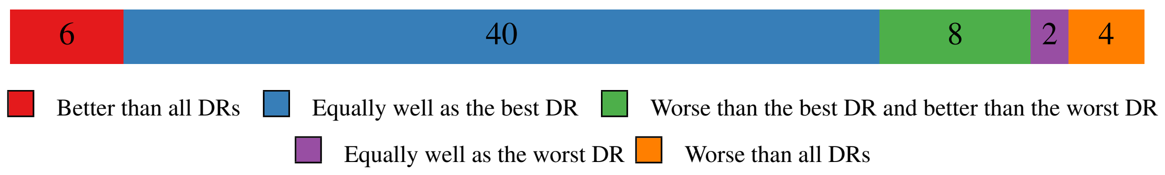


Figure 6.32: Performance of the best ensemble constructed by unweighted BoostGP with ESS and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

On the validation set, the ensemble achieved an improvement of 4.9% over the best individual in the ensemble. On the other hand, on the test set, the ensemble achieved an improvement of 5.0%. Therefore, on both problem sets, the ensemble achieved a similar performance improvement over the best DR in the ensemble. The ensemble obtained by ESS also achieved a better result on the test set than the best ensemble achieved solely by BagGP with the sum combination method, since by using ESS an improvement of 2.2% was achieved. In addition, the ensemble constructed by ESS consists of only two DRs, while the best DR obtained by BagGP consisted of nine DRs. Therefore, ESS also constructed an ensemble with a significantly smaller size.

Table 6.35 represents the best ensemble created by ESS when using the vote combination method. The table shows that ESS generated an ensemble which consists of DRs that specialise on solving different problem instances. Similarly as in previous examples, the entire ensemble has a larger probability of achieving a good performance if more DRs in the ensemble obtain the

Table 6.34: Performance analysis of the best ensemble generated by unweghted BoostGP with ESS and combined by using the sum combination method

Problem instance index	Individual DRs		Ensemble
	1	6	
0	0.285	0	0.018
2	0.596	0.596	**0.596**
10	0.440	0.338	0.499
12	0.019	0.014	**0.014**
15	0.001	0.001	**0**
16	0.001	0.044	**0**
18	0.850	0.852	**0.621**
22	1.865	1.813	**1.705**
25	0.113	0.005	0.076
37	0.127	0.177	0.152
Total fitness on all instances	13.74	14.06	13.06
Fitness on the test set	15.59	15.54	14.77

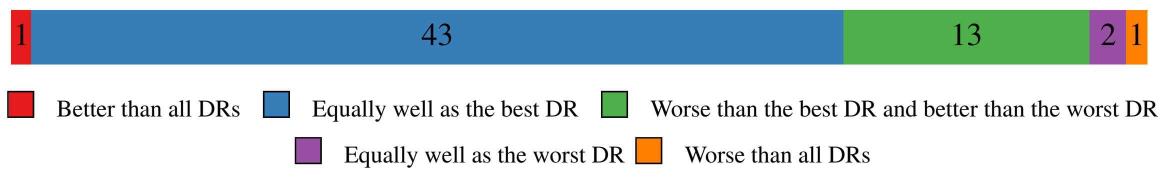


Figure 6.33: Performance of the best ensemble constructed by unweighted BoostGP with ESS and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

best performance for a problem instance. This behaviour can be observed on problem instances 22, 23, and 27. In cases where the individual DRs in the ensemble achieve various performances for a single problem instance, the entire ensemble achieves values similar to the best value obtained by any DR, like for problem instances 18, 26, and 33. Even if the ensemble can not match the very best value, it will still perform quite well, since it often achieved a performance which is better than that of most of the DRs which form the ensemble. Figure 6.33 represents the performance of the ensemble on the entire validation set, when compared to the individual DRs in the ensemble.

On the validation set the ensemble obtained by ESS for the vote combination method achieved an improvement of 3.9% over the best DR in the ensemble. On the other hand, on the test set the ensemble achieved an improvement of 3%. Therefore, the constructed ensemble outperforms the best individual DR in the ensemble on both of the problem sets. When compared to the ensemble generated by ESS with the sum combination method, the ensemble with the vote combination method achieves a similar performance on the test set, while on the validation set it achieves a much better performance. The ensemble generated by ESS also achieved a better result than the ensemble generated without using ESS, thus demonstrating the ability of ESS to improve the performance of ensembles generated by other approaches.

Table 6.36 represents the best ensemble constructed by the weighted BoostGP approach. Most of the DRs perform well by themselves, and this also seems to reflect on the performance the entire ensemble. The ensemble achieved an equal performance as the best DR in the ensemble on only two instances denoted in the table. On the other problem instances, the ensemble either performs almost as good as the best DR, like for instances 5, 53, and 59, or it achieves a performance similar to the worst DR in the ensemble, like for problem instances 24 and 38. Figure 6.34 represents the performance of the ensemble on the entire test set, when compared to the individual DRs in the ensemble.

Even though the ensemble did not perform equally good as the best DR on most of the problem instances, it nevertheless achieved only a slightly worse performance than the best DR, amounting to around 0.9%. On the test set the ensemble achieved a better result than the best individual DR by 1.6%. Using the confidences as weights for the DRs is beneficial, since this variant of the approach achieved better results on both problem sets than the ensemble which did

Table 6.35: Performance analysis of the best ensemble generated by unweighted BoostGP with ESS and combined by using the vote combination method

Problem instance index	Individual DRs					Ensemble
	0	1	2	3	8	
9	0.396	0.397	0.270	0.270	0.378	0.300
13	0.772	0.939	0.945	0.853	0.839	**0.770**
18	0.745	0.735	0.734	0.553	0.656	0.596
22	1.860	1.685	1.840	1.685	1.889	**1.685**
23	0.225	0.160	0.225	0.351	0.160	**0.160**
26	0.842	0.857	0.924	0.857	0.978	0.857
27	0.026	0.026	0.026	0.033	0.026	**0.026**
33	0.218	0.220	0.193	0.330	0.196	0.218
36	0.211	0.213	0.206	0.231	0.180	0.237
40	1.386	1.233	1.386	1.518	1.246	**1.233**
Total fitness on all instances	14.83	12.98	14.39	13.44	13.27	12.48
Fitness on the test set	15.44	15.23	16.06	15.91	15.55	14.78

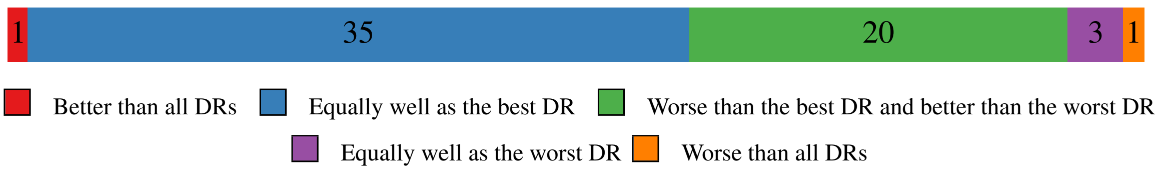


Figure 6.34: Performance of the best ensemble constructed by weighted BoostGP and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

Table 6.36: Performance analysis of the best ensemble generated by weighted BoostGP and combined by using the sum combination method

Problem instance index	Individual DRs							Ensemble
	0	1	2	3	4	5	6	
	Confidences							
	0.185	0.424	0.646	0.813	0.866	1.027	0.949	
4	0.476	0.476	0.476	0.476	0.476	0.476	0.476	**0.476**
5	0.753	0.731	0.876	0.802	0.741	0.775	0.753	0.753
19	0.396	0.348	0.275	0.360	0.219	0.341	0.348	0.341
21	0.293	0.183	0.231	0.262	0.326	0.231	0.326	0.293
22	1.438	1.515	1.589	1.589	1.589	1.589	1.589	**1.438**
24	1.552	1.381	1.381	1.474	1.381	1.474	1.381	1.552
33	0.128	0.112	0.112	0.112	0.101	0.123	0.128	0.128
38	0.549	0.590	0.460	0.576	0.477	0.604	0.586	0.581
53	0.308	0.302	0.331	0.308	0.323	0.308	0.308	0.308
59	0.256	0.260	0.246	0.243	0.232	0.243	0.243	0.246
Total fitness on all instances	14.52	14.42	14.82	14.76	14.36	15.10	14.89	14.49
Fitness on the test set	15.62	15.78	15.19	15.56	15.55	17.45	15.99	14.95

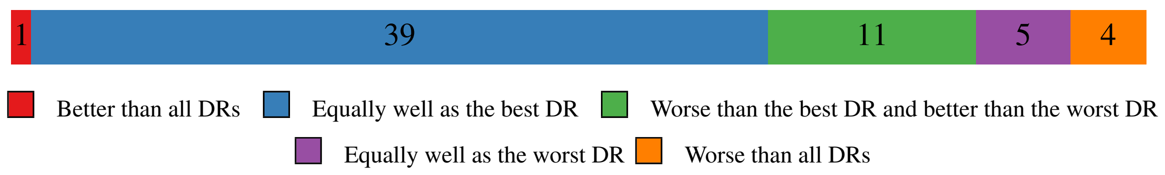


Figure 6.35: Performance of the best ensemble constructed by weighted BoostGP and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

not use the confidences as weights. Therefore, for the sum combination method, the confidences represent information which can be used to improve the performance of the ensembles.

Table 6.37 represents the best individual obtained by the weighted BoostGP approach, with the vote combination method. For only two problem instances in the table, the ensemble achieved an equally good performance as the best DR in the ensemble. For both of these problem instances, several DRs in the ensemble achieved the overall best result, which seems to help the ensemble to also achieve the best result. However, on most of the other problem instances the ensemble did not perform equally good as the best DR, and usually achieved a quite bad performance. For example, for problem instances 4, 5, 19, and 38, the ensemble achieved poor results when compared to the DRs in the ensemble. In most of these problem instances, the ensemble performed either equally bad as the worst DR in the ensemble, or even worse. Figure 6.35 shows the performance of the ensemble on the entire training set, when compared to the individual DRs in the ensemble.

Based on the performance of the ensemble on individual problem instances, it is immediately clear that the ensemble will not achieve a good performance on the training set. The results show that the ensemble performs worse than any of the DRs in the ensemble, while when compared to the best DR in the ensemble, it achieved an inferior result by 7%. On the test set, the ensemble achieved a better performance than all DRs in the ensemble, and outperforms the best DR in the ensemble by around 1.6%. By comparing the results achieved by this approach with the ensemble that did not use confidences as weights for the DRs, it is evident that the additional use of weights did not lead to significantly better performance neither on the training set, nor on the test set. When compared to the ensemble which used the sum combination method, the ensemble with the vote combination method achieved a similar value for on the test. However, on the training set the ensembles using the vote combination method achieved a significantly worse result.

Table 6.38 represents the best ensemble achieved by ESS with the weighted BoostGP approach, for the sum combination method. ESS reduced the size of the ensemble to only two DRs, thus demonstrating that for the sum combination method it is preferable to use DRs of smaller sizes. Out of the ten problem instances denoted in the table, the ensemble achieved at least an equal performance as the best DR in six of them. Although for problem instances 18

Table 6.37: Performance analysis of the best ensemble generated by weighted BoostGP and combined by using the vote combination method

Problem instance index	Individual DRs							Ensemble
	0	1	2	3	4	5	6	
	Confidences							
	0.182	0.407	0.694	0.800	0.903	0.911	1.005	
4	0.476	0.462	0.476	0.376	0.476	0.229	0.476	0.476
5	0.786	0.883	0.753	0.786	0.808	0.731	0.753	1.136
19	0.372	0.333	0.368	0.362	0.332	0.356	0.366	0.407
21	0.231	0.231	0.262	0.231	0.231	0.262	0.326	0.262
22	1.589	1.589	1.515	1.589	1.600	1.589	1.589	1.589
24	1.381	1.474	1.474	1.474	1.472	1.381	1.472	1.472
33	0.112	0.112	0.106	0.123	0.131	0.123	0.097	0.123
38	0.627	0.550	0.620	0.542	0.549	0.549	0.586	0.642
53	0.308	0.308	0.308	0.312	0.308	0.308	0.322	**0.308**
59	0.256	0.246	0.243	0.243	0.243	0.246	0.256	**0.243**
Total fitness on all instances	14.69	14.68	14.58	14.52	14.73	14.24	14.72	15.31
Fitness on the test set	16.17	16.48	15.20	15.75	15.76	16.03	16.79	14.96

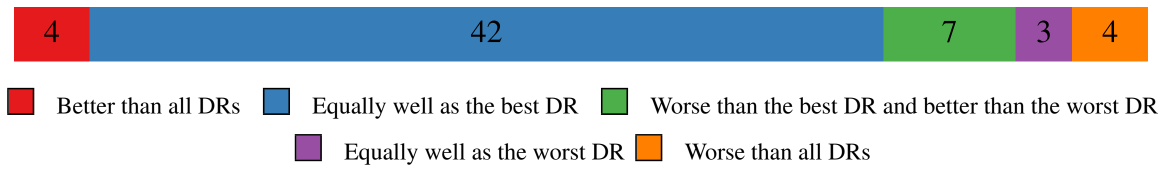


Figure 6.36: Performance of the best ensemble constructed by weighted BoostGP with ESS and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

and 37, the ensemble did not achieve a performance which is equal to that of the best DR, it still achieved results which were close to the value achieved by the best performing DRs. Therefore, for the ten denoted problem instances, the ensemble achieves a good performance on most of the problem instances. Figure 6.36 represents the performance of the ensemble on the entire validation set, when compared to the individual DRs that form the ensemble.

Although DR 3 achieves a good individual performance, the ensemble nevertheless performs better on both problem sets. On the validation set, the ensemble achieved an improvement of 4.1%, while on the test set the ensemble achieved an improvement of 1.3%. The ensemble constructed by ESS obtained a better value on the test set, than the ensemble which was constructed only by BoostGP. In addition, the size of the ensemble obtained by BoostGP was 7, which demonstrates that ESS was able to create an ensemble with not only a better performance, but also a much smaller size. By comparing the results to the ensemble obtained by ESS for the unweighted BoostGP approach, it is evident that the ensemble constructed by ESS for the weighted BoostGP approach achieved a slightly better result on the validation set. However, the performance on the test set was the same. Therefore, using the confidences as weights or not, does not have a significant influence on the performance of ESS.

Table 6.39 represents the best individual obtained by ESS when using the vote combination method. Although most of the times until now ESS achieved the best performance for smaller ensembles, this time the best performance was obtained by using an ensemble of size nine. For four of the problem instances in the table, the ensemble achieved at least and equally good performance as the best DR in the ensemble. Since the ensemble consists out of a large number of DRs, it is difficult for the ensemble to achieve the best result for each problem instance. However, on problem instances 10, 18, and 57 the ensemble tends to achieve a performance similar to the performance of the better DRs in the ensemble. An interesting result of the ensemble can be observed for problem instance 23. In this case, all DRs achieve only two different values for the optimisation criterion. Although more DRs achieved a better value for for this instance, in the end the ensemble was unable to also obtain this performance. However, this is probably due to the fact that the DRs which achieved an inferior performance on this problem instance had a larger confidence value. Therefore, these DRs will have a larger influence when the ensemble is used to perform the scheduling decisions. Figure 6.37 represents the performance on the entire

Table 6.38: Performance analysis of the best ensemble generated by weighted BoostGP with ESS and combined by using the sum combination method

Problem instance index	Individual DRs		Ensemble
	3	5	
	Confidences		
	0.795	1.056	
6	0.025	0.001	**0.001**
10	0.336	0.312	0.328
13	0.893	1.032	**0.882**
18	0.588	0.726	0.626
23	0.225	0.225	**0.225**
25	0.082	0.092	**0.074**
28	0.032	0.032	**0.032**
37	0.134	0.166	0.140
39	0.971	1.028	**0.967**
57	0.067	0.067	0.074
Fitness on the training set	13.48	14.21	12.93
Fitness on the test set	14.97	15.45	14.77

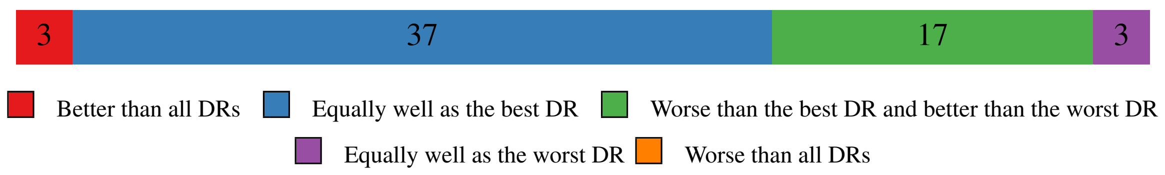


Figure 6.37: Performance of the best ensemble constructed by weighted BoostGP with ESS and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

validation set, when compared to the individual DRs contained in the ensemble.

Although the ensemble consisted out of DRs which achieve a good performance on both problem sets, the ensemble achieved an improvement of only 0.8% on the validation set, and 1.3% for the test set. Although the improvements are not large, it must be stressed out that a single DR is not able to achieve the best performance on both problem sets. By comparing the achieved results to the ones achieved by ESS, but when using the sum combination method, it is evident that the sum method achieved better results for both problem instance sets. However, the ensemble constructed by ESS achieved a better performance than the ensemble obtained only by using the weighted BoostGP approach, thus again outlining the ability of ESS to improve the ensembles achieved by other ensemble learning approaches. Finally, by comparing the results with the ensemble obtained by ESS with the unweighted BoostGP approach, it is evident that ESS obtained a better ensemble when the confidences are not used as weight for the individual DRs.

6.6.5 Analysis of ensembles generated by cooperative coevolution

Table 6.40 represents the analysis of the best ensemble created by the cooperative coevolution approach, with the sum combination method. This result was achieved by using the second configuration, which performs more iterations. The table illustrates some interesting behaviours of an ensemble which is evolved by the cooperative coevolution approach. First of all, the DRs which form the ensemble achieve bad results individually. This is especially true for DR 1, which achieved terrible results if applied independently. On all problem instances DR 0 achieved better results than DR 1. However, the ensemble constructed from these two DRs achieved good results, which demonstrates that the DRs in the ensembles created by cooperative coevolution heavily depend on each other, and thus perform poorly by themselves. The table shows that for most of the presented problem instances, the ensemble can achieve equal or better values than either of the DRs that form the ensemble. It is quite interesting to note that for many of those problem instances both DRs achieve bad performance, but the ensemble performs nevertheless quite well (for example on problem instances 16, 21, 47, and 51). On problem instances 17 and 23, on which the ensemble did not achieve the same performance as

Table 6.39: Performance analysis of the best ensemble generated by weighted BoostGP with ESS and combined by using the vote combination method

Problem instance index	Individual DRs									Ensemble
	0	1	2	3	4	5	6	7	9	
	Confidences									
	0.184	0.399	0.676	0.817	0.871	0.934	0.972	1.024	1.065	
6	0.002	0.008	0.129	0	0	0.024	0.002	0.001	0.144	**0**
10	0.356	0.395	0.256	0.294	0.391	0.196	0.269	0.362	0.365	0.242
13	0.921	0.879	1.081	0.954	0.842	0.874	1.154	0.908	0.813	**0.768**
18	0.855	0.746	0.795	0.761	0.739	0.776	0.669	0.661	0.725	0.684
23	0.160	0.160	0.160	0.225	0.160	0.160	0.255	0.255	0.255	0.255
25	0.113	0.113	0.005	0.034	0.103	0.005	0.135	0.092	0.005	0.117
28	0.091	0.091	0.032	0.032	0.091	0.032	0.032	0.032	0.145	**0.032**
37	0.179	0.164	0.143	0.121	0.136	0.182	0.140	0.160	0.152	**0.115**
39	1.181	0.982	1.002	1.044	1.062	1.050	0.989	1.066	1.031	1.030
57	0.074	0.068	0.068	0.067	0.074	0.068	0.070	0.070	0.067	0.068
Fitness on the training set	15.12	13.13	13.82	14.48	13.91	13.63	13.63	14.16	13.69	13.03
Fitness on the test set	16.00	15.76	15.97	16.19	16.05	15.77	15.13	15.25	16.25	14.94

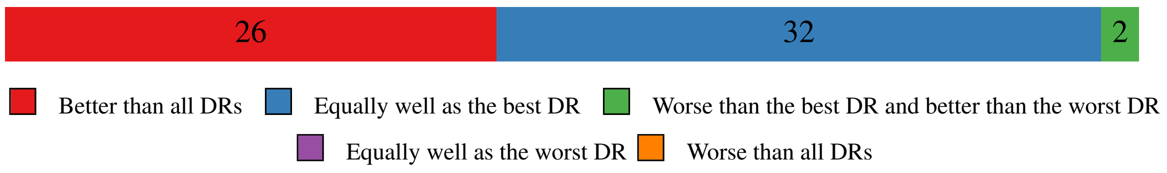


Figure 6.38: Performance of the best ensemble constructed by cooperative coevolution and combined by using the sum combination method compared to the performance of individual DRs out of which it was constructed

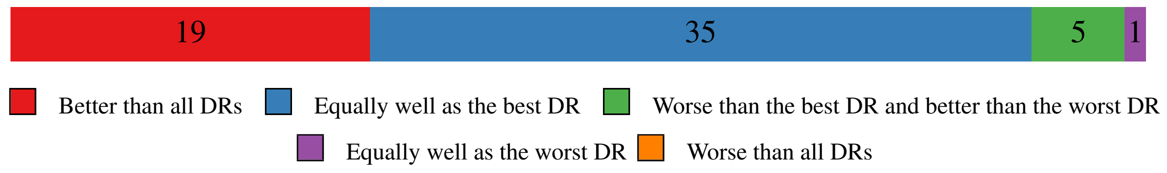


Figure 6.39: Performance of the best ensemble constructed by cooperative coevolution and combined by using the vote combination method compared to the performance of individual DRs out of which it was constructed

the best DR in the ensemble, it achieved results which are only slightly worse than those of the best DR, therefore these problem instances do not have an effect on the performance of the entire ensemble. Figure 6.38 represents the performance of the ensemble on the test set, when compared to the individual DRs of the ensemble.

Because of the bad individual performance of the DRs, the ensemble achieved an improvement over the best DR in the ensemble of 32% on the training set, and an improvement of 37% on the test set. By comparing the result achieved on the training set to that of several previous ensembles, the ensemble created by cooperative coevolution approach achieved one of the best results for the training set. However, for the test set the obtained performance is among the worst performances of all ensembles. Therefore, it is highly probable that the cooperative coevolution overfits on the training set and achieves a good performance on it. However, as a consequence the ensemble looses the ability to perform well on unseen problem instances.

The analysis of the best ensemble evolved by cooperative coevolution with the vote combination method is presented in Table 6.41. This ensemble was evolved by using the first configuration, with the smaller number of iterations. Even when the vote combination method is used, the DRs which form the ensemble perform poorly when they are applied individually on the problem instances. The ensemble once again performs better than any of the individual DRs on their own, which means that the DRs are once again heavily dependant on each other. The table denotes that the ensemble achieved equal or better values than any of the DRs which form the ensemble in nine out of ten examples in the table. For the remaining problem instance, it achieved the same results as the second best DR in the ensemble. Figure 6.39 shows the performance of the ensemble on the entire training set, when compared to the individual DRs in the ensemble.

Table 6.40: Performance analysis of the best ensemble generated by cooperative coevolution and combined by using the sum combination method

Problem instance index	Individual DRs		Ensemble
	0	1	
1	0.949	3.085	**0.924**
7	1.500	5.286	**0.965**
16	1.181	14.17	**0.564**
17	0	3.176	0.002
21	1.161	3.714	**0.326**
23	0.797	4.757	0.802
38	0.827	13.63	**0.490**
42	0.222	3.710	**0.222**
47	0.736	10.61	**0.577**
51	0.403	20.30	**0.360**
Total fitness on all instances	20.88	234.0	14.20
Fitness on the test set	20.73	277.4	15.11

Table 6.41: Performance analysis of the best ensemble generated by cooperative coevolution and combined by using the vote combination method

Problem instance index	Individual DRs			Ensemble
	0	1	2	
1	0.945	1.112	1.256	**0.924**
7	1.283	1.127	1.525	**0.914**
16	1.022	2.781	8.095	**0.630**
17	0	0.338	6.421	**0**
21	0.293	0.904	0.231	0.293
23	0.802	0.856	0.823	**0.802**
38	0.733	1.431	1.194	**0.606**
42	0.222	0.222	0.222	**0.222**
47	0.588	0.939	0.643	**0.455**
51	0.243	0.642	0.372	**0.243**
Total fitness on all instances	18.10	32.98	57.15	14.20
Fitness on the test set	18.02	36.49	71.31	15.14

Once again the ensemble achieved large improvements over the individual DRs in the ensemble, because of their poor individual performance. The ensemble achieved an improvement of 21.5% on the training set, and an improvement of 16% on the test set. Based on the results, it is evident that the procedure achieved a good value for the training set, and therefore probably overfitted on it as was the case when the sum method was used. For both ensemble combination methods, the ensemble achieved a similar performance, therefore the cooperative coevolution approach does not show preference for using either ensemble combination method.

6.7 Conclusion

Ensemble learning approaches represent a simple and powerful way of improving the performance of different classifier systems and decision makers. These approaches have rarely been used with GP, especially when GP is used for creation of new DRs. In the few occasions when ensembles of DRs were used, it was possible to obtain improved results over the individual DRs.

The aim of this chapter was to apply several different ensemble learning methods for solving certain scheduling problems. Out of the five applied approaches, three of them were selected from the literature (BagGP, BoostGP and cooperative coevolution), while the other two were proposed in this thesis (SEC and ESS). Through experiments, the influence of different parameters like the ensemble combination method and the ensemble size, was analysed for each of the approaches. The results obtained by all the tested approaches were compared to the results obtained by different standard manually designed DRs, and automatically generated DRs. Finally, an analysis was performed to gain insights in the composition of ensembles generated by different ensemble learning approaches.

The results obtained through the chapter show that the applied ensemble learning approaches achieved superior performance over individual DRs, whether they were designed manually or evolved by GP. Out of the tested methods, the best performance was achieved by the proposed SEC method which, although being quite simple, outperforms the other ensemble learning methods for most experiments. The main reason why BagGP and BoostGP usually achieved worse performance than SEC is due to the fact that they independently evolve DRs that form the ensemble, which can mean that the ensemble contains DRs which have a negative influence on its performance. The results of these ensembles can be improved with the ESS method, which filters out the unneeded DRs from the obtained ensembles. The analysis performed on the best ensembles also gives a deeper insight on how the different ensemble learning methods work.

Based on the results achieved through this chapter, it can be concluded that ensembles of DRs are able to achieve significantly better results than individual DRs. Therefore, these methods have shown great potential for solving different scheduling problems. Further research in this topic can be directed into testing other ensemble learning approaches, or ensemble combination methods. For the SEC approach it would be interesting to analyse more ensemble construction methods, or even try to design new construction methods based on the knowledge acquired through the analysis of the constructed ensembles. Finally, it would also prove interesting to apply these methods to other machine environments to get a better insight into their performance.

Chapter 7

Selection of DRs based on problem instance characteristics

In Section 2.4.2 a list of 26 manually designed DRs for the unrelated machines environment was given. The reason why such a substantial number of DRs was designed comes from the fact that a single DR can not perform well for all possible problem instances. This is clearly illustrated in the study by Branke and Pickardt [277], in which they use an evolutionary algorithm to design problem instances for which certain DRs from the literature perform suboptimal choices. These problem instances can then be used to reveal and analyse the limitations of an existing DR. Therefore, many DRs which perform well for problems with specific characteristics are designed. However, designing DRs for various situations is not enough. If the characteristics of the scheduling problem are unknown before the execution of the system, it is not possible to select the appropriate DR which would achieve the best performance on the given problem. Although GP does solve the problem of manual creation of DRs, the issue of selecting the appropriate DR out of all the automatically designed DRs, for the given problem instance, still remains. However, several studies have already dealt with the problem of choosing the appropriate DR for the currently considered problem instance. These studies used different machine learning methods to determine which of the available DRs should be used to perform the next scheduling decision. However, in all studies the selection was performed usually between a few manually selected DRs, and in none of the studies was the selection process applied on automatically generated DRs. In addition, most of the studies used neural networks to perform the selection of the DRs, while other machine learning methods were applied in rare occasions.

This chapter will describe how machine learning methods can be used together with DRs generated by GP, to define a procedure which can be applied for automatic selection of DRs based on the characteristics of the current problem instance. In the first part of the chapter the literature overview of automatic selection of DRs will be provided. After that, the procedure which is used for automatic selection of automatically generated DRs will be presented. Two variants of this procedure will be tested, one of which will assume that the characteristics of the scheduling problem are available before the start of the system execution, thus making it possible for the procedure to determine which DR should be applied for solving the problem prior to the start of the system execution. The second variant will not have this information available, therefore it will need to approximate the characteristics of the scheduling problem during the execution of the system, and then use this approximation to select the appropriate DR for solving this problem instance. Both of these variants will be tested on several problem instances to analyse their performance. The chapter will be concluded with a short overview of the achieved results and directions for further research.

7.1 Overview of DR selection literature

The selection of dispatching rules based on the system status and problem characteristics has been studied to a certain extent in the literature. Pierreval [278] applied an expert system to dynamically select dispatching rules in the flexible manufacturing systems. Sun and Yih [279] have proposed a controller for a manufacturing cell which uses a neural network to select a proper DR based on the current performance of the system. Pierreval [280] has also used neural networks to select the appropriate DR for a simplified job shop environment consisting of two work centres. In the aforementioned work the neural network performs its decision by using information about the configuration of the shop floor, characteristics of the manufacturing program, and the optimised performance criteria. Liu and Dong [281] have proposed an algorithm for job shop scheduling problems which uses neural networks to determine the appropriate DRs. Their procedure uses simulation to evaluate the efficiency of all the evolved DRs. The results of the simulations are then used for training the neural network which is used in the real-time scheduling process, to select the DRs that should be applied. Pierreval and Mebarki [282] have analysed dynamic rule selection in a manufacturing system, where the rule selection is carried out each time a machine becomes available, and the rules are selected based on certain system parameters.

Instead of using a machine learning approach, Yu et al. [283] opted to use a fuzzy inference based system to select appropriate DRs. The system uses two fuzzy rules and several environment variables for selecting the DR which should be used for scheduling. Subramaniam et al. [284] have also used fuzzy logic for defining a scheduler which dynamically selects the most appropriate DR from a set of candidate DRs. Their results have shown that the fuzzy scheduler performs better than some other commonly used DRs, and requires the same computational time as the simple DRs. In [285], Subranamian el al. have proposed a scheduling method based on the analytic hierarchy process which dynamically selects the most appropriate DR from several available rules. El-Bouri and Shah [39] have used neural networks for DR selection in the job shop environment. Their approach does not only select a single DR, but rather it selects an appropriate DRs for each machine in the shop. The experiments have shown that by using such an approach, they achieve better results than by using the same DR for every machine. Shiue and Guh [286] used a hybrid learning methodology which applies a neural network to select the appropriate DR for the current system conditions, and a GA which is used to select the best set of attributes for the input layer of the neural network.

Mouelhi-Chibani and Pierreval [40] used a neural network approach for selecting the most suited DR after each machine becomes available. In this study the neural network is not trained on prepared training instances, but rather by using a simulation approach. This means that weights of the neural network, which acts as a decision maker, are optimised with an optimisation method to achieve the best performance on the simulation. The method is able to automatically select efficient DRs for the given situation. Heger et al. [41] have used Gaussian processes for the selection of an appropriate DR. In their approach they used two system parameters, system utilisation and the due date factor, as well as three manually defined DRs. Their research showed that it was possible to achieve better results by rule switching than by applying a single DR for the entire problem instance. Zahmani et al. [287] also use a simulation approach to extract data that can be used to infer which DR should be used in which situations. An overview of other research concerned with the automatic selection of DRs can be found in [239, 240, 288].

7.2 DR selection procedure

The aim of the DR selection procedure, which is used in this chapter, is to determine the appropriate DR for solving the currently considered problem instance. However, to select the appropriate DR the procedure needs to learn which DRs should be used in which situations, and then apply this knowledge on unseen scheduling problems. The rest of this section will describe all the individual parts of the DR selection procedure.

7.2.1 Problem instance features

The first, and probably most important step which needs to be performed, is to identify the most informative features of the problem instances, which can be used to perform the decision on which of the available DRs should be applied for the current problem instance. The goal here is to find the smallest, but also the most informative set of features. In this chapter the influence of the following five features will be analysed:

- Expected total duration of the schedule: $\hat{p} = \frac{\sum_{j=1}^{n}\sum_{i=1}^{m} p_{ij}}{m^2}$,
- Due date tightness: $DDT = 1 - \frac{\sum_j (d_j - r_j)}{n\hat{p}}$,

- Due date range: $DDR = \frac{\max_j(d_j - r_j) - \min_j(d_j - r_j)}{\hat{p}}$,
- Machine job ratio: $MJR = \frac{m}{n}$,
- Release time tightness: $RTT = 1 - \frac{\bar{r}}{\hat{p}}$,

where n represents the total number of jobs in the problem instance, m represents the total number of machines in the problem instance, while $\bar{r}$ represents the average value of all job release times in the problem instance.

Since the *Twt* criterion will be optimised in the experiments, the due date tightness and due date range are the most important features, due to the fact that they provide the most information about the due dates of the jobs. The expected total duration approximates the makespan of the schedule, while the machine job ratio gives an information on the average number of jobs that will be scheduled per machine. The release time tightness gives information of the distribution of the release times.

7.2.2 The learning process

The next step in the automatic rule selection procedure is to define the learning process which will be used to learn which problem instance should be solved by which DR. To achieve this, two things need to be defined: the *learning set* on which the classifier will learn which DR should be used on which problem instance, and a classifier which will be trained on the aforementioned learning set, and will determine which DR to use for a new problem instance.

In order to learn which DR is appropriate for solving which problem instance, it is required to define the learning set which will be used to train the classification method. The learning set will consist of features which were calculated for each problem instance, and labels that denote which of the available DRs are associated with which problem instances. In the last subsection the feature calculation technique of the problem instances was defined, but still it is required to determine which DR should be used for solving which problem instance. For the purpose of associating DRs to problem instances a procedure called the *grouping association scheme* (GAS) is defined.

Algorithm 7.1 represents the pseudo-code of GAS. This procedure uses a set of problem instances and a set of DRs to construct the learning examples. In the first step, for each problem instance the set of DRs which obtain the best solution for that problem instance is determined. All problem instances which have only one DR in their set are marked as *solved*, since for them there is no ambiguity regarding which DR should be associated with them. In addition, all DRs which are associated with *solved* problem instances are collected in the *rlist* set. In the next phase, it is determined for which problem instances the DRs contained in *rlist* can not obtain the best solution, and those problem instances are collected in a set denoted as *nonopt*. The idea is now to add the minimum number of DRs to *rlist*, so that for each problem instance there is a DR which can find the best solution for it. In order to achieve this, for each DR the number of problem instances in *nonopt* for which it finds the best solution is calculated. The DR which finds the best solution for most of the problem instances in *nonopt* is added to *rlist*, while the problem instances for which it finds the best solution, are removed from *nonopt*, and marked as solved. This is repeated until *nonopt* becomes empty, since then *rlist* contains DRs which can find the best solution for each problem instance. The entire procedure is done in this way to minimise the number of DRs which will be used as labels. In the final step the procedure determines the DRs which should be associated with problem instances which are not yet marked as solved. For each unsolved problem instance and each DR in *rlist* which finds the best solution for this problem instance, the average Euclidean distance is calculated between the features of the current problem instance and all problem instances which are until now associated with the considered DR. The DR with the smallest Euclidean distance is chosen and associated with the considered problem instance. This part represents a very simple grouping procedure, the goal of which is to group problem instances with similar characteristics to the same DR, and also to balance the dataset so that the best DR is not used as the label for most of the problem instances, since by using an unbalanced set would lead the procedure to mostly use a single DR on new problem instances. Finally, the features of each problem instance are calculated, and the DR associated to that problem instance is used as the label. An important characteristic of the GAS procedure is that it does not necessarily generate a learning set which contains all of the DRs which were supplied to the procedures, meaning that it automatically eliminates DRs for which it determined that they do not provide useful information.

After its construction, the learning set can be used to learn a classifier, which will be able to determine which DRs should be applied for unseen problem instances. Unfortunately, the selection of a good classifier for a given classification problem is in itself a very difficult task, especially since many different classification methods exist. For that reason several popular classification methods will be used in this chapter, and their performances will be compared with each other. The following machine learning methods will be used: k nearest neighbours (knn) [289], naive Bayes [290], logistic regression (log) [291], support vector machine (SVM) [292], artificial neural network (ANN) [293] and C4.5 [294]. Many of the aforementioned classification methods have parameters which should also be fine-tuned to achieve better results. However, fine-tuning parameters of all the methods would be too time consuming, therefore the values of the parameters where chosen as a rule of thumb. The selected parameter values are denoted in Table 7.1. Although the ANN method is stochastic because of randomness in the initialisation of the weights, it will nevertheless be trained only once to reduce the computation time. The classification process was performed by using the Accord machine learning library [295].

Algorithm 7.1 Pseudo-code of the GAS procedure

```
1: Let $P$ denote the set of all problem instances
2: Let $R$ denote the set of all available DRs
3: Let $rlist$ represent an empty set of DRs
4: Evaluate all DRs in $R$ on $P$
5: for each problem instance $i$ in $P$ do
6:     Let $opt[i]$ denote the set of DRs which achieve the best result for this problem instance
7:     Let $rule[i]$ denote the DR chosen for this problem instance (empty in the beginning)
8: end for
9:
10: for each problem instance $i$ with only one DR in $opt[i]$ do
11:     Set $rule[i]$ to the DR in $opt[i]$, and add the DR to $rlist$
12:     Mark the problem instance as solved
13: end for
14:
15: Let $nonopt$ denote the set of all unsolved problem instances $i$ for which $opt[i] \cap rlist = \emptyset$
16: while $nonopt$ is not empty do
17:     For each DR define $count$ which denotes the number of problem instances in $nonopt$
        for which the given DR achieves the best solution
18:     Let $BR$ denote the rule with the largest $count$ value. If there are more DRs with the same
        $count$ value, choose the one which has the best total fitness on all problem instances
19:     Add $BR$ to $rlist$
20:     for each problem instance $i$ in $nonopt$ that contains $BR$ in $opt[i]$ do
21:         Remove $i$ from $nonopt$
22:         Mark $i$ as solved
23:     end for
24: end while
25:
26: for each unsolved problem instance $i$ do
27:     for each DR in $rlist$ which is also contained in $opt[i]$ do
28:         Calculate the average Euclidean distance between the features of the current prob-
            lem instance, and all problem instances which have the chosen DR in their $rule$ field
29:     end for
30:     Set $rule[i]$ to the DR with the smallest average Euclidean distance
31: end for
32:
33: for each problem instance $i$ in $P$ do
34:     Create a learning example with features of the problem instance and with $rule[i]$ repre-
        senting the label associated with the features
35: end for
```

Table 7.1: Parameter values used by the machine learning methods

Algorithm	Parameters
knn	Neighbourhood size: 3, 5 and 7
Naive Bayes	Regularisation: 1e-5
Log	Training procedure: lower bound Newton Raphson Stopping criteria: maximum of 100 iterations or error less than 1e-6
SVM	Kernel: Gaussian Optimisation: sequential minimal optimisation (SMO)
C4.5	-
ANN	Hidden layer count: 1 Hidden layer nodes: 3, 5 and 7 Weight initialisation: Nguyen-Widrow Learning algorithm: backpropagation with Levenberg-Marquardt Stopping criteria: maximum of 30 iterations or error less than 1e-6

7.2.3 The decision process

Now that the the process of learning the classifier has been defined, the only thing which still needs to be specified is the way in which the procedure will select the DR that should be used for solving the current problem instance. For that purpose, two different ways of selecting DRs will be analysed in this chapter, the static and dynamic selection process.

The static DR selection process

The static DR selection process will be performed with an assumption that the features of the problem instances are known in advance with sufficient precision, even though the scheduling process is performed under dynamic conditions. This means that even though the exact information about the properties of jobs are unknown before the execution of the system, the basic features are known before the scheduling is performed. Because of this it is possible to apply the selection procedure and determine which DRs should be applied for solving the scheduling problem before the system starts with its execution. Therefore, the DR selection procedure, which selects a concrete DR that should be used for solving the currently considered problem instances, is performed prior to the execution of the system.

The dynamic DR selection process

Unfortunately, it is very unlikely that the features of the problem instances will be known in advance, before the execution of the system. This is possible if there exists a similar problem which was already solved previously, and from which it would be possible to approximate the features, or if the system has already been executing for a longer period of time from which it would be possible to calculate the features of the problem. However, for new problem instances the features will not be known in advance, which renders the previously described approach unusable. Therefore, the DR selection procedure needs to be extended so that it can be used even in dynamic environments. The intuition behind the extension is to allow the system to execute for a certain amount of time, during which the DR selection procedure will collect information about the system and use that information to approximate the features which will be used to select the appropriate DR that should be applied for scheduling. Naturally, this adaptation introduces several new parameters into the DR selection procedure, since it now has to deal with dynamic scheduling conditions.

The first thing which needs to be defined is at which points in time the DR selection procedure will be applied to determine the DR that should be used to create the schedule. In this thesis, three mechanisms of determining the moment in time at which the DR selection procedure should be applied, will be tested. The first mechanism uses the DR selection procedure when a certain number of jobs is released into the system. The second mechanism is similar, however, it performs the DR selection procedure when a certain number of jobs is scheduled. The third mechanism uses a fixed time interval after which it will perform the DR selection procedure, regardless of the number of jobs which were released or executed during that period.

The previous mechanism only defines at which point in time the DR selection procedure should be applied. However, the mechanism does not specify how the features of the scheduling problem should be calculated, or how often the procedure should be applied. The most simple way is to perform the decision only once, and to calculate the features of the scheduling problem based on all jobs released so far. On the other hand, it is also possible to perform the decisions several times, each time after a certain period of time has elapsed, or a certain number of jobs is scheduled or released. In this variant there are two ways in which the features of the problem instance can be approximated. The first way is to use all jobs from the start of the system to calculate the features of the problem instance. On the other hand, it is also possible that at each decision period, the features of the problem instance are calculated based only on those jobs which were released since the last decision point.

One thing still needs to be defined so that the entire procedure can be applied, and that is which DR will be used in the beginning, until the DR selection procedure is executed for the first time. Naturally, any DR can be selected as the initial rule, and the performance of the entire procedure will depend on this selection. In the experiments performed in this chapter, the rule which achieved the best results on the training set will be selected as the initial rule.

In addition to the previously described adaptations, an additional feature will be introduced for the dynamic DR selection procedure. This feature will be denoted as *load disbalance*, and is defined as $LD = \frac{\max_i(L_i) - \min_i(L_i)}{\max_i(L_i)}$, where L_i represent the sum of processing times for machine i, but only of those jobs which achieve the shortest processing time on machine i. This additional feature is introduced since through the experiments it was observed that this information could also provide to be useful for the classification methods.

7.3 Static DR selection procedure

In this section the performance of the static DR selection procedure will be analysed. Since there are several parameters upon which the effectiveness of this procedure depends, the first part of the section will be concerned with the analysis of the influence of different parameter values on the performance of the procedure. After good values for the parameters have been obtained, the performance of the DR selection procedure will be compared to the results obtained by using only a single DRs.

7.3.1 Parameter analysis of the DR selection procedure

This section will present the influence of different parameters on the DR selection procedure. Through the section the influence of the applied feature set, classification method, learning set size, and number of DRs will be analysed. For each parameter several values were selected, and the DR selection procedure was executed for all value combinations of the considered parameters. Since it is hardly possible to analyse the influence of all parameters at the same time, the influence of each parameter will be analysed independently from other parameters. This means that when a concrete parameter is analysed, for each parameter value the results of all experiments which were executed by using that parameter value will be aggregated. In that way it is possible to analyse the influence of only one parameter at a time. The DR selection procedure will be trained on learning sets of different sizes, while the performance of the procedure will be denoted on an independent validation set. In all experiments the *Twt* criterion will be optimised.

The influence of the feature set, which is used by the DR selection procedure, will be analysed first. Table 7.2 denotes the feature set combinations which will be tested. The *DDR* and *DDT* features will always be included in the feature sets, since they represent the most informative features when the *Twt* criterion is optimised. The remaining feature sets include the rest of the features in various combinations.

Figure 7.1 shows the influence of the different feature set combinations on the performance

Table 7.2: Feature set combinations used by the static DR selection procedure

Feature set index	Features
1	DDR, DDT
2	$DDR, DDT, \hat{p}$
3	DDR, DDT, RTT
4	DDR, DDT, MJR
5	$DDR, DDT, \hat{p}, RTT$
6	$DDR, DDT, \hat{p}, MJR$
7	DDR, DDT, RTT, MJR
8	$DDR, DDT, \hat{p}, MJR, RTT$

of the DR selection procedure. As expected, not all feature combinations will lead to equally good results. For example, it is interesting to note that by using only the due date related features the DR selection procedure will not obtain good results. If the RTT feature is used in addition to the due date related features, the procedure achieves even worse results. However, if the $\hat{p}$ or MJR features are included in the feature set, the procedure will achieve better results. Therefore, it is evident that the due date related features do not contain enough information for the DR selection procedure to perform well, but that in addition to those features it is necessary to include features which contain information about the size of the problem which is solved. The best median value of the experiments is achieved by using the feature set denoted with the index 8, which consists all features, while the second best median value is achieved by the feature set consisting of the two due date related features and the $\hat{p}$ feature. Therefore, out of the features which do not use due date related information, the $\hat{p}$ feature provides the most information to the DR selection procedure.

The influence of different classification methods on the DR selection procedure is illustrated in Figure 7.2. Although the choice of the classification method is shown to have an influence on the performance of the DR selection procedure, the influence is not as large as for the choice of the feature set. Although all classification methods achieve quite similar results, it can nevertheless be noticed that some methods perform better to a smaller extent. The best results for the median values are achieved by the knn, ANN, and C4.5 classification methods. Among the different values of the neighbourhood parameter which was used by the knn method, the best median value was achieved when using a neighbourhood of 5 samples. Both the smaller and larger parameter values achieve a larger median value, which should denote that a good parameter value was selected. On the other hand, the ANN method achieved the best median values when either a hidden layer consisting of five or seven nodes was used. It is possible that the

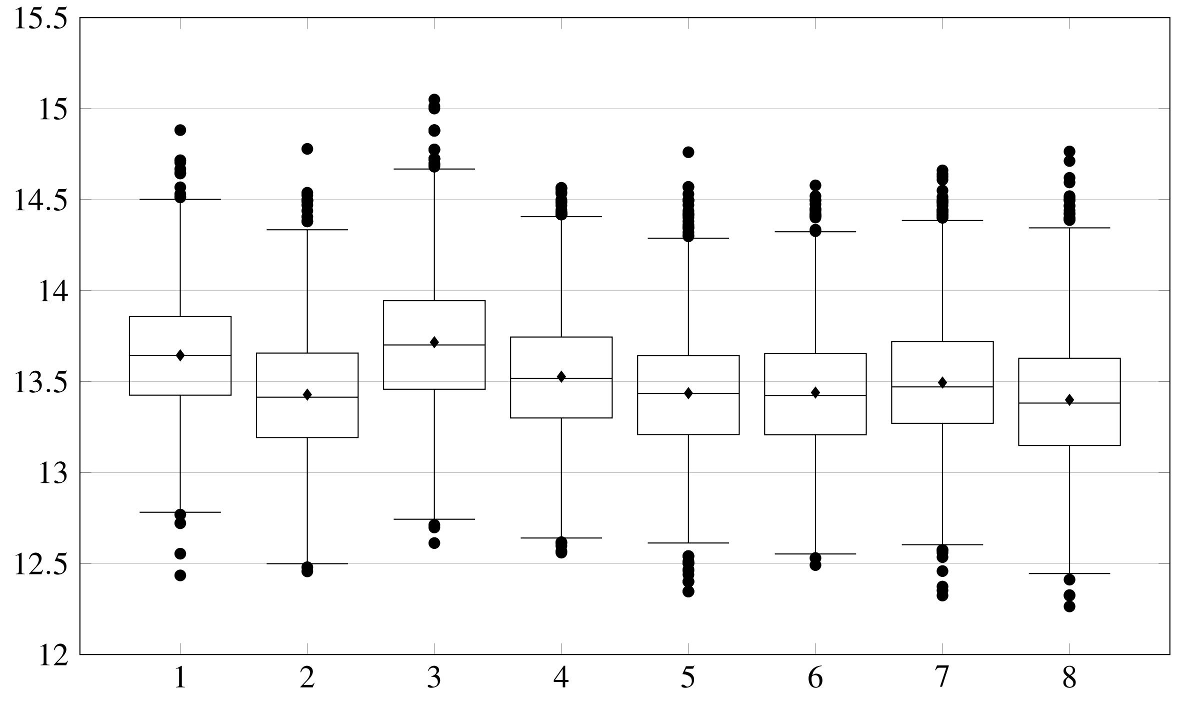


Figure 7.1: Influence of the feature set combination on the results obtained by the DR selection procedure

results could be improved with a further increase in the number of nodes in the hidden layer, however this also largely increases the time needed to train the model.

The size of the learning set which is used to train the classification methods also has a large influence on the performance of the DR selection procedure. If a too small learning set is used, then there will not be enough samples from which classification methods will be able to infer new knowledge. On the other hand, the use of a too large learning set size leads to an increased time needed to perform the training of the classification methods. In addition, increasing the size of the learning set does not necessarily lead to better performance, since it could be possible to create smaller learning sets which contain all the necessary information for the classification methods. Figure 7.3 represents the influence of the learning set size on the performance of the DR selection procedure. The figure shows that the procedure does not achieve good results when a learning set of 60 problem instances is used. However, as the size of the learning set increases, so does the performance of the DR selection procedure which is trained on that set. The procedure achieves the best median value when the learning set of 600 problem instances is used, however it is also evident that for this learning set size the procedure achieved more dispersed results. For smaller learning set sizes, similar median values can be achieved as for the case when 600 problem instances are used, and the results tend to be less dispersed. Surprisingly, it was noticed that the procedure does not perform well when the size of 300 problem instances is used. The most likely reason why the procedure did not perform well in this case, is due to the possibility that for this learning set size an

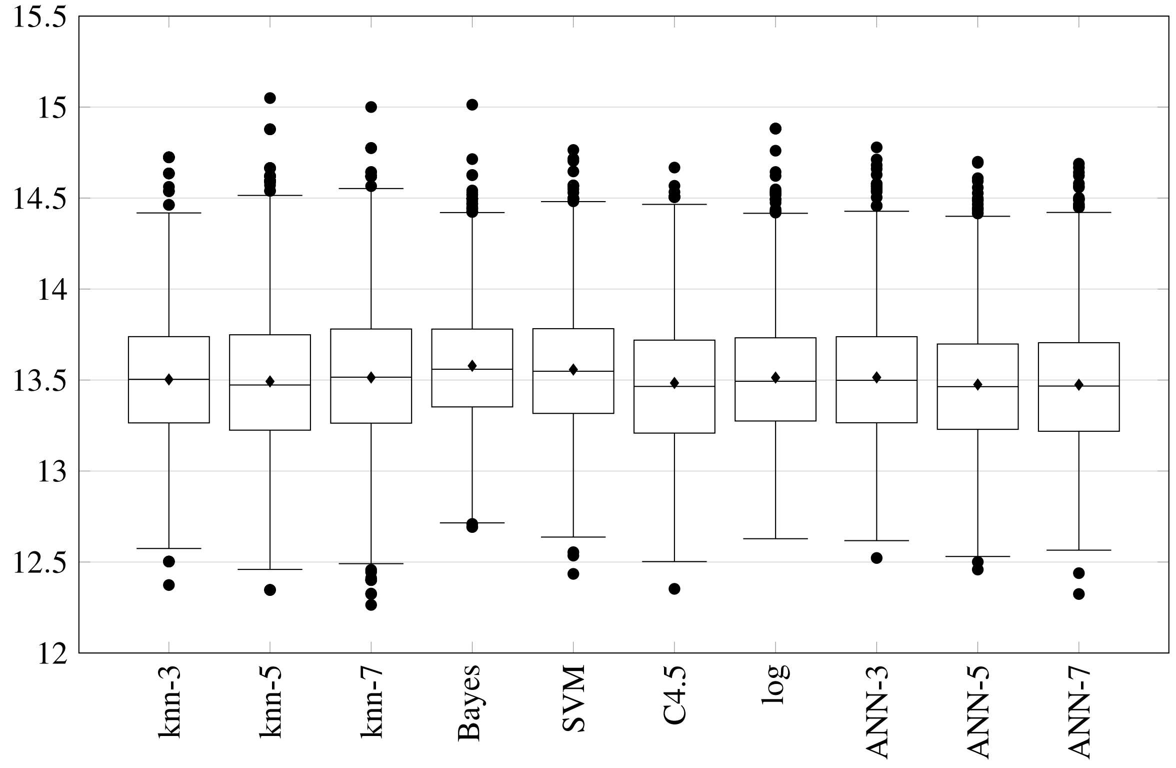


Figure 7.2: Influence of the classification methods on the results obtained by the DR selection procedure

unrepresentative learning set was generated, and therefore the classification method could not extract the necessary knowledge from that set. Based on all the previous observations, it seems that the size between 120 and 240 problem instances is the most appropriate for the learning set size.

The final parameter, whose influence will be analysed, is the number of DRs that will be used when generating the learning set by the GAS method. Although the GAS method will in itself try to minimise the number of DRs which will be used as labels, it could nevertheless prove beneficial to supply only a subset of available DRs to the GAS method. Therefore, the performance of the DR selection method will be tested on different learning sets, which were created by supplying a different number of DRs to the GAS method. In each experiment, a number of DRs which achieve the best performance on the training set will be selected and supplied to the GAS method. The number of DRs which will be used will range from two to fifty DRs. However, since this leads to a large number of results, the experiments were grouped into five groups by the number of DRs which were used, as shown in Figure 7.4. The figure shows that the worst results of the DR selection procedure are achieved when up to 10 DRs are supplied to the GAS method. This means that by largely limiting the number of DRs which are provided to the GAS method, the entire procedure will not be expressive enough when applied on unseen problem instances. The best median value was obtained when 11 to 20 DRs were supplied to the GAS method. When the largest number of DRs is supplied to the GAS

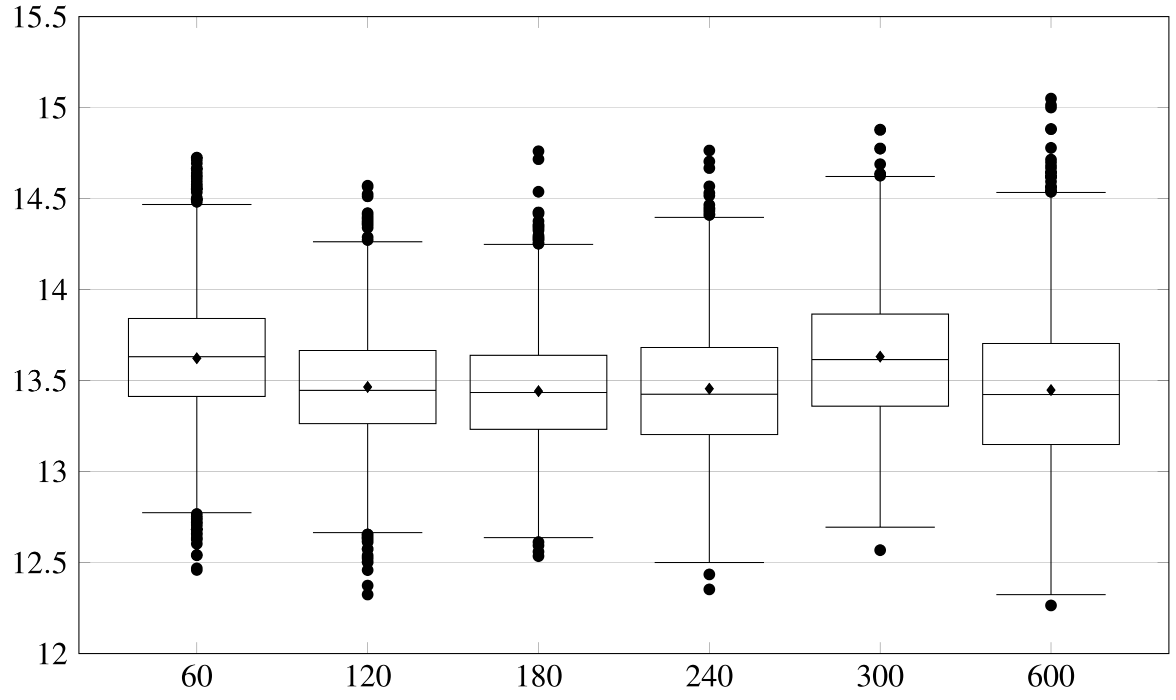


Figure 7.3: Influence of the learning set size on the results obtained by the DR selection procedure

procedure, the results start to slowly deteriorate once again. The reason for this is probably due to the fact that including too many DRs in the GAS procedure will result in learning sets which will contain more distinct DRs as labels, which makes the training of classifiers more difficult. Therefore, based on all the previous observations, the DR selection procedure achieves the best performance if a moderate number of DRs is provided to the GAS procedure (between 10 and 40 DRs).

7.3.2 Performance comparison with a manually selected DR

Based on the parameter sensitivity analysis performed in the previous section, it is now possible to select better parameter values for the DR selection procedure, and test the effectiveness of the procedure for the selected parameter combinations. Table 7.3 represents the results achieved by the static DR selection procedure for several selected parameter combinations. The performance of the procedure is presented on two data sets. The first data set, denoted as the validation set, was used to perform the analysis of the parameter sensitivity. Therefore, to provide a more objective performance measure, the DR selection procedure was also tested on an independent and unseen problem instance set, which is denoted as the test set. In addition to the results obtained by the DR selection procedure, the table will also include results obtained by a manually selected DR generated by GP. Naturally, there are many ways in which the DR could be selected, however, in this chapter it was decided to select the DR which achieved the overall best performance on the training set. This means that after all DRs have been evolved, each of them will

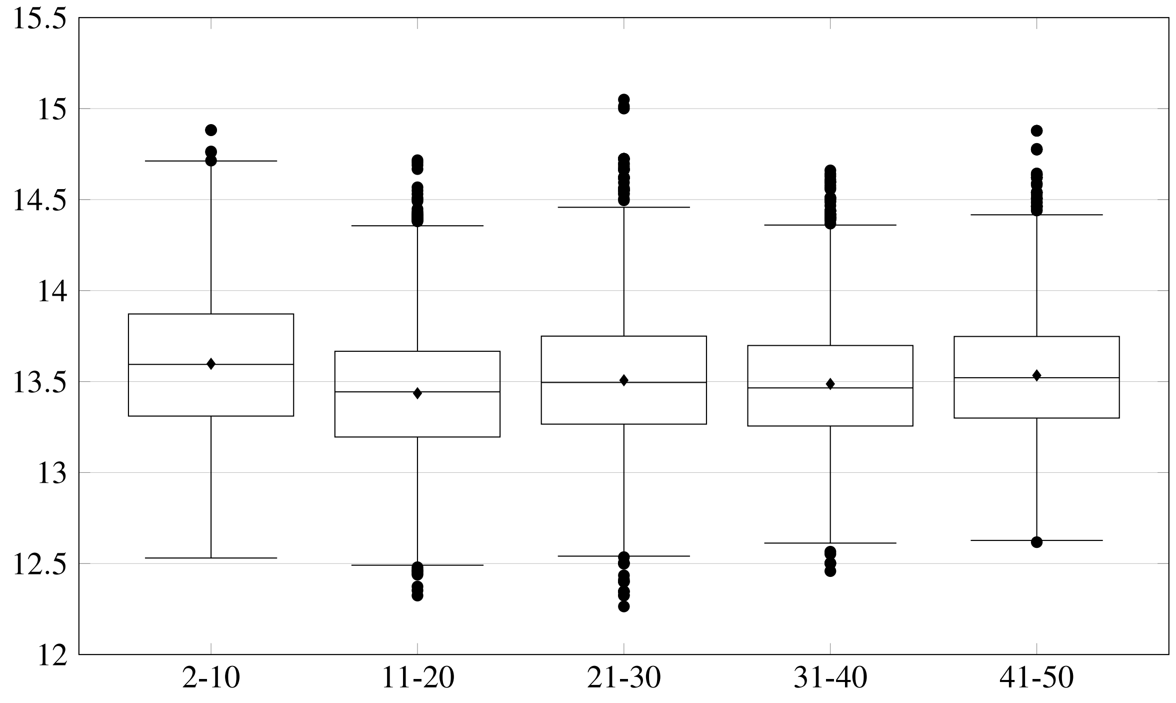


Figure 7.4: Influence of the number of DRs on the results obtained by the DR selection procedure

be evaluated on the training set which was used to evolve them, and the one which achieves the best performance on this set will be selected and applied on unseen problem instances, namely the validation and test sets.

The results denoted in the table show that the DR selection procedure achieved better results than the selected DR generated by GP. The amount of improvement which was achieved depends quite heavily on the selected parameter values. The DR selection procedure achieved very good results when using the knn-5 classification method. When knn-5 is applied together with feature set 2, the DR selection method achieved improvements of 9.4% for the validation set, and 5.3% for the test set when compared to the manually selected DR. Similar improvements can be achieved when knn-5 is used with feature set 6. The benefit of the knn-5 classification method is that it achieved a good performance on both problem instance sets. In both cases, 17 DRs were used by the GAS method for creating the learning set. On the other hand, the performance of C4.5 is shown to vary between the validation and test set more than that of knn-5. For example, when C4.5 is applied with feature set 7, the DR selection procedure achieved a quite good result on the validation set, with an improvement over the manually selected DR of 10.3%. On the other hand, on the test set, the procedure achieved an improvement of only 2.7% over the manually selected DR. In the other two cases where C4.5 was used, the improvements on the validation set were much smaller, around 6%, whereas better improvements were achieved on the validation set, ranging up to 5%. It seems that for the C4.5 classification method it is more difficult to perform extremely well on both problem sets. Regarding the number of DRs which were used by with the C4.5 classification method, in two occasions 16 DRs were

Table 7.3: Results of the DR selection procedure for several selected parameter values

DR selection procedure parameters	*Twt* value on the validation set	Improvement on the validation set	*Twt* value on the test set	Improvement on the test set
Feature set: 2 Classification method: knn-5 Learning set size: 600 Number of DRs used: 17	12.48	9.37%	14.82	5.30%
Feature set: 5 Classification method: C4.5 Learning set size: 600 Number of DRs used: 16	12.86	6.60%	15.20	2.88%
Feature set: 5 Classification method: C4.5 Learning set size: 120 Number of DRs used: 28	12.97	5.81%	14.86	5.05%
Feature set: 6 Classification method: knn-5 Learning set size: 600 Number of DRs used: 17	12.59	8.57%	14.70	6.07%
Feature set: 7 Classification method: C4.5 Learning set size: 240 Number of DRs used: 16	12.35	10.31%	15.39	1.66%
Feature set: 8 Classification method: ANN-5 Learning set size: 120 Number of DRs used: 40	12.63	8.28%	15.22	2.75%
Feature set: 8 Classification method: ANN-7 Learning set size: 600 Number of DRs used: 32	13.12	4.72%	15.01	4.09%
Manually selected DR	13.77	-	15.65	-

used, while in one occasion 28 DRs were used. Finally, the ANN classification method has shown a similar behaviour as the C4.5 classification method. When ANN-5 was used, the DR selection procedure achieved an improvement of 8.3% on the validation set, and 2.7% on the test set, when compared to the manually selected DR. On the other hand, if seven neurons are used in the hidden layer, then the DR selection procedure can achieve an improvement of 4.7% on the validation set, and 4.1% on the test set. In both cases over 30 DRs were used by the GAS method to create the learning set. Therefore it seems that the ANN classification method performs better when a larger number of DRs are used to construct the learning set, while the other two classification methods performed better when a smaller number of DRs was used to construct the learning set.

7.4 Dynamic DR selection procedure

This section will analyse the performance of the dynamic DR selection mechanism. Since this variant of the procedure also contains many parameters, some of which were not even present in the static procedure, this section will also analyse the influence of the different parameters on the performance of the procedure. In addition, the approach will also be tested on three different types of problems, on one whose features remain the same through the entire problem, and on two for which the features change over time.

7.4.1 Experimental design

Based on the obtained results for the static DR selection method, values of certain parameters will be filtered out to reduce the number of experiments which need to be performed. For the dynamic DR selection procedure, the following five classification methods will be used: knn-5, knn-7, naive Bayes, C4.5, and ANN-5. Although the static DR selection procedure achieved poor results when using the naive Bayes classification method, it was nevertheless included to test whether the method will also exhibit poor performance on other problem types. For the dynamic DR selection procedure the influence of the problem set size will not be analysed, since for the static variant it was already shown what sizes are preferable. Therefore, a set of 180 problem instances will be used for creating the learning set for the dynamic DR selection procedure. Finally, regarding the feature set, Table 7.4 shows the four feature set combinations which will be applied. The feature set containing all the features was selected since the best overall median value was obtained for it by the static DR selection procedure. The feature set consisting of only the due date related features was also included to test how it will perform when used by the dynamic DR selection procedure. In addition, the number of DRs supplied to the GAS procedure will also be limited to 10, 15, 20, 25, 30, 35, and 40 DRs. The experi-

Table 7.4: Feature set combinations used by the dynamic DR selection procedure

Feature set index	Features
1	DDR, DDT
2	DDR, DDT, LD
3	$DDR, DDT, \hat{p}, MJR, RTT$
4	$DDR, DDT, \hat{p}, MJR, RTT, LD$

ments will be performed in the same way as was the case for the static scheduling procedure. This means that the procedure will be executed for all value combinations of the different parameters. When analysing the performance for a concrete parameter, all experiments will be aggregated for each of the tested values for that parameter. The procedure was trained by using the learning set, while the performance of the procedure will be denoted on an independent validation set. In all experiments the Twt criterion was optimised. In addition, the box plot figures which represent the influence of different parameter values will not include outliers, since quite dispersed results were achieved by the procedure, since some parameter values lead to a quite bad performance of the procedure.

The definition of the problem instances will also change slightly when compared to the previously used problem instances. All properties of jobs will be generated in the same way as denoted in Section 4.3.2, except the due dates which are calculated as

$$d_j \in \left[r_j + p_{min} + \max\left(p_{avg} * \left(1 - T - \frac{R}{2}\right), 0\right), r_j + p_{min} + \max\left(p_{avg} * \left(1 - T + \frac{R}{2}\right), 0\right)\right],$$

where p_{min} represents the minimum processing time of job j, p_{avg} the average processing time of job j on all machines, R and T the due date range and due date tightness, respectively. The reason for using these definitions was that the problem characteristics can be approximated more precisely, since the due dates will be much less dispersed than by using the previous way of generating the due dates.

All three generated problem sets consist of 60 problem instances. Each of the problem instances in those sets will contain 1000 jobs and 10 machines. The number of jobs was increased to simulate larger systems which have a longer execution time. In the first problem set type, all jobs were generated with the same due date tightness and due date range parameters which were defined for the entire problem instance. Therefore, the characteristics of all jobs should be roughly the same during the execution of the system. In the second problem set type, the due date tightness and due date range characteristics of the problem instance will change over time. In this problem type, each instance will have three values defined for those two parameters.

The first third of the released jobs will have due dates generated by the first values of due date tightness and due date range parameters. The due date values of the second third of released jobs will be generated by using the second pair of values, while the due dates for the last third of the released jobs will be generated by the third pair of values. In the final problem type, the due dates were generated in the same way as described for the second problem set type. However, in addition to the changing due date characteristics of the system, the release time of jobs will also be distributed more unevenly. Namely, one third of jobs will have their release times randomly generated from the interval

$$r_j \in [0, 0.2 * \hat{p}],$$

the second third will have their release times generated from the interval

$$r_j \in [0.2 * \hat{p}, 0.3 * \hat{p}],$$

while the release times of the remaining jobs are generated from the interval

$$r_j \in [0.3 * \hat{p}, 0.5 * \hat{p}].$$

$\hat{p}$ is defined like in Section 4.3.2 as

$$\hat{p} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}}{m^2}.$$

Therefore, less jobs will be released at the beginning and the end of the system, while most of the jobs will be released during the middle of the system execution. This will also result in more tardy jobs, which will be seen from larger Twt values in later sections.

7.4.2 Results obtained for experiments with constant problem parameters

This section will present the results achieved by the dynamic DR selection method when it is applied on scheduling problems whose characteristics do not change during the execution of the system.

Parameter influence analysis

Figure 7.5 denotes the influence of the classification methods on the performance of the dynamic DR selection procedure. Out of the tested classification methods, the best performance was clearly achieved by the C4.5 classification method. The knn method also achieved quite good results, with slightly better results being achieved if the neighbourhood of size seven was used.

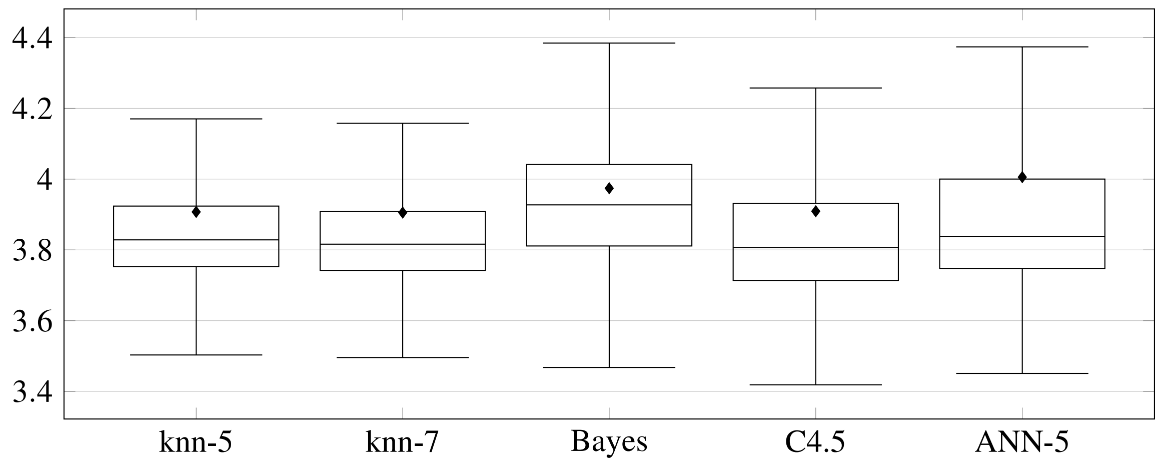


Figure 7.5: Performance of the DR selection procedure depending on the classification methods, for problems with constant characteristics

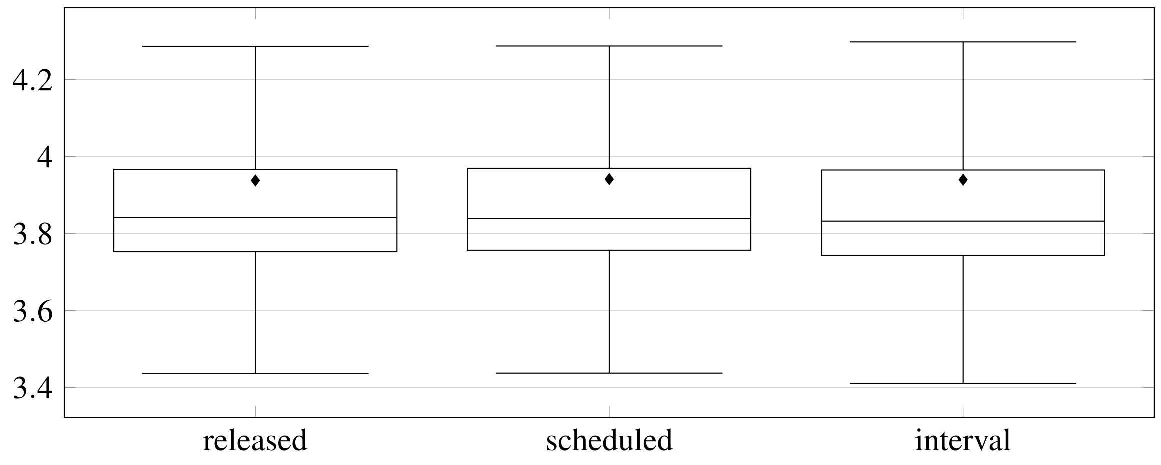


Figure 7.6: Performance of the DR selection procedure depending on the methods used for determining when the procedure should be used, for problems with constant characteristics

The ANN method achieved a similar median value as the knn method, although ANN usually obtained more dispersed results. The naive Bayes classification method achieved the overall worst results among all the tested methods.

Figure 7.6 represents the influence of the different methods which are used to determine when the DR selection procedure should be applied. The labels *released* and *scheduled* denote that the number of released or scheduled jobs is used to perform the decision, while the *interval* label is used to denote that the decision will be performed after a certain time interval has passed. The figure shows that all three methods perform equally well. Slightly better results for the median value are obtained when a fixed time interval is used. Therefore, from the results it is evident that the method for choosing when the DR selection procedure should be applied does not have any significant effect on the performance of the DR selection procedure.

Figure 7.7 represents the influence of the choice on how many times the DR selection procedure should be applied during the execution of the system. The results labelled as *no repeat*

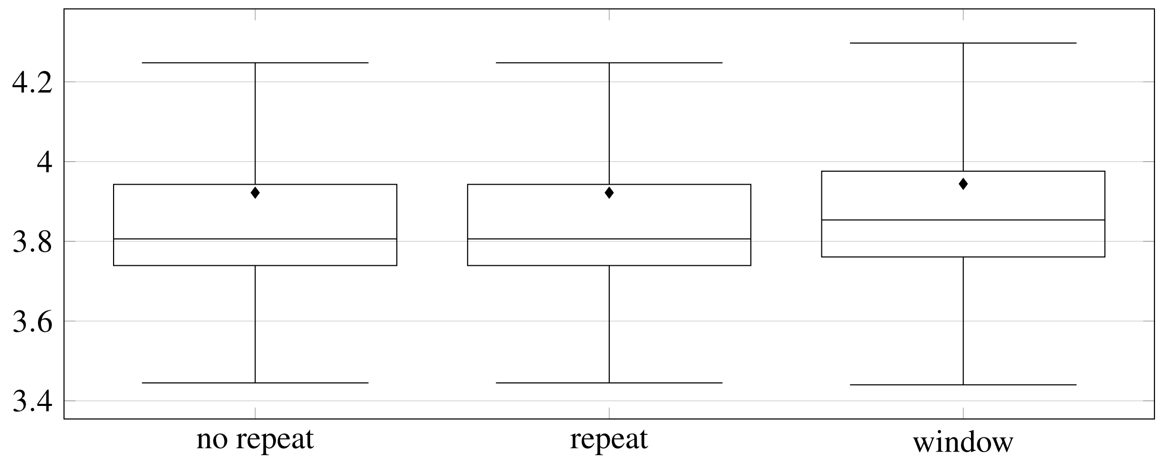


Figure 7.7: Performance of the DR selection procedure depending on the frequency of performing the selection, for problems with constant characteristics

denote that the DR selection procedure is executed only once, and that all released jobs are used to calculate the features of the problem instance. On the other hand, *repeat* denotes that the DR selection procedure is executed repeatedly, and that the features are calculated based on all released jobs until the decision moment. Finally, the *window* method also denotes that the selection procedure is executed repeatedly, however, the features of the problem are calculated only based on the jobs which were released from the last moment in time when the selection procedure was applied. The figure shows that there is no difference if the method is applied only once or several times during the execution. In addition, it is also evident that the window method achieved to a small extent worse results when compared to the other two methods. However, such results are expected since the characteristics of the problems do not change over time. Therefore, it is preferred to calculate the features based on all jobs released into the system.

The influence of the different feature sets on the performance of the DR selection procedure is outlined in Figure 7.8. As with the static DR selection procedure, better results are achieved when using the larger feature sets. The introduction of the *LD* feature into the sets did not lead to any significant improvements in the results. However, by using this additional feature the procedure has a greater chance of achieving better results. Therefore, using the two larger sets seems to be more beneficial for the DR selection procedure even when applied under dynamic conditions.

The influence of the number of DRs on the performance of the DR selection procedure is illustrated in Figure 7.9. This figure shows some quite interesting behaviour of the DR selection procedure. The less DRs are used, the better median values are achieved by the DR selection procedure. However, with the increase of the number of applied DRs, the procedure is also able to achieve better minimum values. Unfortunately, the dispersion of the results increases with the number of DRs which are used for generating the learning set. This can especially be seen

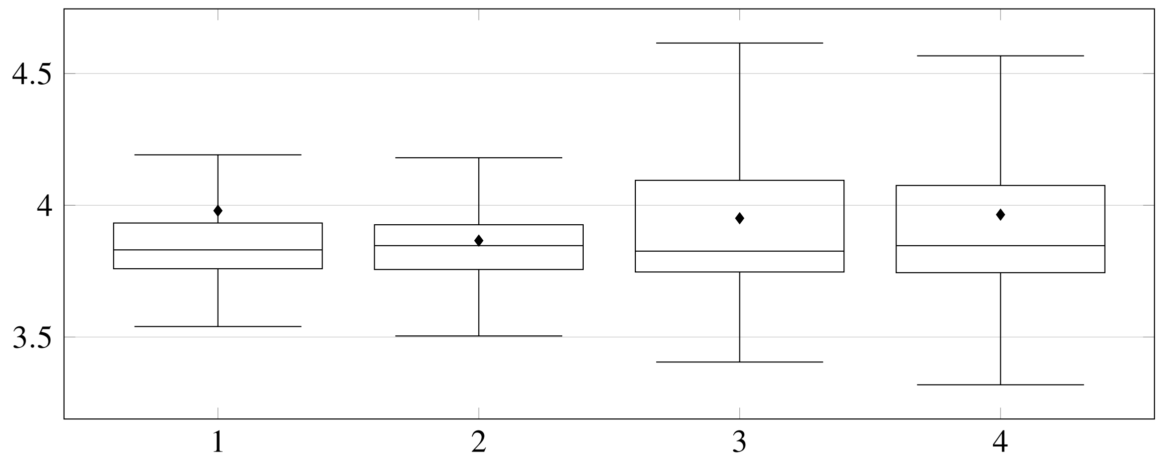


Figure 7.8: Performance of the DR selection procedure depending on the applied feature set, for problems with constant characteristics

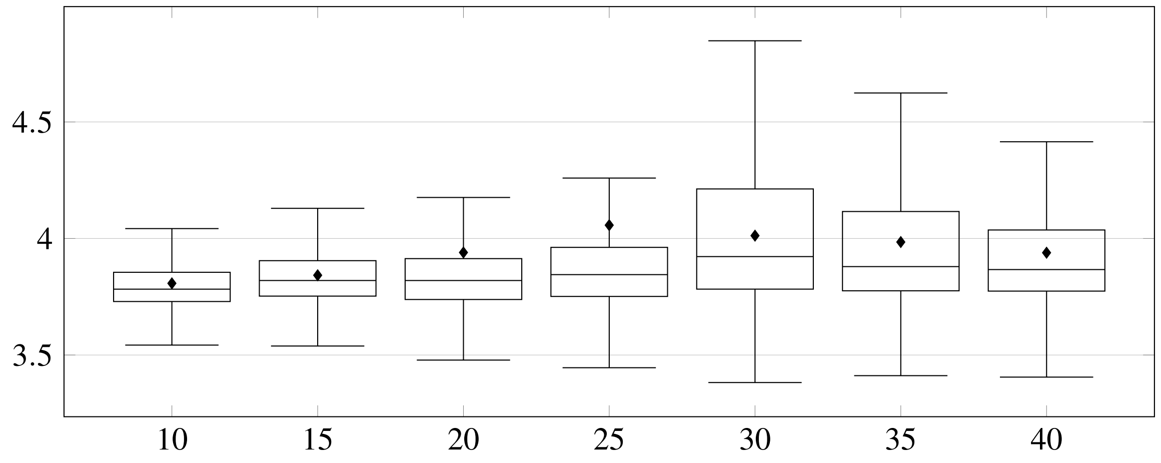


Figure 7.9: Performance of the DR selection procedure depending on the number of DRs used to construct the learning set, for problems with constant characteristics

when 30 or more DRs are used, since in those cases the procedure achieved the most dispersed results. Based on the previous observations, it is hard to determine which number of DRs should ideally be used. With a smaller number of DRs the procedure will be more stable, however, with a larger number of DRs better results can be achieved. Therefore the choice depends on which of the aforementioned properties is more important.

Figure 7.10 shows the influence of the number of jobs that need to be released or scheduled, before the decision process is performed. The figure shows that as the number of jobs which are used to perform the decision increases, the results of the approach deteriorate up to a certain point, after which the results mostly stagnate. This means that even a smaller number of jobs is enough to make a good approximation about the characteristics of the problem instance. It is also preferred to perform the DR selection process as soon as possible, since this increases the chance that if an inappropriate DR was selected as the initial DR, that it will be replaced as soon as possible. Therefore the initial DR will not have a large negative influence on the performance

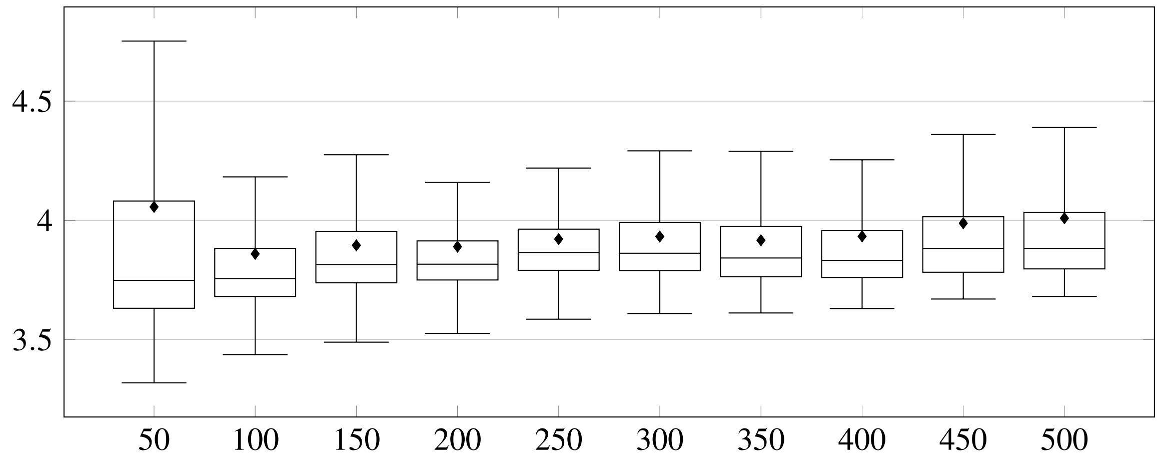


Figure 7.10: Performance of the DR selection procedure depending on the number of released or scheduled jobs used to calculate the features, for problems with constant characteristics

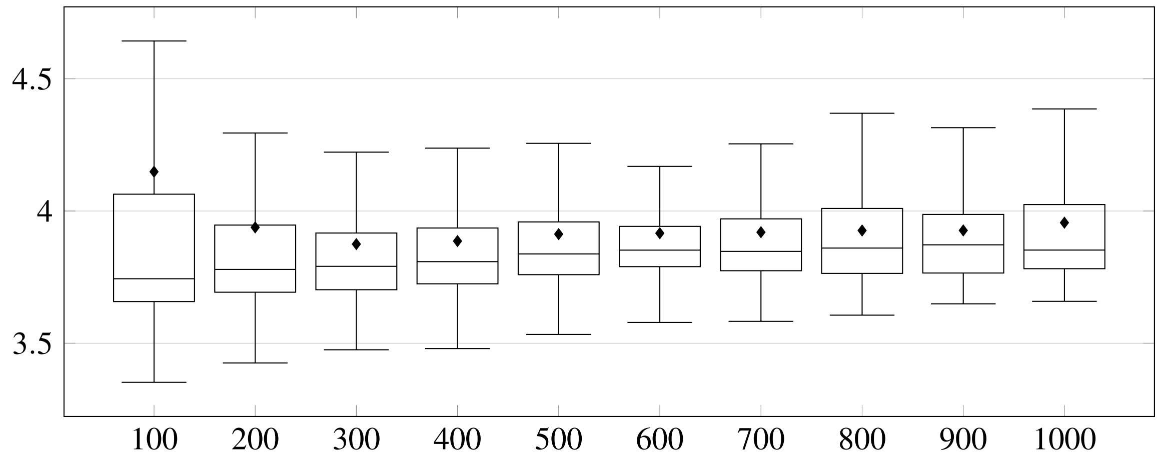


Figure 7.11: Performance of the DR selection procedure depending on the interval length between performing the selection, for problems with constant characteristics

of the created schedule.

Figure 7.11 represents the influence of the time interval length which is used before performing the DR selection procedure. For this parameter, a similar behaviour can be observed from the figure, as was the case when a fixed number of released or scheduled jobs was used. For smaller interval values the procedure does not only achieve better median values, but it can also achieve a better minimum value as well. Therefore, it is more beneficial to use smaller interval values for the DR selection procedure.

Performance comparison with a manually selected DR

This section will present the results of the DR selection procedure, for several selected parameter value combinations. Table 7.5 represents the results obtained for five selected parameter combinations, as well as for the DR which achieved the best overall results on the training set. The first thing which can be noticed is that the worst result is achieved when the ANN classification method is used. In this case the procedure achieved an improvement of only 1.2% on the validation set, however, on the test set it achieved an improvement of 8.1%.

For the other four results, the procedure performs quite well for all of the tested parameters. In three occasions, the procedure performs its decision after a small time period, or after a small number of jobs is released. For all these experiments the procedure achieved good results on both problem instance sets. On the other hand, for the experiment where the decision is performed after 150 jobs are released into the system, the procedure achieved to a certain extent worse results than in other experiments, where a smaller number of released jobs was used. Therefore, the DR selection procedure obtains a good approximation of the characteristics of the problem instance after only a small number of jobs is released. As a consequence, it is preferred to use the DR selection procedure as soon as possible, to select which of the available DRs is appropriate for solving the current problem instance. Therefore, by performing the selection of the appropriate DR as soon as possible, the influence of the initially applied DR, which might not be suitable for solving the current problem instance, will be reduced. When using a smaller feature set, the procedure can perform well by using only a smaller number of DRs, as can be seen from last two examples in the table. As the number of features in the feature set increases, the procedure achieves better results as more DRs are used by the procedure. This is demonstrated by the first two examples in the table. The best results for the validation and test set are achieved when the DR selection procedure uses the *no repeat* method, which is expected since the characteristics of the scheduling problem do not change over time. By using the repeat or window methods, the procedure achieves to a certain extent worse results, but it still easily achieves better results than by using only a single manually selected DR. For the validation set, the best result was achieved when using the knn-7 classification method in the DR selection procedure, while the best result on the test set was achieved when the C4.5 method was used by the procedure.

The DR selection procedure achieved an improvement of at most 14.11% on the validation set, and 15.9% on the test set, when compared to the manually selected DR. Therefore, the procedure does not achieve only good improvements on the validation set, on which its parameters were tuned, but also on an unseen problem set. More importantly, the results denoted in the table show that the procedure performs well on both problem instance sets for the same parameter combination. This demonstrates that with a good choice of parameters, the DR selection procedure can perform well on various problem instances.

7.4.3 Results obtained for experiments with changing due dates

In this section, the parameter influence on the DR selection procedure will be analysed when the procedure is applied for solving problems in which the due date characteristics of jobs change throughout the problem.

Table 7.5: Results of the dynamic DR selection procedure for several selected parameter values, when applied on problems with constant characteristics

DR selection procedure parameters	*Twt* value on the validation set	Improvement on the validation set	*Twt* value on the test set	Improvement on the test set
Feature set: 3 Classification method: knn-7 Number of DRs used: 30 Calculation method: released Calculation frequency: no repeat Number of released jobs: 50	3.444	14.11%	3.667	15.21%
Feature set: 4 Classification method: C4.5 Number of DRs used: 30 Calculation method: interval Calculation frequency: window Interval length: 100	3.452	13.92%	3.656	15.47%
Feature set: 4 Classification method: ANN-5 Number of DRs used: 20 Calculation method: interval Calculation frequency: no repeat Interval length: 100	3.993	0.42%	3.976	8.07%
Feature set: 2 Classification method: C4.5 Number of DRs used: 10 Calculation method: released Calculation frequency: repeat Number of released jobs: 150	3.526	12.07%	3.783	12.5%
Feature set: 2 Classification method: C4.5 Number of DRs used: 15 Calculation method: released Calculation frequency: no repeat Number of released jobs: 50	3.456	13.82%	3.639	15.86%
Manually selected DR	4.010	-	4.325	-

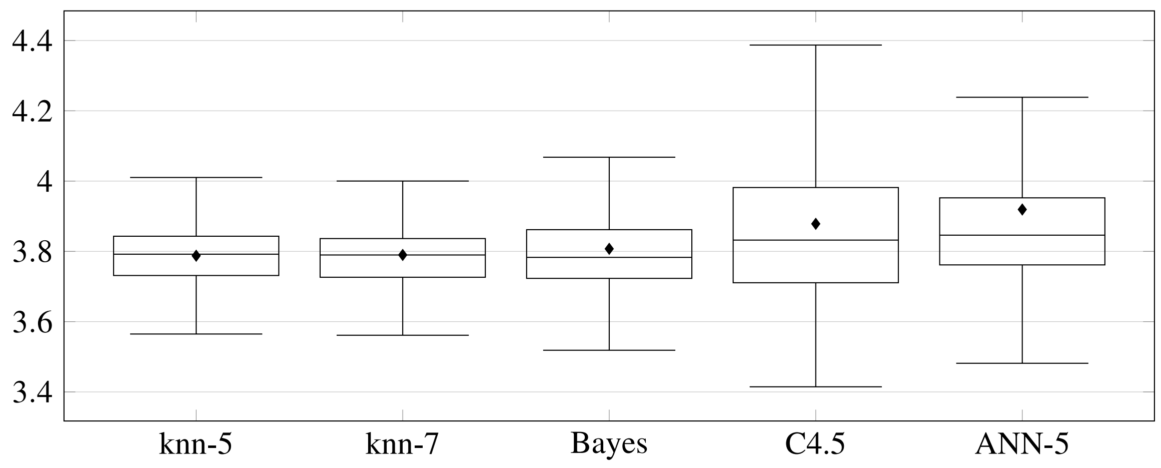


Figure 7.12: Performance of the DR selection procedure depending on the classification methods, for problems with changing due date characteristics

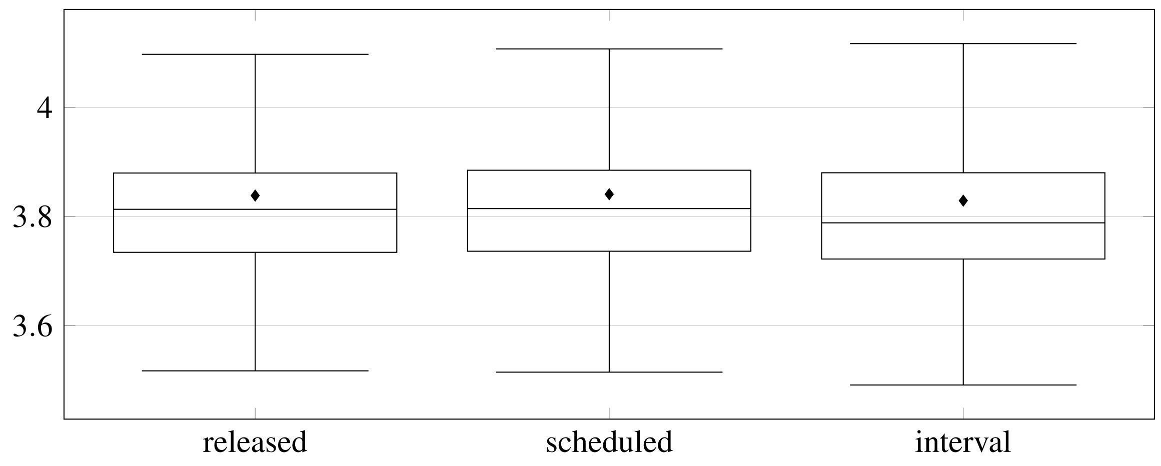


Figure 7.13: Performance of the DR selection procedure depending on the methods used for determining when the procedure should be used, for problems with changing due date characteristics

Parameter influence analysis

Figure 7.12 denotes the influence of the classification methods which are used by the DR selection procedure. It is immediately evident that this time the naive Bayes classifier leads to the best median value of the results. The knn method performs mostly equally well as the naive Bayes, however the other two methods achieved worse median values, and also more dispersed results. These results just prove that the performance of the classification method heavily depends on the problem type on which the methods are applied.

The influence on the methods for determining when the DR selection procedure should be applied is shown in Figure 7.13. For this parameter it can be seen that the choice of the method once again does not have a large influence on the performance of the DR selection procedure. However, once again the procedure achieves a slightly better median value when using a constant time interval before the selection of a DR is performed. Nevertheless, by using any of the three suggested methods the procedure will obtain similar results.

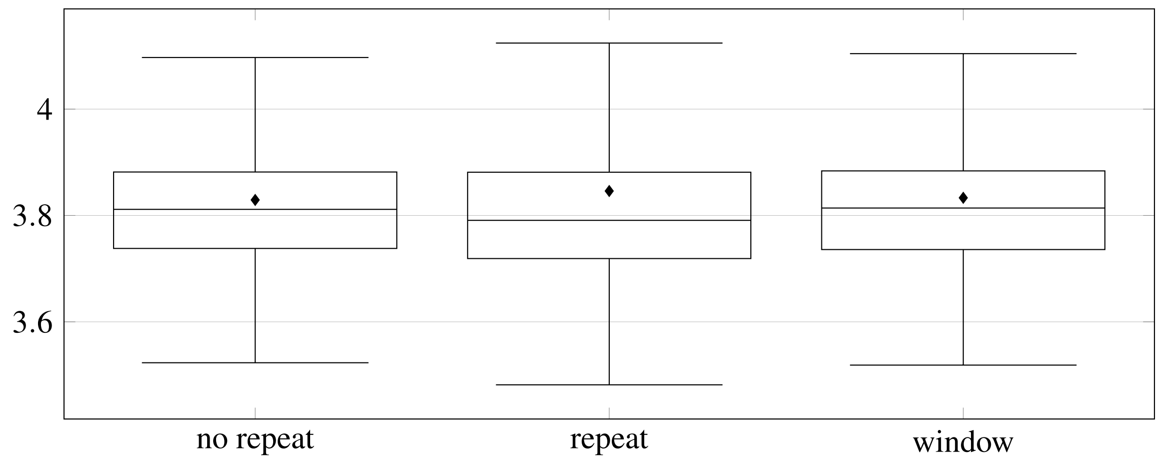


Figure 7.14: Performance of the DR selection procedure depending on the frequency of performing the selection, for problems with changing due date characteristics

Figure 7.14 denotes the influence of the frequency of performing the DR selection procedure. The figure shows that the best results are achieved if the selection procedure is performed repeatedly. This is expected since the characteristics of the problem change slightly during the execution of the system. However, it is surprising that the variant which uses all released jobs to calculate the features achieved better performance, than when using only the jobs which were released after the previous decision point. The cause for this can be that the characteristics of the problems do not change drastically enough, and therefore better approximations can be obtained by using all released jobs. In the end, any of the selected calculation frequencies will lead the procedure to obtain good results, and therefore the choice of this parameter does not seem to be overly important.

Figure 7.15 represents the influence of the feature set combinations on the performance of the DR selection procedure. The figure shows some interesting behaviour of the procedure depending on the choice of the feature set that is used. For this concrete problem, the best median value was achieved when using the feature set consisting only of the due date related features, denoted as feature set 1. The feature set 3, which previously performed better, is now unable to match the results achieved by feature set 1. Adding the *LD* feature into the sets did not show any benefit in this occasion. Therefore, the procedure seems to perform the best when using only the features which will capture the changing conditions of the problem instance. The inclusion of other features which are mostly constant during the execution of the system seems to only complicate the training of the classifiers, and does not lead to any improvement in the performance.

The influence of the number of applied DRs on the performance of the DR selection procedure is shown in Figure 7.16. The results obtained for this problem type are similar to those obtained for the problem with constant characteristics. The DR selection procedure achieved the best performance when using a smaller number of DRs. Once again the results of the pro-

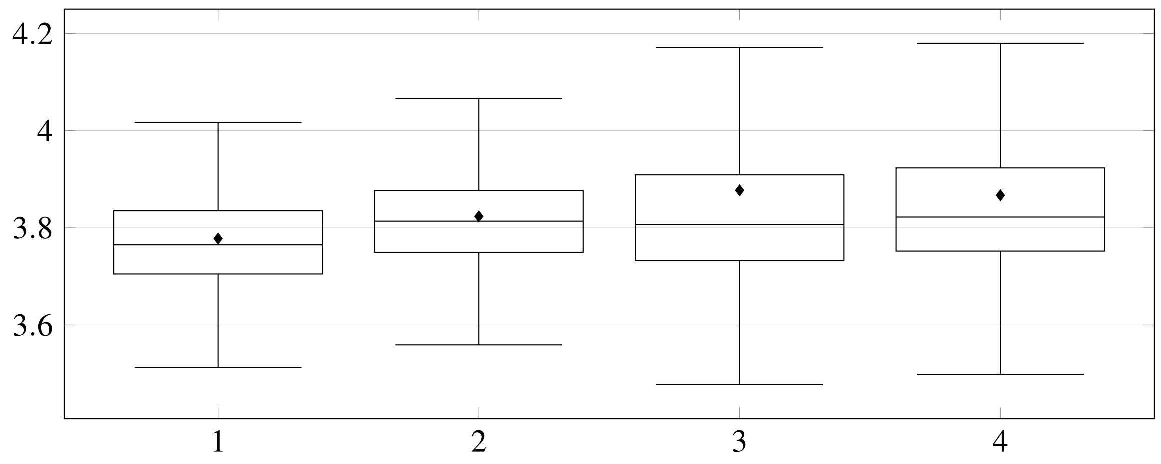


Figure 7.15: Performance of the DR selection procedure depending on the applied feature set, for problems with changing due date characteristics

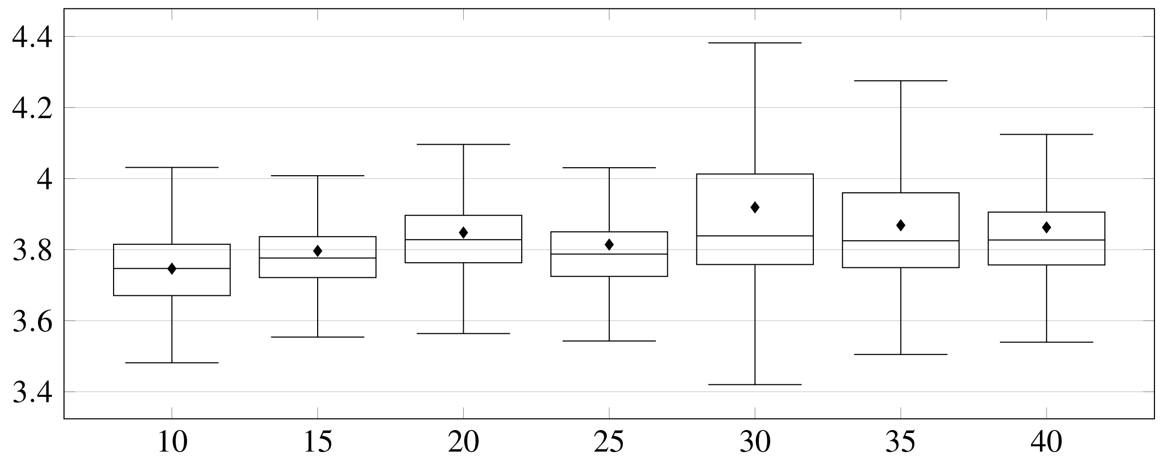


Figure 7.16: Performance of the DR selection procedure depending on the number of DRs used to construct the learning set, for problems with changing due date characteristics

cedure deteriorate heavily after more than 25 DRs are used. Therefore, for this problem type it is better to use a smaller number of DRs for the DR selection procedure. For this problem type the procedure achieved the best median value when only 10 DRs were used.

Figure 7.17 illustrates the influence of the number of released or scheduled jobs after which the DR selection procedure is applied. The figure shows that it is better to perform the selection procedure as soon as possible. This makes sense since it decreases the influence of the initial DR on the quality of the schedule, but also allows for changes of the DRs based on the changes of the problem characteristics. As the value of this parameter increases to include a significant portion of the jobs, the performance of the procedure starts deteriorating. This is expected since the procedure will have less opportunity to select good DRs. In addition, if the DR selection procedure is not performed frequently enough, then the procedure might not be able to pick up the changes which happen in the system, or it will not be able to react to them in timely manner.

Figure 7.18 represents the influence of the time interval before the DR selection procedure is

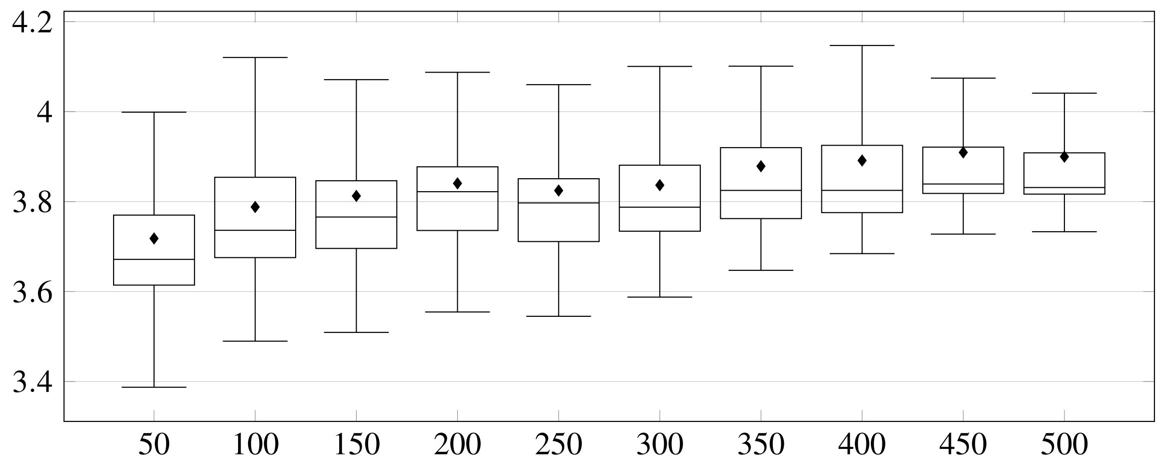


Figure 7.17: Performance of the DR selection procedure depending on the number of released or scheduled jobs used to calculate the features, for problems with changing due date characteristics

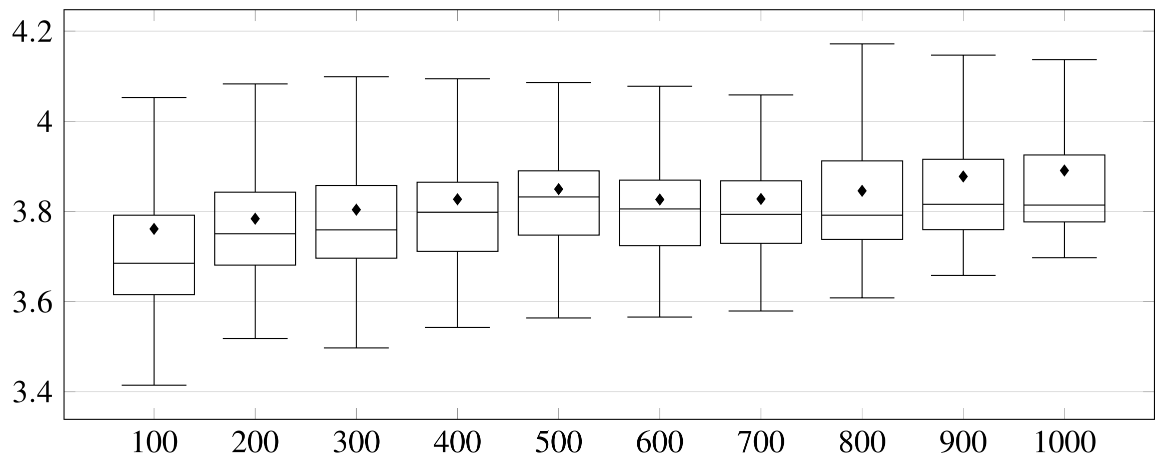


Figure 7.18: Performance of the DR selection procedure depending on the interval length between performing the selection, for problems with changing due date characteristics

applied. The results obtained for this parameter are similar to those obtained for the number of released or scheduled jobs. The best results are again achieved when a smaller time interval is used. Therefore, based on all previous observations it can be concluded that much better results can be achieved when the DR selection procedure is applied in earlier stages of the system execution, since this will allow for most flexibility of the procedure.

Performance comparison with a manually selected DR

In this section several best parameter combinations for the DR selection procedure will be selected, and by using them the procedure will be tested on an additional test set. Table 7.6 represents the results of the five selected parameter combinations on the validation and test set.

The worst results are achieved by the second experiment, where the smallest feature set, consisting only of the due date related features, was used. The other experiments in the table all achieved very good results on both problem instance sets. Even by using the naive Bayes classi-

Table 7.6: Results of the DR selection procedure for several selected parameter values, when applied on problems with changing due date characteristics

DR selection procedure parameters	*Twt* value on the validation set	Improvement on the validation set	*Twt* value on the test set	Improvement on the test set
Feature set: 2 Classification method: C45 Number of DRs used: 10 Calculation method: released Calculation frequency: repeat Number of released jobs: 50	3.425	14.65%	3.353	16.63%
Feature set: 1 Classification method: knn-5 Number of DRs used: 30 Calculation method: released Calculation frequency: no repeat Number of released jobs: 50	3.559	11.31%	3.673	8.68%
Feature set: 4 Classification method: Bayes Number of DRs used: 25 Calculation method: interval Calculation frequency: no repeat Interval length: 200	3.556	11.39%	3.440	14.47%
Feature set: 4 Classification method: knn-7 Number of DRs used: 25 Calculation method: interval Calculation frequency: no repeat Interval length: 100	3.523	12.21%	3.512	12.68%
Feature set: 4 Classification method: C4.5 Number of DRs used: 10 Calculation method: released Calculation frequency: repeat Number of released jobs: 50	3.525	12.16%	3.412	15.17%
Manually selected DR	4.013	-	4.022	-

fication method, which achieved quite bad results when applied for static scheduling problems, the DR selection procedure achieved quite good results for the problem with changing due date characteristics. From the table it is evident that the C4.5 classification method was in both cases applied with a small number of DRs in it, while the knn and naive Bayes classification methods used a larger number of DRs in the learning set. Therefore, C4.5 seems to generally perform better when a smaller number of DRs is supplied to it. For all experiments the procedure uses a small interval or number of released jobs between subsequent selections of DRs. This again proves that an earlier and more frequent application of the DR selection procedure leads to better results. Most of the presented experiments use the feature set which consists out of all features, which leads to the conclusion that the procedure is more likely to achieve good results if it has access to a larger number of problem characteristics.

The best result was achieved by the first experiment, for both the validation and test set. With these parameter values the DR selection procedure outperformed the manually selected DR by 14.7% on the validation set, and 16.6% on the test set. When achieving the best results, the DR selection procedure was used with the C4.5 classification method, and with only ten DRs for creating the learning set. Thus, the procedure does not require the use of many DRs to perform well. Based on the results it is evident that the DR selection procedure performs quite well for the selected parameter combinations on both the validation and the test set, even when applied on problems with changing characteristics.

7.4.4 Results obtained for experiments with changing due dates and release times

In this section, the performance of the DR selection procedure will be analysed on a scheduling problem in which the due date and release time characteristics change during the execution of the system.

Parameter influence analysis

The influence of the different classification methods is shown in Figure 7.19. The results show that in most cases the knn method achieved the best median value of the results. For this problem type the naive Bayes has also achieved quite good results, whereas C4.5 and ANN achieved worse results than the other methods. It is quite surprising that ANN, although having the most dispersed results, achieved a better median value of solutions than the C4.5 method. This once again proves that the performance of a concrete method heavily depends on the problem which is solved.

Figure 7.20 represents the influence of the methods for determining when the selection of DRs should be executed, on the performance of the DR selection procedure. Once again the best median values are achieved when using the interval decision method. On the other hand, there seems to be no difference when using either the number of released or scheduled jobs for determining when the selection procedure should be invoked. Thus, even for this type of problem this parameter does not have a large influence on the performance of the procedure. Therefore, the DR selection procedure should achieve good results by using any of the three methods.

Figure 7.21 represents the influence of the frequency of applying the DR selection procedure on its performance. The obtained results are a bit surprising, since it was expected that the window method would achieve much better results than the other methods, since it calculates the features using only jobs which were released after the last decision point. However, the window method was still able to achieve the overall best result out of the three methods. Therefore, when the window method is used, it is evident that the performance of the procedure depends much more on the choice of the other parameters. Also, it is interesting to note that the procedure performs extremely well even when performing the decision only once. It seems that even if the characteristics of the system change slightly over time, the changes do not seem to be drastic enough so that performing the DR selection procedure repeatedly would lead to significant improvements in the performance.

The influence of the different feature sets on the performance of the DR selection procedure is shown in Figure 7.22. The best median value was achieved when using feature set 1, which consist only of the two due date related features. As the feature set grows, the median value of the obtained results deteriorates and the DR selection procedure achieved more dispersed results. However, with the increase of the feature set size, the procedure is also able to achieve better minimum values. This means that the with the larger feature sets, the procedure becomes more expressive, but also much more sensitive to the selected values of all other parameters. Therefore, it will be more difficult to find parameter values for which the entire

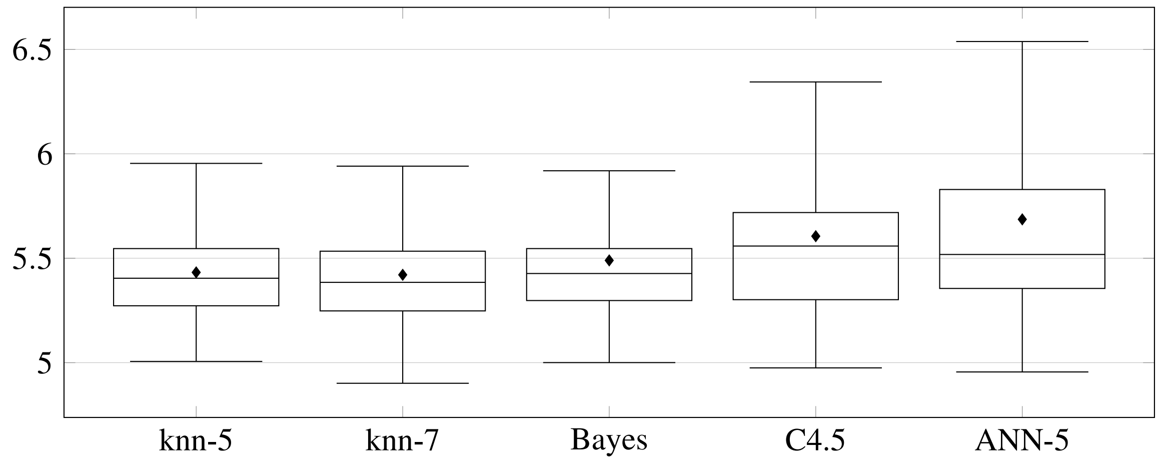


Figure 7.19: Performance of the DR selection procedure depending on the classification methods, for problems with changing due date and release time characteristics

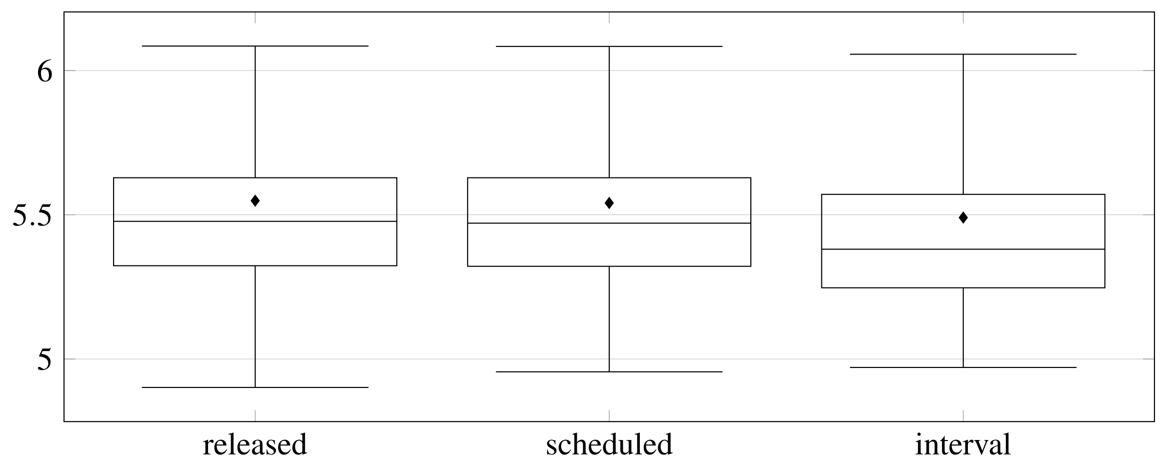


Figure 7.20: Performance of the DR selection procedure depending on the methods used for determining when the procedure should be used, for problems with changing due date and release time characteristics

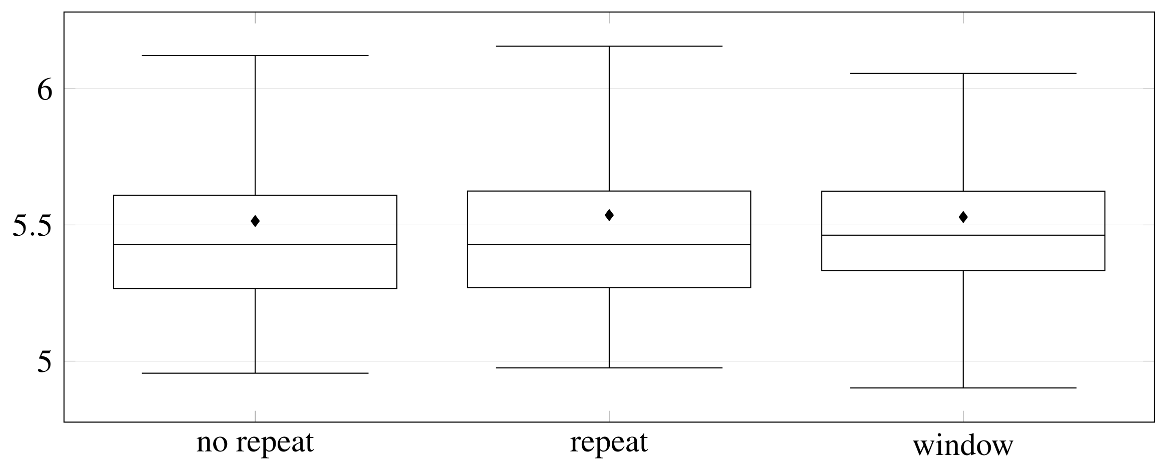


Figure 7.21: Performance of the DR selection procedure depending on the frequency of performing the selection, for problems with changing due date and release time characteristics

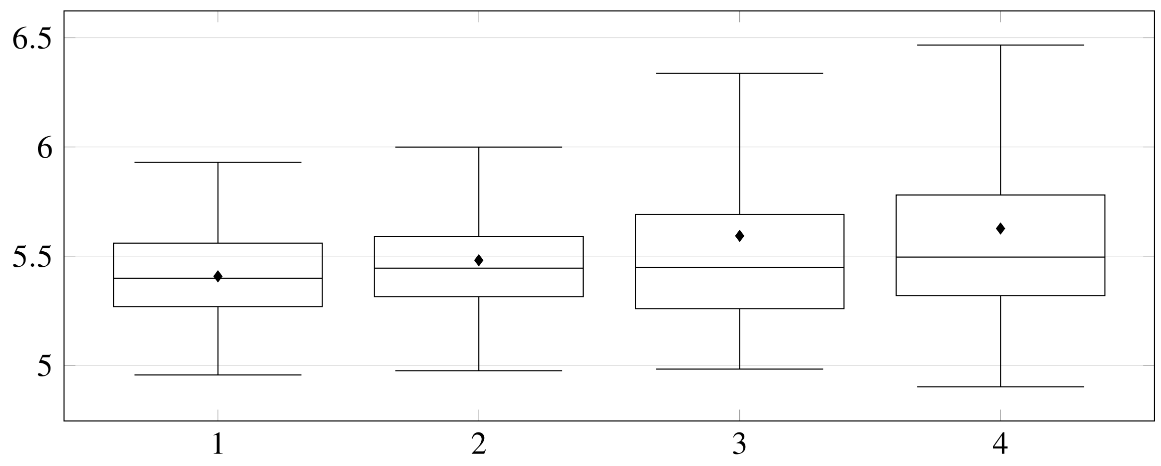


Figure 7.22: Performance of the DR selection procedure depending on the applied feature set, for problems with changing due date and release time characteristics

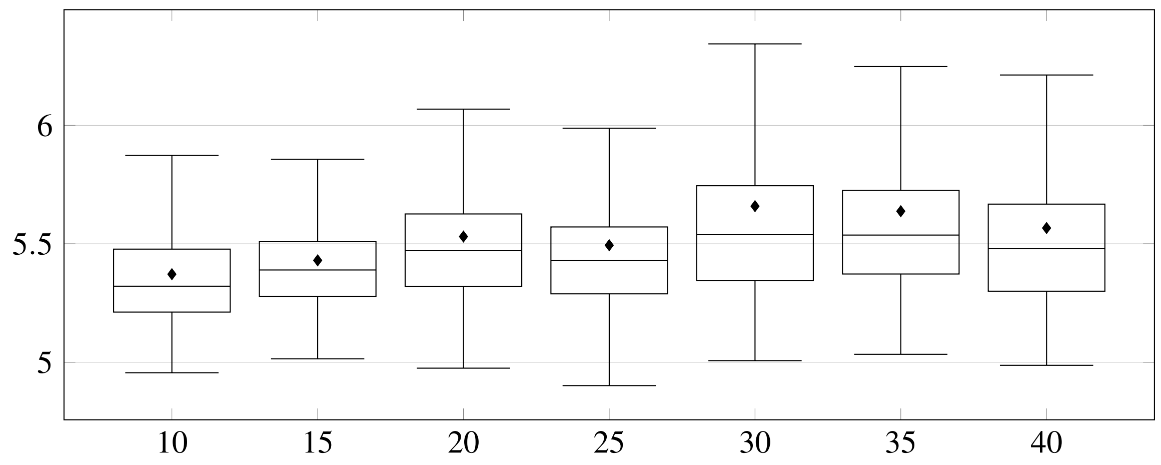


Figure 7.23: Performance of the DR selection procedure depending on the number of DRs used to construct the learning set, for problems with changing due date and release time characteristics

approach achieves good results.

Figure 7.23 represents the influence of the number of used DRs on the performance of the DR selection procedure. The results once again show that the procedure performs better if a smaller number of DRs is used. Even though the GAS procedure filters out some DRs from the initially supplied set of DRs, including too many DRs will nevertheless result in inferior performances of the DR selection procedure.

The influence of the number of released and scheduled jobs, which are used for calculating features, is denoted in Figure 7.24. The procedure achieved the best results when performing the decision after a smaller or medium sized number of jobs has been scheduled or released. Naturally, performing the decision sooner is beneficial since it will reduce the influence of the initial DR. However, using more released jobs will lead to more precise approximations of the problem characteristics. Thus, for the smallest parameter values (50 and 100) the procedure achieved better minimum results, while for parameter values of 150 and 200 jobs the procedure

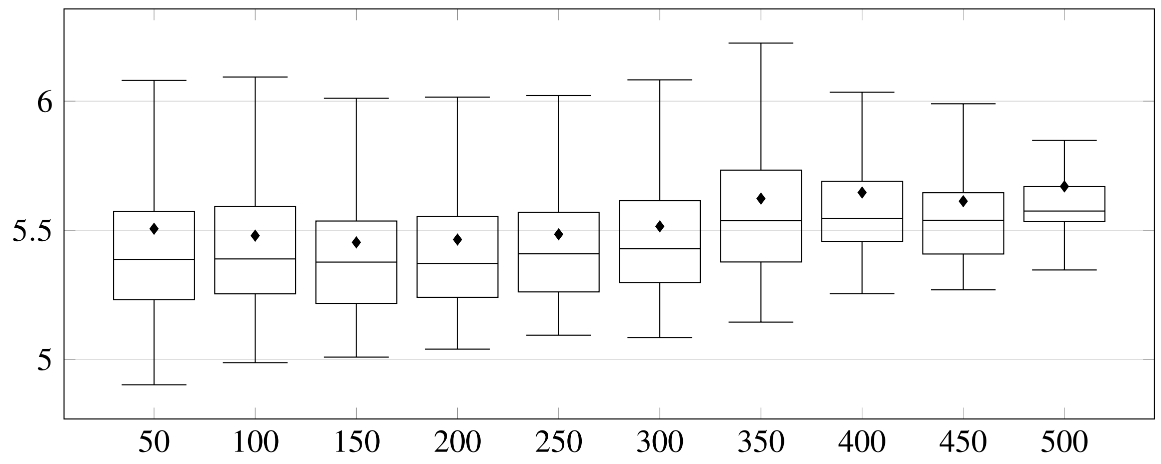


Figure 7.24: Performance of the DR selection procedure depending on the number of released or scheduled jobs used to calculate the features, for problems with and release time changing due date characteristics

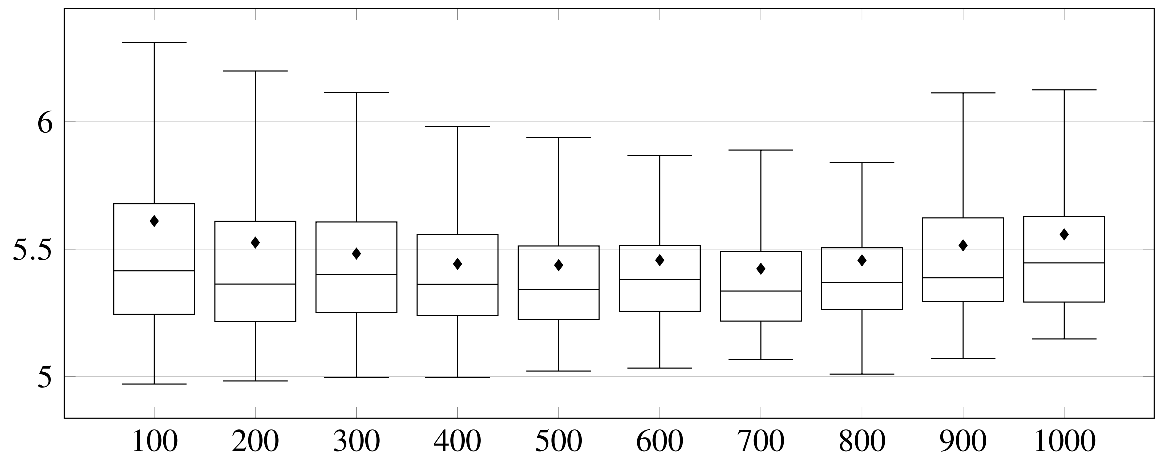


Figure 7.25: Performance of the DR selection procedure depending on the interval length between performing the selection, for problems with changing due date and release time characteristics

achieved better median values, but the value of the overall best solution deteriorates. The figure also shows that using a parameter value larger than 300 leads to serious deterioration in the results.

Figure 7.25 represents the influence of the interval length before applying the DR selection procedure. The figure shows a quite similar behaviour of the procedure with the one observed when using the number of released or scheduled jobs. For smaller intervals it is possible to obtain better minimum values. However, better median values are achieved when using intervals of medium lengths. The intuition behind such a behaviour is the same as was denoted for experiments when using the number of released or scheduled jobs.

Performance comparison with a manually selected DR

This section will analyse the performance of the DR selection procedure on five selected problem instances. Table 7.7 represents the results for the five selected parameter combinations of the DR selection procedure.

The DR selection procedure achieved the worst results when the ANN classification method was used. In this case, the procedure outperformed the manually selected DR by 5.3% on the validation set, and by 3.4% on the test set. Therefore, the ANN classification method is once again unable to achieve equally good performance as the other applied methods. Most of the experiments used around 20 DRs, regardless of the classification method, which seems to be an optimal choice for this problem type. It is interesting to observe that the best results were achieved by experiments which used the window and no repeat methods. The experiment which used the repeat method achieved inferior results when compared to both of those methods, without considering the experiment when the ANN classification method was used. Thus, it seems that the repeat method is inferior to the other two methods when the release times in the problem also change. In three experiments the procedure used the largest feature set, while in the remaining two the procedure used the feature set consisting of the due date related features and the *LD* feature. Therefore, it seems easier for the procedure to obtain good results when using more information about the problem instances. In addition, all feature sets contain the *LD* feature, which means that this feature seems to be useful for this problem type, probably due to the increased load which occurs in the middle of the system. Although for most of the experiments the selection procedure was applied after a short time interval, or after only a small number of jobs were released, the second experiment shows that the procedure achieves good results even if the selection procedure is applied after a larger number of jobs was released.

The largest improvement that the DR selection procedure can achieve over the manually selected DR are 12.9% for the validation set, and 13.3% for the test set. Unfortunately, the procedure was unable to achieve the best results on both problem sets for the same parameter values. On the validation set the best result was achieved by the knn classification method, while the best result on the test set was achieved by the C4.5 classification method. It seems as if the C4.5 method can better extract general knowledge from the learning set and thus perform better on unseen problem instances, whereas knn can adapt to the learning set more easily. However, with its better adaptation to the learning set the knn methods seems to be more susceptible to overfitting. As for the previous problem types, the DR selection procedure has again shown to achieve consistently good results on both the validation and test set, given that good parameter values are chosen.

Table 7.7: Results of the DR selection procedure for several selected parameter values, when applied on problems with changing due date and ready time characteristics

DR selection procedure parameters	*Twt* value on the validation set	Improvement on the validation set	*Twt* value on the test set	Improvement on the test set
Feature set: 4 Classifier: knn-7 Number of DRs used: 25 Calculation method: released Calculation frequency: window Number of released jobs: 50	4.901	12.92%	5.450	10.70%
Feature set: 4 Classifier: C4.5 Number of DRs used: 20 Calculation method: released Calculation frequency: no repeat Number of released jobs: 150	5.009	11.00%	5.395	11.60%
Feature set: 4 Classifier: C4.5 Number of DRs used: 15 Calculation method: interval Calculation frequency: repeat Interval length: 100	5.177	8.01%	5.545	9.14%
Feature set: 2 Classifier: C4.5 Number of DRs used: 20 Calculation method: interval Calculation frequency: no repeat Interval length: 100	4.975	11.60%	5.292	13.29%
Feature set: 2 Classifier: ANN-5 Number of DRs used: 20 Calculation method: interval Calculation frequency: no repeat Interval length: 200	5.328	5.33%	5.895	3.41%
Manually selected DR	5.628	-	6.103	-

Table 7.8: Performance of the DR selection procedure on several selected problem instances from the test set

Problem instance index	DR selection procedure		Manually selected DR *Twt* value	Value difference
	DR index	*Twt* value		
2	7	0.279	0.210	-0.069
3	0	0.121	0.034	-0.087
4	13	0.168	0.246	**0.078**
7	3	0.523	0.581	**0.058**
16	9	0.786	0.867	**0.081**
26	7	0.015	0.100	**0.085**
27	0	0.035	0.162	**0.127**
30	3	0.338	0.289	-0.049
33	6	0.623	0.827	**0.204**
44	4	1.239	1.687	**0.448**
Fitness on the entire test set		14.70	15.65	**0.956**

7.5 Analysis of the rule selection procedure

In this section a short analysis of the selection process on the obtained results will be performed. In the first part of the section the results obtained by the static DR selection procedure will be analysed, while the second part of the section will analyse the results of the dynamic DR selection procedure.

7.5.1 Analysis of the static selection procedure

Table 7.8 represents the performance comparison of the DR selection procedure and the manually selected DR on several problem instances from the test set. The table includes an additional column denoted as *value difference*, which represents the difference between the values achieved by the DR selection procedure and by the manually selected DR. Therefore, a positive value will denote that the DR selection procedure achieved better results, while negative values denote that the manually selected DR achieved a better performance. The values for the problem instances on which the DR selection procedure performed better will be denoted in bold. The parameters of the DR selection procedure were set to the values of the fourth experiment in Table 7.3.

The table represents ten selected problem instances, in three of which the DR selection procedure does not outperform the manually selected DR, and seven in which the DR selection procedure achieved a better performance. Based on the results obtained on the individual problem instances it is evident that the DR selection procedure achieved better improvements over the manually selected DR than vice versa. On the other four problem instances in the table, the DR selection procedure achieved improvements that are similar to those which the manually selected DR achieved over the DR selection procedure for problem instances 2, 3, and 30. Therefore, the procedure can achieve worse results than the manually selected DR in some cases, but it will usually achieve much better improvements on other problem instances. In total, the DR selection procedure achieved an improvement of 6.1% over the manually selected DR on all problem instances.

When the complete problem instance set is observed, the DR selection procedure outperforms the manually selected DR for 19 problem instances, while for 10 of them it performs worse. On the remaining problem instances both performed equally well. It might seem surprising that for ten problem instances the DR selection procedure was unable to outperform the manually selected DR, however the DR selection procedure still selected a DR which performs well on the given problem instances, and therefore the difference between the value achieved by the DR selection procedure and the manually selected DR is small, as denoted previously in the table. The real strength of the DR selection procedure is evident on problem instances on which the manually selected DR does not perform well. On these problem instances the DR selection method can, by selecting a DR suited for solving that concrete problem instance, significantly outperform the manually selected DR, and thus achieve a better performance on the entire problem instance set. Because of that reason it is not mandatory that the DR selection procedure selects the absolutely best available DR for each problem instance, which is in itself hardly possible. However, the DR selection procedure is required to provide a DR which performs well for the given problem instance. As a consequence the DR selection procedure will be able to perform well over many problem instances, while a single DR will perform well only on those problem instances for which it specialised.

Aside from observing the performance of the DR selection procedure on the different problem instances, it is also interesting to additionally examine how often each DR is used by the DR selection procedure. Figure 7.26 represents a histogram of the number of times that each DR was applied on the problem instances from the test set. The histogram shows that certain DRs have a much higher frequency of being applied, which is due to the fact that these DRs performed well on the learning set for several problem instances. In most cases, the DR selection procedure used the DRs 0, 3, 4, and 7, which were all selected for eight or more problem instances. The remaining DRs are mostly selected just for a few problem instances, while DRs 2, 5, 8, and 12, were not selected for even one problem instance.

The frequency of using DRs on the learning set and the performance of the individual DRs

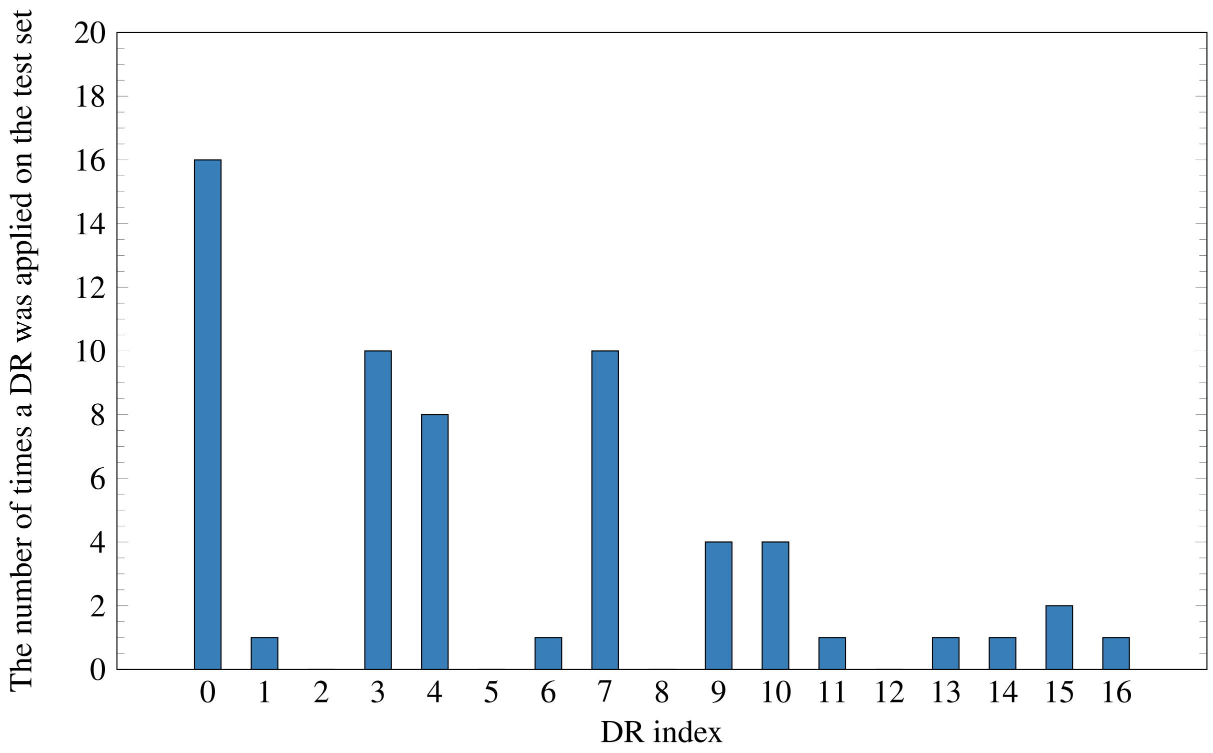


Figure 7.26: Histogram of the frequency of applying the DRs on the test set

on the learning set is also analysed to get a better impression of why DRs denoted in the histogram were most often used on the learning set. Table 7.9 represents the frequencies and *Twt* values of DRs when applied on the learning set and the test set. The occurrence frequencies on the learning set and test set show that DRs which were more often applied on the learning set, will mostly have a higher application rate on the test set as well. Therefore, rules 0, 3, and 7, will be used for most of the problem instances on both of the problem sets. However, for some rules there is a certain difference in the number of times that they are applied on the two problem instances. Although rules 4 and 13 are used for solving almost the same number of problem instances in the learning set, rule 13 is used only in one occasion on the test set, while rule 4 was applied on eight problem instances. Furthermore, rules 9 and 10 are used for solving several more problem instances than rule 4 on the learning set, however, each of them is selected for solving only four problem instances on the test set. Therefore, even though the DRs which were more often used on the learning set will have a higher probability of being selected on new problem sets, certain discrepancies are possible due to a different composition of the problem instance set.

Apart from the similarity between the frequencies on the two problem instance sets, it is also important to analyse how the fitness of the DRs influences the frequency of their appearance in the learning set. The results show that the two best rules, 10 and 15, do not have a big frequency of being applied by the DR selection procedure. The largest frequency is achieved by the rule 0, which achieved a moderate value for the *Twt* criterion, and is applied for solving one fourth of problem instances. Rule 7, which has the third largest frequency, achieved even worse results. This rule is used for about 10% of problem instances. This shows that although the best DRs are included in the DR selection procedure, they will not necessarily be the dominantly applied rules. When observing the test set, the rules which achieved the best results for the validation set do not necessarily perform well on this set as well. Out of the four most frequently applied DRs on the validation set, namely rules 0, 7, 3, and 4, neither achieved especially good results on the test set. However, this does not seem to influence the performance of the DR selection procedure, since it only applies those DRs on problem instances for which they are best suited, therefore achieving overall good results. Thus, based on all the previous observations, it is evident that the overall fitness of the DRs does not have a significant influence on the number of times it will be applied by the DR selection procedure. This conclusion is also backed up by the Spearman's Rho test, for which the values $\rho = -0.188$ and $p = 0.470$ were obtained. These values prove that no significant association between the two variables exists.

Since certain DRs are rarely used by the DR selection procedure, one could argue that this information could be used to select a smaller set of DRs, and use it when training the procedure on the learning set. In order to test this hypothesis, seven DRs, which were applied on most of the problem instances from the learning set, were selected and used for training the DR selection procedure. Unfortunately, this process did not lead to any improvement in the results, but rather resulted in a serious deterioration of the results which caused the DR selection procedure to perform similarly as the manually selected DR. This demonstrates that it is important not to restrict the number of DRs which are used by the DR selection procedure, since the procedure will not have access to DRs which could perform well on certain problem instances.

In order to gain an insight on how the DRs are associated with different problem instances, Figure 7.27 represents the distribution of DRs for different samples in the learning set, with regards to the due date range and due date tightness features. The figure demonstrates that a large amount of the learning set space is covered by DRs 0, 3, and 7, especially problem instances which have a due date tightness value which is smaller than 0.65. Therefore, it does not come as a surprise that these DRs are selected for solving the majority of problem instances. The figure shows that large portions of the problem instance space are usually solved well by only a single DR, if the due date tightness of the problem instances is less than 0.65. For example, problem instances which have a due date tightness value smaller than 0.65, and have a due date range value larger than 0.6, are mostly associated with DR 0. On the other hand, the problem instances which have a due date range smaller than 0.6 are most commonly associated with rules 3 and 7. Therefore, most of the problem instances in that part of the problem instance space are associated with only a few DRs. The reason why this happens, is because most of the problem instances with a smaller due date tightness are usually more easily solved by DRs. Thus, a single DR will usually perform well on many of such problem instances. However,

Table 7.9: Performance of the individual DRs used by the DR selection procedure

DR index	Learning set		Test set	
	Frequency	*Twt*	Frequency	*Twt*
0	157	109.6	16	16.20
1	14	109.6	1	15.49
2	8	109.0	0	16.47
3	94	107.3	9	15.90
4	35	108.8	8	15.77
5	18	112.2	0	15.72
6	14	112.5	1	16.01
7	66	112.0	10	15.92
8	8	109.6	0	15.56
9	40	112.6	4	15.95
10	40	106.0	4	15.67
11	19	113.2	1	16.39
12	5	110.7	0	15.69
13	36	112.2	1	15.38
14	10	113.3	1	15.90
15	25	106.5	2	15.65
16	11	109.6	1	15.31

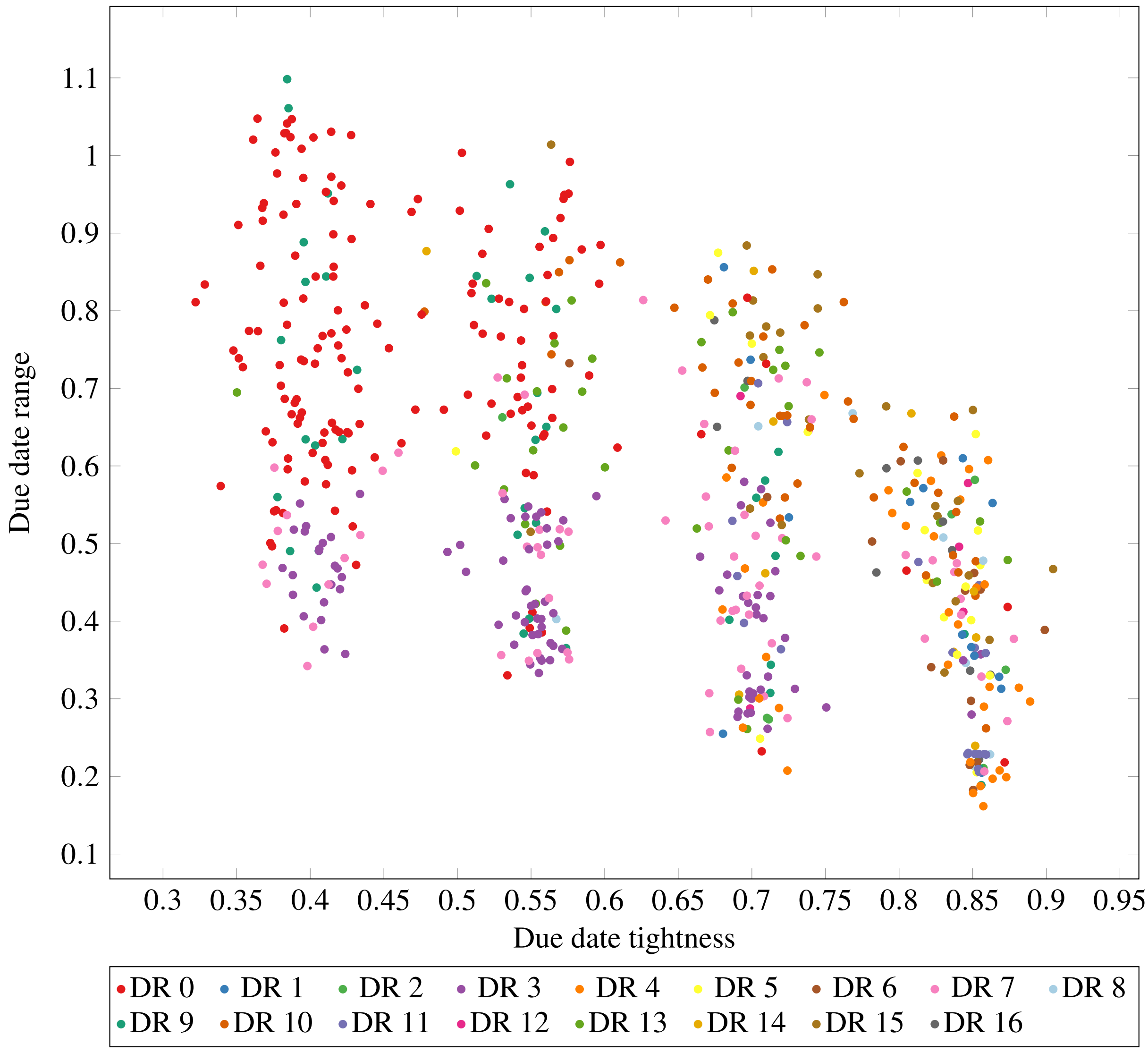


Figure 7.27: Association of DRs to samples in the learning sample space

as the due date tightness value increases, it is harder only for a single DR to perform well on many problem instances. As a consequence, in this part of the problem instance space DRs will usually be associated with a smaller number of problem instances. In addition, problem instances with similar due date characteristics are also quite commonly associated with different DRs, which just further proves that a DR which performs well for all these problem instances does not exist. This happens because of the higher due date tightness which causes more jobs to be tardy, and therefore it will be much harder for DRs to find the best solution. Therefore, a single DR will usually not be able to perform well on too many of such problems, which is the reason why more DRs are associated with these problem instances.

7.5.2 Analysis of the dynamic selection procedure

In this section the the dynamic DR selection procedure will be analysed. For that purpose the problem type with the changing due date and release time characteristics was selected. The parameters for the DR selection procedure will be set to those of the first experiment denoted in Table 7.7, but all three frequencies of performing the DR selection procedure will be tested to illustrate the differences in behaviour of the procedure.

Table 7.10 represents the behaviour of the DR selection procedure during the execution of several problem instances. At each decision point, when enough jobs were released, the table shows the index of the DR that is selected for further scheduling, as well as the *Twt* value achieved from the start of the system until that decision point. The starting rule in all procedures is DR 13. In addition, since the no repeat method performs the decision only once after 50 jobs are released, all subsequent cells will be marked with "-", since the method does not perform any more selections of DRs. The results of the manually selected DR are also included in the table to additionally illustrate its behaviour. At the end of the table, the results for all methods on the entire test set are also included.

Problem instance 6 shows that all three methods perform better than the manually selected DR. Even at the start of the problem the selected DR performs poorly, while the DR selection procedure changes the rule which is used, and is therefore able to keep the value of *Twt* at zero until 150 jobs are released. It is interesting to note that the no repeat method achieved better results in the beginning of the schedule. However, as more jobs were released into the system, the other two methods slowly decreased the difference between them and the no repeat method. This can best be seen the between 400 and 450 released jobs, where the window method achieved the smallest increase in the *Twt* value by switching to rule 5. A similar thing can also be observed between 650 and 700 released jobs. In the end with these changes the window method is outperforms the other two methods, while the repeat method slightly outperformed the no repeat method. On the other hand, on problem instance 8 only the window method achieved a better performance than the manually selected DR. The window method again shows that by performing good rule changes it outperforms all the other results. The no repeat method, on the other hand, achieved a quite bad result. The reason for this is that it selects a good DR for the start of the schedule, which unfortunately performs poorly as soon as the load of the system increases. Therefore, after around 400 jobs are released, the selected rule starts to perform bad decisions, and the *Twt* value of the system increases. The repeat method also achieved worse results than the manually selected DR, but not as much as the repeat method.

Table 7.10: Dynamic DR selection procedure behaviour analysis

Problem instance	Method		Number of released jobs																			
			50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
7	No repeat	DR index	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Twt	0	0	0	0.002	0.002	0.004	0.013	0.024	0.051	0.077	0.096	0.126	0.151	0.172	0.176	0.176	0.176	0.176	0.176	0.176
	Repeat	DR index	3	8	2	2	2	18	11	6	9	9	8	8	8	8	8	8	8	8	8	8
		Twt	0	0	0	0.005	0.005	0.008	0.019	0.030	0.050	0.088	0.103	0.127	0.149	0.171	0.174	0.174	0.174	0.174	0.174	0.174
	Window	DR index	3	15	13	1	3	8	3	5	3	3	2	3	3	2	3	3	3	0	3	8
		Twt	0	0	0	0.005	0.005	0.006	0.016	0.028	0.041	0.073	0.095	0.121	0.141	0.155	0.159	0.159	0.159	0.159	0.159	0.159
	Selected DR		0.000	0.008	0.008	0.013	0.013	0.019	0.035	0.046	0.079	0.122	0.137	0.179	0.203	0.219	0.224	0.224	0.224	0.224	0.224	0.224
8	No repeat	DR index	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Twt	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.010	0.013	0.022	0.031	0.032	0.039	0.041	0.041	0.041	0.041	0.041	0.041
	Repeat	DR index	1	8	8	3	2	11	11	6	9	9	8	8	8	8	8	8	8	8	8	8
		Twt	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.013	0.016	0.016	0.021	0.028	0.028	0.029	0.029	0.029	0.029	0.029	0.029
	Window	DR index	1	3	3	3	3	3	0	3	3	3	3	3	3	3	0	3	1	3	0	0
		Twt	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.005	0.007	0.012	0.012	0.013	0.013	0.013	0.013	0.013	0.013
	Selected DR		0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.008	0.009	0.009	0.011	0.019	0.020	0.020	0.020	0.020	0.020	0.020	0.020
30	No repeat	DR index	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Twt	0.001	0.007	0.016	0.017	0.022	0.022	0.022	0.022	0.022	0.022	0.023	0.025	0.029	0.032	0.032	0.032	0.032	0.033	0.033	0.033
	Repeat	DR index	1	8	8	2	2	2	8	8	6	6	9	9	8	8	8	8	8	8	8	8
		Twt	0.001	0.007	0.015	0.019	0.019	0.020	0.020	0.020	0.022	0.026	0.028	0.031	0.034	0.035	0.035	0.035	0.035	0.036	0.036	0.037
	Window	DR index	1	1	1	3	1	3	1	3	3	3	3	3	3	3	15	3	1	4	3	3
		Twt	0.001	0.007	0.016	0.017	0.017	0.017	0.017	0.017	0.019	0.019	0.019	0.020	0.024	0.025	0.025	0.025	0.025	0.026	0.027	0.028
	Selected DR		0.001	0.015	0.029	0.033	0.036	0.036	0.036	0.036	0.039	0.041	0.043	0.045	0.047	0.049	0.049	0.049	0.050	0.051	0.051	0.052

Table 7.10: Dynamic DR selection procedure behaviour analysis

Problem instance	Method		Number of released jobs																			
			50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
37	No repeat	DR index	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Twt	0.002	0.004	0.006	0.009	0.012	0.015	0.018	0.038	0.055	0.081	0.094	0.109	0.118	0.139	0.140	0.140	0.140	0.140	0.140	0.140
	Repeat	DR index	3	8	2	2	2	2	11	6	9	9	8	8	8	8	8	8	8	8	8	8
		Twt	0.002	0.004	0.006	0.009	0.018	0.023	0.027	0.056	0.084	0.109	0.123	0.146	0.164	0.188	0.189	0.189	0.189	0.189	0.189	0.189
	Window	DR index	3	1	3	3	3	3	3	3	5	3	5	3	3	3	0	3	0	3	3	3
		Twt	0.002	0.004	0.006	0.009	0.012	0.015	0.017	0.038	0.055	0.083	0.114	0.131	0.143	0.164	0.165	0.165	0.165	0.165	0.165	0.165
	Selected DR		0.002	0.005	0.006	0.010	0.012	0.015	0.018	0.042	0.061	0.100	0.119	0.142	0.186	0.212	0.212	0.212	0.212	0.212	0.212	0.212
50	No repeat	DR index	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Twt	0.000	0.000	0.000	0.000	0.000	0.003	0.003	0.003	0.005	0.017	0.031	0.032	0.034	0.034	0.038	0.040	0.047	0.052	0.063	0.064
	Repeat	DR index	3	8	8	2	2	11	11	9	9	8	8	8	8	8	8	8	8	8	8	8
		Twt	0.000	0.000	0.000	0.000	0.000	0.005	0.005	0.005	0.007	0.013	0.028	0.030	0.031	0.031	0.034	0.037	0.044	0.049	0.062	0.064
	Window	DR index	3	3	3	1	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3
		Twt	0.000	0.000	0.000	0.000	0.000	0.003	0.003	0.003	0.005	0.017	0.030	0.032	0.034	0.034	0.038	0.040	0.046	0.051	0.062	0.063
	Selected DR		0.000	0.000	0.001	0.001	0.005	0.009	0.009	0.009	0.016	0.030	0.043	0.045	0.049	0.055	0.058	0.060	0.067	0.072	0.074	0.076
Test set	No repeat		0.140	0.259	0.421	0.548	0.648	0.801	0.937	1.362	1.933	2.501	3.065	3.661	4.194	4.656	4.801	4.933	5.050	5.184	5.336	5.488
	Repeat		0.140	0.255	0.409	0.571	0.685	0.837	0.962	1.328	2.037	2.624	3.176	3.823	4.364	4.787	4.949	5.077	5.199	5.326	5.478	5.638
	Window		0.140	0.255	0.410	0.557	0.679	0.835	0.964	1.280	1.870	2.442	3.048	3.636	4.175	4.630	4.795	4.919	5.038	5.163	5.305	5.450
	Selected DR		0.140	0.310	0.486	0.660	0.776	0.944	1.109	1.566	2.342	2.924	3.513	4.095	4.749	5.187	5.336	5.474	5.616	5.772	5.914	6.103

For problem instance 30, all three methods achieved better results than the manually selected DR. The table shows that all three methods achieve quite similar results for this problem instance. Because it made a good decision at the beginning of the schedule, the no repeat method outperforms the repeat method. The window method again achieved the best results, even though at the end of the schedule it performed some bad decisions which resulted in an increase of the *Twt* value. However, at the beginning and the middle of the schedule, the window method performed some good decisions, which allowed it to perform better than the no repeat method. The manually selected DR performs bad decision even in the beginning of the schedule, because of which the *Twt* of the schedule increases significantly even from the start. For problem instance 37 the no repeat method achieved the best results among all three methods. The reason for this is that the rule which is first selected performs well for the given problem instance. The window method also uses this rule for scheduling most of the jobs, however in certain cases it switched to rule 5, which in the end did not lead to good improvements in the results, and even later on had a negative effect when rule 3 was again selected for scheduling. This is best evident in the period between the release of the 600th and 700th job, where although the no repeat and window methods used the same DR, the window method obtained a slightly higher *Twt* value, as a result of the decisions which were previously performed when rule 5 was used by the procedure. Finally, for problem instance 50, all three methods show to outperform the manually selected DR, however in this case all three methods achieved similar results. It is interesting to note how the window method uses DR 3 for the most of the schedule, and only in two occasions it switches to other DRs. Although the first switch was unnecessary, since it did not lead to any improvement in the *Twt* value, the second one resulted in a slightly better *Twt* value.

By comparing all methods on the entire problem set, the manually selected DR starts to perform worse than the DR selection procedure immediately from the beginning of the schedule. On the other hand, the DR selection procedure performs much better since it immediately switches the DR which is used after 50 jobs are released into the system. It is also interesting to observe that the no repeat method outperforms the window method in several occasions. However, by selecting different DRs the window method is, in the end, still able to achieve the best performance. On the other hand, the repeat method achieved the worst result among the three tested methods, which shows that calculating the features from all released jobs might not be beneficial in some cases. The repeat method seems to have most problems during the middle of the system, where the load becomes the highest. Nevertheless, all three methods achieved much better results than the manually selected DR, and thus prove their advantage over selecting and using only a single DR.

7.6 Discussion

The last few sections have shown that the proposed DR selection method achieved better performance than by using only a single manually selected DR for creating the entire schedule. The procedure is applicable in both static selection of DRs, where the DR is selected before the start of system execution, and in dynamic selection of DRs, where the DR is selected during the execution of the system. However, the procedure introduces a large amount of new parameters which have a serious impact on the performance of the procedure, and the values of which need to be optimised in order for the procedure to perform well.

One of the most important parameters of the procedure is the choice of the classification method which will be used to select the DR which should be applied for selecting the appropriate DR. Out of the six applied classification methods, the knn, C4.5 and ANN methods have achieved the best performance for the static DR selection procedure. However, when considering the dynamic DR selection procedure, the knn method obtained a better median value when all experiments are considered, although the C4.5 method also achieved solutions of good quality. The reason why the knn classification method seems to perform so well is because it keeps all the learning samples and tries to determine to which of these samples the problem instance in question is most similar. In addition, the performance of the naive Bayes classifier also improves when used by the dynamic DR selection procedure, which demonstrates that the performance of the classification methods also depends on the considered problem. Based on all the previous observations it can be concluded that knn and C4.5 represent the best choices for the classification method.

The number of problem instances and number of DRs which are used to construct the learning set, also represent two important parameters of the procedure. The experiments have shown that with the largest learning set size the procedure achieved the best results. However, medium sized learning sets achieved similarly good results, therefore they were selected since the learning process of the classification methods is shorter when using a smaller learning set. For the number of DRs that were used, the procedure performed better when between 10 and 30 DRs were used. With a smaller number of DRs the procedure does not seem to have enough expressiveness to cover all problem instances with an appropriate DR, and therefore ends up using an inappropriate rule for certain problem instances. On the other hand, a too large number of DRs also has a negative effect on the learning process, since many DRs will be associated to only a few problem instances. Because of this large variability, it will be much harder to train a good classifier from such a learning set.

The features which are used to perform the classification have also a large influence on the performance of the DR selection procedure. The static DR selection procedure achieved the best results when a larger number of features was used. It is interesting to note that although the

Twt criterion was optimised, using only the two due date related features did not lead to good results of the DR selection procedure. However, including the $\bar{p}$ feature, which approximates the makespan of the schedule, improved the performance of the procedure. Therefore, it seems to be important for the procedure to also have access to the information about the expected length of the schedule. When the dynamic DR selection procedure is used, the largest feature sets again lead to good results. However, the procedure also achieved good results even if the two smaller feature sets were used. Therefore it seems that for dynamic DR selection the additional features are much less useful, since the DR selection process is performed in discrete moments in time, and therefore the approximated duration of the schedule, or the job machine ratio do not hold as much information as in the static case, where they are used before the execution of the system to perform the decisions.

In addition to the previously described parameters, the dynamic scheduling procedure introduced several additional parameters, which are used to determine when the DR selection procedure should be performed. The first parameter was the one which determines when the decision should be performed. Although using a fixed time interval leads to the best results, the other two methods perform equally well, meaning that there is no significant difference in the performance of the procedure when using any of the three proposed methods. The second parameter determines how often the procedure should be applied. Surprisingly, even this parameter does not have a large influence on the performance of the procedure. Even when applied on problems with changing characteristics, most of the time the three methods performed equally well, with the repeat method sometimes achieving worse performance than the other two methods. The reason why the repeat method achieved worse performance than the other two, is because it calculates the features of the problem based on all released jobs, which can have a negative effect if the characteristics of the problem change over time. Therefore, the calculated features in the latter parts of the schedule might not be precise and correct. The no repeat method performs well because in many cases the DR it selected works well throughout the entire problem instance. The final parameter which is introduced determines the interval length before the DR selection is performed. The results show that for both, the time interval and number of released or scheduled jobs, it is more beneficial to use smaller values, meaning that the DR selection should be performed sooner, and more often if the procedure is performed several times. The benefit of this is that the influence of the initial DR is minimised. Also, if the DR selection is performed several times, it gives the procedure the possibility to perform its decision more often, and therefore to adapt better to the current problem instance and its changing characteristics.

From all the results denoted in the previous sections it can be concluded that the DR selection procedure performs better than by using a single manually selected DR for solving all problem instances. The procedure has proven to be viable in both the static and dynamic conditions. Therefore, if characteristics about the problem instance are known in advance, the procedure can be used to select the DR before the system even starts with its execution. However, since it is more likely that this information is not available, the procedure may also be applied in dynamic conditions as well. The results have shown that even for dynamic conditions the DR selection procedure achieved better results than a manually selected DR. The procedure achieved constantly good results, even when it was applied on different problem types, where in some the characteristics of the problems were constant through its execution, while in others the characteristics changed. The improvements which the procedure achieved were mostly the same for all three tested problem types. Therefore, it is evident that the procedure is appropriate for solving the different types of problems it was applied on.

7.7 Conclusion

Since a single DR cannot perform well on all possible problem instances, it is important to select an appropriate DR to obtain better a performance. However, in dynamic scheduling conditions it is difficult to know in advance which of the available DRs would be best suited for solving the concrete problem instance. Therefore, several studies proposed different procedures of selecting appropriate DRs for solving the current problem instance. Unfortunately, in all cases only manually designed DRs were considered, and neither of the studies used automatically generated DRs.

The purpose of this chapter was to present a procedure for selecting automatically generated DRs, based on characteristics of the currently solved problem instance, and analyse if by using this method it is possible to achieve better results than by using a single DR. The proposed procedure was applied in both static and dynamic conditions. In addition, the procedure was applied for several different problem types, in some of which the characteristics of the problem instances changed over time. Through the experiments the influence of different parameters of this procedure was analysed. In addition, for several parameter values the procedure was applied on an independent problem set in order to validate it and analyse its behaviour even further.

The obtained results have shown that the DR selection procedure performs well when applied in both static and dynamic scheduling conditions. In both cases the procedure achieved much better performance than by using a single manually selected DR would be applied for the entire problem set. Even when applied on problems where the characteristics change over time, the DR selection procedure achieved good results. The only drawback of the procedure is the large number of parameters for which good values need to be selected, since the performance of the procedure depends heavily on the selected values.

Based on all the previous observations, it can be concluded that the described DR selection procedure represents a viable addition to automatic generation of DRs, since it enables that an appropriate DR is selected for each problem instance. However, even if good results were achieved by this procedure, there are still many open topics which can be researched in the future. One possible research direction is to further examine which characteristics of scheduling problems could be extracted into features that could be used to describe the problem. These additional features could lead to a better performance of the DR selection procedure. The procedure can also be tested with other classification methods and machine learning methods which could be used to perform the decision. An especially interesting topic here would be to use GP to create the decision function for selecting the DRs, and evolve it at the same time when the DRs are evolved. In addition, instead performing the DR selection at fixed time moments, the procedure could also be extended so that it determines when the selection procedure should be performed.

Chapter 8

Design of DRs for static scheduling conditions

Although DRs are most commonly used in dynamic scheduling environments, there is no obstacle in applying them also for the static scheduling environments. Various metaheuristic methods are most often used for solving scheduling problems under static conditions, however, DRs provide two advantages over those methods. The first advantage is the superior execution speed of DRs over metaheuristic methods, which means that DRs will construct the schedule in a smaller amount of time. The second advantage is that DRs construct the schedule incrementally, which means that parts of the schedule which have been constructed can already be executed, while the rest of the schedule is being constructed by the DR. However, since DRs are designed for dynamic environments, they cannot usually match the performance obtained by metaheuristic approaches, since DRs do not use all the static information of the problem. Therefore there is a need to construct DRs which use the available static information about the problem to increase their performance.

Until now, research on this topic has been quite sparse. In one study, DRs which use look-ahead information were generated by GP [42]. Unfortunately, the experiments have shown that no significant improvements were achieved by additionally using look-ahead, and that there was a large variance of the obtained results. A new DR method, called iterative dispatching rules, which is more suitable for static scheduling, was also recently proposed in [43]. The proposed method achieved better performance than standard DRs evolved by GP. In addition, the method was also combined with look-ahead information, which was also evolved as a part of the DRs. The results have shown that the addition of the look-ahead information leads to even better performance. However, look-ahead was not applied by its own to analyse its effectiveness, and neither were the approaches compared to any metaheuristic methods, to determine how their performance compares with such methods. Iterative DRs were also applied for the unrelated machines scheduling problem [151], and their results were compared with results achieved by metaheuristic methods. However, this study did not provide any further analysis of other static scheduling methods.

The purpose of this chapter is to analyse several methods for adapting automatically generated DRs to static scheduling problems. In addition, all the methods will be compared to the results obtained by a metaheuristic method, and the difference between the execution times of the different methods will also be analysed. First, an overview of methods of adapting automatically generated DRs for static scheduling will be presented. After that, the performance of the different methods will be analysed. Finally, an analysis of the execution time of different methods will be presented and compared with the execution time of a metaheuristic optimisation method. The chapter is concluded with a short overview and guidelines for future research.

8.1 Design and adaptation of DRs for static scheduling

This section will describe four methods which will be used to adapt DRs for scheduling under static conditions. The first method will simply introduce new terminal nodes which will include certain static information about the scheduling environment. The second method will allow the DRs to use look-ahead to consider jobs which are still not released into the system. To further increase its performance the iterative DR method will also be applied with a larger number of nodes than in previous studies. Finally, the rollout algorithm will be applied for the first time with automatically generated DRs. This algorithm will use an automatically generated DR to approximate which scheduling decision leads to the best result.

8.1.1 Terminal nodes with static information

A simple and intuitive way of generating DRs for the static scheduling environment is to design additional terminal nodes, which contain static information about the system. With these terminals GP can design priority functions which will also use information about the future of the system to perform the next scheduling decision. However, the terminal nodes need to be selected very carefully to include only the information which will be useful for DRs.

The static terminal nodes which will be used in this thesis are presented in Table 8.1. These terminals can be divided into two groups. The first seven terminals denoted in the table represent the first group of terminals, which do not depend on the choice of which job would be scheduled next. This means that at the current decision point, these terminals will have the same value for each job which is considered. However, some of the terminals depend on the currently considered machine, which means that they will have different values for all the available machines. The terminals in this group represent general information about the future of the system, like the time until the next job arrives or the slack of the next job which will be released into the system.

The rest of the terminals in the table belong to the second group of static terminal nodes, that represents terminals which denote how the scheduling of one job could affect the future of the system. These terminals will be denoted as *casual static nodes*, since they analyse the causality on the future of the system if the currently considered job j would be scheduled on a given machine. The first four terminals in this group (*NREL*, *NRELM*, *SLAVGD* and *MLOADD*) are quite simple, since they only extract information about jobs which would be released during the execution of job j. On the other hand, the other twelve terminals try to approximate how much the scheduling of job j could influence the tardiness of jobs which would be released during the execution of job j. The first six terminals (from *FUTLATES* until *WLATEL*) approximate the tardiness of jobs which would be released during the execution of the currently considered job j. The other six terminals approximate the difference between the tardiness of job j, if it would be delayed and other jobs would be scheduled before it, and the tardiness of other jobs which would be released during the execution of job j, if job j would be executed immediately.

Table 8.1: List of static terminal nodes

Node name	Node description
NSHORT	Number of unreleased jobs which have the shortest processing time for the given machine
SLNXT	Slack of the next job that is released into the system
SLNXTM	Slack of the next job that is released into the system, and has the shortest processing time for the given machine
SLAVG	Average slack of jobs with the shortest processing time for the given machine
TTAR	Time until the next job arrives into the system
TTARM	Time until the next job, which has the shortest processing time for the given machine, arrives into the system
MLOAD	Sum of processing times of unreleased jobs, which have the shortest processing time for the given machine
NREL	Number of jobs which will be released during the execution of the selected job
NRELM	Number of jobs which will be released during the execution of the selected job, and have the shortest processing time for the given machine

SLAVGD	Slack of jobs which will be released during the execution of the selected job, and have the shortest processing time for the given machine
MLOADD	Sum of processing times of jobs released during the execution of the selected job, and which have the shortest processing time for the given machine
FUTLATES	Approximation of the weighted tardiness of the job which has the fastest execution time for the considered machine, and is released first during the execution of job j. The approximation is performed as if the considered job would be executed immediately after the completion of job j
WLATES	Approximation of the unit penalty of the job which has the fastest execution time for the considered machine, and is released first during the execution of job j. The approximation is performed as if the considered job would be executed immediately after the completion of job j
FUTLATE	Approximation of the weighted tardiness of all jobs which are released first during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed right after the completion of job j
WLATE	Approximation of the weighted number of tardy jobs, of all jobs which are released first during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed right after the completion of job j
FUTLATEL	Approximation of the weighted tardiness of all jobs which are released first during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed sequentially after the completion of job j
WLATEL	Approximation of the weighted number of tardy jobs, of all jobs which are released first during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed sequentially after the completion of job j
FLDS	Difference between the approximation of the weighted tardiness of job j and the value of the *FUTLATES* terminal. The weighted tardiness is approximated as if job j would be executed after the job that has the fastest execution time for the considered machine, and is released first during the execution of job j, finishes with its execution

WLDS	Difference between the approximation of the tardiness weight of job j and the value of the *WLATES* terminal. The tardiness weight is approximated as if job j would be executed after the job that has the fastest execution time for the considered machine, and is released first during the execution of job j, finishes with its execution
FLD	Difference between the approximation of the weighted tardiness of job j and the value of the *FUTLATE* terminal. The approximation is performed as if job j would be executed after all the jobs with the fastest execution time for the considered machine, which are released during the execution of job j, finish with their execution
WLD	Difference between the approximation of the tardiness weight of job j and the value of the *WLATE* terminal. The approximation is performed as if job j would be executed after all jobs with the fastest execution time for the considered machine, which are released during the execution of job j, finish with their execution
FLDL	Difference between the approximation of the weighted tardiness of job j and the value of the *FUTLATEL* terminal. The approximation is performed as if job j would be executed after all jobs with the fastest execution time for the considered machine, which are released during the execution of job j, finish with their execution
WLDL	Difference between the approximation of the tardiness weight of job j and the value of the *WLATEL* terminal. The approximation is performed as if job j would be executed after all jobs with the fastest execution time for the considered machine, which are released during the execution of job j, finish with their execution

Since the last twelve terminals are not trivial to calculate, details about their calculation are given in Algorithm 8.1. The terminals *FUTLATES*, *FUTLATEL*, and *FUTLATE* approximate the tardiness of jobs that would be released into the system during the execution of the currently considered job j. The difference between those terminals is only in the manner in which they approximate the tardiness. The *FUTLATES* terminal approximates the tardiness of the next job which would be released into the system during the execution of job j and has the fastest processing time on the machine m on which job j is executing. The approximation is performed in a way that the tardiness of the next job is calculated as if it would be scheduled on machine m after job j finished with its execution. The *FUTLATE* terminal considers all jobs which would be released during the execution of job j, and which achieve the fastest processing time on machine m. The tardiness of all these jobs is approximated as if each job would be executed on machine m after the completion of job j, independently of each other. Since this is a quite optimistic approximation, the *FUTLATEL* terminal, which represents a much more pessimistic approximation, is introduced. This terminal approximates the tardiness in a similar way as *FUTLATE*, however, it calculates the tardiness as if all the jobs were scheduled sequentially on machine m, in order of their release times, after job j finishes with its execution. The terminals *WLATES*, *WLATE* and *WLATEL* perform the approximation of the weighted number of tardy jobs in the same manner as *FUTLATES*, *FUTLATE*, and *FUTLATEL*, respectively.

Unfortunately, the six previously described terminals approximate only the tardiness of jobs which would be released during the execution of job j, but do not take into account the tardiness of job j if its execution would be delayed. For that reason, six additional terminals are defined, which also approximate the tardiness of job j if it were delayed. The *FLDS* terminal approximates the tardiness of job j in a way that it delays its execution until the job which would first be released into the system during the execution of job j, and has the smallest processing time on machine m, finishes with its execution. The value of the *FUTLATE* terminal is additionally subtracted to determine which decision would lead to a greater tardiness value. The *WLDS* terminal uses the same concept, just for approximating the weighted number of tardy jobs, and subtracting its value with the value of the *WLATES* terminal. For the *FLD* and *FLDL* terminals, the approximation of the tardiness of job j is performed somewhat differently. For those two terminals, the tardiness of job j is approximated as if job j would be executed after all jobs which were released during its execution and have the smallest processing time on machine m, would be executed sequentially on the considered machine m. For the *FLD* terminal, the approximation is additionally subtracted by the value of the *FUTLATE* terminal, while for the *FLDL* terminal the approximated tardiness value is subtracted with value of the *FUTLATEL* terminal. The last two terminals, *WLD* and *WLDL*, also use the same concepts as *FLD* and *FLDL* to approximate the number of tardy jobs, and subtract the approximation by the *WLATE* and *WLATEL* terminals, respectively.

8.1.2 Look-ahead

In the dynamic scheduling environment, DRs use the priority function to calculate the priorities only of those jobs which were already released into the system. Since in the static scheduling environment the information about all jobs is known in advance, the DRs can be extended so that they also calculate the priorities for jobs which are not yet released into the system. This property, that DRs also calculate priorities for yet unreleased jobs, is called *look-ahead*. Algorithm 8.2 represents the adapted schedule generation scheme which uses look-ahead. The first thing which can be noticed in the schedule generation scheme, is that it uses the look-ahead fac-

Algorithm 8.1 Calculation procedure of terminals which approximate the tardiness and weighted number of tardy jobs values

```
1:  Let $j$ denote the job which should be scheduled on machine $i$
2:  Let $time$ denote the current system time
3:  $startTime \leftarrow time$
4:  $shouldSet \leftarrow true$
5:  $tSum \leftarrow \max(time, r_j) + p_{ij}$
6:  $wtardL \leftarrow 0$
7:  $wlateL \leftarrow 0$
8:  $wtard \leftarrow 0$
9:  $wlate \leftarrow 0$
10: $wtardS \leftarrow 0$
11: $wlateS \leftarrow 0$
12: $wEnd \leftarrow 0$
13: for each job $k$ which has the minimum processing time for machine $i$, and is released during
    the execution of job $j$ do
14:     if $r_k > startTime$ then
15:         $startTime \leftarrow r_k$
16:     end if
17:     $startTime += p_{ik}$
18:     if $d_k < (tsum + p_{ik})$ then
19:         $initialLate \leftarrow 0$
20:         if $d_k < (r_k + p_{ik})$ then
21:             $initialLate \leftarrow (p_{ik} + r_k - d_k) * w_{T_k}$
22:         end if
23:         $wtardL += (p_{ik} + tSum - d_k) * w_{T_k} - initialLate$
24:         if $initialLate == 0$ then
25:             $wlateL += w_{T_k}$
26:         end if
27:     end if
28:     $tSum += p_{ij}$
29:     if $d_k < (\max(r_j, time) + p_{ij} + p_{ik})$ then
30:         $initialLate \leftarrow 0$
31:         if $d_k < (r_k + p_{ik})$ then
32:             $initialLate \leftarrow (p_{ik} + r_k - d_k) * w_{T_k}$
33:         end if
34:         $wtard += (p_{ij} + p_{ik} - d_k + \max(r_j, time)) * w_{T_k} - initialLate$
35:         if $initialLate == 0$ then
36:             $wlate += w_{T_k}$
37:         end if
38:         if $shouldSet$ then
39:             $shouldSet \leftarrow false$
40:             $wtardS \leftarrow wtard$
41:             $wlateS \leftarrow wlate$
42:             $wEnd \leftarrow r_k + p_{ik}$
43:         end if
44:     end if
45: end for
```

```
46: $jInitialLate \leftarrow 0$
47: if $d_j < (r_j + p_{ij})$ then
48:     $jInitialLate \leftarrow (p_{ij} + r_j - d_j) * w_{T_j}$
49: end if
50: if $d_j < (startTime + p_{ij})$ && $initialLate == 0$ then
51:     $lateWeight \leftarrow w_{T_j}$
52: else
53:     $lateWeight \leftarrow 0$
54: end if
55: if $d_j < (wEnd + p_{ij})$ && $initialLate == 0$ then
56:     $lateWeightS \leftarrow w_{T_j}$
57: else
58:     $lateWeightS \leftarrow 0$
59: end if
60: $tardinessWeight \leftarrow 0$
61: $tardinessWeightS \leftarrow 0$
62: if $d_j < (startTime + p_{ij})$ then
63:     $tardinessWeight \leftarrow (startTime + p_{ij} - d_j) * w_{T_j} - initialLate$
64: end if
65: if $d_j < (wEnd + p_{ij})$ then
66:     $tardinessWeightS \leftarrow (wEnd + p_{ij} - d_j) * w_{T_j} - initialLate$
67: end if
68: $FUTLATES \leftarrow wTards$
69: $WLATES \leftarrow wLates$
70: $FUTLATEL \leftarrow wtardL$
71: $WLATEL \leftarrow wlateL$
72: $FUTLATE \leftarrow wTard$
73: $WLATE \leftarrow wLate$
74: $FLDS \leftarrow tardinessWeightS - wTards$
75: $WLDS \leftarrow lateWeightS - wLates$
76: $FLDL \leftarrow tardinessWeight - wTardL$
77: $WLDL \leftarrow lateWeight - wlateL$
78: $FLD \leftarrow tardinessWeight - wTard$
79: $WLD \leftarrow lateWeight - wLate$
```

tor α to determine the amount of unreleased jobs which will be considered when calculating the priority values. The set of jobs which are considered when calculating the priority values will be denoted as the *look-ahead horizon*. The look-ahead parameter is quite sensitive to the size of the problem instance, meaning that more unreleased jobs will be considered in each iteration if there are more jobs in the scheduling problem. Instead of using this look-ahead factor, a fixed number of unreleased jobs that should be considered in each iteration can also be used. This has the advantage that regardless of the problem instance size, the same number of jobs will always be considered. The second important thing which can be observed from the algorithm, is that jobs which are not yet released will not be scheduled until they are actually released into the system. Therefore, there is a possibility that in the meantime another job is scheduled on the machine for which previously an unreleased job achieved the largest priority. When using look-ahead, a new terminal node also needs to be introduced. This terminal, denoted as *AR*, determines the amount of time until the job is released into the system. Without this terminal DRs can not take into account when the jobs become ready, and would consequentially be unable to prioritise jobs which are released sooner. The overall benefit of look-ahead is that it allows the schedule generation scheme to introduce idle times into the schedule, if it determines that in the near future a job that needs to be scheduled immediately will be released.

Algorithm 8.2 Schedule generation scheme used for DRs with look-ahead

```
1: while unscheduled jobs are available do
2:     Wait until at least one job and one machine are available
3:     for all jobs where $r_j < (time + (\max_j(r_j) - time) * \alpha)$ and all machines do
4:         Obtain the priority $\pi_{ij}$ of scheduling job $j$ on machine $i$
5:     end for
6:     for all jobs where $r_j < (time + (\max_j(r_j) - time) * \alpha)$ do
7:         Determine the best machine (the one for which the best value of priority $\pi_{ij}$
8:         is achieved)
9:     end for
10:    while jobs whose best machine is available exist do
11:        Determine the best priority of all such jobs
12:        Schedule the job with best priority if it is released
13:    end while
14: end while
```

8.1.3 Iterative dispatching rules

Iterative dispatching rules (IDRs), unlike standard DRs, construct the schedule several times. Each time a new schedule is constructed IDRs use information from previously generated schedules to improve the newly constructed schedules. The schedule is reconstructed until the fitness of the newly constructed schedule stops improving. The motivation behind this approach is that by using information from previously created schedules, IDRs could correct mistakes made in previous iterations. The steps of the schedule generation scheme used by IDRs are shown in Algorithm 8.3. The algorithm shows that new schedules are created as long as the fitness of the schedule is improved. In the end, the procedure does not return the schedule which was created in the last iteration, but the previous one, since that schedule achieved the best fitness value.

Algorithm 8.3 Schedule generation scheme used by IDRs

```
1:  Let $R$ represent the set of parameters extracted from the previous schedule which are used
    by the priority function, and let $R_0$ represent their initial values
2:  $R \leftarrow R_0$
3:  $Fitness^* \leftarrow \infty$
4:  Let $S$ represent the current schedule (empty at the begging), and $bestS$ the best created
    schedule
5:  do
6:      $bestS \leftarrow S$
7:      Generate the schedule using the standard schedule generation scheme and the priority
        function $\pi$
8:      $S \leftarrow$ generated schedule
9:      $Fitness^* \leftarrow Fitness$
10:     $Fitness \leftarrow$ fitness value of the generated schedule $S$
11:     Calculate new values for schedule dependant nodes, based on the constructed schedule
        $S$, and store the calculated values in $R$
12: while ($Fitness^* > Fitness$)
13: Return $bestS$ as the result
```

In order for the priority function to use information about previously created schedules, additional nodes need to be defined, the values of which will be calculated based the schedule created previously by the IDR. These nodes are updated every time a new schedule is created. Table 8.2 represents additional nodes which use information from previously created schedules. All nodes represent terminal nodes, apart from the *ISLATE* node, which is a function that executes one branch if the currently considered job was late in the previous schedule, and the other branch if not. The aim of this node is to create priority functions where one part of the function is appropriate for scheduling jobs which were late in the previous schedule, making it possible to apply a different scheduling strategy for those jobs. The terminal nodes were designed to provide different information from the previous schedule. The *NLATE*, *LATENESS*, and *TARDINESS* nodes provide information about the entire previously created schedule, in the form of the total number of tardy jobs, total lateness of the schedule, and total weighted tardiness of the schedule. Nodes *INDLATE*, *INDWTARD*, and *INDTARD* provide information about the lateness, weighted tardiness, and tardiness of a single job in the previous schedule. By using these nodes the priority function can put more emphasis on jobs which were tardy in the previous schedule. Finally, nodes *JOBFINISH* and *FLOWTIME* provide information about the completion time and the flowtime of jobs in the previous schedule. Although these nodes

Table 8.2: Additional nodes used by IDRs

Node name	Node description
NLATE	number of tardy jobs in the previous schedule
LATENESS	total lateness of the entire previously created schedule
INDLATE	lateness of a concrete job in the previous schedule
TARDINESS	total weighted tardiness of the entire previously created schedule
INDTARD	tardiness of a concrete job in the previous schedule
INDWTARD	weighted tardiness of a concrete job in the previous schedule
ISLATE	if the job was late in the previous schedule the left branch of the node is executed, otherwise the right branch is executed
JOBFINISH	completion time of a concrete job in the previous schedule
FLOWTIME	flowtime of a concrete job in the previous schedule

do not provide any due date related information, they were included to analyse whether this kind of information could also be useful for IDRs. An additional thing which also needs to be defined for these nodes are their initial values which are used in the first iteration, when there is no previous schedule from which the information could be extracted. In that case, all the nodes are initialised to large values which can not be achieved by any schedule constructed by the schedule generation scheme, while for the $ISLATE$ node all jobs are denoted as late.

8.1.4 Rollout algorithm

The rollout algorithm is a simple approach which can improve the results of different heuristic methods [296, 297, 298, 299, 300]. The algorithm tries to balance between exhaustive search and heuristic methods to perform better than heuristic methods, but still be able to obtain solutions faster than exhaustive search. In order to achieve this, at each decision moment the rollout algorithm considers all possible decisions. However, to determine which one of these decisions is the best, the algorithm does not perform an exhaustive search for each of those decisions, but rather uses a heuristic method which, for each possible decision, constructs the rest of the solution. The algorithm then performs the decision which leads to the best solution after applying the heuristic method. These steps are repeated for each decision moment until the entire solution is constructed.

The rollout algorithm can easily be combined with DRs in a way that at each decision moment and for each possible decision at that point, the rollout algorithm applies a DR to construct the rest of the schedule. After constructing the schedule for each decision, the algorithm performs the decision for which the DR obtained a schedule of the best quality. Algorithm 8.4 denotes the steps of the rollout algorithm for solving scheduling problems with DRs. In the first part, the algorithm tries out all possible scheduling decisions at the current moment in time, and uses a predefined DR to construct the rest of the schedule from that decision onwards. Unfortunately, applying the rollout algorithm in this way will lead to bad solutions, since the rollout algorithm can only create schedules in which jobs are scheduled immediately on a machine if it is free. On the other hand, DRs can introduce idle times in the schedule even if there are available machines. Therefore, it is possible that the DRs construct a schedule approximation which can not be obtained by the rollout algorithm, since it can not introduce idle times in the schedule. As a consequence, instead of improving the fitness of the schedule during the execution of the rollout algorithm, the fitness will oscillate and will sometimes be even worse than that obtained by the DR. To solve this problem it is required to determine if in the current iteration the best decision, which can be performed by the rollout algorithm, leads to a fitness value which is worse than the one obtained by the algorithm in the last iteration. When this situation occurs, instead of performing the decision selected by the rollout algorithm, the DR is used to perform the next scheduling decision. Therefore, if the rollout algorithm will in itself not be able to perform the best decision, it will delegate this task to the underlying DR, which can then easily introduce idle times into the schedule. This modification of the rollout algorithm will ensure that the fitness of the schedule decreases monotonically during the execution of the rollout algorithm.

The execution time of the rollout algorithm can also be improved if not all jobs need to be considered at each decision moment. The intuition behind this is that jobs which will be released far in the future will have a small probability of being scheduled at the current decision moment. Therefore, it is preferable to consider only a smaller number of yet unreleased jobs which have a closer release time to the current decision moment. For that purpose it is possible to define a rollout factor γ or a number of unreleased jobs which will be considered in each iteration, similarly as in the look-ahead approach. The set of jobs which is considered in each iteration will be denoted as the *rollout horizon*.

8.1.5 Combination of static methods

The benefit of the previous four methods is that they can be combined in various ways to improve the results even further. All methods can easily be combined with each other without any additional adjustments, except for two cases. The first case is when combining static terminals with look-ahead. The problem here is that static terminals are calculated based on all unreleased jobs, however, in look-ahead the priorities are calculated even for some unreleased jobs. This would mean that the properties of unreleased jobs which are considered by the DRs would still be used in the calculation of the static terminal nodes. Therefore, when using look-ahead, it

Algorithm 8.4 Rollout algorithm for scheduling with DRs

```
1: $time \leftarrow 0$
2: $previousFitness \leftarrow \infty$
3: $bestFitness \leftarrow \infty$
4: while unscheduled jobs are available do
5:     Set $time$ to the next point in time where there is at least one released job and one available machine
6:     for each unscheduled job $j$ where $r_j < (time + (\max_j(r_j) - time) * \gamma)$ do
7:         for each machine $m$ do
8:             Use a DR to construct the rest of the schedule when job $j$ would be scheduled on machine $m$.
9:             Let $fitness$ denote the fitness of the constructed schedule.
10:            if $fitness < bestFitness$ then
11:                $bestFitness \leftarrow fitness$
12:                Let $bestPair$ denote the selected job-machine pair
13:            end if
14:        end for
15:    end for
16:    if $previousFitness > bestFitness$ then
17:        $previousFitness \leftarrow bestFitness$
18:        Schedule the job from $bestPair$ on the machine from $bestPair$
19:    else
20:        Execute the DR to perform the next scheduling decision
21:    end if
22: end while
```

could prove to be beneficial to calculate the static terminals only from those jobs which are currently outside the look-ahead horizon. In the result section, both methods of calculating the static terminals will be tested, to determine their influence on the quality of the look-ahead method.

The second problematic case is when trying to combine IDRs with the rollout algorithm. The problem here arises from the fact that IDRs need to reconstruct the entire schedule, which would mean that the schedule constructed by the rollout algorithm would be lost, and that probably a schedule of inferior quality would be constructed. This could be fixed by not recreating the entire schedule with IDRs, but only those parts which were not constructed by the rollout algorithm. However, because IDRs use certain information from previously generated schedules, any changes introduced in the schedules outside the IDRs can have a significant influence on the performance of IDRs. This leads to a great instability of the approach, since in each iteration the approximation obtained by IDRs would be quite different. Therefore, the entire rollout algorithm would perform poorly, since the IDRs would not properly guide the algorithm. Because of those reasons the combination of the rollout algorithm and IDRs will not be considered.

8.2 Results

In this section, the results for all the tested methods that adapt DRs for scheduling under static conditions will be presented. Each experiment was executed 30 times and the minimum, median, and maximum values were calculated based on the best results achieved in each run. In all the experiments the *Twt* criterion was be optimised. Statistical tests were also performed to additionally test whether there is a significant difference between DRs evolved by GP which uses only dynamic information (DGP), and methods which are developed for static scheduling. The results of these tests are presented in an additional column denoted as *stat. diff.* The experiments in which the static methods achieve significantly better results than DGP will be denoted with $+$ in the column. On the other hand, $\approx$ will denote that there is no significant difference between the results, whereas $-$ will denote that the static methods achieved significantly worse results than DGP. The results of DGP are also included in tables to allow for an easier comparison. However, for all experiments with the rollout algorithm the statistical difference was tested against the results of a GA, since the rollout algorithm always achieves significantly better results than DGP. The applied GA will use the floating point representation described in Section 2.4.1.

8.2.1 Results obtained by DRs with static terminal nodes

In this section the results of using additional static terminal nodes will be presented. All the proposed static nodes will be used in addition to all terminal nodes used for generating DRs for the dynamic environment. Since 23 static nodes were proposed, it is impossible to try out all node combinations to find the best one. For that reason, two greedy heuristics are used to guide the selection of static nodes, the *constructive* and *destructive* heuristic. The constructive heuristic starts with a node set which contains only nodes used for the dynamic environment. For each static node the heuristic adds it to the set of terminal nodes which are used by GP, performs 30 runs and saves the overall minimum value, and then removes the selected terminal node from the set. The node for which GP achieved the best minimum value is selected and added permanently to the set of terminal nodes used by GP. In the next iteration the procedure is repeated, however, the nodes which were already permanently added to the set of nodes used by GP are not considered any more. The entire procedure continues until all the static nodes are added to the set of terminal nodes used by GP. From the description it is evident that the heuristic adds static nodes, one by one, to the set of terminal nodes used by GP, in a way that it selects the node for which the best result will be obtained. Naturally, it is possible that the fitness deteriorates from one iteration to the other. Although the procedure could also be stopped at the point where there is no improvement in the fitness by adding any static terminal node, it is better to continue the procedure, since the quality of the generated DRs can still increase if further nodes are added. Out of all the created sets of static terminal nodes, the one which achieved the best performance can be selected. The destructive heuristic, on the other hand, starts by adding all static nodes to the set of nodes used by GP. After that, it determines the node whose removal would result in the best performance of GP. The selected node is removed, and the entire procedure is repeated until there are no more static nodes which can be removed from the set of nodes used by GP. Therefore, this procedure gradually decreases the number of nodes, trying to remove those nodes which do not provide any important information.

Since the number of tested combinations of static nodes is vast even when using the two heuristics, only 21 combinations which achieved the best minimum values are selected and presented in Table 8.3. The last line in the table, denoted as DGP, represents the results achieved by GP without the use of static terminal nodes. The best values achieved by all the results are denoted in bold. The table shows that although GP achieved a better minimum value by using all presented static node combinations, the median value is not always better than that of DGP. DRs with static nodes achieved a maximum improvement over DGP by 6.9% for the minimum value, and 2.9% for the median value. Therefore, by using static terminal nodes it is possible to obtain much better DRs, although it is more difficult for the algorithm to evolve such good rules. The statistical tests also show that only in three cases GP with static terminal nodes achieves statistically better results than DGP, while in all other cases there was no statistically significant difference. Out of all the combinations denoted in the table, the first five were generated by the constructive heuristic, while the remaining were generated by the destructive heuristic. Therefore, better terminal node sets can be constructed by iteratively removing the nodes from the terminal node set. The best overall DR was generated when using the static terminal node combination denoted with index 21. This terminal set consists mostly of nodes which provide information about the tardiness of future jobs, as well as about their slack values. An additional thing which is evident from the table is that the best minimum values are mostly achieved by experiments which use smaller terminal node sets, such as those used by experiments 1, 17, 18, 19, and 21. In addition, in all these terminal node sets different terminal node types were used, which leads to the conclusion that better results will be achieved by simultaneously using nodes which provide different kinds of information to the DR.

Table 8.3: Results obtained by using additional static terminal nodes

	Static terminal node set	Min	Med	Max	Stat. diff.
1	*FLDL , FUTLATES, SLAVGM, TTAR*	12.17	13.82	15.27	$\approx$
2	*FLDL, FUTLATES, MLOAD, NREL, NRELM, SLAVGM, TTAR*	12.14	13.53	15.77	$\approx$

3	*FLDL, FLD, FUTLATES, MLOAD, NSHORT, NREL, NRELM, SLAVGD, SLAVGM, SLNEXT, TTAR, TTARM, WLATES, WLDS*	12.25	13.59	15.44	≈
4	*FLDL, FLD, FUTLATE, FUTLATES, MLOAD, NSHORT, NREL, NRELM, SLAVGD, SLAVGM, SLNEXT, TTAR, TTARM, WLATES, WLDS*	12.26	13.70	15.93	≈
5	*FLD, FLDL, FUTLATE, FUTLATEL, FUTLATES, MLOAD, MLOADD, NREL, NRELM, NSHORT, SLAVGD, SLAVGM, SLNEXT, SLNEXTM, TTAR, TTARM, WLATE, WLATES, WLATEL, WLDL, WLDS*	12.29	13.40	14.76	≈
6	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOAD, MLOADD, NREL, NSHORT, SLAVGD, SLAVGM, SLNEXT, SLNEXTM, TTAR, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.29	13.69	15.44	≈
7	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOAD, MLOADD, NREL, NRELM, NSHORT, SLAVGD, SLAVGM, SLNEXT, TTAR, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.30	13.55	15.66	≈
8	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOAD, MLOADD, NREL, NRELM, NSHORT, SLAVGD, SLAVGM, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.25	13.32	15.08	≈
9	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOADD, NREL, NRELM, NSHORT, SLAVGD, SLAVGM, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.27	13.72	15.65	≈

10	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOADD, NREL, NRELM, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.22	13.46	16.87	$\approx$
11	*FLD, FLDS, FUTLATE, FUTLATES, MLOADD, NREL, NRELM, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.28	13.62	15.40	$\approx$
12	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.20	13.83	15.37	$\approx$
13	*FLD, FLDS, FUTLATE, FUTLATES, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDS*	12.29	13.43	15.39	$\approx$
14	*FLD, FLDS, FUTLATE, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDS*	12.27	**13.21**	15.10	$+$
15	*FLD, FLDS, FUTLATE, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, TTARM, WLATE, WLATEL, WLATES, WLD, WLDS*	12.30	13.46	16.01	$\approx$
16	*FLD, FUTLATE, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, WLATES, WLD*	12.26	13.30	15.34	$\approx$
17	*FLD, FUTLATE, MLOADD, NREL, SLAVGD, SLNEXT, WLATES, WLD*	12.08	13.30	**14.62**	$+$
18	*FLD, FUTLATE, MLOADD, NREL, NSHORT, SLNEXT, WLATES, WLD*	12.08	13.42	16.06	$\approx$

19	*FLD, FUTLATE, MLOADD, NREL, NSHORT, SLAVGD, WLATES, WLD*	12.15	13.62	14.70	≈
20	*FLD, FUTLATE, NREL, SLAVGD, SLNEXT, WLATES, WLD*	12.24	13.29	16.50	≈
21	*FUTLATES, NREL, SLAVGD, SLNEXT, WLD*	**12.06**	13.30	16.59	+
	DGP	12.96	13.60	14.62	

Figure 8.1 shows the box plot representation of the results obtained from different static terminal combinations. The box plot shows that by using the static terminal nodes GP usually finds quite dispersed results, with a much greater possibility of obtaining solutions which represent outliers. Because of that large dispersion, the improvement in the median and average values of GP with static terminal nodes over DGP are in best cases quite small. It is also evident that for all the static terminal node combinations quite similar minimum values are obtained, meaning that most of the tested node combinations have a similar expressiveness.

8.2.2 Results obtained by DRs with look-ahead

This section will present the results achieved by using look-ahead in automatically designed DRs. Look-ahead will be tested with both the look-ahead factor and a fixed number of jobs in the look-ahead horizon, to determine how this parameter influences the overall procedure.

Table 8.4 represents the results achieved by DRs which additionally use look-ahead when calculating the priorities of jobs. In this case, DRs will use priority functions generated by using the terminal set consisting only of nodes used for the dynamic scheduling environment, and the additional *AR* node. The table demonstrates that by using look-ahead DRs achieved better results than those generated by DGP. The only time when DRs with look-ahead significantly outperformed DRs generated by DGP is when a small look-ahead factor was used. However, this is expected since this means that for smaller problem instances the DRs will not even use look-ahead, and will therefore perform similarly as DRs without look-ahead. However, with larger look-ahead factor values, DRs with look-ahead significantly outperform DRs without look-ahead by at most 13.8% for the minimum value, and 11.8% for the median value. The best results are achieved when using larger values for the look-ahead factor, which is expected since DRs will have a broader overlook on jobs which arrive into the system.

On the other hand, when using a fixed number of jobs in the horizon, DRs with look-ahead consistently performed better than DRs without look-ahead, even for a smaller number of jobs in the look-ahead horizon. The improvements which are achieved over DRs without look-ahead

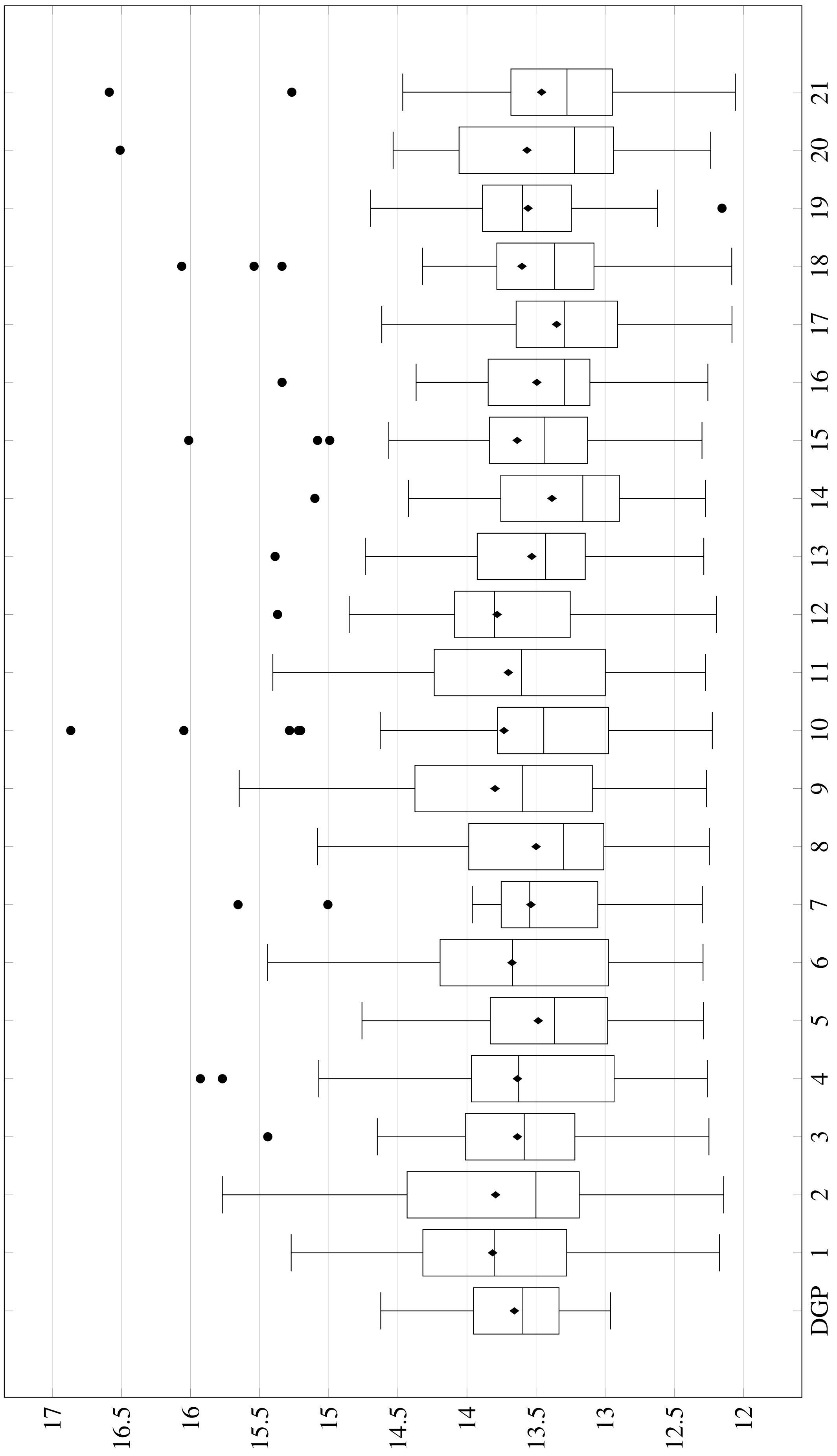


Figure 8.1: Box plot representation of the results obtained by using different static node combinations

Table 8.4: Results for DRs with look-ahead obtained by using only terminal nodes for dynamic scheduling

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	11.88	13.45	15.56	$\approx$	3	11.24	12.25	15.06	+
0.05	12.20	13.53	15.16	$\approx$	5	11.31	12.06	14.32	+
0.1	12.12	12.78	14.55	+	10	**10.82**	11.83	14.60	+
0.2	11.67	12.44	**13.89**	+	20	10.96	11.83	13.60	+
0.5	11.30	**11.99**	14.86	+	50	11.17	11.69	15.30	+
1	**11.17**	12.02	15.35	+	100	11.02	**11.64**	**13.53**	+
DGP	12.96	13.60	14.62			12.96	13.60	14.62	

are at most 16.5% for the minimum value, and 14.4% for the median value. The results demonstrate that with the increase of the number of jobs in the look-ahead horizon the performance gradually increases. However, the improvement of the performance is not as drastic as when using the look-ahead factor, since DRs that use a small number of jobs in the look-ahead horizon also achieve quite good results. An additional interesting behaviour which can be observed from the results is that the increase in the number of jobs in the look-ahead horizon will not always lead to better results. This is evident by comparing results achieved when using 10 and 20, or 50 and 100 jobs in the look-ahead horizon, since the median values achieved for those experiments were mostly similar.

The results also display that DRs achieve better performance when using a fixed number of jobs in the look-ahead horizon, than when using a look-ahead factor to determine which jobs belong to the horizon. The main reason for such a behaviour is that by using a constant number of jobs in the horizon the procedure is more stable since it will always consider the same number of unreleased jobs. On the other hand, when using the look-ahead factor, the number of unreleased jobs which are considered depends not only on the number of jobs in the problem instance, but also on the distribution of the release times of jobs. This is due to the fact that the look-ahead factor is used only to define a time window, and jobs that are released during that time window belong to the look-ahead horizon. However, it is possible that in certain time windows no jobs are released, and therefore no unreleased jobs would be considered. Because of this variability the performance of DRs is not as good as when using a constant number of jobs in the look-ahead horizon.

Figure 8.2 represents the box plot representation of the results. The figure demonstrates that when designing DRs with look-ahead it is more likely that outlier solutions will appear. How-

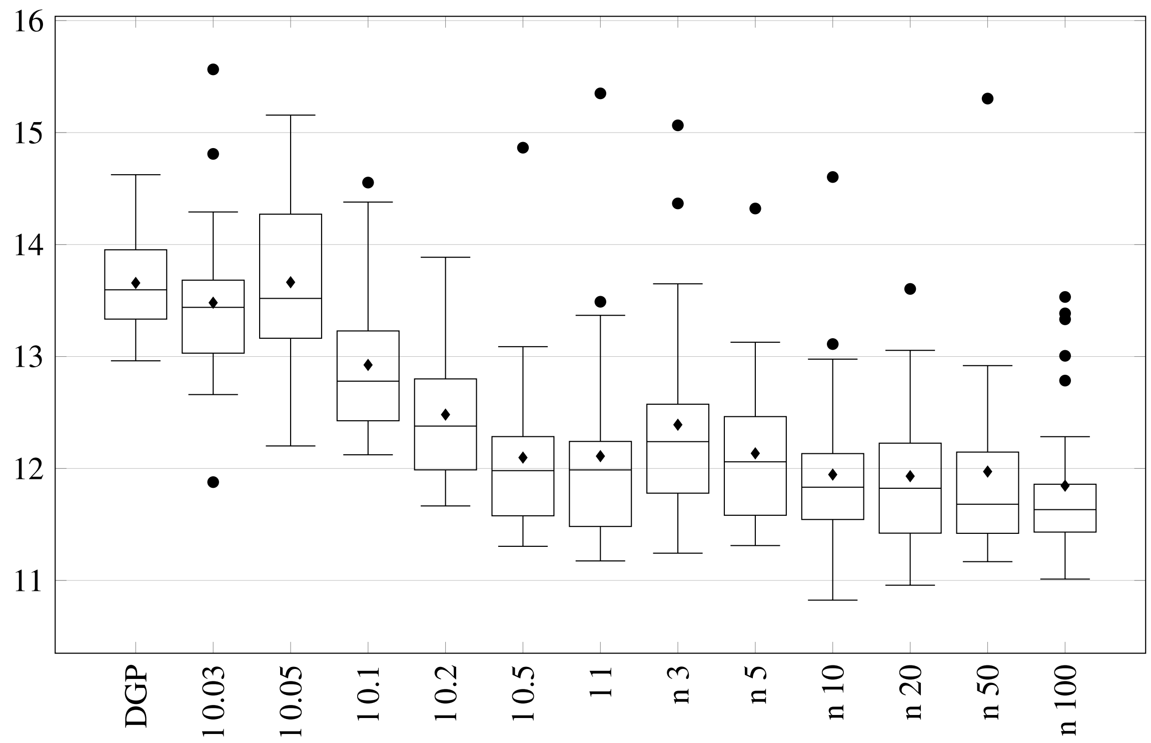


Figure 8.2: Box plot representation of the results obtained by using DRs with look-ahead

ever, most of the remaining solutions will obtain better performance than the DRs generated by DGP. In addition, the figure depicts that the solution distributions for look-ahead factors of 0.5 and 1, and for certain numbers of jobs in the look-ahead horizon (those of 10, 20, 50, 100) are quite similar with smaller variations in the median and minimum values. This means that it is possible to use smaller values of the parameters without a large deterioration in the results. Based on all the aforementioned observations it can be concluded the DRs with look-ahead generally achieve much better results than DRs generated by DGP.

8.2.3 Results obtained by look-ahead with static terminal nodes

In the previous section no additional nodes which contain static information, except for the *AR* node, were used. To test whether it is beneficial to provide additional static information to DRs which use look-ahead, certain static nodes will be included into the terminal set of GP when generating DRs which use look-ahead. Since trying out all possible combinations of static terminal nodes and look-ahead parameters would be too time consuming, only the set of static terminal nodes which achieved the best overall minimum value will be used to analyse the influence of the look-ahead parameters. This terminal node combination is denoted with the index 21 in Table 8.3. After the influence of the look-ahead parameters is analysed, the values of the look-ahead parameters were fixed to a certain value, and the remaining 20 static node combinations from Table 8.3 were tested with the optimised value of the look-ahead parameter.

Table 8.5: Results obtained by DRs with look-ahead and static terminal nodes calculated based on all unreleased jobs

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	12.25	13.60	15.34	≈	3	11.43	12.92	14.23	+
0.05	12.19	13.99	15.46	−	5	12.05	12.82	16.10	+
0.1	11.97	12.98	14.44	+	10	**11.22**	**12.46**	**14.21**	+
0.2	12.15	12.94	**13.94**	+	20	11.53	12.91	19.98	+
0.5	**11.23**	**12.55**	15.04	+	50	11.53	12.67	17.89	+
1	11.60	12.78	15.07	+	100	11.59	12.83	14.38	+
DGP	12.96	13.60	14.62			12.96	13.60	14.62	

In addition, two methods of calculating the static terminal nodes were used, the first which calculates the static terminals based on all unreleased jobs, and the other which calculates the static terminals based on unreleased jobs outside the look-ahead horizon.

Table 8.5 represents the results of DRs which use look-ahead together with static terminals calculated based on all yet unreleased jobs. The results show that, similarly as when not using static nodes, DRs with look-ahead outperform DRs generated by DGP in all cases, except when using the look-ahead parameter values of 0.03 and 0.05. The improvements achieved over DGP were this time at most 13.3% for the minimum value, and 8.4% for the median value. By comparing the results of look-ahead with and without static nodes, it can be concluded that the addition of static nodes does not lead to improvements in the results, but rather that the results deteriorate. This is backed up by statistical tests which show that for the same look-ahead parameter values, DRs which use look-ahead and static nodes were unable to outperform DRs which only use look-ahead.

Figure 8.3 represents the box plot representation of the results. The "l" label represents that the look-ahead parameter of the specified value is used, while the "n" label represents that the specified number of jobs in the look-ahead horizon is used. The figure shows that this time there is no large difference between the results obtained by using either the look-ahead parameter or the number of jobs in the look-ahead horizon. By using small values for the number of jobs in the look-ahead horizon the procedure achieves good results, however, the improvements in the results do not increase significantly when increasing the value of the parameter. The figure also denotes that the procedure is less stable than DGP, since it achieved more dispersed results, and quite large outlier solutions for certain parameter values.

Table 8.6 represents the results achieved by DRs with look-head and static terminal nodes

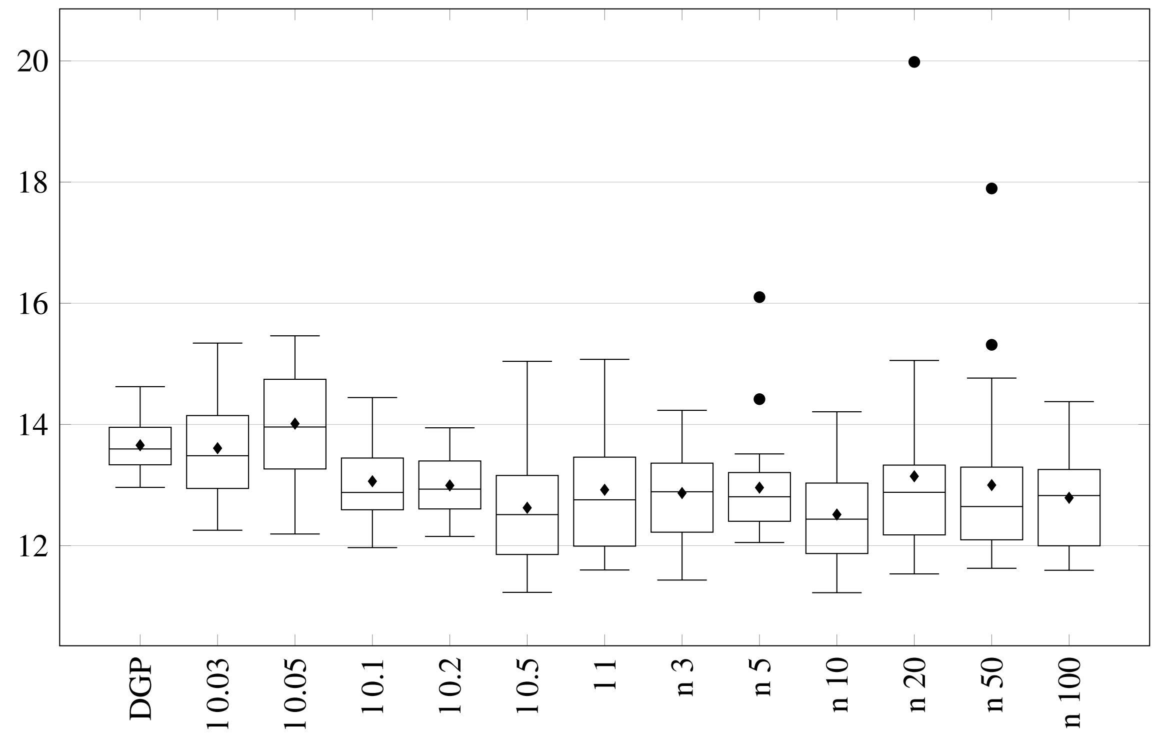


Figure 8.3: Box plot representation of the results obtained by DRs with look-ahead and static terminal nodes calculated based on all unreleased jobs

calculated out of unreleased jobs outside the look-ahead horizon. With this calculation method of static nodes, improvements of at most 15.9% for the minimum value, and 14.4% for the median value over DRs generated by DGP were achieved. Therefore, when calculating the values of static nodes, better results were obtained when the values of those nodes were calculated only based on the unreleased jobs outside the look-ahead horizon. In such a way, jobs which are in the look-ahead horizon will not influence the values of the static terminal nodes, and the whole procedure will be more stable. However, with this calculation method it can happen that if a too large look-ahead horizon is selected, that very soon in the procedure static terminals will become useless, since all jobs will be inside of the look-ahead horizon. This can best be seen when using the constant number of jobs in the look-ahead horizon, where the results improve until the value of the parameter reaches 20, after which the results start to deteriorate. Therefore, smaller look-ahead horizons should be preferred with this calculation method, so that static terminal nodes still provide useful information. The table shows that for this calculation method GP obtained results which were significantly better than that of DGP for all parameter values, except for the two smallest values of the look-ahead factor. Unfortunately, neither this calculation method achieved results which were significantly better than those when using look-ahead without static terminals.

Figure 8.4 represents the box plot representation of the results. The figure shows that by using a constant number of jobs in the look-ahead horizon, better results and solution distributions were achieved. The figure also denotes that better results were achieved if not a too large

Table 8.6: Results obtained by DRs with look-ahead and static terminal nodes calculated based on unreleased jobs outside the look-ahead horizon

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	12.08	13.48	17.59	≈	3	11.42	12.10	15.30	+
0.05	12.11	13.65	15.75	≈	5	11.10	11.95	15.22	+
0.1	11.81	12.98	14.44	+	10	**10.90**	12.24	**14.11**	+
0.2	11.86	12.39	14.94	+	20	11.10	**11.64**	14.54	+
0.5	**11.02**	**12.24**	16.48	+	50	11.04	11.86	14.15	+
1	11.10	12.33	**14.27**	+	100	10.95	12.35	14.51	+
DGP	12.96	13.60	14.62			12.96	13.60	14.62	

look-ahead horizon was used, which is consistent with the previous observations about the calculation method of static nodes. However, for certain parameter values, the method obtained several large outlier solutions.

Through the previously denoted results the influence of the look-ahead parameters for a concrete set of static terminal nodes was analysed. Now the look-ahead parameter will be fixed to a certain value, and the other 20 combinations of static terminal nodes will be tested to analyse if the results can further be improved by using a different combination of static nodes. The experiments were executed with the second calculation method, in which the static nodes are calculated only based on unreleased jobs outside the look-ahead horizon, since GP achieved better results for this calculation methods. Although the overall best result was achieved when using 10 jobs in the lookahead horizon, this parameter will not be selected since it it did not achieve a good median value. Therefore, a parameter value of 20 jobs in the look-ahead horizon will be selected, since it achieved the best median value, and the second best minimum value.

Table 8.7 represents the results obtained when using different static terminals node combinations with look-ahead. The index in the first column denotes the considered static terminal combination from Table 8.3. The results in the table show that the performance of DRs with look-ahead depends heavily on the static terminal set which is used. Although for all the combinations of static terminal nodes the DRs significantly outperformed DRs designed by DGP, not even one combination achieved better results than look-ahead with the same parameter values, but without the static terminals. It should be noted that none of the other 20 tested static terminal node combinations achieved a better median value than the one which was selected for optimising the look-ahead parameters (denoted in the table with the index 21). For all static terminal node combinations quite similar minimum values were achieved, which means that all the se-

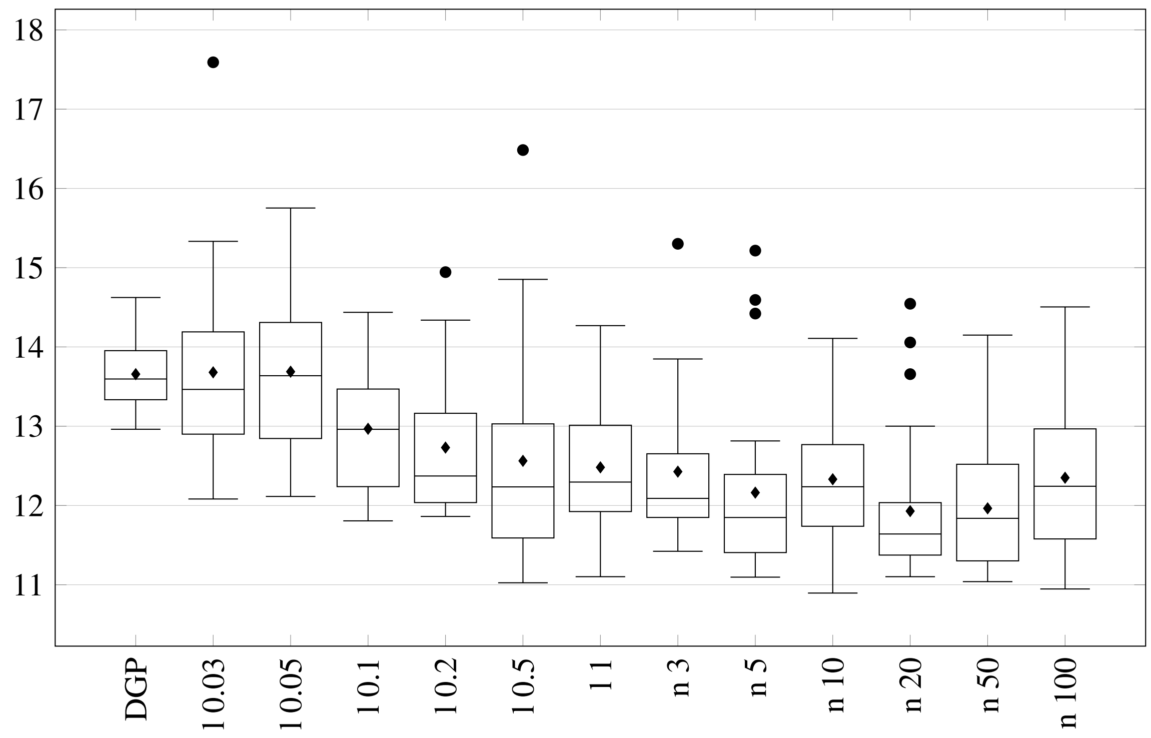


Figure 8.4: Box plot representation of the results obtained by DRs with look-ahead and static terminal nodes calculated based on unreleased jobs outside the look-ahead horizon

lected terminal node combinations are similarly expressive. However, much larger differences in the median and maximum values can be observed when using the different node combinations. In most cases, better median values were achieved by experiments where a smaller set of additional static terminal nodes was used.

Figure 8.5 represents the box plot for the different combinations of static nodes with look-ahead. In the box plot the result for the DRs which use look-ahead with the same parameter value, but without static terminals, is also included and denoted as *n-20*. The figure denotes that by using static terminals the entire procedure becomes more unstable. This is evident by the great dispersion of the results, and the many outlier solutions which were obtained by the method. In addition, the outlier solutions obtained by the method mostly achieved objective values that were much worse than the maximum values obtained by DGP. Nevertheless, this did not have an influence on the possibility of the method to outperform the results achieved by DGP.

Based on all the results and observations in this section, it can be concluded that the combination of look-ahead with static terminal nodes does not lead to significant improvements in the results. Neither by adjusting the look-ahead parameters, nor by using different static node combinations was it possible to achieve improvements over DRs which only use lookahead. The reason for this is probably due to the fact that look-ahead already enables the DRs to consider the important jobs which arrive in the future, and therefore there seems to be no need to provide additional information about future jobs through the static terminal nodes.

Table 8.7: Results obtained by using DRs with a look-ahead horizon 20 jobs and different combinations of static terminal nodes

Static terminal combination	Min	Med	Max	Stat. diff.
1	11.10	12.20	18.52	$+$
2	11.28	11.99	16.68	$+$
3	11.25	12.25	15.29	$+$
4	11.26	12.45	16.44	$+$
5	11.22	12.30	15.10	$+$
6	11.24	12.22	15.35	$+$
7	11.10	12.51	19.19	$+$
8	11.29	12.67	17.59	$+$
9	11.01	12.61	16.70	$+$
10	11.02	12.12	15.53	$+$
11	11.28	12.24	15.96	$+$
12	11.03	12.33	16.98	$+$
13	11.14	12.00	18.24	$+$
14	11.09	12.08	15.19	$+$
15	**11.00**	12.25	13.27	$+$
16	11.02	11.94	**13.98**	$+$
17	11.09	12.50	17.54	$+$
18	11.20	**11.64**	14.99	$+$
19	11.07	11.94	18.29	$+$
20	11.20	11.81	15.66	$+$
21	11.10	**11.64**	14.54	$+$
DGP	12.96	13.60	14.62	

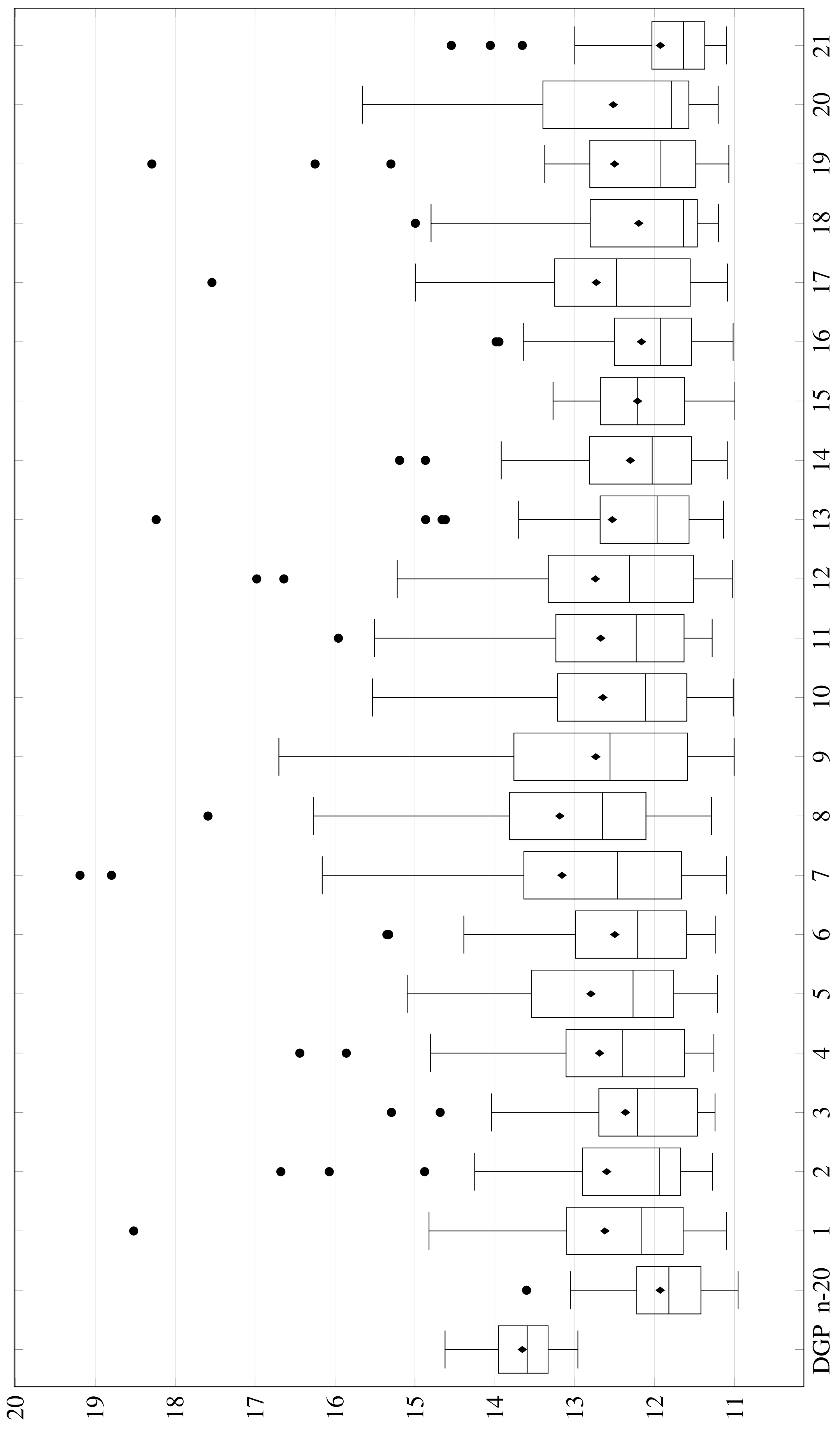


Figure 8.5: Box plot representation of the results obtained by DRs with look-ahead and different static node combinations

8.2.4 Results obtained by IDRs

In this section the results for IDRs with different combinations of IDR nodes will be presented. Since nine IDR nodes were proposed, testing all possible combinations of those nodes would be too time consuming. For that reason, the same destructive and constructive heuristics, which were used for creating the sets of static terminal nodes, will also be used here to create sets of IDR nodes. Since the number of combinations tested even by those two heuristics is quite substantial, only the ten combinations which achieved the best minimum values were selected.

Table 8.8 represents the results achieved by the ten best combinations of IDR nodes. The first seven combinations were obtained by using the constructive heuristic, while the remaining three combinations were obtained by using the destructive heuristic. The table shows that IDRs, which used the node combinations generated by the constructive heuristic, achieved significantly better results than DGP. On the other hand, for the three remaining node sets created by the destructive heuristic, there was no significant difference between the results achieved by IDRs and DGP. The table also shows that better results were achieved when using smaller sets of IDR nodes, usually between one and three nodes. Therefore, it is evident that for IDRs to work well, not many additional nodes are needed, but rather it is important to select those which represent useful information. The experiments have shown that for optimising the *Twt* criterion, the best results are achieved when using nodes which contain information about the tardiness of the jobs. The IDRs usually achieved the best results when using the *INDTARD*, *INDWTARD*, and *NLATE* nodes. The first two nodes are important since they denote the tardiness and weighted tardiness for each of the jobs, while the third node gives a notion about the number of jobs which were late in the previous schedule. Nodes that do not contain tardiness information, like *FLOWTIME* or *JOBFINISH*, do not seem to be very useful. Furthermore, the results also demonstrate that by using the *LATENESS* and *TARDINESS* nodes, which represent the tardiness and lateness of the entire schedule, IDRs usually do not achieve good results. Therefore, nodes which provide information about the tardiness of individual jobs lead to much better performance of IDRs.

Figure 8.6 shows the box plot representation of IDRs for the different combinations of IDR nodes. The figure shows that IDRs achieved more dispersed results than DGP, but most of these results were of good quality, which leads to good average and median values of the results. Although the IDRs generated by using node sets constructed by the destructive heuristic obtain better minimum values than DGP, the distribution of most solutions is similar to the one obtained by DGP, and therefore there is no significant difference between the two methods. On the other hand, when using node combinations generated by the constructive heuristic, IDRs achieved quite good solution distributions, especially for node sets 2 and 3.

Table 8.8: Results obtained by IDRs with various IDR node combinations

	IDR node combinations	Min	Med	Max	Stat. diff.
1	*INDTARD*	12.09	13.19	14.77	$+$
2	*INDWTARD*	12.09	**13.07**	14.40	$+$
3	*INDWTARD, NLATE*	11.87	13.08	**13.94**	$+$
4	*INDTARD, INDWTARD, NLATE*	**11.82**	13.18	14.39	$+$
5	*FLOWTIME, INDTARD, INDWTARD, NLATE*	12.03	13.37	15.05	$+$
6	*INDLATE, INDTARD, INDWTARD, NLATE*	12.01	13.23	14.30	$+$
7	*INDWTARD, NLATE, INDTARD, INDLATE, LATE, TARDINESS*	12.06	13.21	14.51	$+$
8	*FLOWTIME, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	12.08	13.60	18.73	$\approx$
9	*FLOWTIME, INDLATE, INDTARD, INDWTARD, LATE, LATENESS, NLATE, TARDINESS*	11.90	13.53	15.02	$\approx$
10	*FLOWTIME, INDLATE, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	12.06	13.56	15.09	$\approx$
	DGP	12.96	13.60	14.62	

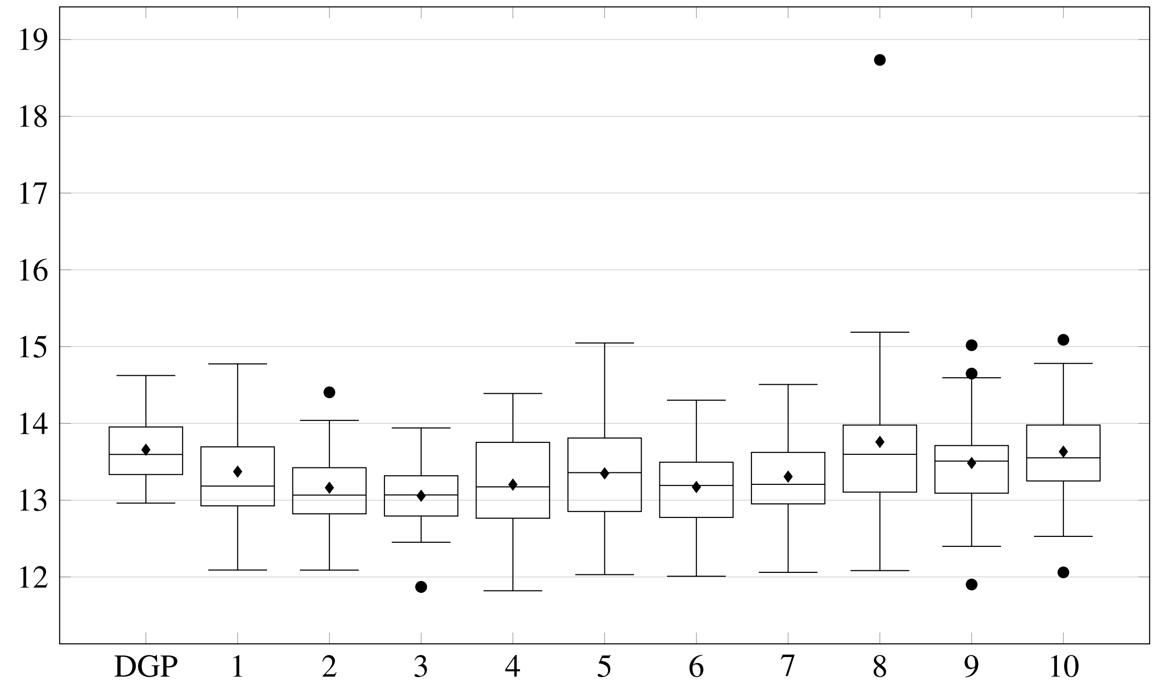


Figure 8.6: Box plot representation of the results obtained by IDRs for various combinations of IDR nodes

8.2.5 Results obtained by IDRs with static terminal nodes

In this section IDRs will be combined with static terminal nodes to determine whether combining these two approaches can lead to improved results, when compared to the results obtained by using each approach individually. First, the influence of the different IDR node combinations will be tested when the best combination of static terminal nodes is used (the one denoted with index 21 in Table 8.3). After that, the set of IDR nodes will be fixed to the best combination and the procedure will be tested with different combinations of static terminal nodes.

Table 8.9 represents the results achieved by the combination of IDRs and static terminal nodes for different IDR node combinations. The experiments which achieve significantly better results than DRs that use only the best combination of static terminal nodes are denoted with a grey background. On the other hand, experiments which perform significantly better than the IDRs with the same node combination will be underlined. The results show that although the procedure outperformed the results achieved by DGP in most cases, it struggled to perform better than either DRs with static terminal nodes or IDRs on their own. Only for experiment 4 the combination of IDRs and static terminal nodes significantly outperformed the best results obtained by using only static terminal nodes, which is hardly enough to justify the combination of these two methods.

Figure 8.7 shows the box plot representation of the results. The box plot denotes that for all IDR node combinations the achieved solution distributions are mostly similar. As in the previous experiments, better results are usually obtained by small sets of nodes, since the results

Table 8.9: Results obtained by IDRs with static terminal nodes and different combinations of IDR nodes

	IDR node combination	Min	Med	Max	Stat. diff.
1	*INDTARD*	**11.89**	13.16	14.79	$+$
2	*INDWTARD*	12.12	13.31	14.45	$+$
3	*INDWTARD, NLATE*	11.93	13.16	14.55	$+$
4	*INDTARD, INDWTARD, NLATE*	12.11	**12.88**	14.31	$+$
5	*FLOWTIME, INDTARD, INDWTARD, NLATE*	12.07	13.17	14.45	$+$
6	*INDLATE, INDTARD, INDWTARD, NLATE*	12.26	13.28	14.98	$+$
7	*INDWTARD, NLATE, INDTARD, INDLATE, LATE, TARDINESS*	12.07	13.37	14.69	$+$
8	*FLOWTIME, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	12.47	13.38	14.48	$+$
9	*FLOWTIME, INDLATE, INDTARD, INDWTARD, LATE, LATENESS, NLATE, TARDINESS*	12.25	13.36	14.66	$\approx$
10	*FLOWTIME, INDLATE, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	12.26	13.33	**14.25**	$+$
	DGP	12.96	13.60	14.62	

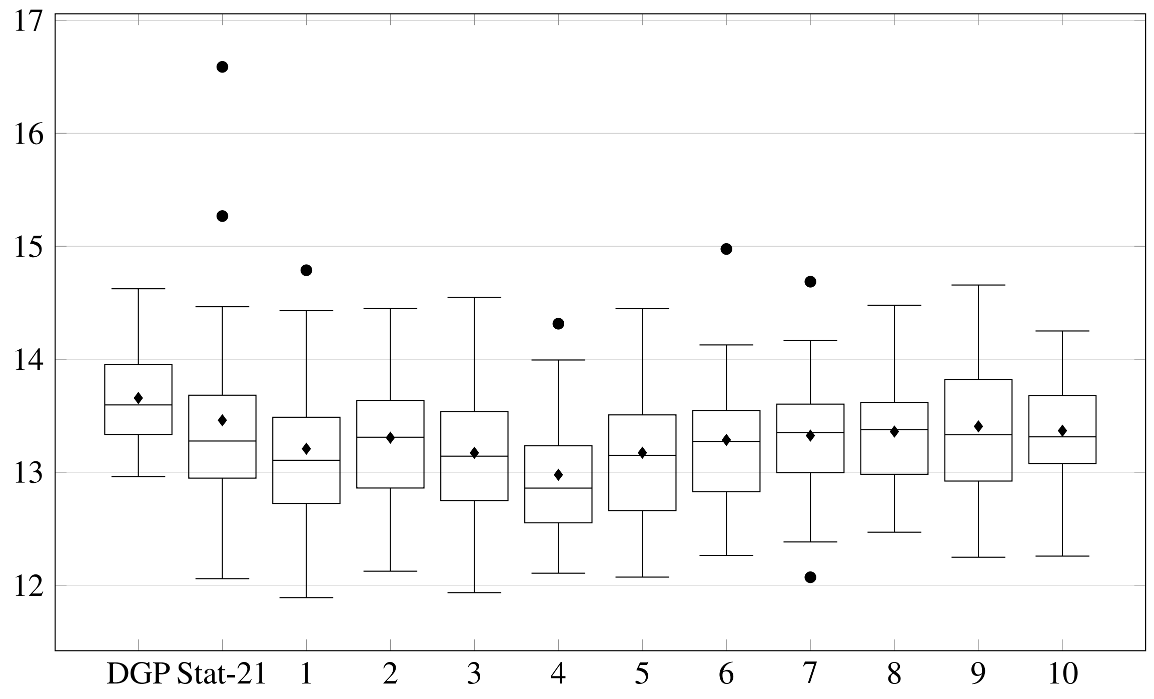


Figure 8.7: Box plot representation of the results obtained by IDRs with static terminal nodes and different combinations of IDR nodes

slowly deteriorate as the number of nodes increases. This can be seen from both the achieved minimum and median values of the results. In the figure the best results achieved by DRs using static terminal nodes are additionally included and denoted as *Stat-21*. These results additionally illustrate that the combination of static nodes and IDRs is unable to achieve improvements over the best results achieved by DRs using only static terminal nodes.

Table 8.10 represents the results achieved by the combination of IDRs and static terminal nodes, but for different combinations of static nodes. For these experiments the IDR node combination denoted with index 4 in Table 8.8 was used. The results which are significantly better than those of the best result achieved by IDRs are underlined. On the other hand, the results which are significantly better than those of DRs with the same static node combinations are denoted with a grey background. The results show that the combination of IDRs and static terminals performed significantly better than DGP. The combination has even shown to significantly outperform, for certain static node combinations, DRs which use only static terminal nodes. However, this usually occurred for node combinations where DRs with static terminal nodes did not perform well. Therefore, by using IDRs it was easier to achieve better results for those terminal node combinations. On the other hand, the combination of IDRs and static terminal nodes did not achieve significantly better results when compared to the best result achieved by IDRs. This just confirms the previous observations, which have shown that the addition of static terminal nodes does not significantly improve the results of IDRs.

Figure 8.8 shows the box plot representation of the results. The figure demonstrates that

Table 8.10: Results obtained by IDRs and different combinations of static terminal nodes

Static terminal node combination	Min	Med	Max	Stat. diff.
1	12.03	13.26	14.57	+
2	12.03	13.29	14.58	+
9	12.13	13.37	14.80	+
10	**11.91**	13.10	14.45	+
12	12.31	13.28	14.34	+
17	12.35	13.30	**14.13**	+
18	12.33	13.23	14.40	+
19	12.25	13.20	14.31	+
20	12.24	13.06	15.28	+
21	12.11	**12.88**	14.31	+
DGP	12.96	13.60	14.62	

all combinations of static nodes achieve quite similar distributions of results. The best result achieved by IDRs (denoted as *IDR-4*) is also included in the figure to denote that it achieved similar solution distributions as most combinations of IDRs and static nodes. Therefore, the plot demonstrates that no significant improvements are achieved by adding static terminal nodes to IDRs.

Based on all the results denoted in this section, it can be concluded that the combination of IDRs and static terminal nodes did not achieve results which would justify the use of this combination, since in most cases it even struggled to outperform results of IDRs without static terminals. Therefore, it is more beneficial to use IDRs on their own, without the addition static terminal nodes.

8.2.6 Results obtained by IDRs with look-ahead

In this section IDRs will be combined with look-ahead to analyse whether this combination can lead to improved results. First, the set of IDRs which achieves the best minimum value will be selected (the one with the index 4 in Table 8.8), while the look-ahead parameters will be optimised. After that, the look-ahead parameter will be fixed to a concrete value, while the different combinations of IDR nodes will be tested.

Table 8.11 represents the results achieved for IDRs with look-ahead, for different look-ahead parameters. To additionally denote whether there are improvements over IDRs and look-ahead

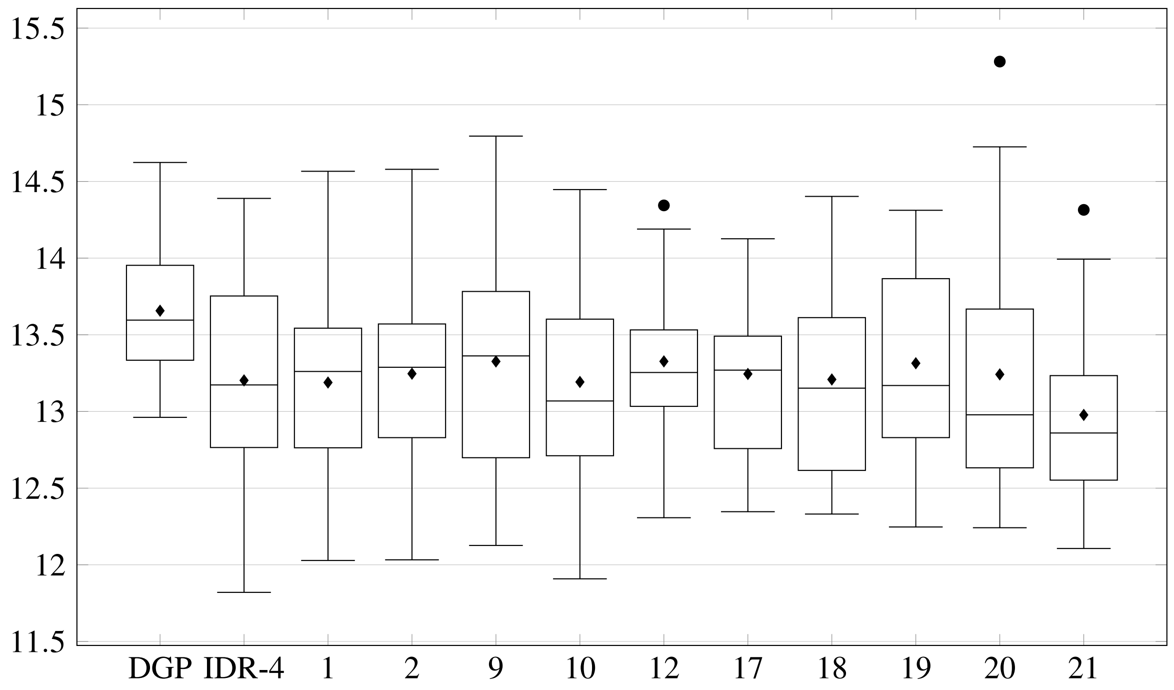


Figure 8.8: Box plot representation of the results obtained by IDRs and different combinations of static terminal nodes

on their own, those results which achieve significantly better results than those two methods will be specially denoted in the table. The experiments which achieve better results than the IDRs with the same combination of nodes will be underlined. On the other hand, the experiments which achieve significantly better results than look-ahead with the same parameters, but without using IDRs, will be denoted with a grey cell. The table shows that the combination of IDRs and look-ahead does not only outperform results of DGP, but also that it, in almost all cases, significantly outperforms the results of DRs with look-ahead, and IDRs on their own. This proves that the combination of these two methods is beneficial for increasing their performance. The reason why this combination performs well is due to the fact that DRs with look-ahead already perform quite well, however, since IDRs create the schedule several times by using information from the previously created schedules, they iteratively increase the performance of the schedule even further. Although the best results were achieved by using the largest values for the look-ahead parameters, extremely good results were achieved even for smaller values of the look-ahead parameters, for example for five and ten jobs in the look-ahead horizon.

Figure 8.9 shows the box plot representation of the results. The results for the best IDR node combination (denoted with the index 4 in Table 8.8) without look-ahead have also been included in the figure, and are denoted as IDR-4. The figure clearly demonstrates that for most of the look-ahead parameter values, much better solution distributions are achieved by IDRs with look-ahead than by DGP or IDRs on their own. Except for the few outlier solutions, all other solutions obtained by IDRs with look-ahead are better than the best solution found by

Table 8.11: Results obtained by IDRs and look-ahead when using different look-ahead parameter values

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	11.94	12.64	14.35	+	3	10.80	11.37	13.20	+
0.05	11.96	12.93	16.11	+	5	10.75	11.17	14.41	+
0.1	10.86	11.78	17.16	+	10	10.80	11.17	16.46	+
0.2	11.14	12.01	13.73	+	20	10.79	11.50	13.72	+
0.5	11.06	11.62	21.66	+	50	10.73	11.15	14.94	+
1	**10.63**	**11.54**	**12.95**	+	100	**10.53**	**11.10**	**12.44**	+
DGP	12.96	13.60	14.62			12.96	13.60	14.62	

DGP, when using a constant number of jobs in the look-ahead horizon. Therefore, this approach can easily obtain much better results than DGP.

Table 8.12 represents the results for the combination of IDRs with look-ahead for different combinations of IDR nodes. For these experiments five jobs in the look-ahead horizon were used. This parameter value was selected since it lead to the largest improvement when combining IDRs and look-ahead over the results obtained by using look-ahead on its own. In the table, the experiments which achieve significantly better results than IDRs with the same node combinations, but without look-ahead, are underlined. On the other hand, experiments which achieve significantly better results than look-ahead of the same parameters, but without IDRs, will be denoted with a grey cell. The results in the table show that the combination of IDRs and look-ahead consistently achieved better results than either DGP or IDRs. By comparing the results to those achieved by look-ahead, it can be noticed that for smaller combinations of IDR nodes better results are achieved when additionally using IDRs. However, for larger combinations there was no significant difference between using a combination of IDRs and look-ahead, and using only look-ahead. Such a result is expected since IDRs on their own did not achieve good results for larger combinations of IDR nodes. The experiments also show that neither of the other nine node combinations achieved a better median value than the combination of IDR nodes used for optimising the look-ahead parameters.

Figure 8.10 shows the box plot representation of the results. The results obtained by the look-ahead method with 5 jobs in the look-ahead horizon are also included in the figure, and denoted as *n-5*. The figure illustrates that the best solution distributions are achieved when using between two and four IDR nodes, while for a smaller or larger number of nodes the methods obtained more dispersed results. As the number of IDR nodes used in the primitive set grows,

Table 8.12: Results obtained by IDRs with look-ahead when using different IDR node combinations

	IDR node combinations	Min	Med	Max	Stat. diff.
1	*INDTARD*	10.67	11.43	13.14	+
2	*INDWTARD*	10.96	11.53	12.88	+
3	*INDWTARD, NLATE*	10.89	11.41	**12.36**	+
4	*INDTARD, INDWTARD, NLATE*	10.75	**11.17**	14.41	+
5	*FLOWTIME, INDTARD, INDWTARD, NLATE*	10.76	11.41	13.07	+
6	*INDLATE, INDTARD, INDWTARD, NLATE*	10.85	11.42	13.22	+
7	*INDWTARD, NLATE, INDTARD, INDLATE, LATE, TARDINESS*	10.73	11.52	13.38	+
8	*FLOWTIME, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	10.94	11.64	13.57	+
9	*FLOWTIME, INDLATE, INDTARD, INDWTARD, LATE, LATENESS, NLATE, TARDINESS*	10.71	11.71	13.77	+
10	*FLOWTIME, INDLATE, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	**10.66**	11.34	14.15	+
	DGP	12.96	13.60	14.62	

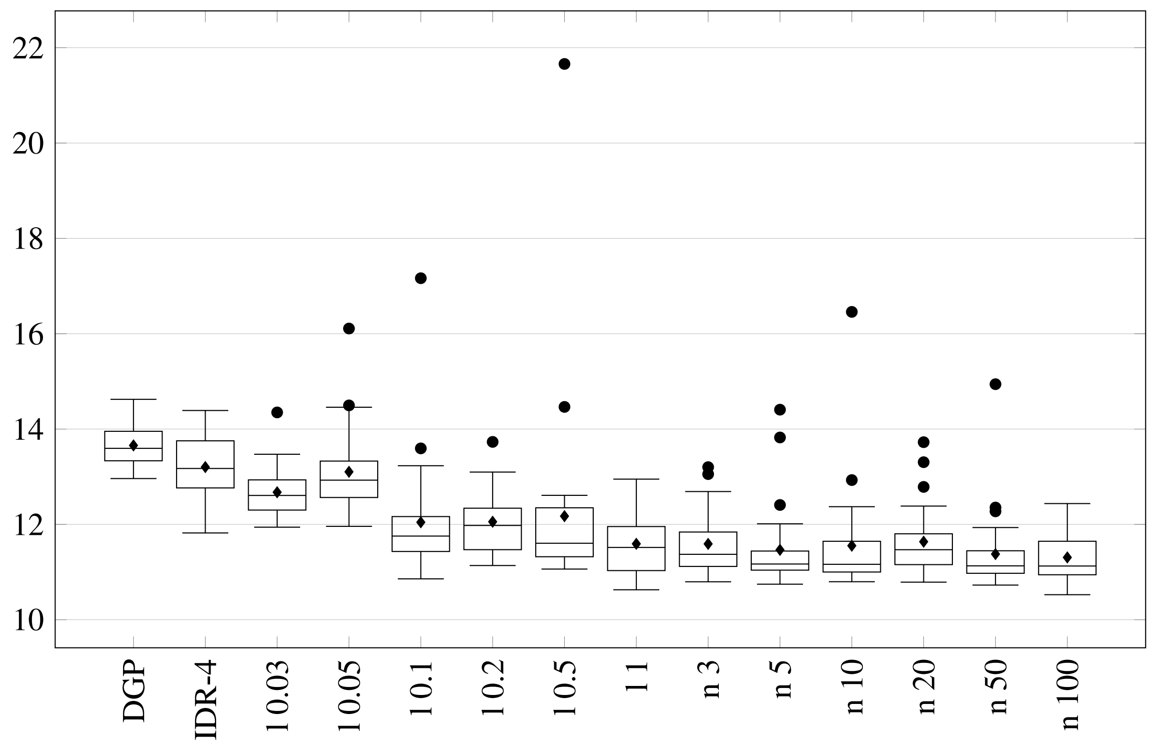


Figure 8.9: Box plot representation of the results obtained by IDRs with look-ahead when using different look-ahead parameter values

especially beyond the size of six nodes, the obtained results start to deteriorate. Therefore, it can be concluded that a combination of IDRs and look-ahead achieves good results when a small or moderate number of IDR nodes is used in the primitive set, whereas for a larger number of IDR nodes the results deteriorate.

8.2.7 Results obtained by IDRs with static terminals and look-ahead

This section will analyse whether the combination of IDRs, look-ahead and static terminals at the same time can lead to improved results. Through three experiment sets it will be analysed how different look-ahead parameters, IDR node combinations and static terminal node combinations influence the quality of the results. Through all the experiments the static nodes will be calculated based on all unreleased jobs outside the look-ahead horizon.

Table 8.13 represents the results achieved for IDRs with look-ahead and static terminal nodes. This table analyses the influence of the different look-ahead parameters on the quality of the results. The experiments used the set of static terminal nodes denoted with the index 21 in Table 8.3, and the set of IDR nodes with the index 4 from Table 8.8. The experiments which achieve significantly better results than look-ahead for the same parameter combination will be denoted with a grey cell. On the other hand, the results which achieve better results than look-ahead with static terminal nodes for the same parameter values will be underlined. The results denote that except for one experiment, all other experiments achieved significantly

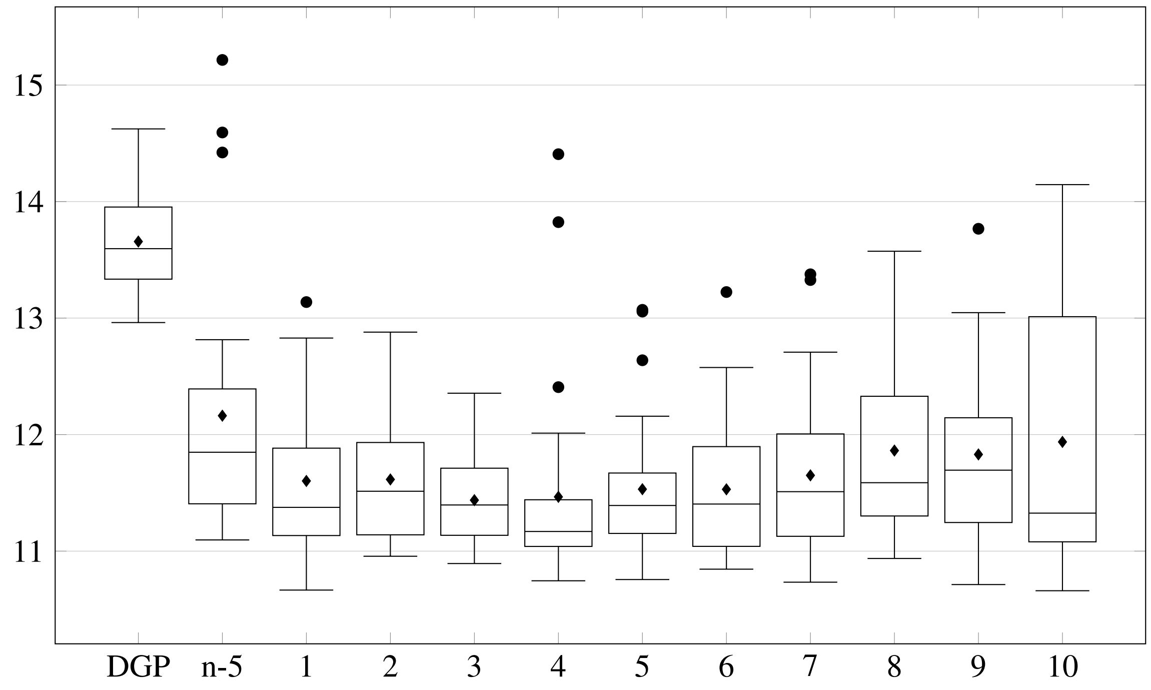


Figure 8.10: Box plot representation of the results obtained by IDRs with look-ahead when using different IDR node combinations

better results than DRs generated by DGP. Half of the experiments achieved better results than look-ahead on its own, however, no experiment achieved significantly better results than IDRs with look-ahead.

Figure 8.11 represents the box plot representation of the results. The figure denotes that look-ahead with a fixed number of jobs in the look-ahead horizon performs much better than when using smaller values for the look-ahead factor. In addition, when using the fixed number of jobs in the look-ahead horizon, better distributions can be achieved when using smaller parameter values. This is expected since with the growth of the number of jobs in the look-ahead horizon, the static nodes will hold less useful information. Therefore, with this approach a smaller value of the number of jobs in the look-ahead horizon should be used. The figure also includes the results achieved by the best combination of static terminal nodes and IDR nodes, which is denoted as *IDR-4 S-21* (the indices denote which IDR and static node combinations were used). The results show that by adding look-ahead to IDRs with static terminal nodes leads to improved results for most look-ahead parameter values.

Table 8.14 represents the influence of the different IDR node combinations on the results for IDRs with look-ahead and static nodes. These experiments use the static terminal node set denoted with the index 21 in Table 8.3, and 5 jobs in the look-ahead horizon. The underlined experiments in the table represent those for which significantly better results are achieved, when compared to the results obtained by the same IDR node combinations with the best set of static terminal nodes. The cells denoted in grey represent results in which the experiments significantly outperform IDRs with look-ahead for the same IDR node combinations, and the same

Table 8.13: Results obtained by IDRs with look-ahead and static terminals when using various look-ahead parameter values

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	11.92	12.56	13.98	+	3	10.87	11.77	13.75	+
0.05	11.86	13.62	15.05	≈	5	10.63	11.52	14.00	+
0.1	11.04	12.12	13.94	+	10	10.77	**11.42**	14.29	+
0.2	**11.58**	12.12	**13.68**	+	20	10.61	11.56	14.49	+
0.5	10.66	**11.61**	14.42	+	50	10.53	11.76	13.39	+
1	10.59	11.62	13.86	+	100	**10.52**	11.85	**13.07**	+
DGP	12.96	13.60	14.62			12.96	13.60	14.62	

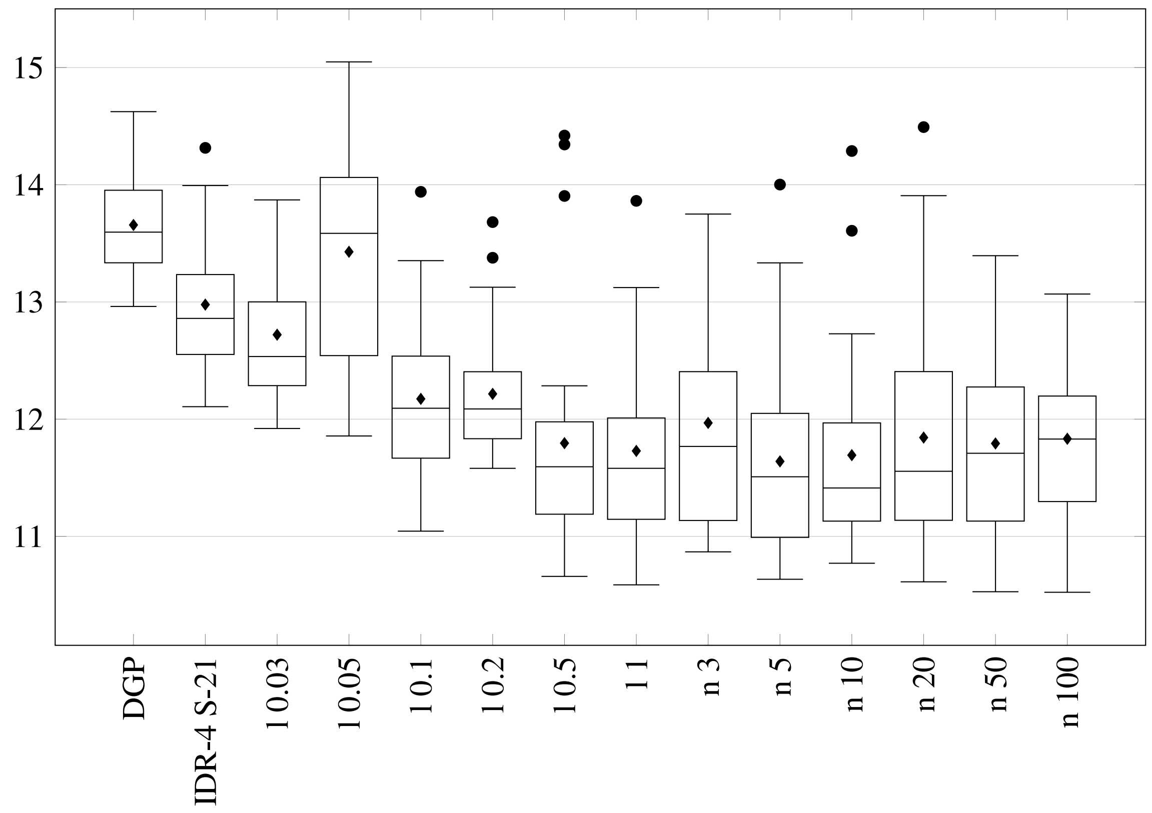


Figure 8.11: Box plot representation of the results obtained by IDRs with look-ahead and static terminals when using various look-ahead parameter values

Table 8.14: Results obtained by IDRs with look-ahead and static terminals when using various combinations of IDR nodes

	IDR node combination	Min	Med	Max	Stat. diff.
1	*INDTARD*	10.98	12.08	14.09	+
2	*INDWTARD*	10.89	11.39	14.07	+
3	*INDWTARD, NLATE*	10.82	**11.33**	13.46	+
4	*INDTARD, INDWTARD, NLATE*	**10.63**	11.52	14.00	+
5	*FLOWTIME, INDTARD, INDWTARD, NLATE*	10.74	11.61	**13.38**	+
6	*INDLATE, INDTARD, INDWTARD, NLATE*	10.86	11.98	15.23	+
7	*INDWTARD, NLATE, INDTARD, INDLATE, LATE, TARDINESS*	10.98	11.70	14.97	+
8	*FLOWTIME, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	10.75	11.66	14.03	+
9	*FLOWTIME, INDLATE, INDTARD, INDWTARD, LATE, LATENESS, NLATE, TARDINESS*	10.80	11.62	14.96	+
10	*FLOWTIME, INDLATE, INDTARD, INDWTARD, JOBFINISH, LATE, LATENESS, NLATE, TARDINESS*	10.86	11.90	13.50	+
	DGP	12.96	13.60	14.62	

look-ahead parameter value. The results denote that this combination achieved significantly better results than DRs evolved by DGP, or when using static terminal nodes with IDRs. However, the achieved results are inferior to the results obtained by IDRs with look-ahead for the same IDR combinations. Again, the best results were achieved when smaller sets of IDR nodes were used.

Figure 8.12 shows the box plot representation of the results. The figure denotes that better solution distributions are mostly achieved when using a smaller number of IDR nodes. When compared to DGP, the method has shown to consistently achieve superior results. The figure also includes the result obtained by look-ahead with static terminal nodes, denoted as *n-5 S-21*, which used the same parameters as the other experiments in the figure, just without the IDR nodes. These results show that by adding IDRs to look-ahead with static terminal nodes leads to better median values for most IDR node combinations, especially for combinations 2, 3, and 4.

Table 8.15 represents the influence of the different static terminal node combinations on

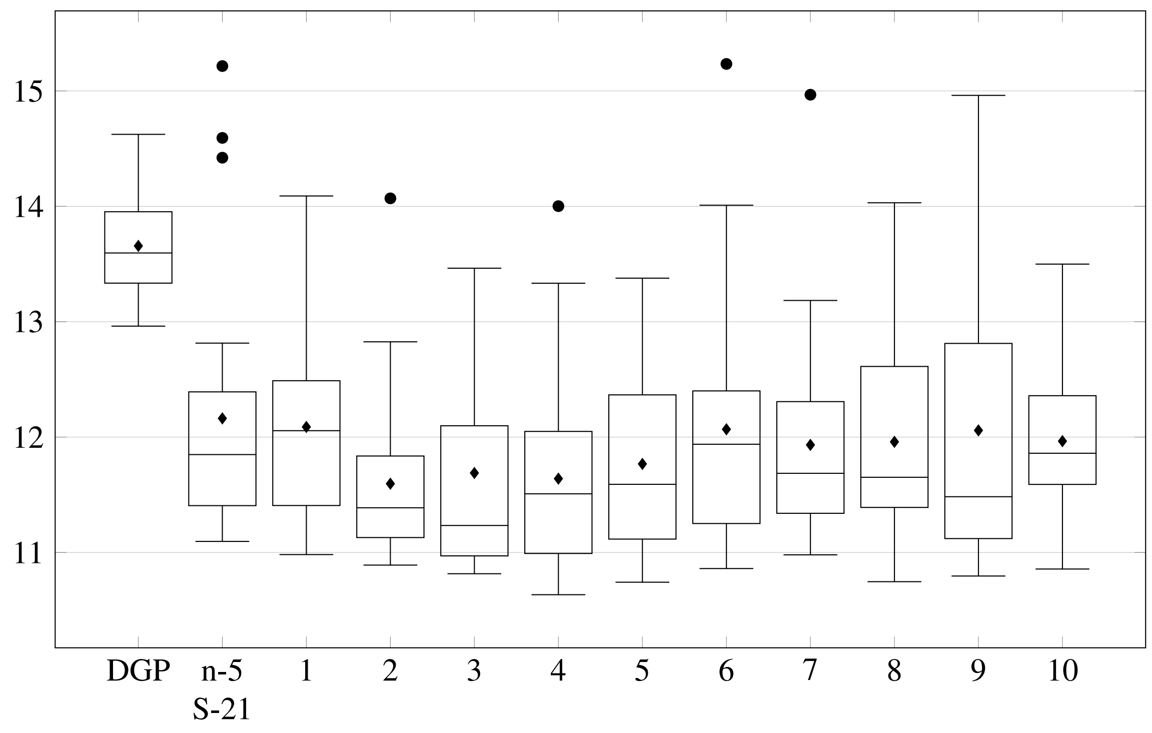


Figure 8.12: Box plot representation of the results obtained by look-ahead with IDRs, when using different IDR node combinations

IDRs with look-ahead and static nodes. The experiments will use the IDR node combination denoted with the index 4 in Table 8.8, and five jobs in the look-ahead horizon. The grey cells denote experiments for which significantly better results were achieved than those of DRs with look-ahead and the same static node combinations. On the other hand, the underlined experiments denote those which achieved better results than IDRs with the same static node combinations. The table shows that all experiments achieved significantly better results than GP which uses the same static node combinations. Furthermore, for each combination of static nodes the experiments outperform DGP and and IDRs with static terminal nodes. Therefore, the inclusion of look-ahead to IDRs with static nodes leads to improved results. On the other hand, the addition of IDRs to look-ahead with static nodes also leads to improved results, but only for certain static node combinations.

Figure 8.13 shows the box plot representation of the results. The figure denotes that the chosen combination of static nodes influences the obtained results. This is evident from the fact that for certain node combinations the approach achieved quite dispersed results, which consequentially lead to worse median values for those node combinations. This is especially evident for static node combinations denoted with indices 12 and 18. On the other hand, for static node combination like 1, 17, and 19 the method achieved the best solution distributions. The figure also includes the results achieved by IDRs with look-ahead for the same parameter values which were used by the other experiments. This result is denoted as *IDR-4 n-5* in the figure. The experiments show that by adding static terminal nodes to look-ahead with IDRs does

Table 8.15: Results obtained by IDRs with look-ahead and static terminal nodes, when using various combinations of static terminal nodes

Static terminal combination	Min	Med	Max	Stat. diff.
1	10.71	11.53	**12.98**	$+$
2	10.71	11.85	13.89	$+$
9	10.87	11.67	14.35	$+$
10	**10.63**	11.61	13.98	$+$
12	10.67	11.76	19.75	$+$
17	10.86	**11.32**	19.22	$+$
18	10.87	12.03	14.10	$+$
19	10.78	11.54	13.46	$+$
20	**10.63**	11.56	14.11	$+$
21	**10.63**	11.52	14.00	$+$
DGP	12.96	13.60	14.62	

not increase the performance of the method, regardless of the static node combination which was used.

The experiments in this section have demonstrated that the combination of IDRs, static terminal nodes, and look-ahead can achieve quite good results when compared to DGP. However, the procedure did not show any improvements when compared to the results of IDRs with look-ahead. Therefore, there is little benefit of using the combination of all three methods.

8.2.8 Results obtained by the rollout algorithm

In this section the results for the rollout algorithm by using DRs generated by DGP will be presented. As denoted previously, since the rollout algorithm always achieves significantly better results than DGP, the statistical tests will check whether the rollout algorithm achieves significantly better results than a GA. The results of these tests will be denoted in the *stat. diff.* column.

Table 8.16 represents the results achieved by the rollout algorithm when using DRs generated by DGP. It is evident that the performance of the rollout algorithm depends heavily on whether the rollout factor or number of jobs in the rollout horizon will be used. The rollout algorithm achieved much better performance when a constant number of jobs in the rollout horizon was used. This is supported by the fact that by using a constant number of jobs in the rollout horizon, the rollout algorithm can in most cases perform equally well or better than the

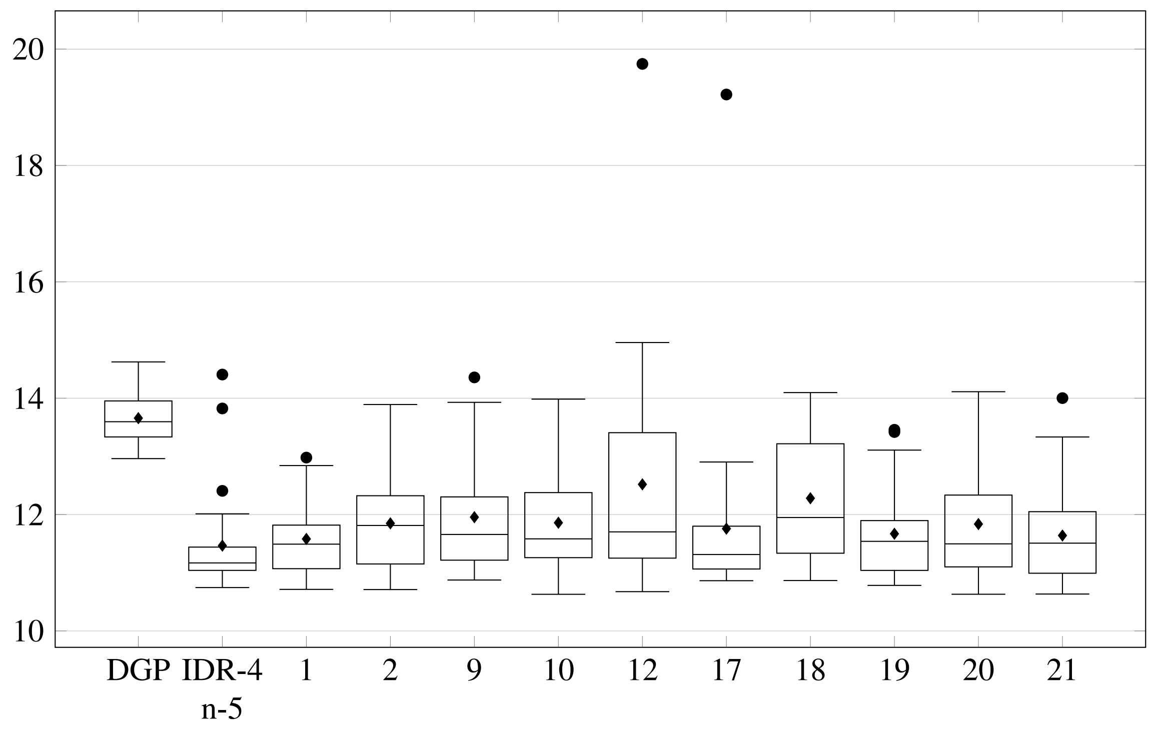


Figure 8.13: Box plot representation of the results obtained by IDRs with static terminal nodes and look-ahead, when using various combinations of static terminal nodes

GA. On the other hand, the rollout algorithm outperforms the GA only when the largest value of the rollout factor is used. An additional thing which can be observed from the experiments is that after a certain value for the number of jobs in the rollout horizon the results do not improve any further, which means that considering more jobs in the rollout horizon is not beneficial. Such a behaviour is expected, since it is highly unlikely that at the current scheduling decision a job which is released far in the future would be scheduled. Therefore, it is useful to consider only a smaller number of jobs which arrive the closest to the current decision time.

Figure 8.14 shows the box plot representation of the results. The figure illustrates how different values of the rollout parameters influence the performance of the rollout algorithm. The experiments which use the rollout factor are denoted with "r" and the value of the parameter, while the experiments which use a fixed number of jobs in the rollout horizon are denoted with "n" and the value of the parameter. It is interesting to note that the rollout algorithm usually achieved less dispersed results than the GA. In addition, the figure also denotes that good results are achieved even by small values for the number of jobs in the rollout horizon, and that those results slowly improve as the value of the parameter increases until the value of 20 is reached, after which no further improvement is achieved. On the other hand, when using smaller values for the rollout factor, the results that are achieved are quite bad compared to the GA, and only for larger values does the rollout algorithm achieve equally good results as the GA. The only disadvantage of the rollout algorithm, especially when using a constant number of jobs in the rollout horizon, is that it obtained more outlier values than the GA.

Table 8.16: Results obtained by the rollout algorithm, when using different rollout parameter values and DRs generated by DGP

Rollout factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0	10.51	10.87	11.21	–	0	10.51	10.87	11.21	–
0.03	10.07	10.63	10.98	–	3	9.956	10.37	**10.65**	–
0.05	10.16	10.56	11.07	–	5	9.883	10.24	11.18	≈
0.1	10.07	10.41	**10.77**	–	10	9.956	10.22	11.09	≈
0.2	9.956	10.36	10.79	≈	20	**9.790**	**10.08**	10.88	+
0.5	10.05	12.27	11.12	≈	50	**9.790**	**10.08**	10.88	+
1	**9.790**	**10.08**	10.88	+	100	**9.790**	**10.08**	10.88	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

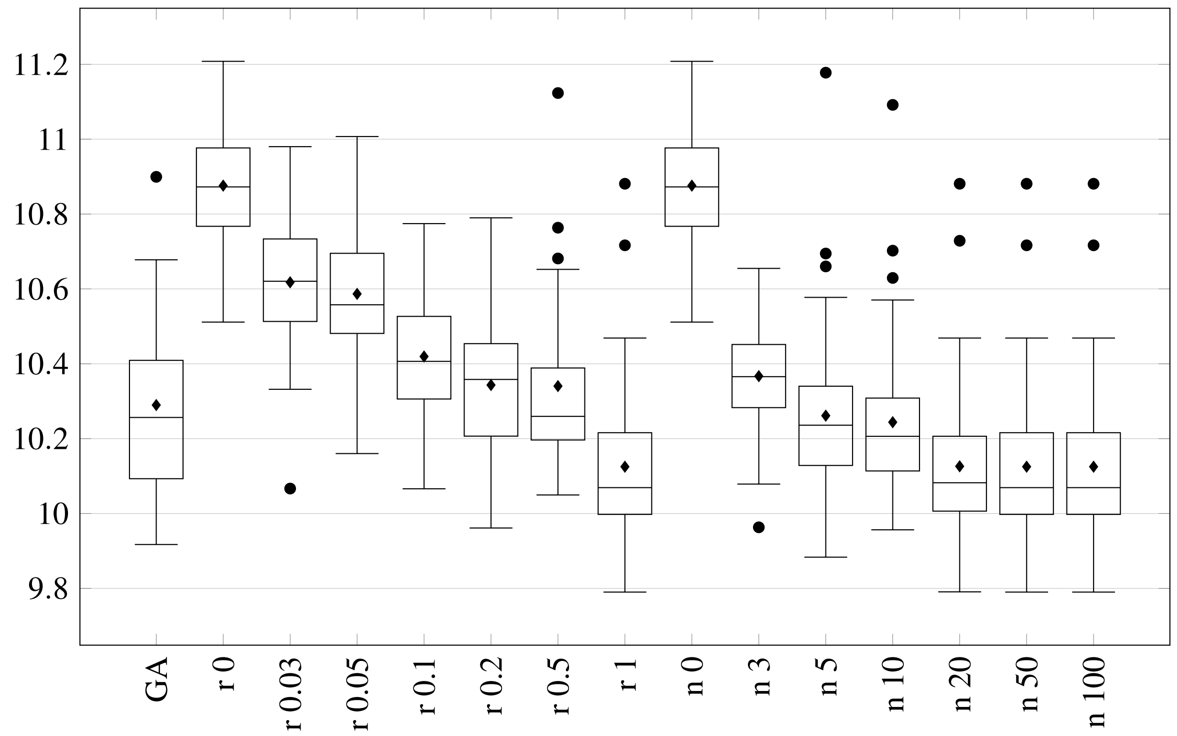


Figure 8.14: Box plot representation of the results obtained by the rollout algorithm, when using different rollout parameter values and DRs generated by DGP

Based on the outlined results, it is evident that the rollout algorithm represents a viable alternative to the GA, since depending on the parameter value it can achieve equally good or even better results.

8.2.9 Results obtained by the rollout algorithm with static terminals

This section will analyse whether combining the rollout algorithm with DRs which use additional static terminal nodes can lead to improved results.

Table 8.17 represents the results achieved by the rollout algorithm when using DRs with static terminal nodes. For the experiments the static node combination denoted with index 21 in Table 8.3 was chosen, since it achieved the overall best result out of all the tester terminal node combinations. For the experiments denoted in grey the rollout algorithm achieved significantly better results by using DRs that contain static terminals, instead of using DRs generated by DGP. The addition of static nodes has more benefit when using the rollout factor, where for all but one experiment the addition of static terminal nodes leads to statistically better results. Even when comparing the achieved results to those of the GA can it be seen that for more parameter values the rollout algorithm is now able to perform equally well or better than the GA. When using the fixed number of jobs in the rollout horizon, the rollout algorithm achieved significantly better results for three, mostly smaller parameter values. Therefore, the addition of static terminal nodes has shown to be beneficial when using the rollout algorithm. The reason for this is probably due to the fact that DRs which use static terminal nodes can give a better approximation of the rest of the schedule, and therefore guide the rollout algorithm to improved solutions.

Figure 8.15 represents the box plot representation of the results. The figure shows that when using a constant number of jobs in the rollout horizon there is a quite small difference between the results achieved for the different values of the parameter. It can be noticed that for smaller parameter values the rollout algorithm can achieve some quite bad results. However, this happens for only one or two DRs, and these results have shown to represent outliers when the entirety of all results are considered.

In the previous experiments the rollout algorithm used DRs which were all generated by using the same combination of static terminal nodes. However, it is interesting to test whether having DRs created by different node combinations leads to a significant difference in the results. For that purpose, 30 DRs which achieve the best results, but are generated by different static node combinations, were chosen. These selected DRs are then used by the rollout algorithm to generate the schedules. Table 8.18 represents the results for the rollout algorithm when using such a combination of DRs. The experiments denoted in grey achieve significantly better results than the rollout algorithm, which uses DRs generated by DGP for the same parameter values. On the other hand, the underlined experiments achieve significantly better results than

Table 8.17: Results obtained by the rollout algorithm with static terminal nodes, when using different rollout parameter values and DRs generated by using the best static terminal node combination

Rollout factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0	10.35	10.74	11.69	–	0	10.35	10.74	11.69	–
0.03	10.18	10.50	11.48	–	3	9.907	10.15	11.29	≈
0.05	10.11	10.47	11.40	–	5	9.806	10.17	11.31	+
0.1	10.08	10.29	11.53	≈	10	**9.765**	10.12	11.08	+
0.2	9.994	10.21	11.18	≈	20	9.769	**10.08**	10.74	+
0.5	9.828	10.13	**10.48**	+	50	9.770	10.10	**10.61**	+
1	**9.770**	**10.10**	10.61	+	100	9.770	10.10	**10.61**	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

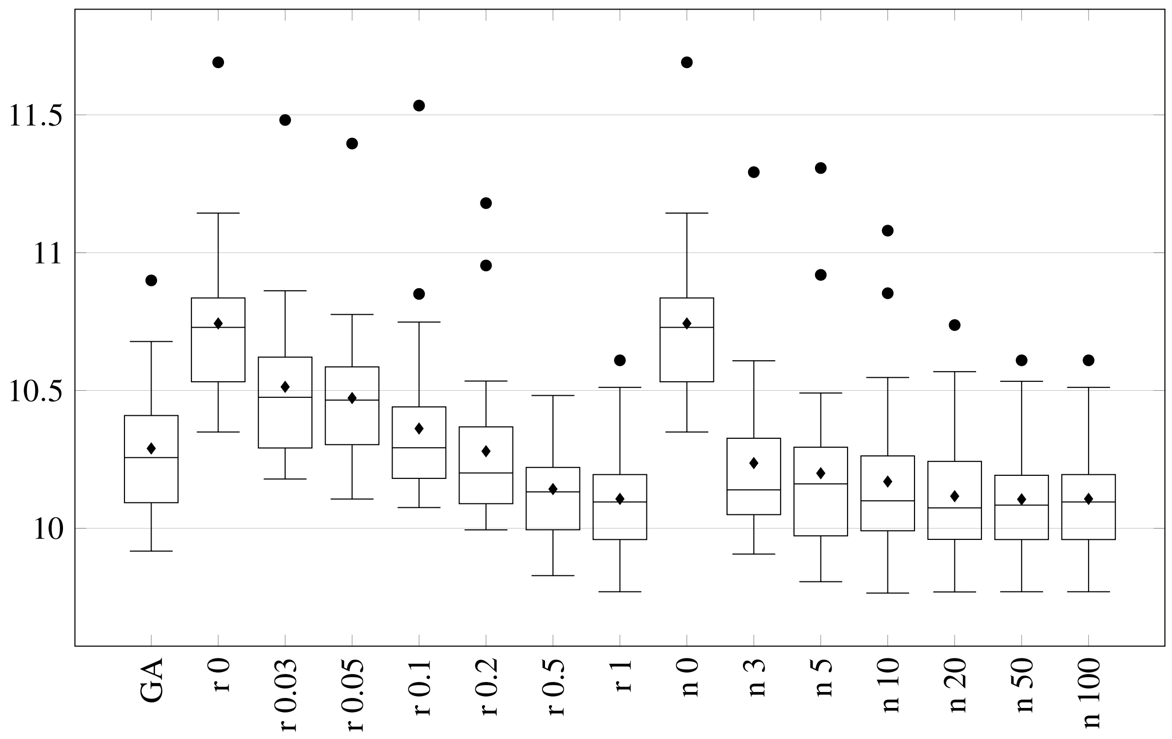


Figure 8.15: Box plot representation of the results obtained by the rollout algorithm, when using different rollout parameter values and DRs generated by using the best static terminal node combination

Table 8.18: Results obtained by the rollout algorithm with static terminal nodes, when using various rollout parameter values and DRs generated by using different static node combinations

Rollout factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0	10.27	10.48	10.89	−	0	10.27	10.48	10.89	−
0.03	10.09	10.34	10.77	≈	3	9.911	10.08	10.65	+
0.05	10.02	10.34	10.78	≈	5	9.786	10.04	10.66	+
0.1	9.987	10.22	10.68	≈	10	9.832	10.04	10.77	+
0.2	9.976	10.14	10.75	+	20	**9.683**	10.04	10.72	+
0.5	9.862	10.05	10.65	+	50	9.729	**10.03**	**10.55**	+
1	**9.719**	**10.03**	**10.64**	+	100	9.719	**10.03**	10.64	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

the rollout algorithm which used only DRs generated by the best combination of static terminal nodes. The experiments show that by using a combination of DRs generated by different combinations of static terminal nodes, the rollout algorithm can in all cases, except for the smallest parameter value, achieve at least equally good results as the GA. When using the constant number of jobs in the rollout horizon, the rollout algorithm achieved significantly better results than the GA for all parameter values except 0. The experiments also achieved significantly better results than rollout with DRs generated by DGP in most of the cases. However, when compared to the results achieved by the rollout algorithm using DRs generated by the best combination of static terminals, it can be seen that significantly better results are achieved only for smaller parameter values, while for larger parameter values there is no significant improvement in the results.

Therefore, using DRs generated with various static node combinations leads to quite good results. However, this is more due to the fact that the very best DRs were used, than to the fact that they were generated by different static node combinations. In addition, it is much more time consuming to create such a set of DRs, since DRs need to be generated for various static node combinations, instead of just one. Since the improvements achieved when using such a combination of DRs by the rollout algorithm are quite small compared to the results achieved when using DRs generated by the very best combination of static nodes, there is no evident benefit of using such a combination of methods.

Figure 8.16 represents the box plot representation of the results. The figure denotes that the rollout algorithm achieves solutions which are, except for a few outliers, less dispersed than those obtained by the GA. In addition, the rollout algorithm achieved better results than

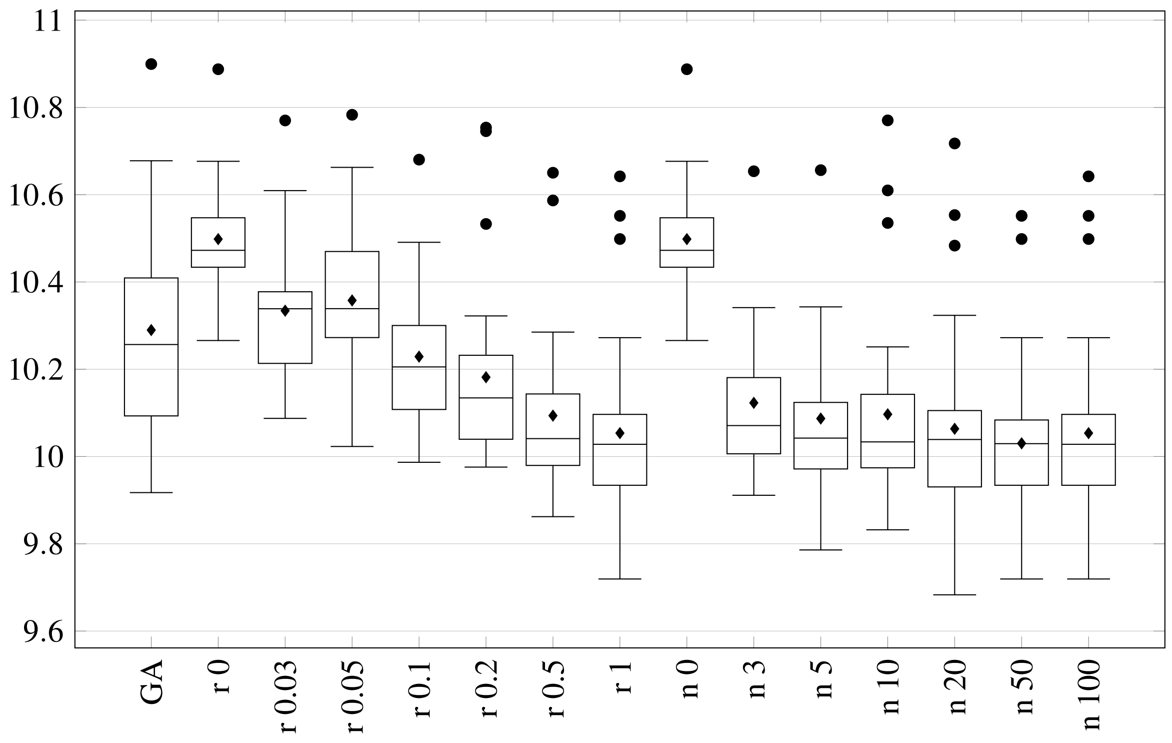


Figure 8.16: Box plot representation of the results obtained by the rollout algorithm, when using different rollout parameter values and DRs generated by using different static node combinations

the GA even when using a smaller number of jobs in the rollout horizon. Better results are also achieved when the rollout algorithm uses a constant number of jobs in the rollout horizon, instead of using the rollout factor.

Table 8.19 denotes the results achieved by the rollout algorithm when using DRs generated by using different combinations of static terminal nodes. In the experiments three jobs in the rollout horizon were used. This parameter value was chosen since in the previous experiments it was already demonstrated that good results were achieved for it, and that an increase in the parameter value did not lead to any significant improvements in the results. The cells denoted in grey represent experiments for which significantly better results were achieved than by using the rollout algorithm with DRs generated by DGP. The experiments show that for 16 out of the 21 tested combinations of static nodes significantly better results were achieved by the rollout algorithm when using static terminal nodes instead of DRs generated by DGP. While for the most static node combinations the rollout algorithm achieved the same results as the GA, for the combination denoted with the index 17 the rollout algorithm managed to outperform the GA. The results also demonstrate that for all combinations the rollout algorithm performed quite well.

Figure 8.17 shows the box plot representation for the results obtained by the rollout algorithm when using different static node combinations. The figure shows that the combination of nodes which is used by the rollout algorithm heavily influences the dispersion of the re-

Table 8.19: Results obtained by the rollout algorithm with static terminal nodes, when using DRs generated by different combinations of static terminal nodes

Static terminal combination	Min	Med	Max	Stat. diff.
1	9.952	10.24	10.88	≈
2	10.12	10.32	11.53	≈
3	9.976	10.31	11.16	≈
4	9.945	**10.14**	11.19	≈
5	9.963	10.25	11.18	≈
6	9.950	10.20	11.12	≈
7	9.994	10.17	10.81	≈
8	**9.874**	10.24	12.68	≈
9	10.01	10.25	12.83	≈
10	10.00	10.15	10.89	≈
11	10.03	10.19	12.63	≈
12	9.998	10.29	11.95	≈
13	9.987	10.16	10.75	≈
14	9.889	10.16	10.83	≈
15	9.952	10.22	10.92	≈
16	9.923	**10.14**	11.18	≈
17	9.891	10.15	11.91	+
18	9.908	10.15	11.01	≈
19	9.996	10.18	**10.74**	≈
20	9.918	10.19	10.96	≈
21	9.907	10.15	11.29	≈
GA	9.917	10.27	10.90	

sults. However, for certain node combinations, like those denoted with indices 16 and 17, less dispersed results are achieved. The figure also includes the results achieved by the rollout algorithm when using DRs generated by DGP for three jobs in the rollout horizon, which is denoted as *n-3*. For most static node combinations the rollout algorithm achieved a better performance when compared to the rollout algorithm which uses DRs generated by DGP.

Based on all the results presented in this section, it can be concluded that using DRs with static terminal nodes in the rollout algorithm leads to better results than by using DRs without additional static information.

8.2.10 Results obtained by the rollout algorithm with look-ahead

This section will analyse if combining the rollout algorithm with DRs that additionally use look-ahead can further improve the results. The first part of the section will analyse the influence of the rollout parameter values on the obtained results when a fixed look-ahead parameter value is used. After that, the experiments will test the influence of the look-ahead parameter values on the results when using a fixed rollout parameter value.

Table 8.20 represents the results achieved for different parameter values by the rollout algorithm when using DRs with look-ahead. For all the experiments ten jobs in the look-ahead horizon were used. The cells denoted in grey represent the experiments which achieved significantly better results than the rollout algorithm which used DRs generated by DGP. The table shows that for all parameter values the rollout algorithm achieves significantly better results when using DRs with look-ahead instead of DRs generated by DGP. Furthermore, when using a constant number of jobs in the rollout horizon, the rollout algorithm achieves significantly better results than the GA for all parameter values larger than 0. For the best parameter value the rollout algorithm with look-ahead achieves a better performance than the GA by 3.6% for the median value. Therefore, by using DRs with look-ahead good improvements in the results over the GA can be achieved.

Figure 8.18 shows the box plot representation of the results. The figure demonstrates that for a number of parameter values the rollout algorithm achieves a better distribution of solutions than the GA. This is especially evident when using a larger number of jobs in the rollout horizon. In addition, better results are again achieved when using a constant number of jobs in the look-ahead horizon instead of using the look-ahead parameter. Furthermore, when using the constant number of jobs in the rollout horizon, the results do not change drastically when using more than 10 jobs in the rollout horizon. Therefore, the procedure once again performs well without the need of considering too many jobs at each decision moment.

Table 8.21 represents the influence of different look-ahead parameter values on the results achieved by the rollout algorithm with look-ahead. For these experiments the number of jobs in the rollout horizon was set to 3. The values denoted with a grey cell in the table represent

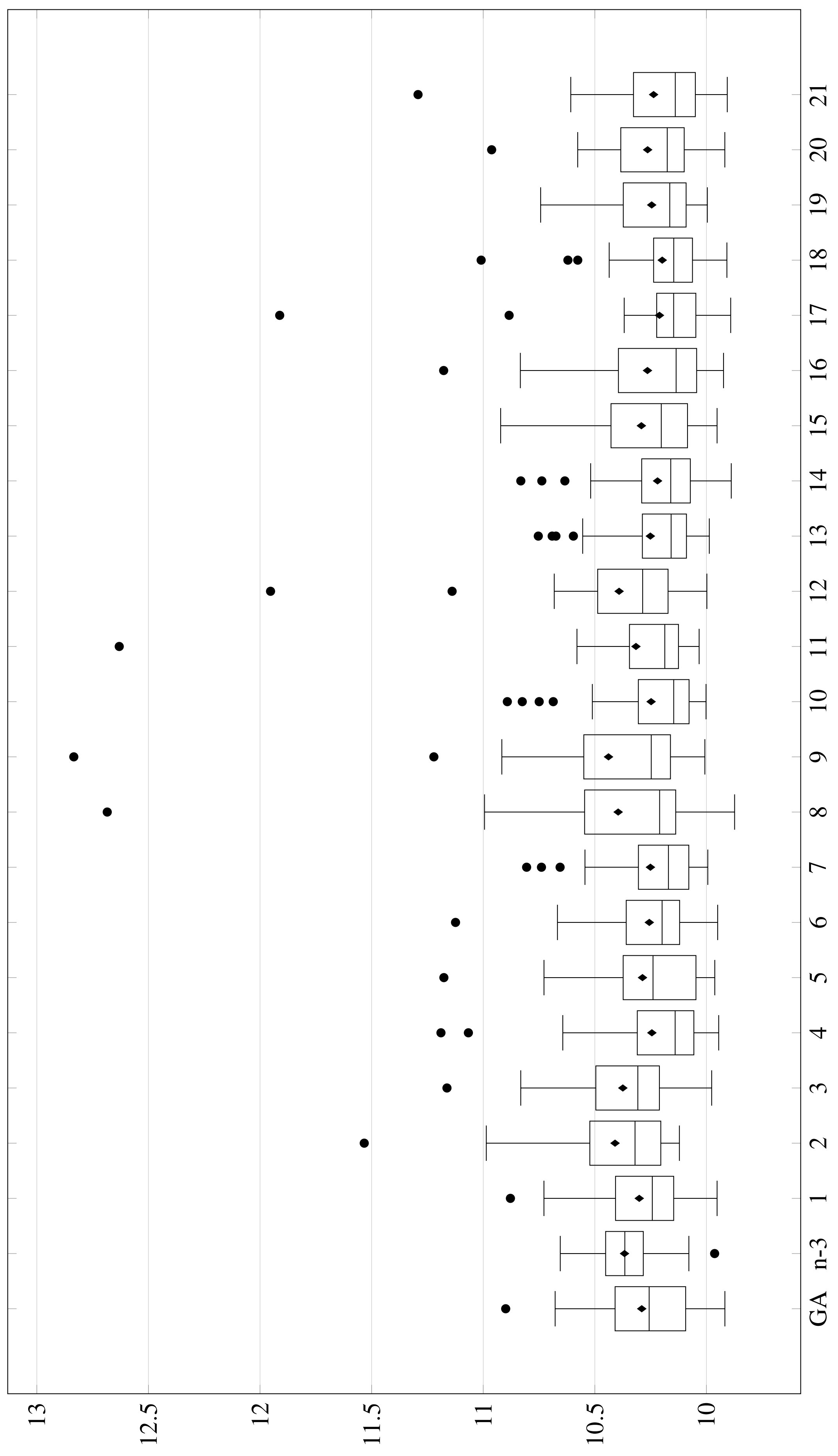


Figure 8.17: Box plot representation of the results obtained by the rollout algorithm with static terminal nodes, when using DRs generated by different combinations of static terminal nodes

Table 8.20: Results obtained by the rollout algorithm with look-ahead, when using different rollout parameter values

Rollout factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0	9.990	10.59	12.01	−	0	9.990	10.59	12.01	−
0.03	9.939	10.50	11.73	−	3	9.868	10.10	11.08	+
0.05	9.921	10.40	11.54	≈	5	9.785	10.04	11.20	+
0.1	9.853	10.26	11.27	≈	10	**9.713**	9.914	**10.62**	+
0.2	9.906	10.21	11.09	≈	20	9.744	**9.903**	10.76	+
0.5	9.824	10.08	**10.64**	+	50	9.780	9.955	10.78	+
1	**9.780**	**9.955**	10.85	+	100	9.780	9.955	10.85	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

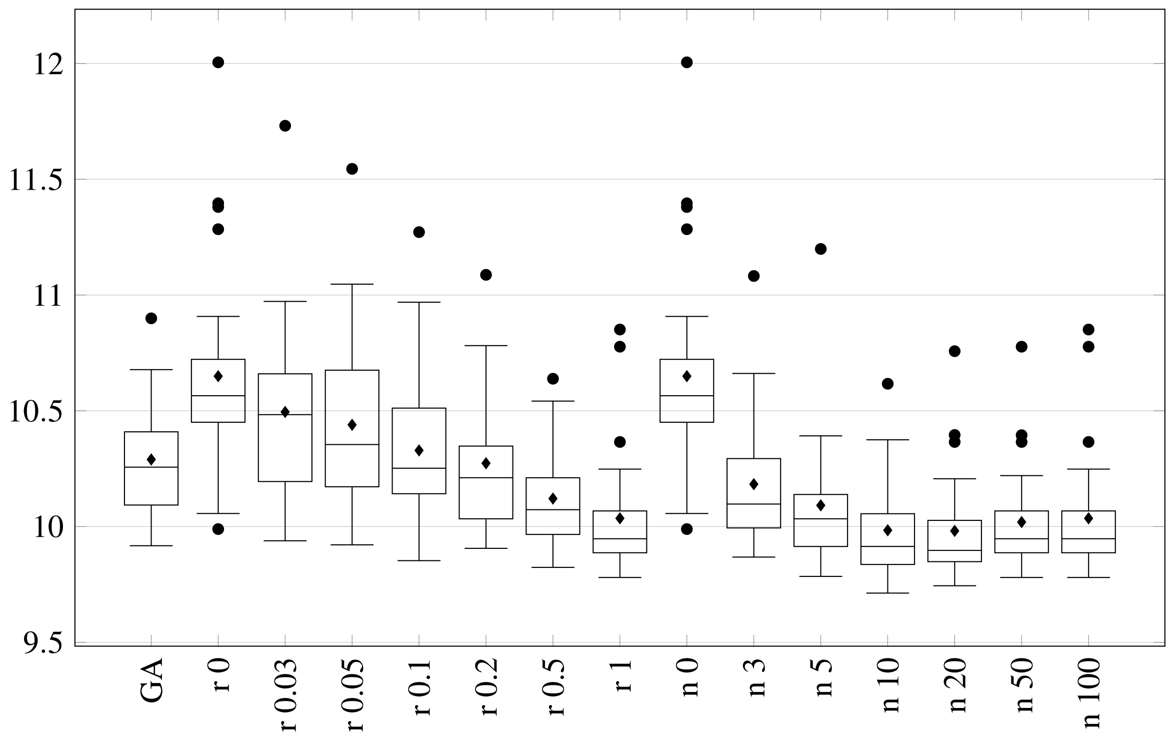


Figure 8.18: Box plot representation of the results obtained by the rollout algorithm with look-ahead, when using different rollout parameter values

Table 8.21: Results obtained by the rollout algorithm with look-ahead, when using different look-ahead parameter values

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	10.09	10.73	11.15	–	3	9.919	10.26	11.07	≈
0.05	10.11	10.61	11.79	–	5	9.877	10.15	10.85	≈
0.1	9.920	10.42	**10.81**	–	10	9.868	10.10	11.08	+
0.2	10.02	10.30	10.83	≈	20	9.932	10.18	10.88	≈
0.5	9.884	10.23	11.09	≈	50	9.782	10.16	10.96	≈
1	**9.842**	**10.12**	11.59	≈	100	**9.773**	**10.05**	**10.69**	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

the results where for the same parameter value the rollout algorithm with DRs that use look-ahead achieved significantly better results than the rollout algorithm which uses DRs generated by DGP. The results demonstrate that when using DRs with constant number of jobs in the look-ahead horizon the rollout algorithm achieves significantly better results, for most of the parameter values, than the rollout algorithm which uses DRs generated by DGP. By using the look-ahead factor, far worse results were achieved than by using a constant number of jobs in the look-ahead horizon. However, by using a constant number of jobs in the rollout horizon very similar results are achieved for various values of the look-ahead parameter. Therefore, by using DRs with look-ahead the results of the rollout algorithm can be improved further. However, the concrete value of the number of jobs in the look-ahead horizon will not have a significant influence on the performance of the rollout algorithm.

Figure 8.19 shows the box plot representation of the results. The figure denotes that when the look-ahead factor is used, the results constantly improve with the increase of the value of the parameter. On the other hand, when a constant number of jobs in the look-ahead horizon is used, the algorithm achieved similar results for parameter values larger than 5. The results obtained by the rollout algorithm which uses DRs generated by DGP and three jobs in the rollout horizon, are additionally included in the figure and denoted as *R n-3*. These results demonstrate that when a constant number of jobs in the look-ahead horizon is used by the rollout algorithm, it will achieve much better results than by using DRs generated by GP.

The results presented in this section show that the performance of the rollout algorithm can be improved by using DRs with look-ahead. It was shown that the number of jobs which are used in the look-ahead horizon do not have a large influence on the performance of the rollout algorithm. It is only important not to select a too small number of jobs in the look-ahead

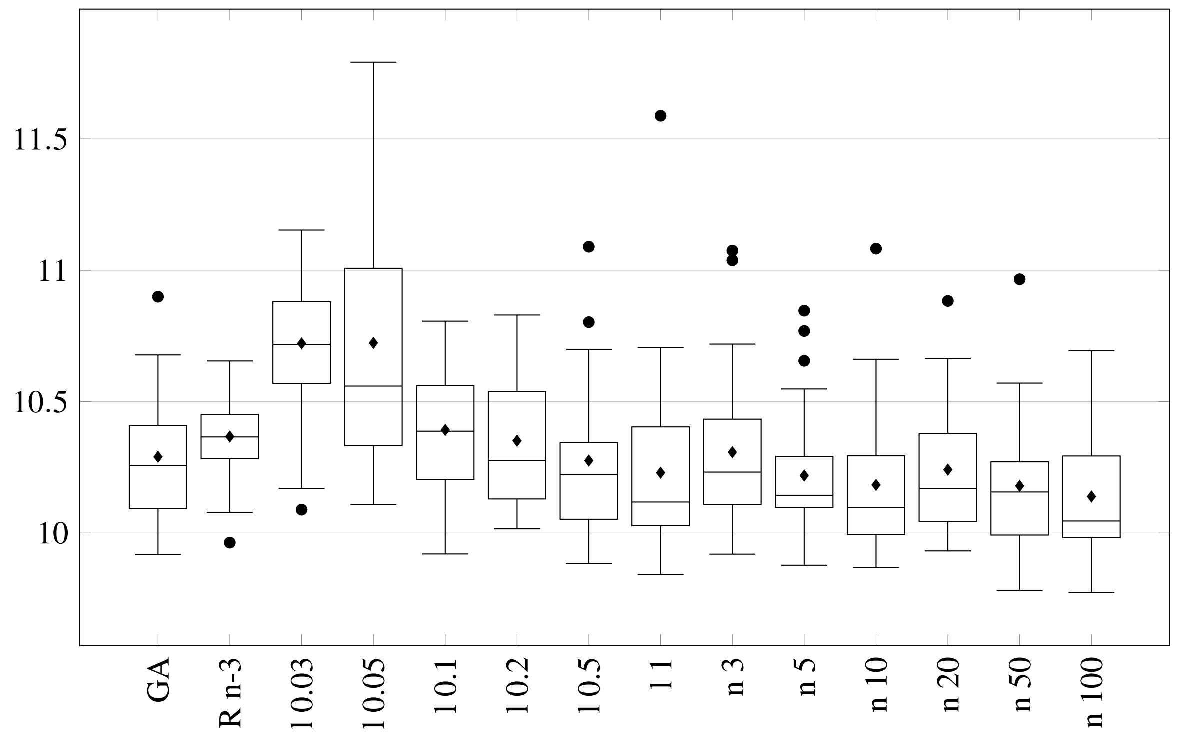


Figure 8.19: Box plot representation of the results obtained by the rollout algorithm with look-ahead, when using different look-ahead parameter values

horizon.

8.2.11 Results obtained by the rollout algorithm with static terminals and look-ahead

This section will analyse whether by combining the rollout algorithm with look-ahead and static terminal nodes it is possible to achieve improved results than by using a combination of only two of those approaches. The results will analyse how the rollout parameters, look-ahead parameters and different combinations of static terminal nodes have an influence on the quality of the results. In addition, the influence of the two calculation methods of static terminal nodes will be tested in the experiments.

Table 8.22 represents the influence of the rollout parameters on the rollout algorithm which uses look-ahead and static terminal nodes that are calculated based on all unreleased jobs. The experiments use the static node combination denoted with index 21 in Table 8.3 and 20 jobs in the look-ahead horizon. The statistical tests show that the obtained results do not perform significantly better than neither the rollout algorithm with DRs generated by using static terminal nodes, nor the rollout algorithm with look-ahead. However, for certain rollout parameter values, especially when using a constant number of jobs in the rollout horizon, the experiments outperformed the results obtained by the GA.

Figure 8.20 shows the box plot representation of the results for the influence of the rollout

Table 8.22: Results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs, when using different rollout parameter values

Rollout factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0	10.04	10.71	11.61	–	0	10.04	10.71	11.61	–
0.03	9.975	10.51	11.47	–	3	9.827	10.15	11.21	≈
0.05	9.905	10.49	11.04	–	5	**9.729**	10.06	10.71	+
0.1	9.828	10.33	11.13	≈	10	9.731	**9.999**	10.94	+
0.2	9.852	10.27	11.05	≈	20	9.732	10.03	11.05	+
0.5	9.878	10.11	10.96	+	50	9.738	10.03	**10.92**	+
1	**9.738**	**10.05**	**10.92**	+	100	9.738	10.05	10.92	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

parameters. The figure shows that the rollout algorithm achieved quite bad results for smaller parameter values, especially when using the rollout factor. In addition it is evident that for larger parameter values it is more common that the approach obtains outlier values. Nevertheless, for several parameter values the rollout algorithm achieved good solution distributions.

Table 8.23 represents the influence of the rollout parameters on the rollout algorithm with look-ahead and static terminal nodes which are calculated based on unreleased jobs outside the look-ahead horizon. The same node static terminal node combination and look-ahead parameters are used as for the previous experiments. The experiments which are denoted in grey achieved significantly better results than the rollout algorithm with static terminal nodes. On the other hand, the experiments which are underlined achieved significantly better results than the rollout algorithm with look-ahead for the same parameter values. By using this calculation method of static terminal nodes, better results are achieved when compared to the previous calculation method. This is evident from the fact that all experiments, except for the smallest parameter value, achieved equally good or better results than the GA. In addition, for several parameter values the experiments also achieved significantly better results than the rollout algorithm with static terminal nodes. Therefore, adding look-ahead to the rollout algorithm with static terminal nodes generally leads to better results. However, the experiments were unable to outperform the rollout algorithm with look-ahead, which means that with the addition of static terminal nodes it is not possible to obtain better results in that case.

Figure 8.21 denotes the box plot representation of the results. The figure shows that the rollout algorithm achieved much better solution distributions than the GA for several rollout parameter values. In addition, the experiments have also shown that the procedure is more

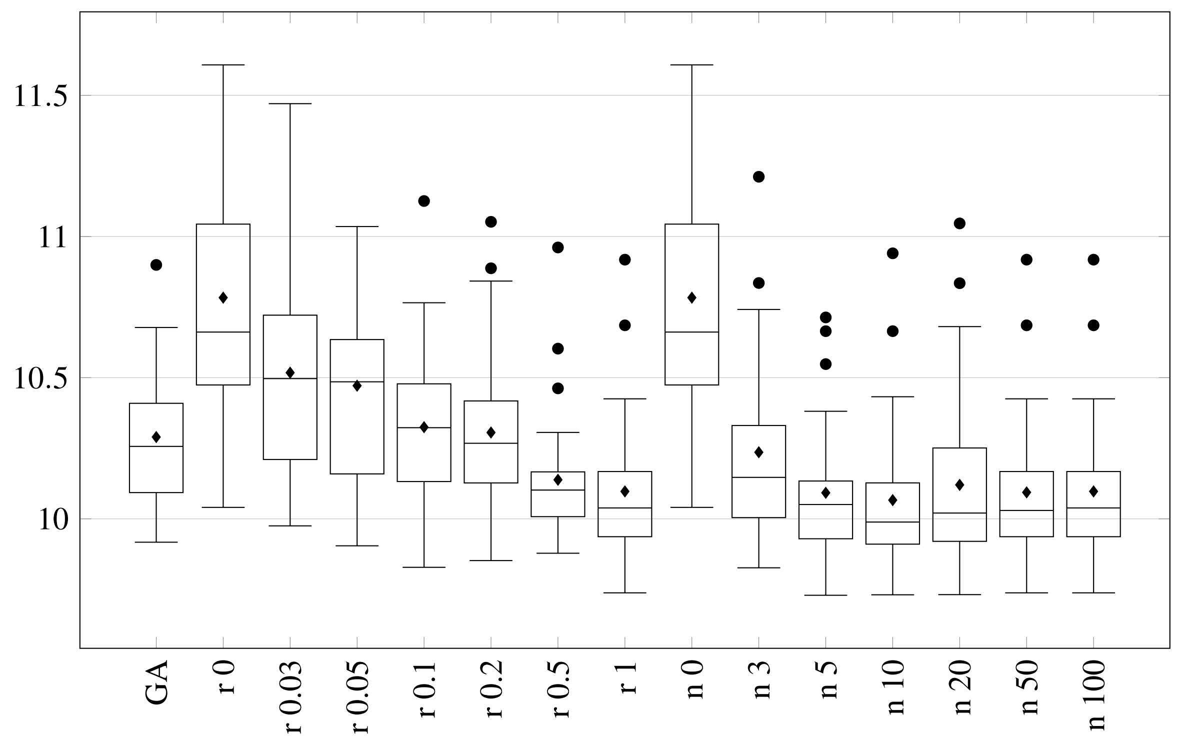


Figure 8.20: Box plot representation of the results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs, when using different rollout parameter values

Table 8.23: Results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon, when using different rollout parameters

Rollout factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0	9.977	10.60	11.36	–	0	9.977	10.60	11.36	–
0.03	9.942	10.36	10.95	≈	3	9.810	10.10	10.64	+
0.05	9.884	10.34	10.92	≈	5	9.713	10.04	10.63	+
0.1	9.774	10.31	10.91	≈	10	**9.674**	**9.898**	10.25	+
0.2	9.793	10.09	10.53	+	20	9.744	9.922	**10.12**	+
0.5	9.793	10.04	10.55	+	50	9.750	9.925	10.27	+
1	**9.750**	**9.917**	**10.27**	+	100	9.750	9.917	10.27	+
GA	9.917	10.27	10.90			9.917	10.27	10.90	

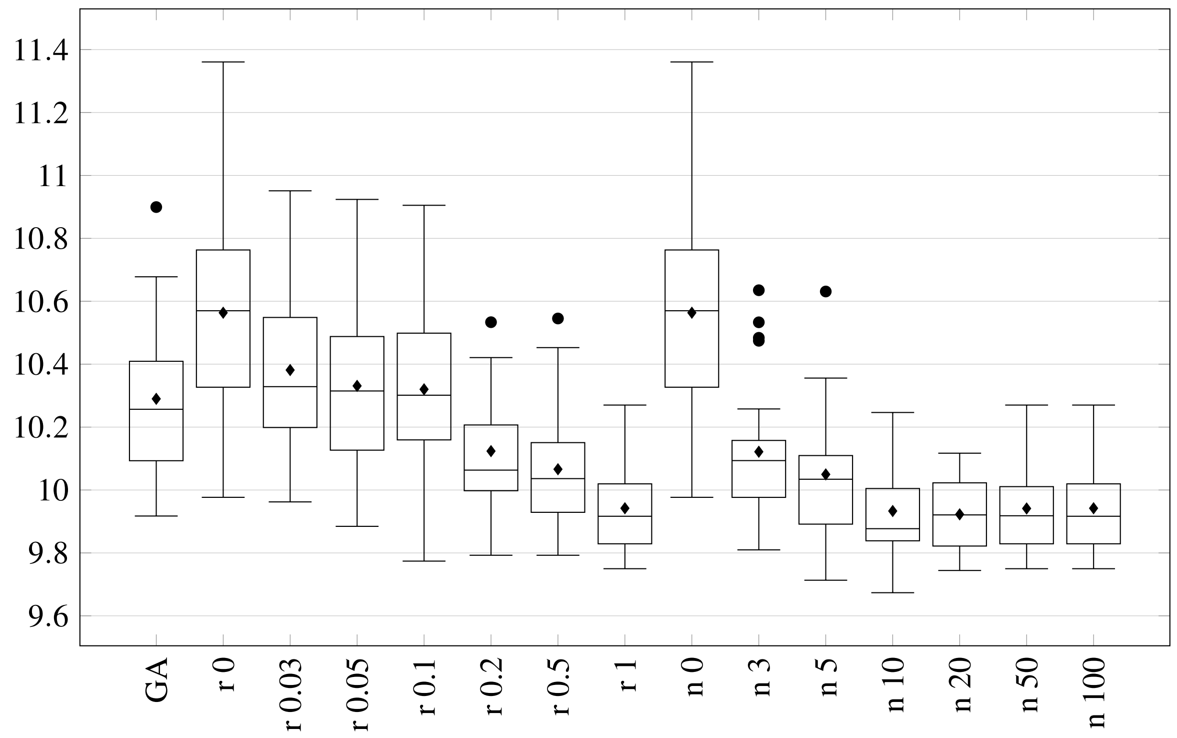


Figure 8.21: Box plot representation of the results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon, when using different rollout parameters

stable than the GA, since it achieved less dispersed results. This is most evident from the results achieved when using a constant number of jobs in the rollout horizon. Even when compared to the previous calculation method it is evident that this calculation method achieved a much better distribution of solution.

Table 8.24 represents the influence of the look-ahead parameters on the performance of the rollout algorithm with look-ahead and static terminal nodes which are calculated based on all unreleased jobs. The experiments use the static terminal node combination denoted with the index 21 in Table 8.3 and three jobs in the rollout horizon. The underlined experiments achieved a significantly better performance than the rollout algorithm with the same parameters, whereas the experiments denoted in grey achieved significantly better results than the rollout algorithm with look-ahead. The experiments have shown that only in a few occasions this method achieved significantly better results than the rollout algorithm with and without look-ahead. In none of the experiments were able to significantly outperform the results achieved by the GA.

Figure 8.22 denotes the box plot representation of the results. The figure shows that the rollout algorithm achieved quite bad solution distributions, which are usually more dispersed than those of the GA. The result of the rollout algorithm with static terminal nodes, which uses the same parameters as the other experiments, is also included in the table and is denoted with the label *R-n-3 S-21*.

Table 8.25 represents the influence of the look-ahead parameters on the results achieved by

Table 8.24: Results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs, when using different look-ahead parameter values

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	9.983	10.27	11.25	≈	3	9.870	10.27	10.82	≈
0.05	10.05	10.45	11.70	–	5	9.889	10.27	**10.73**	≈
0.1	9.894	10.31	11.58	≈	10	9.827	**10.15**	11.21	≈
0.2	9.893	10.32	10.80	≈	20	9.891	10.27	12.22	≈
0.5	**9.765**	**10.19**	**10.61**	≈	50	9.965	10.27	11.11	≈
1	9.913	10.29	10.71	≈	100	**9.773**	10.33	11.32	≈
GA	9.917	10.27	10.90			9.917	10.27	10.90	

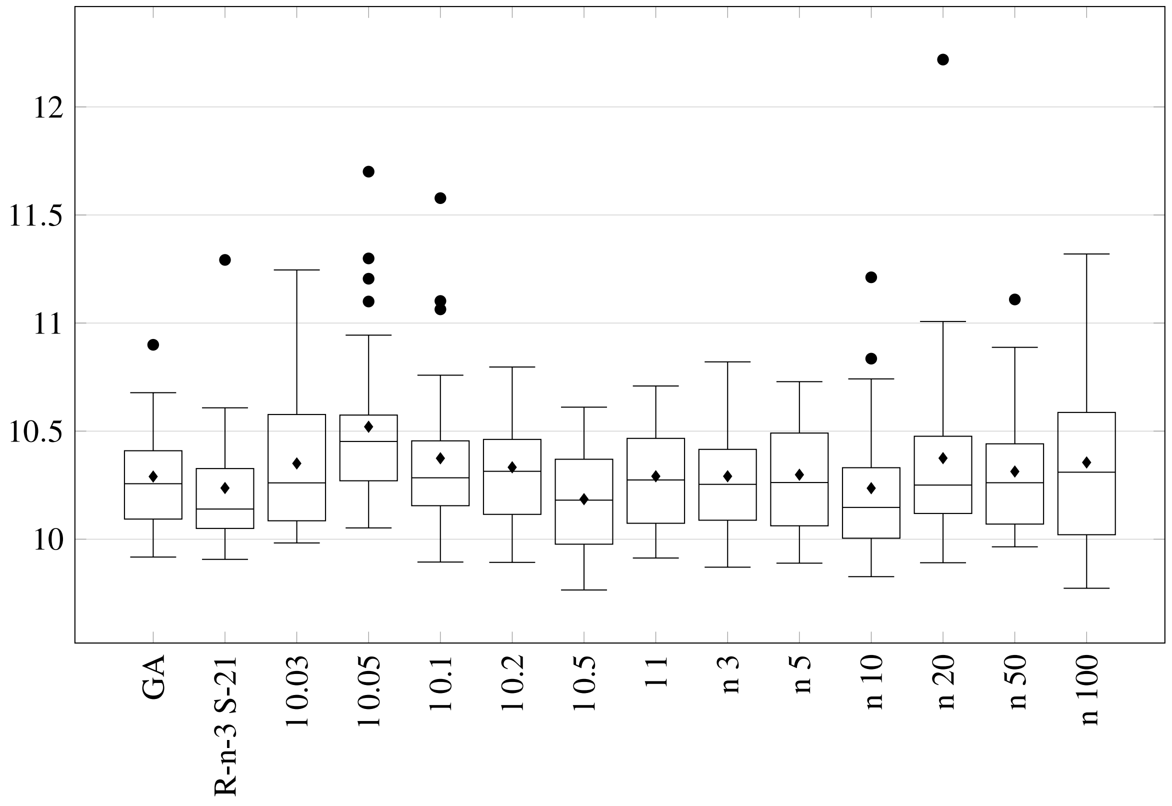


Figure 8.22: Box plot representation of the results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs, when using different look-ahead parameter values

Table 8.25: Results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon, when using different look-ahead parameter values

Look-ahead factor	Min	Med	Max	Stat. diff.	Number of jobs	Min	Med	Max	Stat. diff.
0.03	10.09	10.47	11.27	–	3	9.855	10.14	10.82	≈
0.05	9.932	10.46	11.62	–	5	9.855	**10.10**	11.61	+
0.1	9.977	10.44	12.50	–	10	9.852	10.18	**10.52**	≈
0.2	10.05	10.32	11.15	≈	20	9.810	**10.10**	10.64	+
0.5	9.928	10.33	11.30	≈	50	**9.803**	10.12	11.12	+
1	**9.891**	**10.21**	**10.95**	≈	100	9.903	10.22	11.15	≈
GA	9.917	10.27	10.90			9.917	10.27	10.90	

the rollout algorithm with look-ahead and static terminal nodes calculated from all unreleased jobs outside the look-ahead horizon. The experiments use the static terminal node combination denoted with the index 21 in Table 8.3 and three jobs in the rollout horizon. The underlined experiments in the table achieved significantly better results than the rollout algorithm, while on the other hand the experiments denoted in grey achieved significantly better results than the rollout algorithm with look-ahead. The results show that the values of the look-ahead parameters do not have a strong influence on the performance of the rollout algorithm. When observing the results for the number of jobs in the look-ahead horizon, depending on the choice of the parameter value, the rollout algorithm will in certain cases outperform the GA, however, the overall differences between the different experiments are small. Furthermore, in most cases significantly better results were achieved than by using the rollout algorithm with DRs generated by DGP. However, only in one occasion the experiments achieved a better value than by using the rollout algorithm with look-ahead. Finally, better results are generally achieved than by using the previous calculation method of the static terminal nodes.

Figure 8.23 shows the box plot representation of the results. The figure denoted that the rollout algorithm achieves good solution distributions for certain look-ahead parameter values when compared to the GA. This is especially true when using a constant number of jobs in the look-ahead horizon. In addition, it is also evident that this calculation method of static nodes again achieves much better results than the calculation method based on all unreleased jobs. The figure also includes the result for the rollout algorithm with static terminal nodes, where the same parameter values are used as in the other experiments. This result is denoted with the label *R-n-3 S-21*. The results show that most of the experiments did not achieve equally good

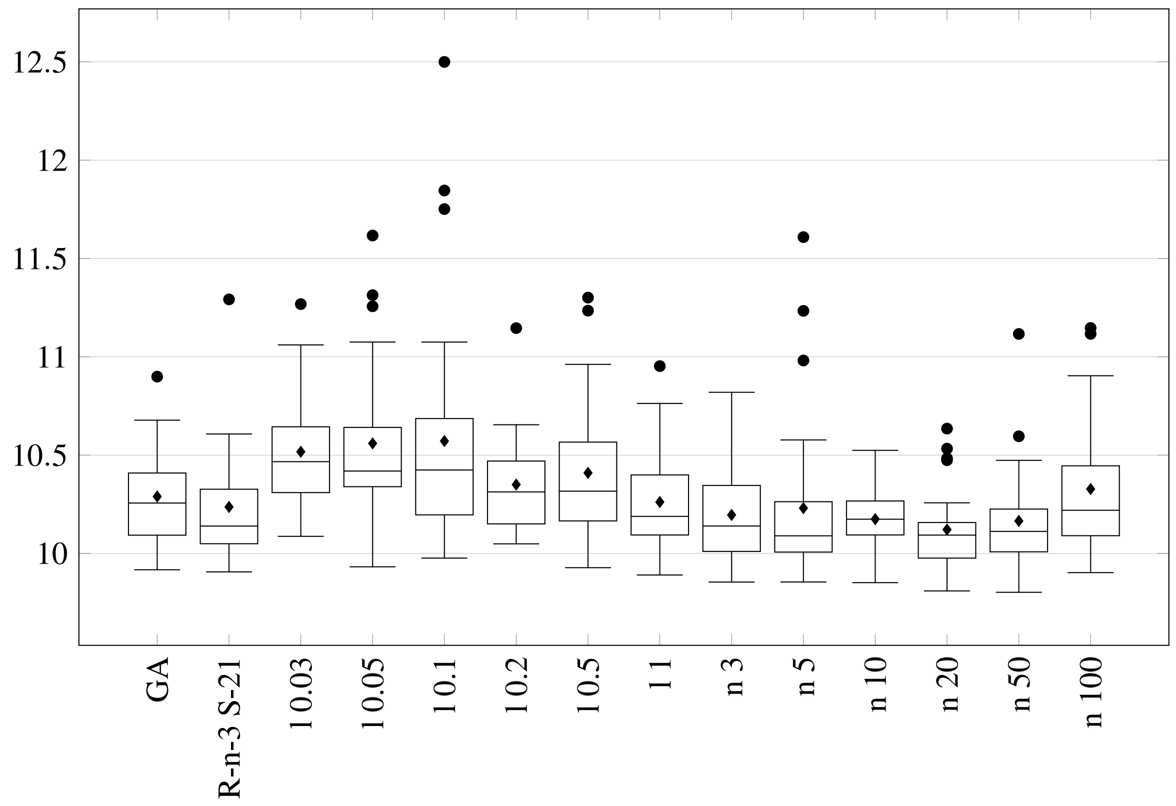


Figure 8.23: Box plot representation of the results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon, when using different look-ahead parameter values

results as the rollout algorithm which used only static terminal nodes.

Table 8.26 represents the results for different static node combinations achieved by rollout with look-ahead and static terminal nodes calculated from all unreleased jobs. The experiments use 20 jobs in the look-ahead horizon and 5 jobs in the rollout horizon. Unfortunately, it is evident that the rollout algorithm achieved quite bad results for the different static node combinations. For none of the tested static node combinations the rollout algorithm achieved significantly better results when compared to those obtained by the GA. The tested combinations of methods also did not achieve better results than the rollout algorithm with DRs generated by DGP or DRs with static terminal nodes.

Figure 8.24 represents the box plot representation of the results for the different static node combinations, when calculating them based on all unreleased jobs. Based on the previous observation, it can be concluded that by using static terminals with this calculation method should be avoided since it leads to an unstable method that achieves generally bad results.

Table 8.27 represents the influence of different static node combinations on the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon. The experiments use 20 jobs in the look-ahead horizon and 5 jobs in the rollout horizon. Those experiments which achieved significantly better results than rollout are

Table 8.26: Results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs, when using different static node combinations

Static terminal combination	Min	Med	Max	Stat. diff.
1	9.865	10.46	15.74	–
2	9.853	10.23	18.75	≈
3	9.754	10.35	17.20	≈
4	9.716	10.46	19.23	≈
5	9.783	10.21	13.29	≈
6	9.754	10.22	14.73	≈
7	9.782	10.31	19.89	≈
8	9.829	10.74	18.22	–
9	9.829	10.61	15.66	≈
10	**9.695**	10.34	16.32	≈
11	9.792	10.80	16.45	≈
12	9.835	10.48	14.88	≈
13	9.780	10.50	16.48	–
14	9.902	10.64	25.85	–
15	9.880	10.36	15.79	≈
16	9.835	10.68	16.45	≈
17	9.808	10.40	14.08	≈
18	9.902	10.66	15.76	–
19	9.907	10.42	15.34	≈
20	9.907	10.30	15.33	≈
21	9.735	**10.18**	**12.87**	≈
GA	9.917	10.27	10.90	

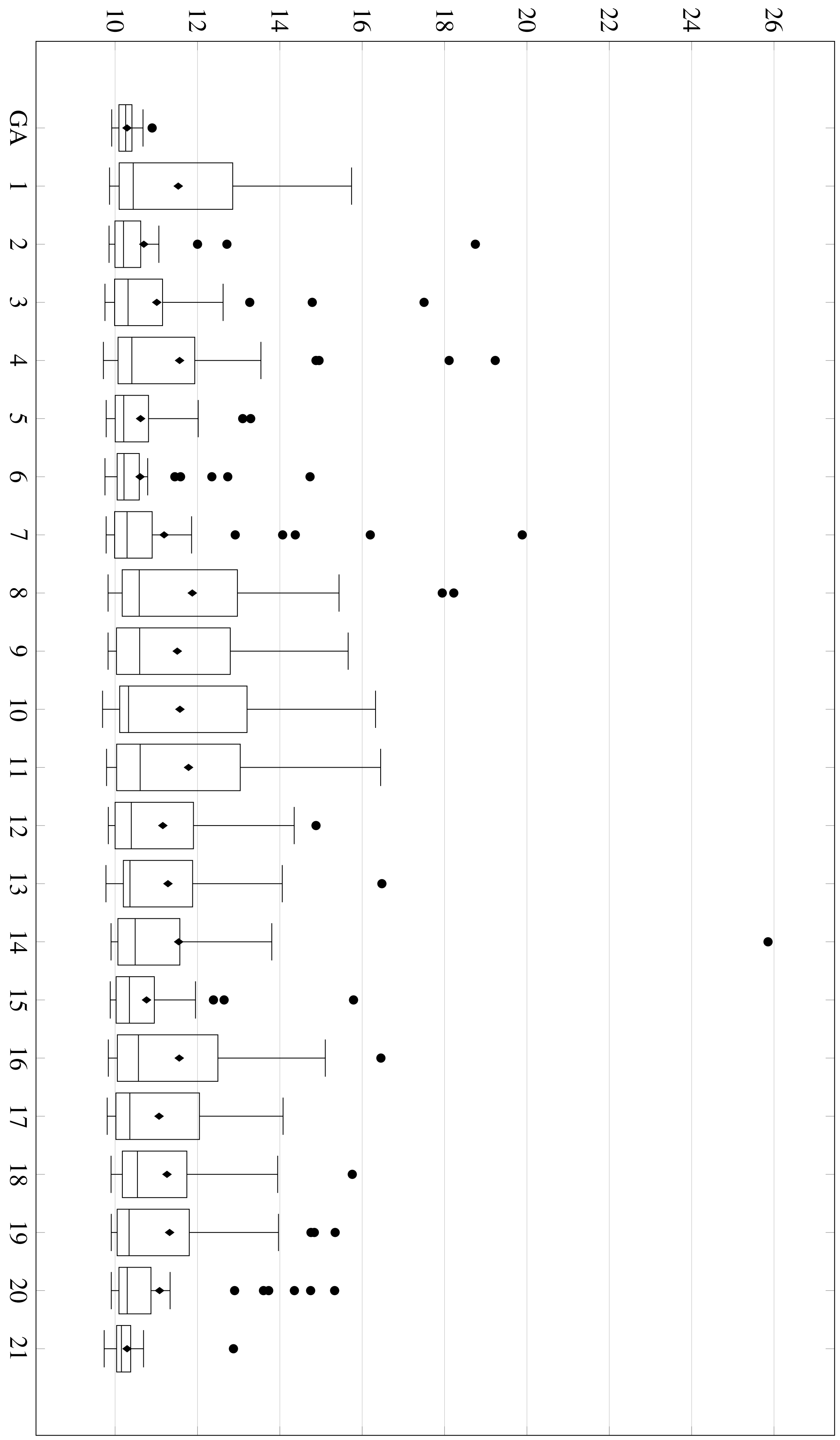


Figure 8.24: Box plot representation of the results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs, when using different static node combinations

underlined, while on the other hand the experiments which achieved significantly better results than the rollout algorithm with look-ahead are denoted in grey. Although for most of the static node combinations the achieved results can not outperform the results achieved by the GA or the rollout algorithm, there are a few node combinations for which good results are certainly achieved. For all node combinations the rollout algorithm performed at least equally good as the GA. In addition, based on the median values it is evident that big differences are not achieved by the different static node combinations.

Figure 8.25 represents the box plot representation of the results for the different static node combinations. The first thing which can be noticed is that with this calculation method the rollout algorithm achieved much less dispersed solutions than by calculating static terminals based on all unreleased jobs. There is still a chance of obtaining some bad outlier values, but to a smaller extent than for the previous calculation method of the static terminal nodes. The figure also shows that for certain node combinations, like 3 and 11, it is possible to achieve very good solution distributions, much better than those of GA.

Based on the results denoted in this section it can be concluded that the the calculation method of static nodes, which considers only unreleased jobs outside the look-ahead horizon, achieves better results and should therefore be preferred. In addition, some experiments have shown to achieve very good performance, which denotes that the combination of all three methods can perform well, but only with a good choice of parameters.

8.2.12 Comparison of all static scheduling methods

In the previous sections the results were presented for each of the methods individually, and the influence of the parameters of each of those methods was analysed. In this section, the best results from all the different methods will be collected and compared with each other. Since experiments from different methods will be used, a proper nomenclature needs to be defined. If static terminal nodes were used in the experiments, this will be denoted with "S" and the index of the combination of static nodes that was used. The experiments which use look-ahead will be denoted with "L". In addition, "l" will denote that the look-ahead factor is used, while "n" denotes that a constant number of jobs in the look-ahead horizon is used. The value of the look-ahead parameter will be denoted immediately after the "l" or "n" flag. If the static terminal nodes are used together with look-ahead in the experiment, then the flag "u" will be used to denote that the static terminals are calculated based on all unreleased jobs, while the "ul" flag will be used if the static terminals are calculated based on all unreleased jobs outside the look-ahead horizon. The use of IDRs will be denoted with "I", after which the index of the applied IDR node combination will be denoted. The experiments which use the rollout algorithm will be denoted with "R", followed by either "r", if the rollout factor is used, or "n", if the number of jobs in the rollout-horizon is used. The value of the rollout parameter will be

Table 8.27: Results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon, when using different static node combinations

Static terminal combination	Min	Med	Max	Stat. diff.
1	9.927	10.27	12.01	$\approx$
2	9.769	10.07	10.94	$+$
3	9.754	10.05	12.37	$+$
4	9.802	10.17	11.82	$\approx$
5	9.783	10.15	12.56	$\approx$
6	9.754	10.15	12.35	$\approx$
7	9.780	10.12	11.30	$+$
8	9.873	10.12	11.84	$\approx$
9	9.799	10.24	11.36	$\approx$
10	9.834	10.17	10.94	$\approx$
11	9.737	10.08	10.73	$+$
12	9.841	10.05	10.73	$+$
13	9.778	10.08	10.79	$+$
14	9.793	10.11	11.31	$\approx$
15	9.882	10.26	11.13	$\approx$
16	9.814	10.07	11.65	$\approx$
17	9.753	10.18	11.58	$\approx$
18	9.885	10.26	12.71	$\approx$
19	9.754	10.22	12.62	$\approx$
20	9.839	10.16	10.88	$\approx$
21	**9.713**	**10.04**	**10.63**	$+$
GA	9.917	10.27	10.90	

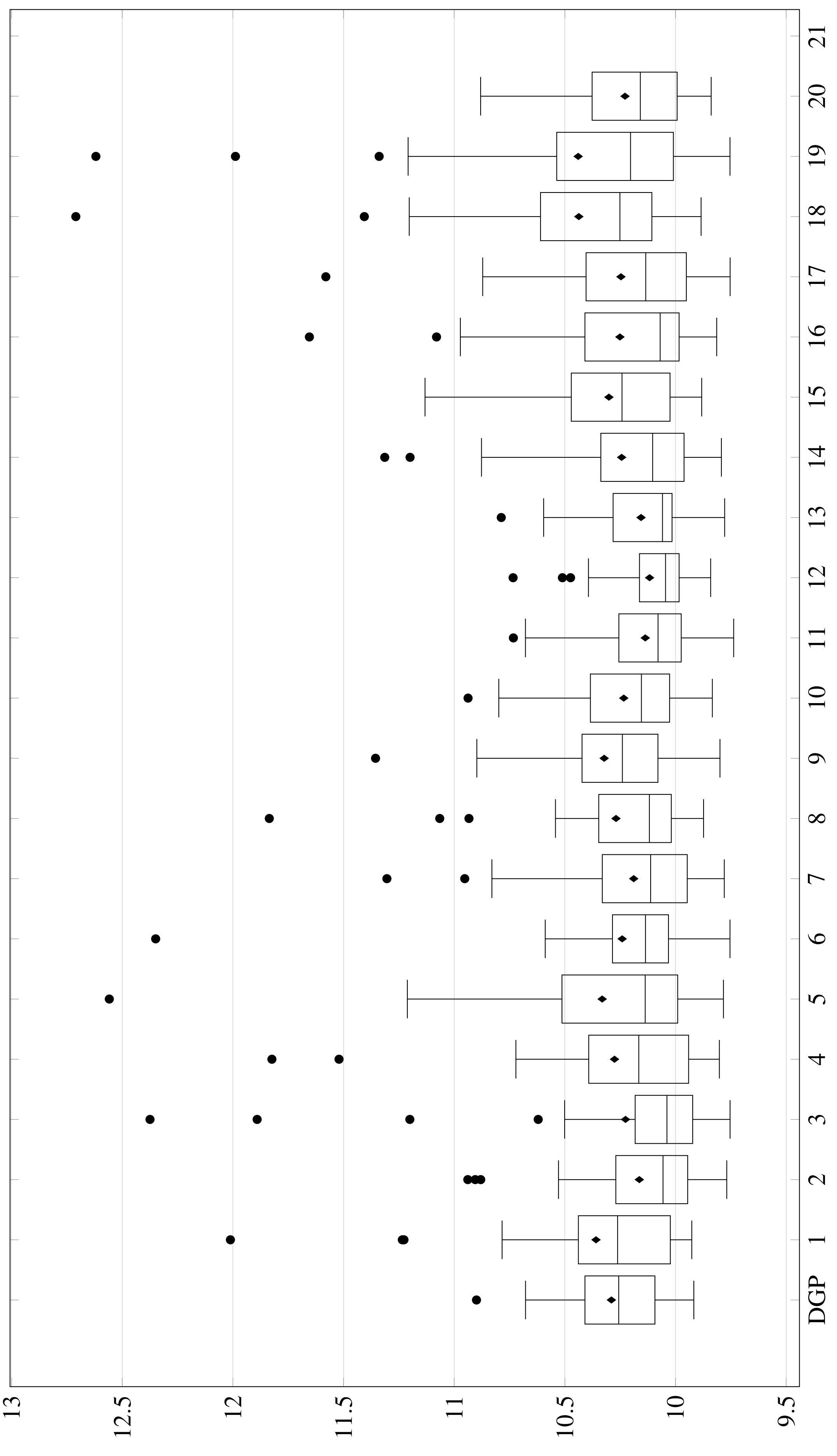


Figure 8.25: Box plot representation of the results obtained by the rollout algorithm with look-ahead and static terminal nodes calculated based on all unreleased jobs outside the look-ahead horizon, when using different static node combinations

denoted immediately after the "r" or "n" flag.

The experiments will additionally include the results achieved by the static version of the ATC rule [111]. This version of the ATC rule will calculate the priorities of jobs using the following priority function:

$$\pi_{i,j} = \frac{w_{T_j}}{p_{ij}} \exp\left[-\frac{\max\left(d_j - p_{i,j} - \max\left(r_j, time\right), 0\right)}{k_1 \bar{p}}\right] \exp\left[-\frac{\max\left(r_j - time, 0\right)}{k_2 \bar{p}}\right],$$

where $time$ denotes the current time of the system, $\bar{p}$ the average processing time of all jobs waiting to be scheduled, k_1 and k_2 the scaling parameters. The priority function shows that the static ATC rule additionally uses release times of jobs to take into account the time when unreleased jobs arrive in the system. Therefore, unlike the dynamic DRs where the priority function is only calculated for unscheduled jobs which are already released, the static ATC rule will calculate the priorities for all unscheduled jobs, whether they were already released into the system or not. This will have the consequence that the procedure will be quite time consuming, since it will have to calculate the priorities for all jobs each time when at least one job and one machine are free. The parameters which were used by the ATC rule were 0.7 for k_1 and 0.2 for k_2, since for those values the rule achieved the best results on the training set.

Table 8.28 contains the best results of all the methods which were tested in the previous section. The results are additionally represented using box plots in Figure 8.26. The first thing which can be noticed is that the methods achieve a wide range of results. The worst results are achieved by IDRs and DRs which use static terminal nodes. Out of these two methods better results were achieved by IDRs, but only by a very small margin. DRs which use look-ahead already achieve much better results than IDRs or DRs with static terminal nodes, with improvements of around 8.4% for the minimum value and 11% for the median value. Therefore, by using look-ahead significantly better results can be achieved than by using either static terminal nodes or IDRs. Combining look-ahead and static terminal nodes did not lead to any improvements in the results over look-ahead on its own. On the other hand the combination of IDRs and look-ahead outperformed the results of look-ahead by 2.7% for the minimum and 4.6% for the median value. However, by additionally using static terminal nodes with IDRs and look-ahead does not lead to any improvements in the results. Therefore, combining the static terminal nodes with the other two methods did not prove to be worthwhile.

The rollout algorithm achieved the best results out of all four tested static DR methods. The algorithm outperformed the results of IDRs and DRs with static terminal nodes by 24% for the minimum value, and 23% for the median value. Furthermore, the rollout algorithm outperformed DRs with look-ahead by 9.5% for the minimum value, and 13.4% for the median value. Based on these results it is more than evident that the rollout algorithm achieves superior performance compared to the other three tested methods. However, by combining the rollout

Table 8.28: The results obtained by the different static scheduling methods

Method	Min	Med	Max
DGP	12.96	13.60	14.62
ATC	12.45	-	-
GA	9.917	10.27	10.90
S-14	12.27	13.21	15.10
S-21	12.06	13.30	16.59
L-n-10	10.82	11.83	14.60
L-n-100	11.02	11.64	13.53
L-n-10 S-u-21	11.22	12.46	14.21
L-n-10 S-ul-21	10.90	12.24	14.11
L-n-20 S-ul-21	11.10	11.64	14.54
I-3	11.87	13.08	13.94
I-4	11.82	13.18	14.39
I-4 S-21	12.11	12.88	14.31
I-4 L-n-5	10,75	11.17	14.41
I-4 L-n-100	10.53	11.10	12.44
I-4 L-n-10 S-ul-21	10.77	11.42	14.29
I-4 L-n-100 S-ul-21	10.52	11.85	13.07
R-n-3	9.956	10.37	10.65
R-n-20	9.790	10.08	10.88
R-n-20 S-21	9.769	10.08	10.74
R-n-10 L-n-10	9.713	9.914	10.62
R-n-20 L-n-10	9.744	9.903	10.74
R-n-3 L-n-100	9.773	10.05	10.69
R-n-10 L-n-20 S-u-21	9.731	9.999	10.94
R-n-3 L-n-50 S-u-21	9.965	10.27	11.11
R-n-3 L-n-20 S-ul-21	9.810	10.10	10.64
R-n-10 L-n-20 S-ul-21	**9.674**	**9.898**	**10.25**
R-n-10 L-n-100 S-ul-21	9.750	9.917	10.27

algorithm with look-ahead and static terminal nodes it is possible to even further increase the performance. This combination of methods can outperform the rollout algorithm by 1.2% for the minimum value, and 1.8% for the median value. Although the improvements are not large, a very good distribution of the results was achieved, such that 75% of the results obtained a *Twt* value smaller than 10. This combination of methods also achieved the overall best results for the minimum, median, and maximum values. Therefore, it is evident that adding look-ahead and static terminal nodes to the rollout algorithm is valuable, since it does not only improve the overall performance of the approach, but also increases its stability.

Comparing the achieved results to those of a GA, it is evident that only the rollout algorithm performed equally well or better than the GA. The GA was able to outperform DRs with static terminal nodes by 17.7% for the minimum value, and 22.8% for the median value. On the other hand, the GA outperformed IDRs by 16% for the minimum value, and 22% for the median value. Therefore, both of these methods achieved quite poor performance when compared to those of the GA. Although DRs with look-ahead perform better than IDRs and DRs with static terminal nodes, the GA outperforms them by 10% for the minimum value, and 13% for the median value. For the combination of IDRs and look-ahead, the GA outperforms their results by 5.8% for the minimum value, and 7.5% for the median value. Therefore, the improvements of the GA are now reduced almost by half when compared to the improvements it achieved over DRs with look-ahead. These are also the smallest improvement which the GA achieved over the static DR methods, if the rollout algorithm is not used. On the other hand, the rollout algorithm outperformed the results of the GA by 1.3% for the minimum value, and 1.9% for the median value. Although the improvements are small, they can be further increased by combining the rollout algorithm with look-ahead and static terminal nodes. This combination of methods outperformed the GA by 2.5% for the minimum value, and 3.6% for the median value. Therefore, the rollout algorithm has demonstrated to be expressive enough to achieve much better performance than the GA.

8.3 Execution time analysis

Apart from the performance of the different algorithms for a certain criterion, the execution time of the algorithm also represents an important factor. For example, in certain occasions it might be desirable to select a procedure which achieves worse results to a certain degree, but on the other hand executes much faster. Sometimes it is also possible that certain procedures perform similarly for the same criterion, but have vastly different execution times. Because of those reasons it is important to also outline the execution times of the tested methods. For that purpose, this section will give an overview of the execution times of the different methods which were tested. It needs to be stressed out that only the time required to create the schedule

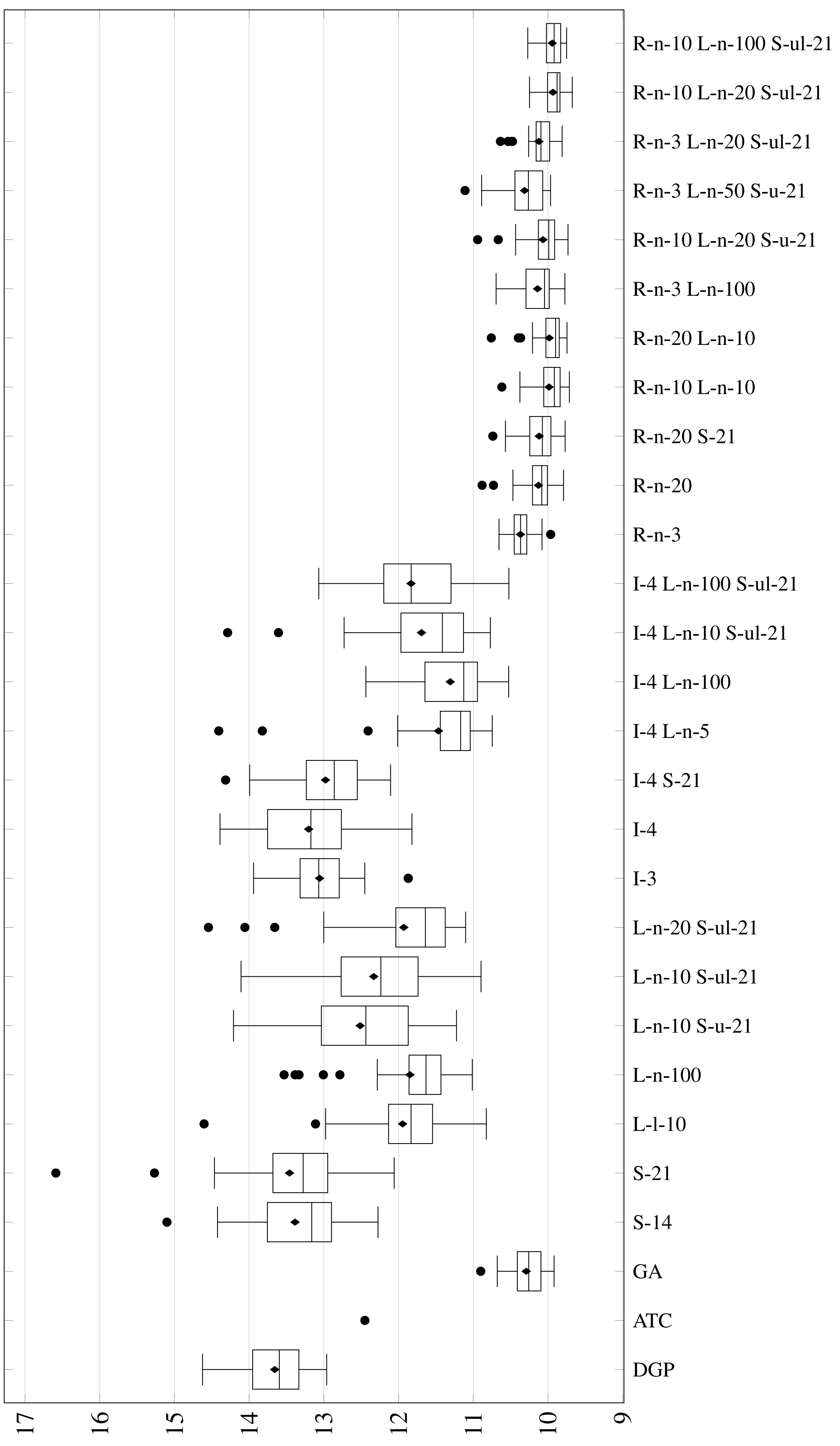


Figure 8.26: Box plot representation of the results obtained by the different static scheduling methods

will be considered in this section, and not the time needed to evolve the various DRs, since the evolution process can be done at a prior time. In the first subsection the influence of the different parameter values on the execution time will be analysed. In the second subsection all the methods will be compared based on their execution time and performance.

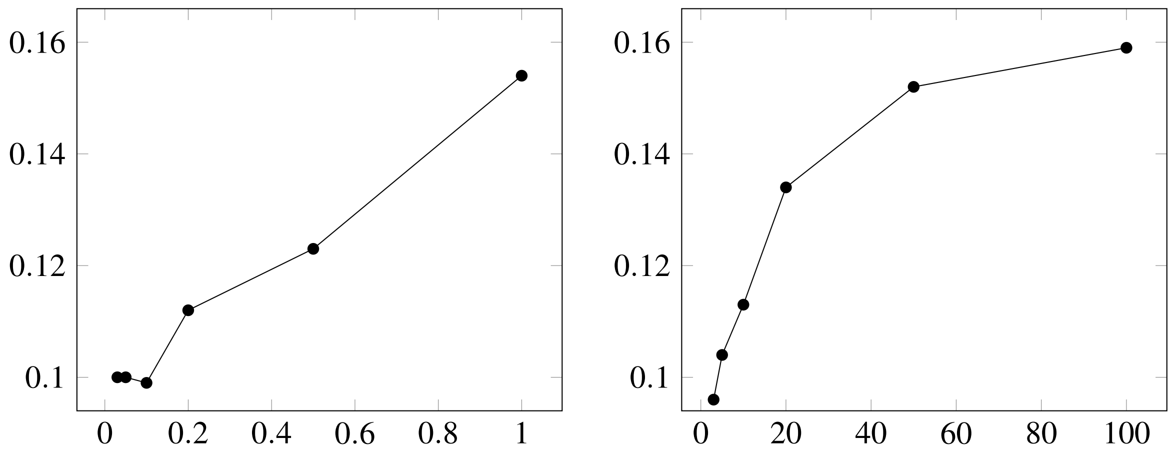
8.3.1 Influence of the parameter values of static methods on the execution time

Through the experiments it was shown that the execution time of different methods depends heavily on the selected parameter values. The combination of static terminal nodes and IDR nodes does not have a large influence on the execution times, and therefore will not be considered in this section. However, the choice of look-ahead and rollout parameters influences the execution time of the generated DRs. For that reason, this section will analyse the execution times for the different values of the look-ahead and rollout parameters, when using DRs generated by DGP. The execution times for the combinations of different methods will not be analysed, since the increase in the parameter values has demonstrated to have the same influence. All the execution times are calculated as average values of the 30 runs for each experiment, and will be denoted in seconds.

Table 8.29 represents the influence of the look-ahead parameters on the execution time of DRs which use look-ahead. Figure 8.27 graphically represents this influence. The results show that when using small values for the look-ahead factor, there is little difference in the execution time for the different values. This behaviour is expected, since for smaller problem instances almost none of the unreleased jobs appear in the look-ahead horizon for smaller parameter values. However, for the look-ahead factor values larger than 0.1, the execution time of the DRs seems to be increasing linearly. This is also expected since the number of jobs which will be considered in the look-ahead horizon will mostly linearly depend on the value of the parameter. Therefore, by using a larger look-ahead parameter value the look-ahead horizon will contain more jobs for which the priorities must be calculated, which leads to an increased execution time of the method. On the other hand, when using a fixed number of jobs in the look-ahead horizon the execution time does not linearly increase with the number of jobs in the look-ahead horizon. The largest increase in the execution time is obtained for the smaller parameter values, while for the two largest the increase in the execution times is much less prominent. Such a behaviour is caused by the different problem instance sizes. When the number of jobs in the look-ahead horizon becomes larger than the number of jobs in the problem instance, this problem instance will have no effect on the execution time of the DR, since all jobs in the instance will be in the look-ahead horizon. Therefore, the largest increase in the execution times will occur for smaller parameter values, since the increase will have an effect on all problem

Table 8.29: Influence of the look-ahead parameter values on the execution time of the DRs

Look-ahead factor	Execution time	Number of jobs	Execution time
0.03	0.100	3	0.096
0.05	0.100	5	0.104
0.1	0.099	10	0.113
0.2	0.112	20	0.134
0.5	0.123	50	0.152
1	0.154	100	0.159



(a) Influence of the look-ahead parameter (b) Influence of the number of jobs

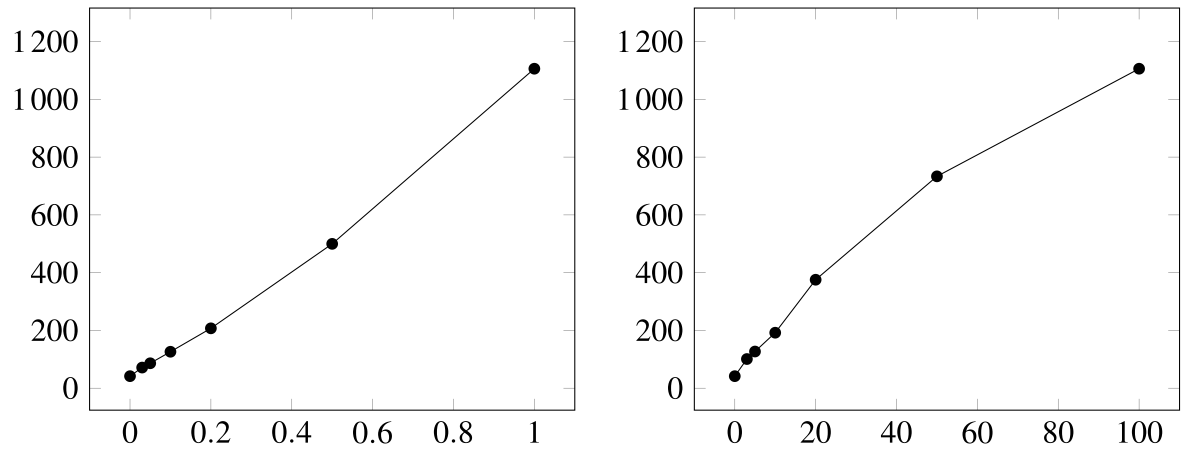
Figure 8.27: Graphical representation of the influence of the look-ahead parameter values on the execution time of the DRs

instances, whereas for larger parameter values the increase of the parameter will only influence the larger problem instances, therefore causing a smaller increase in the execution times. By using the largest value for the number of jobs in the look-ahead horizon, the method executes 1.7 times slower than when using the smallest parameter value. Therefore, the results denote that there is no significant difference between the execution times for smaller or larger parameter values, meaning that larger values can be used without any significant increase in the execution time.

Table 8.30 represents the influence of the rollout parameters on the execution time of the rollout algorithm. This influence is also presented in Figure 8.28. The influence is quite similar to that of the look-ahead parameters. With the increase of the rollout factor, the execution time of the algorithm also increases mostly linearly. When using the number of jobs in the rollout horizon, the increase in the execution times is again larger for the smaller parameter values, because of the same reason that was outlined for look-ahead. It is interesting to note that the

Table 8.30: Influence of the rollout parameter values on the execution time of DRs

Rollout factor	Execution time	Number of jobs	Execution time
0	41.69	0	41.69
0.03	71.47	3	101.0
0.05	86.35	5	127.0
0.1	126.2	10	192.1
0.2	207.3	20	375.5
0.5	499.7	50	733.4
1	1106	100	1106



(a) Influence of the rollout parameter (b) Influence of the number of jobs

Figure 8.28: Graphical representation of the influence of the rollout parameter values on the execution time of DRs

rollout parameters have a large influence on the execution time, since for the largest parameter value the algorithm is 27 times slower than for its smallest value. Therefore, depending on whether the performance or the execution time is more important, the appropriate rollout parameter needs to be selected.

8.3.2 Execution time comparison of all methods

This section will give an overview of the execution times of all methods in addition to their performance, to analyse how the different methods balance between the quality of the obtained solutions and the time needed to construct the schedules. This is necessary since depending on the scheduling problem it might be required to make a compromise between the quality of the solution and the execution time, so that the solution can be obtained faster but with a decreased quality.

Table 8.31 contains the results and the execution times of all the selected methods. The execution times in the table are denoted in seconds, and are calculated as the average value of the execution times obtained from all experiments. The table shows that the methods differ vastly by their execution times, ranging from 0.1 seconds up to 1603 seconds. The smallest execution time is achieved by DRs evolved for the dynamic conditions. By using the static terminal nodes, the execution times increase only slightly. The reason for this increase of the execution time is because additional static terminal nodes need to be calculated. DRs which use look-ahead also do not significantly increase the execution times, executing at most 1.8 times slower than DGP. However, look-ahead leads to significant improvements in the results, outperforming DGP by 17% for the minimum value, and 14% for the median value. Therefore, with only a small increase in the execution times it is possible to achieve significantly better results than by using DRs evolved by DGP. By combining look-ahead with static terminal nodes the execution time increases, however, the performance of the approach does not improve, therefore making this combination of methods useless. IDRs achieve the same execution time as look-ahead for the largest parameter value, and outperform DGP by 8.8% for the minimum value, and 3.8% for the median value. Unfortunately, the results obtained by IDRs are much worse than those obtained by look-ahead, thus making this method inferior. The different combinations of these three approaches increase the execution time, however, for all the combinations the execution time did not increase beyond 0.45 seconds, which is 5 times larger than the execution time of DRs evolved by DGP. The best results were achieved when using the combination of IDRs and look-ahead, which outperformed DGP by 19% for the minimum value, and 18.4% for the median value. This combination of methods achieved an execution time which is only 3.5 times larger than the one obtained by DRs evolved using DGP. This demonstrates that by only a small increase of the execution time a much better performance can be achieved than by using DGP.

When comparing the previous three approaches to the results achieved by the GA, it is evident that they obtain solutions in a much faster time. Naturally, the solutions obtained by these approaches will not be equally good as those achieved by the GA. For example, the GA outperforms the best solution of IDRs with look-ahead by 5.8% for the minimum value, and 7.5% for the median value. However, the time needed by the GA to obtain the solutions is 1500 times larger than that of the IDRs with look-ahead. Therefore, it is possible to achieve results which are to a certain extent worse than those obtained by the GA, but in time which can be considered negligible when compared to that of the GA.

On the other hand, the rollout algorithm has a significantly larger execution time than DRs evolved by DGP or any of the previous three methods. The execution time for the rollout algorithm, when using DRs evolved by DGP and only 3 jobs in the rollout horizon, is around 101 seconds, which is 1110 times slower than that of DRs evolved by DGP. Naturally, the rollout algorithm achieves much better results than any of the previous three methods, outperforming

Table 8.31: Results for the execution times obtained by the different methods

Index	Method	Min	Med	Execution time
1	DGP	12.96	13.60	0.091
2	ATC	12.45	12.45	1.899
3	GA	9.917	10.27	477.0
4	S-14	12.27	13.21	0.105
5	S-21	12.06	13.30	0.105
6	L-n-10	10.82	11.83	0.113
7	L-n-100	11.02	11.64	0.159
8	L-n-10 S-u-21	11.22	12.46	0.144
9	L-n-10 S-ul-21	10.90	12.24	0.144
10	L-n-20 S-ul-21	11.10	11.64	0.156
11	I-3	11.87	13.08	0.155
12	I-4	11.82	13.18	0.155
13	I-4 S-21	12.11	12.88	0.180
14	I-4 L-n-5	10.75	11.17	0.207
15	I-4 L-n-100	10.53	11.10	0.321
16	I-4 L-n-10 S-ul-21	10.77	11.42	0.350
17	I-4 L-n-100 S-ul-21	10.52	11.85	0.456
18	R-n-3	9.956	10.37	101.0
19	R-n-20	9.790	10.08	375.5
20	R-n-20 S-21	9.769	10.08	495.6
21	R-n-10 L-n-10	9.713	9.914	204.9
22	R-n-20 L-n-10	9.744	9.903	356.3
23	R-n-3 L-n-100	9.773	10.05	98.11
24	R-n-10 L-n-20 S-u-21	9.731	9.999	352.9
25	R-n-3 L-n-50 S-u-21	9.965	10.27	179.3
26	R-n-3 L-n-20 S-ul-21	9.810	10.10	175.9
27	R-n-10 L-n-20 S-ul-21	9.674	9.898	372.8
28	R-n-10 L-n-100 S-ul-21	9.750	9.917	1603

the minimum value of DGP by 23.2%, and the median value by 23.8%. Depending on the rollout parameter values and the combinations with other methods, the execution time will gradually increase. The best combination of methods can outperform DGP by 25% for the minimum value, and 27% for the median value, but with an execution time which is 4000 times slower than that of DRs evolved by DGP. The best combination of methods executes 10 times slower than the rollout algorithm with DRs generated by DGP, and achieves an improvement of 2.8% for the minimum value, and 3.7% for the median value. Therefore, by combining different methods with the rollout algorithm it is possible to further increase its performance, at the expense of a larger execution time.

For all the results which are included in the table, except the last one, the rollout algorithm obtained a smaller execution time than the GA. In addition, except for two experiments, the rollout algorithm always achieved significantly better results than the GA. The rollout algorithm with DRs generated by DGP achieved significantly better results than the GA, outperforming it by 1.5% for the minimum value, and 2.1% for the median value, with an execution time which is almost five times smaller. Even the combination of methods, which achieved the best improvements over the GA that amount to 2.5% for the minimum value, and 3.6% for the median value, has a smaller execution time than the GA. Therefore, for a good choice of parameters the rollout algorithm achieves significantly better results than the GA, and constructs the schedule in a smaller amount of time.

To further illustrate the relation of the execution times and the performances of the various methods, Figure 8.29 represents the relation between the minimum values and the execution times obtained for each of the methods, while Figure 8.30 represents the relation between the median values and the execution times obtained by the methods. Each point is denoted with the index of the experiment it represents from Table 8.31. Furthermore, the points which represent the Pareto front of solutions are denoted with red points, while the rest of solutions are represented by blue points. Since the execution times for different methods have largely different values, a logarithmic scale was used for the axis which represents the execution times of the methods. The figure shows how the different approaches are grouped together based on their performance and execution times. The shortest execution times, but also the worst results, are achieved by DRs evolved by DGP which is denoted with the index 1 in the figure. The group of results which use static terminal nodes, look-ahead, IDRs, or any combinations of those methods are represented by points with indices ranging from 4 to 17. These methods can be seen to cover quite a large part of the values for the *Twt* criterion, with only small differences in the execution times. Therefore, by increasing the execution time by a small extent a large improvement in the achieved results can be obtained. The static ATC rule, denoted with the index 2 achieved better results than DGP, but due to its slow execution time the approach is not competitive to the other tested methods.

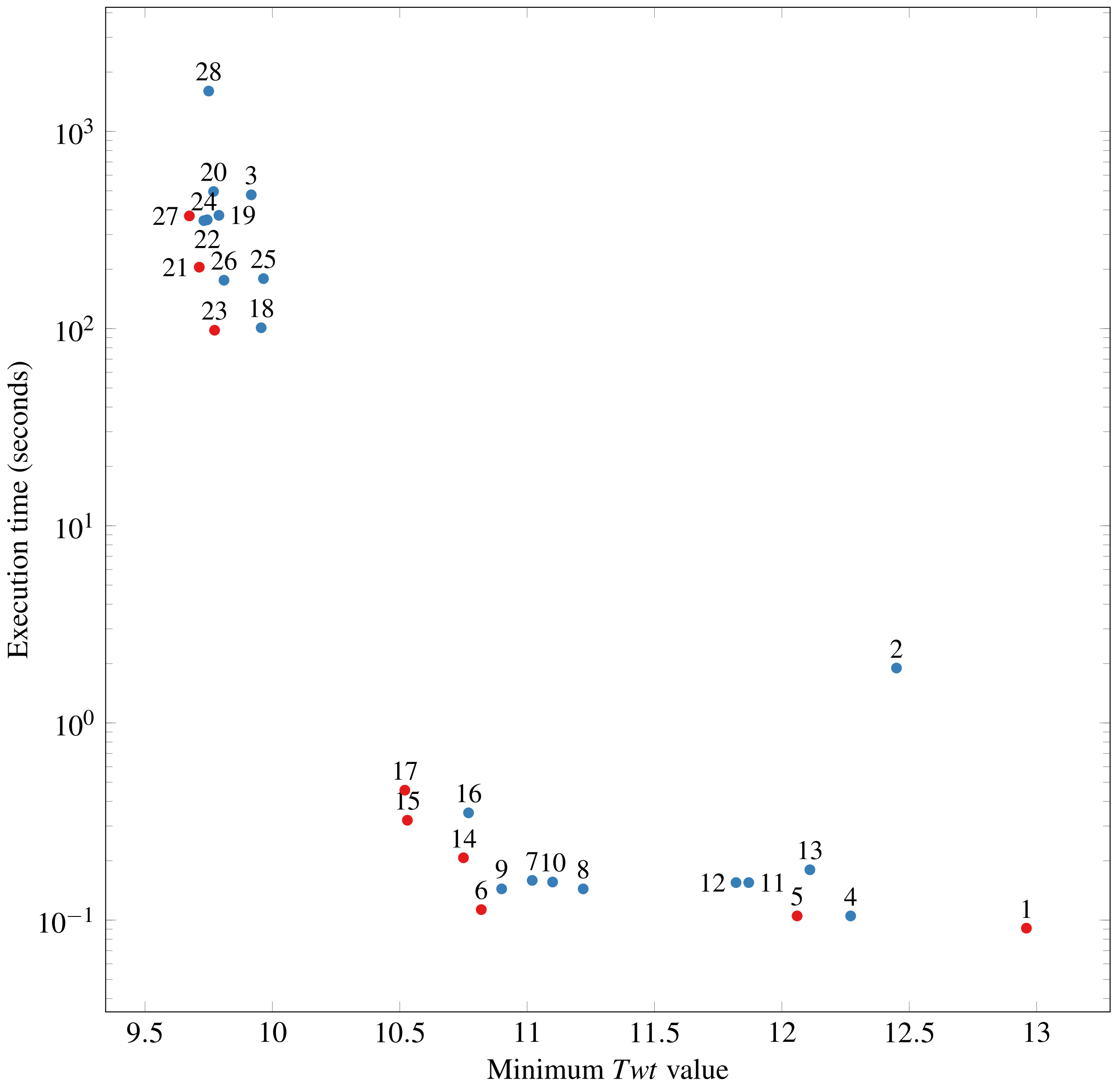


Figure 8.29: The relation between the execution times and minimum values obtained by the tested methods

The second group of results, those achieved by the rollout algorithm and combinations of it with other methods, are denoted by indices ranging from 18 to 28. The results of these methods are centred around the result achieved by the GA, which is denoted with the index 3. This group of results covers a small range of the values for the *Twt* criterion, but covers a large range for the execution time values. This means that to increase the results even for a small extent it is required to significantly prolong the execution times of the methods. Nevertheless, many of the results for the rollout algorithm do not only outperform the results obtained by the GA, but also construct the schedule in a smaller amount of time. For some parameter values, such as for those used in the experiment denoted with index 18, it is also possible to achieve results which are only to a small extent worse than those obtained by the GA, but which are constructed in a much smaller amount of time. The rollout algorithm also constructed the schedule in a smaller amount of time for the experiment denoted with index 27, for which the rollout algorithm achieved the overall best results.

Based on the presented results, several conclusions can be drawn about the tested methods. Static terminal nodes, look-ahead, IDRs, and various combinations of those approaches have all achieved better results than DGP, with an increase of the execution times that are at most five times larger than the execution time obtained by DGP. This makes these methods preferable if the execution time is equally important as the quality of the achieved results. By selecting the appropriate method and parameter values, it is possible to make a trade off between the execution time and the quality of the obtained results. In addition, because of their small execution times, it is also possible to execute several DRs for a certain problem, and select the best of the obtained solutions. On the other hand, the rollout algorithm achieved the overall best results, however, with much larger execution times. Although by using different parameter values or combinations with other methods it is possible to reduce the execution times and improve the performance of the rollout algorithm, even the lowest execution times achieved by it are still significantly larger than those of the other methods. Nevertheless, the rollout algorithm achieved better results in a smaller execution time than the GA. This makes the rollout algorithm a better choice if the quality of the obtained results is of primary importance, and the execution time is of less concern.

8.4 Analysis of the static scheduling methods

This section will give a short analysis of the four methods for adapting DRs to static scheduling conditions. The aim of this section is to give more detail on how the different methods work.

The first thing which will be analysed is the frequency with which the static terminal nodes appear in the individuals. For that purpose, 30 DRs which achieved the best performance on the test set were selected, and the number of occurrences of each static node in those individuals

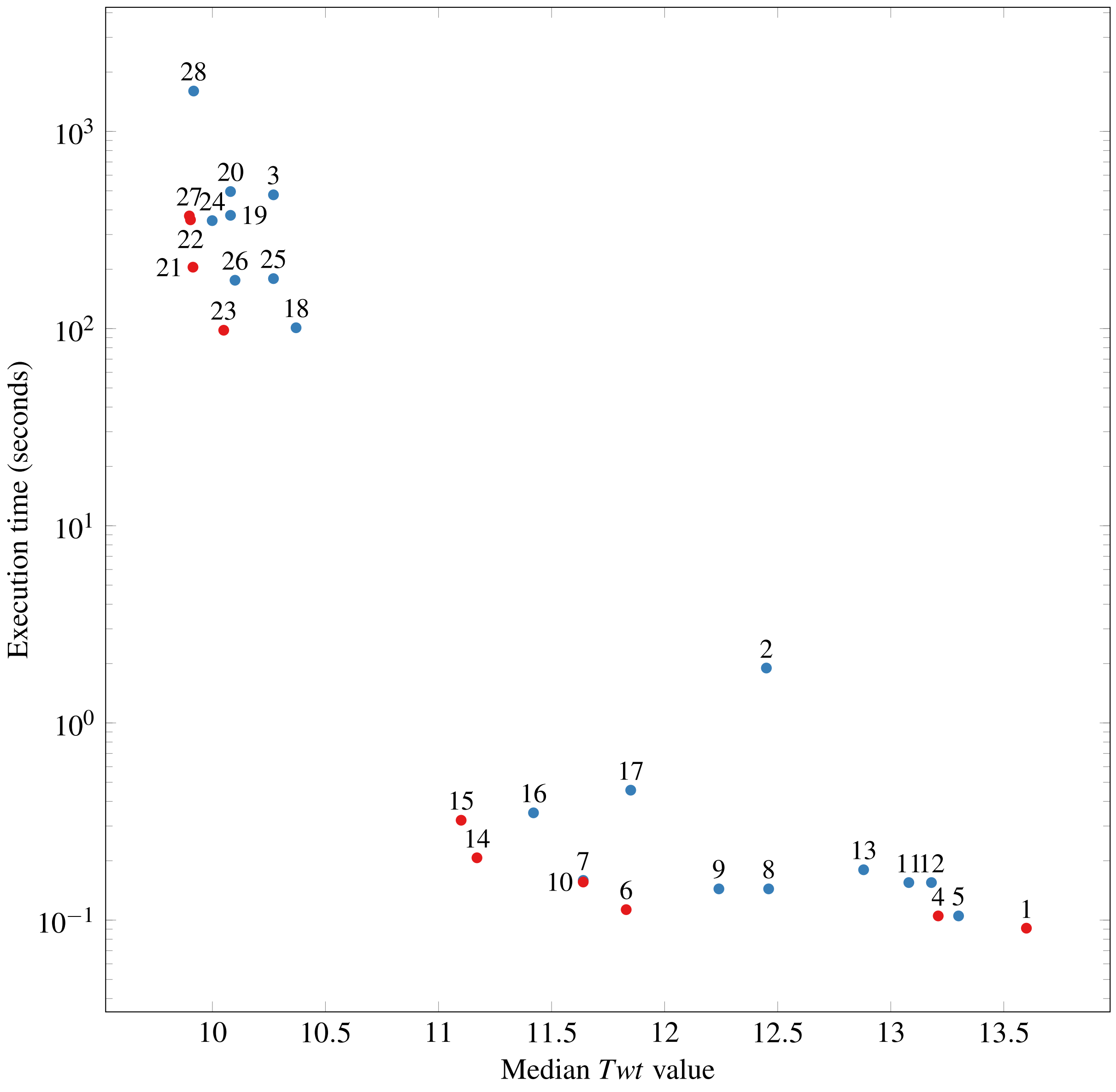


Figure 8.30: The relation between the execution times and median values obtained by the tested methods

was calculated. Although the information about the number of times a node appears in a DR can be misleading, since it is possible that the nodes appear in sub-expressions which do not provide any information to the DR, it can nevertheless give a rough overview about the usefulness of individual nodes.

Figure 8.31 shows a histogram which represents the number of occurrences of each static node. The figure clearly shows that several static terminal nodes appear more frequently, and thus seem to provide more useful information to the DRs during scheduling. The $FUTLATE$ node appears by far most frequently in the DRs, which leads to the conclusion that it is the most informative of the nodes. The $FUTLATES$ node also appears quite often, which means that the information about the possible tardiness of jobs which would be released during the execution of the current job is useful to the DRs. However, the $FUTLATEL$ node, or the other nodes which also take into account the possible tardiness of the current job if it would not be scheduled right away, were seldom used. This leads to the conclusion that the more simple approximations are already informative enough. The nodes which approximate the weighted number of tardy jobs, if the current job would not be scheduled, also appear quite often in the individuals. Out of these nodes the $WLATES$ node, which approximates whether the next released job will be tardy or not if the currently considered job is scheduled for execution, appears most often. For this group of nodes it is evident that the more complex nodes appear more often, such as nodes which approximate the difference of the weighted number of tardy jobs between the current job and all those which would be released during the execution of the current job. Based on the previous observations it is evident that the nodes which approximate the tardiness of jobs appear most often and seem to be the most informative.

Out of the remaining nodes the most informative seem to be the ones providing information about the number of jobs that will be released during the execution of the current job ($NREL$, $NRELM$, $NSHORT$), the ones which provide information about the slack of the next job which will be released ($SLNEXT$, $SLNEXTM$), the ones which provide the information about the time until the next job will be released ($TTAR$) and the information about the machine load ($MLOAD$). Out of these nodes, the $NREL$ node appears eight times, while the others appear three to five times. Therefore, these nodes appear a lot less often than some of the nodes which provide information about the tardiness of jobs, which probably means that they provide less useful information to the DRs.

The occurrence frequency of IDR nodes will be analysed in the same way, meaning that the 30 IDRs, which achieved the best results on the test set regardless of the IDR node set which was used to generate them, were collected and the number of occurrences of each IDR node was calculated. Figure 8.32 represents a histogram of the total number of times that each IDR node occurred in the selected IDRs. Clearly it can be seen that the *FLOWTIME* and *JOBFINISH* nodes, which do not provide any information about the tardiness, appear the

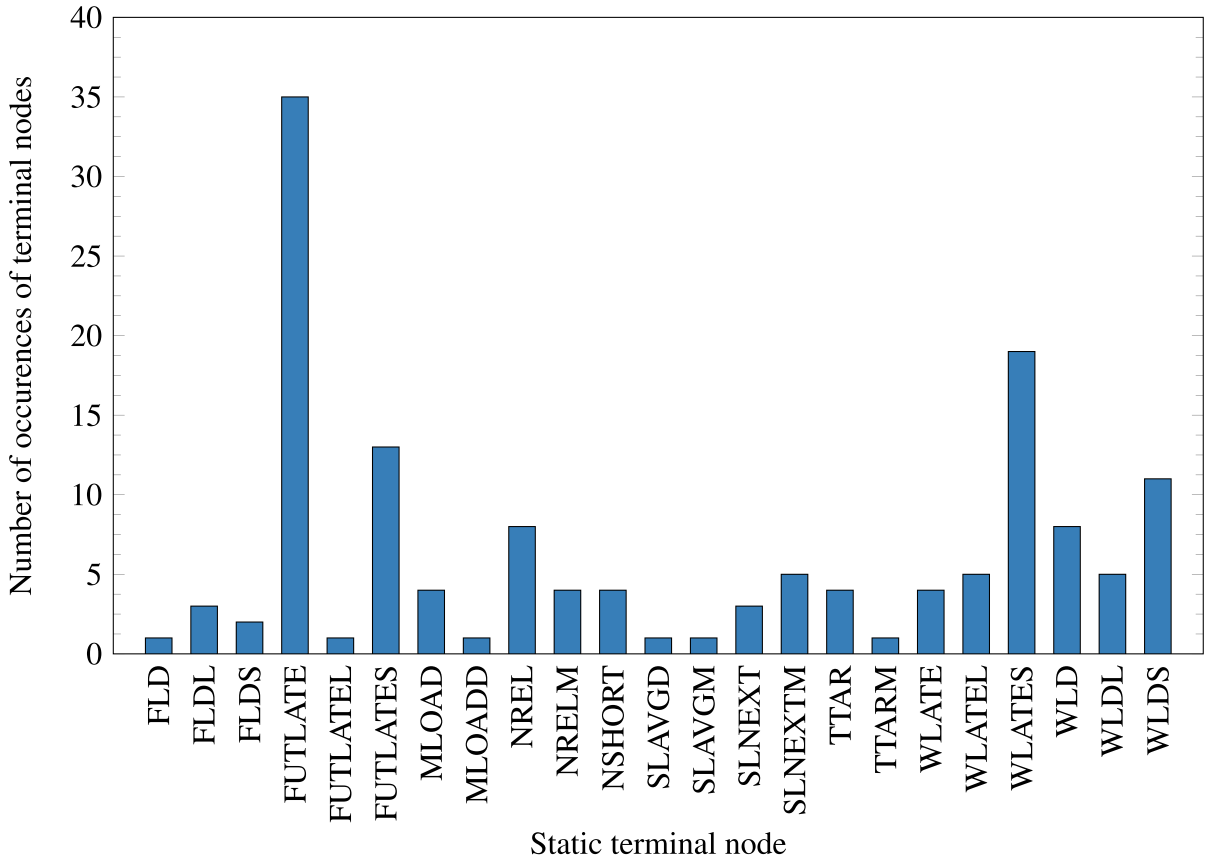


Figure 8.31: The frequency of static terminal nodes in the generated DRs

least number of times. The *LATENESS* node is also used quite rarely by the IDRs, which can lead to the conclusion that it does not provide very useful information. On the other hand, the *INDWTARD* node appears most often in the IDRs. This node seems to be useful since it directly supplies the information about the weighted tardiness of a job in the previously constructed schedule. The *LATE* function node also appears very often, meaning that allowing for parts of the DR to specialise for tardy jobs, while others to specialise for jobs which are not tardy, provides to be beneficial for the performance of the IDRs. The *NLATE* node is also frequently used, thus the IDRs also seem to benefit from the information on how many jobs were tardy in the previous schedule. The *INDLATE* and *INDTARD* nodes are used to a certain extent less than the three previously mentioned nodes. The reason for this seems to be the absence of the job weight in their calculation, which is also the reason why the *INDWTARD* appears more often than these two nodes. Finally, the *TARDINESS* node appears in even less occasions, meaning that the information about the total weighted tardiness of the previously constructed schedule does not provide useful information to the IDRs. Based on the previous observations and the number of occurrences of individual nodes, the *INDWTARD*, *INDTARD*, *INDLATE*, *NLATE*, and *LATE* nodes seem to contain the most useful information for the IDRs.

For the IDRs it is also interesting to analyse the number of times the rules recreate the

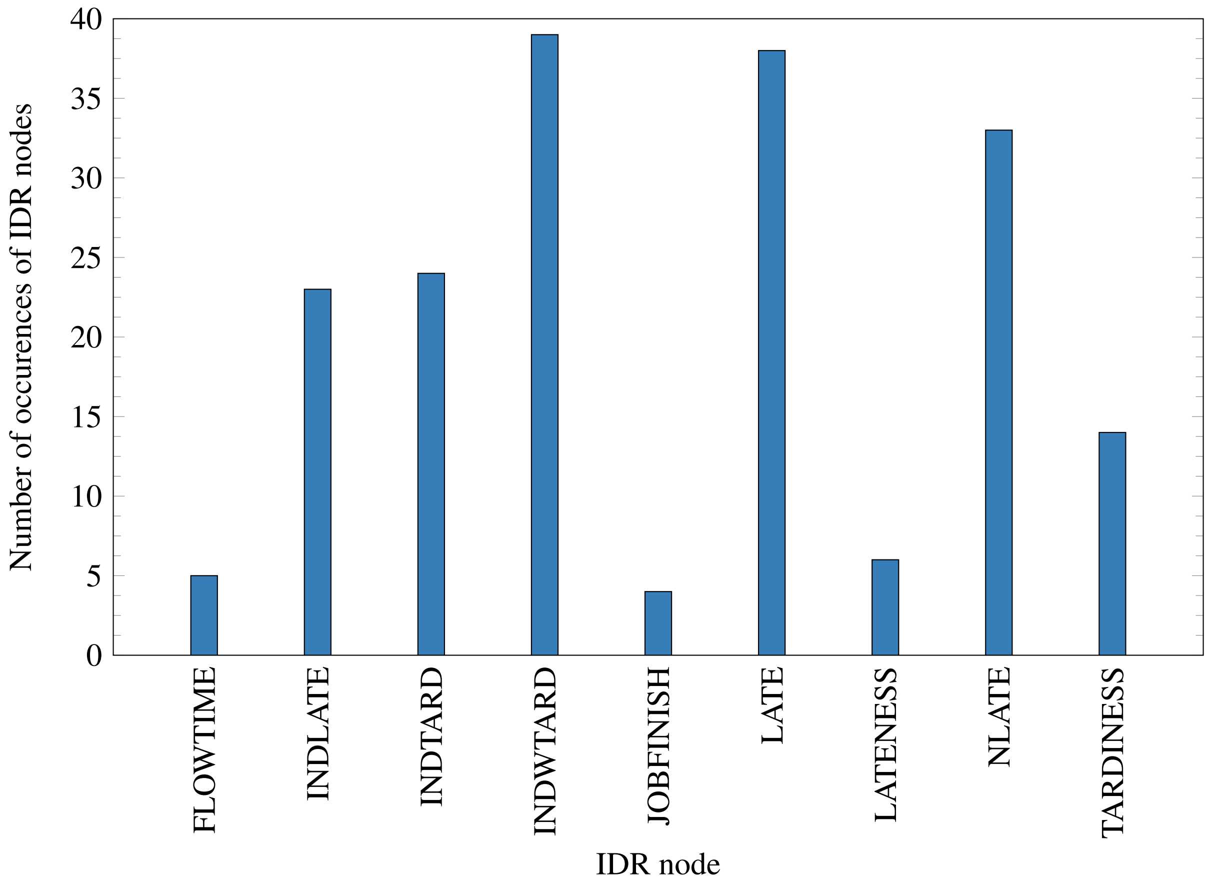


Figure 8.32: The frequency of IDR nodes in the generated IDRs

schedule. This analysis will use the same 30 best IDRs which were used for the previous analysis. On average, when all problem instances and all IDRs are considered, the IDRs created the schedule 2.465 times. The last schedule which is created by the IDR, but not returned as the results, was also included in the calculations. The value shows that, on average, the IDRs create the schedule two or three times. By analysing the number of created schedules for each problem instance independently, it was shown that for the easier instances the IDRs can find the best schedule immediately in the first iteration. However, the IDRs still need to validate that they can not improve the schedule, which in the end means that for the easier problem instances they will create the schedule two times. For the more difficult problem instances, the schedule is usually created three or four times, with the maximum number of created schedules being seven. This shows that the IDRs do not have the tendency to recreate the schedule many times, but that they are rather capable of performing all improvements in only a few iterations. Also, during the analysis it was observed that a few of the better IDRs usually recreated the schedule only a few times. Therefore it was tested whether there is a correlation between the fitness of the IDRs and the number of times they recreate the schedule. To test the correlation between those two variables, the Spearman's rho test was performed, and the values of $\rho = -0.097$ and $p = 0.612$ were obtained. Based on these two values it can be concluded that there exists no correlation between the two variables.

To further illustrate the behaviour of the different methods, and outline their strengths and weaknesses, the schedules they created for a simple scheduling problem instance will be shortly analysed. The details about the problem instance which will be used for the analysis are presented in Table 8.32.

The schedules created by the dynamic DR and various static DR methods are presented in Figure 8.33. Subfigure 8.33a represents the schedule which is created by the best DR which was created by GP for dynamic scheduling conditions. From the schedule it can immediately be seen that the DR performs two bad decisions, which lead to an increased value of the total weighted tardiness criterion. The first bad decision is that job j_{11} was scheduled before job j_{10}. Since the execution time of job j_{10} would be only one time unit, it would be more beneficial to first execute this job and delay the execution of job j_{11} until job j_{10} is finished. Unfortunately, job j_{11} is released prior to job j_{10}, and the DR has no means of detecting that very soon a new job will be released, which should have a higher priority of being scheduled. Therefore, job j_{10} will have to wait until job j_{11} finishes with its execution, so that it can execute on machine M_0. This will consequentially lead to a larger tardiness value. The second bad decision can be observed at the end of the schedule, where the DR scheduled job j_4 on machine M_2. This job could have been scheduled at a much earlier time on M_0 to reduce the tardiness. However, the DR chose to rather schedule job j_4 on the machine for which it has the minimum processing time, and keep the machine M_0 free in case that other jobs, which prefer this machine, would

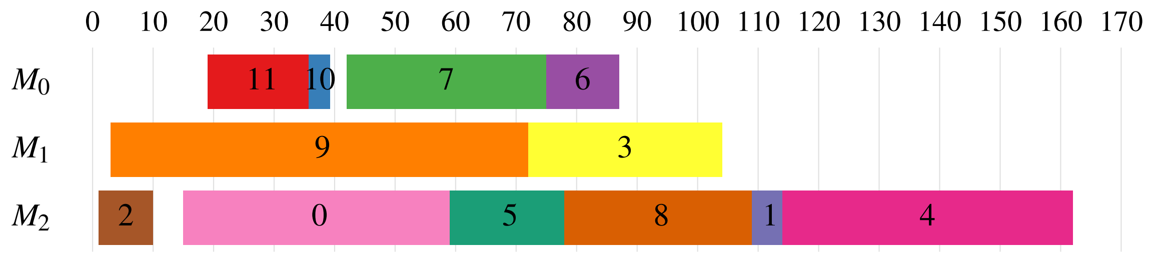
Table 8.32: Properties of the problem instance used for analysis

Job index j	r_j	d_j	w_j	p_{0j}	p_{1j}	p_{2j}
0	15	15	0.9	73	35	45
1	98	104	0.07	62	64	5
2	1	43	0.87	47	89	9
3	25	76	0.06	58	31	24
4	47	96	0.06	65	75	47
5	31	70	0.26	38	82	19
6	59	84	0.78	12	93	92
7	42	78	0.94	33	24	62
8	56	102	0.63	92	62	31
9	3	64	0.33	99	70	92
10	21	27	0.49	1	80	18
11	19	43	0.95	18	15	96

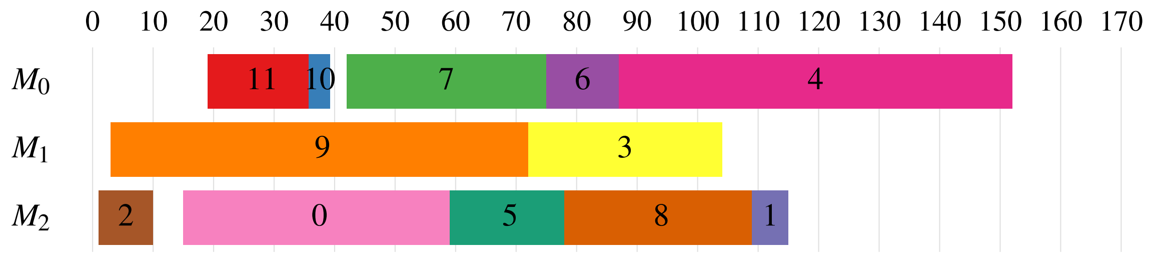
arrive into the system. In the end, the schedule generated by the dynamic DR achieves the total weighted tardiness value of 0.596.

Subfigure 8.33b represents the schedule created by a selected DR with static terminal nodes. The figure illustrates that the DR with static terminal nodes was able to fix only one of the bad decisions which were performed by the dynamic DR. This DR performed a better decision at the end of the schedule, by scheduling job j_4 on machine M_0 rather than on machine M_2. Therefore, the static terminal nodes provide useful information which the DR uses to determine that by scheduling job j_4 on machine M_0 there would be no consequences for the future. The total weighted tardiness of this schedule is 0.590. Therefore, the DR with static terminal nodes achieved a slight improvement over the dynamic DR.

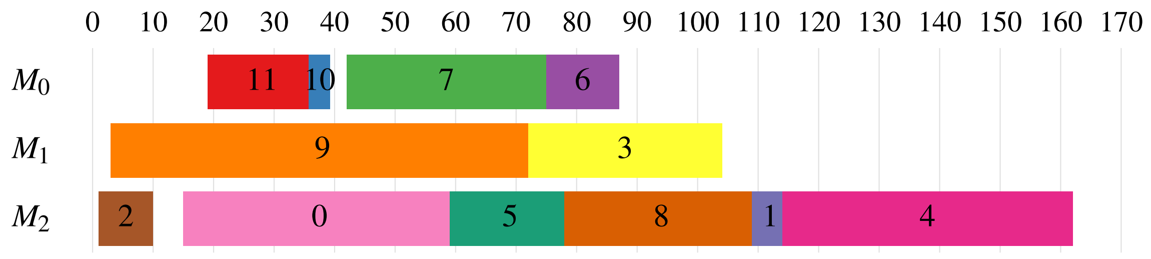
The IDR which was selected created the schedule three times. The first schedule it created is presented in Subfigure 8.33c. The figure shows that the schedule created by this IDR is the same as the one created by the dynamic DR. It is expected that in the first iteration the IDR will perform equally well or worse than the dynamic DR, since the IDR did not yet have access to any additional static information. Subfigure 8.33d denotes the schedule which was created in the second iteration by the IDR. The figure shows that the IDR created the same schedule as the DR with static terminal nodes. It seems that the IDR was able to capture the tardiness of this job in the last schedule, and use this information to schedule it to another machine in the second iteration. In the third iteration the IDR obtained a schedule of the same fitness value,



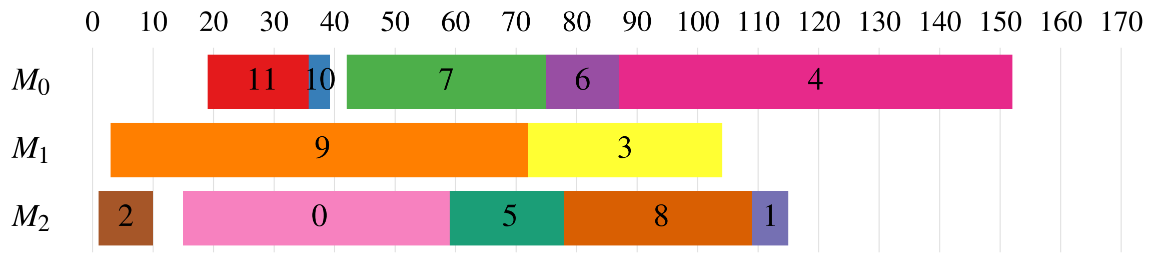
(a) Schedule generated by the best dynamic DR



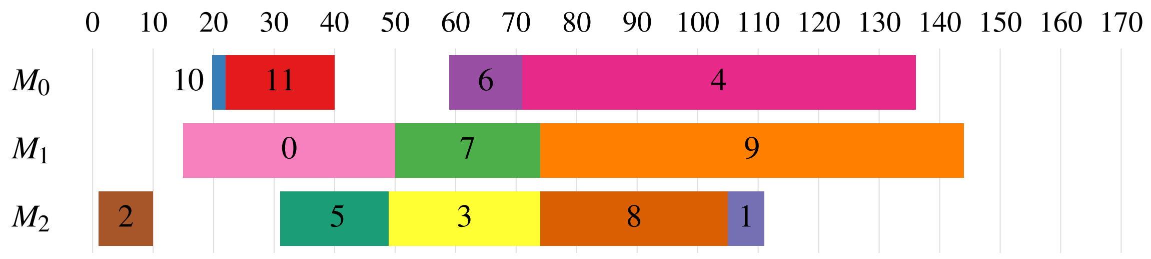
(b) Schedule generated by a DR with static terminal nodes



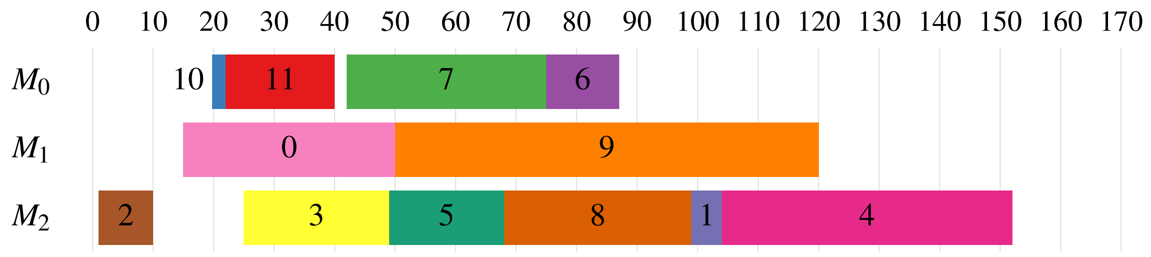
(c) Schedule generated by an IDR in the first iteration



(d) Schedule generated by an IDR in the second iteration



(e) Schedule generated by a DR with look-ahead



(f) Schedule generated by the rollout algorithm

Figure 8.33: Schedules generated by the static DR methods

and therefore the procedure was terminated. In the end the IDR obtained the same schedule as the DR with static terminal nodes. This leads to the conclusion that both of the methods are similarly expressive. In the end, the IDR created a schedule with a Twt value of 0.590.

Subfigure 8.33e shows the schedule which was created by a DR which uses look-ahead. The figure shows that the DR was able to correct both bad decisions which the dynamic DR made. First, it was able to determine that job j_{11} should not be scheduled on machine M_0 at the very moment it arrives into the system, but with the help of look-ahead it determined that a job which has a higher priority will very soon arrive into the system, and therefore it delayed the scheduling of job j_{11}. When job j_{10} arrives into the system, it is immediately scheduled on machine M_0, and after it has finished with its execution, job j_{11} is executed. At the end of the schedule, the DR also determined that it would be better to schedule job j_4 immediately on machine M_0. Look-ahead can also be helpful in this situation, since the DR will have a clear oversight of the look-ahead horizon and will be aware that no new jobs will be released during it, which means that it can prioritise the available jobs and schedule them without any drawbacks. Furthermore, from the figure it is evident that the DR did some other changes to the schedule. The schedule shows that the DR introduced several idle times into the schedule to keep the machines free for high priority jobs which arrive in the near future. Therefore, the DR kept machine M_1 free until job j_0 arrives into the system, since this job has a high weight and executes the fastest on machine M_1. The same happens for machine M_0, where the DR does not schedule job j_7, but rather waits for job j_6, and schedules job j_7 on machine M_1 on which it achieves the fastest processing time. For machine M_2 the same thing can be observed when scheduling jobs j_3 and j_5. Based on these observations, it can be concluded that the look-ahead provides DRs with more information than the previous two methods, and therefore allows them to create better schedules. Even though the DR introduced a lot of idle times into the system, it nevertheless achieved a better performance than the previous two methods, with the Twt value being 0.574.

Finally, Subfigure 8.33f represents the schedule created by the rollout algorithm. The schedule is very similar to the one obtained by the DR with look-ahead, however the rollout algorithm tries to reduce the amount of idle times which are introduced in the schedule. Once again for machine M_0 it is preferred if job j_{10} is executed prior to job j_{11}. The rollout algorithm also determined that a certain amount of idle time should be introduced on machine M_1 so that job j_0 can start immediately with its execution, to minimise the tardiness caused by it. However, the rollout algorithm determined that it was better to execute job j_7 on machine M_0 as soon as it arrives, since this will allow for job j_9 to be executed immediately on machine M_1. This will lead to a very small increase in the tardiness value for job j_6, but will allow for job j_9 to finish significantly earlier, and thus greatly reduce its tardiness value. For machine M_2 the algorithm determined that it is better to immediately start executing job j_3, since it will not only prevent job j_5 of being late, but will additionally reduce the the tardiness of jobs j_8 and j_1. In this schedule, job j_4 finishes at a later moment in time with its execution than it was in the case when a DR with look-ahead was used. However, since the job has a very small weight, it will not have a large effect on the *Twt* value of the entire schedule. In the end, the *Twt* value of this schedule amounts to 0.510, which is the best value achieved by any of the methods.

Based on all the previous observations, it can be concluded that the rollout algorithm has the best overlook on the problem, and is thus able to achieve the best results. This can best be seen in the comparison with look-ahead, which has a more limited overview on the problem. The DR with look-ahead tried to give higher priority to jobs which had a shorter execution time on the current machine, and therefore introduced several idle times to keep the machines free for those job. Unfortunately, this had a negative effect on the later parts of the schedule, where these decisions lead to an increased tardiness for some other jobs. Naturally, even with look-ahead it is quite hard for the DR to predict all the effects a scheduling decision could have on the future of the system. However, the rollout algorithm can try out various decisions at each moment when a scheduling decision needs to be performed, and approximate the influence of this decision on the rest of the schedule with a good DR. This gives the rollout algorithm an unparalleled overview of the problem, and allows it perform decisions at the beginning of the schedule, which will not have a negative influence on the latter parts of the schedule.

For the rollout algorithm it is additionally interesting to analyse whether there is a connection between the quality of the results produced by the rollout algorithm and the quality of the DR which was used by it for the approximations. In order to test this, the rollout algorithm with dynamic DRs was used. By performing the Spearman's rho test, the values $\rho = 0.487$ and $p = 0.00034$ were obtained. Therefore it can be seen that there is a positive correlation between the quality of the rollout algorithm and the DR it uses. However, the correlation is not quite strong to accept it as a general rule. This can be also seen from the results, since several better results for the rollout algorithm are achieved when better DRs are being used, however there are also cases where the rollout algorithm achieves quite good results even if the DR it uses performs poorly by itself. The best performance in this case was nevertheless achieved when using the DR which also individually achieved the best result.

8.5 Conclusion

Even though DRs are mostly applied for scheduling problems under dynamic conditions, they can nevertheless be used to solve static scheduling problems as well. They offer several benefits over metaheuristic methods, which are commonly used for solving scheduling problems under static conditions. The first benefit is that DRs create good solutions much faster than the different metaheuristic methods. Therefore, if it is required to obtain a relatively good solution in a small amount of time, DRs represent a better option. The second benefit of DRs is that they create schedules incrementally, meaning that it is possible to start executing the beginning of the schedule while the rest is still being constructed by the DR. Naturally, to achieve better results with DRs, they need to be adapted for solving scheduling problems under static conditions.

The purpose of this chapter was to analyse different ways of adapting automatically generated DRs to improve their performance for the static scheduling problem. In the chapter it was analysed how additional static terminal nodes, look-ahead, IDRs, and the rollout algorithm improve the performance of DRs for solving scheduling problems under static conditions. For each of the tested methods an in depth analysis on the influence of different parameters was additionally performed. The obtained results were compared to those achieved by DRs evolved for scheduling under dynamic conditions, and also to results achieved by a GA. Finally, an analysis of the execution times of the different methods was also performed.

The results have shown that the various methods offer different levels of improvement over the dynamic DRs, and also obtain vastly different execution times. The rollout algorithm achieved the overall best results and even significantly outperformed the results achieved by the GA. However, this procedure also had the highest execution time out of the tested methods. Nevertheless, in most cases it obtained better results than the GA, and required less time to construct the schedule. The other methods did not achieve better results than the GA. On the other hand, their execution time is almost negligible when compared to that of the GA, and in most cases they achieved better results than the dynamic DRs. Out of the remaining three methods, look-ahead has proven to achieve the best results. Through additional tests it was also demonstrated that combining different methods, in most cases, leads to even better performance of the DRs.

Based on the previously outlined observations, it can be concluded that the tested static methods achieved improved results over the DRs generated for solving scheduling problem under dynamic conditions. The proposed rollout algorithm outperformed even the results achieved by the GA. Since the tested methods have different execution times, it is possible to select the one which offers the best trade-off between the quality of the obtained results and the time needed to create the schedule. As the tested methods obtained a quite good performance, there is a lot of motivation for further extending this research to additionally improve the execution times and the obtained results. One possibility for further research would be to develop novel methods of adapting DRs for static scheduling problems, and compare them with the results obtained in this chapter. The second line of research would be to improve the applied methods to obtain better results in less time. For example, it would be interesting that in each step the rollout algorithm does not create the entire schedule by using a DR, but rather a part of the schedule, so that the scheduling decision is based only on that partially created schedule. This should improve the execution time of the method, however, the question is how it would effect the performance of the method.

Chapter 9

Conclusion

Scheduling is a decision-making process which can be observed in many real world situations. Because it is widely used in different areas, it has become an extensively researched field of science. However, since scheduling problems are so widespread, many different types, conditions, and criteria exist for those problems. This hinders the possibility of developing a single method which could be used and would perform well for all problems. Because of that reason, a great deal of research has focused on applying different metaheuristic methods for solving scheduling problems. These methods have the benefit that they can be applied for various optimisation problems, with only small adjustments needed to be performed on the core method. On the other hand, these methods additionally have a serious drawback, which is that all the information about the scheduling problem needs to be available, for them to be able to construct solutions. However, for many scheduling problems this will not be the case, but rather the information about the system will become known during its execution.

Dispatching rules, on the other hand, are simple problem specific heuristics which can be used for solving dynamic scheduling problems. However, since they are problem specific, new dispatching rules need to be designed for different scheduling environments, conditions and criteria. The process of manually designing new dispatching rules is a tedious and time consuming process. For that reason, an increasing amount of research is performed to develop methods for automatically generating dispatching rules. Among the most commonly applied methods for that purpose are genetic programming, and similar evolutionary computation methods. Genetic programming is an evolutionary computation method which has the ability to design complex expressions, which can be applied as dispatching rules. Because of its great potential for designing new dispatching rules, a great number of studies applied genetic programming to automatically generate new dispatching rules for various scheduling environments and criteria.

9.1 Achieved contributions and main conclusions

This thesis focused on developing new dispatching rules for the unrelated machines scheduling environment, by using genetic programming. The main objective of the thesis was to enhance the performance of dispatching rules generated by genetic programming. This objective was achieved by combining genetic programming with various other methods, but also by introducing certain changes into dispatching rules. It should be stressed out that, although this thesis dealt only with scheduling problems in the unrelated machines environment, most of the methods and approaches used in this thesis can be applied not only for other scheduling environments as well, but also for problems which can be solved in a similar manner by using a priority function. The rest of this section will list the major contributions achieved in the thesis, and give conclusions about each of the achieved contributions.

9.1.1 Design of dispatching rules for simultaneous optimisation of multiple criteria

In the thesis, four different multi-objective and many-objective genetic programming algorithms were used to design dispatching rules for optimising several criteria simultaneously. The selected methods were applied on various multi-objective and many-objective scheduling problems, in which the number of optimised criteria ranged from three to nine. Dispatching rules which were generated by the applied methods were compared with several manually designed dispatching rules from the literature. The obtained results demonstrate that the applied genetic programming algorithms generated dispatching rules which in most cases perform better than manually designed dispatching rules. Therefore, the multi-objective and many-objective methods are capable of obtaining good dispatching rules for various combinations of scheduling criteria.

Based on the obtained results, it can be concluded that, by using genetic programming, it is possible to create dispatching rules which perform well on several criteria simultaneously. Although the methods have shown to be sensitive to the number of objectives which were optimised, they were much more sensitive to the combination of criteria which was optimised. Nevertheless, even for larger criteria combinations the methods obtain good dispatching rules. Although this thesis demonstrated the possibility of obtaining dispatching rules for optimising various combinations of scheduling criteria, there is still much potential to further improve the performance of the applied algorithms.

9.1.2 Designing ensembles of dispatching rules

In order to improve the performance of dispatching rules designed by genetic programming, the thesis proposed the application of several ensemble learning approaches for solving scheduling problems under dynamic conditions. In addition to the two proposed ensemble learning approaches, three prominent approaches were selected from the literature and additionally used to create ensembles of dispatching rules. To evaluate the performance of the tested ensemble learning methods, they were used to construct ensembles for four scheduling criteria. The obtained results demonstrate that the constructed ensembles of dispatching rules outperformed individual rules designed by genetic programming. Out of the tested ensemble learning approaches, the proposed SEC approach achieved the best performance. On the other hand, the proposed ESS method improved the performance of ensembles obtained by the other approaches in several occasions.

The results obtained by ensembles of dispatching rules significantly outperform those achieved by individual manually or automatically designed dispatching rules, on all four scheduling criteria. These results demonstrate the potential which ensemble learning approaches provide in obtaining better results. Another benefit of ensembles is that they can be constructed in various ways, which allows for new methods of creating ensembles to be defined quite easily. This is especially true for the proposed ensemble learning approach, which allows for much flexibility in defining how the ensembles are constructed. Therefore, by defining novel ways of creating the ensembles, it is possible that the performance of the approach could be increased even further.

9.1.3 Procedure for the selection of dispatching rules based on problem instance characteristics

By using genetic programming it is possible to obtain a large number of new dispatching rules. However, it is not known in advance which of these rules are best suitable for solving unseen scheduling problems. Even though genetic programming is used to create high quality dispatching rules, there will still be certain problem instances and situations for which these rules will perform poorly. To solve this problem, a procedure for the selection of automatically generated dispatching rules based on the characteristics of problem instances was proposed in this thesis. The proposed procedure can be applied both in situations when all problem instance characteristics are known in advance, or if they need to be approximated during the execution of the system. In both cases the proposed method has demonstrated that it can select appropriate dispatching rules for different problem instances, and therefore achieve a better performance than if only a single dispatching rule would be used.

Selecting dispatching rules based on problem characteristics has shown to be a viable method to adapt for different problems which could occur during the execution of the system. Although the proposed procedure can select the appropriate dispatching rules for different problem instances, it consists of a wide range of parameters which need to be fine tuned in order for the procedure to achieve good performance. Nevertheless, selecting the appropriate dispatching rules for the currently considered problem instance is an important issue in dynamic scheduling environments which needs to be dealt with. Therefore, further research should be conducted in this area to design more robust methods, and possibly combine them with genetic programming simultaneously evolve dispatching rules, and the decision functions which determine the dispatching rule that should be applied for the current problem instance.

9.1.4 Developing dispatching rules for the static scheduling environment

Since dispatching rules are mostly used in dynamic scheduling environments, they are designed in a way that they use only dynamic information of the problem. The thesis proposed various ways to adapt automatically generated dispatching rules for the static scheduling environment. In addition, the thesis also proposed the use of the rollout algorithm in combination with automatically generated dispatching rules, to obtain results which are comparable or even better than those of genetic algorithms. Various combinations of the proposed methods were also tested in the thesis to further improve the results. The obtained results demonstrate that the various methods provide different trade-offs between the quality of the results, and the time needed to create the schedule. In addition, results obtained by the proposed methods are compared to the results achieved by a genetic algorithm. This comparison determined that certain procedures achieved equally good or better results than genetic algorithms, in a smaller or similar time frame. Other methods obtained results which are to a certain extent worse than those obtained by the genetic algorithm. However, those methods can obtain the results in a considerable smaller amount of time than the genetic algorithm.

The obtained results show that dispatching rules have a lot of potential of being applied for the static scheduling environment. The results demonstrated that the adapted dispatching rules achieved similar or even better performance than the genetic algorithm, but additionally offer several benefits like faster execution times and incremental construction of the schedules. Because of these reasons, the adapted dispatching rules could have potential to become a preferred method for solving scheduling problems under static conditions. Therefore, it is important to continue the research of adapting dispatching rules for static scheduling conditions, to obtain methods which can achieve even better performance in a smaller amount of time.

9.2 Future research

This thesis has dealt with several open issues in the field of automatic generation of dispatching rules. Regardless of this, there are still many open research areas which need further consideration. This section will outline several possible future research topics. Some research possibilities have already been mentioned in the conclusions of individual chapters, therefore they will not be repeated in this section.

One possible research direction is concerned with testing different schedule generation schemes which are used by the dispatching rules. Throughout this thesis one main schedule generation scheme was used, however it would be interesting to analyse how different schedule generation schemes would influence not only the quality of the generated dispatching rules, but also the interpretability of the generated dispatching rules. A possible schedule generation scheme could use two priority functions, one to determine the job that should be scheduled, and the other to determine the machine on which it should be scheduled. Another schedule generation scheme could, for example, only select the job which should be scheduled, but the job would always be scheduled on the machine on which it achieves its minimum execution time.

The interpretability of dispatching rules is another important topic which is also quite often in the focus of many studies, since genetic programming usually produces rules which are quite hard to interpret. Interpretability of dispatching rules is important since it is easier to extract information from dispatching rules which are easier to interpret. In addition, dispatching rules which are easier to interpret are more likely to be applied in real world environments, since it is easier to deduce the way in which they perform scheduling decisions. Therefore, obtaining simpler dispatching rules is certainly another field in which a lot of research will be focused in the future.

The scheduling problem which was considered in this thesis did not contain any additional constraints or conditions, aside for the release times of the jobs. However, scheduling problems usually contain several additional constraints, like setup times or precedence constraints. Some research was already concerned with scheduling problems which included additional constraints, however, the studies usually focused on problems with only one constraint. Therefore, it would be interesting to investigate the creation of dispatching rules for scheduling problems which include several constraints. Here it would also be interesting to put more emphasis on solving the unrelated machines environment scheduling problem with batch processing.

In real scheduling environments it is often possible that not all parameters of the system are deterministic. For example, it is possible that processing times of jobs are not known in advance, but that rather only an approximation of the real processing times is available. The real processing time would become available only when the job finishes with its execution. Most of the research on the automatic design of dispatching rules does not deal with such uncertainties, and only a few recent studies began researching this problem into more depth.

Genetic programming uses a predefined set of terminal nodes to construct the priority function used by the dispatching rule. However, providing either a too large or a too small number of terminal nodes can have a negative effect on the performance of genetic programming. In order to determine which terminal nodes are important, a great deal of experiments would need to be performed. Therefore, it would be beneficial if genetic programming could perform a selection of useful terminals during the evolution process, and thus eliminate all the unnecessary terminal nodes. In addition, it could also prove useful if genetic programming could determine which expressions appear most commonly in dispatching rules, and replace those expressions with new terminal nodes. This could improve the performance of genetic programming since it would not need to create entire expressions each time, but could rather use them as a terminal node.

Bibliography

[1] Pinedo, M. L., Scheduling: Theory, algorithms, and systems: Fourth edition. Boston, MA: Springer US, 2012, Vol. 9781461423614, available at: http://link.springer.com/10.1007/978-1-4614-2361-4

[2] Cheng, V., Crawford, L., Menon, P., "Air traffic control using genetic search techniques", in Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328), Vol. 1. IEEE, 1999, pp. 249–254, available at: http://ieeexplore.ieee.org/document/806209/

[3] Hansen, J. V., "Genetic search methods in air traffic control", Computers & Operations Research, Vol. 31, No. 3, Mar. 2004, pp. 445–459, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305054802002289

[4] Dimopoulos, C., Zalzala, A., "Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons", IEEE Transactions on Evolutionary Computation, Vol. 4, No. 2, Jul. 2000, pp. 93–113, available at: http://ieeexplore.ieee.org/document/850651/

[5] Ouelhadj, D., Petrovic, S., "A survey of dynamic scheduling in manufacturing systems", Journal of Scheduling, Vol. 12, No. 4, Aug. 2009, pp. 417–431, available at: http://link.springer.com/10.1007/s10951-008-0090-8

[6] Sarin, S. C., Varadarajan, A., Wang, L., "A survey of dispatching rules for operational control in wafer fabrication", Production Planning & Control, Vol. 22, No. 1, Jan. 2011, pp. 4–24, available at: http://www.tandfonline.com/doi/abs/10.1080/09537287.2010.490014

[7] Kofler, M., Wagner, S., Beham, A., Kronberger, G., Affenzeller, M., "Priority Rule Generation with a Genetic Algorithm to Minimize Sequence Dependent Setup Costs", in Computer Aided Systems Theory - EUROCAST 2009: 12th International Conference, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, Revised Selected Papers, Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A., (ed.).

Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 817–824, available at: http://link.springer.com/10.1007/978-3-642-04772-5_105

[8] Petrovic, S., Castro, E., “A genetic algorithm for radiotherapy pre-treatment scheduling”, in Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II, Di Chio, C., Brabazon, A., Di Caro, G. A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A. G. B., Urquhart, N., Uyar, A. Ş., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 454–463, available at: https://doi.org/10.1007/978-3-642-20520-0_46

[9] Hart, E., Ross, P., Corne, D., “Evolutionary Scheduling: A Review”, Genetic Programming and Evolvable Machines, Vol. 6, No. 2, Jun. 2005, pp. 191–220, available at: http://link.springer.com/10.1007/s10710-005-7580-7

[10] Sels, V., Gheysen, N., Vanhoucke, M., “A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions”, International Journal of Production Research, Vol. 50, No. 15, Aug. 2012, pp. 4255–4270, available at: http://www.tandfonline.com/doi/abs/10.1080/00207543.2011.611539

[11] Kaban, A., Othman, Z., Rohmah, D., “Comparison of dispatching rules in job-shop scheduling problem using simulation: A case study”, International Journal of Simulation Modelling, Vol. 11, No. 3, 2012, pp. 129–140.

[12] Tseng, L.-Y., Chin, Y.-H., Wang, S.-C., “A minimized makespan scheduler with multiple factors for Grid computing systems”, Expert Systems with Applications, Vol. 36, No. 8, Oct. 2009, pp. 11 118–11 130, available at: http://linkinghub.elsevier.com/retrieve/pii/S0957417409002401

[13] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., Freund, R. F., “Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems”, Journal of Parallel and Distributed Computing, Vol. 59, No. 2, Nov. 1999, pp. 107–131, available at: http://linkinghub.elsevier.com/retrieve/pii/S0743731599915812

[14] Branke, J., Nguyen, S., Pickardt, C. W., Zhang, M., “Automated Design of Production Scheduling Heuristics: A Review”, IEEE Transactions on Evolutionary Computation, Vol. 20, No. 1, Feb. 2016, pp. 110–124, available at: http://ieeexplore.ieee.org/document/7101236/

[15] Nguyen, S., Mei, Y., Zhang, M., “Genetic programming for production scheduling: a survey with a unified framework”, Complex & Intelligent Systems, Vol. 3, No. 1, Mar. 2017, pp. 41–66, available at: http://link.springer.com/10.1007/s40747-017-0036-x

[16] Koza, J. R., “Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems”, Stanford, CA, USA, Tech. Rep., 1990.

[17] Jakobović, D., “Raspoređivanje zasnovano na prilagodljivim pravilima”, 2005.

[18] Nguyen, S., “Automatic design of dispatching rules for job shop scheduling with genetic programming”, 2013.

[19] Hunt, R. J., “Genetic programming hyper-heuristics for job shop scheduling”, 2016.

[20] Li, B., Li, J., Tang, K., Yao, X., “Many-objective evolutionary algorithms: A survey”, ACM Comput. Surv., Vol. 48, No. 1, Sep. 2015, pp. 13:1–13:35, available at: http://doi.acm.org/10.1145/2792984

[21] Bagchi, T. P., Multiobjective Scheduling by Genetic Algorithms. Boston, MA: Springer US, 1999, available at: http://link.springer.com/10.1007/978-1-4615-5237-6

[22] Minella, G., Ruiz, R., Ciavotta, M., “A Review and Evaluation of Multiobjective Algorithms for the Flowshop Scheduling Problem”, INFORMS Journal on Computing, Vol. 20, No. 3, Aug. 2008, pp. 451–471, available at: http://pubsonline.informs.org/doi/10.1287/ijoc.1070.0258

[23] Wang, X.-J., Zhang, C.-Y., Gao, L., Li, P.-G., “A Survey and Future Trend of Study on Multi-Objective Scheduling”, in 2008 Fourth International Conference on Natural Computation. IEEE, 2008, pp. 382–391, available at: http://ieeexplore.ieee.org/document/4667864/

[24] Sun, Y., Zhang, C., Gao, L., Wang, X., “Multi-objective optimization algorithms for flow shop scheduling problem: a review and prospects”, The International Journal of Advanced Manufacturing Technology, Vol. 55, No. 5-8, Jul. 2011, pp. 723–739, available at: http://link.springer.com/10.1007/s00170-010-3094-4

[25] Yenisey, M. M., Yagmahan, B., “Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends”, Omega, Vol. 45, Jun. 2014, pp. 119–135, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305048313000832

[26] Tay, J. C., Ho, N. B., “Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems”, Computers & Industrial

Engineering, Vol. 54, No. 3, Apr. 2008, pp. 453–473, available at: http://linkinghub.elsevier.com/retrieve/pii/S0360835207002008

[27] Nie, L., Gao, L., Li, P., Wang, X., “Multi-Objective Optimization for Dynamic Single-Machine Scheduling”, in Advances in Swarm Intelligence: Second International Conference, ICSI 2011, Chongqing, China, June 12-15, 2011, Proceedings, Part II, Tan, Y., Shi, Y., Chai, Y., Wang, G., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–9, available at: http://link.springer.com/10.1007/978-3-642-21524-7_1

[28] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Dynamic multi-objective job shop scheduling: A genetic programming approach”, in Automated Scheduling and Planning: From Theory to Practice, Uyar, A. S., Ozcan, E., Urquhart, N., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 251–282, available at: https://doi.org/10.1007/978-3-642-39304-4_10

[29] Nguyen, S., Zhang, M., Tan, K. C., “Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems”, in 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, May 2015, pp. 2781–2788, available at: http://ieeexplore.ieee.org/document/7257234/

[30] Freitag, M., Hildebrandt, T., “Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization”, CIRP Annals - Manufacturing Technology, Vol. 65, No. 1, 2016, pp. 433–436, available at: http://linkinghub.elsevier.com/retrieve/pii/S000785061630066X

[31] Masood, A., Mei, Y., Chen, G., Zhang, M., “Many-objective genetic programming for job-shop scheduling”, in 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2016, pp. 209–216, available at: http://ieeexplore.ieee.org/document/7743797/

[32] Karunakaran, D., Chen, G., Zhang, M., “Parallel Multi-objective Job Shop Scheduling Using Genetic Programming”, in Artificial Life and Computational Intelligence: Second Australasian Conference, ACALCI 2016, Canberra, ACT, Australia, February 2-5, 2016, Proceedings, Ray, T., Sarker, R., Li, X., (ed.). Springer International Publishing, 2016, pp. 234–245, available at: http://link.springer.com/10.1007/978-3-319-28270-1_20

[33] Hunt, R., Johnston, M., Zhang, M., “Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming”, in Proceedings of the 2014 conference on Genetic and evolutionary computation - GECCO ’14. New York, New York, USA: ACM Press, 2014, pp. 927–934, available at: http://dl.acm.org/citation.cfm?doid=2576768.2598224

[34] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem”, IEEE Transactions on Evolutionary Computation, Vol. 17, No. 5, Oct. 2013, pp. 621–639, available at: http://ieeexplore.ieee.org/document/6353198/

[35] Park, J., Nguyen, S., Zhang, M., Johnston, M., “Evolving ensembles of dispatching rules using genetic programming for job shop scheduling”, in Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings, Machado, P., Heywood, M. I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K., (ed.). Cham: Springer International Publishing, 2015, pp. 92–104, available at: https://doi.org/10.1007/978-3-319-16501-1_8

[36] Hart, E., Sim, K., “A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling”, Evolutionary Computation, Vol. 24, No. 4, Dec. 2016, pp. 609–635, available at: http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00183

[37] Park, J., Mei, Y., Chen, G., Zhang, M., “Niching Genetic Programming based Hyper-heuristic Approach to Dynamic Job Shop Scheduling”, in Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO ’16 Companion. New York, New York, USA: ACM Press, 2016, pp. 109–110, available at: http://dl.acm.org/citation.cfm?doid=2908961.2908985

[38] Park, J., Mei, Y., Nguyen, S., Chen, G., Johnston, M., Zhang, M., “Genetic Programming Based Hyper-heuristics for Dynamic Job Shop Scheduling: Cooperative Coevolutionary Approaches”, in Genetic Programming: 19th European Conference, EuroGP 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Heywood, M. I., McDermott, J., Castelli, M., Costa, E., Sim, K., (ed.). Cham: Springer International Publishing, 2016, pp. 115–132, available at: http://link.springer.com/10.1007/978-3-319-30668-1_8

[39] El-Bouri, A., Shah, P., “A neural network for dispatching rule selection in a job shop”, The International Journal of Advanced Manufacturing Technology, Vol. 31, No. 3-4, Nov. 2006, pp. 342–349, available at: http://link.springer.com/10.1007/s00170-005-0190-y

[40] Mouelhi-Chibani, W., Pierreval, H., “Training a neural network to select dispatching rules in real time”, Computers & Industrial Engineering, Vol. 58, No. 2, Mar. 2010, pp. 249–256, available at: http://linkinghub.elsevier.com/retrieve/pii/S0360835209000953

[41] Heger, J., Hildebrandt, T., Scholz-Reiter, B., “Dispatching rule selection with Gaussian processes”, Central European Journal of Operations Research, Vol. 23, No. 1, Mar. 2015, pp. 235–249, available at: http://link.springer.com/10.1007/s10100-013-0322-7

[42] Hildebrandt, T., Heger, J., Scholz-Reiter, B., “Towards improved dispatching rules for complex shop floor scenarios”, in Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO ’10. New York, New York, USA: ACM Press, 2010, pp. 257, available at: http://portal.acm.org/citation.cfm?doid=1830483.1830530

[43] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Learning iterative dispatching rules for job shop scheduling with genetic programming”, The International Journal of Advanced Manufacturing Technology, Vol. 67, No. 1-4, Jul. 2013, pp. 85–100, available at: http://link.springer.com/10.1007/s00170-013-4756-9

[44] Pfund, M. E., Mason, S. J., Fowler, J. W., “Semiconductor Manufacturing Scheduling and Dispatching”, in Handbook of Production Scheduling. Boston: Kluwer Academic Publishers, 2006, pp. 213–241, available at: http://link.springer.com/10.1007/0-387-33117-4_9

[45] Chiang, T. C., Shen, Y. S., Fu, L. C., “A new paradigm for rule-based scheduling in the wafer probe centre”, International Journal of Production Research, Vol. 46, No. 15, aug 2008, pp. 4111–4133, available at: https://doi.org/10.1080/00207540601137199

[46] Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., Li, Y., “Cloud computing resource scheduling and a survey of its evolutionary approaches”, ACM Computing Surveys, Vol. 47, No. 4, jul 2015, pp. 1–33, available at: https://doi.org/10.1145/2788397

[47] Singh, S., Chana, I., “A survey on resource scheduling in cloud computing: Issues and challenges”, Journal of Grid Computing, Vol. 14, No. 2, feb 2016, pp. 217–264, available at: https://doi.org/10.1007/s10723-015-9359-2

[48] Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D., “Staff scheduling and rostering: A review of applications, methods and models”, European Journal of Operational Research, Vol. 153, No. 1, feb 2004, pp. 3–27, available at: https://doi.org/10.1016/s0377-2217(03)00095-x

[49] Hou, E., Ansari, N., Ren, H., “A genetic algorithm for multiprocessor scheduling”, IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 2, 1994, pp. 113–120, available at: https://doi.org/10.1109/71.265940

[50] Leung, J. Y.-T., Handbook of scheduling : algorithms, models, and performance analysis. Boca Raton, Fla.: Chapman & Hall/CRC, 2004, available at: http://www.worldcat.org/search?qt=worldcat_org_all&q=1584883979

[51] Allahverdi, A., Gupta, J. N., Aldowaisan, T., “A review of scheduling research involving setup considerations”, Omega, Vol. 27, No. 2, Apr. 1999, pp. 219–239, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305048398000425

[52] Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M. Y., “A survey of scheduling problems with setup times or costs”, European Journal of Operational Research, Vol. 187, No. 3, Jun. 2008, pp. 985–1032, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221706008174

[53] Baker, K. R., Trietsch, D., Principles of Sequencing and Scheduling. Wiley, 2013, available at: https://www.amazon.com/Principles-Sequencing-Scheduling-Kenneth-Baker-ebook/dp/B00D8Y5C1C?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B00D8Y5C1C

[54] Jain, A. S., Meeran, S., “A state-of-the-art review of job-shop scheduling techniques”, 1998.

[55] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Węglarz, J., Scheduling Computer and Manufacturing Processes. Springer Berlin Heidelberg, 2001, available at: https://doi.org/10.1007/978-3-662-04363-9

[56] Land, A. H., Doig, A. G., “An Automatic Method of Solving Discrete Programming Problems”, Econometrica, Vol. 28, No. 3, 1960, pp. 497–520.

[57] Graham, R. L., Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”, Annals of Discrete Mathematics, Vol. 5, No. C, 1979, pp. 287–326.

[58] Rocha, P. L., Ravetti, M. G., Mateus, G. R., Pardalos, P. M., “Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times”, Computers & Operations Research, Vol. 35, No. 4, Apr. 2008, pp. 1250–1264.

[59] Wotzlaw, A., Scheduling Unrelated Parallel Machines: Algorithms, Complexity, and Performance. AV Akademikerverlag, 2012.

[60] Lenstra, J. K., Shmoys, D. B., Tardos, E., “Approximation algorithms for scheduling unrelated parallel machines”, Mathematical Programming, Vol. 46, No. 1-3, Jan. 1990, pp. 259–271, available at: http://link.springer.com/10.1007/BF01585745

[61] Chen, B., Potts, C. N., Woeginger, G. J., “A review of machine scheduling: Complexity, algorithms and approximability”, in Handbook of Combinatorial Optimization. Springer US, 1998, pp. 1493–1641, available at: https://doi.org/10.1007/978-1-4613-0303-9_25

[62] Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[63] Mitchell, M., An Introduction to Genetic Algorithms. Cambridge, MA, USA: MIT Press, 1998.

[64] Eiben, A., Smith, J., Introduction to Evolutionary Computing, ser. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, available at: http://link.springer.com/10.1007/978-3-662-44874-8

[65] Kennedy, J., Eberhart, R., “Particle swarm optimization”, in Proceedings of ICNN’95 - International Conference on Neural Networks, Vol. 4. IEEE, 1995, pp. 1942–1948, available at: http://ieeexplore.ieee.org/document/488968/

[66] Engelbrecht, A. P., Computational Intelligence. John Wiley & Sons, Ltd, oct 2007, available at: https://doi.org/10.1002/9780470512517

[67] Dorigo, M., Maniezzo, V., Colorni, A., “Ant system: optimization by a colony of cooperating agents”, IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), Vol. 26, No. 1, 1996, pp. 29–41, available at: http://ieeexplore.ieee.org/document/484436/

[68] Glover, F., “Future paths for integer programming and links to artificial intelligence”, Computers & Operations Research, Vol. 13, No. 5, Jan. 1986, pp. 533–549, available at: http://linkinghub.elsevier.com/retrieve/pii/0305054886900481

[69] Glover, F., “Tabu Search—Part I”, ORSA Journal on Computing, Vol. 1, No. 3, Aug. 1989, pp. 190–206, available at: http://pubsonline.informs.org/doi/abs/10.1287/ijoc.1.3.190

[70] Glover, F., “Tabu Search—Part II”, ORSA Journal on Computing, Vol. 2, No. 1, Feb. 1990, pp. 4–32, available at: http://pubsonline.informs.org/doi/abs/10.1287/ijoc.2.1.4

[71] Khachaturyan, A., Semenovsovskaya, S., Vainshtein, B., “The thermodynamic approach to the structure analysis of crystals”, Acta Crystallographica Section A, Vol. 37, No. 5, Sep. 1981, pp. 742–754, available at: http://scripts.iucr.org/cgi-bin/paper?S0567739481001630

[72] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., “Optimization by Simulated Annealing”, Science, Vol. 220, No. 4598, May 1983, pp. 671–680, available at: http://www.sciencemag.org/cgi/doi/10.1126/science.220.4598.671

[73] Talbi, E.-G., Metaheuristics. John Wiley & Sons, Inc., jun 2009, available at: https://doi.org/10.1002/9780470496916

[74] Cheng, R., Gen, M., Tsujimura, Y., “A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation”, Computers & Industrial Engineering, Vol. 30, No. 4, sep 1996, pp. 983–997, available at: https://doi.org/10.1016/0360-8352(96)00047-2

[75] Wang, L., Siegel, H. J., Roychowdhury, V. P., Maciejewski, A. A., “Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach”, Journal of Parallel and Distributed Computing, Vol. 47, No. 1, nov 1997, pp. 8–22, available at: https://doi.org/10.1006/jpdc.1997.1392

[76] Cheng, R., Gen, M., Tsujimura, Y., “A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies”, Computers & Industrial Engineering, Vol. 36, No. 2, apr 1999, pp. 343–364, available at: https://doi.org/10.1016/s0360-8352(99)00136-9

[77] Zhou, H., Feng, Y., Han, L., “The hybrid heuristic genetic algorithm for job shop scheduling”, Computers & Industrial Engineering, Vol. 40, No. 3, jul 2001, pp. 191–200, available at: https://doi.org/10.1016/s0360-8352(01)00017-1

[78] Ishibuchi, H., Yoshida, T., Murata, T., “Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling”, IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2, apr 2003, pp. 204–223, available at: https://doi.org/10.1109/tevc.2003.810752

[79] Gao, J., Gen, M., Sun, L., Zhao, X., “A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems”, Computers & Industrial Engineering, Vol. 53, No. 1, aug 2007, pp. 149–162, available at: https://doi.org/10.1016/j.cie.2007.04.010

[80] Otto, A., Otto, C., “How to design effective priority rules: Example of simple assembly line balancing”, Computers & Industrial Engineering, Vol. 69, mar 2014, pp. 43–52, available at: https://doi.org/10.1016/j.cie.2013.12.013

[81] Holthaus, O., Rajendran, C., “Efficient jobshop dispatching rules: Further developments”, Production Planning & Control, Vol. 11, No. 2, jan 2000, pp. 171–178, available at: https://doi.org/10.1080/095372800232379

[82] Rajendran, C., Holthaus, O., “A comparative study of dispatching rules in dynamic flowshops and jobshops”, European Journal of Operational Research, Vol. 116, No. 1, jul 1999, pp. 156–170, available at: https://doi.org/10.1016/s0377-2217(98)00023-x

[83] Fanjul-Peyro, L., Ruiz, R., “Scheduling unrelated parallel machines with optional machines and jobs selection”, Computers & Operations Research, Vol. 39, No. 7, Jul. 2012, pp. 1745–1753, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305054811003005

[84] Lee, J.-H., Yu, J.-M., Lee, D.-H., “A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness”, The International Journal of Advanced Manufacturing Technology, Vol. 69, No. 9-12, Dec. 2013, pp. 2081–2089, available at: http://link.springer.com/10.1007/s00170-013-5192-6

[85] Wang, I.-L., Wang, Y.-C., Chen, C.-W., “Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics”, Flexible Services and Manufacturing Journal, Vol. 25, No. 3, Sep. 2013, pp. 343–366, available at: http://link.springer.com/10.1007/s10696-012-9150-7

[86] Fanjul-Peyro, L., Ruiz, R., “Iterated greedy local search methods for unrelated parallel machine scheduling”, European Journal of Operational Research, Vol. 207, No. 1, Nov. 2010, pp. 55–69, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221710002572

[87] Fanjul-Peyro, L., Ruiz, R., “Size-reduction heuristics for the unrelated parallel machines scheduling problem”, Computers & Operations Research, Vol. 38, No. 1, Jan. 2011, pp. 301–309, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305054810001188

[88] Cota, L. P., Haddad, M. N., Souza, M. J. F., Coelho, V. N., “AIRP: A heuristic algorithm for solving the unrelated parallel machine scheduling problem”, in 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2014, pp. 1855–1862, available at: http://ieeexplore.ieee.org/document/6900245/

[89] de C. M. Nogueira, J. P., Arroyo, J. E. C., Villadiego, H. M. M., Goncalves, L. B., “Hybrid GRASP Heuristics to Solve an Unrelated Parallel Machine Scheduling Problem with Earliness and Tardiness Penalties”, Electronic Notes in

Theoretical Computer Science, Vol. 302, Feb. 2014, pp. 53–72, available at: http://linkinghub.elsevier.com/retrieve/pii/S1571066114000218

[90] Logendran, R., McDonell, B., Smucker, B., “Scheduling unrelated parallel machines with sequence-dependent setups”, Computers & Operations Research, Vol. 34, No. 11, Nov. 2007, pp. 3420–3438, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305054806000438

[91] Ying, K.-C., Lee, Z.-J., Lin, S.-W., “Makespan minimization for scheduling unrelated parallel machines with setup times”, Journal of Intelligent Manufacturing, Vol. 23, No. 5, nov 2010, pp. 1795–1803, available at: https://doi.org/10.1007/s10845-010-0483-3

[92] Behnamian, J., Zandieh, M., Fatemi Ghomi, S., “Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm”, Expert Systems with Applications, Vol. 36, No. 6, Aug. 2009, pp. 9637–9644, available at: http://linkinghub.elsevier.com/retrieve/pii/S0957417408007252

[93] Lin, C.-W., Lin, Y.-K., Hsieh, H.-T., “Ant colony optimization for unrelated parallel machine scheduling”, The International Journal of Advanced Manufacturing Technology, Vol. 67, No. 1-4, Jul. 2013, pp. 35–45, available at: http://link.springer.com/10.1007/s00170-013-4766-7

[94] Glass, C., Potts, C., Shade, P., “Unrelated parallel machine scheduling using local search”, Mathematical and Computer Modelling, Vol. 20, No. 2, jul 1994, pp. 41–52, available at: https://doi.org/10.1016/0895-7177(94)90205-4

[95] Mönch, L., Balasubramanian, H., Fowler, J. W., Pfund, M. E., “Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times”, Computers & Operations Research, Vol. 32, No. 11, Nov. 2005, pp. 2731–2750, available at: http://linkinghub.elsevier.com/retrieve/pii/S0305054804000759

[96] Chyu, C.-C., Chang, W.-S., “A competitive evolution strategy memetic algorithm for unrelated parallel machine scheduling to minimize total weighted tardiness and flow time”, in The 40th International Conference on Computers & Indutrial Engineering. IEEE, Jul. 2010, pp. 1–6, available at: http://ieeexplore.ieee.org/document/5668388/

[97] Vallada, E., Ruiz, R., “A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times”, European Journal of Operational Research, Vol. 211, No. 3, Jun. 2011, pp. 612–622, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221711000142

[98] Balin, S., “Non-identical parallel machine scheduling using genetic algorithm”, Expert Systems with Applications, Vol. 38, No. 6, Jun. 2011, pp. 6814–6821, available at: http://linkinghub.elsevier.com/retrieve/pii/S0957417410014272

[99] Haddad, M. N., Coelho, I. M., Souza, M. J. F., Ochi, L. S., Santos, H. G., Martins, A. X., “GARP: A New Genetic Algorithm for the Unrelated Parallel Machine Scheduling Problem with Setup Times”, in 2012 31st International Conference of the Chilean Computer Science Society. IEEE, Nov. 2012, pp. 152–160, available at: http://ieeexplore.ieee.org/document/6694085/

[100] Costa, A., Cappadonna, F. A., Fichera, S., “A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times”, The International Journal of Advanced Manufacturing Technology, Vol. 69, No. 9-12, Dec. 2013, pp. 2799–2817, available at: http://link.springer.com/10.1007/s00170-013-5221-5

[101] Ðurasević, M., Jakobović, D., “Comparison of solution representations for scheduling in the unrelated machines environment”, in 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, May 2016, pp. 1336–1342, available at: http://ieeexplore.ieee.org/document/7522347/

[102] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., Freund, R. F., “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems”, Journal of Parallel and Distributed Computing, Vol. 61, No. 6, Jun. 2001, pp. 810–837, available at: http://linkinghub.elsevier.com/retrieve/pii/S0743731500917143

[103] Lee, Y. H., Bhaskaran, K., Pinedo, M., “A heuristic to minimize the total weighted tardiness with sequence-dependent setups”, IIE Transactions, Vol. 29, No. 1, Jan. 1997, pp. 45–52, available at: http://www.tandfonline.com/doi/abs/10.1080/07408179708966311

[104] Munir, E. U., Li, J., Shi, S., Zou, Z., Yang, D., “Maxstd: A task scheduling heuristic for heterogeneous computing environment”, Information Technology Journal, Vol. 7, No. 4, 2008, pp. 679–683.

[105] e Santos, A. S., Madureira, A. M., “Ordered minimum completion time heuristic for unrelated parallel-machines problems”, in 2014 9th Iberian Conference on

Information Systems and Technologies (CISTI). IEEE, Jun. 2014, pp. 1–6, available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6876939

[106] Izakian, H., Abraham, A., Snasel, V., “Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments”, in 2009 International Joint Conference on Computational Sciences and Optimization. IEEE, Apr. 2009, pp. 8–12, available at: http://ieeexplore.ieee.org/document/5193632/

[107] Pfund, M., Fowler, J. W., Gadkari, A., Chen, Y., “Scheduling jobs on parallel machines with setup times and ready times”, Computers & Industrial Engineering, Vol. 54, No. 4, May 2008, pp. 764–782, available at: http://linkinghub.elsevier.com/retrieve/pii/S036083520700229X

[108] Morton, T. E., Pentico, D. W., Heuristic Scheduling Systems. John Wiley And Sons, Inc., 1993.

[109] Morton, T. E., Rachamadugu, R. M. V., “Myopic heuristics for the single machine weighted tardiness problem.”, DTIC Document, Tech. Rep., 1982.

[110] Vepsalainen, A. P. J., Morton, T. E., “Priority Rules for Job Shops with Weighted Tardiness Costs”, Management Science, Vol. 33, No. 8, Aug. 1987, pp. 1035–1047, available at: http://pubsonline.informs.org/doi/abs/10.1287/mnsc.33.8.1035

[111] Yang-Kuei, L., Chi-Wei, L., “Dispatching rules for unrelated parallel machine scheduling with release dates”, The International Journal of Advanced Manufacturing Technology, Vol. 67, No. 1-4, Jul. 2013, pp. 269–279, available at: http://link.springer.com/10.1007/s00170-013-4773-8

[112] Rafsanjani, M. K., Bardsiri, A. K., “A new heuristic approach for scheduling independent tasks on heterogeneous computing systems”, International Journal of Machine Learning and Computing, Vol. 2, No. 4, 2012, pp. 371.

[113] Xhafa, F., Barolli, L., Durresi, A., “Batch mode scheduling in grid systems”, International Journal of Web and Grid Services, Vol. 3, No. 1, 2007, pp. 19-37.

[114] Du Kim, H., Kim, J. S., An Online Scheduling Algorithm for Grid Computing Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 34–39, available at: https://doi.org/10.1007/978-3-540-24680-0_5

[115] Koza, J. R., Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press, 1992.

[116] Koza, J. R., Genetic Programming II: Automatic Discovery of Reusable Programs. Cambridge, MA, USA: MIT Press, 1994.

[117] Koza, J. R., Andre, D., Bennett, F. H., Keane, M. A., Genetic Programming III: Darwinian Invention & Problem Solving, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[118] Koza, J. R., Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[119] Banzhaf, W., Francone, F. D., Keller, R. E., Nordin, P., Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[120] Poli, R., Langdon, W. B., McPhee, N. F., A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008, (With contributions by J. R. Koza), available at: http://www.gp-field-guide.org.uk

[121] Koza, J., “Genetically breeding populations of computer programs to solve problems in artificial intelligence”, in [1990] Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence. IEEE Comput. Soc. Press, 1990, pp. 819–827, available at: http://ieeexplore.ieee.org/document/130444/

[122] Espejo, P. G., Ventura, S., Herrera, F., “A Survey on the Application of Genetic Programming to Classifcation”, Ieee Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews, Vol. 40, No. 2, 2010, pp. 121–144.

[123] Koza, J. R., “Human-competitive results produced by genetic programming”, Genetic Programming and Evolvable Machines, Vol. 11, No. 3-4, Sep. 2010, pp. 251–284, available at: http://link.springer.com/10.1007/s10710-010-9112-3

[124] Burke, E. K., Hyde, M. R., Kendall, G., Woodward, J., “Automatic heuristic generation with genetic programming”, in Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO ’07. New York, New York, USA: ACM Press, 2007, pp. 1559, available at: http://portal.acm.org/citation.cfm?doid=1276958.1277273

[125] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J. R., “Exploring Hyper-heuristic Methodologies with Genetic Programming”, Computational Intelligence, Vol. 1, 2009, pp. 177–201, available at: http://www.cs.nott.ac.uk/~gxo/papers/ChapterGPasHH09.pdf

[126] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J. R., "A Classification of Hyper-heuristic Approaches", in Handbook of Metaheuristics, 2010, pp. 449–468, available at: http://dx.doi.org/10.1007/978-1-4419-1665-5_15

[127] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., "Hyper-heuristics: a survey of the state of the art", Journal of the Operational Research Society, Vol. 64, No. 12, Dec. 2013, pp. 1695–1724, available at: http://link.springer.com/10.1057/jors.2013.71

[128] Kumar, R., Joshi, A. H., Banka, K. K., Rockett, P. I., "Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming", in Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08. New York, New York, USA: ACM Press, 2008, pp. 1227, available at: http://portal.acm.org/citation.cfm?doid=1389095.1389335

[129] Özcan, E., Parkes, A. J., "Policy matrix evolution for generation of heuristics", in Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11. New York, New York, USA: ACM Press, 2011, pp. 2011, available at: http://portal.acm.org/citation.cfm?doid=2001576.2001846

[130] Burke, E. K., Hyde, M. R., Kendall, G., Woodward, J., "Automating the Packing Heuristic Design Process with Genetic Programming", Evolutionary Computation, Vol. 20, No. 1, Mar. 2012, pp. 63–89, available at: http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00044

[131] Oltean, M., Dumitrescu, D., "Evolving TSP Heuristics Using Multi Expression Programming", in Computational Science - ICCS 2004: 4th International Conference, Bubak, M., van Albada, G. D., Sloot, P. M. A., Dongarra, J., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 670–673, available at: http://link.springer.com/10.1007/978-3-540-24687-9_99

[132] Beham, A., Kofler, M., Wagner, S., Affenzeller, M., "Agent-Based Simulation of Dispatching Rules in Dynamic Pickup and Delivery Problems", in 2009 2nd International Symposium on Logistics and Industrial Informatics. IEEE, Sep. 2009, pp. 1–6, available at: http://ieeexplore.ieee.org/document/5258763/

[133] Vonolfen, S., Beham, A., Kommenda, M., Affenzeller, M., "Structural Synthesis of Dispatching Rules for Dynamic Dial-a-Ride Problems", in Computer Aided Systems Theory - EUROCAST 2013: 14th International Conference, Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 276–283, available at: http://link.springer.com/10.1007/978-3-642-53856-8_35

[134] Bader-El-Den, M., Poli, R., Fatima, S., "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework", Memetic Computing, Vol. 1, No. 3, Nov. 2009, pp. 205–219, available at: http://link.springer.com/10.1007/s12293-009-0022-y

[135] Pillay, N., "Evolving hyper-heuristics for the uncapacitated examination timetabling problem", Journal of the Operational Research Society, Vol. 63, No. 1, Jan. 2012, pp. 47–58, available at: http://link.springer.com/10.1057/jors.2011.12

[136] Frankola, T., Golub, M., Jakobovic, D., "Evolutionary algorithms for the resource constrained scheduling problem", in ITI 2008 - 30th International Conference on Information Technology Interfaces. IEEE, Jun. 2008, pp. 715–722, available at: http://ieeexplore.ieee.org/document/4588499/

[137] Keijzer, M., "Improving Symbolic Regression with Interval Arithmetic and Linear Scaling", Genetic Programming Proceedings of EuroGP2003, Vol. 2610, 2003, pp. 70–82, available at: http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2610&spage=70

[138] Langdon, W. B., Poli, R., Foundations of Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, available at: http://link.springer.com/10.1007/978-3-662-04726-2

[139] Luke, S., Panait, L., "A comparison of bloat control methods for genetic programming", Evolutionary Computation, Vol. 14, No. 3, sep 2006, pp. 309–344, available at: https://doi.org/10.1162/evco.2006.14.3.309

[140] Whigham, P., Dick, G., "Implicitly controlling bloat in genetic programming", IEEE Transactions on Evolutionary Computation, Vol. 14, No. 2, apr 2010, pp. 173–190, available at: https://doi.org/10.1109/tevc.2009.2027314

[141] Poli, R., McPhee, N. F., "Parsimony pressure made easy: Solving the problem of bloat in GP", in Theory and Principled Methods for the Design of Metaheuristics. Springer Berlin Heidelberg, nov 2013, pp. 181–204, available at: https://doi.org/10.1007/978-3-642-33206-7_9

[142] Poli, R., "Parallel distributed genetic programming", The University of Birmingham, Tech. Rep., 1996.

[143] Brameier, M. F., Banzhaf, W., Linear Genetic Programming, 1st ed. Springer Publishing Company, Incorporated, 2010.

[144] Whigham, P., Science, D. O. C., “Grammatically-based genetic programming”, 1995.

[145] McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., O’neill, M., “Grammar-based Genetic programming: A survey”, Genetic Programming and Evolvable Machines, Vol. 11, No. 3-4, 2010, pp. 365–396.

[146] Miller, J. F., Thomson, P., “Cartesian genetic programming”, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 1802, 2000, pp. 121–132.

[147] PD’haeseleer, P., “Context preserving crossover in genetic programming”, in Proceedings of the 1994 IEEE World Congress on Computational Intelligence, 1994, pp. 256—-261, available at: http://scholar.google.es/scholar?hl=es&q=D’haeseleer,+1994]+D’haeseleer,+P.+(1994).+Context+preserving+&btnG=Buscar&lr=&as_ylo=&as_vis=0#0

[148] Langdon, W. B., “Size fair and homologous tree crossovers for tree genetic programming”, Genetic Programming and Evolvable Machines, Vol. 1, No. 1, 2000, pp. 95–119, available at: http://dx.doi.org/10.1023/A:1010024515191

[149] Keijzer, M., Babovic, V., “Dimensionally Aware Genetic Programming”, Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, 1999, pp. 1069–1076, available at: http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GP-420.pdf

[150] Ferreira, C., “Gene expression programming: a new adaptive algorithm for solving problems”, Complex Systems, Vol. 13, No. 2, 2001, pp. 87-129, available at: http://arxiv.org/abs/cs/0102027

[151] Ðurasević, M., Jakobović, D., Knežević, K., “Adaptive scheduling on unrelated machines with genetic programming”, Applied Soft Computing, Vol. 48, Nov. 2016, pp. 419–430, available at: http://linkinghub.elsevier.com/retrieve/pii/S1568494616303519

[152] Zhang, M., Wong, P., “Genetic programming for medical classification: a program simplification approach”, Genetic Programming and Evolvable Machines, Vol. 9, No. 3, 2008, pp. 229–255, available at: http://dx.doi.org/10.1007/s10710-008-9059-9

[153] Kinzett, D., Johnston, M., Zhang, M., “Numerical simplification for bloat control and analysis of building blocks in genetic programming”, Evolutionary Intelligence, Vol. 2, No. 4, 2009, pp. 151, available at: http://dx.doi.org/10.1007/s12065-009-0029-9

[154] Zhang, M., Smart, W., “Learning weights in genetic programs using gradient descent for object recognition”, in Proceedings of the 3rd European Conference on Applications of Evolutionary Computing, ser. EC’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 417–427.

[155] Johnston, M., Liddle, T., Zhang, M., A Relaxed Approach to Simplification in Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 110–121.

[156] Dimopoulos, C., Zalzala, A., "Investigating the use of genetic programming for a classic one-machine scheduling problem", ARRAY(0x7f0faa5322f0), Research Report, December 1998, available at: http://eprints.whiterose.ac.uk/82572/

[157] Dimopoulos, C., Zalzala, A., "A genetic programming heuristic for the one-machine total tardiness problem", in Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). IEEE, 1999, pp. 2207–2214, available at: http://ieeexplore.ieee.org/document/785549/

[158] Dimopoulos, C., Zalzala, A., "Investigating the use of genetic programming for a classic one-machine scheduling problem", Advances in Engineering Software, Vol. 32, No. 6, Jun. 2001, pp. 489–498, available at: http://linkinghub.elsevier.com/retrieve/pii/S0965997800001095

[159] Miyashita, K., "Job-shop scheduling with genetic programming", in Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation, ser. GECCO'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 505–512, available at: http://dl.acm.org/citation.cfm?id=2933718.2933809

[160] Adams, T. P., "Creation of simple, deadline, and priority scheduling algorithms using genetic programming. in: Genetic algorithms and genetic programming at stanford", in in Genetic Algorithms and Genetic Programming at Stanford 2002, 2002. [Online]. Available: http://www.genetic-programming.org/sp2002/Adams.pdf, 2002.

[161] Wen-Jun Yin, Min Liu, Cheng Wu, "Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming", in The 2003 Congress on Evolutionary Computation, 2003. CEC '03., Vol. 2. IEEE, 2003, pp. 1050–1055, available at: http://ieeexplore.ieee.org/document/1299784/

[162] Nhu Binh Ho, Joc Cing Tay, "Evolving Dispatching Rules for solving the Flexible Job-Shop Problem", in 2005 IEEE Congress on Evolutionary Computation, Vol. 3. IEEE, 2005, pp. 2848–2855, available at: http://ieeexplore.ieee.org/document/1555052/

[163] Geiger, C. D., Uzsoy, R., Aytuğ, H., "Rapid Modeling and Discovery of Priority Dispatching Rules: An Autonomous Learning Approach", Journal of Scheduling, Vol. 9, No. 1, Feb. 2006, pp. 7–34, available at: http://link.springer.com/10.1007/s10951-006-5591-8

[164] Geiger, C. D., Uzsoy, R., "Learning effective dispatching rules for batch processor scheduling", International Journal of Production Research, Vol. 46, No. 6, Mar. 2008, pp. 1431–1454, available at: http://www.tandfonline.com/doi/abs/10.1080/00207540600993360

[165] Jakobović, D., Budin, L., "Dynamic scheduling with genetic programming", in Genetic Programming: 9th European Conference, EuroGP 2006, Budapest, Hungary, April 10-12, 2006. Proceedings, Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–84, available at: https://doi.org/10.1007/11729976_7

[166] Jakobović, D., Jelenković, L., Budin, L., "Genetic Programming Heuristics for Multiple Machine Scheduling", in Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 321–330, available at: http://link.springer.com/10.1007/978-3-540-71605-1_30

[167] Tay, J. C., Ho, N. B., "Designing Dispatching Rules to Minimize Total Tardiness", in Evolutionary Scheduling, Dahal, K. P., Tan, K. C., Cowling, P. I., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 101–124, available at: http://link.springer.com/10.1007/978-3-540-48584-1_4

[168] Beham, A., Winkler, S., Wagner, S., Affenzeller, M., "A genetic programming approach to solve scheduling problems with parallel simulation", in 2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, Apr. 2008, pp. 1–5, available at: http://ieeexplore.ieee.org/document/4536362/

[169] Jia-Wei Yang, Hsueh-Chien Cheng, Tsung-Che Chiang, Li-Chen Fu, "Multiobjective lot scheduling and dynamic OHT routing in a 300-mm wafer fab", in 2008 IEEE International Conference on Systems, Man and Cybernetics. IEEE, Oct. 2008, pp. 1608–1613, available at: http://ieeexplore.ieee.org/document/4811517/

[170] Baykasoglu, A., Gocken, M., "Gene expression programming based due date assignment in a simulated job shop", Expert Systems with Applications, Vol. 36, No. 10, Dec. 2009, pp. 12 143–12 150, available at: http://linkinghub.elsevier.com/retrieve/pii/S095741740900311X

[171] Baykasoglu, A., Gocken, M., Ozbakir, L., "Genetic programming based data mining approach to dispatching rule selection in a simulated job shop", SIMULATION, Vol. 86, No. 12, 2010, pp. 715-728.

[172] Furuholmen, M., Glette, K., Hovin, M., Torresen, J., "Coevolving heuristics for the Distributor's Pallet Packing Problem", in 2009 IEEE Congress on

Evolutionary Computation. IEEE, May 2009, pp. 2810–2817, available at: http://ieeexplore.ieee.org/document/4983295/

[173] Kofler, M., Beham, A., Wagner, S., Affenzeller, M., "Evaluation of dispatching strategies for the optimization of a real-world production plant", in 2009 2nd International Symposium on Logistics and Industrial Informatics. IEEE, sep 2009, available at: https://doi.org/10.1109/lindi.2009.5258765

[174] Pickardt, C., Branke, J., Hildebrandt, T., Heger, J., Scholz-Reiter, B., "Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness", in Proceedings of the 2010 Winter Simulation Conference. IEEE, Dec. 2010, pp. 2504–2515, available at: http://ieeexplore.ieee.org/document/5678946/

[175] Vázquez-Rodríguez, J. A., Ochoa, G., "On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming", Journal of the Operational Research Society, Vol. 62, No. 2, Feb. 2011, pp. 381–396, available at: http://link.springer.com/10.1057/jors.2010.132

[176] Nie, L., Shao, X., Gao, L., Li, W., "Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems", The International Journal of Advanced Manufacturing Technology, Vol. 50, No. 5-8, Sep. 2010, pp. 729–747, available at: http://link.springer.com/10.1007/s00170-010-2518-5

[177] Nie, L., Gao, L., Li, P., Zhang, L., "Application of gene expression programming on dynamic job shop scheduling problem", in Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD). IEEE, Jun. 2011, pp. 291–295, available at: http://ieeexplore.ieee.org/document/5960088/

[178] Nie, L., Gao, L., Li, P., Li, X., "A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates", Journal of Intelligent Manufacturing, Vol. 24, No. 4, Aug. 2013, pp. 763–774, available at: http://link.springer.com/10.1007/s10845-012-0626-9

[179] Wang, X., Nie, L., Bai, Y., "Discovering scheduling rules with a machine learning approach based on GEP and PSO for dynamic scheduling problems in shop floor", in Computational Intelligence in Industrial Application. CRC Press, Jun. 2015, pp. 365–370, available at: http://www.crcnetbase.com/doi/10.1201/b18590-71

[180] Pitzer, E., Beham, A., Affenzeller, M., Heiss, H., Vorderwinkler, M., "Production fine planning using a solution archive of priority rules", in 3rd IEEE International Symposium on Logistics and Industrial Informatics. IEEE, Aug. 2011, pp. 111–116, available at: http://ieeexplore.ieee.org/document/6031130/

[181] Nie, L., Bai, Y., Wang, X., Liu, K., “Discover Scheduling Strategies with Gene Expression Programming for Dynamic Flexible Job Shop Scheduling Problem”, in Advances in Swarm Intelligence: Third International Conference, ICSI 2012, Shenzhen, China, June 17-20, 2012 Proceedings, Part II, Tan, Y., Shi, Y., Ji, Z., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 383–390, available at: http://link.springer.com/10.1007/978-3-642-31020-1_45

[182] Nie, L., Gao, L., Li, P., Shao, X., “Reactive scheduling in a job shop where jobs arrive over time”, Computers & Industrial Engineering, Vol. 66, No. 2, Oct. 2013, pp. 389–405, available at: http://linkinghub.elsevier.com/retrieve/pii/S0360835213002209

[183] Nguyen, S., Zhang, M., Johnston, M., “A genetic programming based hyper-heuristic approach for combinatorial optimisation”, in Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO ’11. New York, New York, USA: ACM Press, 2011, pp. 1299, available at: http://portal.acm.org/citation.cfm?doid=2001576.2001752

[184] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming”, in Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings, Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 121–133, available at: https://doi.org/10.1007/978-3-642-29139-5_11

[185] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Genetic Programming for Evolving Due-Date Assignment Models in Job Shop Environments”, Evolutionary Computation, Vol. 22, No. 1, Mar. 2014, pp. 105–138, available at: http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00105

[186] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Automatic Programming via Iterated Local Search for Dynamic Job Shop Scheduling”, IEEE Transactions on Cybernetics, Vol. 45, No. 1, Jan. 2015, pp. 1–14, available at: http://ieeexplore.ieee.org/document/6807725/

[187] Jakobović, D., Marasović, K., “Evolving priority scheduling heuristics with genetic programming”, Applied Soft Computing, Vol. 12, No. 9, Sep. 2012, pp. 2781–2789, available at: http://linkinghub.elsevier.com/retrieve/pii/S1568494612001780

[188] Abednego, L., Hendratmo, D., “Genetic programming hyper-heuristic for solving dynamic production scheduling problem”, in Proceedings of the 2011 International

Conference on Electrical Engineering and Informatics. IEEE, Jul. 2011, pp. 1–4, available at: http://ieeexplore.ieee.org/document/6021768/

[189] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems”, in 2012 IEEE Congress on Evolutionary Computation. IEEE, Jun. 2012, pp. 1–8, available at: http://ieeexplore.ieee.org/document/6252968/

[190] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Automatic Design of Scheduling Policies for Dynamic Multi-objective Job Shop Scheduling via Cooperative Coevolution Genetic Programming”, IEEE Transactions on Evolutionary Computation, Vol. 18, No. 2, Apr. 2014, pp. 193–208, available at: http://ieeexplore.ieee.org/document/6468087/

[191] Hansen, P., Mladenović, N., “Variable neighborhood search: Principles and applications”, European Journal of Operational Research, Vol. 130, No. 3, May 2001, pp. 449–467, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221700001004

[192] Hansen, P., Mladenović, N., Pérez, J. A. M., “Variable neighbourhood search: methods and applications”, Annals of Operations Research, Vol. 175, No. 1, oct 2009, pp. 367–407, available at: https://doi.org/10.1007/s10479-009-0657-6

[193] Hunt, R., Johnston, M., Zhang, M., “Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming”, in 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2014, pp. 618–625, available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6900655

[194] Branke, J., Hildebrandt, T., Scholz-Reiter, B., “Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations”, Evolutionary Computation, Vol. 23, No. 2, Jun. 2015, pp. 249–277, available at: http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00131

[195] Park, J., Nguyen, S., Zhang, M., Johnston, M., “Genetic programming for order acceptance and scheduling”, in 2013 IEEE Congress on Evolutionary Computation. IEEE, Jun. 2013, pp. 1005–1012, available at: http://ieeexplore.ieee.org/document/6557677/

[196] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Learning Reusable Initial Solutions for Multi-objective Order Acceptance and Scheduling Problems with Genetic Programming”, in Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings, Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A. Ş., Hu, B., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 157–168, available at: http://link.springer.com/10.1007/978-3-642-37207-0_14

[197] Park, J., Nguyen, S., Johnston, M., Zhang, M., “Evolving Stochastic Dispatching Rules for Order Acceptance and Scheduling via Genetic Programming”, in AI 2013: Advances in Artificial Intelligence: 26th Australasian Joint Conference, Dunedin, New Zealand, December 1-6, 2013. Proceedings, Cranefield, S., Nayak, A., (ed.). Cham: Springer International Publishing, 2013, pp. 478–489, available at: http://link.springer.com/10.1007/978-3-319-03680-9_48

[198] Park, J., Nguyen, S., Zhang, M., Johnston, M., “Enhancing Heuristics for Order Acceptance and Scheduling Using Genetic Programming”, in Simulated Evolution and Learning: 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings, Dick, G., Browne, W. N., Whigham, P., Zhang, M., Bui, L. T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K. C., Tang, K., (ed.). Cham: Springer International Publishing, 2014, pp. 723–734, available at: http://link.springer.com/10.1007/978-3-319-13563-2_61

[199] Nguyen, S., Zhang, M., Johnston, M., “A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling”, in 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2014, pp. 1824–1831, available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6900347

[200] Nguyen, S., “A learning and optimizing system for order acceptance and scheduling”, The International Journal of Advanced Manufacturing Technology, Vol. 86, No. 5-8, Sep. 2016, pp. 2021–2036, available at: http://link.springer.com/10.1007/s00170-015-8321-6

[201] Nguyen, S., Zhang, M., Johnston, M., “Enhancing Branch-and-Bound Algorithms for Order Acceptance and Scheduling with Genetic Programming”, in Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers, Nicolau, M., Krawiec, K., Heywood, M. I., Castelli, M., García-Sánchez, P., Merelo, J. J., Rivas Santos, V. M., Sim, K., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 124–136, available at: http://link.springer.com/10.1007/978-3-662-44303-3_11

[202] Han, S., Seo, J., Park, J., “Designing an Effective Scheduling Scheme Considering Multi-level BOM in Hybrid Job Shop”, in Proceedings of the International Conference on Industrial Engine, 2012, pp. 1302–1310.

[203] Hildebrandt, T., Goswami, D., Freitag, M., “Large-scale simulation-based optimization of semiconductor dispatching rules”, in Proceedings of the Winter Simulation Conference 2014. IEEE, Dec. 2014, pp. 2580–2590, available at: http://ieeexplore.ieee.org/document/7020102/

[204] Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., "From Grammars to Parameters: Automatic Iterated Greedy Design for the Permutation Flow-Shop Problem with Weighted Tardiness", in Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers, Nicosia, G., Pardalos, P., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 321–334, available at: http://link.springer.com/10.1007/978-3-642-44973-4_36

[205] Pickardt, C. W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B., "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems", International Journal of Production Economics, Vol. 145, No. 1, Sep. 2013, pp. 67–77, available at: http://linkinghub.elsevier.com/retrieve/pii/S0925527312004574

[206] Qin, W., Zhang, J., Sun, Y., "Multiple-objective scheduling for interbay AMHS by using genetic-programming-based composite dispatching rules generator", Computers in Industry, Vol. 64, No. 6, Aug. 2013, pp. 694–707, available at: http://linkinghub.elsevier.com/retrieve/pii/S0166361513000626

[207] Masood, A., Mei, Y., Chen, G., Zhang, M., "A PSO-Based Reference Point Adaption Method for Genetic Programming Hyper-Heuristic in Many-Objective Job Shop Scheduling", in Artificial Life and Computational Intelligence: Third Australasian Conference, ACALCI 2017, Geelong, VIC, Australia, January 31 – February 2, 2017, Proceedings, Wagner, M., Li, X., Hendtlass, T., (ed.). Cham: Springer International Publishing, 2017, pp. 326–338, available at: http://link.springer.com/10.1007/978-3-319-51691-2_28

[208] Sim, K., Hart, E., "A Novel Heuristic Generator for JSSP Using a Tree-Based Representation of Dispatching Rules", in Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion '15. New York, New York, USA: ACM Press, 2015, pp. 1485–1486, available at: http://dl.acm.org/citation.cfm?doid=2739482.2764697

[209] Riley, M., Mei, Y., Zhang, M., "Improving job shop dispatching rules via terminal weighting and adaptive mutation in genetic programming", in 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2016, pp. 3362–3369, available at: http://ieeexplore.ieee.org/document/7744215/

[210] Shi, W., Song, X., Sun, J., "Automatic Heuristic Generation with Scatter Programming to Solve the Hybrid Flow Shop Problem", Advances in Mechanical Engineering, Vol. 7, No. 2, Feb. 2015, pp. 587038, available at: http://journals.sagepub.com/doi/10.1155/2014/587038

[211] Hedar, A.-R., Osman, M. K., “Scatter programming”, in 2010 2nd International Conference on Computer Technology and Development. IEEE, Nov. 2010, pp. 451–455, available at: http://ieeexplore.ieee.org/document/5645839/

[212] Park, J., Nguyen, S., Zhang, M., Johnston, M., “A Single Population Genetic Programming based Ensemble Learning Approach to Job Shop Scheduling”, in Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion ’15. New York, New York, USA: ACM Press, 2015, pp. 1451–1452, available at: http://dl.acm.org/citation.cfm?doid=2739482.2764651

[213] Branke, J., Groves, M. J., Hildebrandt, T., “Evolving control rules for a dual-constrained job scheduling scenario”, in Proceedings of the 2016 Winter Simulation Conference, ser. WSC ’16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 2568–2579, available at: http://dl.acm.org/citation.cfm?id=3042094.3042415

[214] Chen, L., Zheng, H., Zheng, D., Li, D., “An ant colony optimization-based hyper-heuristic with genetic programming approach for a hybrid flow shop scheduling problem”, in 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, May 2015, pp. 814–821, available at: http://ieeexplore.ieee.org/document/7256975/

[215] Hunt, R., Johnston, M., Zhang, M., “Using local search to evaluate dispatching rules in dynamic job shop scheduling”, in Evolutionary Computation in Combinatorial Optimization: 15th European Conference, EvoCOP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings, Ochoa, G., Chicano, F., (ed.). Cham: Springer International Publishing, 2015, pp. 222–233, available at: https://doi.org/10.1007/978-3-319-16468-7_19

[216] Hunt, R., Johnston, M., Zhang, M., Hunt, R., Johnston, M., Zhang, M., “Evolving dispatching rules with greater understandability for dynamic job shop scheduling”, 2015.

[217] Montana, D. J., “Strongly Typed Genetic Programming”, Evolutionary Computation, Vol. 3, No. 2, Jun. 1995, pp. 199–230, available at: http://www.mitpressjournals.org/doi/10.1162/evco.1995.3.2.199

[218] Karunakaran, D., Mei, Y., Chen, G., Zhang, M., “Dynamic Job Shop Scheduling Under Uncertainty Using Genetic Programming”, in Intelligent and Evolutionary Systems: The 20th Asia Pacific Symposium, IES 2016, Canberra, Australia, November 2016, Proceedings, Leu, G., Singh, H. K., Elsayed, S., (ed.). Cham: Springer International Publishing, 2017, pp. 195–210, available at: http://link.springer.com/10.1007/978-3-319-49049-6_14

[219] Li, D., Zhan, R., Zheng, D., Li, M., Kaku, I., “A Hybrid Evolutionary Hyper-Heuristic Approach for Intercell Scheduling Considering Transportation Capacity”, IEEE Transactions on Automation Science and Engineering, Vol. 13, No. 2, Apr. 2016, pp. 1072–1089, available at: http://ieeexplore.ieee.org/document/7270346/

[220] Mei, Y., Zhang, M., “A comprehensive analysis on reusability of GP-evolved job shop dispatching rules”, in 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2016, pp. 3590–3597, available at: http://ieeexplore.ieee.org/document/7744244/

[221] Mei, Y., Zhang, M., Nyugen, S., “Feature Selection in Evolving Job Shop Dispatching Rules with Genetic Programming”, in Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO ’16. New York, New York, USA: ACM Press, 2016, pp. 365–372, available at: http://dl.acm.org/citation.cfm?doid=2908812.2908822

[222] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Selection Schemes in Surrogate-Assisted Genetic Programming for Job Shop Scheduling”, in Simulated Evolution and Learning: 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings, Dick, G., Browne, W. N., Whigham, P., Zhang, M., Bui, L. T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K. C., Tang, K., (ed.). Cham: Springer International Publishing, 2014, pp. 656–667, available at: http://link.springer.com/10.1007/978-3-319-13563-2_55

[223] Nguyen, S., Zhang, M., Tan, K. C., “Surrogate-Assisted Genetic Programming With Simplified Models for Automated Design of Dispatching Rules”, IEEE Transactions on Cybernetics, 2016, pp. 1–15, available at: http://ieeexplore.ieee.org/document/7473913/

[224] Ingimundardottir, H., Runarsson, T. P., “Evolutionary Learning of Linear Composite Dispatching Rules for Scheduling”, in Computational Intelligence: International Joint Conference, Merelo, J. J., Rosa, A., Cadenas, J. M., Dourado, A., Madani, K., Filipe, J., (ed.). Cham: Springer International Publishing, 2016, pp. 49–62, available at: http://link.springer.com/10.1007/978-3-319-26393-9_4

[225] Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M., “Investigating the generality of genetic programming based hyper-heuristic approach to dynamic job shop scheduling with machine breakdown”, in Artificial Life and Computational Intelligence: Third Australasian Conference, ACALCI 2017, Geelong, VIC, Australia, January 31 – February 2, 2017, Proceedings, Wagner, M., Li, X., Hendtlass, T., (ed.). Cham: Springer International Publishing, 2017, pp. 301–313, available at: https://doi.org/10.1007/978-3-319-51691-2_26

[226] Mei, Y., Nguyen, S., Zhang, M., "Evolving time-invariant dispatching rules in job shop scheduling with genetic programming", in Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P., (ed.). Cham: Springer International Publishing, 2017, pp. 147–163, available at: https://doi.org/10.1007/978-3-319-55696-3_10

[227] Karunakaran, D., Yi Mei, Gang Chen, Mengjie Zhang, "Evolving dispatching rules for dynamic Job shop scheduling with uncertain processing times", in 2017 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jun. 2017, pp. 364–371, available at: http://ieeexplore.ieee.org/document/7969335/

[228] Karunakaran, D., Mei, Y., Chen, G., Zhang, M., "Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty", in Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '17. New York, New York, USA: ACM Press, 2017, pp. 282–289, available at: http://dl.acm.org/citation.cfm?doid=3071178.3071202

[229] Lee, C.-Y., Piramuthu, S., Tsai, Y.-K., "Job shop scheduling with a genetic algorithm and machine learning", International Journal of Production Research, Vol. 35, No. 4, Apr. 1997, pp. 1171–1191, available at: http://www.tandfonline.com/doi/abs/10.1080/002075497195605

[230] Koonce, D., Tsai, S.-C., "Using data mining to find patterns in genetic algorithm solutions to a job shop schedule", Computers & Industrial Engineering, Vol. 38, No. 3, oct 2000, pp. 361–374, available at: https://doi.org/10.1016/s0360-8352(00)00050-4

[231] Li, X., Olafsson, S., "Discovering Dispatching Rules Using Data Mining", Journal of Scheduling, Vol. 8, No. 6, Dec. 2005, pp. 515–527, available at: http://link.springer.com/10.1007/s10951-005-4781-0

[232] Olafsson, S., Li, X., "Learning effective new single machine dispatching rules from optimal scheduling data", International Journal of Production Economics, Vol. 128, No. 1, Nov. 2010, pp. 118–126, available at: http://linkinghub.elsevier.com/retrieve/pii/S0925527310002124

[233] Ingimundardottir, H., Runarsson, T. P., "Supervised learning linear priority dispatch rules for job-shop scheduling", in Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers, Coello, C. A. C., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 263–277, available at: https://doi.org/10.1007/978-3-642-25566-3_20

[234] El-Bouri, A., Balakrishnan, S., Popplewell, N., “Sequencing jobs on a single machine: A neural network approach”, European Journal of Operational Research, Vol. 126, No. 3, Nov. 2000, pp. 474–490, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221799003021

[235] Weckman, G. R., Ganduri, C. V., Koonce, D. A., “A neural network job-shop scheduler”, Journal of Intelligent Manufacturing, Vol. 19, No. 2, Apr. 2008, pp. 191–201, available at: http://link.springer.com/10.1007/s10845-008-0073-9

[236] Eguchi, T., Oba, F., Toyooka, S., “A robust scheduling rule using a Neural Network in dynamically changing job-shop environments”, International Journal of Manufacturing Technology and Management, Vol. 14, No. 3/4, 2008, pp. 266, available at: http://www.inderscience.com/link.php?id=17727

[237] Petrovic, S., Fayad, C., Petrovic, D., Burke, E., Kendall, G., “Fuzzy job shop scheduling with lot-sizing”, Annals of Operations Research, Vol. 159, No. 1, dec 2007, pp. 275–292, available at: https://doi.org/10.1007/s10479-007-0287-9

[238] Kapanoglu, M., Alikalfa, M., “Learning IF–THEN priority rules for dynamic job shops using genetic algorithms”, Robotics and Computer-Integrated Manufacturing, Vol. 27, No. 1, feb 2011, pp. 47–55, available at: https://doi.org/10.1016/j.rcim.2010.06.001

[239] PRIORE, P., DE LA FUENTE, D., GOMEZ, A., PUENTE, J., “A review of machine learning in dynamic scheduling of flexible manufacturing systems”, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol. 15, No. 3, 2001, pp. 251–263.

[240] Shahzad, A., Mebarki, N., “Learning dispatching rules for scheduling: A synergistic view comprising decision trees, tabu search and simulation”, Computers, Vol. 5, No. 1, 2016, available at: http://www.mdpi.com/2073-431X/5/1/3

[241] Nguyen, S., Mei, Y., Ma, H., Chen, A., Zhang, M., “Evolutionary scheduling and combinatorial optimisation: Applications, challenges, and future directions”, in 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2016, pp. 3053–3060, available at: http://ieeexplore.ieee.org/document/7744175/

[242] Mann, H. B., Whitney, D. R., “On a test of whether one of two random variables is stochastically larger than the other”, The Annals of Mathematical Statistics, Vol. 18, No. 1, mar 1947, pp. 50–60, available at: https://doi.org/10.1214/aoms/1177730491

[243] McGill, R., Tukey, J. W., Larsen, W. a., “Variations of Box Plots”, The American Statistician, Vol. 32, No. 1, 1978, pp. 12–16.

[244] Frigge, M., Hoaglin, D. C., Iglewicz, B., "Some Implementations of the Boxplot", The American Statistician, Vol. 43, No. 1, 1989, pp. 50–54, available at: http://links.jstor.org/sici?sici=0003-1305(198902)43:1<50:SIOTB>2.0.CO;2-E

[245] Fowler, L., Pfund, M., Yu, L., Fowler, J. W., Carlyle, W. M., "Development of a robust scheduling rule for a printed wiring board drilling operation with multiple scheduling objectives and fixed order release/pickup times", in IIE Annual Conference. Proceedings. Institute of Industrial and Systems Engineers (IISE), 2002, pp. 1.

[246] YU, L., SHIH, H. M., PFUND, M., MATTHEW CARLYLE, W., FOWLER, J. W., "Scheduling of unrelated parallel machines: an application to PWB manufacturing", IIE Transactions, Vol. 34, No. 11, Nov. 2002, pp. 921–931, available at: http://www.tandfonline.com/doi/abs/10.1080/07408170208928923

[247] Kolahan, F., Kayvanfar, V., "A heuristic algorithm approach for scheduling of multi-criteria unrelated parallel machines", in World Academy of Science, Engineering and Technology, 2009.

[248] Lin, Y.-K., Fowler, J. W., Pfund, M. E., "Multiple-objective heuristics for scheduling unrelated parallel machines", European Journal of Operational Research, Vol. 227, No. 2, Jun. 2013, pp. 239–253, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221712007357

[249] Pfund, M., Fowler, J. W., Gupta, J. N. D., "A SURVEY OF ALGORITHMS FOR SINGLE AND MULTI-OBJECTIVE UNRELATED PARALLEL-MACHINE DETERMINISTIC SCHEDULING PROBLEMS", Journal of the Chinese Institute of Industrial Engineers, Vol. 21, No. 3, Jan. 2004, pp. 230–241, available at: http://www.tandfonline.com/doi/abs/10.1080/10170660409509404

[250] Tian, Y., Cheng, R., Zhang, X., Jin, Y., "Platemo: A matlab platform for evolutionary multi-objective optimization.", Vol. abs/1701.00879, 2017, available at: http://dblp.uni-trier.de/db/journals/corr/corr1701.html#TianCZJ17

[251] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, Apr. 2002, pp. 182–197, available at: http://ieeexplore.ieee.org/document/996017/

[252] Okabe, T., Yaochu Jin, Sendhoff, B., "A critical survey of performance indices for multi-objective optimisation", in The 2003 Congress on Evolutionary Computation, 2003. CEC '03., Vol. 2. IEEE, 2003, pp. 878–885, available at: http://ieeexplore.ieee.org/document/1299759/

[253] Siwei Jiang, Yew-Soon Ong, Jie Zhang, Liang Feng, “Consistencies and Contradictions of Performance Metrics in Multiobjective Optimization”, IEEE Transactions on Cybernetics, Vol. 44, No. 12, Dec. 2014, pp. 2391–2404, available at: http://ieeexplore.ieee.org/document/6766232/

[254] Zitzler, E., Thiele, L., “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach”, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 4, 1999, pp. 257–271, available at: http://ieeexplore.ieee.org/document/797969/

[255] van Veldhuizen, D. A., Lamont, G. B., “Multiobjective evolutionary algorithm test suites”, in Proceedings of the 1999 ACM symposium on Applied computing - SAC ’99. New York, New York, USA: ACM Press, 1999, pp. 351–357, available at: http://portal.acm.org/citation.cfm?doid=298151.298382

[256] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., da Fonseca, V., “Performance assessment of multiobjective optimizers: an analysis and review”, IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2, Apr. 2003, pp. 117–132, available at: http://ieeexplore.ieee.org/document/1197687/

[257] Coello Coello, Carlos, Lamont, Gary B., van Veldhuizen, D. A., Evolutionary Algorithms for Solving Multi-Objective Problems, ser. Genetic and Evolutionary Computation Series. Boston, MA: Springer US, 2007, available at: http://link.springer.com/10.1007/978-0-387-36797-2

[258] Wang, Z., Tang, K., Yao, X., “Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems”, IEEE Transactions on Reliability, Vol. 59, No. 3, Sep. 2010, pp. 563–575, available at: http://ieeexplore.ieee.org/document/5549979/

[259] Qingfu Zhang, Hui Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”, IEEE Transactions on Evolutionary Computation, Vol. 11, No. 6, Dec. 2007, pp. 712–731, available at: http://ieeexplore.ieee.org/document/4358754/

[260] Deb, K., Jain, H., “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints”, IEEE Transactions on Evolutionary Computation, Vol. 18, No. 4, Aug. 2014, pp. 577–601, available at: http://ieeexplore.ieee.org/document/6600851/

[261] Ishibuchi, H., Setoguchi, Y., Masuda, H., Nojima, Y., “Performance of Decomposition-Based Many-Objective Algorithms Strongly Depends on Pareto Front Shapes”, IEEE

Transactions on Evolutionary Computation, Vol. 21, No. 2, Apr. 2017, pp. 169–190, available at: http://ieeexplore.ieee.org/document/7509682/

[262] Ishibuchi, H., Imada, R., Setoguchi, Y., Nojima, Y., "Performance comparison of NSGA-II and NSGA-III on various many-objective test problems", in 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2016, pp. 3045–3052, available at: http://ieeexplore.ieee.org/document/7744174/

[263] Zitzler, E., Laumanns, M., Thiele, L., "SPEA2: Improving the Strength Pareto Evolutionary Algorithm", Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, 2001, pp. 95–100.

[264] Gomez, R. H., Coello, C. A., "MOMBI: A new metaheuristic for many-objective optimization based on the R2 indicator", in 2013 IEEE Congress on Evolutionary Computation, CEC 2013, 2013, pp. 2488–2495.

[265] Jain, H., Deb, K., "An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: Handling constraints and extending to an adaptive approach", IEEE Transactions on Evolutionary Computation, Vol. 18, No. 4, 2014, pp. 602–622.

[266] Polikar, R., "Ensemble learning", Scholarpedia, Vol. 4, No. 1, 2009, pp. 2776, revision #91224.

[267] Breiman, L., "Bagging Predictors", Machine Learning, Vol. 24, No. 2, 1996, pp. 123–140, available at: http://link.springer.com/10.1023/A:1018054314350

[268] Freund, Y., Schapire, R. E., "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", Journal of Computer and System Sciences, Vol. 55, No. 1, Aug. 1997, pp. 119–139, available at: http://linkinghub.elsevier.com/retrieve/pii/S002200009791504X

[269] Bhowan, U., Johnston, M., Zhang, M., Yao, X., "Evolving Diverse Ensembles Using Genetic Programming for Classification With Unbalanced Data", IEEE Transactions on Evolutionary Computation, Vol. 17, No. 3, Jun. 2013, pp. 368–386, available at: http://ieeexplore.ieee.org/document/6198882/

[270] Bhowan, U., Johnston, M., Mengjie Zhang, Xin Yao, "Reusing Genetic Programming for Ensemble Selection in Classification of Unbalanced Data", IEEE Transactions on Evolutionary Computation, Vol. 18, No. 6, Dec. 2014, pp. 893–908, available at: http://ieeexplore.ieee.org/document/6677603/

[271] Folino, G., Pizzuti, C., Spezzano, G., “Training Distributed GP Ensemble With a Selective Algorithm Based on Clustering and Pruning for Pattern Classification”, IEEE Transactions on Evolutionary Computation, Vol. 12, No. 4, Aug. 2008, pp. 458–468, available at: http://ieeexplore.ieee.org/document/4439200/

[272] Folino, G., Pizzuti, C., Spezzano, G., “Gp ensemble for distributed intrusion detection systems”, in Proceedings of the Third International Conference on Advances in Pattern Recognition - Volume Part I, ser. ICAPR’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 54–62, available at: http://dx.doi.org/10.1007/11551188_6

[273] Iba, H., “Bagging, boosting, and bloating in genetic programming”, in Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2, ser. GECCO’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1053–1060, available at: http://dl.acm.org/citation.cfm?id=2934046.2934063

[274] Paris, G., Robilliard, D., Fonlupt, C., “Applying boosting techniques to genetic programming”, in Artificial Evolution: 5th International Conference, Evolution Artificielle, EA 2001 Le Creusot, France, October 29–31, 2001 Selected Papers, Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 267–278, available at: https://doi.org/10.1007/3-540-46033-0_22

[275] de Souza, L. V., R. Pozo, A. T., C., A., da Ros, J. M. C., “Genetic Programming and Boosting Technique to Improve Time Series Forecasting”, in Evolutionary Computation. InTech, Oct. 2009, available at: http://www.intechopen.com/books/evolutionary-computation/genetic-programming-and-boosting-technique-to-improve-time-series-forecasting

[276] Potter, M. A., Jong, K. A. D., “A cooperative coevolutionary approach to function optimization”, in Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, ser. PPSN III. London, UK, UK: Springer-Verlag, 1994, pp. 249–257, available at: http://dl.acm.org/citation.cfm?id=645822.670374

[277] Branke, J., Pickardt, C. W., “Evolutionary search for difficult problem instances to support the design of job shop dispatching rules”, European Journal of Operational Research, Vol. 212, No. 1, Jul. 2011, pp. 22–32, available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221711000981

[278] PIERREVAL, H., “Expert system for selecting priority rules in flexible manufacturing

systems", Expert Systems with Applications, Vol. 5, No. 1-2, 1992, pp. 51–57, available at: http://linkinghub.elsevier.com/retrieve/pii/0957417492900949

[279] Sun, Y.-L., Yih, Y., "An intelligent controller for manufacturing cells", International Journal of Production Research, Vol. 34, No. 8, Aug. 1996, pp. 2353–2373, available at: http://www.tandfonline.com/doi/abs/10.1080/00207549608905029

[280] Pierreval, H., "Neural Network to Select Dynamic Scheduling Heuristics", Journal of Decision Systems, Vol. 2, No. 2, Jan. 1993, pp. 173–190, available at: http://www.tandfonline.com/doi/abs/10.1080/12460125.1993.10511572

[281] Liu, H., Dong, J. J., "Dispatching rule selection using artificial neural networks for dynamic planning and scheduling", Journal of Intelligent Manufacturing, Vol. 7, No. 3, Jun. 1996, pp. 243–250, available at: http://link.springer.com/10.1007/BF00118083

[282] Pierreval, H., Mebarki, N., "Dynamic scheduling selection of dispatching rules for manufacturing system", International Journal of Production Research, Vol. 35, No. 6, Jun. 1997, pp. 1575–1591, available at: http://www.tandfonline.com/doi/abs/10.1080/002075497195137

[283] Lian Yu, Shih, H., Sekiguchi, T., "Dynamic selection of dispatching rules by fuzzy inference", in 1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36228), Vol. 2. IEEE, 1998, pp. 979–984, available at: http://ieeexplore.ieee.org/document/686251/

[284] Subramaniam, V., Ramesh, T., Lee, G. K., Wong, Y. S., Hong, G. S., "Job Shop Scheduling with Dynamic Fuzzy Selection of Dispatching Rules", The International Journal of Advanced Manufacturing Technology, Vol. 16, No. 10, Aug. 2000, pp. 759–764, available at: http://link.springer.com/10.1007/s001700070029

[285] Subramaniam, V., Lee, G. K., Hong, G. S., Wong, Y. S., Ramesh, T., "Dynamic selection of dispatching rules for job shop scheduling", Production Planning & Control, Vol. 11, No. 1, Jan. 2000, pp. 73–81, available at: http://www.tandfonline.com/doi/abs/10.1080/095372800232504

[286] Shiue, Y.-R., Guh, R.-S., "Learning-based multi-pass adaptive scheduling for a dynamic manufacturing cell environment", Robotics and Computer-Integrated Manufacturing, Vol. 22, No. 3, Jul. 2006, pp. 203–216, available at: http://linkinghub.elsevier.com/retrieve/pii/S0736584505000402

[287] Zahmani, M. H., Atmani, B., Bekrar, A., Aissani, N., "Multiple priority dispatching rules for the job shop scheduling problem", in 2015 3rd International Conference on

Control, Engineering & Information Technology (CEIT). IEEE, May 2015, pp. 1–6, available at: http://ieeexplore.ieee.org/document/7232991/

[288] Priore, P., Gómez, A., Pino, R., Rosillo, R., “Dynamic scheduling of manufacturing systems using machine learning: An updated review”, Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, Vol. 28, No. 1, 2014, pp. 83–97, available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84896442633&partnerID=40&md5=c6d5b3ad421a09adf167cf12f86aa134

[289] Fix, E., Hodges, J. L., “Discriminatory analysis, nonparametric discrimination: Consistency properties”, US Air Force School of Aviation Medicine, Vol. Technical Report 4, 1951.

[290] Alpaydin, E., Introduction to Machine Learning. The MIT Press, 2014.

[291] Cox, D. R., “The regression analysis of binary sequences (with discussion)”, J Roy Stat Soc B, Vol. 20, 1958, pp. 215–242.

[292] Cortes, C., Vapnik, V., “Support-vector networks”, Machine Learning, Vol. 20, No. 3, Sep. 1995, pp. 273–297, available at: http://link.springer.com/10.1007/BF00994018

[293] Werbos, P. J., “Beyond regression: New tools for prediction and analysis in the behavioral sciences”, Phd thesis, Harvard University, 1974.

[294] Quinlan, J. R., C4.5: Programs for Machine Learning. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[295] Souza, C. The accord.net framework, available at: http://accord-framework.net

[296] Bertsekas, D., Castanon, D., “Rollout algorithms for stochastic scheduling problems”, in Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171), Vol. 2. IEEE, 1998, pp. 2143–2148, available at: http://ieeexplore.ieee.org/document/758655/

[297] Bertsekas, D., Castanon, D., “Rollout algorithms for stochastic scheduling problems”, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR INFORMATION AND DECISION SYSTEMS, Tech. Rep., 1998.

[298] Bertsekas, D. P., Castanon, D. A., “Rollout Algorithms for Stochastic Scheduling Problems”, Journal of Heuristics, Vol. 5, No. 1, 1999, pp. 89–108, available at: http://link.springer.com/10.1023/A:1009634810396

[299] Geirsson, E., “Rollout algorithms for job-shop scheduling”.

[300] Bertsekas, D. P., Rollout Algorithms for Discrete Optimization: A Survey. New York, NY: Springer New York, 2013, pp. 2989–3013, available at: http://dx.doi.org/10.1007/978-1-4419-7997-1_8

List of Figures

List of Tables

List of Algorithms

Biography

Marko Đurasević was born on the eleventh of August 1990 in Zagreb, Croatia. In 2009, he enrolled the Faculty of Electrical Engineering and Computing, on which he completed the undergraduate study in computing. On the same faculty in 2014 he graduated in the field of computing with great honour with the topic "Optimisation of scheduling in the unrelated machines environment", under the mentorship of Professor Domagoj Jakovović. During his graduation studies, he was awarded the "Josip Lončar" award for his outstanding achievements in the first year of graduation studies, as well as the "bronze plaque Josip Lončar" for his overall achievement at the graduate study. Together with Dino Šantl he was awarded the Rector's award for their work under the title "Implementation of Medical Visual Data Compression Algorithm on GPU for Increased Throughput and Reduced Energy Consumption", which was mentored by Associate Professor Josip Knezović.

Since October 2014 he has been employed as a teaching and research assistant at the Department of Electronics, Microelectronics, Computer and Intelligent Systems of the Faculty of Electrical Engineering and Computing. He is actively involved in teaching of the graduate course "Computer Analysis and Design", and the undergraduate course "Interactive Computer Graphics". He also assisted in mentoring for an undergraduate and graduate thesis. His research interests include the field of evolutionary computing, optimisation methods, machine learning and scheduling problems. In the current research process, he published three papers in CC-indexed journals and three conference papers.

List of published works

Papers in journals

1. Đurasević, M., Jakobović, D., Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment, Genetic Programming and Evolvable Machines, 2017.
2. Đurasević, M., Jakobović, D., Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment, Genetic Programming and Evolvable Machines, 2017.

3. Đurasević, M., Jakobović, D., Knežević, K., Adaptive scheduling on unrelated machines with genetic programming, Applied Soft Computing, Vol. 48, November 2016, 419-430.

Papers at international scientific conferences

1. Đurasević, M., Jakobović, D., Comparison of solution representations for scheduling in the unrelated machines environment, 39th International Convention MIPRO (2016), June 2016, 1336-1342.
2. Šantl, D., Đurasević, M., Knezović, J., GPU Implementation of a Medical Imaging Data Compression Algorithm, IT Systems 2013, e-Health 2013 Electronic Proceedings, September 2013.
3. Dujak, M., Parać, V., Đurasević, M., Herić, A., Machine-to-machine communication as key enabler in smart metering systems, 36th International Convention MIPRO (2013), June 2013, 409-414.

Životopis

Marko Đurasević rođen je 11. 08. 1990. godine u Zagrebu, Hrvatska. U 2009. godini upisuje Fakultet elektrotehnike i računarstva, na kojem 2012. godine završava preddiplomski studij računarstva. Na istom fakultetu je 2014. godine diplomirao je s velikom pohvalom na smjeru računarstvo, s temom "Optimizacija raspoređivanja u okruženju nesrodnih strojeva", pod mentorstvom prof. dr. sc. Domagoja Jakobovića. Tijekom diplomskog studija nagrađen je nagradom "Josip Lončar" za izvrstan uspjeh na prvoj godini diplomskog studija, kao i "brončanom plaketom Josip Lončar" za ukupno ostvaren uspjeh na diplomskom studiju. Zajedno s Dinom Šantlom dobitnik je Rektorove nagrade za njihov rad pod naslovom "Izvedba algoritma za kompresiju medicinskih slikovnih podataka korištenjem grafičkog procesora s ciljem povećanja propusnosti i smanjenja utrošene energije", pod mentorstvom izv. prof. dr. sc. Josipa Knezovića.

Od listopada 2014. godine zaposlen je kao asistent na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva. Sudjeluje u izvođenju diplomskog kolegija "Analiza i projektiranje računalom" te preddiplomskog kolegija "Interaktivna računalna grafika". Također je asistirao u vođenju jednog završnog i diplomskog rada. Njegovi istraživački interesi obuhvaćaju područje evolucijskog računarstva, metoda optimizacija, strojnog učenja te problema raspoređivanja. U dosadašnjem tijeku istraživanja objavio je tri rada u časopisima indeksiranim u bazi CC i tri konferencijska rada.