SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Domagoj Jakobović

RASPOREĐIVANJE ZASNOVANO NA PRILAGODLJIVIM PRAVILIMA

DOKTORSKA DISERTACIJA

Zagreb, 2005.

Disertacija je izrađena na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu

Mentor: prof.dr.sc. Leo Budin

Disertacija ima 133 stranice.

Disertacija br.: _____.

Povjerenstvo za ocjenu doktorske disertacije:

1. Dr. sc. Vedran Mornar, red. prof. FER-a Zagreb
2. Dr. sc. Leo Budin, akademik red. prof. FER-a Zagreb
3. Dr. sc. Goran Martinović, doc. Elektrotehničkog fakulteta Osijek

Povjerenstvo za obranu doktorske disertacije:

1. Dr. sc. Vedran Mornar, red. prof. FER-a Zagreb
2. Dr. sc. Leo Budin, akademik red. prof. FER-a Zagreb
3. Dr. sc. Goran Martinović, doc. Elektrotehničkog fakulteta Osijek
4. Dr. sc. Nikola Bogunović, red. prof. FER-a Zagreb
5. Dr. sc. Bojana Dalbelo Bašić, izv. prof. FER-a Zagreb

Datum obrane disertacije: 07. prosinca 2005. godine

Sadržaj

1 UVOD 1

2 PREGLED ODABRANIH PROBLEMA RASPOREĐIVANJA 4

2.1 POSTUPAK RASPOREĐIVANJA 4
2.1.1 Pojam raspoređivanja 4
2.1.2 Svojstva poslova 4
2.2 ZAPIS OKRUŽENJA RASPOREĐIVANJA 5
2.2.1 Elementi zapisa okruženja raspoređovanja 5
2.2.2 Okolina strojeva 6
2.2.3 Okolina zadataka 7
2.2.4 Vrednovanje rasporeda 8
2.2.5 Manje česta okruženja raspoređivanja 10
2.3 UVJETI RASPOREĐIVANJA 11
2.3.1 Podjela uvjeta raspoređivanja 11
2.3.2 Načini izrade rasporeda 12
2.4 POSTUPCI RASPOREĐIVANJA 13
2.4.1 Složenost algoritama raspoređivanja 13
2.4.2 Heuristički postupci raspoređivanja 14

3 GENETSKO PROGRAMIRANJE 16

3.1 STRUKTURA GENETSKOG PROGRAMIRANJA 17
3.1.1 Nastanak genetskog programiranja 17
3.1.2 Elementi genetskog programiranja 17
3.1.3 Primjer uporabe genetskog programiranja 20
3.2 PODRUČJA ISTRAŽIVANJA NA POLJU GENETSKOG PROGRAMIRANJA 22
3.2.1 Predstavljanje rješenja 22
3.2.2 Funkcija dobrote 23
3.2.3 Parametri i oblik genetskog programa 24
3.2.4 Korištenje spremničkog prostora 24
3.2.5 Operatori genetskog programiranja 25
3.2.6 Konvergencija i zalihost rješenja genetskog programiranja 26
3.3 PRIMJENE GENETSKOG PROGRAMIRANJA 27

4 RASPOREĐIVANJE ZASNOVANO NA PRAVILIMA 29

4.1 PRIMJENA PRAVILA RASPOREĐIVANJA 29
4.1.1 Raspoređivanje uz pomoć pravila 29
4.1.2 Vrste pravila raspoređivanja 30
4.1.3 Prednosti i nedostaci pravila raspoređivanja 30
4.2 PARAMETRI GENETSKOG PROGRAMIRANJA 31
4.2.1 Struktura genetskog programa 31
4.2.2 Definiranje funkcije cilja 33
4.2.3 Oblikovanje ispitnih primjera 34
4.3 OSTVARENJE ISPITNE OKOLINE 36
4.4 RASPOREĐIVANJE NA JEDNOM STROJU 37
4.4.1 Okruženje jednoga stroja 37
4.4.2 Pregled raspoređivanja na jednom stroju uz pomoć genetskog programiranja 38
4.4.3 Statička okolina raspoređivanja 39
4.4.4 Dinamička okolina raspoređivanja 48

4.4.5 Ograničenja u redoslijedu zadataka *52*
4.4.6 Slijedno ovisna trajanja postavljanja *56*
4.4.7 Pravila raspoređivanja u složenijim okolinama *61*
4.5 RASPOREĐIVANJE NA PARALELNIM JEDNOLIKIM STROJEVIMA 66
4.5.1 Okruženje paralelnih strojeva *66*
4.5.2 Oblikovanje ispitnih primjera *67*
4.5.3 Raspoređivanje uz pomoć pravila na jednolikim strojevima *69*
4.5.4 Statička okolina raspoređivanja *70*
4.5.5 Dinamička okolina raspoređivanja *72*
4.6 RASPOREĐIVANJE NA PARALELNIM NESRODNIM STROJEVIMA 76
4.6.1 Okruženje nesrodnih strojeva *76*
4.6.2 Postupci raspoređivanja na nesrodnim strojevima *77*
4.6.3 Raspoređivanje uz pomoć genetskog programiranja *78*
4.6.4 Oblikovanje ispitnih primjera *80*
4.6.5 Raspoređivanje na nesrodnim strojevima *80*
4.7 RASPOREĐIVANJE ZA PROIZVOLJNU OBRADU 83
4.7.1 Okruženje proizvoljne obrade *83*
4.7.2 Pregled raspoređivanja proizvoljne obrade uz pomoć genetskog programiranja *84*
4.7.3 Postupci raspoređivanja za proizvoljnu obradu *84*
4.7.4 Oblikovanje ispitnih primjera *87*
4.7.5 Raspoređivanje uz pomoć genetskog programiranja *88*
4.8 METODOLOGIJA IZVOĐENJA PRAVILA RASPOREĐIVANJA 93
4.8.1 Izvođenje pravila raspoređivanja za proizvoljnu okolinu *93*
4.8.2 Osvrt na predložene metode *94*

5 POBOLJŠAVANJE POSTUPAKA IZVOĐENJA PRAVILA RASPOREĐIVANJA 96

5.1 ODABIR PODATKOVNIH ELEMENATA RJEŠENJA 97
5.1.1 Problem potpunosti *97*
5.1.2 Metoda vrednovanja podatkovnih čvorova *98*
5.1.3 Ispitivanje učinkovitosti metode vrednovanja čvorova *100*
5.2 PRILAGODBA GENETSKIH OPERATORA 102
5.2.1 Postupak prilagodbe operatora *102*
5.2.2 Određivanje stanja populacije *102*
5.2.3 Prilagodljivi genetski operatori *103*
5.2.4 Ispitivanje učinkovitosti prilagodbe operatora *104*
5.3 PRIMJENA POSTUPAKA PRILAGODBE 107
5.3.1 Problem raspoređivanja u dinamičkim uvjetima *107*
5.4 UBRZAVANJE GENETSKOG PROGRAMIRANJA 108
5.4.1 Određivanje dobrote rješenja *108*
5.4.2 Vektorsko ocjenjivanje rješenja *109*
5.4.3 Uređivanje rješenja *110*

6 ZAKLJUČAK 113

LITERATURA 116

PRILOG A: PARAMETRI ISPITNIH PRIMJERA 124

A1. Okruženje jednoga stroja *124*
A2. Okruženje paralelnih jednolikih strojeva *125*

A3. Okruženje paralelnih nesrodnih strojeva 126
A4. Okruženje proizvoljne obrade 127

PRILOG B: IZVEDENA PRAVILA RASPOREĐIVANJA 129

B1. Okruženje jednoga stroja 129
B2. Okruženje paralelnih jednolikih strojeva 131
B3. Okruženje paralelnih nesrodnih strojeva 132
B4. Okruženje proizvoljne obrade 133

1 Uvod

Raspoređivanje je proces koji se bavi dodjelom ograničenih sredstava skupu aktivnosti u cilju optimiranja jednog ili više mjerila vrednovanja. Ovisno o danim uvjetima, sredstva i aktivnosti mogu poprimiti mnoge oblike. Sredstva (engl. *resources*) mogu biti strojevi u proizvodnom pogonu, procesor računala, spremnički i komunikacijski uređaji u računalnom sustavu, piste na aerodromu, ljudska radna snaga u postupku održavanja itd. Aktivnosti mogu biti razne operacije u proizvodnji, izvođenja računalnih programa, popravci u radionici, slijetanje ili uzlijetanje aviona i druge. Rješenje problema raspoređivanja je raspored koji definira kada će se i na kojem sredstvu odvijati pojedina aktivnost. Vrednovanje rasporeda moguće je definirati po puno različitih kriterija, kao što je ukupno vrijeme izvođenja, broj zakašnjelih aktivnosti itd.

Proučavanje problema raspoređivanja je disciplina koja se razvija više od pola stoljeća. Smanjeni troškovi uporabe sredstava bili su najveći razlog i ekonomska osnovica proučavanja načina raspoređivanja. Usporedo s razvojem tehnologije, problemi raspoređivanja postaju sve složeniji i sve je teže naći učinkovite algoritme za njihovo rješavanje. Rezultati dobiveni uz pomoć teorije složenosti pokazuju da je velik dio takvih problema po svojoj prirodi vrlo težak za rješavanje. Složenost raspoređivanja općenito raste s povećanjem složenosti okoline u kojoj se raspoređivanje odvija. Za većinu problema nije moguće razviti ili upotrijebiti egzaktni algoritam, pa se, u cilju dobivanja rješenja koje ne mora biti optimalno već samo 'dovoljno dobro', razvijaju heuristički algoritmi.

Heuristički postupci se u današnjim uvjetima koriste u većini problema raspoređivanja iako ne jamče pronalaženje optimalnog rješenja. Ovisno o uvjetima rada, rješavanju problema raspoređivanja heurističkim postupcima možemo pristupiti na dva načina. Ukoliko su veličine koje određuju svojstva problema unaprijed poznate, ne mijenjaju se znatno tijekom rada sustava te ukoliko izrada rješenja problema nije vremenski ograničena, uporabljeni postupak rješavanja može koristiti neke od metoda pretraživanja prostora stanja (engl. *search based optimization*), kao što su dinamičko programiranje, tabu pretraživanje (engl. *tabu search*), simulirano kaljenje (engl. *simulated annealing*), genetski algoritmi (engl. *genetic algorithms*), ograničeno pretraživanje stabla (engl. *branch-and-bound*), zrakasto pretraživanje (engl. *beam search*), neuronske mreže itd. Svim ovim postupcima zajedničko je da su obično vremenski zahtjevni te stoga primjenjivi samo u statičkom okruženju raspoređivanja.

S druge strane, rješenje je moguće dobiti i algoritmima koji ne pretražuju prostor mogućih rješenja već grade raspored u konačnom broju koraka. Njihovo je trajanje obično mnogo kraće od metoda pretraživanja, što omogućuje uporabu u dinamičkim uvjetima rada ili uz promjenjive parametre sustava. Takvi algoritmi u većini slučajeva imaju jednostavniju strukturu i zahtijevaju manje troškove ugradnje, no uz cijenu nerijetko slabije učinkovitosti od postupaka pretraživanja. U velikom broju okruženja, uvjeti raspoređivanja ne dozvoljavaju uporabu dugotrajnih metoda pa su ovi algoritmi često jedino moguće rješenje. U slučajevima kada je omogućena uporaba vremenski zahtjevnijih postupaka pretraživanja, heuristički algoritmi stvaranja rasporeda mogu se iskoristiti za dobivanje početnog rješenja problema koje u velikoj mjeri utječe na kvalitetu krajnjeg rješenja. Štoviše, u velikom broju slučajeva je rješenje dobiveno heurističkim algoritmom moguće tek neznatno poboljšati uporabom postupka pretraživanja, pa je u tim uvjetima uporaba vremenski složenijih postupaka neisplativa.

Različita okruženja i uvjeti raspoređivanja, te ujedno i različita mjerila vrednovanja rasporeda, zahtijevaju uporabu algoritma koji će biti najprimjereniji u dotičnoj situaciji. Za velik broj specifičnih okruženja i kriterija definirani su algoritmi za koje se očekuje da daju zadovoljavajuće rješenje. Nažalost, u stvarnosti se okolina raspoređivanja često razlikuje od uvjeta za koji su ti algoritmi namijenjeni, pa dobiveni rezultati nisu dovoljno dobri ili rezultat koji algoritam daje uopće nije primjenjiv (jer primjerice narušava neka od ograničenja). Često je postojeći algoritam potrebno prilagoditi specifičnim uvjetima, što zahtijeva dobro poznavanje mehanizama raspoređivanja. Takav pristup stoga iziskuje više vremena za prilagodbu te eventualno stručno znanje o radu sustava, a samim time i veće troškove. U velikom broju slučajeva bilo bi isplativo upotrijebiti postupak koji bi olakšao prilagodbu ili oblikovanje heurističkog algoritma raspoređivanja namijenjenog dotičnoj okolini. U ovom se radu opisuje upravo takav pristup: automatizirano stvaranje algoritma koji bi bio najpogodniji za raspoređivanje sustava s jedinstveno definiranim uvjetima i kriterijima. Predloženi postupak ubraja se u metode strojnog učenja, a tehnologija koja se u tu svrhu koristi je genetsko programiranje.

Genetsko programiranje (engl. *genetic programming*, *GP*) [Koz 92, Ban 98] je jedan od pokušaja odgovora na pitanje kako navesti računalo da riješi neki zadatak bez davanja izravnih uputa o postupku rješavanja; drugim riječima, kako postići da računalo učini nešto korisno a da mu nismo rekli *kako* to treba učiniti (ovakav način rješavanja problema se može nazvati još i *automatsko programiranje*). Genetsko programiranje pripada skupini postupaka koji se zajednički nazivaju evolucijskim računanjem (engl. *evolutionary computation*, *EC*), među koje se još ubrajaju genetski algoritmi, evolucijske strategije itd. Kao i kod svih postupaka strojnog učenja, računalu ne kažemo kako treba riješiti neki problem, ali definiramo način na koji može *naučiti* rješavanje nekog problema. U slučaju evolucijskog računanja, postupak održava skup rješenja (populaciju) nad kojima primjenjuje operatore koji oponašaju proces evolucije u prirodi. Posebnost genetskog programiranja je u tome što je svako rješenje predstavljeno strukturom promjenjive veličine i oblika, što je idealna postavka za predstavljanje računalnih programa. Na kraju postupka, najbolje rješenje (ili više njih) u skupu predstavlja rješenje originalnog problema.

U ovom radu predstavljen je način izvođenja heurističkih algoritama raspoređivanja uporabom genetskog programiranja. Dobiveni algoritmi imaju predefiniranu globalnu strukturu, dok se neki dijelovi pronalaze automatski, odnosno strojnim učenjem. Postupak raspoređivanja kojega algoritmi koriste je prioritetno raspoređivanje (engl. *priority scheduling*), pri kojemu se zadano trenutno stanje sustava preslikava u sljedeće stanje. Prijelaz iz jednoga u drugo stanje definiran je pridruživanjem određene aktivnosti nekome sredstvu, a odabir aktivnosti i sredstva obavlja se računanjem prioriteta svakog raspoloživog sredstva i aktivnosti, ovisno o dotičnom okruženju. Budući se načini raspoređivanja u raznolikim okolinama međusobno razlikuju, za pojedine vrste okruženja definirane su prikladne strukture algoritama. Ovakvi algoritmi izvode se vrlo brzo što im daje mogućnost primjene u dinamičkim uvjetima rada te uz uvjete raspoređivanja u kojima nemamo uvid u buduće događaje u sustavu. Također se mogu koristiti i u statičkom raspoređivanju, samostalno ili kao postupak generiranja početnog rješenja za složenije metode pronalaženja rasporeda, kojima se kvaliteta dobivenog rješenja može poboljšati.

Namjena postupaka raspoređivanja prikazanih u ovome radu nije pronalaženje optimalnog rješenja nekoga problema, već davanje rješenja prilagođenog zadanoj okolini, kriterijima raspoređivanja i eventualnim specifičnim potrebama korisnika. Ispunjavanje ovih zahtjeva postiže se definiranjem odgovarajućih podatkovnih struktura i građe rješenja te unapređivanjem postupka izvođenja pravila u procesu genetskog programiranja.

U općenitom je slučaju svrha strojnog učenja u tome da pronađeno rješenje nekoga problema bude po kvaliteti usporedivo sa rješenjima dobivenim uz pomoć stručnog ljudskog znanja, te da sustav učenja samostalno izvodi učinkovita rješenja za nove primjere problema. Namjera ovoga rada je pokazati da je korištenjem genetskog programiranja moguće stvaranje heurističkih algoritama raspoređivanja koji se po učinkovitosti mogu mjeriti s algoritmima iste namjene stvorenim od strane stručnjaka, te da je proces stvaranja takvih algoritama moguće u određenoj mjeri automatizirati.

Rad je podijeljen na nekoliko cjelina u kojima se postupno opisuje korištena metodologija i predlažu rješenja za dotična okruženja raspoređivanja. U drugom poglavlju opisani su problemi iz klasične teorije raspoređivanja kao i primjeri nekih problema koji nisu obuhvaćeni klasičnom podjelom. Definirani su glavni elementi okoline raspoređivanja i načini na koji se raspoređivanje može obaviti. Izložen je pregled najčešće korištenih načina raspoređivanja prilagođenih danim uvjetima okoline i kriterijima vrednovanja.

Poglavlje 3 opisuje metodu genetskog programiranja i najvažnije korake procesa pronalaženja rješenja. Opisani su elementi koji se moraju definirati unutar sustava za učenje i navedeni neki uobičajeni operatori postupka. Posebna pažnja posvećena je mogućnostima poboljšanja rada genetskog programiranja te područjima u kojima se ta poboljšanja nastoje postići. Postavke opisane na tom mjestu služit će kao polazište za uvođenje vlastitih metoda povećanja učinkovitosti u sljedećim poglavljima.

U 4. poglavlju opisan je način izvođenja algoritama raspoređivanja uz pomoć genetskog programiranja. Zasebna struktura algoritma definirana je za najznačajnije vrste uvjeta raspoređivanja, kao što su statička ili dinamička raspoloživost aktivnosti ili sredstava, ovisnosti u poretku aktivnosti itd. Dio algoritma koji se stvara u evolucijskom procesu odgovoran je za odabir 'najboljih' aktivnosti i sredstava. Ovisno o okolini raspoređivanja i zadanom kriteriju, algoritam koristi različite podatke u cilju određivanja prioriteta svakog elementa u rasporedu. U ovom su poglavlju izvedeni algoritmi za najčešće korištene kriterije raspoređivanja na jednom ili više strojeva te uspoređeni s odgovarajućim algoritmima iz literature i praktične uporabe. Također su pokazani primjeri algoritama za neka manje uobičajena okruženja raspoređivanja za koje je teže pronaći odgovarajuće algoritme, pogotovo uz proizvoljne kriterije.

Poglavlje 5 opisuje postupke uz pomoć kojih možemo poboljšati pronalaženje rješenja genetskog programiranja. Moguća poboljšanja se u prvom redu odnose na kvalitetu dobivenog rješenja te na veću vjerojatnost dobivanja zadovoljavajućeg rješenja, odnosno smanjenje trajanja učenja koje je potrebno kako bi se došlo do takvog rješenja. Jedan od najvažnijih čimbenika za pronalaženje uspješnog algoritma je odabir odgovarajućih podataka kojima će algoritam manipulirati. U radu se stoga definira metoda koja u procesu evolucije pomaže u identifikaciji relevantnih podataka među onima koji su programu na raspolaganju. Također su predložene vlastite metode kojima je cilj unaprijediti uporabu genetskih operatora nad članovima populacije. U zaključnom poglavlju izneseni su glavni elementi rada te, na temelju dobivenih rezultata, izložena ocjena korisnosti predloženog načina stvaranja algoritama raspoređivanja.

2 Pregled odabranih problema raspoređivanja

U ovom su poglavlju prikazani elementi koji čine problem raspoređivanja i neki pristupi rješavanju toga problema. Cilj ovoga poglavlja nije sustavni pregled teorije raspoređivanja i svih postojećih inačica navedenoga problema, što bi samo za sebe bilo dovoljno opširan i zahtjevan zadatak. Umjesto toga, na ovom su mjestu opisana ona okruženja raspoređivanja čiji postupci rješavanja zauzimaju ponajviše pažnje u literaturi i koja služe kao osnova za izgradnju nekih složenijih modela raspoređivanja. Postupci rješavanja tih složenijih modela često se temelje na postupcima razvijenim za, uvjetno rečeno, jednostavnije inačice. Pored pregleda najvažnijih modela iz 'klasične' teorije raspoređivanja, bit će navedene i neke manje česte, ali zato i ne manje složene okoline. Upravo se u slučaju takvih manje uobičajenih okruženja pokazuje potreba za razvojem algoritama posebno prilagođenih danim uvjetima, u kojim bi slučajevima sustav za (barem djelomično) automatizirano izvođenje algoritama raspoređivanja bio od posebne koristi.

2.1 Postupak raspoređivanja

2.1.1 Pojam raspoređivanja

Raspoređivanje je u širem smislu postupak izrade bilo kakvog rasporeda. U različitim okruženjima postupak izrade rasporeda može se podijeliti na više stupnjeva, ovisno o algoritmu i problemu. U problemima koji uključuju samo jedno sredstvo, raspoređivanje se sastoji od određivanja redoslijeda aktivnosti na tom sredstvu. Ako se koristi više odvojenih sredstava, prije izrade redoslijeda potrebno je odrediti koje aktivnosti će se odvijati na kojem sredstvu. U oba slučaja, postupak izrade cjelokupnog rasporeda nazivat će se raspoređivanjem. U problemima sa više sredstava, postupak dodjeljivanja aktivnosti sredstvima nazivat će se *pridruživanjem* (engl. *matching*), a postupak izrade redoslijeda na pojedinom sredstvu *uređivanjem* (engl. *sequencing*). Ovisno o postupku raspoređivanja, te dvije radnje mogu biti odvojene ili isprepletene.

Raspoređivanje je postupak koji se odvija u vrlo širokom skupu uvjeta, no uvijek uključuje obavljanje radnji (aktivnosti) koje zauzimaju određena sredstva u nekom vremenskom periodu. Radnje koje se obavljaju mogu se nazivati poslovima ili projektima, a mogu se sastojati i od manjih cjelina kao što su zadaci ili operacije. Svaka aktivnost zahtijeva određeno sredstvo u određenoj količini vremena. Sredstva također mogu imati elementarne dijelove koji se nazivaju strojevi, procesori ili ćelije, itd. Problemi raspoređivanja se često usložnjavaju raznim ograničenjima koja međusobno vezuju različite aktivnosti ili sredstva i aktivnosti ili oboje sa nekim vanjskim elementima. Na primjer, mogu biti postavljena ograničenja u redoslijedu čime se definira koje aktivnosti moraju prethoditi nekim drugima i to uz koje eventualno trajanje postavljanja itd. U cilju formalnog opisa problema, potrebno je definirati svojstva poslova koji se raspoređuju, svojstva sredstava koja se koriste u tom procesu kao i mjerila vrednovanja izrađenog rasporeda.

2.1.2 Svojstva poslova

Objekt čije izvođenje želimo omogućiti postupkom raspoređivanja najčešće se naziva posao (engl. *job*) ili zadatak (engl. *task*). U nekim okruženjima raspoređivanja posao se može sastojati od više aktivnosti ili operacija, dok se u nekim drugim uvjetima (npr. kod raspoređivanja na jednom stroju) smatra da se posao sastoji samo od jedne nedjeljive aktivnosti. Iako se pojmovi posla i zadatka često međusobno izmjenjuju, iz

konteksta dotičnog problema raspoređivanja uvijek će biti očito radi li se o jednoj ili o nizu aktivnosti.

Skup svih poslova se obično označava sa *J*, a pojedini posao sa J_j. Isto tako, skup zadataka se označava sa *T*, a pojedini zadatak sa T_j. Svakom od elemenata iz navedenih skupova pridružena su neka od svojstava opisanih u nastavku. Pri opisu svojstava i okoline raspoređivanja, pod pojmom 'vrijeme' podrazumijevat će se neki jedinstveni trenutak u vremenskom slijedu, dok će se pod pojmom 'trajanje' podrazumijevati određena količina vremena, odnosno razlika između dva jedinstvena trenutka.

Trajanje izvođenja (p_j)

Svaka nedjeljiva aktivnost zahtijeva određenu količinu vremena koja se naziva trajanje izvođenja (engl. *processing time*) i označava kao p_j, gdje je *j* indeks dotične aktivnosti (posla). Ako se radi o okruženju sa više strojeva, tada se trajanje izvođenja posla J_j na stroju *i* označava sa p_{ij}.

Vrijeme pripravnosti (r_j)

Vrijeme pripravnosti (engl. *ready time, release time*) ili vrijeme pojavljivanja je trenutak u kojemu posao dolazi u sustav, odnosno u kojemu postaje raspoloživ za izvođenje. Nijedan posao ne može se izvoditi prije svog vremena pripravnosti. S druge strane, posao može čekati početak izvođenja neodređeno dugo nakon svog vremena pripravnosti, bilo zbog zauzetosti sredstava, bilo zbog nekog drugog neispunjenog uvjeta.

Vrijeme željenog završetka (d_j)

Vrijeme željenog završetka (engl. *due date*) nekog posla je trenutak do kojega se očekuje da posao treba završiti s izvođenjem. Posao može završiti i nakon toga trenutka, ali se u tom slučaju stvara određeni trošak (npr. zbog kašnjenja isporuke naručitelju).

Vrijeme nužnog završetka ($\overline{d}_j$)

Vrijeme nužnog završetka (engl. *deadline, drop dead time*) je strogo vremensko ograničenje do kojega posao mora završiti. Ovo se ograničenje najčešće javlja u sustavima za rad u stvarnom vremenu.

Težina (w_j)

Težina (engl. *weight*) odnosno razina prvenstva nekog posla određuje prioritet dotičnog zadatka u sustavu. Težina se najčešće koristi u određivanju troškova pri ocjenjivanju rasporeda, gdje ona predstavlja neku stvarnu mjeru kvalitete rasporeda (npr. novčani trošak posla po danu kašnjenja). U stvarnim uvjetima, gdje se ocjena rasporeda obavlja na temelju više kriterija, poslu se može pridružiti i više od jedne težinske vrijednosti (više u odjeljku 2.2.4).

2.2 Zapis okruženja raspoređivanja

2.2.1 Elementi zapisa okruženja raspoređovanja

Za potrebe učinkovite podjele problema raspoređivanja, kao i nedvosmislene komunikacije, krajem sedamdesetih godina uveden je $\alpha|\beta|\gamma$ zapis okoline raspoređivanja [Leu 04, Mar 04]. Taj zapis nije konačan već se kontinuirano proširuje kako se javlja potreba za opisom složenijih okruženja, no sva eventualna proširenja definiraju se kao nadopuna postojećih elemenata. Tri polja u zapisu odgovaraju trima glavnim elementima

koji moraju biti definirani u postupku raspoređivanja, a to su važeća svojstva strojeva (procesora), svojstva zadataka i kriterij vrednovanja rasporeda.

2.2.2 Okolina strojeva

Prvo polje u zapisu, α, određuje okolinu strojeva na kojima se izvode poslovi. Osim vrste stroja, u ovom se polju još mogu naznačiti i broj strojeva (kao i njegova eventualna promjenjivost), građa komunikacijske mreže i mogućnost istovremene komunikacije i obrade. U ovom će se pregledu prikazati samo najčešće korištene vrste strojeva. U zagradi iza vrste okruženja prikazana je oznaka te vrste u α polju zapisa.

*Jedan stroj (*1*)*

Postoji samo jedan stroj u sustavu na kojemu se izvode svi poslovi. Ovaj primjer je specijalni slučaj svih ostalih složenijih okolina.

Paralelni identični strojevi (Pm*)*

Postoji m identičnih strojeva koji rade paralelno. Ako m nije naveden, broj strojeva je proizvoljan (također i za ostale vrste okruženja). Svaki posao sastoji se od samo jedne operacije (zadatak) i može se izvesti na bilo kojem stroju. Brzina obrade pojedinog zadatka jednaka je na svim strojevima, pa u trajanju izvođenja nije potrebno navoditi oznaku stroja.

Jednoliki strojevi (Qm*)*

U okruženju jednolikih strojeva (engl. *uniform machines*) svaki stroj ima različitu brzinu, no neovisnu o vrsti zadatka koji se na njemu obavlja. Brzina stroja *i* se označava sa s_i. Ako svaki posao ima definirano trajanje izvođenja p_j, tada će se posao *j* na stroju *i* obaviti u trajanju od $p_{ij} = p_j / s_i$.

Nesrodni strojevi (Rm*)*

U okruženju nesrodnih strojeva (engl. *unrelated machines*), svaki stroj obrađuje svaki pojedini posao proizvoljno definiranom brzinom, odnosno za svaku moguću kombinaciju zadatka i stroja definirano je posebno trajanje izvođenja p_{ij}.

Proizvoljna obrada (Jm*)*

Okruženje proizvoljne obrade (engl. *job shop*) obuhvaća skup poslova *J* od kojih se svaki posao J_j sastoji od više nedjeljivih operacija. Broj operacija pojedinoga posla označava se sa n_j, a pojedine operacije sa $T_{1j}, T_{2j}, \ldots, T_{n_j j}$. Operacije se moraju izvoditi određenim redoslijedom i na određenim strojevima, pri čemu je moguće da se operacije jednoga posla odvijaju više puta na jednom stroju ili nijednom na nekom drugom stroju. Trajanje obrade svake od operacija ovisno je o operaciji i o stroju na kojemu se izvodi.

Obrada tijeka (Fm*)*

Obrada tijeka (engl. *flow shop*) razlikuje se od prethodne okoline u tome da svaki posao ima jednak broj operacija i taj broj je jednak broju strojeva. Svaka operacija mora se obaviti točno jedanput na svakom stroju i to po redoslijedu koji je jednak za sve poslove (tj. operacija T_{1j} nekog posla obavlja se na stroju P_1, operacija T_{2j} obavlja se na stroju P_2 itd., u navedenom redoslijedu).

Otvorena obrada (Om*)*

Otvorena obrada (engl. *open shop*) predstavlja najveće pojednostavljenje prethodna dva okruženja, a od obrade tijeka razlikuje se samo po tome da je redoslijed obrade pojedinih operacija proizvoljan (važno je samo da se svaka operacija obavi na za nju

predviđenom stroju). Ova i prethodne dvije vrste okruženja nazivaju se još i okolinama predodređenih procesora.

2.2.3 Okolina zadataka

Drugo polje u zapisu označava se sa β i određuje svojstva zadataka u problemu raspoređivanja. Svojstva o kojima je ovdje riječ obuhvaćaju moguće postojanje dodatnih uvjeta ili relaksacija na raspored, kao što su prekidivost zadataka, ograničenja redoslijeda, dodatna sredstva potrebna za izvođenje, postojanje vremena pripravnosti, proizvoljna trajanja izvođenja, vremena nužnog završetka, komunikacijska kašnjenja itd. Neka od tih svojstava opisana su u nastavku.

Prekidivost zadataka

Jedna od najčešće korištenih relaksacija u postupcima izrade rasporeda je dopuštanje prekidivosti zadataka (engl. *preemption*), tj. mogućnosti da se neki zadatak obavi samo djelomično te nakon proizvoljne količine vremena nastavi. Ukoliko su zadaci bezuvjetno prekidivi, u polje se zapisuje oznaka *pmtn*. Određeni problem raspoređivanja često se pojednostavljuje dopuštanjem prekidanja čime se može dobiti dobra donja granica ocjene kvalitete rasporeda, a ponekad i uvid u rješenje neprekidive (složenije) inačice problema.

U slučaju vrednovanja rješenja sa prekidivim zadacima moguće je više načina interpretacije vrijednosti troškova. Ukoliko je prekidivi zadatak izveden kao skup vremenski međusobno odvojenih podzadataka, postavlja se pitanje kako ocijeniti trošak s obzirom na sve podzadatke. Najčešće korištene metode su:

- trošak na kraju (engl. *cost at end*) – promatra se samo izvođenje posljednjeg podzadatka, odnosno njegovi troškovi (npr. kašnjenje);
- raspodijeljeni trošak (engl. *spread cost*) – u ocjeni troškova promatraju se svi podzadaci (npr. uzet ćemo u obzir sve podzadatke koji se izvode nakon vremena završetka izvornog zadatka, a ne samo posljednji podzadatak);
- usko raspodijeljeni trošak (engl. *tight spread cost*) – svaki podzadatak dobiva svoje vrijeme završetka na osnovu kojega se ocjenjuje trošak; ukupni trošak je zbroj troškova svih podzadataka.

Ukoliko nije drugačije navedeno, u ocjeni rasporeda podrazumijeva se trošak na kraju (samo za posljednji podzadatak).

Ograničenja u redoslijedu

Ograničenja u redoslijedu (engl. *precedence constraints*) zahtijevaju da se neki zadaci završe prije nego može započeti izvođenje nekih drugih zadataka. Najopćenitiji oblik međuovisnosti zadataka može se predstaviti usmjerenim acikličkim grafom u kojemu su čvorovi zadaci a veze predstavljaju njihove međuovisnosti. U takvom slučaju, oznaka u polju zapisa je *prec*. Ukoliko svaki zadatak ima najviše jednog prethodnika i najviše jednog sljedbenika, zadaci su povezani u obliku lanaca što se označava sa *chains*. Ukoliko svaki zadatak ima najviše jednog sljedbenika (npr. u slučaju gdje se iz više dijelova proizvodi jedan konačni proizvod), ovisnost se označava sa *intree*. Ako pak svaki zadatak ima najviše jednog prethodnika (što odgovara situaciji gdje se iz jednog materijala izrađuje više proizvoda), oznaka je *outtree*. Postojanje ograničenja u redoslijedu može znatno usložniti zadatak raspoređivanja.

Vremena pripravnosti

U sustavima raspoređivanja česte su dvije inačice problema s obzirom na postojanje vremena pripravnosti zadataka. Najjednostavnija je ona inačica u kojoj se za sve zadatke smatra da su raspoloživi od zadanog početnog trenutka, odnosno $r_j = 0$ za sve

zadatke. Ovaj problem se još naziva i *statičkim*, a u polju β se ne upisuje oznaka. Za razliku od navedenog, ukoliko je za svaki posao definirano vrijeme pripravnosti koje je u općenitom slučaju različito on nule, tada je problem *dinamički*, a u polje se upisuje r_j. Statička inačica problema uvijek predstavlja olakšanje uvjeta raspoređivanja.

Trajanja postavljanja

U uvjetima gdje se na jednom stroju izvodi više vrsta zadataka (najčešće upravo u problemima sa jednim procesorom), ponekad je nakon završetka jednog zadatka potrebno stroj prilagoditi za obradu druge vrste zadatka. Ovaj dodatni zahtjev opisuje se postojanjem trajanja postavljanja među zadacima (engl. *sequence dependent setups*) [Cic 03, Lee 04]. Za svaki mogući par prethodnika i sljedbenika definira se potrebno trajanje; npr. ako nakon zadatka *i* dolazi zadatak *j* tada je stroj potrebno prilagoditi u trajanju od s_{ij}. U općenitom slučaju, trajanja postavljanja definiraju se elementima kvadratne matrice dimenzije jednake broju zadataka.

Trajanje izvođenja

U općenitom slučaju svaki zadatak ima proizvoljno zadano trajanje izvođenja i tada se u polje β ne upisuje dodatni simbol. Ponekad se kao olakšanje problema, a i ako je prikladno, pretpostavlja da svi zadaci imaju jednako trajanje izvođenja p, u kojem se slučaju u polje zapisuje ta oznaka. Također je trajanja izvođenja svih zadataka moguće ograditi donjom i gornjom graničnom vrijednošću.

Dozvoljeno čekanje

U proizvodnim pogonima u kojima poslovi dolaze u različitim vremenskim trenucima (dinamički problem, uz postojanje vremena pripravnosti), često dolazi do situacije da je sredstvo trenutno slobodno, na raspolaganju su neki poslovi čija bi obrada mogla započeti odmah, ali u 'bliskom' trenutku u budućnosti poznato je da u sustav dolazi posao relativno velike važnosti. Ukoliko su poslovi neprekidivi, potrebno je odlučiti da li započeti s obradom manje važnih poslova, i time odgoditi početak važnijeg posla koji tek treba doći, ili pričekati do dolaska važnijeg posla. Ovisno o mogućnosti čekanja u nekom sustavu, govorimo o dozvoljenom čekanju (engl. *inserted idleness*), ili o nedozvoljenom čekanju (engl. *no inserted idleness*) ako uvjeti ne dozvoljavaju da se sredstvo namjerno drži bez obrade [Mor 93]. Naravno, ukoliko niti jedan posao trenutno nije raspoloživ, nema drugog izbora osim čekanja. Ukoliko nije posebno navedeno, za postupak raspoređivanja podrazumijeva se da dozvoljava čekanje.

Čekanje među procesorima

U sustavima gdje se posao sastoji od više operacija, implicitno se pretpostavlja da djelomično završeni posao može čekati neodređenu količinu vremena prije nastavka obrade na sljedećem stroju. U stvarnosti, ova je pretpostavka ograničena kapacitetom spremnika među strojevima koji ima konačnu vrijednost. Ako među strojevima nije dozvoljeno čekanje (tj., spremnici imaju kapacitet jednak nuli), u polje se upisuje oznaka *no-wait*, dok oznaka *wait* označava spremnik ograničenog kapaciteta. Ukoliko je dozvoljeni kapacitet neograničen, u polje se ne upisuje oznaka.

2.2.4 Vrednovanje rasporeda

Vrednovanje rasporeda moguće je obaviti po više kriterija, koji su često vrlo složeni i međusobno proturječni [Jon 98]. Kriteriji vrednovanja rasporeda u velikoj mjeri utječu na izbor odgovarajućeg algoritma raspoređivanja. Ocjena rasporeda moguća je nakon izvođenja rasporeda i nakon prikupljanja izlaznih veličina u sustavu (svi dosada

opisani parametri predstavljaju zapravo ulazne veličine problema). Slijedeći konvenciju iz [Leu 04], izlazne veličine sustava opisane su velikim slovima.

- **Vrijeme završetka** C_j (engl. *completion time*) - trenutak u kojemu aktivnost *j* završava izvođenje
- **Protjecanje** F_j (engl. *flowtime*) – količina vremena koju je neka aktivnost provela u sustavu:

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

- **Kašnjenje** L_j (engl. *lateness*) – razlika (pozitivna ili negativna) između vremena završetka i vremena željenog završetka:

$$L_j = C_j - d_j \tag{2.2}$$

- **Zaostajanje** T_j (engl. *tardiness*) – pozitivni iznos kašnjenja neke aktivnosti; ako je kašnjenje negativno, zaostajanje je jednako nuli:

$$T_j = \max\{0, L_j\} \tag{2.3}$$

- **Preuranjenost** E_j (engl. *earliness*) – negativni iznos kašnjenja neke aktivnosti; ako je kašnjenje pozitivno, preuranjenost je jednaka nuli:

$$E_j = \max\{0, -L_j\} \tag{2.4}$$

- **Zakašnjelost** U_j – označava je li neka aktivnost prekoračila željeno vrijeme završetka:

$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases} \tag{2.5}$$

Valja napomenuti kako se u literaturi i za označavanje zadatka koristi simbol T_j, no u ovom će radu taj simbol označavati isključivo zaostajanje neke aktivnosti.

Mjerila vrednovanja rasporeda

Na osnovu navedenih veličina definira se većina mjerila vrednovanja. Neki od najčešćih kriterija opisani su u nastavku.

- Ukupna duljina rasporeda $C_{\max}$ (engl. *makespan*) – ukupna duljina rasporeda je posljednje vrijeme završetka svih poslova u sustavu:

$$C_{\max} = \max\{C_j\} \tag{2.6}$$

- Najveće kašnjenje $L_{\max}$ (engl. *maximum lateness*) – najveće kašnjenje definirano je kao

$$L_{\max} = \max\{L_j\} \tag{2.7}$$

- Težinsko protjecanje F_w (engl. *weighted flowtime*) – definira se kao suma težinskog protjecanja svih poslova:

$$F_w = \sum_j w_j F_j \tag{2.8}$$

- Težinsko zaostajanje T_w (engl. *weighted tardiness*) – jednako je težinskoj sumi zaostajanja svih poslova:

$$T_w = \sum_j w_j T_j \tag{2.9}$$

- Težinski zbroj zaostalih poslova ili težinska zakašnjelost U_w (engl. *weighted number of tardy jobs*) – definira se kao težinska suma svih zaostalih poslova:

$$U_w = \sum_j w_j U_j \tag{2.10}$$

- Težinska preuranjenost i težinsko zaostajanje ET_w (engl. *weighted earliness and weighted tardiness*) – definira se kao zbroj težinske preuranjenosti i težinskog zaostajanja, uz posebne težinske faktore za obje vrijednosti:

$$ET_w = \sum_j \left(w_{Ej} E_j + w_{Tj} T_j \right) \tag{2.11}$$

Navedena su mjerila vrednovanja najviše upotrebljavana u literaturi i primjeni, iako ni približno ne pokrivaju sve korištene kriterije. Osim opisanih izraza moguće je koristiti i netežinske inačice, gdje se pretpostavlja da je težina svakog zadatka jednaka [Rus 97]. Ovaj uvjet u svim slučajevima predstavlja olakšanje postupka raspoređivanja no najčešće nije prikladan za opis stvarnih uvjeta raspoređivanja gdje postoji različit jedinični trošak za različite poslove.

Posljednji od navedenih kriterija primjer je *nepravilnog* (engl. *nonregular*) mjerila jer uključuje trošak u slučaju ranijeg završetka posla. *Pravilni* kriteriji su oni kod kojih raniji završetak aktivnosti nikada ne povećava funkciju troška. Opravdanost opisanog nepravilnog kriterija valjana je u situaciji kada naručitelj ne želi primiti dovršeni proizvod prije dogovorenog roka (željenog vremena završetka), a cijena skladištenja može biti veća od cijene održavanja posla u sustavu (engl. *just-in-time environment*). U slučajevima kriterija koji uključuju više vrijednosti, kao u potonjem, opravdano je pretpostaviti da je jedinična cijena jedne vrijednosti različita od druge, kao što je u ovom primjeru jedinični trošak zaostajanja (w_{Tj}) u općenitom slučaju različit od jediničnog troška preuranjenosti (w_{Ej}). Isto tako, jedinični trošak protjecanja može biti različit od navedenih, u kojem slučaju se dodatno označava sa w_{Fj}.

2.2.5 Manje česta okruženja raspoređivanja

Opisani modeli problema raspoređivanja pripadaju klasičnoj teoriji raspoređivanja, za koje područje i postoji najviše teoretskih i eksperimentalnih rezultata. U stvarnoj primjeni se, međutim, javljaju okruženja koja se samo približno mogu opisati navedenim okruženjima. Za takve je probleme potrebno definirati pripadne strukture podataka i opisati odgovarajuća mjerila vrednovanja. Vrlo dobar pregled različitih okolina može se naći u [Leu 04], dok su ovdje navedeni samo neki primjeri.

Jedna od vrlo osjetljivih sredina gdje se mogu naći raznovrsni problemi raspoređivanja je organizacija avionskog prijevoza i usluga na aerodromima. Specifični uvjeti, velika osjetljivost na odstupanja i složeni zahtjevi za dodatnim sredstvima čine ovu okolinu pogodnom za razvoj novih metoda raspoređivanja [Che 99, Han 04]. Problemi na koje se ovdje nailazi obuhvaćaju raspoređivanje letova, posada, aviona po aerodromima itd.

Sustav pružanja usluga uključuje raspoređivanje sredstava koja najčešće predstavljaju ljudsku radnu snagu. Usluge u ovom slučaju mogu biti obučavanje, održavanje postojeće opreme [Hel 02] koje se može protezati na duga vremenska razdoblja itd. Najčešći kriterij vrednovanja rasporeda u ovim slučajevima je minimiziranje potrebnog kapaciteta odnosno stalno održavanje potrebe za resursima ispod neke zadane granice, no dodatni specifični uvjeti znatno usložnjavaju primjenu klasičnih algoritama raspoređivanja.

U sustavima za rad u stvarnom vremenu važno je odrediti može li se neki posao obaviti prije vremena nužnog završetka. Osim potpunog obustavljanja posla koji se ne može obaviti u potpunosti, u nekim je sustavima moguće obaviti samo dio posla prije isteka vremena, uz pretpostavku da će i manje kvalitetan rezultat obavljenog dijela posla (npr. neki izračun) biti dovoljno dobar za nastavak rada. Takav pristup omogućuju modeli nepreciznog računanja (engl. *imprecise computation*), a za takve i slične sustave potrebno je razviti posebno prilagođene algoritme raspoređivanja [Bud 99, Bud 00, Leu 95].

2.3 Uvjeti raspoređivanja

2.3.1 Podjela uvjeta raspoređivanja

Uvjeti raspoređivanja odnose se na različite mogućnosti izrade rasporeda te na vremenski odnos između same izrade rasporeda i njegovog izvođenja u nekom stvarnom sustavu. Različiti uvjeti raspoređivanja mogu djelomično ili u potpunosti isključiti mogućnost uporabe nekih postupaka raspoređivanja, bilo zbog raspoloživosti podataka, bilo zbog vremenskih ograničenja u sustavu. Većina postupaka opisanih u teoriji raspoređivanja, na primjer, podrazumijeva raspoloživost i nepromjenjivost svih potrebnih podataka, što u stvarnim uvjetima ne mora uvijek biti ispunjeno. Isto tako, ovisno o prirodi sustava, raspoređivač nema uvijek na raspolaganju neograničenu količinu vremena za izradu rasporeda, već je vrijeme za donošenje odluke o sljedećem stanju sustava ograničeno nekim parametrom. Uvjeti raspoređivanja se stoga mogu podijeliti po nekoliko (uvjetno) neovisnih osnova koje su opisane u nastavku.

Raspoloživost parametara

Budući svi algoritmi raspoređivanja koriste neke od parametara sustava (npr. broj poslova, vrijeme dolaska, trajanje itd.) kao ulazne veličine, najvažniji čimbenik u odabiru algoritma raspoređivanja je raspoloživost parametara sustava. Po pitanju raspoloživosti parametara, sustavi raspoređivanja mogu se podijeliti na dvije skupine:

- *Predodređeno raspoređivanje* (engl. *offline scheduling*) – u ovom obliku raspoređivanja pretpostavlja se da su sve potrebne vrijednosti poznate prije izrade rasporeda i prije samog rada sustava. Na primjer, poznat je ukupni broj poslova i njihove značajke, kao i značajke sustava u kojemu se poslovi izvode. Isto tako, u svakom trenutku rada sustava poznate su sve ulazne veličine koje opisuju budućnost sustava (kao što su dolasci poslova).
- *Raspoređivanje na zahtjev* (engl. *online scheduling*) – za razliku od prethodnog modela, u raspoređivanju a zahtjev odluke se donose na temelju samo trenutno dostupnih podataka, bez znanja o eventualnoj budućnosti sustava. Primjerice, značajke poslova su poznate tek kada neki posao dođe u sustav, odnosno kada postane raspoloživ, a ne prije početka izrade rasporeda [Pru 04]. Isto tako, pretpostavljeno je da se prethodno poznate vrijednosti parametara mogu mijenjati tijekom rada sustava (npr. ispad strojeva i sl.). U takvome okruženju ograničeni smo na uporabu algoritama koji mogu relativno brzo reagirati na promjene u sustavu. S druge strane,

obično nije neophodno oblikovati raspored za sve preostale poslove, već samo definirati sljedeće stanje sustava.

Pouzdanost parametara

Neovisno o tome jesu li vrijednosti parametara poznate ili ne prije izrade rasporeda, okolina raspoređivanja može se razlikovati i po tome koliko su vrijednosti tih parametara pouzdane, odnosno sa kojom preciznošću su iste procijenjene. Po tome se kriteriju raspoređivanje može grubo podijeliti na dvije skupine:

- *Deterministično raspoređivanje* – u determinističkom okruženju smatra se da su vrijednosti parametara sustava (npr. trajanje obrade posla) određene sa dovoljno velikom preciznošću, bez obzira u kojem trenutku postaju poznate. Na primjer, očekivano odstupanje neke vrijednosti, ako uopće postoji, je nekoliko redova veličine manje od same te vrijednosti.
- *Stohastičko raspoređivanje* – unutar stohastičkog okruženja ne postoje pouzdane procjene vrijednosti parametara sustava, već se stvarne vrijednosti mogu očitati tek nakon završetka rada određenog dijela sustava (npr. točno trajanje obrade nekog posla bit će poznato tek nakon završetka njegovog izvođenja [Pin 04, Meg 05]). No vrijednosti parametara nisu također niti potpuno nepoznate, nego su definirane funkcijama koje odgovaraju određenim vjerojatnosnim raspodjelama (engl. *probability distribution*), na temelju čega je moguće planirati izradu rasporeda.

2.3.2 Načini izrade rasporeda

Bez obzira na okruženje i uvjete raspoređivanja, izrada rasporeda može se obavljati kao neprekinuta aktivnost ili u koracima. S obzirom na vrijeme stvaranja rasporeda, pristup izradi rasporeda može biti statički i dinamički, ili kao kombinacija oba načina [Mar 04].

Statički pristup izradi rasporeda podrazumijeva izradu cjelokupnog rasporeda prije izvođenja zadataka, odnosno prije početka rada sustava. Primjena ovog načina izrade rasporeda moguća je samo ukoliko su svi potrebni parametri za rad postupka raspoređivanja dostupni, što odgovara uvjetima predodređenog raspoređivanja. Statičko izrađivanje rasporeda omogućuje uporabu postupaka koji zahtijevaju veću količinu vremena i obično nisu primjenjivi u uvjetima rada u stvarnom vremenu.

Dinamički pristup raspoređivanju gradi raspored u nekoliko iteracija koje se obavljaju paralelno sa radom sustava kojega se raspoređuje. Postupci za dinamičku izradu rasporeda najčešće određuju samo sljedeće stanje sustava i pozivaju se svaki puta kada nastane neka promjena u sustavu. Postupci koji se koriste u dinamičkoj izradi rasporeda ne moraju ali i mogu koristiti podatke o budućnosti sustava. Drugim riječima, dinamički pristup može se koristiti i u slučaju predodređenog raspoređivanja, kada su očekivani parametri raspoloživi, i u slučaju raspoređivanja na zahtjev, kada su parametri nekog posla poznati tek nakon njegovog ulaska u sustav. Način uporabe, naravno, ovisi o zahtjevima primijenjenog algoritma. Dinamičko raspoređivanje ne treba poistovjećivati sa dinamičkim problemom raspoređivanja, tj. onim u kojemu postoje vremena pripravnosti različita od nule

Opravdano je postaviti pitanje postoji li potreba koristiti dinamički način raspoređivanja kada su svi parametri sustava poznati (predodređeno raspoređivanje), odnosno, zašto u tom slučaju odmah ne izraditi cijeli raspored, što bi bilo identično statičkom raspoređivanju. Dinamički način raspoređivanja ima smisla rabiti u uvjetima rada u stvarnom vremenu, kada na početku ne želimo potrošiti vrijeme na izradu cjelokupnog rasporeda već što je prije moguće pokrenuti sustav. Drugi razlog uporabe

dinamičkog načina je moguća *promjenjivost* parametara; možemo, primjerice, dopustiti da se vrijednosti kao što su ukupan broj poslova, njihova trajanja, težine ili broj strojeva promijene tijekom rada sustava. U tome slučaju koristimo metodu koja raspored radi dinamički (određuje sljedeće stanje) ali koja pri tome također može koristiti nove vrijednosti parametara koji opisuju budućnost sustava. Ovim načinom izrade rasporeda omogućujemo brzo reagiranje na promjene u sustavu, kao što su ispadi strojeva, neplanirani dolasci poslova i sl.

Metodologija izvođenja pravila raspoređivanja opisana u ovome radu dozvoljava konstruiranje algoritama pogodnih za sve opisane uvjete. Iako će postupak izvođenja biti ukratko opisan za sve inačice, zbog velikog područja koje ti postupci obuhvaćaju neće se razmatrati algoritmi namijenjeni za stohastičko raspoređivanje. S druge strane, pokazat će se uporaba algoritama i za predodređeno i za raspoređivanje na zahtjev, budući su oba načina značajno zastupljena u današnjoj primjeni. Ovisno o uvjetima raspoređivanja, istovrsni izvedeni algoritmi uspoređeni su sa odgovarajućim postojećim algoritmima iz literature za rješavanje dotičnog problema.

2.4 Postupci raspoređivanja

2.4.1 Složenost algoritama raspoređivanja

Različiti uvjeti izrade rasporeda i mjerila vrednovanja zahtijevaju uporabu algoritma koji je u stanju na vrijeme proizvesti zadovoljavajuće rezultate. Odabir odgovarajućeg algoritma ovisit će o njegovim svojstvima: trajanju izvođenja, mogućnostima reagiranja na promjene parametara i kvaliteti dobivenog rješenja.

Problemi raspoređivanja proučavani početkom i sredinom prošlog stoljeća bili su relativno jednostavni u usporedbi s nekim današnjim primjerima. U to vrijeme bio je predložen velik broj učinkovitih algoritama koji su davali optimalna ili vrlo zadovoljavajuća rješenja. Kako je vrijeme prolazilo, problemi su postajali sve složeniji, a razvoj pogodnog algoritma bio je sve teži. Većina pokušaja rješavanja svodila se na neku vrstu algoritama ograničenog pretraživanja prostora stanja, što u biti predstavlja algoritme eksponencijalne složenosti. Usporedo s napretkom teorije složenosti algoritama, za mnoge se probleme pokazalo da su inherentno teški za rješavanje.

U stvarnim uvjetima rada postupak izrade rasporeda često je vremenski ograničen, pa uporabljeni algoritam mora imati neku ocjenu vremenske složenosti kako bi ga se moglo primijeniti u zadanoj okolini. Kao ocjena vremenske složenosti klasičnih algoritama raspoređivanja najčešće se koristi O notacija [Cor 01, Lop 01]. Računalni algoritmi često imaju polinomsku složenost, tj. vrijeme izvođenja algoritma može se izraziti nekim polinomom varijable koja predstavlja veličinu problema, što se smatra dobrim svojstvom. Nažalost, velik broj problema raspoređivanja je NP strog, što podrazumijeva da za taj problem ne postoji algoritam koji bi dao rješenje u polinomnom vremenu. Algoritmima koji mogu riješiti takve probleme trajanje izvođenja u najgorem slučaju raste eksponencijalno, što, osim za vrlo jednostavne 'školske' primjere, nije prihvatljivo u praktičnoj uporabi.

U cilju smanjenja složenosti okruženja raspoređivanja usvajaju se razna olakšanja, kao što su jedinična trajanja izvođenja, mogućnost prekidanja itd. Rješenje dobiveno na ovaj način može, u nekim slučajevima, poslužiti kao polazišna točka u pronalaženju rješenja izvornog problema. Drugi pristup je razvijanje deterministčkih algoritama koji neće u općenitom slučaju naći optimalno rješenje ali imaju polinomsku složenost. Takvi postupci nazivaju se približni (aproksimacijski) algoritmi. Treći pristup rješavanju su heuristički algoritmi, za koje ne postoji jamstvo dobivanja optimalnog rješenja, ali se zbog težine problema i neprikladnosti drugih pristupa često upotrebljavaju.

2.4.2 Heuristički postupci raspoređivanja

Heuristički postupci raspoređivanja pokušavaju pronaći rješenje tehnikama koje mogu ali i ne moraju biti svojstvene zadanom problemu (npr. isti heuristički postupak može se primijeniti u različitim domenama). Ovi postupci često uključuju neki element stohastičnosti pa rješenje koje daju ne mora uvijek biti jednako za jednake početne uvjete. Isto tako, ne postoji jamstvo za kvalitetu dobivenog rješenja, već se za većinu ovih algoritama provode simulacije koje daju procjenu ponašanja na stvarnim problemima. Heurističke postupke raspoređivanja možemo grubo podijeliti na dvije skupine: metode pretraživanja prostora stanja i algoritme koji rješenje problema grade izravno.

U prvu skupinu ubrajamo postupke koji nekom metodom definiraju moguće rješenje problema te ga iterativno ocjenjuju i modificiraju dok ne zadovolje zadani uvjet završetka. Zajednička osobina svih takvih postupaka je da se gotovo isključivo koriste u predodređenom statičkom raspoređivanju, odnosno u uvjetima kada su poznate sve promjene sustava u budućnosti i kada imamo relativno veliku količinu vremena na raspolaganju za izradu rasporeda (iznimka je npr. [Vaz 00] gdje se koristi dodatni algoritam za brzo reagiranje na nepredviđene događaje u sustavu). Neki od ovih postupaka su sljedeći (detaljniji pregled može se naći u [Mor 93, Jon 98]:

- matematički osnovani postupci: dinamičko programiranje, dekompozicijski postupci, ograničeno pretraživanje stabla, Lagrangeova relaksacija;
- postupci umjetne inteligencije: neuronske mreže, ekspertni sustavi itd.;
- evolucijski postupci: genetski algoritmi, simulirano kaljenje, tabu pretraživanje, itd.

S druge strane, niz heurističkih algoritama namijenjen je za rad u sasvim drugačijim uvjetima: njihov je cilj izrada rasporeda uz dinamički pristup ili rad u stvarnom vremenu, ne koristeći pritom dodatnu evaluaciju i popravljanje rješenja. Algoritmi iz ove skupine najčešće su oblikovani tako da reagiraju na događaje u sustavu i kao izlaz daju sljedeće stanje sustava (npr. pridruživanje zadatka procesoru). Za ovaj oblik algoritama između ostalog koriste se nazivi *pravila raspoređivanja* (engl. *dispatching rules, scheduling rules*) ili čak jednostavno *heuristike* (u užem smislu). Princip rada pravila raspoređivanja je uvođenje metrike kojom se ocjenjuju elementi sustava (aktivnosti i/ili sredstva) i odabir onih elemenata za koje se dobiva najbolja vrijednost metrike. Jednom definirano pravilo primjenjuje se uzastopno, najčešće paralelno sa radom sustava, dok se ne izgradi cjelokupni raspored, odnosno dok sustav ne dođe u stanje mirovanja. Iako se na većinu ovih postupaka može primijeniti izraz 'pravilo', složenost nekih od njih je takva da ih se uvjetno može uspoređivati sa klasičnim polinomskim približnim algoritmima, no ipak dovoljno jednostavna da omogućuje primjenu u dinamičkim uvjetima rada. Neka od pravila raspoređivanja su npr. SPT (engl. *shortest processing time*), EDD (*earliest due date*), LNS (engl. *largest number of successors*), i druga (više primjera za odgovarajuće okoline bit će prikazano u poglavlju 4).

Uz veliki broj postojećih pravila raspoređivanja i veliki broj različitih okruženja i kriterija vrednovanja, često se postavlja pitanje: koje pravilo koristiti? Želimo li pronaći učinkovito pravilo raspoređivanja za zadani sustav, potrebno je provesti iscrpne simulacije koje bi trebale pokazati najbolji izbor. Čak i tada nije se preporučljivo pouzdati u jedno pravilo za sve instance problema, već je često uputno definirati 'meta' pravilo koje će odlučivati o tome koje od pravila raspoređivanja primijeniti u određenim uvjetima (npr. "rabi SPT dok broj poslova na čekanju ne bude veći od 10, a tada prebaci na FIFO").

U manje uobičajenijim okruženjima raspoređivanja čest je slučaj da nijedno od raspoloživih pravila ne zadovoljava u dovoljnoj mjeri. Tada možemo pokušati promijeniti neko od postojećih pravila ili razviti vlastito pravilo prilagođeno dotičnom problemu. Zbog velikog broja različitih okolina i potrebe dobrog poznavanja rada sustava, kao i

učinkovitih metoda raspoređivanja, ovaj pristup nije uvijek isplativ. Problem odabira prikladnog pravila ostaje aktivno područje istraživanja, a u ovom se radu predlaže jedan od mogućih načina rješavanja toga problema.

3 Genetsko programiranje

Genetsko programiranje je tehnika koja omogućuje rješavanje nekoga problema računalom bez potrebe za stvaranjem programa koji rješava određeni problem. Način rada genetskog programiranja je uporaba evolucijskog procesa u stvaranju računalnih programa. Evolucija se u prirodi javlja poradi preživljavanja i reprodukcije sposobnih jedinki koje predaju svoj genetski kôd sljedećim generacijama. Kao posljedica toga procesa, genetski kôd koji čini sposobnije jedinke ima mogućnosti povećati svoj udio u cjelokupnoj populaciji, što uzrokuje promjene na razini vrste.

Genetsko programiranje je dio šire skupine algoritama zajedničkim imenom nazvanih evolucijsko računanje. Evolucijsko računanje (engl. *evolutionary computation*, EC) je skup postupaka koji oponašaju prirodni evolucijski proces na računalu. Ostvareni optimizacijski program koji se izvodi na računalu i oponaša evolucijski proces nazivamo evolucijskim algoritmom. Isto tako, računalni program koji izvodi postupak genetskog programiranja nazivamo genetskim programom. Postupci evolucijskog računanja podijeljeni na četiri glavne skupine: genetski algoritmi (engl. *genetic algorithms*, GA), genetsko programiranje (engl. *genetic programming*, GP), evolucijske strategije (engl. *evolution strategies*, ES) i evolucijsko programiranje (engl. *evolutionary programming*, EP).

Genetski algoritmi [Mic 92, Sri 94a] provode proces evolucije skupa jedinki koje predstavljaju moguća rješenja nekoga problema. Skup rješenja se po uzoru na evoluciju u prirodi naziva i populacija. Jedinka je obično predstavljena nizom simbola definiranog jezika, najčešće bitova (binarni prikaz rješenja). Poticaj ovakvom prikazu je struktura DNA koja tvori genetski kôd svih živih bića. Proces pretraživanja odvija se iterativnom primjenom genetskih operatora, kao što su operatori odabira, križanja i mutacije, na članove populacije. Način primjene operatora rukovodi se ocjenom kvalitete pojedinoga rješenja, koja se još naziva i dobrota (engl. *fitness value*), dok se ocjenjivanje kvalitete jedinke provodi primjenom funkcije dobrote (engl. *fitness function*).

U evolucijskoj strategiji [Sch 94] svaka jedinka označava skup svojstava pojedinoga rješenja, obično predstavljenoga nizom brojeva stalne duljine s pomičnim zarezom. Iz roditeljske populacije slučajno se odabire veći broj novih jedinki na koje se primjenjuje operator mutacije. Sljedeća generacija tvori se nekom metodom odabira među novim jedinkama i jedinkama iz roditeljskog skupa.

Evolucijsko programiranje [Mic 92] je u početku osmišljeno za evoluiranje konačnih automata te kasnije prošireno na probleme parametarskog optimiranja. Za razliku od ostalih modela, evolucijsko programiranje ne uključuje određeni način prikaza rješenja, a od genetskih operatora upotrebljava se samo mutacija. Operator mutacije koristi se kako bi se od početne slučajno generirane generacije dobio određeni broj novih jedinki, od kojih se potom operatorom odabira stvara nova generacija rješenja.

Posebnost genetskog programiranja, u odnosu na ostale opisane tehnike, je u činjenici da jedinke u populaciji genetskog programa predstavljaju računalne programe, odnosno strukture koje se mogu jednoznačno preslikati u oblik pogodan za izvođenje na računalu. Kako bi se olakšao proces stvaranja novih programa iz jednoga ili više roditeljskih, programi su u većini ostvarenja genetskog programiranja napisani u obliku stabla. Novi se programi dobivaju uklanjanjem grana iz jednoga stabla te dodavanjem tih grana na određeno mjesto u drugom stablu. Ovaj jednostavni proces kao rezultat također daje stablo i ujedno osigurava sintaktičku ispravnost dobivenih rješenja.

U ostatku ovoga poglavlja opisuje se genetsko programiranje kao jedna od metoda evolucijskog računanja. U sljedećem odjeljku navedena je struktura genetskog programiranja te osnovni elementi koje je potrebno definirati prije pokretanja procesa učenja. U narednim odjeljcima prikazane su postojeće inačice genetskog programiranja te aktivna područja istraživanja u cilju poboljšavanja učinkovitosti ove metode.

3.1 Struktura genetskog programiranja

3.1.1 Nastanak genetskog programiranja

Prilikom opisa načina rada genetskog programa, korisno je prepoznati sve elemente ovoga postupka koji izravno potječu od nekih starijih oblika evolucijskog računanja, a najviše od genetskih algoritama. Stvar je definicije možemo li genetsko programiranje shvatiti kao podvrstu genetskog algoritma ili obrnuto, no činjenica je da ova ideja svoju osnovu nalazi u genetskom algoritmu.

Ideja o spajanju genetskih algoritama i računalnih programa nije posve nova, no u prvotnim primjenama rješenja su se najčešće prikazivala u obliku pravila pogodnih za uporabu u ekspertnim sustavima [Lan 98, str. 19]. Jednim od pionirskih radova na ovom području smatra se rad N. Cramera [Cra 85]; u radu se definira vlastiti univerzalni jezik koji zbog svoje strukture može biti prikazan kao rješenje genetskog algoritma. Začetnikom ovoga postupka ipak se smatra J. Koza koji je u prikazu rješenja primijenio programski jezik Lisp i definirao mnoge od tehnika koje se koriste i danas [Koz 90, Koz 90a]. U svom poznatom radu [Koz 92], Koza ustanovljava genetsko programiranje kao paradigmu strojnog učenja i pokazuje uspješnost ovoga pristupa na cijelom nizu primjera. U tom se radu također iznosi tvrdnja o genetskom programiranju kao najopćenitijem obliku evolucijskog računanja poradi svoje promjenjive strukture rješenja. Nakon pojave navedenoga rada, broj članaka sa ovom tematikom u stalnom je porastu [Lan 05]. Godine 1998. pojavljuje se i knjiga [Ban 98] koja se može smatrati udžbenikom genetskog programiranja.

Genetsko programiranje omogućava računalu da do određenog stupnja oponaša ljudski način izrade programa, u obliku postupnih poboljšanja. Promjene se ostvaruju ponavljajućim kombiniranjem dijelova postojećih rješenja i to na način koji osigurava sintaktičku ispravnost novih rješenja. Postupna poboljšanja se postižu provjeravanjem svake promjene i zadržavanjem, uz određenu vjerojatnost, onih promjena koje doprinose povećanju kvalitete. U osnovi, na sličan način djeluje i izgradnja programa od strane čovjeka, s tom razlikom što genetsko programiranje nema znanje o tome gdje i kakvu promjenu treba učiniti, pa se rad računala temelji na metodi pokušaja i pogrešaka. Budući je u postupku uvijek očuvana sintaktička ispravnost programa i mogućnost njegova izvođenja na računalu u neizmijenjenom obliku, ova metodologija suštinski se razlikuje od jednostavnog mijenjanja redoslijeda naredbi strojnog ili nekog višeg programskog jezika.

3.1.2 Elementi genetskog programiranja

Kao i kod primjene bilo koje paradigme strojnog učenja, pri rješavanju problema genetskim programiranjem prethodno je potrebno definirati određene elemente sustava. Slijedeći metodologiju izloženu u [Koz 92], ti elementi su sljedeći:

- podatkovni i funkcijski elementi rješenja,
- funkcija dobrote,
- parametri genetskog programa,
- uvjet zaustavljanja postupka te
- građa rješenja.

U slučaju rješenja predstavljenog stablom, podatkovni elementi su u biti završni čvorovi stabla koje nazivamo podatkovnim čvorovima, dok funkcijske elemente nazivamo funkcijskim čvorovima. Funkcijski čvorovi su u većini slučajeva ujedno i unutarnji čvorovi stabla, no u nekim primjenama funkcijski element može imati i nula argumenta te djelovati samo preko popratnih učinaka (engl. *side effects*), pa takav funkcijski čvor mora biti završni čvor stabla. Odabir podatkovnih i funkcijskih elemenata te funkcije cilja uvelike određuje mogući prostor pretraživanja te stoga i mogućnost rješavanja problema. Parametri algoritma uključuju veličinu populacije, vjerojatnosti primjene operatora i slično, dok uvjet zaustavljanja postupka određuje trenutak završetka evolucijskog procesa. Određivanje građe rješenja podrazumijeva definiranje eventualnih struktura u obliku potprograma koje glavni program može pozivati, koje Koza naziva automatski definiranim funkcijama (engl. *automatically defined functions*, ADF). U sljedećim odjeljcima ukratko su opisani ovi elementi.

Odabir podatkovnih i funkcijskih elemenata

Najvažniji zahtjev koji treba zadovoljiti prilikom odabira podatkovnih i funkcijskih elemenata je taj da je uz pomoć njih moguće izraziti rješenje problema kojega pokušavamo riješiti. Ovo se svojstvo naziva potpunost (engl. *sufficiency*), a zadovoljavanje istoga ovisi o određenom problemu i nije ga u općenitom slučaju moguće definirati. U najvećem broju primjena u skup podatkovnih i funkcijskih elemenata stavljaju se svi elementi za koje se smatra da bi mogli biti od koristi u dotičnom problemu. Ono što uistinu želimo postići je definiranje takvoga skupa koji će ne samo biti potpun, nego i omogućiti genetskom programu pronalaženje zadovoljavajućeg rješenja. Zadovoljavanje svojstva potpunosti s jedne i olakšavanje postupka pretraživanja s druge strane oprečni su zahtjevi, pa ovo ostaje jedan od ključnih koraka pri definiranju rada genetskog programiranja. Više će riječi o ovom problemu biti u 5. poglavlju.

Drugo svojstvo koje skup građevnih elemenata rješenja treba zadovoljiti je zatvorenost (engl. *closure*). Zatvorenost zadanoga skupa elemenata definira se, za bilo koji funkcijski element, kao sposobnost prihvaćanja rezultata od bilo kojeg drugoga funkcijskog ili podatkovnog elementa. Zadovoljavanje ovoga svojstva ovisi o vrsti podataka svojstvenoj za određeni problem; npr. u rješavanju nekog problema u domeni Booleove algebre, funkcijski čvorovi će kao argumente obično primati i vraćati isključivo logičke vrijednosti 0 i 1. U nekoj drugoj primjeni, korišteni podaci mogu biti brojevi sa pomičnim zarezom. U slučajevima miješanja različitih vrsta podataka, potrebno je za svaku moguću kombinaciju definirati pretvorbu ili način interpretacije ulaznih vrijednosti. U suprotnom dobivamo program čije izvođenje nije moguće. Čak i u slučaju jedinstvene vrste podataka, zatvorenost može biti narušena ponašanjem nekih funkcija: rezultat dijeljenja, na primjer, nije definiran ukoliko je djelitelj jednak nuli. Za ovaj često korišteni slučaj najčešća je tehnika mijenjanje definicije određene funkcije, pa se tako u gotovo svim ostvarenjima umjesto običnog operatora dijeljenja koristi takozvani zaštićeni operator dijeljenja, koji za slučaj djelitelja jednakog nuli kao rezultat vraća vrijednost 1. Drugi pristup je definiranje nevaljanog tipa podatka, no tada svaka funkcija u skupu mora biti u mogućnosti prihvatiti takav tip podatka. U svim pokusima u ovom radu uporabljen je prvi pristup zaštite operatora, što je u skladu sa gotovo svim primjerima uporabe genetskog programiranja.

Funkcija dobrote

Funkcija dobrote je element sustava koji u najvećoj mjeri rukovodi evolucijom i upravlja kretanjem cjelokupne populacije. Od velikog je značaja da funkcija dobrote nagrađuje ne samo bolja rješenja, već i sva poboljšanja pronađena tijekom evolucijskog procesa. U većini ostvarenja genetskog programiranja određivanje dobrote rješenja

oduzima najveći dio utrošenog procesorskog vremena. Ukoliko izvedeni programi uključuju iterativne ili rekurzivne strukture, potrebno je uvesti i dodatne nadzorne mehanizme koji će otkriti postojanje nepravilnosti u programu (npr. beskonačne petlje).

Određivanje dobrote nekoga rješenja obično se sastoji od ispitivanja ponašanja toga rješenja na nizu ispitnih primjera, kao kod provjeravanja bilo kojeg računalnog programa. U slučaju programa ljudske izrade, u općenitom slučaju nije moguće ustanoviti postojanje grešaka u algoritmu bez obzira na broj ispitnih primjera, a to pogotovo vrijedi i za programe stvorene genetskim programiranjem. Važno je uočiti da je proces učenja vođen isključivo funkcijom dobrote, te da sustav može vrlo lako iskoristiti bilo kakve nedostatke u skupu ispitnih primjera (budući je nemoguće predvidjeti sve mogućnosti izvođenja, dobiveni programi mogu se vrlo dobro prilagoditi samo ispitnim scenarijima uporabe).

Parametri genetskog programa

Postavljanje procesa genetskog programiranja nužno uključuje odabir vrijednosti mnogih parametara. Odabir 'odgovarajućih' vrijednosti ovisan je o dotičnoj primjeni i ne postoji skup vrijednosti parametara koji su najbolji za svaki problem. Postavljanje parametara još više usložnjava njihova međusobna ovisnost, pa se u većini primjera koriste one vrijednosti koje ispitivač smatra najprikladnijima za konkretni problem. Veliko olakšanje u ovom bi slučaju bilo postojanje metoda automatske prilagodbe vrijednosti parametara koje oslobađaju korisnika potrebe za definiranjem (barem nekih) njihovih vrijednosti.

Jedan od najvažnijih parametara u genetskom programiranju je veličina populacije. Općenito su vrijednosti korištene u genetskim programima razmjerno veće od istovrsnih vrijednosti u drugim metodama evolucijskog računanja. Za većinu netrivijalnih problema ne preporuča se veličina populacije manja od 1000 jedinki, a u nekim ostvarenjima korištene su populacije i sa više od 100000 jedinki.

Uvjet zaustavljanja

Najčešći oblik uvjeta zaustavljanja je dostizanje predefiniranog broja generacija evolucijskog procesa. U svom radu [Koz 92] J. Koza za sve pokuse koristi broj od 50 generacija kao uvjet zaustavljanja postupka, ukoliko već prije nije pronađeno točno rješenje. Argument iza ove vrijednosti je zapažanje, iz većeg broja problema, da nakon toga broja generacija genetski program gubi sposobnost pronalaženja bitno različitih rješenja, te da su eventualna naknadna poboljšanja vrlo mala. Autor također tvrdi da je u većini slučajeva isplativije poduzeti više pokusa nego povećati broj generacija u jednom pokusu.

Uvjet zaustavljanja svojstven je dotičnom problemu, pa se u ovom radu rabi dvojaka provjera: algoritam se zaustavlja ili nakon obavljenog zadanog broja generacija (npr. 300) ili nakon što u zadanom broju uzastopnih generacija (npr. 50) nije postignuto poboljšanje najbolje vrijednosti dobrote, što bi označavalo kraj učinkovite pretrage.

Građa rješenja

Pretpostavimo li uobičajeni prikaz rješenja genetskog programiranja u obliku stabla, građa rješenja može podrazumijevati definiranje potprograma odnosno, po prihvaćenom nazivu, automatski definiranih funkcija (ADF). Automatski definirana funkcija je potprogram (funkcija, procedura, rutina) koji može biti pozvan od strane glavnog programa, a svi eventualni potprogrami, kao i glavni program, dinamički se izgrađuju tijekom evolucijskog procesa. U radu [Koz 94] daje se niz primjera u kojima genetski program sa automatskim funkcijama brže pronalazi rješenje ili uspijeva riješiti teže probleme nego obični genetski program.

Osim uporabe automatskih funkcija, građa rješenja može uključivati i višestablenu strukturu, u kojoj je svako pojedino stablo zaduženo za različiti dio problema. Ovakav je pristup uporabljen i u jednom dijelu pokusa u ovome radu (poglavlje 4.7).

3.1.3 Primjer uporabe genetskog programiranja

Opisani elementi genetskog programiranja mogu se ilustrirati na jednostavnom primjeru u kojemu je zadatak pronaći simbolički opis nepoznate funkcije, što se još naziva i postupkom simboličke regresije (engl. *symbolic regression*). Genetsko programiranje često se koristi za pronalaženje programa koji predstavljaju neku funkciju, tj. nemaju izravnih utjecaja na promjenu stanja sustava. Drugim riječima, u skupu funkcijskih elemenata nema objekata koji, osim računanja neke vrijednosti, uzrokuju promjenu u okolini, kao što je pomak nekog objekta ili pohranjivanje vrijednosti na spremničku lokaciju.

Za potrebe primjera pretpostavimo da želimo otkriti simbolički izraz predstavljen funkcijom $y = x^2$. Genetskom programu su na raspolaganju samo podaci u obliku vrijednosti ulazne varijable i vrijednosti funkcije za danu ulaznu vrijednost. Na osnovu tih informacija, oblikovanih u skup ispitnih primjera, zadatak je algoritma pronaći izraz koji najbolje opisuje ponašanje zadanog skupa podataka. Po uzoru na prethodni odjeljak, definiramo sve potrebne elemente genetskog programa.

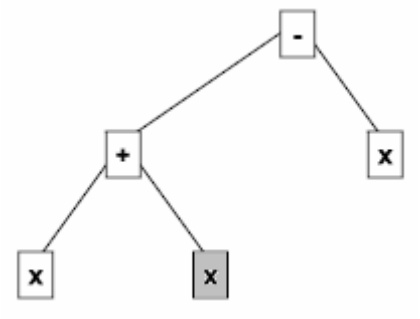
- *Odabir funkcijskih i podatkovnih elemenata*: budući ne znamo prirodu ciljne funkcije, u skup funkcija možemo uvrstiti osnovne računske operacije, s tim da je operator dijeljenja zaštićen na način opisan u prethodnom odjeljku (dijeljenje s nulom kao rezultat daje 1). Od podatkovnih elemenata uvrstit ćemo samo jedan: ulaznu varijablu x. Odabir ovoga podatkovnog čvora zasniva se na našoj pretpostavci da se ciljna funkcija *može* opisati kao polinomska funkcija varijable x, tj. pretpostavljamo da odabrani skup elemenata ima svojstvo potpunosti. Ukoliko je ova pretpostavka kriva, genetski program neće biti u mogućnosti pronaći točno rješenje, no pokušat će pronaći rješenje koje će se najbolje prilagoditi skupu ispitnih primjera. Definirani skup podataka i funkcija zadovoljava svojstvo zatvorenosti jer rukuje sa jednakim tipom podatka, realnim vrijednostima.
- Funkcija dobrote ocjenjuje se uspoređivanjem zadane vrijednosti ciljne funkcije sa vrijednošću dobivenom od promatrane jedinke. Definiramo 11 ispitnih primjera u kojima varijabla x poprima vrijednosti u intervalu $[-1,1]$, na primjer niz vrijednosti $\{-1,-0.8,-0.6,...,0.8,1\}$. Funkcija dobrote može se definirati kao zbroj apsolutnih odstupanja prave vrijednosti i vrijednosti dobivene promatranim rješenjem za svih 11 ispitnih primjera na sljedeći način:

$$f = \sum_{i=1}^{11} \left| y_i - GP(x_i) \right|, \tag{3.1}$$

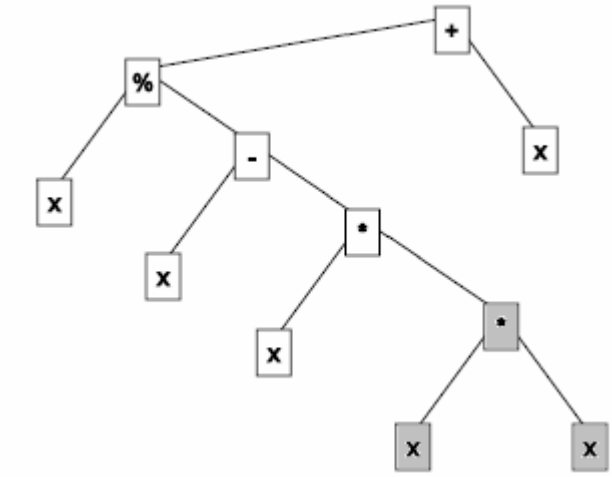
  gdje je y_i pravo rješenje za zadani x_i, a $GP(x_i)$ označava vrijednost koju daje jedinka genetskog programa. Ovako definirana funkcija dobrote podrazumijeva minimiziranje, tj. potragu za minimalnom vrijednošću. Prilikom ocjenjivanja možemo provjeravati razliku vrijednosti za svaki pojedini ispitni primjer te, ukoliko je razlika manja od 0.001 (na primjer), to možemo nazvati pogotkom, smatrajući da je rezultat dovoljno precizan.

- Parametri koje moramo definirati su veličina populacije te način dobivanja nove generacije iz prethodne. Za potrebe ovoga primjera bile bi dostatne gotovo bilo koje vrijednosti, pa se na ovom mjestu neće opisivati posebni oblici odabira i drugih genetskih operatora.
- Uvjet zaustavljanja genetskog programa može se definirati ograničenim brojem generacija (npr. 50) ili nakon 11 pogodaka otkrivenih prilikom ocjenjivanja neke jedinke.
- Građa rješenja je odabrana kao jedno stablo kojim se pokušava predstaviti ciljna funkcija.

U ovom vrlo jednostavnom primjeru genetsko programiranje gotovo je uvijek u stanju pronaći točno rješenje. Jedan primjer dobivanja točnog rješenja može se prikazati primjenom operatora križanja nad dvije roditeljske jedinke. Jedna roditeljska jedinka prikazana je na slici 3.1, a druga na slici 3.2, uz napomenu da je zaštićeni operator dijeljenja prikazan znakom '%'.

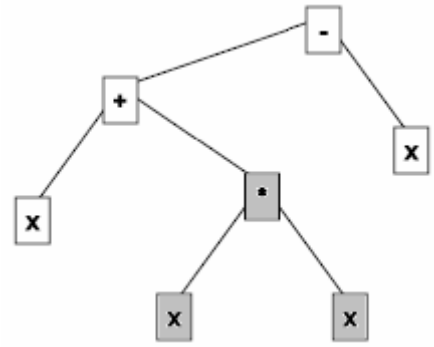


Slika 3.1 Prva roditeljska jedinka, izraz $(2x - x)$



Slika 3.2 Druga roditeljska jedinka, izraz $\left(\frac{x}{x - x^3} + x\right)$

Križanje se obavlja tako da se u obje jedinke slučajno odabere neko podstablo (prikazano osjenčano) te se podstablo jedne jedinke zamijeni sa odabranim podstablom druge jedinke. Za ovaj primjer podstablo druge jedinke zamijenjeno je sa odabranim podstablom prve jedinke, a rezultat križanja prikazan je na slici 3.3.



Slika 3.3 Rezultat križanja, izraz $x + x^2 - x$

Dobivena jedinka predstavlja funkciju $y = x + x^2 - x$ koja je ekvivalentna traženoj ciljnoj funkciji, čime je pronađeno točno rješenje.

Točnost rješenja genetskog programiranja

Potrebno je pojasniti pojam 'točnog' rješenja u genetskom programiranju. Analitički točno rješenje ovoga problema je izraz x^2 kojega je, uz pojednostavljenje, algoritam i pronašao. U općenitom slučaju nismo u mogućnosti ustanoviti analitičko podudaranje rješenja genetskog programa sa pravim rješenjem, jednostavno zato što nemamo egzaktni prikaz rješenja problema. Stoga se u određivanju točnosti oslanjamo na odstupanje koje pronađeno rješenje pokazuje od nekog zamišljenog pravog rješenja. U većini primjena postavljamo najveće zadovoljavajuće granice odstupanja, tako da podudaranje pronađenog rješenja nije savršeno ali je prihvatljivo uz zadanu preciznost (ograničenu npr. preciznošću prikaza broja s pomičnim zarezom u računalu). Pretpostavimo da je u skup podatkovnih elemenata uvrštena i proizvoljna numerička konstanta te pretpostavimo da je genetski program pronašao rješenje predstavljeno sljedećim izrazom:

$$x^2 + 0.0000000001 \cdot x^4 . \qquad (3.2)$$

Rješenje opisano gornjim izrazom nije točno sa gledišta analitičkog prikaza. Međutim, prikazano rješenje je točno za sve primjene u kojima se pridržavamo uobičajenih granica preciznosti. Uporaba ovakvog rješenja u analitičkim izračunima je nezamisliva, no takvo će rješenje savršeno poslužiti za dobivanje rezultata (u promatranom području) u svakodnevnoj inženjerskoj praksi.

Rješenja prikazana u ovom radu, kao i većina rješenja u svim primjenama genetskog programiranja, gotovo su uvijek 'kriva' u analitičkom smislu. No u primjeni, u usporedbi sa drugim postojećim rješenjima u okruženju gdje 'pravo' rješenje ne postoji, ovakav će rezultat uvijek biti zadovoljavajuć. Prilikom prikaza rezultata treba stoga imati na umu da se u rješenju često nalaze elementi koji imaju vrlo mali utjecaj na konačni rezultat, što uzrokuje relativno velika i neintuitivna rješenja. Usprkos nepreglednom prikazu, mogućnost uporabe dobivenih rješenja jednaka je kao i za postojeća, analitički 'čista' rješenja.

3.2 Područja istraživanja na polju genetskog programiranja

3.2.1 Predstavljanje rješenja

Osnovni model genetskog programiranja, u kojemu su rješenja prikazana u obliku stabla a temelj sustava je programski jezik Lisp, doživio je u međuvremenu mnoge preinake. Neke od inačica načinjene su prilagodbom za dotično područje rješavanja (npr. prikaz usmjerenim grafom), dok su druge vođene idejom o poboljšanju učinkovitosti (npr. automatski definirane funkcije). U većini primjena genetskog programiranja za prikaz rješenja koristi se stablo. Oblik stabla pogodan je za opis programa u funkcijskim jezicima, no u zadnje vrijeme koriste se i proceduralni jezici kao C. Najčešće, ipak, autori definiraju vlastiti jezik kojega na neki način, izvan GP sustava, jednoznačno preslikavaju u izvorni tekst programa.

Pokusi su provedeni i sa linearnim prikazom programa, gdje su u uporabi najčešće instrukcije strojnog jezika [Nor 94]. Najveća prednost prikaza rješenja strojnim jezikom su vrlo velika ubrzanja (reda veličine 10^3) koja se mogu postići zbog mogućnosti izravnog izvođenja programa bez potrebe za interpretiranjem. Drugačiji pristup uporabljen je u paralelnom raspodijeljenom genetskom programiranju (PDGP, [Pol 99]) u kojemu su stabla zamijenjena usmjerenim grafovima. Veze u grafovima povezuju čvorove samo po predefiniranom uzorku i služe prijenosu informacija među čvorovima. Na taj način rezultat nekoga čvora može se proslijediti na više mjesta u grafu, što ovome pristupu daje dodatne mogućnosti.

U sustavu PADO [Tel 95] koristi se vlastiti prikaz rješenja u sustavu za prepoznavanje signala i slikovnih informacija (engl. *classifier*). Rješenja su u ovom primjeru načinjena od grupa programa povezanih lukovima kao u usmjerenom grafu, struktura kojega određuje tijek kontrole i grananja prilikom evaluacije rješenja. Budući se u opisanom prikazu mogu oblikovati beskonačne petlje, svako rješenje ima ograničeno vrijeme izvođenja nakon kojega se uzima trenutno postignuti rezultat.

Jedna od inačica genetskog programiranja dopušta uporabu različitih vrsta podataka unutar istog rješenja (engl. *strongly typed genetic programming,* STGP) [Mon 95, Hay 96]. Prilikom primjene genetskih operatora (npr. križanja) poduzimaju se mjere kako bi svaka funkcija uvijek kao argumente imala samo čvorove sa odgovarajućom vrstom podataka. Ovakav pristup dodatno usložnjava operacije nad jedinkama, no iznesena je tvrdnja da se time značajno smanjuje mogući prostor pretraživanja, što omogućava uspješniji rad genetskog programa.

Po uzoru na automatski definirane funkcije, u nekim se radovima kao funkcijski elementi rješenja uvode operatori koji tijekom procesa evolucije mogu promijeniti strukturu samoga rješenja [Tal 03]. Još jedna mogućnost obogaćivanja prikaza su i funkcijski elementi koji mogu promijeniti način interpretacije rješenja ovisno o zadanim uvjetima [Gib 02]. Posebnost ovih pristupa je dinamičko mijenjanje građe odnosno načina ocjenjivanja rješenja bez primjene genetskih operatora.

3.2.2 Funkcija dobrote

Kao i kod genetskih algoritama, u genetskom se programiranju većina procesorskog vremena troši na određivanje dobrote rješenja. Jedna od tehnika ubrzavanja ispitivanja rješenja slučajni je odabir podskupa ispitnih primjera za svako novo rješenje (engl. *stochastic sampling*, [Nor 95]). U ovom je radu omogućena uporaba te tehnike, no kvaliteta rezultata dobivenih na ovaj način općenito se pokazala lošijom od rezultata dobivenih uz stalni broj ispitnih primjera. Razlozi toj pojavi mogu biti različita uspješnost nekoga rješenja za različit podskup primjera te neprikladnost usporedbe na taj način dobivenih vrijednosti dobrote.

Drugi način [Gat 97] mogućeg ubrzavanja evaluacije rješenja je promatranje ukupne vrijednosti greške jedinke za vrijeme ocjenjivanja po pojedinim ispitnim primjerima. Za dotičnu funkciju dobrote može se definirati gornja granica odstupanja nakon koje se ocjenjivanje jedinke prekida, a dobrota se zadržava na zatečenoj vrijednosti (engl. *limited error fitness*). U ovom je radu također ugrađen ovaj mehanizam, te je u nekim primjerima omogućio dodatno ubrzanje procesa evolucije.

Veličina rješenja kao dio dobrote

U većini problema poželjna su rješenja koja su ne samo kvalitetna nego i manje veličine, u svrhu lakše interpretacije. Odnos veličine i kvalitete rješenja u genetskom se programiranju obično naziva sažetost (engl. *parsimony*). Ukoliko se evolucijski proces na neki način ne nadzire, rješenja genetskog programiranja imaju tendenciju rasta. Postoji nekoliko pretpostavki o uzrocima te pojave (odjeljak 3.2.6), no u većini slučajeva ona je nepoželjna. Osim što se znatno usporava ocjenjivanje rješenja, usporava se i proces evolucije budući promjene u većim rješenjima dolaze do slabijeg izražaja. Isto tako, prekomjerna veličina je u nekim područjima pokazatelj prenaučenosti (engl. *overfitting*) i nedovoljno općenitoga rješenja.

Najuobičajenije metode ograničavanja veličine rješenja koje nisu vezane uz funkciju dobrote su postavljanje granice na najveći broj elemenata rješenja (čvorova u stablu) ili najveće dopuštene dubine stabla. Dodatna je mogućnost definiranje posebnih

operatora koji će stvarati uglavnom sažetija rješenja (na primjer smanjujuća mutacija, odjeljak 3.2.5).

Ograničavanje veličine moguće je i uključivanjem dodatnog elementa u funkciju dobrote koji će kažnjavati 'veća' rješenja. U radu [Zha 96] predlaže se postupak u kojemu se funkcija dobrote tvori uz pomoć izvorne funkcije i kazne temeljene na složenosti rješenja (broja elemenata) te dobrote i složenosti najboljeg rješenja u trenutnoj generaciji (engl. *adaptive parsimony pressure*). Autori pokazuju poboljšanje učinkovitosti i dobivanje kraćih rješenja za veliki broj problema, no mogu se naći i primjeri [Koz 92, str. 613] u kojima ocjenjivanje sažetosti otežava pronalaženje točnog (ili zadovoljavajućeg) rješenja.

3.2.3 Parametri i oblik genetskog programa

Na polju određivanja učinkovitog skupa parametara za genetsko programiranje nije provedeno puno istraživanja. Vrijednosti određenih parametara u većini se slučajeva preuzimaju iz velikog broja prethodno izvedenih pokusa. Određivanje prikladnih vrijednosti parametara za određeni problem uglavnom se obavlja po nahođenju i iskustvima istraživača. U ovom se radu stoga izlažu neke tehnike za olakšavanje odabira vrijednosti pojedinih parametara ili ukidanjem potrebe za istima (poglavlje 5.2).

Pored izbora vrijednosti parametara, na raspolaganju nam je i izbor vrste evolucijskog procesa, odnosno načina na koji će se obavljati izmjena generacija. Ovaj se postupak naziva odabirom u širem smislu, što treba razlikovati od *operatora* odabira, koji definira izbor jedne jedinke iz zadanog podskupa jedinki i kojega možemo nazvati odabirom u užem smislu. U prvim se radovima na području genetskog programiranja gotovo isključivo koristi generacijski odabir, u kojemu se nova populacija stvara uz pomoć odabranih jedinki iz postojeće. Nakon procesa odabira na novoj populaciji provode se genetski operatori čiji rezultati zamjenjuju neke od postojećih jedinki.

U posljednjih nekoliko godina, međutim, sve je veća uporaba eliminacijskih metoda odabira (engl. *steady state selection*), u kojima se pojedine jedinke u populaciji uklanjaju i zamjenjuju novima stvorenim uz pomoć genetskih operatora. Na taj se način, korak po korak, odvija cijeli proces evolucije do postizanja uvjeta zaustavljanja. Najpopularnijim operatorom odabira se pokazao turnirski odabir, u kojemu se na slučajan način iz populacije bira manji broj jedinki, a najbolja od njih biva izabrana za primjenu genetskih operatora. Druga inačica operatora među odabranim jedinkama određuje najlošiju, koja potom biva uklonjena iz populacije. Prednost turnirskog operatora odabira je činjenica da ne zahtijeva globalnu informaciju o populaciji i da ne ovisi o iznosu vrijednosti dobrote jedinki već samo o njihovim relativnim odnosima. Zbog svoje lokalne prirode turnirski odabir je pogodan za različite oblike paralelizacije evolucijskog procesa [Gol 98, Gol 00, Gol 01, Gol 01a].

3.2.4 Korištenje spremničkog prostora

Većina računalnih programa prilikom izvođenja u velikoj mjeri upotrebljava spremnički prostor, no gotovo u svim primjenama genetskog programiranja program je isključivo funkcija ulaznih vrijednosti. Sve potrebe za pohranom vrijednosti, ako postoje, u tom slučaju rješava sustav unutar kojega se rješenja ocjenjuju. Ukoliko se spremnik ipak koristi, najčešće su u uporabi dva oblika: skalarne varijable i pobrojani spremnik (engl. *indexed memory*).

Uporaba skalarnih varijabli sastoji se u uvođenju jedne ili više varijabli koje se kao podatkovni čvorovi mogu pojavljivati unutar rješenja, te određenog broja funkcijskih čvorova zaduženih za mijenjanje ili pohranu vrijednosti varijabli. Budući se u većini primjena značenja pojedinih varijabli sa gledišta rješenja genetskog programiranja ne

razlikuju, način na koji će dobiveni program koristiti varijable često nije predvidiv. Iznimka su primjene u kojima se pojedinim varijablama pristupa uz pomoć različitih funkcijskih elemenata.

Drugi najčešći oblik korištenja spremnika je uz pomoć pobrojanog niza vrijednosti. U tom slučaju u program su ugrađeni elementi za rukovanje spremnikom, ali izvedeni je program odgovoran za određivanje rednog broja vrijednosti kojoj želi pristupiti. Budući je količina podataka uvijek ograničena, potrebno je definirati mehanizme kojima se obuhvaćaju slučajevi dohvata nepostojećeg elementa spremnika. Najčešći pristup je dodavanje operatora cjelobrojnog ostatka koji kao rezultat uvijek daje postojeću spremničku lokaciju.

U radu [Lan 98] posebno se obrađuje uporaba složenijih podatkovnih struktura u genetskom programiranju. Autor izlaže primjere u kojima genetski program izvodi funkcije za rukovanje strukturama kao što su stog, red ili lista, te primjere u kojima se korištenjem navedenih struktura rješavaju neki problemi koje nije moguće riješiti uporabom jednostavnog spremničkog prostora

3.2.5 Operatori genetskog programiranja

Osnovni mehanizam istraživanja prostora rješenja je primjena genetskih operatora, od kojih se u genetskom programiranju javljaju i mutacija i križanje. U vjerojatno najutjecajnijem radu na području genetskog programiranja [Koz 92], Koza ne upotrebljava mutaciju uz argument da je za uspješan postupak pretraživanja dovoljan samo operator križanja. Kao posljedica, mnoga ostvarenja genetskog programiranja ne uključuju operator mutacije; no u posljednjih nekoliko godina mutacija se ipak sve češće koristi u primjeni, pa u kasnijim radovima i Koza preporuča postavljanje malog udjela mutacije u primjeni genetskih operatora. Do danas je definirano puno oblika operatora mutacije, od kojih su neki navedeni u nastavku:

- standardna mutacija (engl. *subtree mutation*) – slučajno se odabire čvor u stablu i na njegovom mjestu stvara se podstablo sa slučajnim čvorovima;
- zamjena čvora (engl. *node replacement*) – slučajno odabrani čvor stabla zamjenjuje se slučajno odabranim čvorom iz skupa mogućih čvorova, uz ograničenje da novi čvor ima jednak broj podčvorova;
- mijenjajuća mutacija (engl. *swap mutation*) – slučajno se odabire čvor u stablu i zamjenjuje se sa čvorom koji ima isti broj podčvorova (list se zamjenjuje listom), a funkcijski i podatkovni čvorovi biraju se jednakom vjerojatnošću;
- smanjujuća mutacija (engl. *shrink mutation*) – slučajno se odabire čvor sa podčvorovima i cijelo se podstablo zamjenjuje samo sa jednim od svojih podčvorova;
- mutacija konstanti (engl. *mutating constants*) – ukoliko stablo uključuje slučajne numeričke konstante, odabire se jedna od njih i zamjenjuje novom vrijednošću.

Standardni operator križanja već je opisan u primjeru u odjeljku 3.1.3, no i na ovom su području predložene mnoge izmjene. Neki su autori predložili uporabu križanja koje u najmanjoj mogućoj mjeri mijenja kontekst koji neko podstablo ima u staroj i novoj jedinki [Ban 98, str. 166]. Motivacija iza ovoga postupka je približavanje takozvanom homolognom križanju (engl. *homologous crossover*), koje je prevladavajući oblik križanja u prirodi i za koje se smatra da bi pomoglo učinkovitijem pretraživanju prostora rješenja. Očuvanje konteksta postiže se, u jednoj inačici križanja, primjenom operatora samo na onim mjestima jedne jedinke za koja postoji odgovarajuća identična pozicija u stablu druge jedinke. Za ovaj oblik križanja, koji se može nazvati križanjem sa zajedničkom

točkom (engl. *one-point crossover*), izveden je i prilagođeni teorem sheme [Lan 02] kojim se predviđa globalno kretanje populacije u prostoru rješenja.

Drugi pokušaji poboljšavanja križanja uključuju npr. metodu stvaranja više potomaka (više od 2) te zadržavanja dva potomka koji imaju najveću vrijednost dobrote [Tac 94], uvođenje posebnih čvorova između svaka dva postojeća čvora u stablu kao oblik zaštite [Ban 98] itd.

3.2.6 Konvergencija i zalihost rješenja genetskog programiranja

Kao što je navedeno u odjeljku 3.2.2, veličina rješenja genetskog programiranja obično raste tijekom evolucijskog procesa, što se u ovom kontekstu naziva i napuhavanjem (engl. *bloat*). Uzroci ove pojave nisu do kraja razjašnjeni, no preduvjet napuhavanju je rukovanje rješenjima promjenjive veličine i oblika, što odgovara opisu rada genetskog programa. Kao posljedica ove pojave, velika većina materijala jedinke sačinjena je od dijelova koji ne utječu na ponašanje rješenja u ispitnom okruženju (ne mijenjaju dobrotu rješenja). Neke od teorija o nastanku napuhavanja opisane su u nastavku.

Najvjerojatnije najstarije objašnjenje o rastu rješenja povezano je sa pojmom umetaka (engl. *introns*), koji su definirani kao dio genotipa jedinke (prikaza rješenja) koji ne utječe na sposobnost preživljavanja neke jedinke [Ban 98, str. 186]. Naziv potječe od biološkog pojma dijela DNA lanca koji nema vidljivu ulogu u tvorbi RNA unutar stanice; iako je upitna opravdanost ove paralele, spomenuti naziv uvriježen je u genetskom programiranju. Povećanje rješenja tada se javlja zbog pojave da se bolja rješenja, koja mogu biti opisana i mnogo manjom količinom podataka, na neki način zaštićuju od mogućih negativnih posljedica operatora – ako se operator primijeni unutar umetka, u većini slučajeva ne mijenja se funkcionalnost, a time i dobrota jedinke. Budući 'dobro zaštićene' jedinke imaju veću vjerojatnost preživljavanja, tu sposobnost nasljeđuju i njihovi potomci pa je porast veličine rješenja stalan [Tac 94].

Drugi mogući uzrok nastanku napuhavanja [Sou 98] obrazlaže se nakupljanjem nekorisnih dijelova rješenja bliže vrhovima stabla, pa ako križanje ukloni samo manji dio rješenja dalje od korijenskog čvora, veća je vjerojatnost očuvanja bitnog dijela rješenja. S druge strane, podstablo umetnuto križanjem imat će u prosjeku veću veličinu od uklonjenog podstabla, pa će se opet javljati povećanje broja čvorova.

Konačno, povećanje rješenja može biti objašnjeno činjenicom da, u prikazu rješenja promjenjive veličine, za jednaku vrijednost dobrote postoji puno veći broj mogućih duljih nego kraćih programa [Lan 97, Lan 00]. Nakon što genetsko programiranje pronađe najbolje rješenje određene duljine, tada će se, ukoliko se ne može naći neko bolje rješenje, javljati sve više rješenja iste kvalitete ali puno veće duljine, što je očekivani nastavak u pretraživanju ovakvog prostora rješenja.

Općenito je prihvaćen zaključak da je za nastanak nekontroliranog rasta veličine rješenja nužno međudjelovanje genetskih operatora i postupka odabira na temelju funkcije dobrote. U cilju dokaza te tvrdnje provedeni su pokusi [Ban 02] u kojima se jedan od elemenata odstranjuje (npr. slučajni odabir umjesto odabira temeljenog na dobroti) te opaža i nestanak napuhavanja. Napuhavanje se također često povezuje sa konvergencijom genetskog programa, u smislu da se ova pojava javlja kada je već pronađeno najbolje moguće rješenje, a postupak pretraživanja tada prelazi na istraživanje drugih dijelova prostora rješenja koji uključuju veću količinu podataka.

U primjeni su predloženi mnogi postupci sprečavanja nastanka napuhavanja ili kažnjavanja rješenja velike duljine. Neki od postupaka u uporabi navedeni su u nastavku, osim pristupa u kojemu se veličina rješenja uvažava u funkciji dobrote, koji je opisan u odjeljku 3.2.2.

Ograničenje veličine

Gotovo u svim ostvarenjima genetskog programiranja rabi se ograničenje veličine rješenja, bilo u obliku najveće dubine stabla ili u obliku najvećeg broja čvorova (odnosno općenitih elemenata rješenja). Ovo je ograničenje najčešće predefinirano i konstantno tijekom evolucijskog procesa. Prednost ovih metoda je jednostavnost, no u većini primjena pokazuje se da genetski program vrlo brzo dosegne zadana ograničenja, ma koliko velikodušno postavljena. Hoće li zadana ograničenja pomoći ili odmoći pronalasku rješenja ovisno je o samim ograničenjima i dotičnom problemu. Tehnika promjenjivog ograničenja veličine predložena je u [Sil 03] gdje se definira dinamička i statička najveća dubina stabla. Statička dubina je konstantna dok se dinamička dubina u početku procesa postavlja na neku manju vrijednost, a povećava se u slučaju pronalaska rješenja sa trenutno najboljom dobrotom i dubinom većom od dinamičke granice. Na taj se način dinamičko ograničenje postupno povećava, najviše do predefiniranog statičkog ograničenja. Predložena metoda ne utječe znatno na učinkovitost, no ispitivanja su provedena samo na dva relativno jednostavna problema.

Prilagođeni genetski operatori

U literaturi je predloženo nekoliko inačica genetskih operatora kojima je cilj (ili popratni učinak) usmjeriti pretraživanje na područje sa manjom veličinom rješenja. Jedan od operatora mutacije kojemu je namjena smanjivanje rješenja sastoji se od slučajnog odabira podstabla neke jedinke te tvorbe nove jedinke samo od odabranog podstabla, čime se uvijek dobiva manja jedinka ali sa različitim početnim čvorom. Prilikom uporabe mutacije, također se može promatrati vjerojatnost mutacije jednoga *čvora*, umjesto vjerojatnosti mutacije cijele jedinke. Na taj način veće su jedinke podložnije utjecaju mutacije. Križanje uz zajedničku točku, na primjer, automatski smanjuje napuhavanje budući ne može proizvesti rješenje veće dubine stabla od najveće dubine stabla u oba roditelja.

Otkrivanje nekorisnih dijelova rješenja

Postupci za otkrivanje (i uklanjanje) nekorisnog dijela rješenja općenito su složeniji i rjeđe se koriste u primjeni. Ovi postupci nastoje prepoznati nekorisne dijelove tijekom evolucijskog procesa [Bli 94, Bli 96]; budući se koriste usporedo s pretraživanjem, ostvarenje ovih metoda ne smije biti računalno zahtjevno. U većini primjera način otkrivanja nekorisnih dijelova temelji se na skupu pravila koji je nepromjenjiv i svojstven samo određenom problemu [Mon 01]. Iz toga je razloga primjena tih metoda ograničena prirodom problema i korištenog skupa funkcijskih i podatkovnih elemenata.

3.3 Primjene genetskog programiranja

Od svojih 'službenih' početaka ranih devedesetih godina prošlog stoljeća pa do danas, genetsko programiranje je našlo niz primjena. Izložena je i tvrdnja da je genetsko programiranje primjenjivo na raznovrsniji skup problema nego bilo koja druga metoda strojnog učenja [Koz 95]. U svakom slučaju, u stalnom je porastu broj primjera na kojima je genetsko programiranje pomoglo u pronalaženju rješenja koje se može mjeriti sa rješenjima načinjenim od strane stručnjaka.

Neka od područja primjene genetskog programiranja navedena su u nastavku. Iako se većina primjera bavi složenim problemima iz stvarnog života, uspješnost genetskog programiranja u određenom području ne treba poistovjetiti sa nepostojanjem drugih načina rješavanja dotičnog problema.

Predviđanje i raspoznavanje

Genetsko programiranje je posebno pogodno za primjenu u problemima gdje je potrebno osmisliti model koji najbolje odgovara zadanim podacima. Ovaj pristup pogotovo je koristan kada je u podatke umiješan i šum i kada je osnova kretanja podataka nelinearna po prirodi. Uspješni primjeri primjene na ovom području uključuju predviđanje oblika bjelančevina [Lan 98, str. 39], analizu nizova kaotičnih vrijednosti, analizu satelitskih radarskih slika u geološkom sustavu [Ban 98, str. 351] itd. U radu [And 94] pokazuje se kombinacija genetskog algoritma i genetskog programa koja izvodi program za prepoznavanje znakova (OCR). Već spomenuti sustav PADO [Tel 95] koristi vlastiti prikaz rješenja u raspoznavanju slikovnih i zvučnih informacija. Genetsko programiranje koristi se i u problemima sa područja medicine [Pen 00], prvenstveno uz pomoć dubinskog pretraživanja podataka (engl. *data mining*) u svrhe dijagnoze.

Upravljanje

Još je u prvoj od nekoliko knjiga o genetskom programiranju Koza [Koz 92] pokazao rješavanje problema upravljanja robotima u nepoznatoj okolini. Nešto novija je primjena genetskog programiranja u zadatku učenja hodanju robota sa šest nogu (heksapodalne strukture), te upravljanje procesom dvonožnog hoda [Ok 01].

Oblikovanje sustava

Iako je temeljno svojstvo rješenja genetskog programiranja mogućnost predstavljanja računalnih programa, inačice ove strukture mogu se iskoristiti u oblikovanju sustava koji na prvi pogled nemaju puno zajedničkoga sa klasičnim programima. U najnovijoj od knjiga u seriji o genetskom programiranju [Koz 03], Koza i drugi autori demonstriraju uporabu genetskog programiranja u sintezi analognih električnih krugova, u čemu su postignuti bolji rezultati od postojećih proizvoda te namjene u uporabi, na primjer, u elektroenergetskim sustavima. Ocjenjivanje dobrote rješenja provodi se simulacijom u programu SPICE; budući je ocjenjivanje rješenja vremenski vrlo zahtjevan proces, evolucija se izvodi na paralelnom sustavu od preko 1000 čvorova, uz populaciju od nekoliko stotina tisuća jedinki i tipično trajanje pokusa od preko mjesec dana (što odgovara otprilike 200 generacija).

Primjena genetskog programiranja nalazi se i u oblikovanju neuronskih mreža. Rješenje genetskog programa u ovom se slučaju sastoji od pravila koja upravljaju stvaranjem i mijenjanjem oblika neuronske mreže, uključujući dodavanje, uklanjanje ili mijenjanje težina itd [Ban 98, str. 355].

Potpora računalnim sustavima

Još jedna od zanimljivih ideja je primjena genetskog programiranja u sustavu za otkrivanje neovlaštenih upada u računalnom sustavu. Cijeli je sustav otkrivanja podijeljen na nekoliko agenata od kojih svaki promatra zasebni dio sustava. Svaki agent također izvodi vlastita pravila ponašanja i komunicira s drugim agentima. Ukoliko jedan od agenata otkrije nepravilnosti u radu sustava, javlja se određenom informacijom ostalim agentima koji po vlastitim pravilima reagiraju na poruku. Ukoliko se takva aktivnost u sustavu podigne iznad određene razine, može se poslati poruka ljudskom 'agentu', odnosno administratoru sustava [Ban 98, str. 358].

U radu [Wal 96] demonstrira se primjena genetskog programiranja u razvoju paralelnih algoritama na temelju danih slijednih rješenja. Cilj sustava je automatsko otkrivanje mogućih paralelnih inačica slijednih programa. Rješenje je predstavljeno tako da svaki podatkovni čvor rješenja predstavlja jedan od koraka slijednog programa, a ocjena se provodi na temelju točnosti rješenja te stupnja paralelizacije koju određeno rješenje postiže (veća paralelnost znači i brže izvođenje).

4 Raspoređivanje zasnovano na pravilima

Heurističke postupke raspoređivanja možemo uvjetno podijeliti na postupke pretraživanja prostora rješenja i algoritme koji grade raspored u jednom prolazu (ili ograničenom manjem broju prolaza). Jednostavniji heuristički algoritmi su najčešće predstavljeni u obliku koji računa prioritete elemenata sustava te potom element sa najvećim prioritetom uključuje u sustav. Ovakav oblik postupka, koji se naziva i pravilo raspoređivanja, zbog svoje je strukture pogodan za modifikaciju i relativno brzo ispitivanje učinkovitosti metodom pokušaja i pogrešaka. U ovom se poglavlju opisuje gradnja pravila raspoređivanja prilagođenih različitim okolinama i uvjetima raspoređivanja s pomoću paradigme genetskog programiranja, što je i središnji dio rada. U okviru ugradnje sustava za izvođenje pravila načinjena je podrška za razna okruženja raspoređivanja (jedan stroj, jednoliki strojevi, nesrodni strojevi, proizvoljna obrada), uvjete raspoređivanja (statičko i dinamičko, predodređeno i raspoređivanje na zahtjev), svojstva poslova (statički i dinamički dolasci, ograničenja u redoslijedu, slijedno ovisna trajanja postavljanja) te kriterije ocjene rasporeda. Provedeno je nekoliko stotina pokusa (od kojih pojedini traju i nekoliko tjedana na jednom računalu), a pravila dobivena genetskim programiranjem uspoređena su sa odgovarajućim pravilima u primjeni za određeno okruženje.

U ovom su poglavlju ukratko opisani načini primjene pravila raspoređivanja ovisno o danim uvjetima i okruženju. Navedena su svojstva ostvarenja genetskog programiranja zajednička za sve obrađene probleme, kao i način provođenja pokusa. Prikazani su postupci izvođenja pravila te usporedba dobivenih rješenja sa postojećim pravilima raspoređivanja za dotičnu okolinu.

4.1 Primjena pravila raspoređivanja

4.1.1 Raspoređivanje uz pomoć pravila

U uvjetima raspoređivanja u stvarnom vremenu ili općenito u okruženju u kojemu je potrebno brzo reagirati na promjene stanja sustava, najčešće se za raspoređivanje koristi algoritam koji relativno brzo može doći do potrebnog rješenja. Pri tome potrebno rješenje ne mora biti cjelokupni raspored od početka do (eventualnog) završetka rada sustava, već samo opis sljedećeg stanja sustava, po modelu dinamičkog raspoređivanja. Iako je i postupke koji rješenje pronalaze pretraživanjem moguće posebno prilagoditi za rad u takvim uvjetima (npr. izvođenjem algoritma do zadanog vremenskog trenutka i uzimanjem trenutno najboljeg pronađenog rješenja), takav pristup se ne koristi u većini primjena.

Rješavanje problema u opisanim uvjetima uglavnom se rješava primjenom pravila raspoređivanja. Pravilo raspoređivanja u užem smislu zapravo podrazumijeva samo *funkciju* koja, koristeći određene parametre sustava, definira metriku odnosno prioritet elemenata sustava. U složenijim inačicama funkcija može biti zamijenjena nekim algoritmom koji također mora načiniti odabir najboljeg elementa. Elementi sustava koji se međusobno 'natječu' najčešće su aktivnosti koje traže sredstva sustava, no moguća je i obrnuta situacija (sredstva se natječu za preuzimanje aktivnosti), kao i općeniti slučaj gdje se istovremeno može natjecati više aktivnosti i sredstava. Rješenje koje se dobiva primjenom pravila raspoređivanja definira *pridruživanje* koje je potrebno obaviti prilikom sljedeće promjene stanja sustava. Ukoliko sustav posjeduje samo jedno sredstvo, postupak pridruživanja svodi se na određivanje redoslijeda aktivnosti. Primjena pravila se prema tome razlikuje ovisno o okruženju raspoređivanja, no uvijek se koristi ista funkcija kojom

je pravilo definirano. Detaljniji opis načina uporabe pravila dan je u poglavljima koja opisuju rješavanje problema u dotičnom okruženju.

4.1.2 Vrste pravila raspoređivanja

Podjelu pravila raspoređivanja možemo definirati promatramo li samo funkciju koja određuje prioritete elemenata u sustavu. Podjela se može načiniti po više osnova [Mor 93], od kojih su najvažnije prikazane u nastavku. Pri tome je potrebno naglasiti da je neke od navedenih vrsta pravila moguće primijeniti samo uz određenu vrstu okruženja raspoređivanja.

S obzirom na promjenjivost prioriteta, pravila možemo podijeliti na *statička* i *dinamička*. Statička pravila računaju prioritete samo jednom (na početku raspoređivanja) i te se vrijednosti tijekom rada sustava ne mijenjaju. Drugim riječima, u računanju prioriteta koriste se samo nepromjenjive vrijednosti, kao što su težine poslova, željeno vrijeme završetka itd. S druge strane, dinamička pravila uključuju ponovno izračunavanje prioriteta svaki put kada je potrebno odrediti novo stanje sustava, a zbog korištenja vremenski ovisnih veličina. Dinamička pravila obično postižu bolje rezultate, mada to ne mora vrijediti u svakom slučaju.

Po pitanju odabira podataka, pravila mogu biti *lokalna* i *globalna* ili oboje istovremeno. U lokalnim pravilima koriste se podaci ovisni o lokalnom elementu sustava, kao što je jedno od sredstava (npr. trajanje operacije nekog posla na dotičnom sredstvu). Globalna pravila koriste podatke koji su globalno neovisni o pojedinom sredstvu, kao što su vremena dolaska i željenog završetka poslova. Naravno, pravila mogu kombinirati i lokalne i globalne podatkovne strukture.

Ovisno o dostupnosti podataka, raspoznajemo *deterministička* i *nedeterministička* pravila. U determinističkim pravilima javljaju se samo one varijable čija je vrijednost sa sigurnošću poznata ili prethodno određena. Nedeterministička pravila mogu koristiti i veličine čija će se točna vrijednost znati tek nakon završetka rada sustava. Najčešći primjer za to je preostalo vrijeme koje neki posao mora provesti u sustavu, gdje znamo koja količina vremena će se utrošiti na obradu preostalog dijela posla, no ne znamo koliko će vremena biti utrošeno na čekanje. U tome se slučaju raspoređivanje može odvijati na nekoliko načina:

- umjesto stvarne vrijednosti, koristi se *procjena* dotične veličine predstavljena nekim izrazom (npr. preostalo vrijeme u sustavu je vrijeme preostale obrade pomnoženo sa konstantom);
- procjena veličine određuje se na temelju posebne heuristike ili na temelju povijesti sustava, odnosno prethodnih slučajeva raspoređivanja;
- raspoređivanje se provodi *iterativno*: u prvoj iteraciji koristimo početnu procjenu i izradimo cjelokupni raspored, a potom na temelju dobivenog rasporeda odredimo novu vrijednost procjene koju iskoristimo u ponovnoj izradi rasporeda. Postupak se ponavlja dok nije zadovoljen neki uvjet zaustavljanja (najveći broj iteracija, broj iteracija bez poboljšanja, vremensko ograničenje). Već nakon malog broja iteracija mogu se dobiti vrlo velika poboljšanja rezultata, no moraju se uzeti u obzir i povećani vremenski troškovi izrade rasporeda.

4.1.3 Prednosti i nedostaci pravila raspoređivanja

Prednosti pravila raspoređivanja mogu se odmah prepoznati, a najvažnije su:

- *brzina izvođenja* – vremenski troškovi raspoređivanja uz većinu pravila raspoređivanja mogu se smatrati zanemarivima;

- *jednostavnost primjene* – princip raspoređivanja uz pomoć pravila vrlo je jednostavan i obično manje skup od ugrađivanja složenijih postupaka;
- *razumljivost rezultata* – način rada i rezultati pravila raspoređivanja lako su razumljivi i slični ljudskom načinu razmišljanja, stoga se dobivena rješenja brzo interpretiraju ili po želji mijenjaju;
- *brzo reagiranje na promjene* – budući se pravila primjenjuju u dinamičkim uvjetima rada, svaka eventualna promjena u sustavu može se odmah uzeti u obzir.

Uporaba pravila raspoređivanja povlači sa sobom i neke nedostatke; često je, naime, kvaliteta dobivenih rješenja lošija od kvalitete rješenja dobivenog nekim složenijim postupkom pretraživanja (iako to ne mora biti istina za sva okruženja), što je i najočigledniji nedostatak. Osim same jednostavnosti postupka, mogu se identificirati neki uzroci takve lošije učinkovitosti, pri čemu je važno napomenuti da ti uzroci nisu prisutni kod svih vrsta pravila i kod svih načina uporabe pravila.

- *Promatranje samo sljedećeg stanja sustava* – najuobičajeniji način primjene pravila uključuje određivanje samo prve sljedeće promjene stanja sustava. Na ovaj se način često mogu zanemariti neka buduća stanja koja bi mogla utjecati na donošenje trenutne odluke. Djelomično ublažavanje ovoga učinka može se postići uporabom varijabli koje opisuju stanje sustava u budućnosti (ako je to poznato, tj. samo za predodređeno raspoređivanje) ili iterativnom primjenom nedeterminističkih pravila.
- *Neprecizna interpretacija pravila* – budući da pravilo u užem smislu podrazumijeva samo funkciju metrike, za različita okruženja nije precizno definiran način primjene pravila (npr. koji elementi sustava se pridružuju i u kojem trenutku), što posebno dolazi do izražaja kod okruženja koja nisu obuhvaćena klasičnom teorijom raspoređivanja. Uz pokuse provedene u ovom radu stoga je za svako okruženje detaljno opisana primjena pojedinog pravila.
- *Uporaba neodgovarajućeg pravila* – ovisno o okruženju i mjerilima vrednovanja rasporeda, potrebno je odabrati pravilo koje će postići najbolje rezultate. Iako je za neke kombinacije okruženja i kriterija poznato koja bi pravila trebalo uporabiti, u većini primjena to nije jednostavan odabir [Cha 96]. Iz toga razloga razvijaju se posebne heuristike koje kombiniraju više pravila raspoređivanja za jedan problem [Wal 05, Auy 03].

Upravo se opisani problem odabira pravila nastoji zaobići metodologijom pronalaženja posebnog pravila za dano okruženje i mjerila vrednovanja.

4.2 Parametri genetskog programiranja

4.2.1 Struktura genetskog programa

Proces izvođenja pravila raspoređivanja uz pomoć genetskog programiranja bit će detaljno opisan u narednim poglavljima, no za sve inačice problema koriste se neka zajednička svojstva sustava genetskog programiranja. Primjerice, u svim pokusima koristi se eliminacijska inačica evolucijskog procesa, jednaki genetski operatori i (uglavnom) jednaki parametri koji određuju primjenu operatora. Potrebno je napomenuti da je mijenjanjem vrste evolucijskog procesa vjerojatno moguće dobiti nešto bolja (ili nešto lošija) rješenja od onih prikazanih u ovom radu. Ispitivanje svih mogućih kombinacija operatora ili inačica evolucijskog procesa je samo po sebi zahtjevna i opširna tema koja se ovdje neće detaljno razmatrati. Jedna od tvrdnji koje ovaj rad pokušava pokazati je da se i sa 'pretpostavljenim' parametrima, bez puno pokušaja poboljšavanja, mogu dobiti

kvalitetna rješenja opisanog problema. Usprkos tome, neke predložene metode odabira parametara i pogodnih podatkovnih elemenata bit će predstavljene u 5. poglavlju. Njima je cilj poboljšavanje učinkovitosti rada genetskog programiranja neovisno o danom problemu. U svim opisanim pokusima vrijede dolje opisane postavke, ukoliko u konkretnom slučaju nije naznačeno drugačije.

Ugradnja sustava za izvođenje pravila ostvarena je uz pomoć javno dostupnih biblioteka za potporu evolucijskom računanju [Gag 02, Gag 03, Bea 05] te uz vlastite module za podršku raspoređivanju, ubrzavanju evaluacije jedinki, vrednovanje podatkovnih čvorova, višestablena rješenja itd.

Parametri evolucijskog procesa

Evolucijski proces se odvija po principu eliminacijskog odabira, tj. u svakoj iteraciji se zamjenjuje jedna slučajno odabrana jedinka iz populacije. Zamjena onolikog broja jedinki kolika je veličina populacije smatra se jednom generacijom postupka. Odgovarajuća jedinka nadomješta se novom jedinkom nastalom primjenom nekog od genetskih operatora, a za svaki pojedini operator definirana je vjerojatnost primjene. Roditeljske jedinke se prilikom križanja biraju turnirskim odabirom (pretpostavljena veličina turnira je 3), a veličina populacije je postavljena na 10000. Uvjet zaustavljanja evolucijskog procesa je dvojak: postavljena je granica najvećeg broja generacija (300) i najvećeg broja uzastopnih generacija bez poboljšanja najbolje zabilježene dobrote (najviše 50 generacija). Postupak se zaustavlja kada se ispuni bilo koji od navedenih uvjeta. Početni skup jedinki stvara se metodom *ramped half-and-half* [Bur 03], pri čemu je najveća dubina stabla u novostvorenoj jedinki ograničena na 5 razina. Najveća dubina stabla tijekom cijelog procesa ograničena je na 17-20 razina, ovisno o dotičnom pokusu.

Genetski operatori

U procesu evolucije koristi se binarni operator križanja te unarni operator mutacije. Operator križanja poziva se prilikom stvaranja nove jedinke nakon postupka odabira, tj. nakon eliminacije postojeće jedinke. Prilikom primjene operatora, slučajno se odabire čvor u oba roditeljska stabla i na tom mjestu zamjenjuju se dobivena podstabla između dvije jedinke. Odabir čvora čije će se podstablo zamijeniti sa podstablom iz druge jedinke obavlja se tako da veću vjerojatnost odabira ima funkcijski čvor, odnosno čvor koji nije list stabla. U implementaciji u ovom radu, vjerojatnost odabira funkcijskog čvora u postupku križanja je 90% (sukladno tome, vjerojatnost odabira lista stabla je 10%).

Pored operatora križanja, uporabljena su tri operatora mutacije:

- standardna mutacija – slučajno se odabire čvor u stablu i na njegovom mjestu se stvara podstablo sa slučajnim čvorovima (najveća dubina novog podstabla ograničena je na 5 nivoa);
- mijenjajuća mutacija – slučajno se odabire čvor u stablu i zamjenjuje se sa čvorom koji ima isti broj podčvorova (list se zamjenjuje listom), a funkcijski i podatkovni čvorovi biraju se jednakom vjerojatnošću;
- smanjujuća mutacija – slučajno se odabire čvor sa podčvorovima i cijelo se podstablo zamjenjuje samo sa jednim od svojih podčvorova.

Svi operatori mutacije primjenjuju se jednakom vjerojatnošću od 3%. Osim stvaranja nove jedinke uz pomoć križanja i/ili mutacije, u proces evolucije uvršten je i operator reprodukcije kojim se neka jedinka može neizmijenjena kopirati u trenutnoj populaciji i time zamijeniti eliminiranu jedinku. Vjerojatnost primjene toga operatora postavljena je na 5%. Opisani parametri genetskog programiranja prikazani su pregledno u tablici 4.1.

Tablica 4.1 Popis pretpostavljenih parametara genetskog programiranja

Parametar / operator	Vrijednost / pojašnjenje
veličina populacije	10000
način odabira	eliminacijski
operator odabira	turnirski odabir (veličina turnira 3)
uvjet zaustavljanja	najveći broj generacija (300) ili najveći broj generacija bez poboljšanja (50)
križanje	vjerojatnost primjene 85%, standardno križanje, 90% vjerojatnosti odabira funkcijskog čvora
mutacija	standardna, mijenjajuća i smanjujuća, 3% vjerojatnosti za svaku
reprodukcija	vjerojatnost 5%
inicijalizacija	*ramped half-and-half*, najveća dubina 5 nivoa

4.2.2 Definiranje funkcije cilja

Ocjena dobrote jedinki, a ujedno i učinkovitosti promatranih pravila raspoređivanja, obavlja se ovisno o okruženju i definiranom mjerilu vrednovanja rasporeda na većem broju ispitnih primjera (tvorba ispitnih primjera opisana je u sljedećem odjeljku). Prilikom odabira kriterija, za pojedina okruženja birani su samo netrivijalni kriteriji, odnosno oni za koje ne postoji optimalno rješenje. Najčešća mjerila vrednovanja već su definirana u poglavlju 2.2.4, no u situaciji kada je neko rješenje potrebno ocijeniti s obzirom na cijeli skup ispitnih primjera sa različitim svojstvima, funkciju cilja treba definirati tako da ocjena za sve primjere utječe podjednakom težinom na konačni rezultat. Stoga se po uzoru na [Moh 83, Dha 78] funkcija cilja za različite kriterije za pojedini primjer sa indeksom *i* iz ispitnog skupa definira na sljedeći način:

- za težinsko zaostajanje:

$$f_i = \frac{\sum_{j=1}^{n} w_j T_j}{n \cdot \overline{w} \cdot \overline{p}}, \tag{4.1}$$

- za težinski zbroj zakašnjelih poslova (težinska zakašnjelost):

$$f_i = \frac{\sum_{j=1}^{n} w_j U_j}{n \cdot \overline{w}}, \tag{4.2}$$

- za težinsko protjecanje:

$$f_i = \frac{\sum_{j=1}^{n} w_j F_j}{n \cdot \overline{w} \cdot \overline{p}}, \tag{4.3}$$

- za ukupnu duljinu rasporeda:

$$f_i = \frac{\max\{C_j\}}{n \cdot \overline{p}}, \tag{4.4}$$

gdje je n broj poslova u ispitnom primjeru, $\overline{w}$ je srednja vrijednost težina poslova, a $\overline{p}$ je srednje trajanje poslova. Vrijednost se za pojedini ispitni primjer dijeli sa brojem poslova u dotičnom primjeru kako bi se postigla podjednaka težina za primjere sa

različitim brojem poslova. U slučajevima gdje pojedini kriterij uključuje težine, dodatno se obavlja dijeljenje sa srednjom težinskom vrijednošću, a ako kriterij uključuje i neku količinu vremena ovisnu o trajanju poslova, u nazivnik se stavlja i srednje trajanje obrade. Ukupna dobrota neke jedinke dobiva se zbrajanjem svih funkcija cilja za pojedine ispitne primjere:

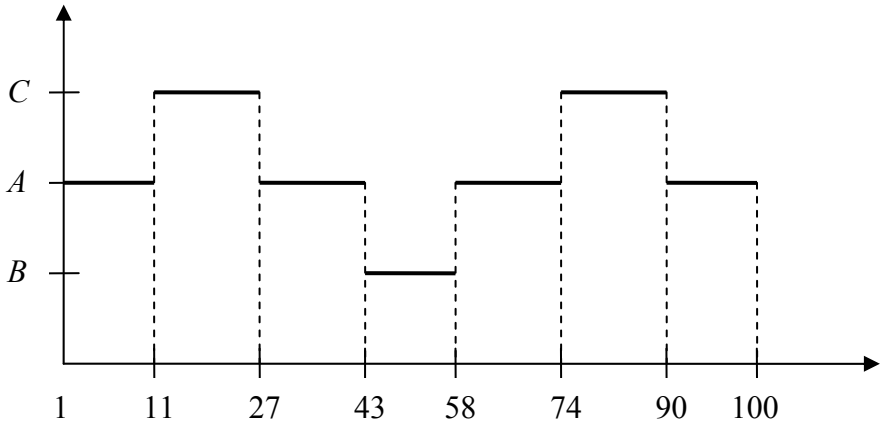
$$F = \sum_i f_i \,. \tag{4.5}$$

4.2.3 Oblikovanje ispitnih primjera

Za sve promatrane vrste okruženja, neovisno o mjerilu vrednovanja, definirani su ispitni primjeri uz pomoć kojih se ocjenjuje učinkovitost izvedenih pravila raspoređivanja. Ispitni primjeri su podijeljeni u dvije skupine: skup primjera koji se koriste u procesu učenja i skup primjera koji se koriste samo za ocjenu. Način stvaranja ispitnih primjera u skladu je sa pristupom koji se koristi u literaturi, a posebno je opisan za svako okruženje u odgovarajućem poglavlju. Gdje god je to moguće, skupini primjera za ocjenu dodani su postojeći raspoloživi primjeri iz literature (dobavljeni preko WWW stranica autora).

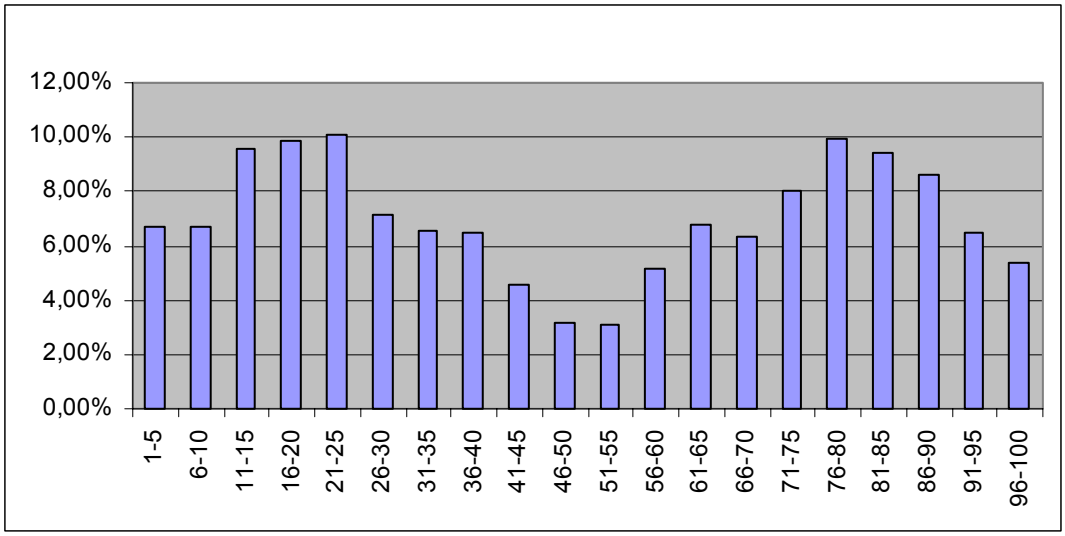
Raspodjele slučajnih varijabli

U postupku definiranja ispitnih primjera nužno je koristiti generatore pseudoslučajnih brojeva koji služe kao osnova za trajanje poslova, vremena dolaska, trajanja postavljanja itd. Pri tome je potrebno odabrati i vjerojatnosnu raspodjelu koju će generirane slučajne varijable slijediti. U skupovima za učenje koristi se jednolika raspodjela slučajnih varijabli dobivena generatorom ugrađenom u MS Visual .NET okolinu. Za potrebe skupa za ocjenu, kombinirane su tri raspodjele: jednolika, normalna (Gaussova) raspodjela i kvazi-bimodalna raspodjela (po uzoru na [Gre 01]). Prilikom sastavljanja skupa za ocjenu navedene raspodjele su korištene u sljedećim udjelima: 20% slučajnih vrijednosti slijedi jednoliku, 50% normalnu a 30% kvazi-bimodalnu raspodjelu. Vjerojatnosna funkcija za kvazi-bimodalnu raspodjelu definirana je kao na slici 4.1 uz vrijednosti konstanti $B = 0.004651$, $A = 2B$ i $C = 3B$ (vrijednosti slučajnih varijabli postavljene su u intervalu [1, 100]).



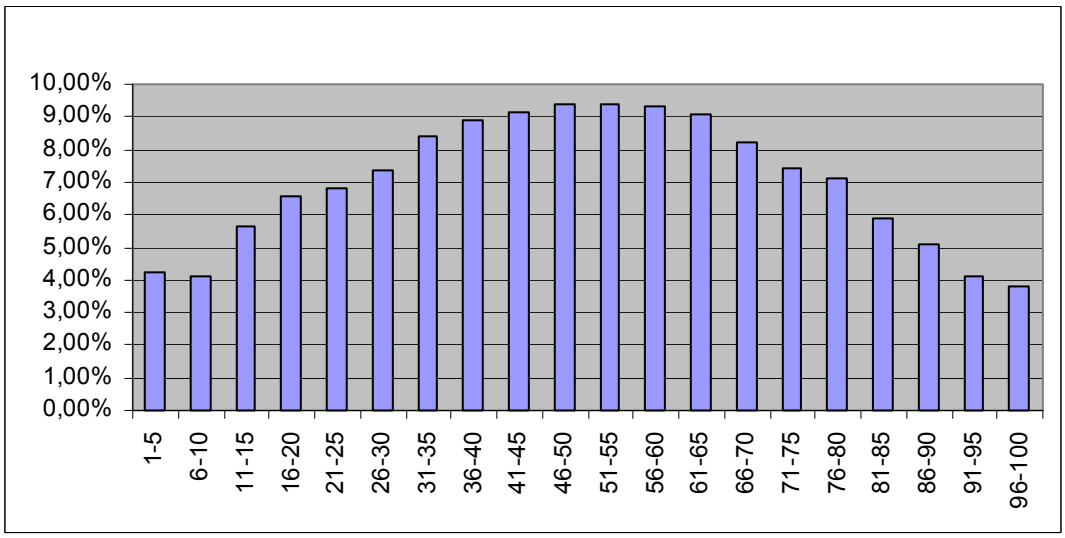
Slika 4.1 Vjerojatnosna funkcija kvazi-bimodalne raspodjele

Primjer skupa vrijednosti slučajnih varijabli dobivenih kvazi-bimodalnom raspodjelom prikazan je na slici 4.2.



Slika 4.2 Primjer skupa vrijednosti dobivenih bimodalnom raspodjelom

Kumulativna raspodjela dobivena kombinacijom prethodne tri prikazana je na slici 4.3.



Slika 4.3 Primjer skupa vrijednosti dobivenih kombinacijom jednolike, normalne i bimodalne raspodjele

Odabir najboljeg rješenja

U svakom izvedenom pokusu u pojedinom okruženju promatrana je konačna funkcija dobrote najboljeg rješenja na kraju rada genetskog programa. Prilikom odabira najboljeg rješenja za sve provedene pokuse, najbolja rješenja iz pojedinih pokusa ocijenjena su uz pomoć skupa ispitnih primjera za ocjenu, tj. na onom skupu koji im nije bio na raspolaganju prilikom učenja. Najbolje rješenje za sve pokuse je tada definirano kao ono rješenje koje je postiglo najbolji rezultat na skupu za ocjenu. Smisao ovakvog postupka je biranje onog rješenja (tj. pravila raspoređivanja) koje se najbolje ponaša na skupovima koje prije 'nije vidjelo', odnosno pronalazak onog rješenja koje najbolje poopćava znanje iz primjera za učenje.

Prilikom odabira najboljeg algoritma također je promatran i broj ispitnih primjera u kojima je određeni algoritam postigao rezultat koji ili nije lošiji od rezultata bilo kojeg drugog promatranog algoritma ili predstavlja najbolji pronađeni rezultat. Ova se mjera može nazvati i postotkom dominacije, a obično je povezana sa ukupnom vrijednošću koju

algoritam postiže za pojedino mjerilo u svim ispitnim primjerima. Postotak dominacije može doprinijeti izboru jednoga algoritma ukoliko se promatrani algoritmi vrlo malo razlikuju po pitanju ukupne vrijednosti rezultata, što je obično slučaj sa rješenjima dobivenim u kasnijim stupnjevima evolucijskog procesa.

4.3 Ostvarenje ispitne okoline

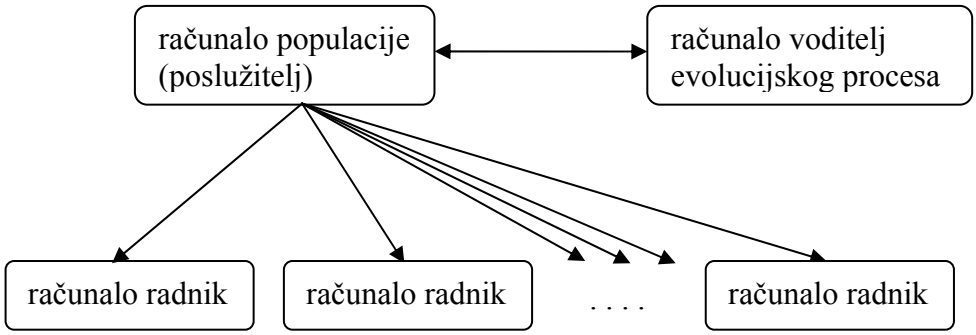
Programska podrška sustava izvođenja pravila raspoređivanja izvedena je u obliku jedinstvenog programa prilagođenog uporabi u svim promatranim okruženjima raspoređivanja. Parametri genetskog programiranja i okoline raspoređivanja zadaju se uz pomoć konfiguracijskih datoteka. U jednoj su datoteci definirani parametri evolucijskog procesa, kao što je vrsta odabira, broj jedinki populacije, uvjeti zaustavljanja itd. U posebnoj datoteci definirani su uvjeti raspoređivanja i osnovne značajke skupa ispitnih primjera. Na ovaj način omogućena je jednostavna izmjena okoline raspoređivanja i pokretanje novog postupka pronalaženja rješenja bez potrebe za ponovnim prevođenjem programa. Svojstva raspoređivanja koja se definiraju uz pomoć datoteke su sljedeća:

- okruženje raspoređivanja (jedan stroj, paralelni jednoliki ili nesrodni strojevi te proizvoljna obrada),
- broj ispitnih primjera,
- brojevi poslova (i strojeva) u ispitnim primjerima,
- postojanje dodatnih uvjeta raspoređivanja (trajanja postavljanja, ograničenja u redoslijedu, dinamički dolasci poslova, dozvoljeno čekanje),
- mjerilo vrednovanja rasporeda,
- nazivi datoteka sa dodatnim parametrima ispitnih primjera (težinske vrijednosti, trajanja, željena vremena završetka itd).

U konfiguracijskoj se datoteci također definiraju i podatkovni elementi koji su na raspolaganju genetskom programu. Postavke za pojedina okruženja navedene su u odgovarajućim poglavljima.

Osim programa za izvođenje pravila raspoređivanja, ostvarena je i programska prodrška za usporedbu učinkovitosti različitih algoritama u obliku zasebnog programa. Ovaj program koristi istu konfiguracijsku datoteku za definiranje okruženja raspoređivanja kao i genetski program. Na osnovu uvjeta raspoređivanja, na zadane ispitne primjere primjenjuju se odgovarajući ugrađeni postupci raspoređivanja kao i postupak dobiven genetskim programiranjem. Za sastavljene rasporede računaju se i u pogodnom obliku zapisuju svi promatrani kriteriji ocjenjivanja. Rezultate dobivene na ovaj način moguće je prikazati u grafičkom ili tabličnom obliku uporabom nekog programa za statističku analizu.

Postupak pronalaženja rješenja genetskim programiranjem može biti vrlo zahtjevan po pitanju utroška procesorskog vremena. Najveći dio ukupnog trajanja obično se sastoji od ocjenjivanja rješenja odnosno određivanja dobrote jedinki populacije. Neke tehnike ubrzavanja rada genetskog programiranja neovisne o ugradnji prikazane su u 5. poglavlju. Međutim, za potrebe ubrzavanja može se iskoristiti i mogućnost paralelnog izvođenja genetskog programa. Postupci evolucijskog računanja posebno su pogodni za razne oblike paralelizacije [Gol 01, Gol 01a]. U ovom je radu omogućeno paralelno izvođenje genetskog programa na razini ocjenjivanja pojedinih rješenja. Struktura korištenog sustava prikazana je na slici 4.4.



Slika 4.4 Struktura paralelnog evolucijskog računanja

Sustav je podijeljen na tri dijela: jedan dio predstavlja računalo poslužitelj na kojemu se nalazi cjelokupna populacija. Na istom ili posebnom računalu odvija se program odgovoran za provođenje evolucijskog procesa: primjenu genetskih operatora i provjeravanje uvjeta zaustavljanja. Budući je evolucijski proces usko vezan za članove populacije, ova se dva dijela nalaze na jednom računalu, mada mogu raditi i odvojeno. Proizvoljan broj računala namijenjen je samo određivanju dobrote rješenja. Ako je računalo radnik slobodno, ono šalje zahtjev poslužitelju za novim jedinkama. Računalo radnik određuje dobrote jedinki koje je primilo od poslužitelja i rezultate šalje natrag poslužitelju. Ciklus se ponavlja sve dok se evolucijski proces ne završi, u kojem trenutku računala radnici dobivaju odgovarajuću poruku od poslužitelja. Sustav kao cjelina istovremeno provodi jedan evolucijski proces, a ocjenjivanje rješenja odvija se paralelno.

Jedan od važnijih parametara sustava je broj jedinki koje će poslužitelj poslati računalu radniku po jednom zahtjevu. Premali broj jedinki uzrokovat će čestu komunikaciju i dodatne troškove, a preveliki broj može usporiti evolucijski proces ako poslužitelj predugo čeka na nekog od radnika. Uz populaciju od 10000 jedinki, broj rješenja koja se šalju u jednom koraku postavljen je na 100. Ovaj broj dovoljno je velik da određivanje dobrote ne traje prekratko, a dovoljno mali u odnosu na veličinu populacije kako bi se postiglo ravnomjerno ujednačavanje opterećenja.

4.4 Raspoređivanje na jednom stroju

4.4.1 Okruženje jednoga stroja

Iako najjednostavniji po svojoj strukturi, problemi raspoređivanja na jednom stroju i dalje su NP teški pod većinom uvjeta, ovisno o kriteriju ocjenjivanja [Mar 04]. Iako su u praksi manje zastupljeni od drugih okruženja, postupci rješavanja problema na jednom stroju često su osnova složenijih postupaka za raspoređivanje u drugim okolinama. Na primjer, u postupcima promjenjivog uskog grla sustava (engl. *shifting bottleneck*) problem raspoređivanja na više strojeva iterativno se svodi na problem raspoređivanja na jednom stroju, pri čemu je potreban učinkovit i brz postupak rješavanja ugrađenog (engl. *embedded*) problema.

Za sve promatrane inačice raspoređivanja na jednom stroju vrijede sljedeće pretpostavke:

- svi podaci o poslovima unaprijed su poznati (predodređeno raspoređivanje),
- raspoređivanje se obavlja dinamički (tijekom rada sustava),
- poslovi su neprekidivi,
- stroj (sredstvo) je neprekidno raspoloživ.

Navedene pretpostavke su u skladu sa uvjetima koji vrijede u većini prikaza ovog problema u literaturi, pa su zadržane i ovdje poradi lakše usporedbe učinkovitosti. S druge

strane, to ne znači da se opisani postupci ne mogu upotrijebiti u sustavima u kojima neke od ovih pretpostavki nisu ispunjene (npr. podaci o poslovima se uvijek mogu mijenjati tijekom rada sustava).

U sljedećem odjeljku dan je pregled rješavanja ovoga problema uporabom genetskog programiranja. Naredni odjeljci opisuju postupke izvođenja pravila raspoređivanja za različite uvjete po pitanju svojstva zadataka koji se raspoređuju.

4.4.2 Pregled raspoređivanja na jednom stroju uz pomoć genetskog programiranja

Primjena evolucijskih algoritama na raspoređivanje do nedavno se svodila na pronalaženje što boljeg rasporeda poslova na sredstvima. Ovim se postupkom pronalazi rješenje samo za jednu instancu problema, a postupak pronalaženja kvalitetnog rješenja često uzima dosta vremena. Genetsko programiranje, budući radi sa podacima promjenjive veličine i strukture, može se iskoristiti i za pretraživanje prostora algoritama raspoređivanja, a ne samo prostora rasporeda.

U radu [Dim 99a] jedinka genetskog programiranja koristi se kao zapis redoslijeda pojedinih pravila raspoređivanja koja se primjenjuju u rješavanju problema. Autori uzimaju u obzir tri pravila (EDD, SPT i Montagne, odjeljak 4.4.3) koje genetski program kombinira i koristi kao alat za raspoređivanje u svakom uzastopnom koraku (u svakom koraku raspoređivanja može se koristiti drugo pravilo). Ovaj pristup prikaza rješenja zapravo je identičan rješenju fiksne duljine, kakvo bi se moglo predstaviti i genetskim algoritmom, budući da autori uzimaju samo onoliko pravila iz jedinke koliko postoji poslova u sustavu. Rješenje dobiveno na ovaj način nije općenito primjenjivo jer ovisi o broju poslova koje treba rasporediti. Kvaliteta dobivenih rješenja također ne odskače od učinkovitosti elementarnih heuristika. Autori pokazuju da se kvaliteta rješenja može povećati kombiniranjem sa postupcima pretraživanja (lokalna pretraga, simulirano kaljenje), no takvi su rezultati očekivani uz većinu heuristika.

Isti autori u [Dim 99, Dim 01] koriste drugačiji pristup: jedinka genetskog programiranja koristi parametre poslova u cilju izračunavanja konačne vrijednosti koja se pridružuje svakom poslu. Prilikom ocjenjivanja, jedinka se iterativno primjenjuje na svaki posao posebno, tako da se istovremeno promatraju parametri samo jednoga posla. Nakon prolaska petlje za sve poslove, posao sa najvećom zabilježenom vrijednošću biva raspoređen na stroj. Na ovaj način genetski program zapravo pronalazi funkciju koja definira pravilo raspoređivanja. Učinkovitost dobivenih rješenja uspoređena je sa gore spomenuta tri pravila raspoređivanja i usporediva je sa učinkovitošću Montagne pravila. Sva pravila za usporedbu, pa tako i pravilo izvedeno od strane genetskog programa, statičke je prirode (dobivene vrijednosti se ne mijenjaju tijekom rada sustava). Isto tako, promatrana je samo statička okolina (svi poslovi su raspoloživi od početka rada) i netežinska inačica problema.

U radu [Ada 02] problem raspoređivanja na jednom stroju rješava se uporabom posebne podatkovne strukture koja sadrži odabrane podatke o poslu (u najmanjem slučaju redni broj posla i trajanje). Pristup određenom dijelu podatkovne strukture omogućen je posebnom funkcijom genetskog programa (za svaki element podatkovne strukture potrebna je posebna funkcija). Posebnost ovoga pristupa je da ne koristi izravno prioritete poslova, već se kao rezultat gleda redni broj posla zapisan u odgovarajuću memorijsku lokaciju, što predstavlja dodatni problem procesu evolucije. Korištenje zasebne funkcije za pristup pojedinom elementu podatkovne strukture čini ovaj model neprikladnim u okruženjima gdje se javlja veći broj podataka vezan uz jedan posao te globalnih podataka nevezanih uz poslove. Nepotrebno povećanje broja različitih funkcijskih čvorova znatno otežava proces pronalaženja učinkovitog rješenja.

Nešto drugačija inačica okruženja razmatra se u radu [Yin 03]. Složenost pristupa očituje se u dopuštenju prestanka rada sredstva (engl. *machine breakdown*) tijekom rada sustava. Nastoji se pronaći odgovarajući način raspoređivanja koji će što više zadovoljiti zadane kriterije (zaostajanje poslova i odstupanje od predviđenih vremena završetka poslova) uz dopuštene prekide u radu sredstva, gdje su prekidi definirani nekom vjerojatnosnom raspodjelom. Pri tome je pretpostavljeno da su poslovi prekidivi, odnosno da posao može bez ikakvih troškova nastaviti obradu nakon popravka stroja. Rješenje je predstavljeno u obliku dvije funkcije, od kojih jedna računa prioritet poslova a druga određuje trajanje čekanja koje se ubacuje između obrade dva uzastopna posla.

U opisanim radovima ne razmatra se postojanje vremena pripravnosti poslova (a time i eventualnog dozvoljenog čekanja), trajanja postavljanja, ograničenja u redoslijedu i različitih mjerila vrednovanja rasporeda.

4.4.3 Statička okolina raspoređivanja

Statička okolina raspoređivanja podrazumijeva raspoloživost svih poslova od početka rada sustava, odnosno vrijedi da su vremena pripravnosti svakog posla jednaka nuli. Pri tome je pretpostavljeno da ne postoje niti ograničenja u redoslijedu poslova niti slijedno ovisna trajanja postavljanja među poslovima.

Pravila raspoređivanja u statičkoj okolini

Ocjena kvalitete rezultata dobivenih genetskim programiranjem provedena je koristeći nekoliko postojećih pravila raspoređivanja. Za svako od pravila navedena je funkcija kojom se određuje prioritet poslova, a najbolji prioritet je onaj sa najvećom vrijednošću. Prioritet pojedinog posla sa rednim brojem j označen je sa π_j.

- WSPT (engl. *weighted shortest processing time*), funkcija prioriteta definirana kao:

$$\pi_j = w_j / p_j \,. \tag{4.6}$$

- EDD (engl. *earliest due date*), definirano sa:

$$\pi_j = 1/d_j \,. \tag{4.7}$$

- Montagne pravilo [Dim 01, Mor 93] (kratica: MON), definirano kao:

$$\pi_j = \left(w_j / p_j\right) \cdot \left(1 - \frac{d_j}{\sum_{i=1}^{n} p_i}\right), \tag{4.8}$$

  gdje je n broj poslova koje treba rasporediti.

- Rachamadugu & Morton pravilo (kratica: RM) [Moh 83], definirano sa:

$$\pi_j = \left(w_j / p_j\right) \left[\exp\left(\frac{-\left(d_j - p_j - time\right)^+}{k \cdot p_{AV}}\right)\right], \tag{4.9}$$

  gdje je p_{AV} prosječno trajanje svih neraspoređenih poslova (tj. svih onih za koje se računaju prioriteti), *time* je trenutno vrijeme, a operator '+' definiran je kao:

$$x^+ = \max\{x, 0\} \,. \tag{4.10}$$

Parametar k može ovisiti o dotičnoj inačici problema, a autori predlažu vrijednost 2 za probleme na jednom stroju. Izraz $\left(d_j - p_j - time\right)$ je dopuštena odgoda (engl. *slack*), tj. količina vremena koju još možemo potrošiti prije početka obrade nekog posla a da on ne zakasni.

Navedena pravila raspoređivanja mogu se koristiti u različitim okruženjima, pa je za svako okruženje potrebno definirati točan način primjene pravila. U statičkom okruženju sa jednim strojem uporaba pravila raspoređivanja opisana je algoritmom 4.1.

```
dok (postoje neraspoređeni poslovi)
{     čekaj dok stroj ne postane raspoloživ;
      odredi prioritete π_j svih neraspoređenih poslova;
      odaberi posao sa najboljim prioritetom;
      rasporedi posao s najboljim prioritetom na stroj;
}
```

Algoritam 4.1 Postupak raspoređivanja po pravilima za statičke uvjete

U algoritmu 4.1 'najbolji' prioritet može značiti onaj sa najvećom ili najmanjom vrijednošću, ovisno o tome kako je definirano pojedino pravilo. Pored navedenih pravila, za kriterij težinskog zbroja zakašnjelih poslova implementirana je heuristika iz [Mor 93] zasnovana na Hodgsonovom algoritmu (koji daje optimalno rješenje za netežinsku inačicu problema). Razlika u odnosu na originalni Hodgsonov algoritam je u tome da se na kraj rasporeda premještaju poslovi sa najmanjim omjerom težine i trajanja, dok se u originalnoj netežinskoj inačici promatra samo najdulje trajanje. Postupak je prikazan kao algoritam 4.2, a u prikazu rezultata označen je sa 'UWT1'. Treba napomenuti da su poslovi koji su u postupku 'izbačeni' iz liste zapravo prebačeni na kraj rasporeda, a njihov međusobni poredak je nebitan (za dani kriterij) jer su svi zakašnjeli.

```
poredaj poslove u listu po EDD algoritmu;
izračunaj vremena završetka svih poslova po trenutnom redoslijedu;
dok (postoje zakašnjeli poslovi u listi)
{     k = prvi zakašnjeli posao;
      pronađi posao r za koji vrijedi w_r/p_r = min{w_i/p_i}, i = 1..k
      obriši posao r iz liste;
      izračunaj nova vremena završetka svih poslova;
}
rasporedi poslove u listi po dobivenom redoslijedu;
rasporedi obrisane poslove po proizvoljnom redoslijedu;
```

Algoritam 4.2 Heuristika za težinski zbroj zakašnjelih poslova (UWT1)

Raspoređivanje uz pomoć genetskog programiranja

Model raspoređivanja pomoću pravila izvedenog genetskim programom u ovoj je okolini istovjetan algoritmu 4.1, s time da se prioriteti poslova određuju funkcijom koja je dobivena kao rješenje postupka genetskog programiranja. Razlika u odnosu na ostala promatrana pravila je da se kao najbolji prioritet uvijek smatra onaj sa *najmanjom* vrijednošću. Zadatak je genetskog programiranja definirati takvu funkciju prioriteta koja će postići što bolje rezultate (ovisno o zadanom kriteriju). U pozadini ovog nastojanja je ideja da je, osim prikazanih pravila raspoređivanja, moguće pronaći još neke izraze koji bi na bolji način od postojećih pravila mogli iskoristiti dostupne podatke o problemu.

Prikaz rješenja u ovom okruženju je stablo koje predstavlja funkciju prioriteta (jedinka je prikazana stablom). Skup mogućih čvorova stabla mora omogućiti sustavu da na učinkovit način predstavi rješenje problema. Odabir elemenata u skupu čvorova načinjen je ručno za svako pojedino okruženje. U odabiru čvorova nastoji se koristiti one

elemente koji su relevantni za specifično okruženje i uvjete raspoređivanja. Iako je u nekim slučajevima odluka o uključivanju ili izostavljanju nekog elementa trivijalna (npr. u statičkom problemu nema smisla uvoditi vrijeme pripravnosti poslova), općenito je to zadatak koji zahtijeva znanje o problemu koji se rješava. Budući da kvaliteta mogućeg rješenja najviše ovisi o svojstvima skupa čvorova, odabir elemenata nije trivijalan i u biti predstavlja najtežu zadaću u rješavanju postavljenog problema. Iz toga će razloga u poglavlju 5 biti predložena metoda prilagodbe koja bi trebala olakšati ovaj korak.

Elemente skupa čvorova dijelimo u dvije skupine: skup funkcijskih čvorova i podatkovnih čvorova (listova stabla). U slučaju statičkih uvjeta raspoređivanja na jednom stroju, relevantni podaci koji bi u svakom slučaju trebali biti uključeni su trajanje poslova, njihove težine i željena vremena završetka. Korisnu informaciju sustavu mogu predstavljati i zbroj trajanja svih poslova te zbroj željenih vremena završetka. Budući je primjena pravila raspoređivanja takva da se u svakom koraku ponovno računaju prioriteti svih neraspoređenih poslova, povijest rada sustava ne utječe na odluku o sljedećem poslu. Iz toga su razloga u skup listova dodani broj preostalih poslova te zbroj njihovih trajanja. Kako bi postupak imao dobru informaciju o žurnosti raspoređivanja nekog posla, dodana je i informacija o vremenu za koje posao još može pričekati prije početka rada, a da ne prekorači željeno vrijeme završetka (dopuštena odgoda).

Tablica 4.2 Popis čvorova za statički problem na jednom stroju

Oznaka funkcijskog čvora	Definicija
ADD	binarni operator zbrajanja
SUB	binarni operator oduzimanja
MUL	binarni operator množenja
DIV	zaštićeno dijeljenje: $DIV(a,b)=\begin{cases}1, \text{ ako } \lvert b\rvert < 0.000001\\ a/b, \text{ inače}\end{cases}$
POS	unarni operator '+': $POS(a)=\max\{a,0\}$
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
N	ukupni broj poslova
Nr	preostali broj poslova (koji još nisu raspoređeni)
SP	zbroj trajanja svih poslova
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$

U skup funkcijskih čvorova uvršteni su čvorovi koji podržavaju četiri osnovne računske operacije, s tom razlikom da je operatoru dijeljenja dodana 'zaštita' od dijeljenja sa premalom vrijednošću. Također je definiran unarni operator koji vraća pozitivnu vrijednost argumenta odnosno nulu ako je argument negativan. Pregled konačnog skupa čvorova prikazan je u tablici 4.2.

Ispitni primjeri za statičku okolinu

Ispitni primjeri za statičku okolinu definiraju se sljedećim elementima: trajanjima poslova, težinama poslova i željenim vremenima završetka. Trajanje svakog posla može

poprimiti cjelobrojne vrijednosti u intervalu od 1 do 100, a težine poslova vrijednosti od 0.01 do 1 u koracima od 0.01. Raspodjele slučajnih varijabli za trajanje poslova opisane su u odjeljku 4.2.3, a sve ostale veličine dobivene su po jednolikoj raspodjeli. Za svaki ispitni primjer su, osim toga, definirana dva dodatna parametra s pomoću kojih se računaju željena vremena završetka poslova. Parametar T je postotak zaostajanja (engl. *due date tightness*), a parametar R je područje zaostajanja (engl. *due date range*) [Lee 97]; oba parametra poprimaju vrijednosti u intervalu $[0,1]$. Za svaki ispitni primjer željena vremena završetka definiraju se jednolikom raspodjelom unutar intervala

$$d_j \in \left[\sum_{j=1}^{n} p_j \left(1-T-R/2\right), \sum_{j=1}^{n} p_j \left(1-T+R/2\right) \right], \tag{4.11}$$

gdje n predstavlja broj poslova u ispitnom primjeru, uz ograničenje da dobivena vrijednost ne može biti manja od nule. Značenje ovih dvaju parametara je sljedeće: parametrom T zadajemo očekivani postotak zakašnjelih poslova, dok parametrom R određujemo širinu intervala vrijednosti koje željena vremena završetka mogu poprimiti. Na primjer, uz $T=1$ očekuje se da će svi poslovi biti zaostali, premda takva situacija nije previše realna.

Za potrebe ocjenjivanja definirano je 100 ispitnih primjera za učenje i 600 ispitnih primjera za naknadnu ocjenu učinkovitosti pravila (genetskom programu su na raspolaganju prvih 100 primjera, dok se kvaliteta izvedenog pravila ocjenjuje na temelju drugih 600 primjera koji nisu dostupni tijekom učenja). Za svaki ispitni primjer određen je broj poslova i vrijednosti parametara T i R. Broj poslova u ovom okruženju poprima vrijednosti od 12, 25, 50 i 100, a parametri T i R poprimaju vrijednosti 0.2, 0.4, 0.6, 0.8 i 1 u različitim kombinacijama. Detaljan popis parametara za tvorbu ispitnih primjera dan je u prilogu A1. Osim toga, iz raznih izvora [Bea 90] preuzeto je 375 dodatnih ispitnih primjera koji se koriste samo za ocjenjivanje.

Problem težinskog zaostajanja

Prva skupina pokusa u statičkom okruženju jednoga stroja provedena je za optimiranje težinskog zaostajanja poslova. Težinsko zaostajanje je najčešće rabljeni kriterij u ovom okruženju, a oznaka ovoga problema u $\alpha|\beta|\gamma$ zapisu je $1|\ |\sum w_j T_j$. Ocjena svakog od pravila dobivena je na način definiran u odjeljku 4.2, tj. dijeljenjem sa prosječnom težinom, trajanjem ili brojem poslova u dotičnom primjeru.

Najbolje dobiveno rješenje odabrano je iz 30 provedenih pokusa po uspješnosti rješenja na skupu ispitnih primjera za ocjenu, koji se ne koriste u učenju. Rješenje se može prikazati u obliku stabla u kojemu je i predstavljeno unutar genetskog programa. Zbog svoje veličine, rješenja se obično prikazuju u nekom drugom zapisu, od kojih je najuobičajeniji infiksni zapis, koji se koristi i u ovom radu. Dobiveno rješenje predstavljeno je izrazom na slici 4.5.

```
π = pt/w-pos((pt+Nr)/pt)+((SL+pt)*(2*SL^2/w-2*w)/Nr^2*w+pos(pos((SL+pt+(2
*SL+pt)*(SL+w)/w)/SD*pt*(2*SL^2/w-w-(SL+pt/w+SL/(w+SD)*(Nr+SL))*(Nr+SPr)
-pos((Nr+SD)/(Nr+SL)+pt+SL+(2*pt+2*Nr+SD)/(Nr+SL)+pt/w-pos(Nr/pt))))))/S
Pr+SL
```

Slika 4.5 Zapis rješenja – izvedena funkcija prioriteta

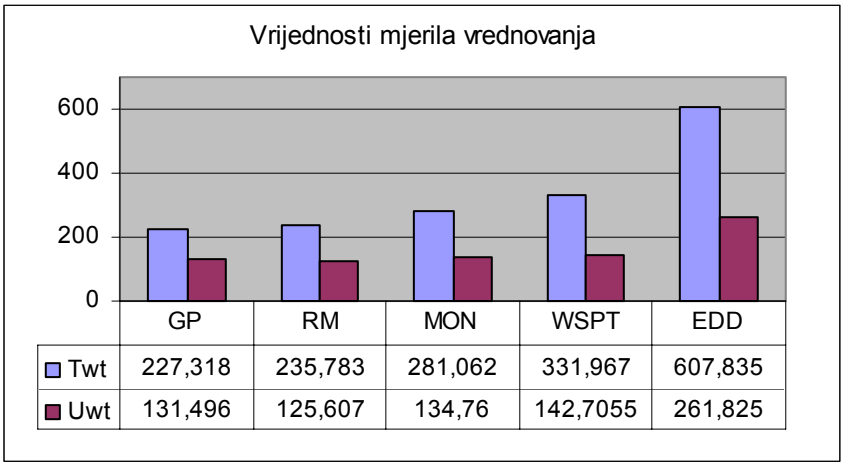
Potrebno je napomenuti da je priroda dobivenog rješenja gotovo uvijek takva da ono nije analitički 'ispravno', budući su u rješenje u većini slučajeva uključeni i elementi koji imaju vrlo mali utjecaj na konačni rezultat, odnosno predstavljaju 'šum'. Ova je osobina svojstvena svim rješenjima u većini primjena genetskog programiranja (poglavlje

3.1.3), no to ne umanjuje njihovu korisnost u uporabi. Rješenje prikazano na slici 4.5 nije potpuno identično dobivenom rješenju, nego je prikazano u djelomično pojednostavljenom obliku. U ovom se radu rabi tehnika pojednostavljivanja rješenja koja je, budući je nevezana uz problem raspoređivanja, posebno opisana u poglavlju 5.4, a sva dobivena rješenja prethodno su pojednostavljena uporabom ove metode.

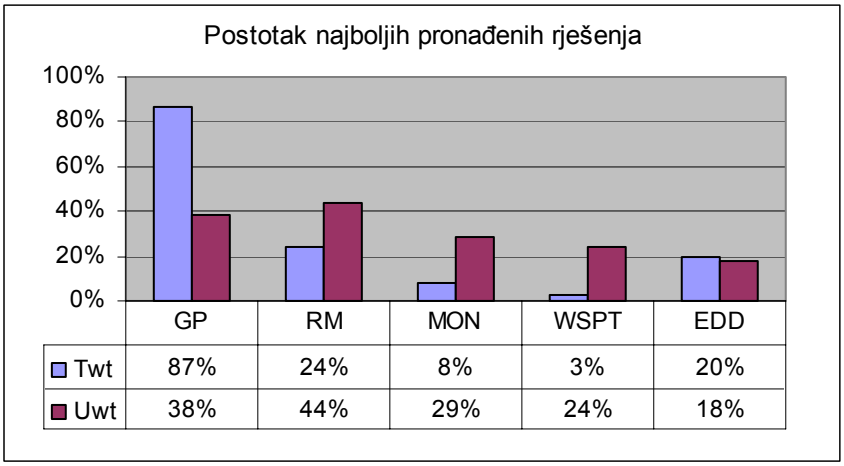
Rezultati za sva promatrana pravila dani su u dva dijela; u prvom dijelu prikazan je ukupni iznos funkcije dobrote za sve ispitne primjere. Budući da iznos dobrote predstavlja troškove rasporeda (u evolucijskom procesu provodi se postupak minimiziranja), za prvi dio prikaza rezultata vrijedi pravilo 'manje je bolje'. U drugom dijelu rezultata, za svaki ispitni primjer pronađeno je najbolje rješenje (dobiveno od bilo kojeg pravila), a potom je za svako pravilo utvrđen postotak ispitnih primjera u kojima pravilo daje rješenje jednako najboljemu. Ovaj dio rezultata pokušava pokazati u kolikom dijelu slučajeva bi određeno pravilo pronašlo rješenje koje nije lošije od svih drugih pravila, odnosno u kolikom postotku određeno pravilo dominira nad drugim promatranim pravilima.

Zbog potpunosti, ovdje će se prikazati i rezultati postignuti na skupu za učenje, dok će u narednim pokusima biti prikazani samo rezultati na skupu za ocjenu. U prikazu kvalitete rješenja, poradi potpunijeg uvida u učinkovitost pojedine metode, dani su rezultati i za težinsko zaostajanje (oznaka 'Twt') i za težinsku zakašnjelost (oznaka 'Uwt'), iako se kao kriterij prilikom postupka učenja koristi samo težinsko zaostajanje. Na ovaj način može se doći i do zaključaka o 'korisnosti' određenog kriterija, tj. po kojem je kriteriju najisplativije optimirati a da i ostala mjerila postignu relativno dobru vrijednost. Vrijednosti težinske zakašnjelosti pomnožene su sa odgovarajućim faktorom za pojedini skup primjera kako bi iznosi u grafičkom prikazu bili sumjerljivi sa vrijednostima težinskog zaostajanja. Za skup primjera za učenje, vrijednosti težinskog zaostajanja prikazani su na slici 4.6, a postoci najboljeg pronađenog rješenja na slici 4.7.

Iz prikaza rezultata može se vidjeti da je pravilo dobiveno genetskim programiranjem uspješnije od ostalih promatranih pravila. Sam iznos ukupnog težinskog zaostajanja nije puno bolji od sljedeće najbolje metode, no u broju primjera u kojima se postiže najbolje nađeno rješenje, izvedeno pravilo uvjerljivo prednjači. Na slikama 4.8 i 4.9 prikazani su rezultati istoga pravila na skupu ispitnih primjera za ocjenu.

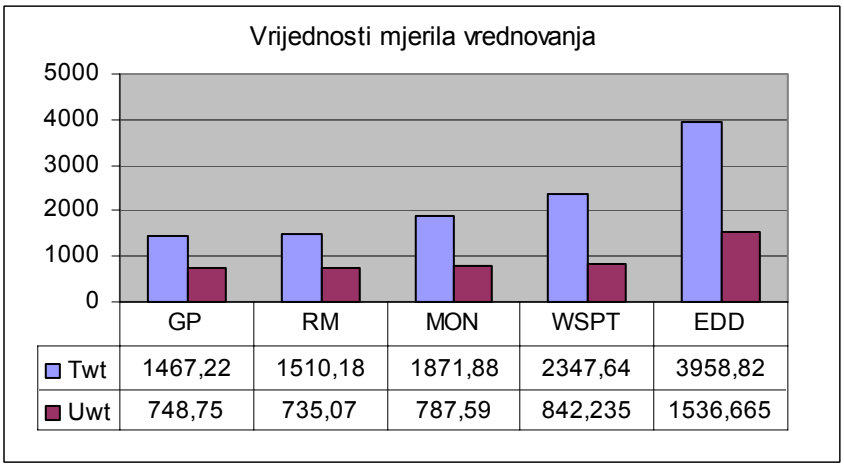


Slika 4.6 Optimiranje težinskog zaostajanja – skup primjera za učenje

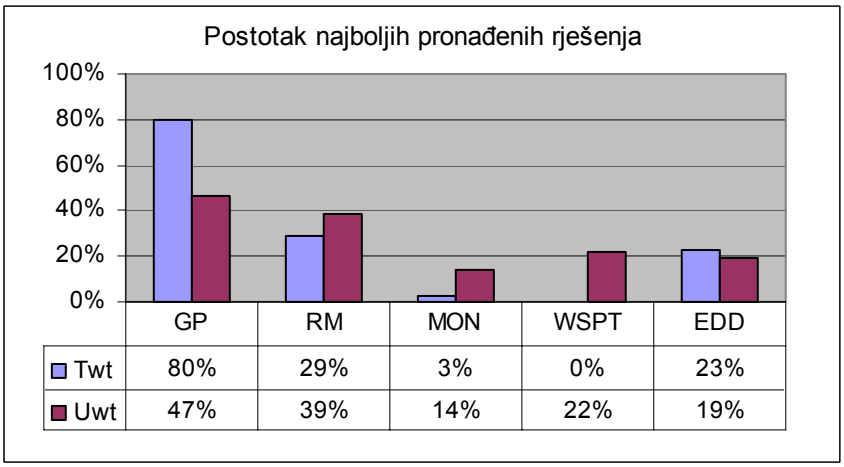


Slika 4.7 Postoci dominacije za težinsko zaostajanje – skup primjera za učenje

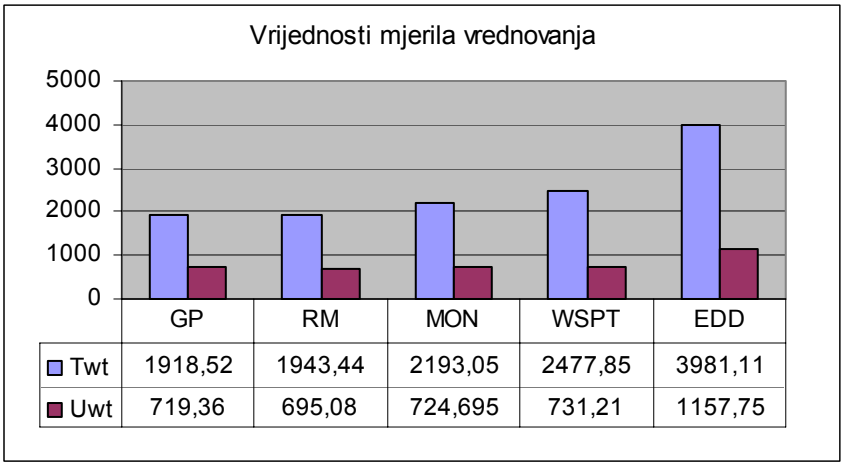
Kao što je vidljivo iz rezultata, izvedeno pravilo uspijeva dominirati i u skupu primjera za ocjenu, mada relativna prednost nije tolika kao kod skupa primjera za učenje. Konačno, uspješnost istoga pravila uspoređena je na skupu primjera iz literature koji se sastoji od 375 primjera (po 125 primjera sa 40, 50 i 100 poslova). Rezultati na ovom skupu prikazani su koristeći jednaka mjerila na slikama 4.10 i 4.11.



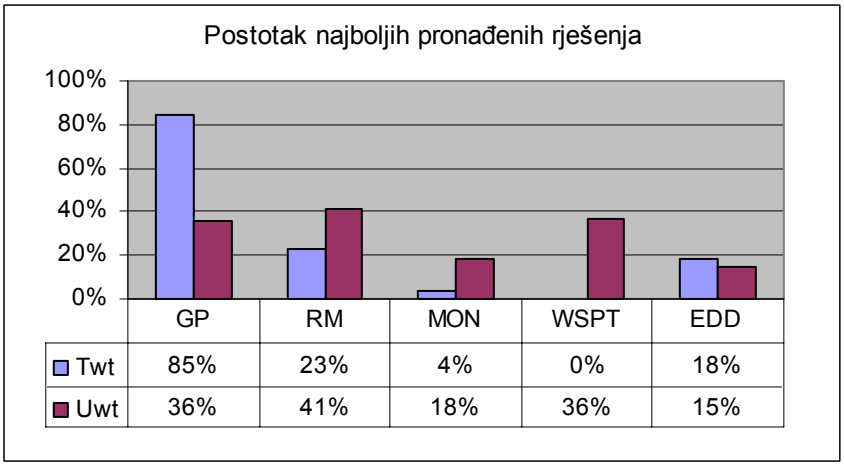
Slika 4.8 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



Slika 4.9 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu



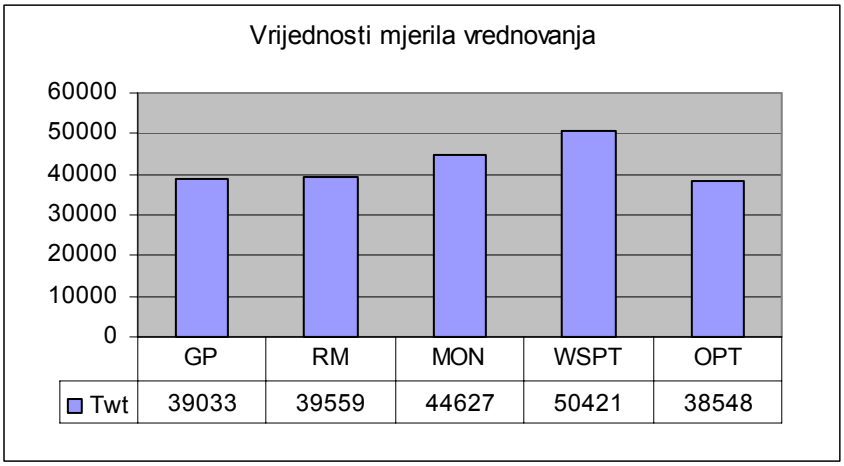
Slika 4.10 Optimiranje težinskog zaostajanja – skup primjera iz literature



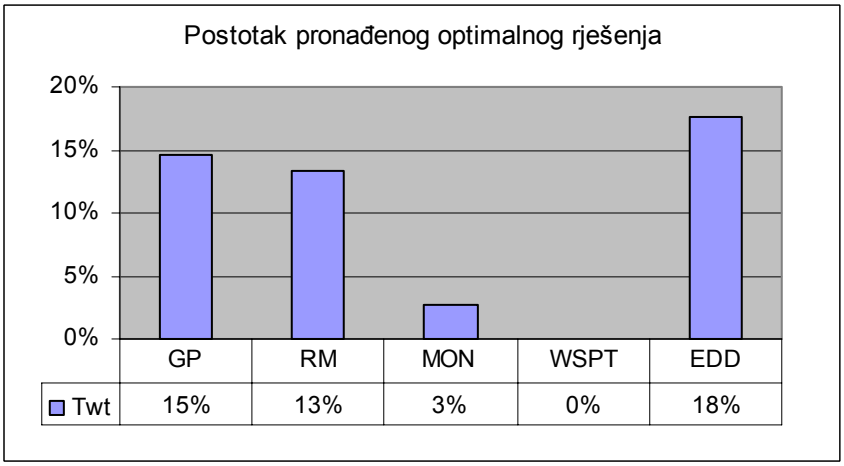
Slika 4.11 Postoci dominacije za težinsko zaostajanje – skup primjera iz literature

Budući su uz primjere iz literature navedene optimalne vrijednosti rješenja za problem težinskog zaostajanja, zanimljivo je usporediti promatrana pravila uz vrijednost koja se postiže optimalnim rasporedom (oznaka 'OPT'). Potrebno je napomenuti da su optimalne vrijednosti za te ispitne primjere na raspolaganju samo u nenormiranom obliku – iznos kriterija dobiven je po definiciji težinskog zaostajanja (izraz (2.9), poglavlje 2.2.4), bez dijeljenja sa brojevima poslova, srednjom težinskom vrijednošću itd. Bez obzira na tu razliku, pravilo izvedeno za normirane kriterije pokazuje gotovo jednaku učinkovitost i uz ovakav način ocjenjivanja. Na slici 4.12 prikazane su vrijednosti za ukupno težinsko zaostajanje svih postupaka (osim za EDD čija je postignuta vrijednost duplo veća – lošija od najbolja tri rješenja) i optimalnog rješenja.

Također je zanimljivo pogledati u kojem broju slučajeva određeno pravilo daje optimalno rješenje. Ovi su postoci prikazani na slici 4.13, a budući da optimalno rješenje ima vrijednost 100%, nije uključeno u prikaz.



Slika 4.12 Vrijednosti težinskog zaostajanja uz optimalno rješenje



Slika 4.13 Postoci pronalaženja optimalnog rješenja – skup primjera iz literature

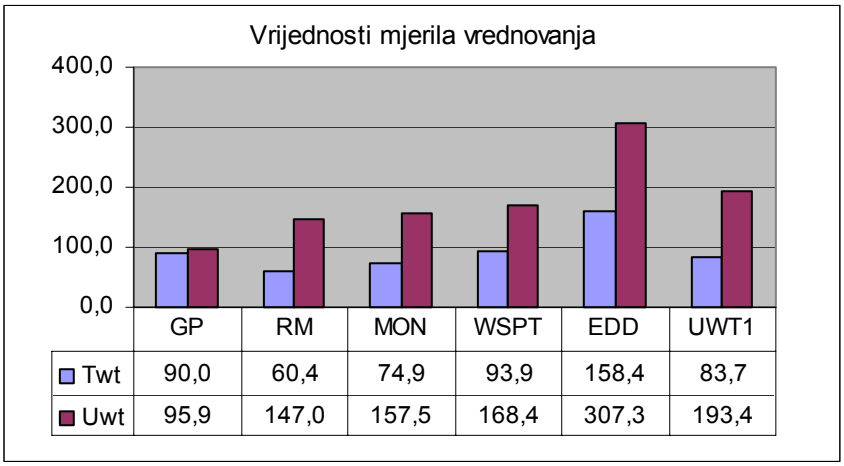
Rezultat koji pokazuje da najveći broj pronalaženja optimalnog rješenja postiže EDD pravilo, iako ima uvjerljivo najlošiju učinkovitost po ukupnom kriteriju, nije iznenađujuć. Naime, svi ispitni primjeri u kojima pravila daju optimalno rješenje gotovo su isključivo primjeri sa 0% očekivanog zaostajanja (tj. parametar T je nula, a parametar R je relativno mali), a vrijednost ukupnog zaostajanja uz optimalni raspored jednaka je nuli. Za slučajeve u kojima niti jedan posao ne zaostaje, može se dokazati [Mor 93] da EDD pravilo daje optimalno rješenje, pa broj takvih slučajeva u ispitnim primjerima izravno utječe na postotak pronalaženja optimuma po EDD pravilu.

Problem težinske zakašnjelosti

Problem težinske zakašnjelosti poslova može se opisati situacijom u kojoj postoje narudžbe i rokovi dostave za neke proizvode, a u slučaju prekoračenja roka za određeni posao plaća se konstantni iznos troška. Iznos troška u slučaju zakašnjelosti ovisi o težini pojedinoga posla ali ne ovisi o količini vremena za koju je posao prekoračio zadani rok. Drugim riječima, nastoji se minimizirati težinski zbroj troškova zakašnjelosti. U $\alpha|\beta|\gamma$ zapisu oznaka problema je $1|\ |\sum w_j U_j$.

Genetski algoritam ovdje ima jednak zadatak kao u slučaju težinskog zaostajanja – cilj je pronaći takvu prioritetnu funkciju koja će, koristeći je u obliku pravila, minimizirati opisane troškove. Algoritam primjene pravila raspoređivanja isti je kao i za prethodni kriterij. Budući je težinska zakašnjelost mjerilo koje ovisi o sličnim veličinama kao i

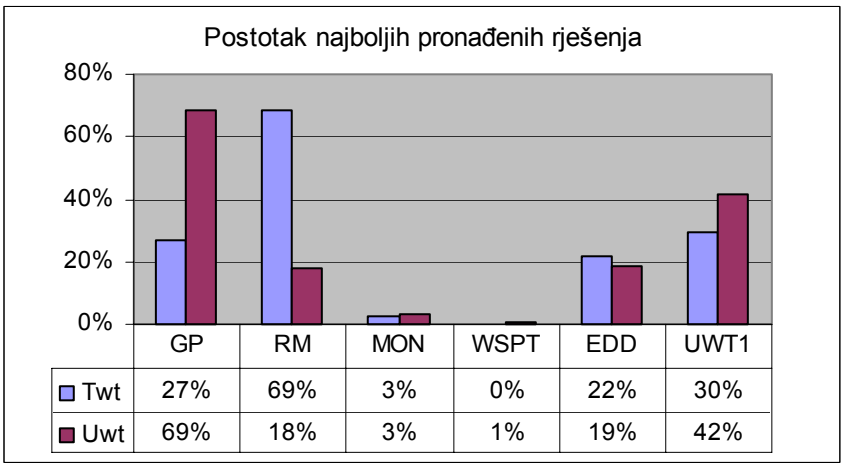
težinsko zaostajanje, genetskom programu su na raspolaganju jednaki funkcijski i podatkovni elementi kao i u prethodnom primjeru (tablica 4.2). Prilikom uspoređivanja rješenja, u skup promatranih pravila dodana je i heuristika za problem težinske zakašnjelosti (algoritam 4.2), koja je u prikazu rezultata označena sa 'UWT1'. Za potrebe ovoga kriterija provedeno je 10 pokusa sa jednakim vrijednostima parametara kao i u prethodnom primjeru. Najbolje rješenje također je odabrano promatrajući učinkovitost pravila na skupu za ocjenu (rezultati na skupu za učenje ovdje nisu uvršteni), a odabrano rješenje je navedeno u prilogu B1. Na slikama 4.14 i 4.15 prikazani su rezultati svih promatranih pravila na skupu od 600 ispitnih primjera za ocjenu.



	GP	RM	MON	WSPT	EDD	UWT1
Twt	90,0	60,4	74,9	93,9	158,4	83,7
Uwt	95,9	147,0	157,5	168,4	307,3	193,4

Slika 4.14 Optimiranje težinske zakašnjelosti – skup primjera za ocjenu

Iz rezultata se vidi da rješenje stvoreno genetskim algoritmom uvjerljivo dominira nad ostalim pravilima, no očit je 'pad učinkovitosti' po pitanju drugog kriterija, težinskog zaostajanja. Ukoliko nam je kao mjerilo važna jedino zakašnjelost, ovakav je rezultat i više nego dobar. Ukoliko, međutim, tražimo ujednačenu kvalitetu pravila, tada je bolji izbor optimiranje po kriteriju zaostajanja.



	GP	RM	MON	WSPT	EDD	UWT1
Twt	27%	69%	3%	0%	22%	30%
Uwt	69%	18%	3%	1%	19%	42%

Slika 4.15 Postoci dominacije za težinsku zakašnjelost – skup primjera za ocjenu

Heuristika za težinsku zakašnjelost (algoritam 4.2) nije pokazala uvjerljive rezultate po pitanju zbroja ukupnih troškova za sve ispitne primjere, no postotak dominacije toga pravila (odnosno broj slučajeva u kojima nije bila lošija od drugih metoda) na drugom je mjestu odmah iza pravila dobivenog genetskim programiranjem.

4.4.4 Dinamička okolina raspoređivanja

U dinamičkoj okolini raspoređivanja uvode se vremena pripravnosti poslova koja označuju trenutak kada može započeti obrada nekog posla. U $\alpha|\beta|\gamma$ zapisu ova okolina ima oznaku $1|r_j|*$. Najčešća mjerila vrednovanja rasporeda koja se uzimaju u obzir u ovoj okolini su također težinsko zaostajanje i težinska zakašnjelost.

Pravila raspoređivanja u dinamičkoj okolini

U dinamičkoj okolini moguće je primjenjivati i pravila opisana u prethodnom odjeljku, no za drugačiju okolinu postavlja se pitanje interpretacije pravila. Budući svi poslovi nisu raspoloživi u svakom trenutku, prilikom odabira mogućeg sljedećeg posla mora se definirati koji podskup poslova se uzima u obzir. Isto tako, potrebno je definirati hoće li se prioritet poslova koji još nisu pripravni računati na način različit od računanja prioriteta za raspoložive poslove.

Prvo pitanje na koje treba odgovoriti je dopustiti li uopće čekanje na neki posao kojemu je vrijeme pripravnosti u budućnosti, te koliko daleko u budućnost je potrebno promatrati. Kao što je spomenuto u poglavlju 2.2.3, u određenom sustavu čekanje može biti i nedozvoljeno, no ovdje će se razmatrati samo slučaj dozvoljenog čekanja (uz minimalne izmjene, pravila se mogu uporabiti i u uvjetima nedozvoljenog čekanja). Moguće je dokazati da, uz regularni kriterij raspoređivanja na jednom stroju, eventualno čekanje na neki budući posao ne treba biti dulje od trajanja najkraćeg raspoloživog neraspoređenog posla (jednaki pristup uporabljen je i u nekim drugim alatima raspoređivanja [Lek 03]). Po tom pravilu, u postupak ocjenjivanja ulaze samo oni poslovi čije je vrijeme pripravnosti prije završetka najkraćeg raspoloživog posla, odnosno samo oni poslovi j za koje vrijedi:

$$|r_j - time| \leq \min_i \{p_i\}, \forall i : r_i \leq time \tag{4.12}$$

Drugi problem javlja se prilikom definicije funkcije prioriteta pojedinog pravila: budući će pravilo izvedeno genetskim programiranjem imati na raspolaganju informaciju o vremenu pripravnosti posla, potrebno je tu informaciju uključiti i u primjenu ostalih pravila koja se koriste za usporedbu. Stoga se u primjeni svih pravila koja sama po sebi podrazumijevaju raspoloživost posla, trajanju obrade posla dodaje količina vremena za koju posao postaje pripravan, a ta se veličina označava sa ar (engl. *arrival*), odnosno:

$$ar = \max\{r_j - time, 0\}. \tag{4.13}$$

Naravno, ukoliko je neko pravilo po svojoj definiciji namijenjeno dinamičkoj okolini (odnosno uključuje informaciju o vremenu pripravnosti posla), tada se vrijednosti u dotičnom pravilu ne mijenjaju. Opisani način primjene pravila raspoređivanja u dinamičkoj okolini može se opisati algoritmom 4.3.

```
dok (postoje neraspoređeni poslovi)
{     čekaj dok stroj ne postane raspoloživ;
      p_jMIN = trajanje najkraćeg raspoloživog posla;
      odredi prioritete π_j svih poslova za koje je |r_j − time| < p_jMIN
      odaberi posao sa najboljim prioritetom;
      rasporedi posao s najboljim prioritetom na stroj;
}
```

Algoritam 4.3 Postupak raspoređivanja po pravilima za dinamičke uvjete

Algoritam 4.3 može se smatrati ovojnicom oko pojedinog pravila raspoređivanja, odnosno *meta-algoritmom* koji definira primjenu različitih pravila. Uz definirani način primjene, promatrana pravila raspoređivanja u dinamičkoj okolini navedena su u nastavku.

- WSPT pravilo, funkcija prioriteta zadana sa:

$$\pi_j = w_j / \left(p_j + ar_j \right). \tag{4.14}$$

- EDD pravilo, funkcija prioriteta jednaka izrazu (4.7).
- Montagne pravilo (MON), uz funkciju prioriteta:

$$\pi_j = \left(w_j / \left(p_j + ar_j \right) \right) \cdot \left(1 - \frac{d_j}{\sum_{i=1}^{n} p_i} \right). \tag{4.15}$$

- Rachamadugu & Morton (RM) pravilo, definirano kao:

$$\pi_j = \left(w_j / \left(p_j + ar_j \right) \right) \left[\exp\left(\frac{-\left(d_j - \left(p_j + ar_j \right) - time \right)^+}{k \cdot p_{AV}} \right) \right]. \tag{4.16}$$

- Pravilo prilagođeno dinamičkoj okolini, *X-dispatch dottleneck dynamics heuristic* [Mor 93] (kratica: XD); funkcija prioriteta je:

$$\pi_j = \left(w_j / p_j \right) \left[\exp\left(\frac{-\left(d_j - p_j - time \right)^+}{k \cdot p_{AV}} \right) \right] \cdot \left(1 - B \frac{\left(r_j - time \right)^+}{p_{AV}} \right), \tag{4.17}$$

  gdje je p_{AV} prosječno trajanje svih neraspoređenih poslova, a parametar B je opisan izrazom $B = 1.3 + \rho$. Parametar ρ opisuje prosječno opterećenje stroja, a za potrebe ispitnih primjera u ovom radu njegova je vrijednost postavljena na 1.

U prethodno opisanoj statičkoj okolini nema potrebe za dopuštanjem prekidivosti poslova, budući su svi poslovi raspoloživi u isto vrijeme. U dinamičkoj okolini je moguće uključiti i prekidanje, u kojem slučaju prestaje potreba za umetnutim čekanjem pa se odluka o pokretanju sljedećeg posla svodi samo na trenutno pripravne poslove.

Raspoređivanje uz pomoć genetskog programiranja

U ovoj je okolini prikaz rješenja genetskog programiranja identičan prikazu u prethodnom poglavlju, tj. jedinka je predstavljena stablom koje zamjenjuje funkciju prioriteta pravila raspoređivanja. I dalje vrijedi pravilo da se najboljim poslom smatra onaj čija je vrijednost funkcije prioriteta najmanja. Procesu učenja potrebno je omogućiti uporabu informacije o vremenu pripravnosti pojedinoga posla, te također definirati koji se poslovi uzimaju u obzir prilikom ocjenjivanja. Stoga je u skup podatkovnih čvorova, u odnosu na skup iz prethodne okoline, dodana količina vremena za koju posao postaje pripravan, odnosno vrijednost definirana izrazom (4.13). Konačni skup čvorova prikazan je u tablici 4.3.

Tablica 4.3 Popis čvorova za dinamički problem na jednom stroju

Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2

Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
N	ukupni broj poslova
Nr	preostali broj poslova (koji još nisu raspoređeni)
SP	zbroj trajanja svih poslova
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
AR	vrijeme do pripravnosti posla, $\max\{r_j - time, 0\}$

U jednom dijelu pokusa, prilikom primjene izvedenog pravila uporabljen je algoritam 4.3. Osim navedenoga postupka, pokusi su provedeni i uporabom nešto drugačijeg algoritma primjene pravila koji je prikazan kao algoritam 4.4.

```
dok (postoje neraspoređeni poslovi)
{     čekaj dok stroj ne postane raspoloživ;
      p_MIN = trajanje najkraćeg raspoloživog posla;
      p_MAX = trajanje najduljeg posla uz r_j ≤ time + p_MIN ;
      odredi prioritete π_j svih poslova za koje je |r_j − time| < p_MAX
      odaberi posao J_i sa najboljim prioritetom;
      ako(|r_i − time| < p_MIN )
          rasporedi J_i ;
      inače
      {   odaberi posao J_k sa najboljim prioritetom uz uvjete r_k + p_k ≤ r_i i
              |r_k − time| < p_MIN ;
          rasporedi J_k ;
      }
}
```

Algoritam 4.4 Postupak raspoređivanja za dinamičke uvjete

Rad navedenog algoritma može se opisati na ovaj način: prilikom odluke o sljedećem poslu uzet ćemo u obzir sve poslove koji postaju pripravni unutar trajanja *najduljeg* posla koji može započeti prije završetka trenutno najkraćeg raspoloživog posla. Ovaj uvjet definira 'obzor promatranja' algoritma, budući da svi poslovi koji odgovaraju navedenom uvjetu mogu biti odgođeni odlukom o raspoređivanju nekog posla koji započinje ranije. Ukoliko odabrani posao može započeti prije završetka najkraćeg raspoloživog posla, odabrani se raspoređuje na stroj. Ukoliko odabrani posao počinje kasnije, odabiremo sljedeći najbolji posao koji može započeti prije završetka najkraćega posla a završiti prije početka prvotno odabranoga posla.

Motivacija iza ovakvoga načina rada je izbjegavanje mogućnosti da se iza završetka najkraćega raspoloživog posla pojavi posao visokog prioriteta koji bi inače mogao biti odgođen. Ako se takav posao pronađe, pokušava se rasporediti posao koji može završiti prije dolaska posla visokog prioriteta. Pri tome je i dalje zadržan princip o

čekanju kraćem od trajanja trenutno najkraćeg pripravnog posla. Naravno, teret odluke o 'dovoljno visokom' prioritetu i usporedbi s ostalim pripravnim poslovima ostaje na pravilu raspoređivanja.

Ispitni primjeri za dinamičku okolinu

Ispitni primjeri definirani su na sličan način kao i za statičku okolinu: pojedini primjer opisan je različitim vrijednostima parametara T i R, a područja dopuštenih vrijednosti jednaka su onima za statičku okolinu. Razlika u odnosu na prethodnu okolinu je uvođenje vremena pripravnosti poslova i različito računanje željenog vremena završetka. Za svaki ispitni primjer prvo su određena trajanja poslova te je izračunato ukupno trajanje svih poslova. Vremena pripravnosti generirana su jednolikom raspodjelom u intervalu

$$r_j \in \left[0, \frac{1}{2}\sum_{i=1}^{n} p_i\right]. \tag{4.18}$$

Željeno vrijeme završetka pojedinog posla definirano je jednolikom raspodjelom u intervalu

$$d_j \in \left[r_j + \left(\sum_{j=1}^{n} p_j - r_j\right)\cdot\left(1 - T - R/2\right), r_j + \left(\sum_{j=1}^{n} p_j - r_j\right)\cdot\left(1 - T + R/2\right)\right]. \tag{4.19}$$

Razlika u odnosu na određivanje željenih vremena završetka u statičkoj okolini (po izrazu (4.11)) je u tome da se za srednju vrijednost uzima razlika ukupnog trajanja i vremena pripravnosti posla, a dobivena vrijednost zbraja se sa vremenom pripravnosti. Kao i u prethodnom odjeljku, načinjeno je 100 ispitnih primjera za učenje i 600 primjera za ocjenu, a vrijednosti parametara T i R za te primjere navedene su u prilogu A1.

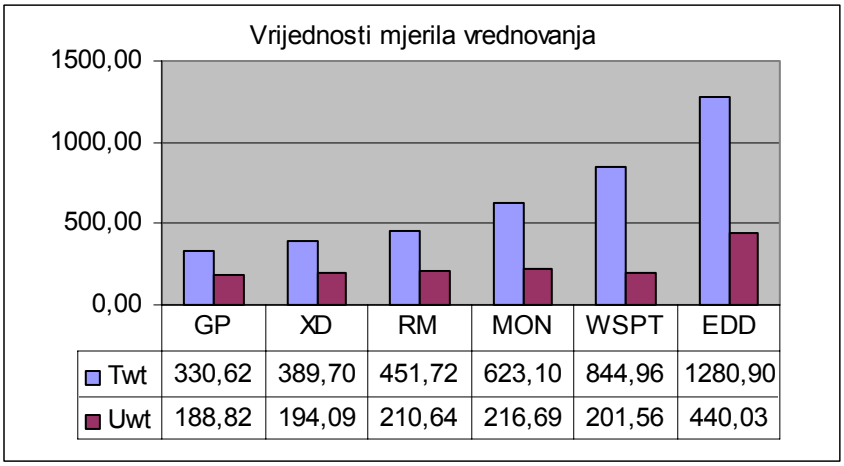
Problem težinskog zaostajanja

Za rješavanje problema težinskog zaostajanja u dinamičkoj okolini provedeno je 20 pokusa po algoritmu 4.3 i 20 pokusa po algoritmu 4.4 (u svim slučajevima korišteni su jednaki ispitni primjeri). U tablici 4.4 prikazane su najbolje postignute vrijednosti u svim pokusima za oba algoritma. Iako je algoritam 4.3 postigao bolje rezultate na skupu za učenje, algoritam 4.4 daje bolji rezultat na skupu za ocjenu. To može upućivati na činjenicu da složeniji oblik algoritma ima veće mogućnosti prilagodbe različitom okruženju. Potrebno je međutim napomenuti da je razlika učinkovitosti pravila prilikom primjene ovih dvaju algoritama u većini primjera mala, što se može protumačiti nedostatkom situacija (unutar ispitnih primjera) u kojima bi algoritam 4.4 mogao iskoristiti veći obzor promatranja i time donijeti bolju odluku. U prikazu usporednih rezultata koriste se rješenja dobivena primjenom algoritma 4.4, iako su obje inačice postigle vrlo dobre rezultate u usporedbi sa ostalim promatranim pravilima.

Tablica 4.4 Rezultati algoritama raspoređivanja u dinamičkim uvjetima

Postupak	najbolji rezultat na skupu za učenje	najbolji rezultat na skupu za ocjenu
po algoritmu 4.3	57,70	335,09
po algoritmu 4.4	58,79	330,62

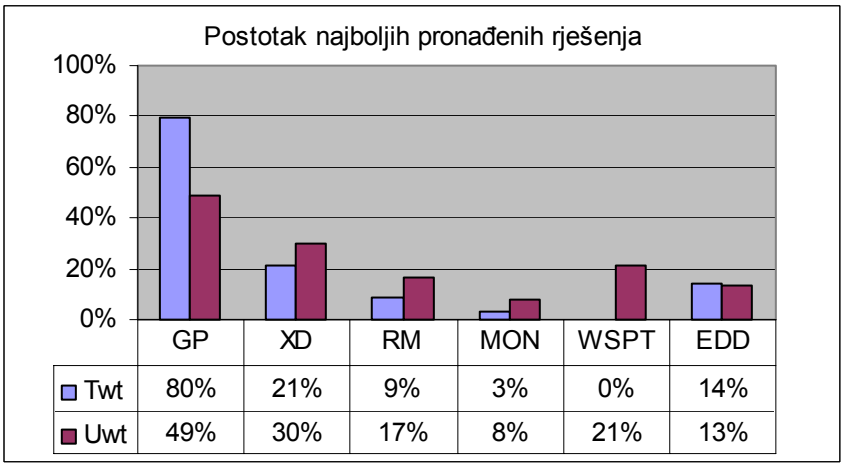
Rezultati najboljeg dobivenog rješenja za problem težinskog zaostajanja i skup ispitnih primjera za ocjenu prikazani su na slikama 4.16 i 4.17, a izraz koji opisuje izvedeno pravilo naveden je u prilogu B1.



	GP	XD	RM	MON	WSPT	EDD
Twt	330,62	389,70	451,72	623,10	844,96	1280,90
Uwt	188,82	194,09	210,64	216,69	201,56	440,03

Slika 4.16 Optimiranje težinskog zaostajanja – skup primjera za ocjenu

Iz prikaza rezultata može se vidjeti sljedeće: pravila koja nisu posebno osmišljena za dinamičku okolinu postižu lošija rješenja, a pravilo koje eksplicitno uzima u obzir informaciju o vremenu pripravnosti posla postiže bolje rezultate. Najbolji postignuti rezultat uvjerljivo je pokazalo pravilo izvedeno genetskim programiranjem, uz 15% bolju vrijednost mjerila vrednovanja nego sljedeće najbolje pravilo. Isto tako, vidljivo je da izvedeno pravilo postiže najbolju učinkovitost i na drugom kriteriju (težinsko kašnjenje), što govori u prilog tvrdnji da je težinska zakašnjelost 'isplativiji' kriterij optimiranja.



	GP	XD	RM	MON	WSPT	EDD
Twt	80%	21%	9%	3%	0%	14%
Uwt	49%	30%	17%	8%	21%	13%

Slika 4.17 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

4.4.5 Ograničenja u redoslijedu zadataka

Postoje li ograničenja u redoslijedu poslova, ona su opisana relacijom $\prec$. Izrazom $J_i \prec J_k$ definira se da posao J_i mora završiti prije pokretanja posla J_k. Ograničenja u redoslijedu poslova najčešće se prikazuju usmjerenim grafom u kojemu čvorovi odgovaraju poslovima, a lukovi definiraju relaciju ovisnosti između dva čvora. Dok je u dosadašnjim uvjetima pripravnost posla označavala i njegovu raspoloživost, pojam raspoloživosti posla u sustavu sa ograničenjima nešto se usložnjava, budući da za pokretanje nekoga posla moraju biti ispunjena dva uvjeta:

- posao J_i mora biti pripravan, odnosno $r_i \leq time$ te
- svi poslovi koji neposredno prethode poslu J_i moraju biti završeni.

U okruženju jednoga stroja, prilikom raspoređivanja dovoljno je provjeriti jesu li svi prethodnici nekoga posla pokrenuti, budući se raspoređivanje uvijek obavlja kada stroj

postane slobodan. U okruženjima sa više strojeva potrebno je provjeriti trenutnu obradu na svim strojevima u sustavu. U nekim okruženjima kao što je proizvoljna obrada, za niz operacija nekoga posla podrazumijeva se da su međusobno ovisne i podložne prethodno zadanom redoslijedu, no osim toga može postojati i ovisnost poslova koje te operacije čine.

U $\alpha|\beta|\gamma$ zapisu ova okolina ima oznaku $1|prec|*$. Najčešća mjerila vrednovanja rasporeda koja se uzimaju u obzir u ovoj okolini su također težinsko zaostajanje i težinska zakašnjelost, dok se za probleme najvećeg zaostajanja i težinskog protjecanja mogu razviti optimalni algoritmi.

Pravila raspoređivanja uz ograničenja u redoslijedu

Uz ograničenja u redoslijedu poslova mogu se koristiti i već opisana pravila, uz dodatnu provjeru koja će spriječiti raspoređivanje posla kojemu nisu već raspoređeni svi neposredni prethodnici. Naravno, budući takva pravila ne koriste informaciju o zadanim ograničenjima, takav način raspoređivanja se obično ne koristi. U sustavu sa ograničenjima, međusobne ovisnosti poslova obično su predstavljene usmjerenim grafom. Pravila raspoređivanja u ovoj okolini kao osnovnu informaciju najčešće upotrebljavaju razinu čvora u stablu (engl. *node level*). Razina čvora definirana je uz pomoć duljine puta u stablu na sljedeći način: duljina puta u stablu (engl. *lentgh of a path*) je zbroj trajanja svih čvorova na zadanom putu, a razina čvora je duljina najduljeg puta od zadanog čvora do nekog izlaznog čvora (do čvora bez sljedbenika). Razina izlaznog čvora jednaka je trajanju posla kojega čvor predstavlja.

Uz prethodne definicije, moguće je opisati sljedeća pravila raspoređivanja:

- HL (engl. *highest level*) pravilo prvo raspoređuje raspoloživi posao sa najvećom razinom (funkcija prioriteta pravila jednaka je razini čvora).
- LNS (engl. *largest number of successors*) pravilo prvo raspoređuje posao koji ima najviše neposrednih sljedbenika (funkcija prioriteta jednaka je broju podčvorova zadanog čvora).

Podrazumijeva se da u obzir za odabir dolaze samo raspoloživi poslovi, odnosno oni za koje su zadovoljena ograničenja redoslijeda. Osim navedenih, u uporabi su još neke inačice pravila, no najčešće se sreću upravo opisana pravila [Auy 03].

Za potrebe ispitivanja u ovome radu implementiran je i heuristički algoritam načinjen po uzoru na Sidneyev postupak, a kojega autori [Dha 78] nazivaju 'Sidney algoristikom'. Izraz 'algoristika' je kovanica od riječi algoritam i heuristika, a označava postupak koji pod nekim uvjetima daje optimalno rješenje (egzaktni algoritam), a pod općenitim uvjetima ponaša se kao heuristika. Navedeni postupak će se u daljem tekstu nazivati Sidney heuristika (kratica: SIDNEY). Ovaj postupak koristi pojmove *početnog skupa* (engl. *initial set*), *jednostavnog početnog skupa* (engl. *simple initial set*) i udruženih vrijednosti skupa koji su definirani na sljedeći način:

- početni skup je skup poslova za koje vrijedi da su svi njihovi prethodnici također članovi toga skupa;
- jednostavni početni skup nekoga posla sadrži zadani posao i sve njegove prethodnike.

Za bilo koji skup poslova možemo definirati udružene vrijednosti skupa (engl. *aggregate values*) kao zbroj odgovarajućih vrijednosti svih elemenata toga skupa. Npr. udruženo trajanje skupa poslova p_S jednako je zbroju trajanja svih poslova u skupu:

$$p_S = \sum_{j \in S} p_j \,. \tag{4.20}$$

Na sličan način možemo definirati udruženu težinu skupa itd. Uz prethodne definicije, Sidney heuristika prikazana je kao algoritam 4.5.

```
S = skup svih neraspoređenih poslova;
dok (S nije prazan)
{     definiraj jednostavne početne skupove za sve poslove iz S;
      A = jedn. poč. skup sa najvećim udruženim w_A/p_A ;
      ako(A ima samo jedan element)
      {   rasporedi posao iz A;
          S = skup svih neraspoređenih poslova;
      }
      inače
          S = A bez posljednjeg elementa;
}
```

Algoritam 4.5 Sidney heuristika za probleme s ograničenjima

U opisanom algoritmu, 'posljednji element' jednostavnog početnog skupa je uvijek onaj posao od kojega počinjemo graditi skup, tj. onaj koji u tome skupu nema sljedbenika. Može se dokazati [Dha 78] da ovaj algoritam daje optimalno rješenje kriterija težinskog zaostajanja za sustave u kojima su ograničenja predstavljena stablom u kojemu svaki čvor ima najviše jednog neposrednog sljedbenika (engl. *assembly tree*, *intree*). Za probleme sa općenitim oblikom stabla, algoritam na promatranom skupu primjera pronalazi rješenja čija su odstupanja unutar 0.2% od optimuma.

Raspoređivanje uz pomoć genetskog programiranja

Jedinka populacije genetskog programiranja predstavljena je stablom koje zamjenjuje funkciju prioriteta pojedinoga posla. Interpretacija prioriteta uzima u obzir samo one poslove koje je moguće rasporediti s obzirom na ograničenja, kao i kod ostalih pravila. Genetskom programu su osim već opisanih podataka na raspolaganju i podaci o razini čvora u stablu ovisnosti te broj neposrednih sljedbenika nekog posla. U tablici 4.5 prikazan je skup čvorova koji su na raspolaganju genetskom programu u gradnji jedinke.

Tablica 4.5 Popis čvorova za problem s ograničenjima na jednom stroju

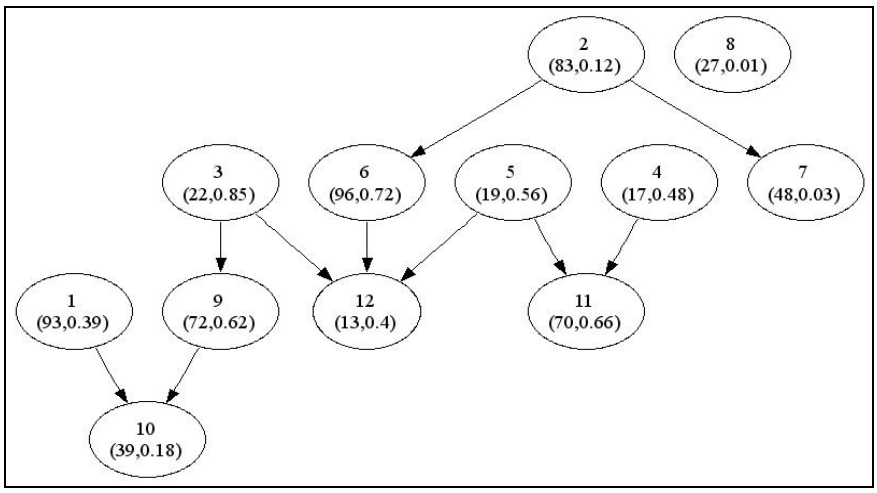
Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
N	ukupni broj poslova
Nr	preostali broj poslova (koji još nisu raspoređeni)
SP	zbroj trajanja svih poslova
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
SC	broj neposrednih sljedbenika posla
LVL	razina čvora koji predstavlja posao u grafu ovisnosti

Ispitni primjeri uz ograničenja u redoslijedu

Stupanj očekivane zakašnjelosti poslova u ovoj je okolini definiran kao i u prethodnim primjerima s pomoću niza vrijednosti parametara T i R za svaki ispitni primjer. Osim toga, za svaki primjer dodana je i informacija o zadanim međuovisnostima poslova. Ograničenja su u ispitnim primjerima oblikovana u najopćenitijem obliku (ne poprimaju oblik lanaca ili jedinstvenog stabla i sl.), što je ujedno i najteži oblik problema. Prilikom stvaranja ograničenja rabljeni su sljedeći parametri:

- prosječan udio poslova koji će imati prethodnike je 80%,
- udio poslova bez prethodnika nije manji od 20% ,
- najveći broj neposrednih prethodnika posla je 3,
- najveći broj neposrednih sljedbenika posla je 4.

Pokusi su obavljeni i sa različitim vrijednostima od navedenih, no relativni odnosi po pitanju učinkovitosti u tim slučajevima su gotovo nepromijenjeni. Ograničenja su predstavljena u obliku grafova koji su za sve ispitne slučajeve zapisani u obliku pogodnom za korištenje prilikom ispitivanja pravila raspoređivanja. Primjer grafa ovisnosti za ispitni primjer sa 12 poslova prikazan je na slici 4.18.



Slika 4.18 Prikaz grafa ovisnosti u ispitnom primjeru s 12 poslova

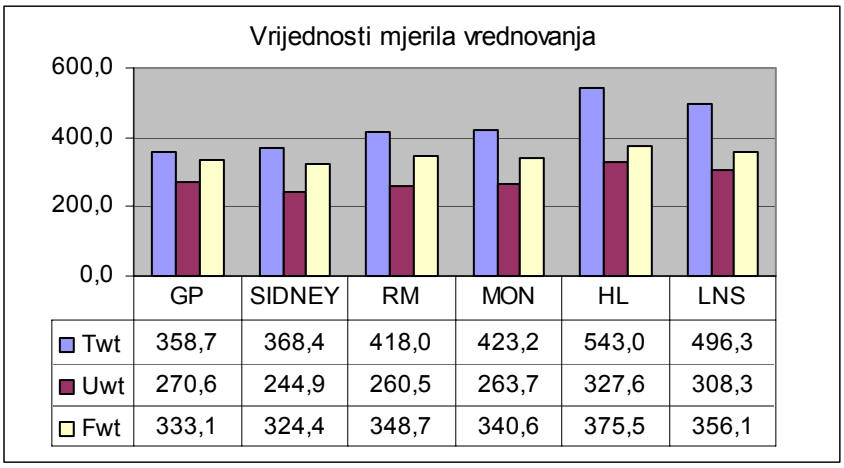
U gornjem primjeru u svaki čvor grafa upisan je redni broj posla, a u zagradi trajanje posla i njegova težina. Slično kao i u prethodnim problemima, definirano je 100 ispitnih primjera za učenje i 600 primjera za ocjenu. Zbog obima podataka, detaljni opis ograničenja za sve ispitne primjere je izostavljen.

Problem težinskog zaostajanja

Iako se u ispitnoj okolini sa ograničenjima osim težinskog zaostajanja javljaju još neki netrivijalni kriteriji, najveća se pažnja i dalje posvećuje navedenom mjerilu. Za optimiranje kriterija težinskog zaostajanja provedeno je 20 pokusa, a rezultati su pregledno prikazani na slikama 4.19 i 4.20. U postupku usporedbe uključena su i RM te MON pravila zbog relativno dobrih postignutih rezultata. U prikaz je također dodan i kriterij težinskog protjecanja (oznaka: Fwt) poradi naglaska na učinkovitost Sidney heuristike. Izraz koji predstavlja rješenje dobiveno genetskim programiranjem naveden je u prilogu B1.

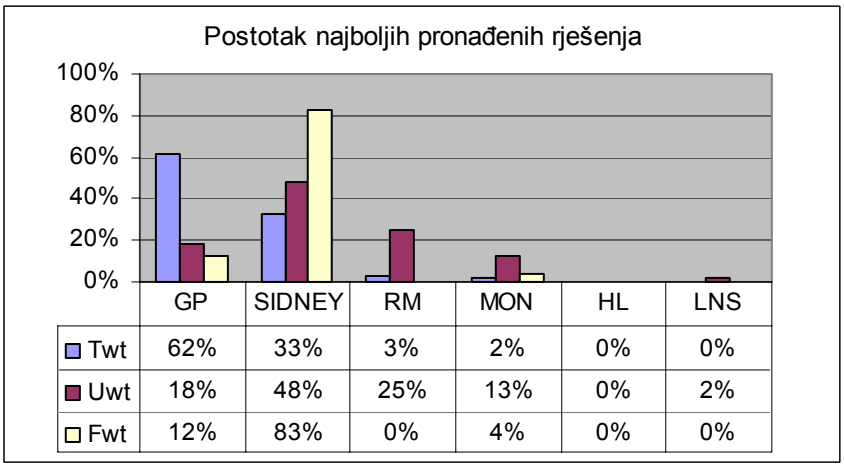
Iz prikaza se može primijetiti da su jednostavna HL i LNS pravila postigla vrlo loše vrijednosti na ispitnim primjerima. Nešto bolje su prošla pravila općenite namjene RM i MON koja ne uzimaju u obzir informaciju o ovisnosti zadataka, no svejedno postižu prihvatljive rezultate. Najbolja rješenja dobivaju se uporabom Sidney algoritma i pravila izvedenog genetskim programiranjem. Potrebno je napomenuti da Sidney algoritam

dominira nad ostalim pravilima po pitanju uspješnosti na više kriterija istovremeno, što je i očekivano budući da ovaj algoritam predstavlja složeniju proceduru od pravila raspoređivanja koje gradi rješenje u jednom prolazu. Poredbe radi, rješavanje skupa ispitnih primjera za ocjenu Sidney algoritmom traje nekoliko minuta na jednom računalu, dok se za pravila raspoređivanja rješenja dobivaju u manje od 2 sekunde.



	GP	SIDNEY	RM	MON	HL	LNS
Twt	358,7	368,4	418,0	423,2	543,0	496,3
Uwt	270,6	244,9	260,5	263,7	327,6	308,3
Fwt	333,1	324,4	348,7	340,6	375,5	356,1

Slika 4.19 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	SIDNEY	RM	MON	HL	LNS
Twt	62%	33%	3%	2%	0%	0%
Uwt	18%	48%	25%	13%	0%	2%
Fwt	12%	83%	0%	4%	0%	0%

Slika 4.20 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

4.4.6 Slijedno ovisna trajanja postavljanja

Prilikom raspoređivanja niza aktivnosti na nekom sredstvu, podrazumijeva se da je u trajanje aktivnosti uključeno i vrijeme potrebno za eventualnu prilagodbu sredstva ili aktivnosti prije početka obrade. Ukoliko je za svaki posao trajanje prilagodbe neovisno o tome koji je posao prethodno završen na stroju, to se trajanje ne odvaja od ukupnog trajanja obrade nekoga posla. Primjer ovakve situacije može biti zamjena konteksta u memoriji računala prije početka rada slijedećeg procesa, koja je obično neovisna o vrsti prethodnog procesa. Moguće je, međutim, da trajanje prilagodbe sredstva poslu (ili obratno) ovisi o tome koji je posao prethodno obavljen, u kojem se slučaju govori o slijedno ovisnim trajanjima postavljanja. Tada se za svaki mogući uređeni par poslova, od kojih jedan prethodi drugome na nekom stroju, definira količina vremena potrebna za prilagodbu. Trajanje postavljanja obično ovisi u većoj ili manjoj mjeri o trajanjima obrade poslova na koje se odnosi, no može biti i potpuno neovisno. Ova se okolina raspoređivanja prikazuje kao $1|s_{lj}|*$.

Po pitanju kriterija raspoređivanja, osim težinskog zaostajanja i protjecanja kao kriterij se može uporabiti i ukupna duljina rasporeda, budući da izmjenom redoslijeda poslova utječemo na ukupno utrošeno vrijeme (što u problemu bez trajanja postavljanja na jednom stroju nije slučaj).

Pravila raspoređivanja uz trajanja postavljanja

Problemu slijedno ovisnih trajanja postavljanja poklanja se dosta pažnje, pa se za ovu okolinu može naći nekoliko kvalitetnih pravila [Lee 04, Lee 97, Cic 01a, Mor 93]. Mogući nedostatak većine raspoloživih pravila može predstavljati to što su ona tek prilagođena okolini uz trajanja postavljanja, a ne napravljena specifično za tu namjenu. Pored samih pravila, dobiveno rješenje se u većini slučajeva može poboljšati uporabom metoda pretraživanja [Cic 03, Cha 04], no kvaliteta krajnjeg rješenja u velikoj mjeri ovisi o početnom rješenju dobivenim nekim od pravila raspoređivanja.

Gotovo sva do sada opisana pravila mogu se prilagoditi ovim uvjetima raspoređivanja na način opisan u [Mor 93]; vrijednosti prioriteta posla dobivenom primjenom originalnog pravila oduzima se određena vrijednost koja procjenjuje trošak čekanja zbog dodatnog trajanja postavljanja. Ako je izvorni prioritet posla po nekom pravilu označen sa π_j, tada se prioritet za slučaj trajanja postavljanja dobiva kao

$$\pi_{lj} = \pi_j - \frac{s_{lj}}{p_{AV} \cdot p_j} \tag{4.21}$$

gdje je *l* indeks prethodno obavljenog posla, s_{lj} trajanje postavljanja između posla *l* i posla *j*, a p_{AV} prosječno trajanje svih neraspoređenih poslova. Primjenom opisane prilagodbe, za potrebe ocjene učinkovitosti promatrana su sljedeća pravila:

- WSPT pravilo uz trajanja postavljanja, definirano sa

$$\pi_j = \frac{w_j}{p_j} - \frac{s_{lj}}{p_{AV} \cdot p_j}, \tag{4.22}$$

- MON pravilo uz trajanja postavljanja, definirano kao

$$\pi_j = \left(w_j / p_j\right) \cdot \left(1 - \frac{d_j}{\sum_{i=1}^{n} p_i}\right) - \frac{s_{lj}}{p_{AV} \cdot p_j}, \tag{4.23}$$

- RM pravilo uz trajanja postavljanja, definirano sa

$$\pi_j = \left(w_j / p_j\right) \left[\exp\left(\frac{-\left(d_j - p_j - time\right)^+}{k \cdot p_{AV}}\right)\right] - \frac{s_{lj}}{p_{AV} \cdot p_j}, \tag{4.24}$$

gdje je *l* indeks prethodno obavljenog posla za sva pravila.

Osim navedenih, uporabljeno je i ATCS pravilo (engl. *Apparent Tardiness Cost with Setups*, [Lee 97]) namijenjeno upravo okolini sa trajanjima postavljanja. ATCS pravilo definirano je sljedećim izrazom:

$$\pi_j = \frac{w_j}{p_j} \exp\left(-\frac{\left(d_j - p_j - time\right)^+}{k_1 \overline{p}} - \frac{s_{lj}}{k_2 \overline{s}}\right), \tag{4.25}$$

gdje je $\overline{p}$ prosječno trajanje svih poslova, $\overline{s}$ je prosječno trajanje postavljanja za sve parove poslova, a k_1 i k_2 su parametri heuristike. Parametri k_1 i k_2 računaju se prije početka raspoređivanja i služe za procjenu stupnja težine problema po pitanju željenih vremena završetaka i trajanja postavljanja. Autori predlažu sljedeći način računanja ovih parametara; prvo se procjenjuju vrijednosti parametra T (postotka zaostajanja) i R (područja zaostajanja):

$$R = \frac{d_{jMAX} - d_{jMIN}}{\hat{C}_{MAX}}, \quad T = 1 - \frac{d_{AVG}}{\hat{C}_{MAX}}, \tag{4.26}$$

gdje su d_{jMAX}, d_{jMIN} i d_{AVG} najveće, najmanje i srednje željeno vrijeme završetka, a $\hat{C}_{MAX}$ je procjena ukupnog trajanja koja se računa kao:

$$\hat{C}_{MAX} = \sum_{j=1}^{n} p_j + n \cdot \overline{s} \ . \tag{4.27}$$

Osim navedenoga, procjenjuje se i omjer trajanja postavljanja i trajanja obrade poslova kao:

$$\eta = \frac{\overline{s}}{\overline{p}}, \tag{4.28}$$

nakon čega se parametri k_1 i k_2 dobivaju sljedećim izrazima:

$$k_1 = \begin{cases} 4.5 + R, & \text{za } R \le 0.5 \\ 6 - 2R, & \text{za } R > 0.5 \end{cases}, \qquad k_2 = \frac{T}{2\sqrt{\eta}}. \tag{4.29}$$

Ovo se pravilo smatra najuspješnijim za probleme sa trajanjima postavljanja i gotovo se uvijek koristi u slučajevima kada neki složeniji postupak pretraživanja traži kvalitetno početno rješenje. Podrazumijeva se da se sva pravila primjenjuju kao i u ostalim statičkim uvjetima, odnosno po algoritmu 4.1.

Raspoređivanje uz pomoć genetskog programiranja

Za potrebe izvođenja funkcije prioriteta koja u obzir uzima i trajanje postavljanja, genetskom je programu za svaki ispitivani posao poznato trajanje postavljanja sa prethodnog posla. Osim toga, algoritmu je na raspolaganju i prosječno trajanje postavljanja, no ne za sve poslove već samo srednje trajanje postavljanja sa prethodnog na sve ostale poslove. U tablici 4.6 prikazan je skup čvorova koji su na raspolaganju genetskom programu u gradnji pojedinog rješenja.

Tablica 4.6 Popis čvorova uz trajanja postavljanja na jednom stroju

Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)

Nr	preostali broj poslova (koji još nisu raspoređeni)
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
STP	trajanje postavljanja sa prethodnog na promatrani posao, s_{lj}
Sav	prosječno trajanje postavljanja sa prethodnog (l) na sve ostale poslove, $\frac{1}{n-1}\sum_{j=1}^{n} s_{lj}$

Ispitni primjeri uz trajanja postavljanja

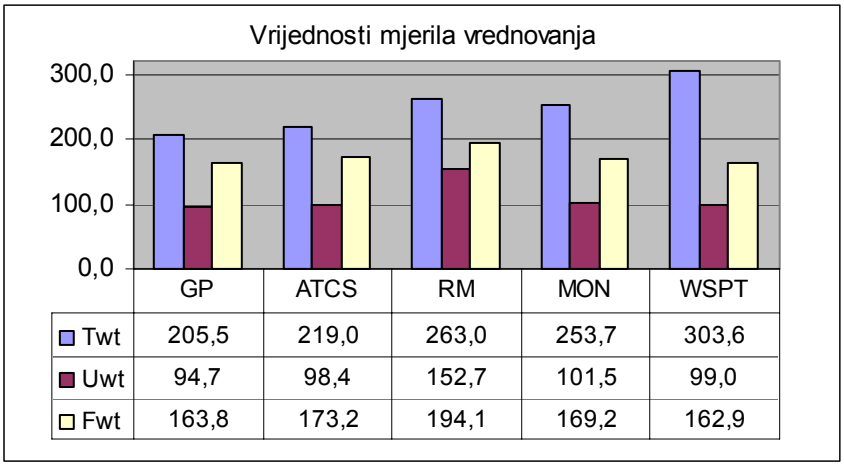
Prilikom definiranja ispitnih primjera također su rabljeni parametri T i R za određivanje stupnja i područja zakašnjelosti poslova kao u izrazu (4.11). Pored tih podataka, potrebno je generirati trajanja postavljanja za svaku moguću kombinaciju prethodnog i trenutnog posla, što za jedan ispitni primjer čini matricu veličine $n \times n$. U određenim se situacijama može pretpostaviti da je trajanje postavljanja između dva posla neovisno o njihovom poretku, tj. da je matrica trajanja simetrična, no ovdje će se promatrati općeniti slučaj.

Količina podataka koja bi bila potrebna za zapisivanje trajanja postavljanja za svaki ispitni primjer je neprikladno velika (npr. za 600 primjera za ocjenu, uz prosječno 50 poslova, bilo bi potrebno zapisati 50 x 50 x 600 različitih vrijednosti). Zbog toga se za svaki ispitni primjer trajanja postavljanja definiraju tijekom rada programa, ali tako da se uvijek dobivaju jednake vrijednosti (radi poredbe različitih metoda). U programskom ostvarenju koristi se ugrađeni generator slučajnih brojeva na način da se za svaki ispitni primjer početna vrijednost generatora postavi na redni broj dotičnog ispitnog primjera. Trajanja postavljanja određuju se po jednolikoj raspodjeli uz dodatni parametar η koji predstavlja omjer prosječnog trajanja postavljanja i prosječnog trajanja obrade posla, kao u izrazu (4.28). Za potrebe ispitivanja u ovome radu vrijednost toga parametra postavljena je na 0.5 u primjerima za učenje te 0.5 i 1 u primjerima za ocjenu. Parametri za sastavljanje ispitnih primjera u ovoj okolini navedeni su u prilogu A1.

Problem težinskog zaostajanja

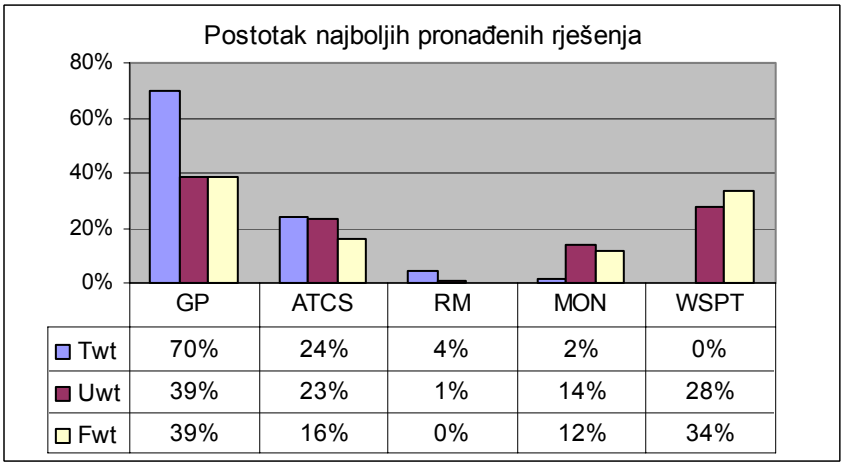
U postupku izvođenja pravila raspoređivanja za problem težinskog zaostajanja provedeno je 20 pokusa uz pretpostavljene vrijednosti parametara. Za potrebe ocjene učinkovitosti prikazani su rezultati pravila ATCS, RM, MON i WSPT, gdje su potonja 4 pravila prilagođena prema izrazu (4.21). Osim težinskog zaostajanja i težinske zakašnjelosti, uvršten je i kriterij težinskog protjecanja budući je uz ove uvjete raspoređivanja također netrivijalan (moguće je prikazati i ukupno trajanje rasporeda, no nijedna od promatranih metoda, s izuzetkom WSPT pravila, nije namijenjena optimiranju toga kriterija).

Na temelju učinka izvedenog pravila na skupu ispitnih primjera za ocjenu, odabrano pravilo navedeno je u prilogu B1. U postupku ocjenjivanja pravila dana su dva prikaza rezultata; za jedan od prikaza vrijednost parametra η postavljena je na 0.5, kao i u primjerima za učenje, a za drugi je vrijednost postavljena na 1 (odnosno, promijenjen je omjer prosječnog trajanja postavljanja i trajanja obrade). Na taj se način može ispitati koliko je izvedeno pravilo osjetljivo s obzirom na promjene prosječnog trajanja postavljanja. Na slikama 4.21 i 4.22 prikazani su rezultati usporedbe uz $\eta = 0.5$.



	GP	ATCS	RM	MON	WSPT
Twt	205,5	219,0	263,0	253,7	303,6
Uwt	94,7	98,4	152,7	101,5	99,0
Fwt	163,8	173,2	194,1	169,2	162,9

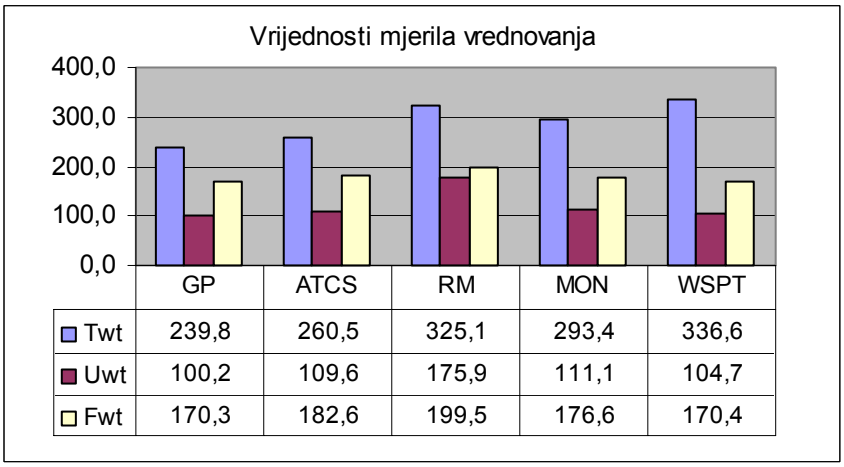
Slika 4.21 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	ATCS	RM	MON	WSPT
Twt	70%	24%	4%	2%	0%
Uwt	39%	23%	1%	14%	28%
Fwt	39%	16%	0%	12%	34%

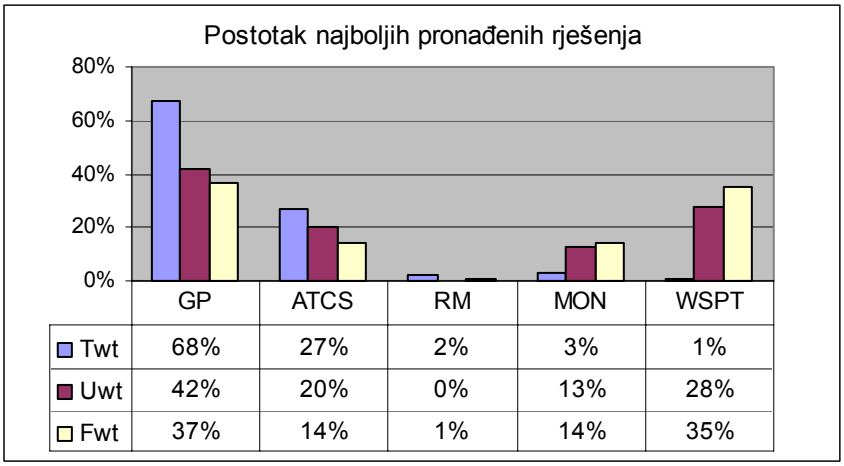
Slika 4.22 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Iz prikaza je vidljivo da pravilo izvedeno genetskim programiranjem pronalazi najbolje rješenje u većini slučajeva, dok ATCS pravilo prednjači nad svim ostalim promatranim pravilima. Na slikama 4.23 i 4.24 prikazani su rezultati uz povećano prosječno trajanje postavljanja, odnosno $\eta = 1$.



	GP	ATCS	RM	MON	WSPT
Twt	239,8	260,5	325,1	293,4	336,6
Uwt	100,2	109,6	175,9	111,1	104,7
Fwt	170,3	182,6	199,5	176,6	170,4

Slika 4.23 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	ATCS	RM	MON	WSPT
Twt	68%	27%	2%	3%	1%
Uwt	42%	20%	0%	13%	28%
Fwt	37%	14%	1%	14%	35%

Slika 4.24 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Iz rezultata se može zaključiti da relativna učinkovitost pravila ne ovisi u velikoj mjeri o prosječnom trajanju postavljanja, iako se u primjeni mogu pojaviti i neki primjeri s ekstremnijim vrijednostima.

4.4.7 Pravila raspoređivanja u složenijim okolinama

U prethodnim odjeljcima, učinkovitost pravila raspoređivanja izvedenog genetskim programiranjem prikazana je u okolinama za koje već postoji relativno velik broj raspoloživih pravila. Opisani ispitni uvjeti predstavljaju najčešće promatrane probleme raspoređivanja na jednom stroju, a rezultati postignuti na tom okruženju mogu se pokazati korisnima u postupcima raspoređivanja složenijih okruženja sa više strojeva. Jedan od ciljeva ovoga rada je pokazati da se i u takvim okolinama mogu izvesti pravila čija je učinkovitost barem toliko dobra kao i učinkovitost postojećih metoda. No još važnije područje primjene izvođenja pravila raspoređivanja su okoline za koje postoji malo prikladnih pravila, a pitanje odabira 'najboljeg' pravila predstavlja dodatni problem. Upravo je u tim uvjetima isplativa uporaba automatiziranog stvaranja pravila prilagođenog promatranom sustavu i danim okolnostima.

U ovom odjeljku prikazat će se izvođenje pravila raspoređivanja za neke manje zastupljene okoline raspoređivanja na jednom stroju. U literaturi se može pronaći velik broj različitih problema raspoređivanja koji se, algoritamski gledano, svode na neka od prethodno opisanih okruženja ili njihove kombinacije. Budući postoji velik broj mogućih varijacija jednog oblika problema, ovdje će se obraditi samo manji broj primjera.

Raspoređivanje uz trajanja postavljanja i ograničenja u redoslijedu

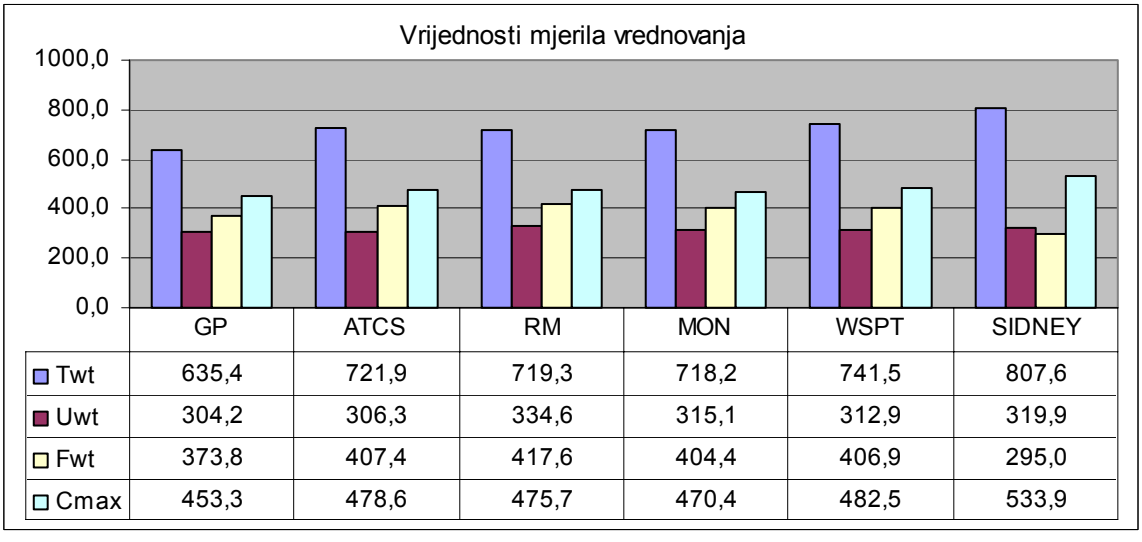
Postojanje slijedno ovisnih trajanja postavljanja može uključivati i istovremeno postojanje ograničenja u redoslijedu izvođenja poslova. Ova okolina raspoređivanja predstavlja dodatni problem jer se u obzir moraju uzimati ograničenja i istovremeno minimizirati troškovi uslijed izmjene poslova. Dok se za slučaj trajanja postavljanja gotovo svako pravilo može prilagoditi postupkom pokazanim u odjeljku 4.4.6, informaciju o ograničenjima je teže uklopiti u neko pravilo. Naravno, uvijek se može postići očuvanje zadanog rasporeda, no za učinkovito rješenje potrebno je na neki način iskoristiti strukturu danih ograničenja. Za potrebe ispitivanja, na ovoj okolini su primijenjena pravila ATCS, RM, MON, WSPT (posljednja tri prilagođena za trajanja postavljanja) te Sidney heuristika. Pravila HL, LNS i EDD su postigla samo značajno lošije rezultate pa nisu niti prikazana u poredbi. Za potrebe definiranja ispitnih primjera rabljena je metodologija opisana u odjeljcima uz ograničenja i trajanja postavljanja (prilikom određivanja trajanja postavljanja ignorira se činjenica da neke kombinacije parova prethodnog i trenutnog

posla nisu moguće zbog dodanih ograničenja). Izvođenje pravila genetskim programiranjem odvija se na jednak način kao i u spomenutim odjeljcima, s tim da se skup čvorova u ovom primjeru dobiva unijom skupova za pojedine okoline. Slijedeći ovu strategiju, ukupan skup mogućih čvorova u stablu rješenja prikazan je u tablici 4.7.

Tablica 4.7 Popis čvorova uz trajanja postavljanja i ograničenja u redoslijedu

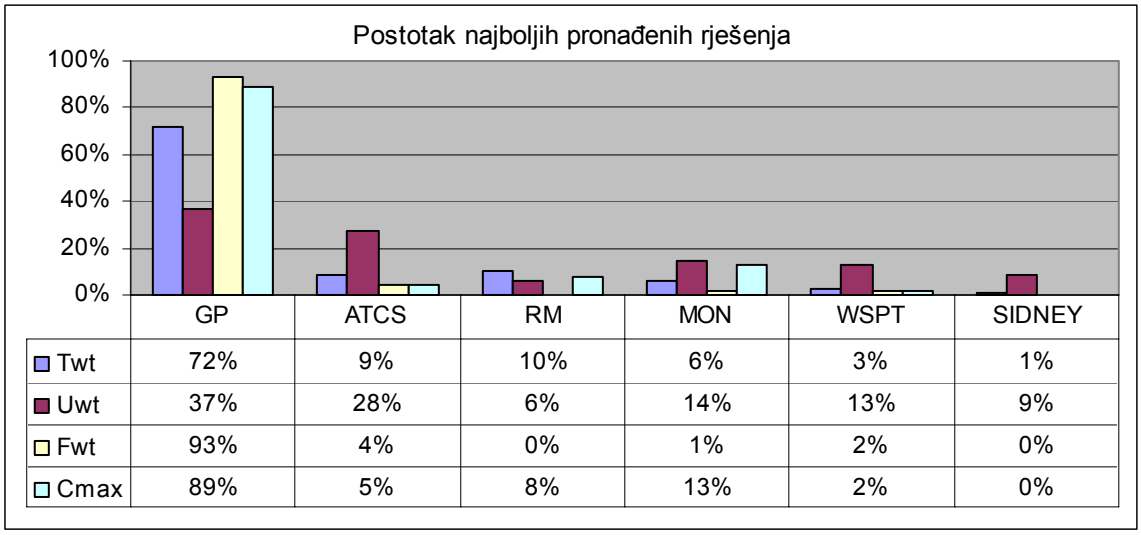
Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
Nr	preostali broj poslova (koji još nisu raspoređeni)
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
STP	trajanje postavljanja s prethodnog na promatrani posao, s_{lj}
Sav	prosječno trajanje postavljanja sa prethodnog (l) na sve ostale poslove, $\frac{1}{n-1}\sum_{j=1}^{n} s_{lj}$
SC	broj neposrednih sljedbenika posla
LVL	razina čvora koji predstavlja posao u grafu ovisnosti

U postupku izvođenja pravila provedeno je 10 pokusa uz pretpostavljene parametre evolucijskog procesa. Proces genetskog programiranja i dalje se temelji na optimiranju težinskog zaostajanja, budući se to mjerilo pokazalo isplativim u smislu učinkovitosti sa gledišta više kriterija. Pravilo izvedeno genetskim programiranjem dano je u prilogu B1. Prilikom prikaza rezultata uveden je i kriterij ukupne duljine rasporeda, definiran po izrazu (4.4), uz kraticu 'Cmax'. Na slikama 4.25 i 4.26 prikazani su usporedni rezultati za 600 ispitnih primjera za ocjenu.



	GP	ATCS	RM	MON	WSPT	SIDNEY
Twt	635,4	721,9	719,3	718,2	741,5	807,6
Uwt	304,2	306,3	334,6	315,1	312,9	319,9
Fwt	373,8	407,4	417,6	404,4	406,9	295,0
Cmax	453,3	478,6	475,7	470,4	482,5	533,9

Slika 4.25 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	ATCS	RM	MON	WSPT	SIDNEY
Twt	72%	9%	10%	6%	3%	1%
Uwt	37%	28%	6%	14%	13%	9%
Fwt	93%	4%	0%	1%	2%	0%
Cmax	89%	5%	8%	13%	2%	0%

Slika 4.26 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Pravilo dobiveno genetskim programiranjem pokazuje bolje rezultate za sve kriterije osim za težinsko protjecanje, gdje najbolji rezultat daje Sidney heuristika. Iako je postotak dominacije genetskog programa veći od dominacije za Sidney heuristiku, očito postoji određen broj primjera gdje je izvedeno pravilo postiglo značajno lošiji rezultat po pitanju težinskog protjecanja. S druge strane, gledamo li težinsko zaostajanje ili ukupnu duljinu rasporeda, vidljiva je dominacija izvedenog pravila.

Raspoređivanje u dinamičkoj okolini uz trajanja postavljanja

Većina opisanih ispitnih okolina, kao i većina ispitnih primjera iz literature, pretpostavlja statičku okolinu raspoređivanja, gdje su svi poslovi pripravni od početka rada sustava. Za razliku od toga, dinamička okolina uključuje i dodatni uvjet u obliku vremena pripravnosti poslova, što u velikom broju slučajeva vjernije prikazuje stvarne uvjete raspoređivanja. Ukoliko u sustavu dopuštamo dinamičke dolaske poslova, postavlja se pitanje kada je moguće započeti proces postavljanja nekoga posla na stroju. U ovakvim slučajevima teorija raspoređivanja ne daje konkretne odgovore, pa će se u ovom radu pretpostaviti da postavljanje stroja za neki posao ne može započeti prije vremena pripravnosti toga posla (ova odluka je u skladu sa većinom sustava raspoređivanja [Lek 03]).

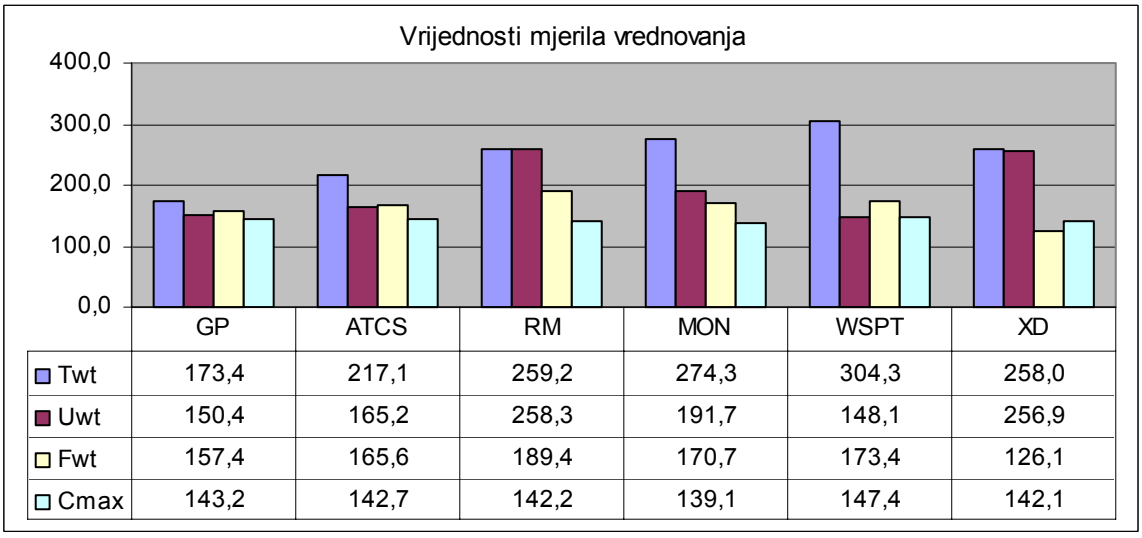
Promatrani postupci raspoređivanja u ovoj okolini su ATCS, MON, RM, XD i WSPT pravilo, od kojih su posljednja četiri prilagođena za trajanja postavljanja. Ispitni primjeri su oblikovani kao u odjeljku 4.4.4, uz dodatno generiranje slučajnih vrijednosti trajanja postavljanja kao u odjeljku 4.4.6. Slično prethodnom primjeru, skup mogućih čvorova koji čine stablo jedinke dobiven je unijom elemenata skupova za dinamički problem i za problem slijedno ovisnih trajanja postavljanja, a definiran je tablicom 4.8. U postupku interpretacije pravila, genetsko programiranje koristi opisani algoritam 4.4.

Tablica 4.8 Popis čvorova za dinamički problem uz trajanja postavljanja

Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)

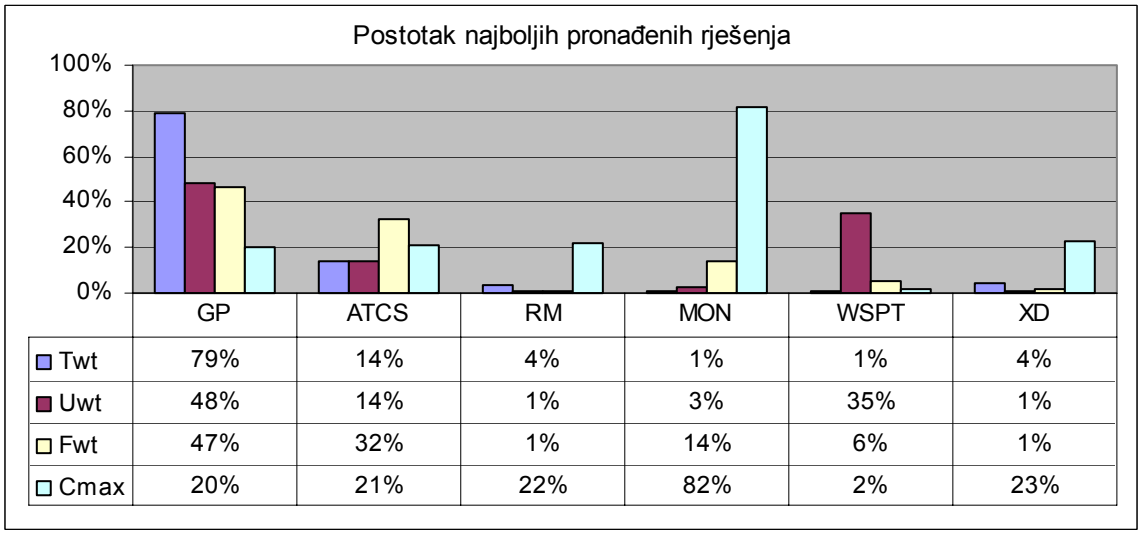
Nr	preostali broj poslova (koji još nisu raspoređeni)
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
STP	trajanje postavljanja sa prethodnog na promatrani posao, s_{lj}
Sav	prosječno trajanje postavljanja sa prethodnog (l) na sve ostale poslove, $\frac{1}{n-1}\sum_{j=1}^{n} s_{lj}$
AR	vrijeme do pripravnosti posla, $\max\{r_j - time, 0\}$

Kao kriterij optimiranja uporabljeno je težinsko zaostajanje, te je uz pretpostavljene parametre provedeno 20 pokusa. Odabrano rješenje za ovu okolinu navedeno je u prilogu B1, a na slikama 4.27 i 4.28 mogu se vidjeti rezultati usporedbe na ispitnim primjerima za ocjenu.



	GP	ATCS	RM	MON	WSPT	XD
Twt	173,4	217,1	259,2	274,3	304,3	258,0
Uwt	150,4	165,2	258,3	191,7	148,1	256,9
Fwt	157,4	165,6	189,4	170,7	173,4	126,1
Cmax	143,2	142,7	142,2	139,1	147,4	142,1

Slika 4.27 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	ATCS	RM	MON	WSPT	XD
Twt	79%	14%	4%	1%	1%	4%
Uwt	48%	14%	1%	3%	35%	1%
Fwt	47%	32%	1%	14%	6%	1%
Cmax	20%	21%	22%	82%	2%	23%

Slika 4.28 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Iz rezultata se može primijetiti da su različita pravila postigla najbolji rezultat za pojedine kriterije: WSPT pravilo daje najbolju vrijednost za težinsku zakašnjelost ('Uwt'), XD pravilo za težinsko protjecanje ('Fwt'), MON pravilo za ukupnu duljinu rasporeda ('Cmax'), a pravilo dobiveno genetskim programiranjem za osnovni kriterij težinskog zaostajanja. Promatramo li stupanj dominacije pravila, izvedeno pravilo prednjači nad ostalima u broju najboljih rješenja, osim za kriterij ukupne duljine rasporeda, gdje uvjerljivo najbolji postotak postiže MON pravilo.

Raspoređivanje u dinamičkoj okolini uz ograničenja u redoslijedu

Kao posljednji primjer u ovom odjeljku promatra se problem raspoređivanja s ograničenjima u redoslijedu uz dinamičke dolaske poslova. Iako se na ovakav primjer rijetko nailazi u stvarnim uvjetima, sasvim je moguće zamisliti i takvu okolinu raspoređivanja; npr. rad sustava može uključivati obradu podataka sa vanjskih mjernih jedinica, s time da se pojedini dijelovi izračuna moraju obaviti prije nekih drugih, a podaci u sustav dolaze u različitim vremenskim trenucima.

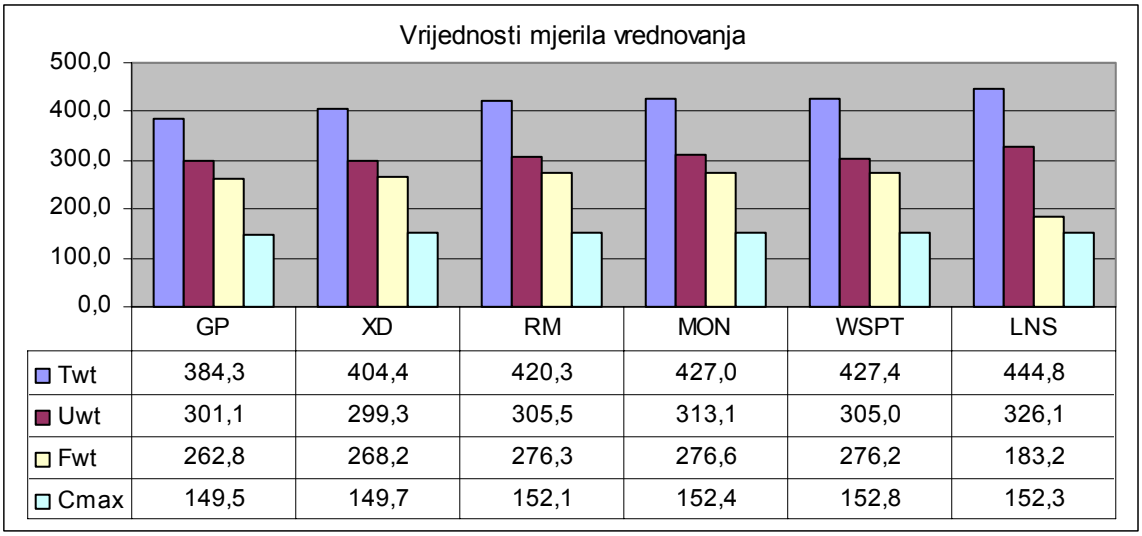
Ispitni primjeri su za ovu okolinu generirani uz proizvoljnu ovisnost poslova te uz proizvoljna vremena pripravnosti po izrazu (4.18). Uz ovakav način definiranja ispitnih primjera, moguće je neslaganje vremena pripravnosti i zadanog redoslijeda poslova (tj. posao prethodnik može imati kasnije vrijeme pripravnosti od posla sljedbenika). Ovakav pristup u svakom slučaju usložnjava postupak raspoređivanja, jer algoritam mora uzeti u obzir i informaciju o dolasku i o poretku poslova kako bi donio što bolju odluku. Takvi algoritmi su vrlo rijetki, pa je smisleno s pomoću postupka učenja oblikovati pravilo prilagođeno zadanim uvjetima. Promatrana pravila raspoređivanja u ovom primjeru su XD, RM, MON, WSPT i LNS pravilo. Ostala pravila nisu uvrštena u rezultate jer daju znatno lošija rješenja – Sidney heuristika, na primjer, budući podrazumijeva statičku raspoloživost poslova, postiže i dvostruko lošije vrijednosti od navedenih pravila.

Izvođenje pravila genetskim programiranjem odvija se na način jednak već opisanome, uz skup mogućih čvorova stabla prikazan u tablici 4.9.

Tablica 4.9 Popis čvorova za dinamički problem uz ograničenja u redoslijedu

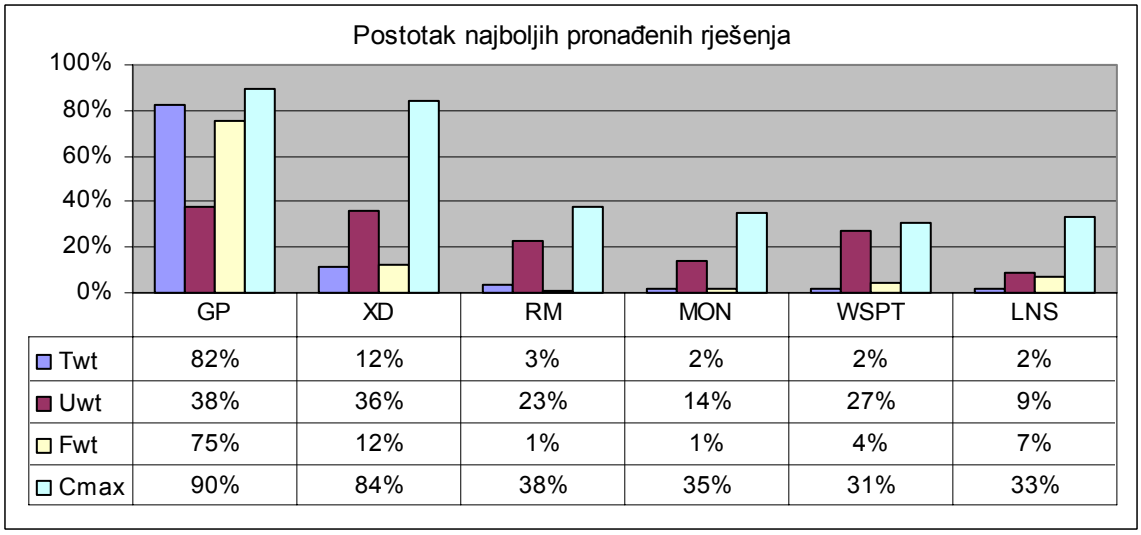
Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
Nr	preostali broj poslova (koji još nisu raspoređeni)
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
AR	vrijeme do pripravnosti posla, $\max\{r_j - time, 0\}$
SC	broj neposrednih sljedbenika posla
LVL	razina čvora koji predstavlja posao u grafu ovisnosti

Za potrebe optimiranja opisane okoline provedeno je 10 pokusa, od kojih je najbolje rješenje prikazano u prilogu B1. Usporedni rezultati za ispitne primjere za ocjenu prikazani su na slikama 4.29 i 4.30.



	GP	XD	RM	MON	WSPT	LNS
Twt	384,3	404,4	420,3	427,0	427,4	444,8
Uwt	301,1	299,3	305,5	313,1	305,0	326,1
Fwt	262,8	268,2	276,3	276,6	276,2	183,2
Cmax	149,5	149,7	152,1	152,4	152,8	152,3

Slika 4.29 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	XD	RM	MON	WSPT	LNS
Twt	82%	12%	3%	2%	2%	2%
Uwt	38%	36%	23%	14%	27%	9%
Fwt	75%	12%	1%	1%	4%	7%
Cmax	90%	84%	38%	35%	31%	33%

Slika 4.30 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

U ovom slučaju rješenje dobiveno genetskim programiranjem dominira nad ostalim pravilima za sve promatrane kriterije, mada je postupak učenja uvažavao samo težinsko zaostajanje. Približno jednak rezultat za težinsku zakašnjelost i ukupnu duljinu rasporeda postiže XD pravilo, a iz postotaka dominacije može se vidjeti da oba postupka u velikom broju slučajeva postižu jednaku vrijednost po pitanju ukupne duljine rasporeda.

4.5 Raspoređivanje na paralelnim jednolikim strojevima

4.5.1 Okruženje paralelnih strojeva

U klasičnoj teoriji raspoređivanja postoje tri glavne vrste okruženja raspoređivanja na paralelnim strojevima: identični, jednoliki i nesrodni strojevi. Okruženje nesrodnih strojeva najopćenitije je od ova tri i obrađuje se u posebnom poglavlju. Problem raspoređivanja na identičnim strojevima može se smatrati specijalnim slučajem raspoređivanja na jednolikim strojevima u kojemu su brzine svih strojeva jednake. Često se i postupci raspoređivanja u literaturi navode zajednički za ta dva okruženja. Iz toga se razloga u ovom radu razmatraju samo primjeri za okruženje jednolikih strojeva, dok su identični strojevi izostavljeni.

Za sve promatrane inačice raspoređivanja na paralelnim jednolikim strojevima vrijede sljedeće pretpostavke:

- svi podaci o poslovima unaprijed su poznati (predodređeno raspoređivanje),
- raspoređivanje se obavlja dinamički (tijekom rada sustava),
- poslovi su neprekidivi,
- strojevi su neprekidno raspoloživi od početka rada sustava.

Pretpostavka o mogućnosti korištenja strojeva od početka rada označava statičku raspoloživost strojeva. Iako se u ovom radu ne promatra dinamička raspoloživost strojeva, odnosno situacija kada su vremena pripravnosti strojeva različita od nule, sva se ispitivana pravila mogu vrlo jednostavno prilagoditi ovoj promjeni, budući se u postupku raspoređivanja ionako čeka da neki stroj postane slobodan prije donošenja odluke.

Za postupke raspoređivanja na paralelnim identičnim ili jednolikim strojevima sa dinamičkom raspoloživošću i uz bilo koji pravilni kriterij (odjeljak 2.2.4), problem se uvijek može svesti na problem sa samo jednim strojem. Može se dokazati [Mor 93] da je optimalno rješenje moguće dobiti na sljedeći način: poredati sve poslove u 'odgovarajući' niz te rasporediti redom svaki posao iz niza na stroj koji najprije može završiti promatrani posao. Ovaj postupak je moguć budući se svaki posao može obaviti na bilo kojem stroju. Naravno, najveći problem je upravo pronalaženje 'odgovarajućeg' redoslijeda poslova koji bi na ovaj način raspoređivanja dao i optimalno konačno rješenje.

Kako se ovaj oblik okruženja može svesti na problem uređivanja poslova u niz, moguće je bez puno promjena za raspoređivanje iskoristiti pravila namijenjena raspoređivanju na jednom stroju. Uputno je, međutim, pravilu raspoređivanja dati dodatne informacije o svojstvima okoline. U okruženju jednolikih strojeva postoji nekoliko netrivijalnih kriterija ocjene rasporeda, no zbog opsega istraživanja pokusi se i dalje temelje na kriteriju težinskog zaostajanja. U ostatku poglavlja prikazan je način definiranja ispitnih primjera za okruženje jednolikih strojeva te nekoliko primjera raspoređivanja s pomoću pravila u ovom okruženju.

Raspoređivanje za paralelne strojeve uz pomoć genetskog programiranja

Metodologija izvođenja pravila raspoređivanja za paralelne jednolike (i nesrodne) strojeve uz pomoć genetskog programiranja do sada, po svim dostupnim podacima, nije prikazana u literaturi. Jedini dodir genetskog programiranja i raspoređivanja za paralelne strojeve izložen je u [Ser 99] i [Ser 01]. U tim se radovima raspoređivanje obavlja uz pomoć genetskog algoritma, no kriterij ocjene rasporeda nije globalni, odnosno ne ocjenjuje se na razini cjelokupnog rasporeda, već je za svaki posao, kojemu je pridružen jedan agent, definiran lokalni kriterij uz pomoć kojega agent (uz uporabu GA) ocjenjuje svoj posao. Uloga genetskog programiranja u ovoj okolini je izvođenje lokalne funkcije cilja koja se koristi u ocjeni ponašanja agenta. Sâmo raspoređivanje obavlja se postupkom pretrage prostora rješenja (uz pomoć genetskog algoritma), a prednost izvedbe je u raspodijeljenoj kontroli rada sustava.

4.5.2 Oblikovanje ispitnih primjera

Tvorba ispitnih primjera za okruženje jednolikih strojeva temelji se na postupcima prikazanim u prethodnom poglavlju, no uz neke dodatne elemente. Broj poslova u ispitnim primjerima za učenje može biti 12, 25, 50 i 100, a u primjerima za ocjenu 25, 50 i 100. Nominalne vrijednosti trajanja poslova dobivaju se opisanim raspodjelama kao cjelobrojne vrijednosti u opsegu od 1 do 100. Težine poslova također poprimaju vrijednosti od 0.01 do 1 u koracima od 0.01. Za svaki ispitni primjer definira se i broj strojeva, koji za primjere za učenje poprima vrijednosti 3, 6 i 10, a u primjerima za ocjenu 3, 6, 10, 15 i 20.

Pored nominalnog trajanja poslova, potrebno je za svaki stroj *i* definirati njegovu brzinu s_i. Trajanje obrade posla j na stroju *i* iznosi

$$p_{ij} = p_j / s_i \,. \tag{4.30}$$

Brzine strojeva se za sve ispitne primjere definiraju na jednak način: definira se slučajna varijabla *spd* koja za svaki stroj poprima vrijednosti u intervalu $[0.1,1]$ u koracima od 0.01 po jednolikoj raspodjeli. Brzina stroja se tada računa kao:

$$s_i = \frac{1}{spd} \tag{4.31}$$

Ovako definirane brzine strojeva poprimaju vrijednosti od 1 do 10, ali uz veće grupiranje kod manjih vrijednosti brzina. Na temelju brzina strojeva moguće je definirati *efektivni broj strojeva* $\hat{m}$ kao zbroj svih njihovih brzina:

$$\hat{m} = \sum_{i=1}^{m} s_i \,, \tag{4.32}$$

gdje je m broj strojeva u dotičnom ispitnom primjeru. Efektivni broj strojeva može se smatrati brzinom jednoga sredstva koje zamjenjuje sve strojeve u sustavu. Uz pomoć efektivnog broja strojeva definiramo očekivano ukupno trajanje poslova $\hat{p}$ kao

$$\hat{p} = \frac{1}{\hat{m}} \sum_{j=1}^{n} p_j \tag{4.33}$$

Vremena pripravnosti poslova generiraju se po jednolikoj raspodjeli u intervalu

$$r_j \in \left[0, \frac{\hat{p}}{2} \right] \tag{4.34}$$

Za definiranje željenih vremena završetaka također se koriste parametri *T* i *R*, no umjesto zbroja nominalnih trajanja poslova, koristi se očekivano ukupno trajanje poslova, pa u statičkoj okolini željena vremena završetka poprimaju vrijednosti u intervalu

$$d_j \in \left[\hat{p}\left(1 - T - R/2\right), \hat{p}\left(1 - T + R/2\right) \right], \tag{4.35}$$

a u dinamičkoj okolini u intervalu

$$d_j \in \left[r_j + \left(\hat{p} - r_j\right) \cdot \left(1 - T - R/2\right), r_j + \left(\hat{p} - r_j\right) \cdot \left(1 - T + R/2\right) \right]. \tag{4.36}$$

Eventualna trajanja postavljanja poslova definiraju se na način jednak opisanom u odjeljku 4.4.6 i ne ovise o brzini stroja na kojemu se postavljanje odvija. Detaljan popis parametara za tvorbu ispitnih primjera za učenje i ocjenu naveden je u prilogu A2.

Definiranje funkcije cilja

U poglavlju 0 općenito je opisan način definiranja funkcije cilja za pojedine ispitne primjere i ujedno dobrote jedinke u genetskom programiranju. Budući se u okruženju sa više strojeva mijenja pojam trajanja obrade posla, funkcija cilja se računa kao u izrazima (4.1) – (4.4), s time da se srednje trajanje poslova $\overline{p}$ računa kao kvocijent očekivanog ukupnog trajanja i broja poslova, odnosno

$$\overline{p} = \frac{\hat{p}}{n} \,. \tag{4.37}$$

4.5.3 Raspoređivanje uz pomoć pravila na jednolikim strojevima

Budući je pravilo raspoređivanja u užem smislu definirano samo funkcijom prioriteta poslova, za svako okruženje potrebno je definirati odgovarajući način primjene pravila. Za okruženje jednolikih strojeva, sva promatrana pravila, uključujući i pravilo izvedeno genetskim programiranjem, primjenjuju se po postupku prikazanom kao algoritam 4.6.

```
dok (postoje neraspoređeni poslovi)
{     čekaj dok neki stroj (k) i barem jedan posao ne postanu
raspoloživi;
      odredi prioritete svih raspoloživih poslova na stroju k;
      rasporedi posao s najboljim prioritetom na stroj k;
}
```

Algoritam 4.6 Postupak raspoređivanja po pravilima na jednolikim strojevima

Ukoliko su u istom trenutku raspoloživa dva ili više strojeva, raspoređivanje se obavlja slijedno na jednom po jednom stroju (drugim riječima, ne obavlja se usporedba prioriteta nekoga posla na više od jednoga stroja istovremeno). Ovakav je pristup opravdan budući se svi poslovi mogu izvoditi na svim strojevima, a brzine strojeva su proporcionalne pa nema potrebe za određeni posao tražiti 'najbolji' stroj. U okruženju nesrodnih strojeva ova tvrdnja ne vrijedi, pa se za to okruženje primjenjuje drugačiji postupak, o čemu više u poglavlju 4.6.

Pravila raspoređivanja definirana u poglavlju o raspoređivanju na jednom stroju ovdje se mogu iskoristiti u neizmijenjenom obliku. Ukoliko je dopuštena dinamička raspoloživost poslova, trajanje obrade posla se također uvećava za količinu vremena do trenutka dolaska posla kako je opisano u odjeljku 4.4.4. Isto tako, u slučaju postojanja trajanja postavljanja određeno pravilo moguće je prilagoditi na način opisan u odjeljku 4.4.6 po izrazu (4.21). Na taj se način u ovom poglavlju upotrebljavaju pravila WSPT, MON, ATCS i EDD. Osim navedenih, uvršteno je i LPT (engl. *longest processing time*) pravilo čija je funkcija prioriteta definirana kao

$$\pi_j = p_j . \tag{4.38}$$

LPT pravilo uvijek odabire najdulji raspoloživi posao, što u ovom okruženju vodi do dobrih rezultata po pitanju ukupne duljine rasporeda.

Promatrana su i pravila definirana posebno za ovo okruženje, iako su po osnovnom obliku najčešće rabljena u okruženju jednoga stroja. Toj skupini pripadaju sljedeća pravila:

- mRM (engl. *modified R&M*) pravilo, definirano sa:

$$\pi_j = \frac{w_j}{\left(p_j/\hat{m}\right)}\left[\exp\left(\frac{-\left(d_j - p_j/s_k - time\right)^+}{\left(p_{AV}/\hat{m}\right)}\right)\right]; \tag{4.39}$$

- mXD (engl. *modified X-dispatch*) pravilo za dinamičku okolinu, definirano sa:

$$\pi_j = \frac{w_j}{\left(p_j/\hat{m}\right)}\left[\exp\left(\frac{-\left(d_j - \left(p_j/s_k\right) - time\right)^+}{\left(p_{AV}/\hat{m}\right)}\right)\right]\cdot\left(1 - B\frac{\left(r_j - time\right)^+}{\left(p_{AV}/\hat{m}\right)}\right), \tag{4.40}$$

gdje je k indeks stroja za koji se računaju prioriteti poslova. Važno je naglasiti bitnu razliku u primjeni ova dva i ostalih pravila: u potonja dva pravila u izrazu se javlja brzina određenog stroja, te je prioritet koji pravilo dodjeljuje poslovima ovisan o tome na kojem se stroju raspoređivanje trenutno obavlja. Također se u slučaju postojanja trajanja postavljanja ova pravila ne prilagođuju na način jednak ostalima. Budući se u definiciji funkcije prioriteta ovih pravila javlja efektivni broj strojeva $\hat{m}$, prilagodba danih pravila za trajanja postavljanja definirana je izrazom

$$\pi_{lj} = \pi_j - \frac{s_{lj}}{\left(p_{AV}/\hat{m}\right)\cdot\left(p_j/s_k\right)}, \tag{4.41}$$

gdje je l indeks prethodnog posla (na stroju k), s_k je brzina stroja a s_{lj} je trajanje postavljanja sa prethodnog na promatrani posao (neovisno o stroju).

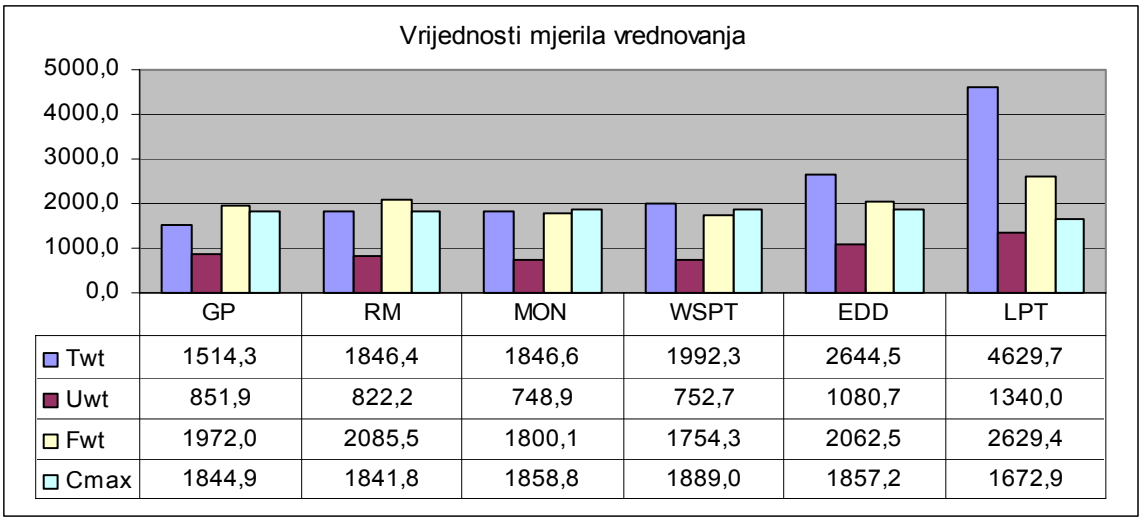
4.5.4 Statička okolina raspoređivanja

U statičkoj okolini svi poslovi su raspoloživi od početka rada sustava, a jednaka pretpostavka vrijedi i za strojeve. Zadatak genetskog programiranja je naći funkciju prioriteta koja će, uz primjenu po algoritmu 4.6, dati što bolje rješenje po pitanju zadanog mjerila vrednovanja rasporeda. Jedinka genetskog programa je i u ovom okruženju predstavljena stablom koje predstavlja funkciju prioriteta. Većina naziva čvorova zadržana je iz prethodnog okruženja, iako je definicija vrijednosti za neke čvorove promijenjena. Genetskom je programu na raspolaganju skup čvorova prikazan u tablici 4.10.

Tablica 4.10 Popis čvorova za raspoređivanje na jednolikim strojevima

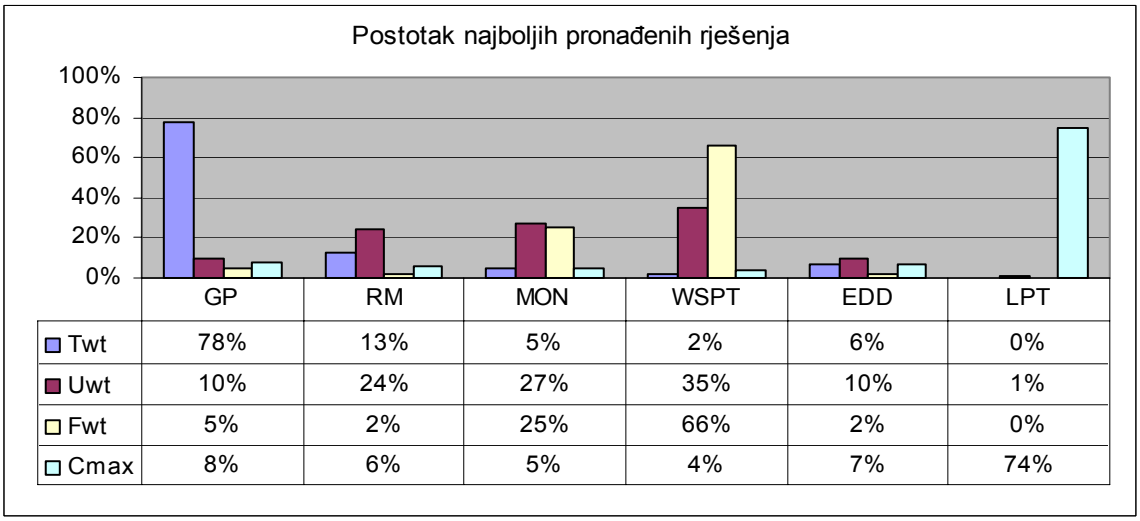
Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	**Definicija vrijednosti podatkovnog čvora**
pt	nominalno trajanje obrade posla (p_j)
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
Nr	preostali broj poslova (koji još nisu raspoređeni)
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\left\{d_j - p_j - time, 0\right\}$
SLs	dopuštena odgoda uz brzinu stroja, $\max\left\{d_j - p_j/s_k - time, 0\right\}$
SPD	brzina stroja na kojemu se trenutno računaju prioriteti (s_k)
Msm	zbroj brzina svih strojeva, tj. efektivni broj strojeva ($\hat{m}$)
STP	trajanje postavljanja sa prethodnog na promatrani posao, s_{lj}
Sav	prosječno trajanje postavljanja sa prethodnog (l) na sve ostale poslove, $\frac{1}{n-1}\sum_{j=1}^{n} s_{lj}$

U statičkoj okolini provedene su dvije vrste pokusa: u jednoj skupini rješavan je osnovni problem težinskog zaostajanja, a u drugoj skupini isti problem uz trajanja postavljanja. Za osnovni problem bez trajanja postavljanja provedeno je 10 pokusa, a najbolje rješenje pronađeno genetskim programiranjem dano je u prilogu B2. Za skup ispitnih primjera za ocjenu, usporedni rezultati prikazani su na slikama 4.31 i 4.32.



	GP	RM	MON	WSPT	EDD	LPT
Twt	1514,3	1846,4	1846,6	1992,3	2644,5	4629,7
Uwt	851,9	822,2	748,9	752,7	1080,7	1340,0
Fwt	1972,0	2085,5	1800,1	1754,3	2062,5	2629,4
Cmax	1844,9	1841,8	1858,8	1889,0	1857,2	1672,9

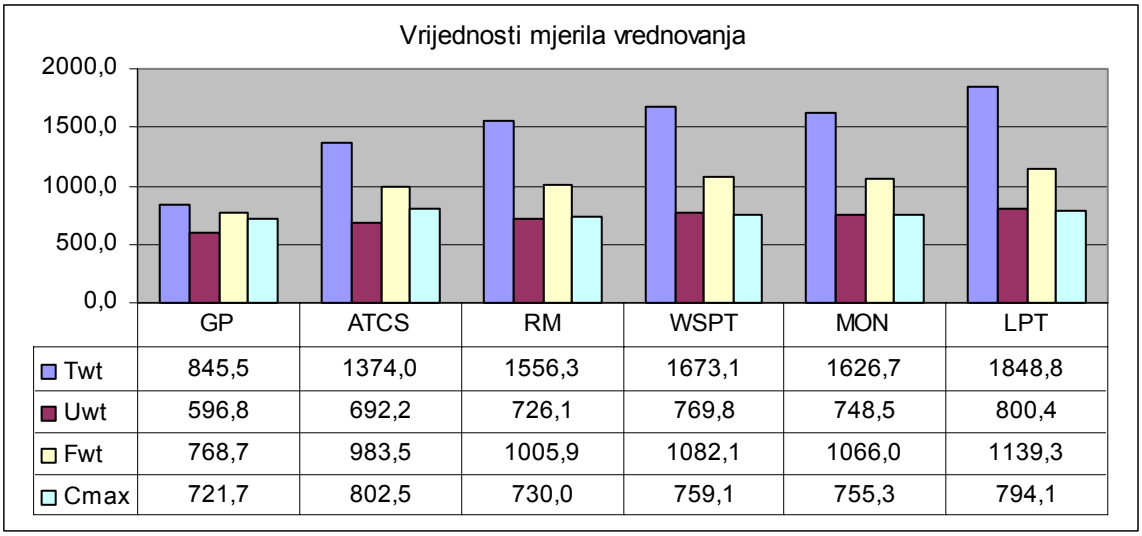
Slika 4.31 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	RM	MON	WSPT	EDD	LPT
Twt	78%	13%	5%	2%	6%	0%
Uwt	10%	24%	27%	35%	10%	1%
Fwt	5%	2%	25%	66%	2%	0%
Cmax	8%	6%	5%	4%	7%	74%

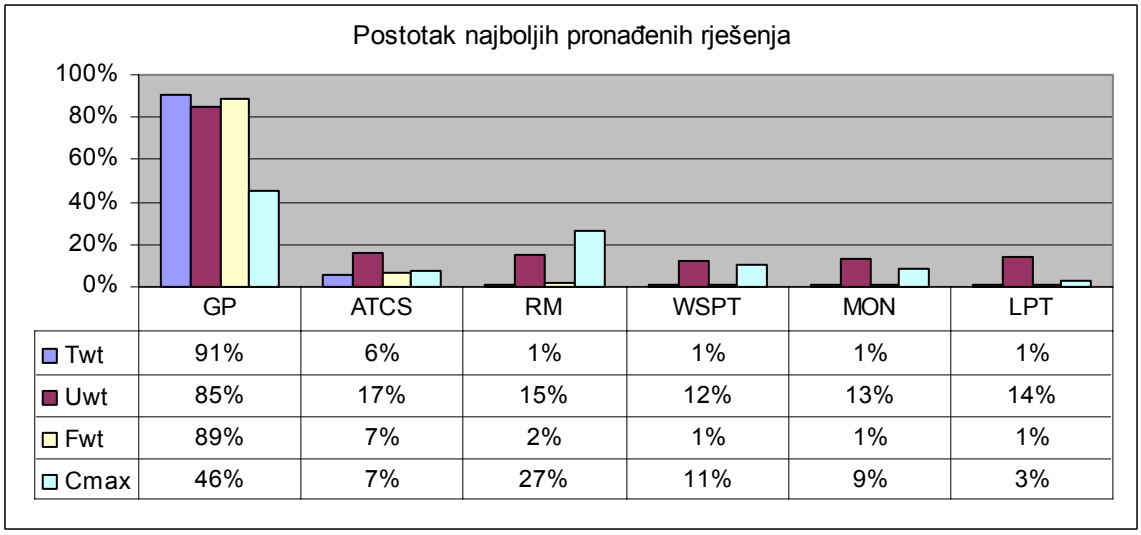
Slika 4.32 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Vidljivo je da su kriteriji optimiranja uglavnom podijeljeni među heuristikama: pravilo dobiveno genetskim programiranjem uvjerljivo je najbolje po pitanju težinskog zaostajanja, WSPT za težinsko protjecanje a LPT pravilo postiže najbolje rezultate za ukupnu duljinu rasporeda. Za problem sa trajanjima postavljanja također je provedeno 10 pokusa, a u promatrana pravila uvršteno je i ATCS pravilo namijenjeno ovoj vrsti problema. Odabrano rješenje genetskog programiranja navedeno je u prilogu B2, a na slikama 4.33 i 4.34 prikazani su usporedni rezultati po pravilima.



	GP	ATCS	RM	WSPT	MON	LPT
Twt	845,5	1374,0	1556,3	1673,1	1626,7	1848,8
Uwt	596,8	692,2	726,1	769,8	748,5	800,4
Fwt	768,7	983,5	1005,9	1082,1	1066,0	1139,3
Cmax	721,7	802,5	730,0	759,1	755,3	794,1

Slika 4.33 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



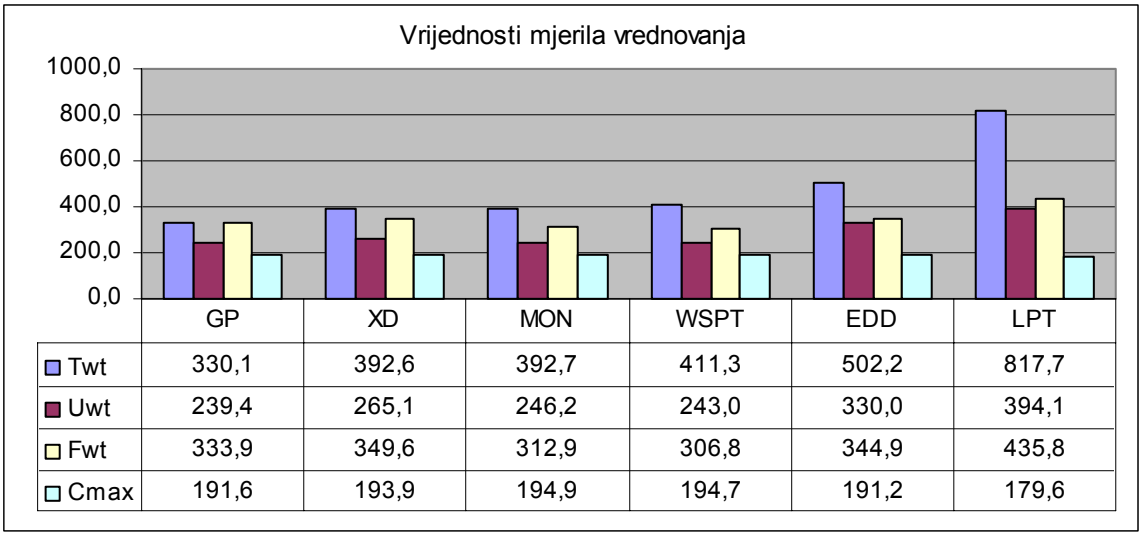
	GP	ATCS	RM	WSPT	MON	LPT
Twt	91%	6%	1%	1%	1%	1%
Uwt	85%	17%	15%	12%	13%	14%
Fwt	89%	7%	2%	1%	1%	1%
Cmax	46%	7%	27%	11%	9%	3%

Slika 4.34 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Iz rezultata se može uočiti da genetsko programiranje lakše nalazi rješenje koje je relativno bolje od ostalih pravila u slučaju 'manje uobičajene' okoline raspoređivanja, kao što je postojanje trajanja postavljanja za jednolike paralelne strojeve.

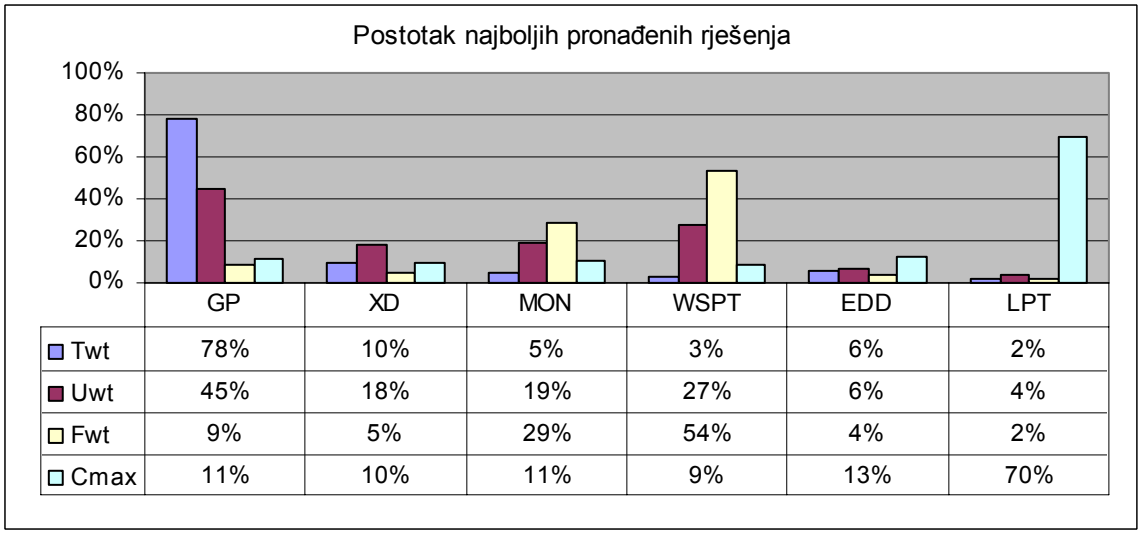
4.5.5 Dinamička okolina raspoređivanja

U dinamičkoj okolini uvedena su i vremena pripravnosti poslova, dok su strojevi i dalje raspoloživi od početka rada sustava. U postupku izvođenja pravila raspoređivanja genetskim programiranjem korišten je isti skup čvorova kao u tablici 4.10. Provedeni pokusi su podijeljeni u dvije skupine, ovisno o postojanju trajanja postavljanja među poslovima. Za obje okoline provedeno je optimiranje težinskog zaostajanja i ukupne duljine rasporeda. Za sve inačice problema načinjeno je po 10 pokusa, a najbolja postignuta rješenja za genetsko programiranje navedena su u prilogu B2. Za problem bez trajanja postavljanja uz optimiranje težinskog zaostajanja rezultati su prikazani na slikama 4.35 i 4.36.



	GP	XD	MON	WSPT	EDD	LPT
Twt	330,1	392,6	392,7	411,3	502,2	817,7
Uwt	239,4	265,1	246,2	243,0	330,0	394,1
Fwt	333,9	349,6	312,9	306,8	344,9	435,8
Cmax	191,6	193,9	194,9	194,7	191,2	179,6

Slika 4.35 Optimiranje težinskog zaostajanja – skup primjera za ocjenu

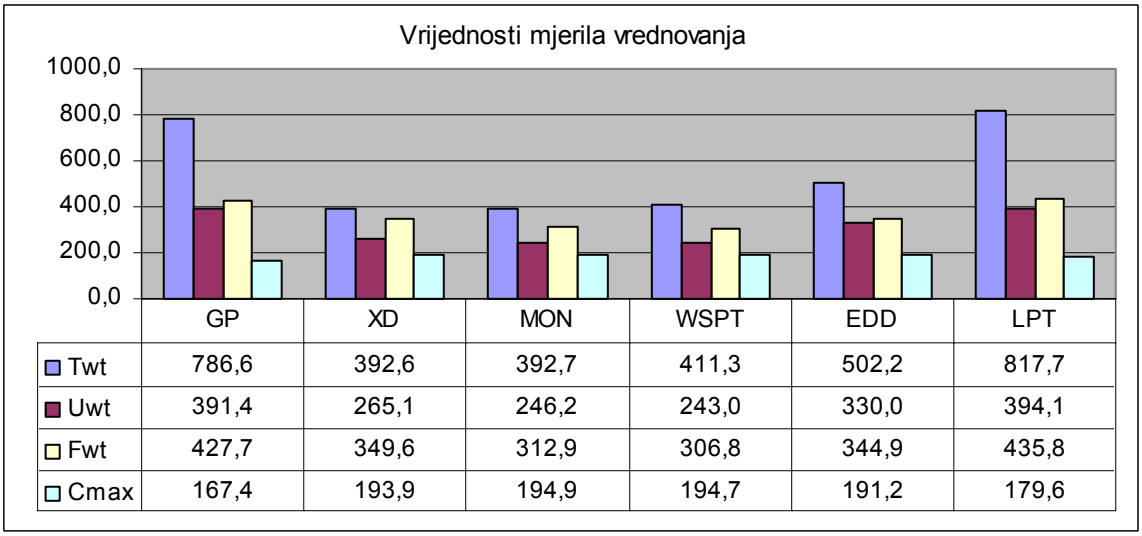


	GP	XD	MON	WSPT	EDD	LPT
Twt	78%	10%	5%	3%	6%	2%
Uwt	45%	18%	19%	27%	6%	4%
Fwt	9%	5%	29%	54%	4%	2%
Cmax	11%	10%	11%	9%	13%	70%

Slika 4.36 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

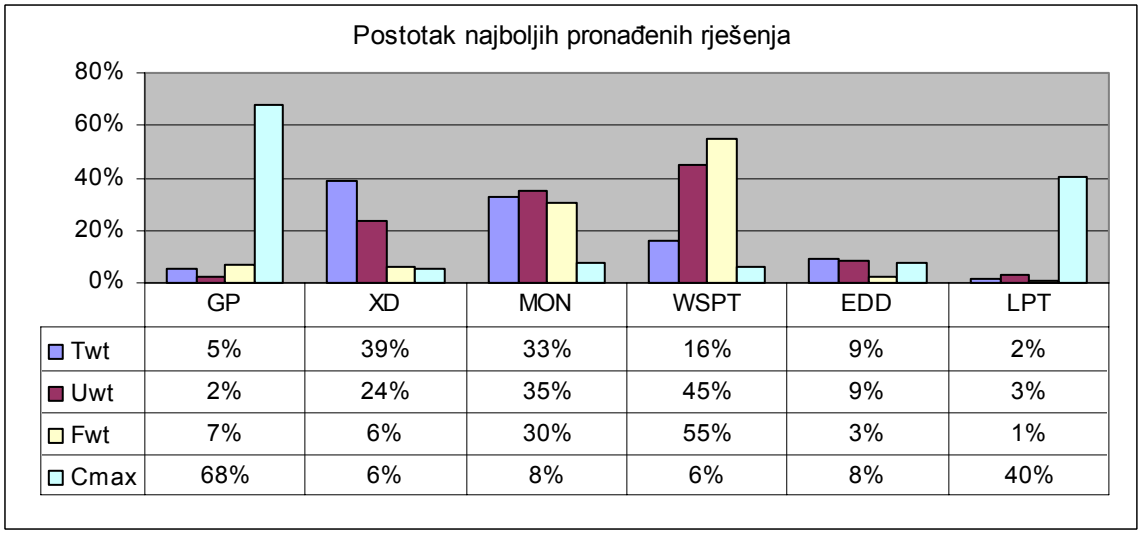
Iz prikaza rezultata je vidljivo da pravilo dobiveno genetskim programiranjem nalazi najbolje rješenje po pitanju težinskog zaostajanja. Za težinsko protjecanje najbolje rješenje postiže se WSPT pravilom, a najbolju ukupnu duljinu rasporeda daje (očekivano) LPT pravilo. Na jednakom skupu promatranih pravila uspoređeni su i rezultati optimiranja ukupne duljine rasporeda, koji su prikazani na slikama 4.37 i 4.38.

Može se uočiti da je pravilo izvedeno genetskim programiranjem uspješno po kriteriju ukupne duljine rasporeda, dok je učinkovitost slaba za ostale kriterije. Iz rezultata je također vidljivo da je teško postići visoku kvalitetu rješenja za više kriterija istovremeno. Tražimo li otprilike jednaku učinkovitost po više kriterija, najbolje je upotrijebiti neko od postojećih pravila 'opće namjene', no želimo li optimirati pojedino mjerilo, tada stvaranje prikladnog pravila postupkom učenja predstavlja dobar izbor.



	GP	XD	MON	WSPT	EDD	LPT
Twt	786,6	392,6	392,7	411,3	502,2	817,7
Uwt	391,4	265,1	246,2	243,0	330,0	394,1
Fwt	427,7	349,6	312,9	306,8	344,9	435,8
Cmax	167,4	193,9	194,9	194,7	191,2	179,6

Slika 4.37 Optimiranje ukupne duljine rasporeda – skup primjera za ocjenu

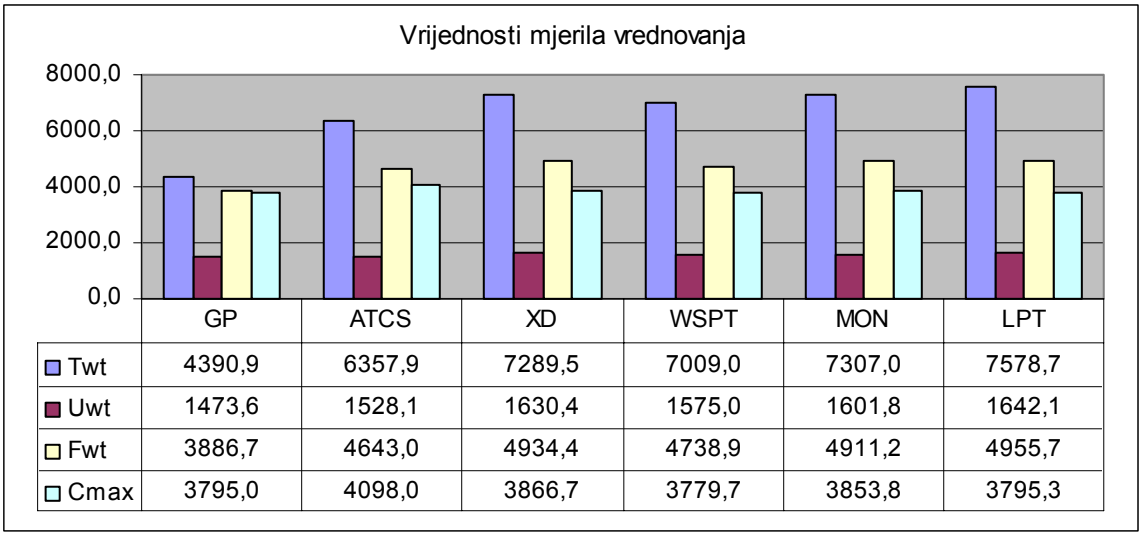


	GP	XD	MON	WSPT	EDD	LPT
Twt	5%	39%	33%	16%	9%	2%
Uwt	2%	24%	35%	45%	9%	3%
Fwt	7%	6%	30%	55%	3%	1%
Cmax	68%	6%	8%	6%	8%	40%

Slika 4.38 Postoci dominacije za ukupnu duljinu rasporeda – skup primjera za ocjenu

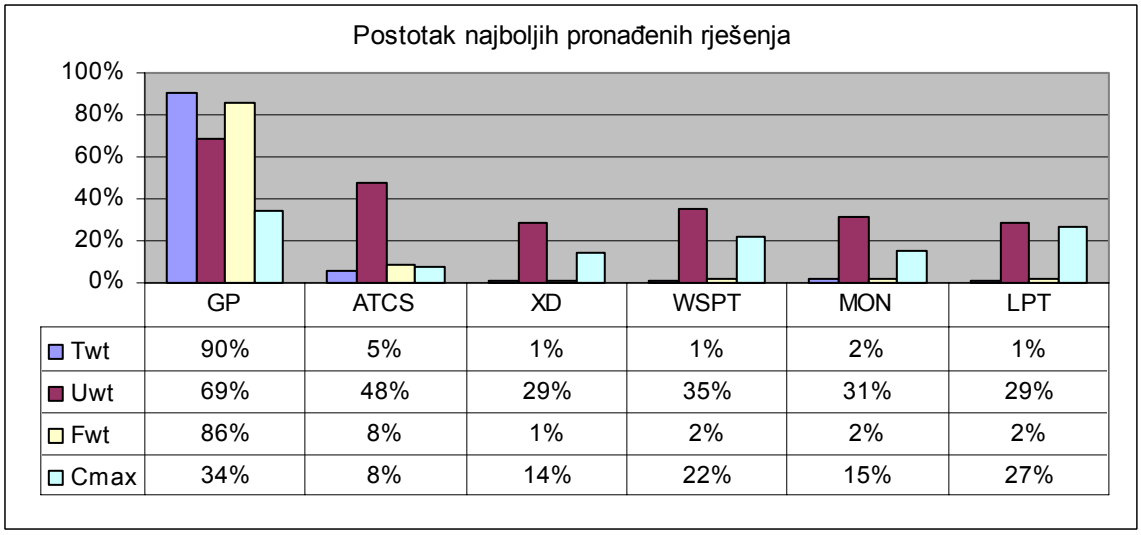
Rezultati problema raspoređivanja uz trajanja postavljanja i optimiranje težinskog zaostajanja prikazani su na slikama 4.39 i 4.40.

Kao što je vidljivo iz prikaza, u okoj okolini optimiranje po jednom kriteriju daje bolju uspješnost na više kriterija, djelomično zbog nedostatka postojećih algoritama u prilagodbi ovim uvjetima raspoređivanja. Za jednake uvjete provedeni su i pokusi uz optimiranje ukupne duljine rasporeda, a rezultati najboljeg pronađenog rješenja prikazani su na slikama 4.41 i 4.42.



	GP	ATCS	XD	WSPT	MON	LPT
Twt	4390,9	6357,9	7289,5	7009,0	7307,0	7578,7
Uwt	1473,6	1528,1	1630,4	1575,0	1601,8	1642,1
Fwt	3886,7	4643,0	4934,4	4738,9	4911,2	4955,7
Cmax	3795,0	4098,0	3866,7	3779,7	3853,8	3795,3

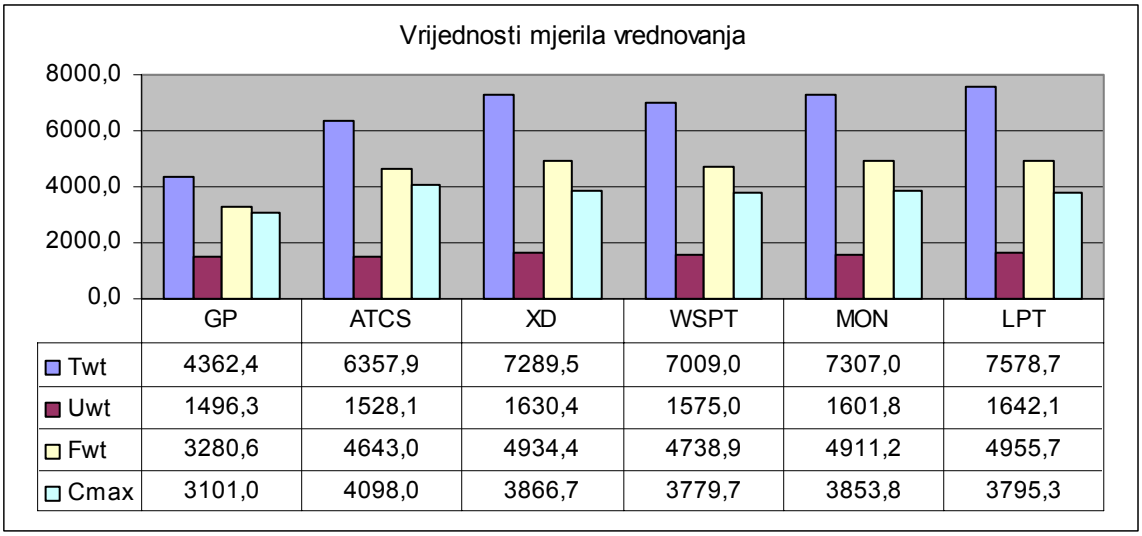
Slika 4.39 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	ATCS	XD	WSPT	MON	LPT
Twt	90%	5%	1%	1%	2%	1%
Uwt	69%	48%	29%	35%	31%	29%
Fwt	86%	8%	1%	2%	2%	2%
Cmax	34%	8%	14%	22%	15%	27%

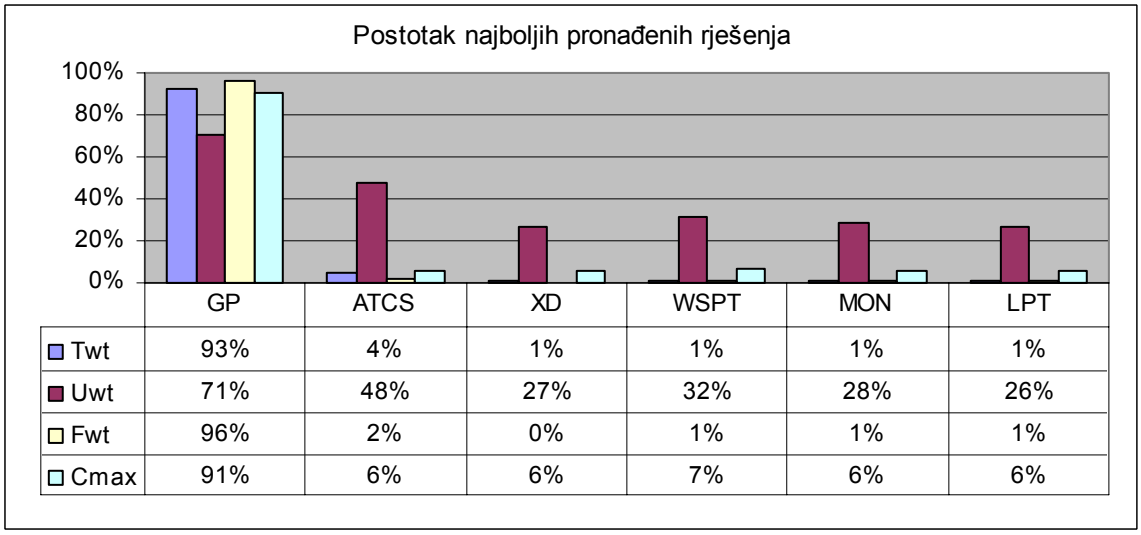
Slika 4.40 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

Na ovom se primjeru može vidjeti da je kriterij ukupne duljine rasporeda isplativiji nego u primjeru bez trajanja postavljanja, budući je pravilo izvedeno genetskim algoritmom postiglo dobre rezultate po većini mjerila. Uzrok ovakvom učinku može se objasniti utjecajem prikladnog izbora redoslijeda poslova i odabirom manjeg ukupnog trajanja postavljanja, što znatno utječe na konačni rezultat. Ipak, ovakvu je dobru učinkovitost po više kriterija teško predvidjeti – na primjer, neko drugo rješenje genetskog programiranja moglo bi imati slične ili čak nešto bolje rezultate po pitanju težinskog zaostajanja, no lošije za ostale kriterije. Zbog ograničenog opsega rada ove popratne pojave nisu podrobnije ispitane, no to predstavlja jedan od mogućih dodatnih smjerova istraživanja.



	GP	ATCS	XD	WSPT	MON	LPT
Twt	4362,4	6357,9	7289,5	7009,0	7307,0	7578,7
Uwt	1496,3	1528,1	1630,4	1575,0	1601,8	1642,1
Fwt	3280,6	4643,0	4934,4	4738,9	4911,2	4955,7
Cmax	3101,0	4098,0	3866,7	3779,7	3853,8	3795,3

Slika 4.41 Optimiranje ukupne duljine rasporeda – skup primjera za ocjenu



	GP	ATCS	XD	WSPT	MON	LPT
Twt	93%	4%	1%	1%	1%	1%
Uwt	71%	48%	27%	32%	28%	26%
Fwt	96%	2%	0%	1%	1%	1%
Cmax	91%	6%	6%	7%	6%	6%

Slika 4.42 Postoci dominacije za ukupnu duljinu rasporeda – skup primjera za ocjenu

4.6 Raspoređivanje na paralelnim nesrodnim strojevima

4.6.1 Okruženje nesrodnih strojeva

Okolina nesrodnih strojeva u velikoj se mjeri razlikuje od okruženja jednolikih strojeva razmatranog u prethodnom poglavlju. Iako se svaki posao i dalje može izvesti na bilo kojem stroju, ovaj put je pridruživanje posla određenom stroju od velike važnosti jer su trajanja obrade poslova na različitim strojevima potpuno neovisna. Potrebno je stoga algoritmu raspoređivanja omogućiti uvid u prikladnost pojedinoga posla pojedinome stroju i na učinkovit način iskoristiti tu informaciju.

U prethodnim poglavljima korišten je model predodređenog raspoređivanja, u kojemu su svi podaci o poslovima i njihovim dolascima bili dostupni. U ovom će se poglavlju, ponajprije zbog načina rada postupaka za usporedbu (sljedeći odjeljak), rabiti model raspoređivanja na zahtjev opisan u poglavlju 2.3. U takvim uvjetima raspoređivanja podaci o poslovima dostupni su tek po dolasku posla u sustav, što se u većini primjera poklapa sa vremenom pripravnosti posla. Uz ovu razliku, vrijede sljedeće pretpostavke:

- podaci o poslovima nisu unaprijed poznati (raspoređivanje na zahtjev), što povlači dinamički način izrade rasporeda,

- poslovi su neprekidivi,
- strojevi su neprekidno raspoloživi od početka rada sustava.

U sljedećem odjeljku opisani su promatrani postupci raspoređivanja na nesrodnim strojevima i način njihove primjene. U narednim odjeljcima definiran je pristup raspoređivanju uz pomoć genetskog programiranja i pokazani su usporedni rezultati raspoređivanja na ispitnim primjerima.

4.6.2 Postupci raspoređivanja na nesrodnim strojevima

Prilikom raspoređivanja na nesrodnim strojevima, dolaze do izražaja dvije faze raspoređivanja poslova: pridruživanje, pri kojemu se poslovi dodjeljuju odgovarajućim strojevima, i uređivanje, pri čemu se određuje redoslijed eventualnog većeg broja poslova na jednom stroju. Neki postupci raspoređivanja ove dvije faze obavljaju odvojeno, a kod nekih je jedna sadržana u drugoj (npr. uređivanje se obavlja redom kojim su poslovi pridruženi stroju). Također je dvije različite metode pridruživanja i uređivanja moguće obuhvatiti u jedinstveni postupak raspoređivanja.

Algoritmi raspoređivanja u dinamičkim uvjetima raspoređivanja na zahtjev mogu se razlikovati po tome u kojem trenutku je neki posao raspoređen na odgovarajući stroj. Jedna skupina algoritama raspoređuje pristigli posao na neki od strojeva odmah po njegovom dolasku, čime se zapravo obavlja samo pridruživanje poslova. Uređivanje poslova na stroju podrazumijeva redoslijed izvođenja jednak redu prispijeća, tj. uređivanje po FCFS (engl. *first come first served*) principu. Druga skupina algoritama ne raspoređuje svaki posao zasebno odmah po njegovu dolasku, već se u predodređenim vremenskim intervalima zajednički raspoređuju svi do tada pristigli a još nepokrenuti poslovi (engl. *batch mode*). Skup svih poslova (zadataka) koji su trenutno u sustavu, a izvođenje im nije pokrenuto, naziva se još i *meta-zadatkom*. Prilikom raspoređivanja, obavlja se i pridruživanje i uređivanje svih poslova u meta-zadatku. U većini primjena algoritmi iz druge skupine postižu bolje rezultate, pa će se u ovom radu promatrati postupci iz te skupine.

Za okruženje nesrodnih strojeva postoji niz heuristika raspoređivanja [Mar 04, Dav 81], no najpoznatije metode su min-min i max-min heuristike. Ova dva algoritma (pogotovo min-min) smatraju se ponajboljim heurističkim algoritmima za veliki broj problema raspoređivanja [He 03, Mar 04]. Algoritmi raspoređivanja koji će kasnije dobiti naziv min-min i max-min predloženi su još u radu [Iba 77] uz nekoliko drugih postupaka. Prije opisa algoritma, potrebno je uvesti sljedeće oznake:

- p_{ij} je trajanje izvođenja posla j na stroju m_i; sva trajanja izvođenja su unaprijed poznata sa dovoljnom preciznošću;
- t_{ri} je vrijeme pripravnosti stroja m_i (s obzirom na poslove koji se na njemu trenutno izvode);
- C_{ij} je vrijeme završetka izvođenja posla j na stroju m_i; budući su vremena završetka vezana samo uz poslove, za određivanje kriterija koristi se oznaka C_j kao vrijeme završetka posla j;
- J^{meta} je skup svih raspoloživih poslova koji još nisu pokrenuti; neki od tih poslova su možda dodijeljeni nekome stroju, ali njihovo izvođenje još nije započelo pa su podložni novom pridruživanju.

Min-min postupak je prikazan kao algoritam 4.7. U svakom koraku algoritam računa očekivano vrijeme završetka izvođenja poslova na svakom od strojeva, uzevši u obzir vrijeme pripravnosti stroja. Potom se za svaki posao pronalazi stroj koji daje najmanje očekivano vrijeme završetka. Postupak tada raspoređuje posao koji postiže

najbliže vrijeme završetka (od svih poslova) na stroj koji tom poslu daje to vrijeme završetka. Max-min algoritam također za svaki posao pronalazi stroj koji daje najmanje vrijeme završetka, no raspoređuje se onaj posao koji ima *najveće* očekivano (minimalno) vrijeme završetka među svim poslovima. U primjeni se mogu naći i postupci zasnovani na min-min algoritmu, koji uz neke prilagodbe danom okruženju najčešće daju nešto bolje rezultate raspoređivanja [He 03]. U ovom radu promatra se samo 'osnovna' inačica navedena dva algoritma.

```
za (sve poslove iz J^meta)
      za (sve strojeve)
          C_ij = t_ri + p_ij ;
dok (postoje neraspoređeni poslovi u J^meta)
{     za (svaki zadatak iz J^meta)
          pronaći najmanji C_ij te stroj m_i na kojemu se postiže najranije
          vrijeme završetka;
      naći posao j sa najmanjim vremenom završetka;
      dodijeliti posao j stroju l koji daje najranije vrijeme završetka;
      ukloniti posao j iz J^meta ;
      obnoviti t_rl = t_rl + p_lj ;
      obnoviti C_lj za sve preostale poslove iz J^meta ;
}
```

Algoritam 4.7 Min-min postupak raspoređivanja

Važan element uporabe opisanih postupaka raspoređivanja je i odabir trenutka raspoređivanja nagomilanih poslova. U stvarnim se uvjetima obično definira neki vremenski interval nakon kojega se algoritam ponovno primjenjuje. Određivanje veličine intervala u velikoj je mjeri ovisno o vrsti sustava u kojemu se raspoređivanje obavlja. U slučaju 'predugog' intervala može se dogoditi da neki pristigli posao, koji bi zbog 'dobrih' svojstava inače bio raspoređen na određeni stroj, ne bude raspoređen na taj stroj jer je na istom stroju već pokrenut posao koji je tom stroju dodijeljen u prethodnoj iteraciji raspoređivanja. Zbog toga se u ovom radu, u postupku simulacije, primjenjuje najbolji slučaj: algoritam se izvodi svaki puta kada novi posao dođe u sustav. Na taj način zajamčena je najbolja moguća učinkovitost algoritma.

4.6.3 Raspoređivanje uz pomoć genetskog programiranja

U postupku primjene pravila raspoređivanja na nesrodnim procesorima potrebno je u odluku uključiti i prikladnost stroja na kojemu se posao raspoređuje. U prethodnom je okruženju jedan od poslova bio raspoređen na prvi raspoloživi stroj, bez uvida u trajanja izvođenja na drugim strojevima (algoritam 4.6). U ovom bi okruženju takav postupak dao lošije rezultate jer ne uzima u obzir sve postojeće strojeve. U ovom radu je stoga predložen algoritam koji u kombinaciji sa funkcijom prioriteta, izvedenom s pomoću genetskog programiranja, raspoređuje poslove na nesrodne strojeve. Ovaj postupak bi se mogao nazvati *meta-algoritmom* budući se kao dio algoritma javlja funkcija odabira koja tek treba biti definirana, a postupak opisuje način korištenja odgovarajuće funkcije. Opisani postupak prikazan je kao algoritam 4.8.

Algoritam se može opisati na sljedeći način: u prvom trenutku kada su barem jedan stroj i jedan posao raspoloživi (tj. barem neki posao se može pokrenuti), računaju se prioriteti svih raspoloživih poslova na svim strojevima. Potom se za svaki posao zabilježi

stroj na kojemu posao postiže najbolju (po pretpostavci najmanju) vrijednost prioriteta i sama ta vrijednost. Postoji li barem jedan posao za kojega je najbolji odabrani stroj raspoloživ (odnosno barem jedan posao koji se, po najboljem prioritetu, može odmah pokrenuti), tada se odabire najbolja kombinacija posla i stroja i odgovarajući posao se pokreće. Ukoliko su za sve poslove najbolji odabrani strojevi raspoloživi tek u budućnosti, raspoređivanje se ne obavlja, već se čeka do prvog sljedećeg trenutka raspoloživosti nekog drugog *stroja* (budući da nijedan od raspoloživih strojeva nije 'dovoljno dobar' za bilo koji raspoloživi posao). Na taj se način prilikom sljedećeg pokušaja raspoređivanja u odabir uključuju i poslovi koji su u međuvremenu došli u sustav.

```
dok (postoje neraspoređeni poslovi)
{     čekaj dok barem jedan stroj i jedan posao nisu raspoloživi;
      za (sve raspoložive poslove)
          za (sve strojeve)
              π_ij = prioritet posla j za stroj i;
      za (sve raspoložive poslove)
          odredi najbolji stroj za svaki posao (onaj stroj koji za neki
              posao daje najmanju vrijednost π_ij );
      ako (postoji barem jedan posao čiji najbolji stroj je raspoloživ)
      {   odredi najbolji posao čiji najbolji stroj je raspoloživ;
          rasporedi odabrani posao na odabrani stroj;
      }
      inače
          čekaj do pojave sljedećeg raspoloživog stroja;
}
```

Algoritam 4.8 Meta-algoritam raspoređivanja na nesrodnim strojevima

Zadatak je genetskog programiranja pronaći takvu funkciju prioriteta koja će najbolje iskoristiti strukturu prikazanog meta-algoritma. Kako bi se u funkciji mogla maksimalno iskoristiti svojstva poslova i strojeva, potrebno je definirati odgovarajuće podatkovne strukture koje će u gradnji funkcije biti na raspolaganju genetskom programu. Skup mogućih funkcijskih i podatkovnih čvorova za ovo okruženje prikazan je u tablici 4.11. Među čvorovima se ne nalaze informacije o ukupnom broju poslova, njihovim dolascima itd., što odgovara raspoređivanju na zahtjev (nije dostupna nikakva informacija o budućnosti sustava).

Tablica 4.11 Popis čvorova za raspoređivanje na nesrodnim strojevima

Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje izvođenja posla na promatranom stroju (p_{ij})
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
SL	dopuštena odgoda uz brzinu dotičnog stroja, $\max\{d_j - p_{ij} - time, 0\}$
pmin	najkraće trajanje izvođenja posla (za sve postojeće strojeve)
pavg	srednje trajanje izvođenja posla

PAT	strpljenje (od engl. *patience*) – količina vremena za koju će stroj koji za promatrani posao daje najkraće trajanje izvođenja biti raspoloživ; 0 ako dotični stroj jest raspoloživ
MR	količina vremena do raspoloživosti promatranog stroja (od engl. *machine ready*); 0 ako stroj jest raspoloživ
age	vrijeme koje je posao proveo u sustavu, $time - r_j$

Čvor 'pmin' označava najkraće moguće trajanje izvođenja koje posao može postići, što obično vrijedi samo za jedan od strojeva, dok čvor 'pavg' označava srednju vrijednost svih trajanja obrade nekoga posla. Čvor 'age' označava koliko vremena je proteklo od dolaska posla u sustav, 'MR' je količina vremena koja treba proći prije nego promatrani stroj postane raspoloživ, a 'PAT' je količina vremena koju bi posao trebao čekati da bude izveden na stroju koji mu daje najkraće vrijeme izvođenja. Potrebno je napomenuti da se, budući se ocjenjivanje obavlja za sve poslove i sve strojeve, vrijednosti ovih varijabli mogu definirati ovisno o promatranom poslu i stroju. To vrijedi za čvorove 'pt' i 'SL', dok čvorovi 'dd', 'w', 'pmin', 'pavg', 'PAT' i 'age' ovise samo o trenutnom poslu, a čvor 'MR' ovisi samo o promatranom stroju. Budući da algoritam računa sve kombinacije, uz oznake čvorova ne navode se indeksi koji bi označavali njihovu ovisnost.

Odabir dostupnih varijabli ovisit će i o funkciji cilja, odnosno kriteriju ocjenjivanja rasporeda, pa npr. u optimiranju ukupne duljine rasporeda neće biti potrebno uzimati u obzir težinu, željeno vrijeme završetka posla i dopuštenu odgodu.

4.6.4 Oblikovanje ispitnih primjera

Ispitni primjeri se u ovome okruženju definiraju slično kao i za jednolike strojeve, uz sljedeće razlike:

- za svaki posao slučajno se definira trajanje izvođenja posla na svakom stroju (p_{ij}), što čini $n \times m$ vrijednosti za svaki ispitni primjer, gdje je *n* broj poslova a *m* broj strojeva u dotičnom primjeru;
- očekivano ukupno trajanje poslova $\hat{p}$ definira se kao

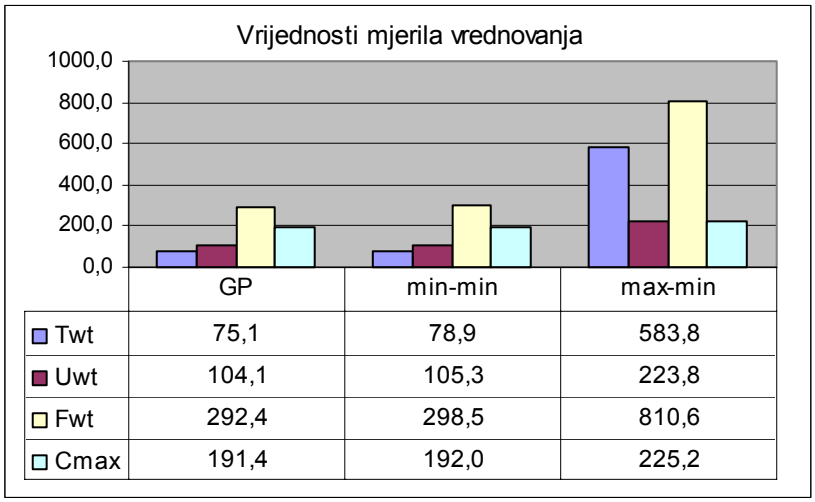
$$\hat{p} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}}{m^2}. \tag{4.42}$$

U nazivniku gornjeg izraza nalazi se kvadrat broja strojeva jer promatrani algoritmi u većini primjera poslove uspijevaju rasporediti na strojeve koji ih izvode u trajanju manjem od prosječnog trajanja izvođenja posla. Uz pomoć očekivanog ukupnog trajanja i parametara *T* i *R*, dolasci poslova generiraju se po izrazu (4.34), a željeno vrijeme završetka poslova računa se po izrazu (4.36). Definirano je 120 primjera za učenje i 600 primjera za ocjenu, čiji parametri su pregledno prikazani u prilogu A3. Funkcija cilja oblikuje se na jednak način kao i za paralelne jednolike strojeve uz srednje trajanje izvođenja izraženo sa $\overline{p} = \hat{p}/n$.

4.6.5 Raspoređivanje na nesrodnim strojevima

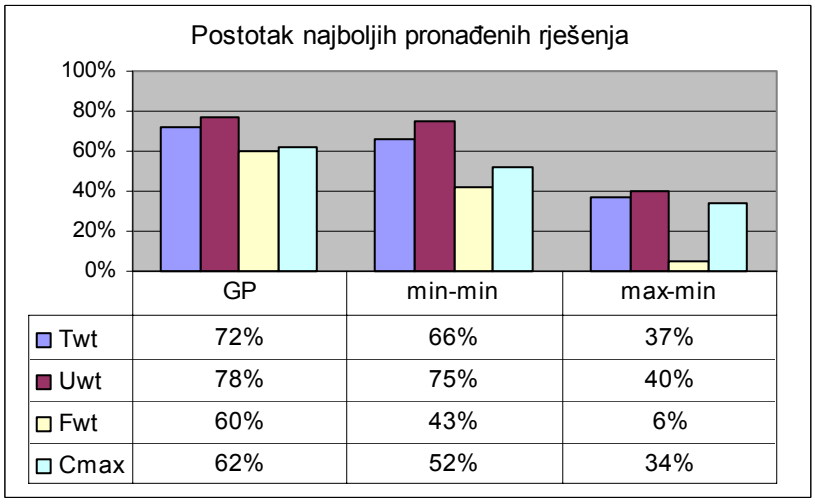
Ispitivanje učinkovitosti raspoređivanja na paralelnim nesrodnim strojevima provedeno je uz pretpostavke navedene na početku poglavlja: raspoređivanje se obavlja na zahtjev (podaci o budućnosti sustava nisu poznati), a poslovi i njihovi podaci postaju dostupni u trenutku njihovog vremena pripravnosti.

Jedan dio pokusa proveden je uz optimiranje netežinskog trajanja protjecanja (budući ostali promatrani algoritmi ne uzimaju težine u obzir); u tome su slučaju iz skupa čvorova izostavljeni čvorovi koji predstavljaju težine poslova, željeno vrijeme završetka i dopuštenu odgodu. Za potrebe optimiranja ovoga kriterija provedeno je 20 pokusa genetskog programiranja uz pretpostavljene vrijednosti parametara (poglavlje 4.2) po algoritmu 4.8. Najbolja rješenja iz primjera za učenje primijenjena su na primjere za ocjenu kako bi se izabralo konačno rješenje u obliku funkcije prioriteta, koje je prikazano u prilogu B3. Usporedni rezultati na skupu primjera za ocjenu prikazani su na slikama 4.43 i 4.44.



	GP	min-min	max-min
Twt	75,1	78,9	583,8
Uwt	104,1	105,3	223,8
Fwt	292,4	298,5	810,6
Cmax	191,4	192,0	225,2

Slika 4.43 Optimiranje trajanja protjecanja – skup primjera za ocjenu

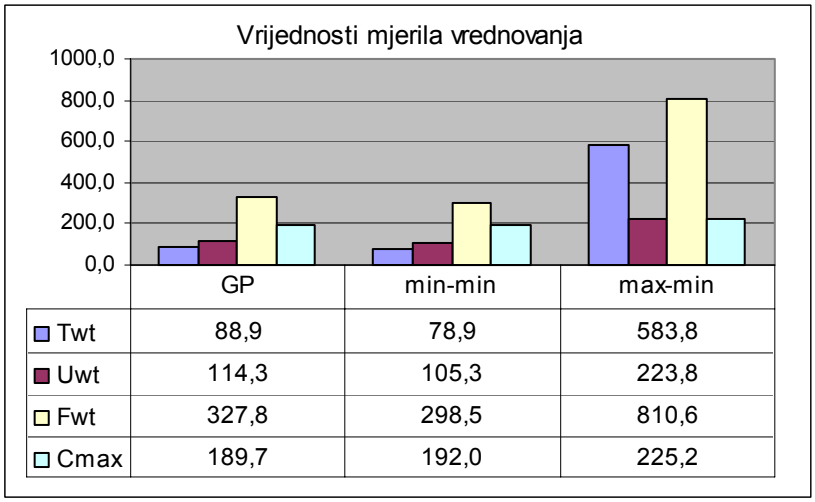


	GP	min-min	max-min
Twt	72%	66%	37%
Uwt	78%	75%	40%
Fwt	60%	43%	6%
Cmax	62%	52%	34%

Slika 4.44 Postoci dominacije za trajanje protjecanja – skup primjera za ocjenu

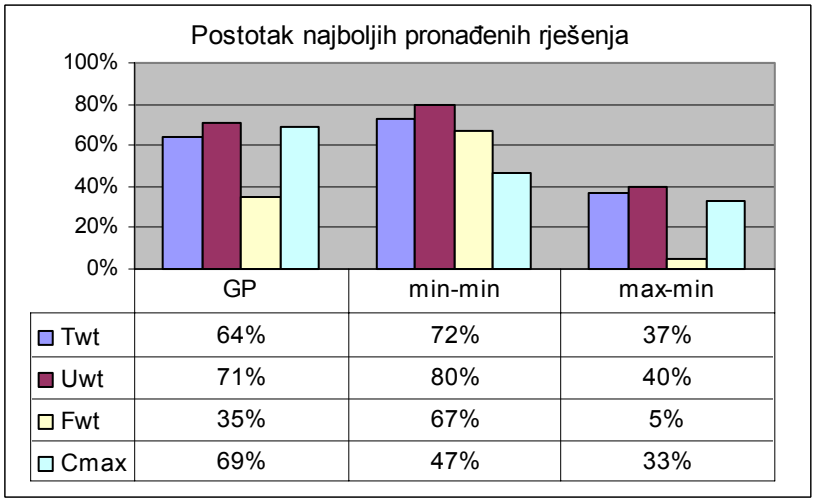
Iz rezultata se može uočiti da je učinkovitost min-min algoritma i algoritma izvedenog genetskim programiranjem uglavnom podjednaka, uz tek nešto izraženiju razliku kod postotaka dominacije za pojedine kriterije. Max-min algoritam je uvjerljivo najlošiji, no takav rezultat je vjerojatno djelomično uzrokovan i svojstvima ispitnih primjera.

Osim trajanja protjecanja, pokusi su provedeni i za kriterij ukupne duljine rasporeda kao dobrote rješenja evolucijskog procesa. Provedeno je 20 pokusa uz jednake podatkovne čvorove za jedinke genetskog programa, a najbolje rješenje dano je u prilogu B3. Rezultati za ovaj kriterij prikazani su na slikama 4.45 i 4.46.



	GP	min-min	max-min
Twt	88,9	78,9	583,8
Uwt	114,3	105,3	223,8
Fwt	327,8	298,5	810,6
Cmax	189,7	192,0	225,2

Slika 4.45 Optimiranje ukupne duljine rasporeda – skup primjera za ocjenu

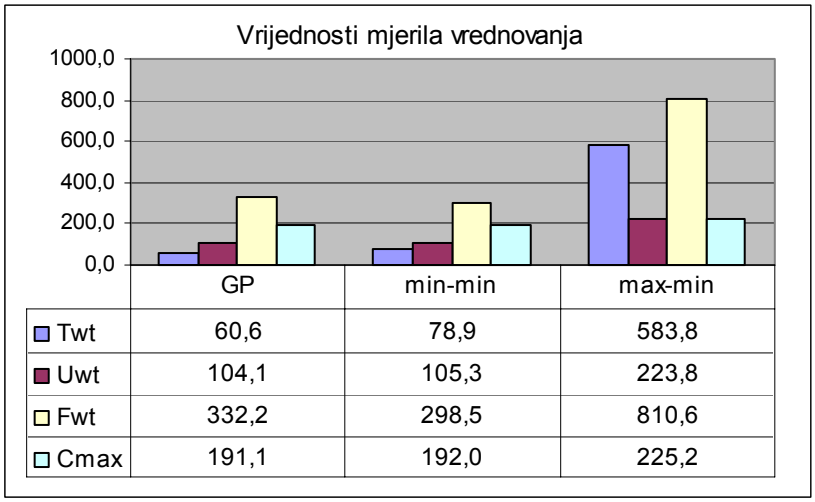


	GP	min-min	max-min
Twt	64%	72%	37%
Uwt	71%	80%	40%
Fwt	35%	67%	5%
Cmax	69%	47%	33%

Slika 4.46 Postoci dominacije za ukupnu duljine rasporeda – skup primjera za ocjenu

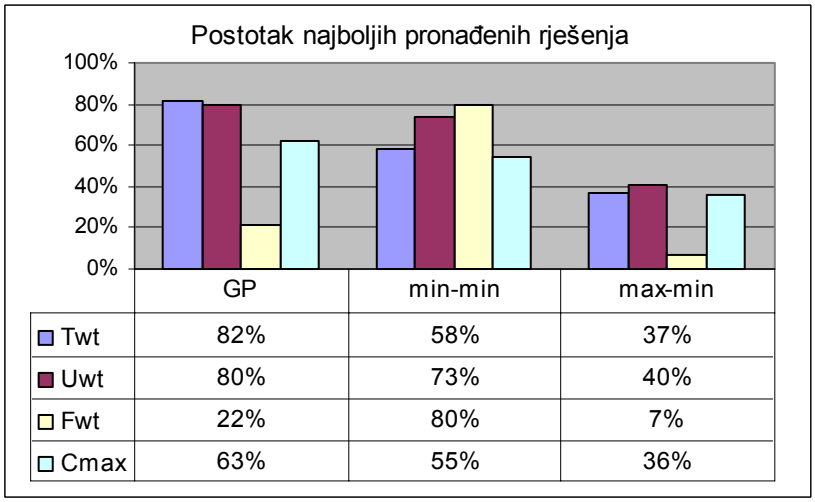
Lako je uočljivo da je rješenje dobiveno genetskim programiranjem uz kriterij ukupne duljine rasporeda kao dobrote jedinke dalo relativno lošije rješenje u usporedbi s ostalim algoritmima nego u prethodnom primjeru. Iako se prednost za promatrani kriterij (neznatno) povećala, učinkovitost algoritma je opala u ostalim prikazanim mjerilima. Iz toga se može zaključiti da je kriterij protjecanja 'isplativiji' u postupku učenja, budući daje rješenja čija je učinkovitost u usporedbi s ostalim algoritmima podjednaka po više mjerila.

Uvrštenje kriterija težinskog zaostajanja i težinske zakašnjelosti poslova u rezultatima u biti nije sasvim opravdano jer promatrani algoritmi, uključujući i onaj izveden genetskim programiranjem, ne koriste informaciju o težinama i željenim vremenima završetka poslova. Općenito je teže pronaći algoritam koji bi za ovo okruženje sastavio raspored po navedenim kriterijima. Stoga je u ovom radu, ilustracije radi, provedeno i izvođenje algoritma sa težinskim zaostajanjem kao funkcijom dobrote rješenja. U ovom su slučaju genetskom programu na raspolaganju i odgovarajući podatkovni čvorovi (težina posla, željeno vrijeme završetka, dopuštena odgoda za pojedini stroj). Za ovaj je kriterij provedeno 10 pokusa, a rezultati su na slikama 4.47 i 4.48. Rješenje genetskog programiranja prikazano je u prilogu B3.



	GP	min-min	max-min
Twt	60,6	78,9	583,8
Uwt	104,1	105,3	223,8
Fwt	332,2	298,5	810,6
Cmax	191,1	192,0	225,2

Slika 4.47 Optimiranje težinskog zaostajanja – skup primjera za ocjenu



	GP	min-min	max-min
Twt	82%	58%	37%
Uwt	80%	73%	40%
Fwt	22%	80%	7%
Cmax	63%	55%	36%

Slika 4.48 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

4.7 Raspoređivanje za proizvoljnu obradu

4.7.1 Okruženje proizvoljne obrade

Okruženje proizvoljne obrade predstavljeno je poslovima od kojih se svaki sastoji od zadanog broja operacija. Operacije se izvode predefiniranim redoslijedom na točno određenim strojevima. U najopćenitijem obliku proizvoljne obrade, svaki posao može imati proizvoljan broj operacija (različit od broja strojeva), a različite se operacije u slijedu mogu izvoditi i na istim strojevima (tzv. višestruka proizvoljna obrada, engl. *reentrant job shop*). U literaturi se ipak najčešće javlja inačica problema u kojoj je broj operacija svakog posla jednak broju strojeva, a svaki posao se izvodi na svakom stroju točno jednom. Iako se heuristike raspoređivanja bez dodatnih izmjena mogu uporabiti i u općenitijem obliku problema, poradi lakše usporedbe učinkovitosti s primjerima iz literature u ovom radu se rješava inačica u kojoj je broj operacija svakog posla jednak broju strojeva.

Raspoređivanje procesa proizvoljne obrade u pravilu se izvodi kao predodređeno raspoređivanje, pri čemu su poznati svi parametri poslova (broj operacija i njihova trajanja na pojedinim strojevima). Primjena pravila raspoređivanja, međutim, podrazumijeva dinamički način izrade rasporeda, gdje se odluka o pokretanju neke operacije na

određenom stroju donosi tijekom rada sustava. Pored navedenoga, pretpostavlja se neprekidivost operacija i dostupnost svih strojeva u cjelokupnom trajanju rada sustava.

U sljedećem odjeljku opisana su svojstva postupaka raspoređivanja za proizvoljnu obradu te neki primjeri pravila raspoređivanja. U narednim odjeljcima navedeni su postupci izvođenja pravila raspoređivanja i usporedba učinkovitosti na ispitnim primjerima.

4.7.2 Pregled raspoređivanja proizvoljne obrade uz pomoć genetskog programiranja

Prvi primjer primjene genetskog programiranja za raspoređivanje u okruženju proizvoljne obrade prikazan je 1994. godine [Atl 94], relativno brzo nakon pojave prvih radova o ovoj paradigmi. U tome radu problem se predstavlja kao multiagentska okolina u kojoj je svaki stroj predstavljen agentom. Ipak, za sve se strojeve izvodi jedinstvena funkcija prioriteta, a način raspoređivanja ne uzima u obzir dozvoljeno čekanje. Opseg pokusa je relativno mali (populacija od samo 36 jedinki), a kriterij optimiranja je ukupna duljina rasporeda. Izvedeno rješenje ima dobar učinak na ispitnim primjerima za učenje, no pokazuje slabu prilagodljivost prilikom promjene veličine problema. Rezultati nisu uspoređeni sa postojećim pravilima raspoređivanja već je prikazano odstupanje od optimalnih (ili najboljih poznatih rješenja) za određene ispitne primjere, a pokazana je bolja učinkovitost od slučajne pretrage (engl. *random search*). Autori također primjećuju da je izbor podatkovnih struktura koje su na raspolaganju genetskom programiranju ključan element za prilagodljivost izvedenog rješenja.

Slični pristup uporabljen je u radu [Miy 00], gdje je problem također predstavljen u multiagentskoj okolini, no ispitano je nekoliko inačica organizacije agenata. U najjednostavnijoj izvedbi postoji jedinstveni agent zadužen za sve strojeve, dok je u drugoj krajnosti svakom stroju pridružen poseban agent i time posebno pravilo raspoređivanja. Vrijednost rada je u prepoznavanju mogućnosti pridruživanja jednog agenta strojevima (ili stroju) koji predstavljaju usko grlo sustava i drugog agenta svim ostalim strojevima. Na takav način postignuti su najbolji rezultati, no uz predefiniranu raspodjelu agenata po sredstvima na temelju poznavanja svojstava pojedinog ispitnog primjera (tj. unaprijed je poznato koji je stroj usko grlo sustava).

4.7.3 Postupci raspoređivanja za proizvoljnu obradu

Postupcima raspoređivanja u okruženju proizvoljne obrade poklanja se puno pažnje, pa za ovaj problem postoji velik broj načina rješavanja uz pomoć različitih tehnika (evolucijske metode, ekspertni sustavi, neizrazito odlučivanje i dr.) [Kas 99, Jon 98, Cic 01]. Primarni naglasak u ovom istraživanju je na postupcima koji se mogu primijeniti u obliku pravila raspoređivanja, odnosno u dinamičkim i promjenjivim uvjetima rada. Znatan broj primjera jednostavnijih pravila raspoređivanja za ovaj problem može se naći u [Cha 96].

Prije opisa samih pravila raspoređivanja, potrebno je definirati način korištenja pravila u okruženju proizvoljne obrade. Pravilo raspoređivanja primjenjuje se kada na nekom stroju treba donijeti odluku o pokretanju sljedeće operacije. Pravilo uzima u obzir informacije o samoj operaciji, poslu kojemu ta operacija pripada i stanju na stroju prije donošenja odluke. Općeniti postupak primjene pravila prikazan je kao algoritam 4.9. Prilikom rada sustava potrebno je voditi računa o tome koja je operacija nekoga posla sljedeća na redu za izvođenje, na kojem stroju se treba izvoditi i kada će biti raspoloživa. Na temelju te informacije pojedini stroj može zaključiti koliko operacija čeka na obradu.

Svim pravilima raspoređivanja na raspolaganju su informacije o poslovima, njihovim operacijama i stanju na strojevima, a prilikom definiranja funkcija prioriteta koriste se sljedeće oznake:

- p_{ij} – trajanje jedne operacije posla j na stroju i (budući svaki posao ima broj operacija jednak broju strojeva, a informacija o redoslijedu operacija ne koristi se u pravilu raspoređivanja, nije potrebno navoditi redni broj operacije);
- w_j – težinska vrijednost posla;
- d_j – željeno vrijeme završetka posla;
- twk_j – (engl. *total work*) ukupno trajanje svih operacija nekoga posla;
- $twkr_j$ – (engl. *total work remaining*) ukupno trajanje svih preostalih operacija nekoga posla (preostale operacije su sve one čije izvođenje još nije započelo).

```
dok (postoje neizvedene operacije)
{     čekaj dok neki stroj za koji postoje operacije ne postane
          raspoloživ;
      odredi prioritete svih operacija koje čekaju na stroj;
      rasporedi operaciju najvećeg prioriteta;
      odredi sljedeću operaciju posla;
      obnovi vrijeme raspoloživosti stroja i sljedeće operacije posla;
}
```

Algoritam 4.9 Postupak raspoređivanja po pravilima za proizvoljnu obradu

U nastavku su definirana pravila raspoređivanja korištena za ocjenu učinkovitosti u ovome radu. Uz svako pravilo navedena je funkcija prioriteta kojom se odabire sljedeća operacija, uz pretpostavku da je najbolja mjera jednaka najvećoj postignutoj vrijednosti funkcije prioriteta.

- WSPT pravilo, definirano sa:

$$\pi_j = \frac{w_j}{p_{ij}}. \tag{4.43}$$

- WSPT/TWKR pravilo definirano kao:

$$\pi_j = \frac{w_j \cdot twkr_j}{p_{ij}}. \tag{4.44}$$

- WTWKR (engl. *weighted total work remaining*) pravilo, definirano sa:

$$\pi_j = \frac{w_j}{twkr_j}. \tag{4.45}$$

- SLACK/TWKR pravilo (engl. *slack per remaining process time*), definirano sa:

$$\pi_j = \frac{w_j \cdot twkr_j}{\left(d_j - twkr_j\right)}. \tag{4.46}$$

- EDD pravilo, definirano sa:

$$\pi_j = \frac{1}{d_j}. \tag{4.47}$$

- COVERT pravilo (engl. *cost over time*) [Mor 93, str. 340], definirano kao:

$$\pi_j = \frac{w_j}{p_{ij}}\left[1 - \frac{\left(d_j - ELT_{ij} - time\right)^+}{h \cdot ELT_{ij}}\right]^+, \tag{4.48}$$

  gdje je h parametar heuristike, a ELT_{ij} (engl. *estimated lead time*) označava procjenu količine vremena koju će posao j provesti u sustavu nakon završetka operacije na trenutnom stroju (i). Ova se veličina može procijeniti na nekoliko načina (poglavlje 4.1.2), no najčešće se postavlja na umnožak zadane konstante sa preostalim trajanjem obrade, tj. $ELT_{ij} = k \cdot twkr_j$. Uobičajena vrijednost za k koja se i ovdje koristi je 3, a za parametar h autori sugeriraju vrijednost 0.5 [Mor 93].
- RM pravilo za proizvoljnu obradu [Mor 93, str. 340], definirano kao:

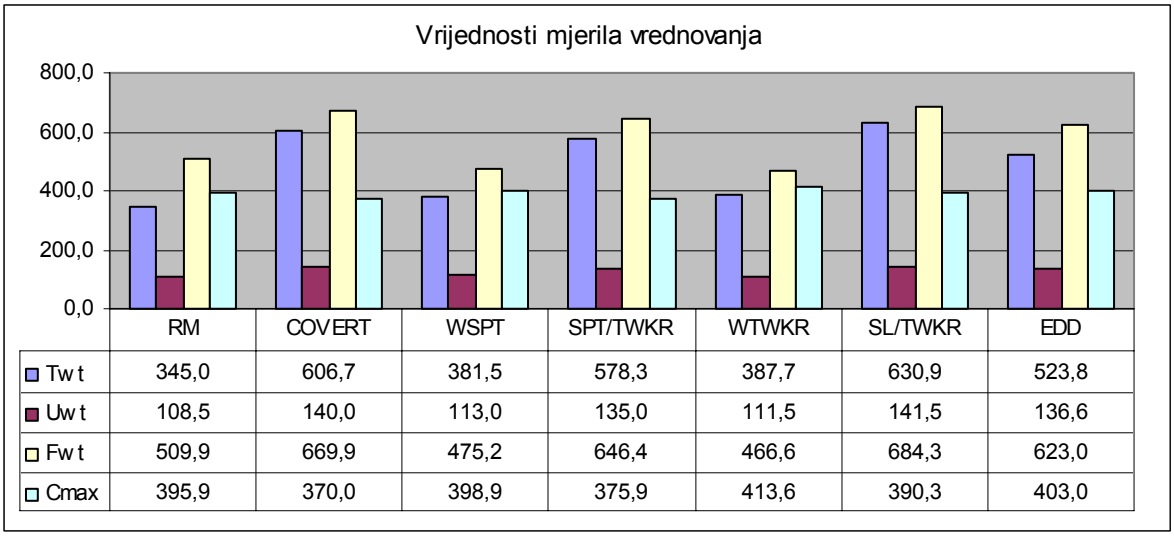
$$\pi_j = \frac{w_j}{p_{ij}} \cdot \exp\left[\frac{\left(d_j - ELT_{ij} - time\right)^+}{h \cdot \overline{p}_i}\right], \tag{4.49}$$

  gdje je $\overline{p}_i$ prosječno trajanje svih operacija na promatranom stroju, a h je parametar heuristike za koji je u ovom radu empirijski (promatrajući rezultate na ispitnim primjerima) postavljena vrijednost od 10. ELT_{ij} se računa na jednak način kao i za COVERT pravilo.

Iako se dolasci poslova u sustav u većini primjena smatraju statičkima, tj. svi su poslovi raspoloživi od početka rada, raspoređivanje na pojedinom stroju je inherentno dinamičko, budući će pojedina operacija nekoga posla biti raspoloživa za obradu tek nakon završetka svih operacija koje joj prethode. Iz tog razloga potrebno je definirati određivanje prioriteta onih operacija za koje se zna da trebaju određeni stroj, ali im je vrijeme dolaska u budućnosti. Taj se problem može riješiti na sljedeće načine:

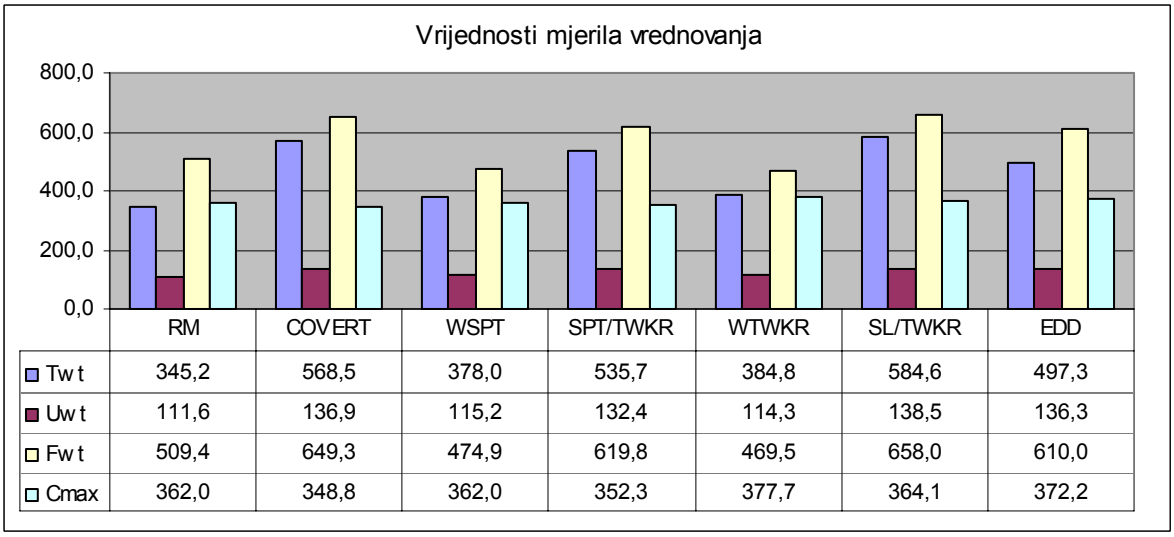
- bez dozvoljenog čekanja – ocjenjuju se i u obzir za pokretanje dolaze samo one operacije koje se odmah mogu pokrenuti, odnosno čije vrijeme pripravnosti je manje ili jednako trenutnom vremenu;
- uz dozvoljeno čekanje – kao u poglavlju 4.4.4, trajanju izvođenja operacije dodaje se količina vremena koju je potrebno pričekati do početka izvođenja;
- uz dozvoljeno čekanje s računanjem prioriteta – trajanje izvođenja ostaje nepromijenjeno, ali se pravilu raspoređivanja na raspolaganje stavlja informacija o vremenu dolaska određene operacije.

Za pravila raspoređivanja iz literature koja se u ovom radu koriste za usporedbu primjenjuje se prvi pristup, budući su rezultati na skupu ispitnih primjera uz dozvoljeno čekanje u prosjeku lošiji od onih u kojima se čekanje ne dozvoljava. Primjeri rezultata dobivenih uz prvu i drugu inačicu prikazani su na slikama 4.49 i 4.50. U postupku izvođenja pravila genetskim programiranjem primjenjuje se treći pristup, budući izvedeno pravilo ima priliku naučiti iskoristiti informaciju o vremenu dolaska operacije.



	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Twt	345,0	606,7	381,5	578,3	387,7	630,9	523,8
Uwt	108,5	140,0	113,0	135,0	111,5	141,5	136,6
Fwt	509,9	669,9	475,2	646,4	466,6	684,3	623,0
Cmax	395,9	370,0	398,9	375,9	413,6	390,3	403,0

Slika 4.49 Rezultati promatranih pravila raspoređivanja uz dozvoljeno čekanje



	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Twt	345,2	568,5	378,0	535,7	384,8	584,6	497,3
Uwt	111,6	136,9	115,2	132,4	114,3	138,5	136,3
Fwt	509,4	649,3	474,9	619,8	469,5	658,0	610,0
Cmax	362,0	348,8	362,0	352,3	377,7	364,1	372,2

Slika 4.50 Rezultati promatranih pravila raspoređivanja bez dozvoljenog čekanja

4.7.4 Oblikovanje ispitnih primjera

U okruženju proizvoljne obrade potrebno je za svaki posao definirati trajanje svake operacije i njihov redoslijed po strojevima. Trajanje pojedinih operacija generira se na način jednak oblikovanju trajanja poslova u ostalim okruženjima, odnosno po definiranim raspodjelama u intervalu od 1 do 100. Redoslijed operacija dobiva se tako da se na početku definira niz rednih brojeva strojeva, poredan po veličini. Potom se redno svaki element niza zamjenjuje sa nekim drugim slučajno odabranim elementom, nakon čega dobiveni niz predstavlja raspored operacija pojedinog posla. Očekivano ukupno trajanje poslova se za pojedini ispitni primjer definira kao:

$$\hat{p} = \frac{1}{m} \sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij} , \tag{4.50}$$

gdje je n broj poslova a m broj strojeva u dotičnom primjeru. Željena vremena završetka poslova definiraju se uz pomoć opisanih parametara T i R kao u poglavlju 4.5.2 za jednolike paralelne strojeve, po izrazima (4.35) i (4.36).

Svi ispitni primjeri podijeljeni su u skup od 160 primjera za učenje i skup od 320 primjera za ocjenu, a korišteni parametri navedeni su u prilogu A4. Osim toga, iz literature je preuzeto dodatnih 80 primjera [Tai 03] za koje se broj poslova kreće od 15 do 100, a broj strojeva je 15 ili 20. Primjeri su definirani kao statička okolina u kojoj su svi poslovi pripravni od početka rada sustava.

4.7.5 Raspoređivanje uz pomoć genetskog programiranja

U ovom se okruženju pravilo raspoređivanja može izvesti na jednak način kao i u prethodnim okolinama; drugim riječima, rješenje genetskog programiranja je funkcija prioriteta predstavljena stablom. Način primjene izvedenog pravila definiran je meta-algoritmom 4.9, s time da je moguće da odabrana operacija ima vrijeme početka u budućnosti, odnosno dozvoljeno je umetnuto čekanje. Zadatak je na samom pravilu da na pravilan način iskoristi informaciju o vremenu dolaska i zaključi isplati li se čekati na određenu aktivnost.

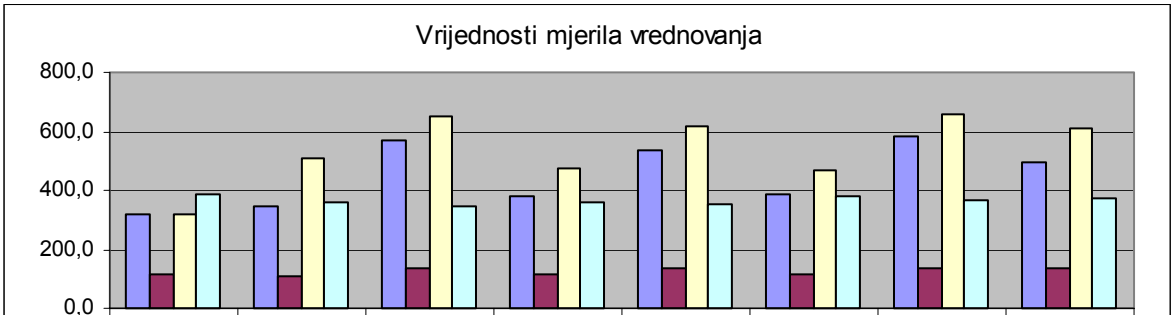
Za potrebe izvođenja pravila, genetskom su programu na raspolaganju podaci o poslu i trenutnoj operaciji, kao i neke veličine koje opisuju trenutno stanje cijeloga sustava. Popis svih funkcijskih i podatkovnih čvorova koji se mogu javiti unutar jedinke dan je u tablici 4.12.

Tablica 4.12 Popis čvorova za raspoređivanje u proizvoljnoj obradi

Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
SQR	zaštićeni unarni operator kvadratnog korijena: $SQR(a) = \begin{cases} 1, \text{ ako } a < 0 \\ \sqrt{a}, \text{ inače} \end{cases}$
IFGT	logički operator odabira: $IFGT(a,b,c,d) = \begin{cases} c, \text{ ako } a > b \\ d, \text{ inače} \end{cases}$
Oznaka podatkovnog čvora	**Definicija vrijednosti podatkovnog čvora**
pt	trajanje izvođenja operacije određenog posla na promatranom stroju (p_{ij})
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
CLK	trenutno vrijeme
AR	količina vremena do dolaska operacije; $\max\{r_{ij} - time, 0\}$, gdje se r_{ij} definira kao završetak prethodne operacije promatranog posla (prije stroja i)
NOPr	broj neobavljenih operacija posla
TWK	ukupno trajanje svih operacija posla (twk_j)
TWKr	trajanje svih neobavljenih operacija posla ($twkr_j$)
PTav	prosječno trajanje svih operacija na promatranom stroju
HTR	omjer vremena koje je posao proveo u sustavu i ukupnog trajanja do sada obavljenih operacija (po engl. *head time ratio*)

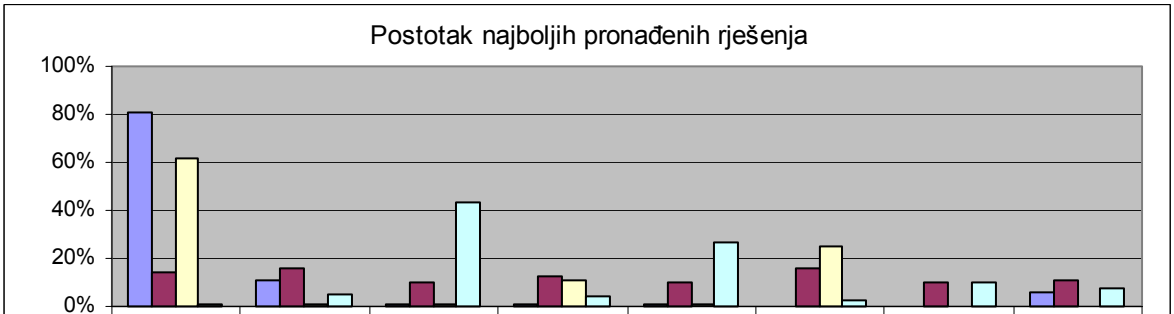
Operator 'IFGT' promatra vrijednosti prva dva argumenta (prva dva podstabla) i na temelju njihovog odnosa kao rezultat vraća vrijednost trećeg odnosno četvrtog argumenta. Budući se može očekivati da je potreba za funkcijskim čvorovima 'IFGT' i 'SQR' manja od potrebe za ostalim čvorovima, oni su uvršteni u skup uz manju vjerojatnost pojavljivanja u stablu (prilikom mutacije). Više o toj tehnici bit će riječi u poglavlju 5. Podatkovni čvor 'HTR' trebao bi pravilu raspoređivanja omogućiti procjenu o količini vremena koju će posao još provesti u sustavu, pod pretpostavkom da je opterećenje sustava podjednako prije i poslije obavljanja trenutne operacije.

Prvi dio ispitivanja proveden je za statičku okolinu, u kojoj su svi poslovi raspoloživi od početka rada sustava, uz optimiranje kriterija težinskog zaostajanja. Provedeno je 20 pokusa iz kojih je odabrano najbolje rješenje prema skupu ispitnih primjera za ocjenu. Dobivena funkcija prioriteta navedena je u prilogu B4, a usporedni rezultati prikazani su na slikama 4.51 i 4.52.



	GP	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Tw t	316,1	345,2	568,5	378,0	535,7	384,8	584,6	497,3
Uw t	115,8	111,6	136,9	115,2	132,4	114,3	138,5	136,3
Fw t	319,2	509,4	649,3	474,9	619,8	469,5	658,0	610,0
Cmax	386,8	362,0	348,8	362,0	352,3	377,7	364,1	372,2

Slika 4.51 Optimiranje težinskog zaostajanja – skup primjera za ocjenu

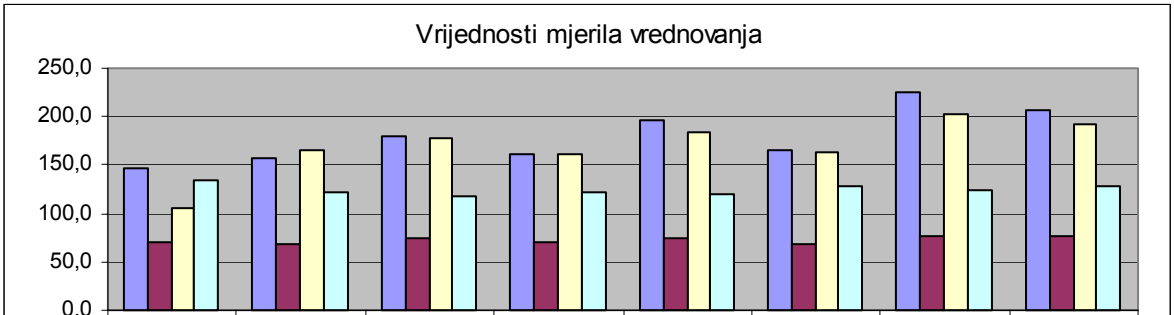


	GP	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Tw t	81%	10%	1%	1%	1%	0%	0%	6%
Uw t	15%	16%	10%	12%	10%	16%	10%	11%
Fw t	62%	1%	1%	11%	1%	25%	0%	0%
Cmax	1%	5%	44%	4%	27%	3%	10%	7%

Slika 4.52 Postoci dominacije za težinsko zaostajanje – skup primjera za ocjenu

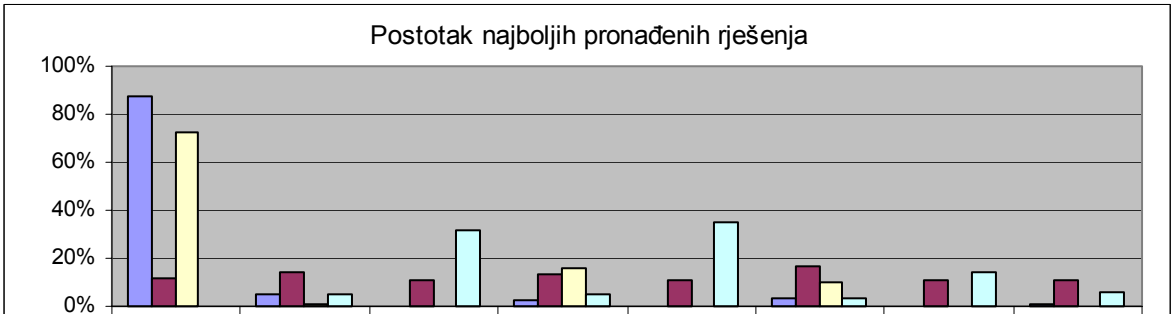
Iz rezultata je vidljivo da je pravilo izvedeno genetskim programiranjem najveću relativnu prednost postiglo za kriterije težinskog zaostajanja i težinskog protjecanja, dok istovremeno daje najlošije rezultate za ukupnu duljinu rasporeda. Po pitanju ukupne duljine rasporeda najveću učinkovitost pokazuje COVERT pravilo, što pomalo iznenađuje

budući je to pravilo, po autorima, namijenjeno optimiranju težinskog zaostajanja. Isto rješenje dobiveno genetskim programiranjem uspoređeno je sa ostalim pravilima na skupu od 80 primjera iz literature, za koji su rezultati prikazani na slikama 4.53 i 4.54.



	GP	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Twt	146,1	158,0	179,8	161,6	195,5	166,1	225,5	206,1
Uwt	70,9	68,5	73,4	69,3	74,6	68,4	77,0	76,5
Fwt	105,8	165,0	178,5	161,9	184,8	163,5	201,5	191,9
Cmax	133,5	121,6	118,7	121,7	118,9	127,8	123,7	128,0

Slika 4.53 Optimiranje težinskog zaostajanja – skup primjera iz literature



	GP	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Twt	88%	5%	0%	2%	0%	4%	0%	1%
Uwt	12%	14%	11%	13%	11%	17%	11%	11%
Fwt	73%	1%	0%	16%	0%	10%	0%	0%
Cmax	0%	5%	31%	5%	35%	4%	14%	6%

Slika 4.54 Postoci dominacije za težinsko zaostajanje – skup primjera iz literature

Iz ovih se rezultata može uočiti da je relativni odnos pravila po pitanju učinkovitosti ostao gotovo nepromijenjen u odnosu na prethodni skup primjera. Na temelju toga mogli bi se izvesti zaključci o dobroj prilagodbi izvedenog pravila; no s obzirom na relativno mali broj ispitnih primjera u ovome skupu, za takve tvrdnje potrebna su dodatna ispitivanja uz promijenjene uvjete raspoređivanja.

Raspoređivanje uz dinamičko prepoznavanje uskog grla sustava

Određeni broj heurističkih algoritama temelji se na identifikaciji eventualnog uskog grla sustava i ugrađivanju te informacije u postupak raspoređivanja. Prethodni su radovi iz područja raspoređivanja uz pomoć genetskog programiranja također pokazali korisnost ovoga pristupa. Kod primjene pravila raspoređivanja, odnosno općenito u dinamičkim uvjetima, problem je otkrivanje postojanja uskog grla sustava i njegova identifikacija. U općenitom slučaju teško je egzaktno definirati što neko sredstvo čini uskim grlom, jer to ovisi o ukupnom opterećenju i načinu rada cijeloga sustava. Stoga se u uporabi mogu naći postupci raspoređivanja koji su zapravo kombinacije nekoliko pravila,

a primjena određenog pravila uvjetovana je trenutnim stanjem sustava i nekom definiranom logikom (npr. 'primijeni EDD u uvjetima manjeg, a WSPT u uvjetima većeg opterećenja').

U okruženju proizvoljne obrade usko grlo sustava može biti jedan ili više strojeva, s time da je podjela strojeva na one koji jesu i one koji nisu usko grlo sasvim uvjetna. Isto tako, opterećenje strojeva, a time i njihova uloga, može se mijenjati tijekom rada sustava. Stoga bi bilo korisno definirati metodu uz pomoć koje bi postupak raspoređivanja mogao prepoznati povećane zahtjeve za nekim strojem i primijeniti odgovarajuće pravilo. Moguće je zamisliti definiranje nekog posebnog pravila koje bi za svaki stroj određivalo njegov stupanj opterećenja i potom primijenilo određeni algoritam raspoređivanja.

U ovom se radu predlaže rješenje koje se temelji na tom pristupu. Postupak raspoređivanja sastoji se od tri pravila: jedno je zaduženo za razlučivanje strojeva na dvije skupine, strojeve koji u danom trenutku predstavljaju usko grlo sustava i one koji to nisu. Po jedno pravilo raspoređivanja definirano je za obje opisane kategorije. Definicija sva tri pravila raspoređivanja prepuštena je genetskom programiranju – jedna jedinka populacije predstavlja tri funkcije (u obliku tri stabla), od kojih svaka ima posebnu zadaću. U takvoj arhitekturi prvo stablo koristi se za odluku o primjeni drugog ili trećeg stabla kod pojedinog slučaja raspoređivanja. Prilikom svakog novog raspoređivanja neke operacije na nekom stroju, prvo se odlučuje koje pravilo raspoređivanja treba primijeniti, te se potom dotično pravilo rabi za odabir sljedeće operacije.

U drugom i trećem stablu rješenja (stabla raspoređivanja), koriste se isti čvorovi kao i u tablici 4.12. Za prvo stablo (stablo odluke), potrebno je definirati podatkovne čvorove koji bi omogućili prepoznavanje povećane potrebe za nekim strojem. U tablici 4.13 definiran je skup mogućih podatkovnih čvorova koji se javljaju samo u stablu odluke, dok su funkcijski čvorovi jednaki onima u stablima raspoređivanja.

Tablica 4.13 Popis podatkovnih čvorova za stablo odluke

Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
MTWK	ukupno trajanje svih operacija na promatranom stroju
MTWKr	trajanje svih neizvedenih operacija na stroju
MTWKav	prosječno trajanje svih operacija za sve strojeve
MNOPr	broj neizvedenih operacija na stroju
MNOPw	broj operacija koje trenutno čekaju na stroj
MUTL	omjer trajanja svih do sada izvedenih operacija na stroju i trenutnog vremena (po engl. *utilization*)

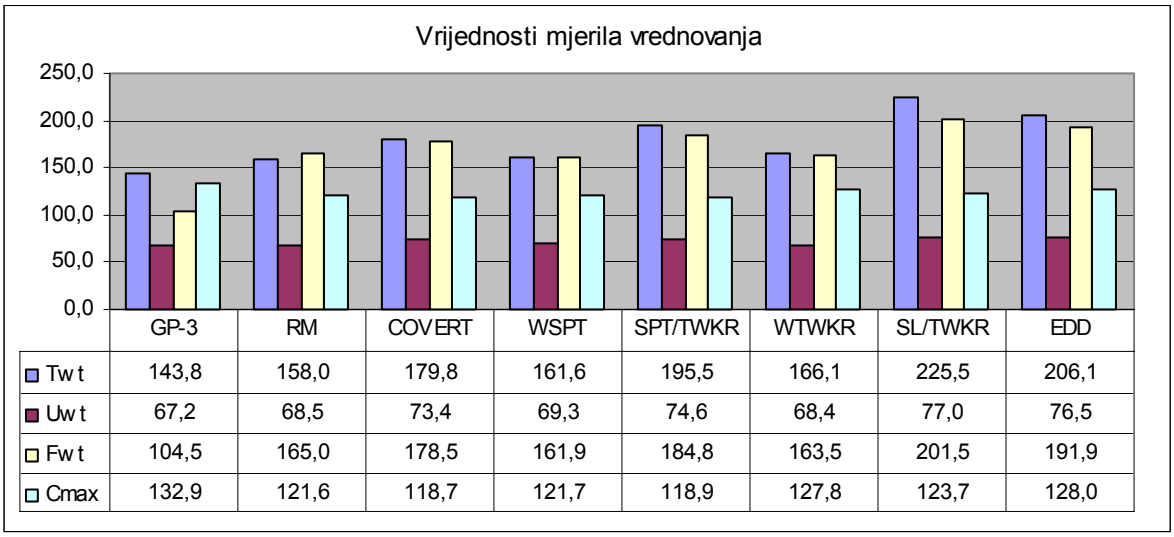
Stablo odluke daje rezultat u obliku brojčane vrijednosti, pa je potrebno definirati način na koji se koristi informacija iz stabla odluke i primjenjuje jedno od dva stabla raspoređivanja. Takav se postupak može nazvati meta-algoritmom za ovu strukturu rješenja, a prikazan je kao algoritam 4.10.

```
za (svaki stroj i)
      P_i = vrijednost stabla odluke;
dok (postoje neizvedene operacije)
{     čekaj dok neki stroj i za koji postoje operacije ne postane
         raspoloživ;
      P_i = vrijednost stabla odluke;
      ako (P_i > P_m, ∀m)
         primijeni drugo stablo raspoređivanja za računanje prioriteta;
```

```
        inače
            primijeni prvo stablo raspoređivanja za računanje prioriteta;
        odredi prioritete svih operacija koje čekaju na stroj;
        rasporedi operaciju najvećeg prioriteta;
        odredi sljedeću operaciju posla;
        obnovi vrijeme raspoloživosti stroja i sljedeće operacije posla;
}
```

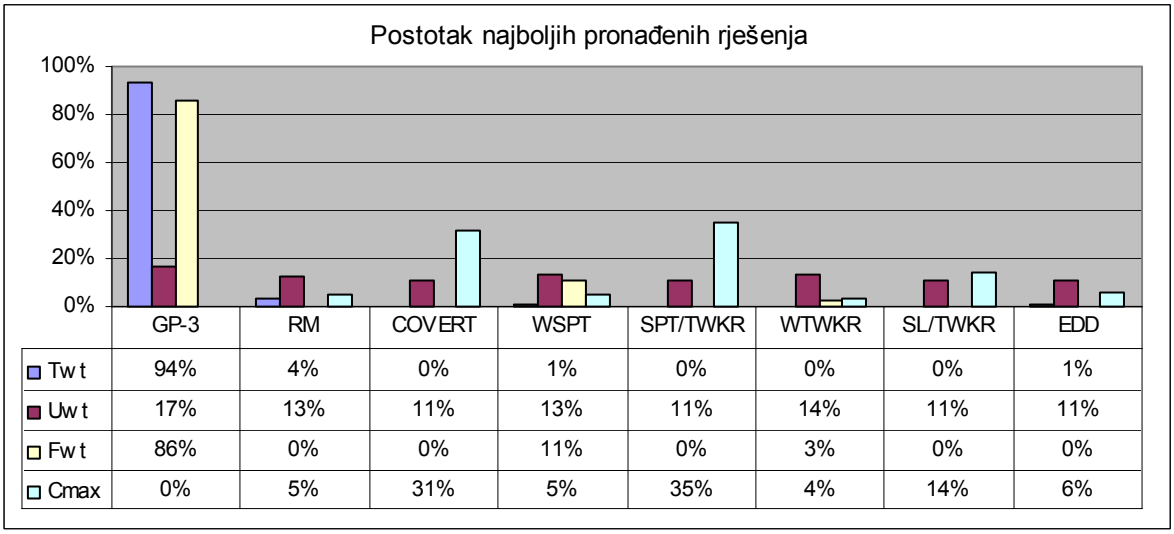
Algoritam 4.10 Meta-algoritam raspoređivanja uz pomoć stabla odluke

Prilikom rada sustava, za svaki novi slučaj raspoređivanja računa se trenutna vrijednost stabla odluke za stroj na kojemu se raspoređivanje obavlja. Ako je vrijednost stabla odluke toga stroja (P_i) veća od trenutnih vrijednosti svih ostalih strojeva, za računanje prioriteta operacija primjenjuje se drugo stablo raspoređivanja. U protivnom, prioriteti se računaju po prvom stablu (odabir jednog ili drugog stabla za pojedinu namjenu je nebitan). Ispitivanje učinkovitosti opisanog postupka raspoređivanja načinjeno je jednako ispitivanju rješenja s jednim stablom. Za potrebe učenja provedeno je 20 pokusa, a najbolje dobiveno rješenje prikazano je u prilogu B4. Kvaliteta rješenja uspoređena je sa ostalim pravilima raspoređivanja na skupu ispitnih primjera iz literature, a rezultati su prikazani na slikama 4.55 i 4.56 (oznaka dobivenog rješenja je 'GP-3').



	GP-3	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Twt	143,8	158,0	179,8	161,6	195,5	166,1	225,5	206,1
Uwt	67,2	68,5	73,4	69,3	74,6	68,4	77,0	76,5
Fwt	104,5	165,0	178,5	161,9	184,8	163,5	201,5	191,9
Cmax	132,9	121,6	118,7	121,7	118,9	127,8	123,7	128,0

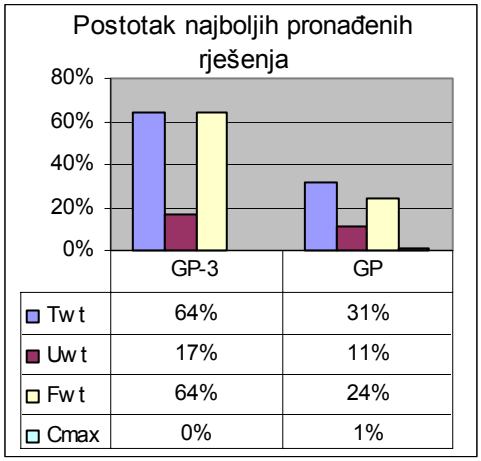
Slika 4.55 Optimiranje težinskog zaostajanja – skup primjera iz literature



	GP-3	RM	COVERT	WSPT	SPT/TWKR	WTWKR	SL/TWKR	EDD
Twt	94%	4%	0%	1%	0%	0%	0%	1%
Uwt	17%	13%	11%	13%	11%	14%	11%	11%
Fwt	86%	0%	0%	11%	0%	3%	0%	0%
Cmax	0%	5%	31%	5%	35%	4%	14%	6%

Slika 4.56 Postoci dominacije za težinsko zaostajanje – skup primjera iz literature

Iz rezultata se može uočiti da je relativna učinkovitost ovoga načina raspoređivanja, u usporedbi sa ostalim promatranim pravilima, jednaka učinkovitosti pravila s jednim stablom. Postupak sa stablom odluke je ipak postigao nešto bolje rješenje po pitanju apsolutnih vrijednosti mjerila vrednovanja. Zanimljivo je pogledati izravnu usporedbu dominacije u skupu ispitnih primjera za oba postupka dobivena genetskim programiranjem. Na slici 4.57 prikazani su međusobni odnosi rješenja sa jednim (oznaka 'GP') i rješenja sa tri stabla (oznaka 'GP-3') po pitanju postotka najboljih pronađenih rješenja.



	GP-3	GP
Twt	64%	31%
Uwt	17%	11%
Fwt	64%	24%
Cmax	0%	1%

Slika 4.57 Usporedni prikaz postotaka dominacije

Može se zaključiti da podijeljena struktura postupka raspoređivanja ima mogućnosti iskoristiti dodatnu informaciju iz stabla odluke kako bi prilagodila postupak raspoređivanja trenutnim uvjetima na dotičnom stroju. Potrebno je napomenuti da dobivanje rješenja sa tri stabla zahtijeva (očekivano) više procesorskog vremena od evolucijskog procesa sa samo jednim stablom. Budući se proces izvođenja pravila ipak odvija nevezano uz rad sustava koji se raspoređuje, isplativo je utrošiti nešto više vremena za dobivanje kvalitetnijeg rješenja.

4.8 Metodologija izvođenja pravila raspoređivanja

4.8.1 Izvođenje pravila raspoređivanja za proizvoljnu okolinu

U ovom su poglavlju prikazani primjeri stvaranja postupaka raspoređivanja za nekoliko raznovrsnih okruženja. Izloženi se postupci nužno razlikuju po načinu primjene, no također pokazuju i neka zajednička svojstva. Raspoređivanje je problem za koji je nemoguće definirati metodu izrade algoritma koji bi se mogao uspješno primijeniti u svakom postojećem primjeru. Određenim se postupcima problem sastavljanja odgovarajućeg algoritma može u određenoj mjeri ipak olakšati. Metodologija izrade algoritma raspoređivanja uporabljena u ovome radu koristi postupke raspoređivanja u obliku pravila. Pravilo je u užem smislu predstavljeno funkcijom kojom se određuju prioriteti elemenata sustava, a način uporabe te funkcije definiran je meta-algoritmom koji je zajednički za sva pravila u određenoj okolini. Korištena metodologija može se stoga sažeto opisati sa tri osnovna elementa: meta-algoritmom pravila, prioritetnom funkcijom i postupkom učenja.

Meta-algoritam određuje ponašanje pravila u dotičnoj okolini. Isti oblik meta-algoritma može se primjenjivati uz više različitih prioritetnih funkcija u zadanom okruženju. Za većinu okruženja meta-algoritam je jednostavan i ne navodi se posebno u opisu postupka raspoređivanja (npr. "sustav se raspoređuje SPT pravilom"). U određenim

situacijama, međutim, oblik meta-algoritma ne podrazumijeva se sam po sebi i potrebno ga je posebno definirati. Ukoliko se susrećemo sa okruženjem koje nije obuhvaćeno klasičnom teorijom raspoređivanja, meta-algoritmom određujemo okvir rada pravila raspoređivanja. Na ovaj se način vrlo učinkovito može definirati struktura raspoređivanja u okruženjima u kojima je potrebno zadovoljiti neka specifična ograničenja, primjerice nemogućnost izvođenja nekih aktivnosti na određenim sredstvima, postupanje u slučaju nepredviđenih okolnosti i sl. Ovim pristupom olakšavamo problem onome dijelu algoritma koji je zadužen za donošenje odluka – prioritetnoj funkciji, a to je dio koji se može definirati automatski.

Funkcija koja određuje prioritete elemenata sustava ne mora 'znati' kakve su izravne posljedice veće ili manje vrijednosti nekog elementa koji se ocjenjuje, budući raspoređivanjem upravlja meta-algoritam. Ukoliko sami definiramo prioritetnu funkciju, takvo je znanje neophodno, no ne i u slučaju izvođenja funkcije nekom metodom strojnog učenja. Kako bi prioritetna funkcija mogla davati valjane rezultate, na raspolaganju mora imati što manji ali dovoljan skup funkcijskih i podatkovnih elemenata. Upravo odabir ovih elemenata ima najveći utjecaj na kvalitetu ne samo prioritetne funkcije nego i na cjelokupni postupak raspoređivanja.

U postupku učenja procjenjuje se dobrota mogućih oblika funkcije prioriteta na zadanom skupu ispitnih primjera. Postupak učenja ne mora biti svjestan posljedica svake odluke proizašle iz uporabe određenog rješenja, što u primjeni i nije uvijek moguće postići. Ono što je potrebno je barem jedno mjerilo na osnovu kojega se različita rješenja mogu međusobno uspoređivati. U tome je procesu odabir odgovarajućih ispitnih primjera izuzetno važan element. Uspješno provedeni postupak učenja dat će rezultat koji se najbolje prilagođava skupu ispitnih primjera, a ne nekom općenitijem problemu čije rješenje uistinu tražimo. Zbog toga u definiranju ispitnih primjera te u zaključivanju o općenitosti dobivenog rješenja valja biti posebno oprezan. U literaturi postoji puno primjera koji služe ispitivanju *učinkovitosti* razvijenih algoritama, no ne postoje jamstva da su isti primjeri dobri i za postupak *učenja* algoritma. Pronalaženje kvalitetnih primjera za učenje, pogotovo uz ovako veliku raznovrsnost mogućih okruženja, zahtjevna je i opširna tema koja predstavlja zasebno područje istraživanja.

4.8.2 Osvrt na predložene metode

Na početku svakog potpoglavlja vezanog uz određeno okruženje navedeni su dosadašnji objavljeni radovi, ako postoje, u kojima se istražuje slična problematika. U odnosu na prethodne radove, u ovom su radu promatrane mnogobrojne inačice problema i uvedene nove tehnike rješavanja. Za svaku su ispitnu okolinu detaljno definirani ispitni primjeri te podjela istih na skup za učenje i skup za ocjenjivanje. Ispitni primjeri uzimaju u obzir značajke dotičnog problema i omogućuju rješavanje po više kriterija, što uz neke primjere iz literature nije uvijek moguće (na primjer, nisu definirana željena vremena završetka poslova ili nije predviđeno proširenje za težinsku inačicu problema). Opisani načini tvorbe ispitnih primjera za različite okoline mogu biti iskorišteni i za ispitivanje učinkovitosti drugih algoritama uz proizvoljne kriterije.

Većina kriterija kvalitete rasporeda implicitno uključuje svojstva ispitnog primjera, kao što su broj i trajanje poslova, u iznosu funkcije cilja. Postupak učenja na skupu primjera različitih svojstava može uzrokovati veću sklonost algoritma određenom dijelu primjera koji u najvećoj mjeri doprinose ukupnoj kvaliteti (npr. primjeri sa većim brojem poslova). Zbog toga se u definiciji mjerila vrednovanja uzimaju u obzir značajke dotičnog ispitnog primjera i na taj način postiže ujednačena učinkovitost algoritma. Za potrebe ispitivanja kvalitete dobivenog rješenja ostvaren je veći broj postojećih algoritama. U svakom okruženju definiran je odgovarajući način primjene, a algoritmi su prilagođeni

specifičnim uvjetima raspoređivanja kad god je to bilo moguće (npr. uz postojanje trajanja postavljanja), kako bi se ostvarila što ravnopravnija usporedba sa predloženim rješenjima.

Dosadašnji radovi na ovom području ne uzimaju u obzir dinamičku okolinu raspoređivanja, tj. onu u kojoj je uz svaki posao vezano i vrijeme pripravnosti posla. U ovom su radu definirani prikladni podatkovni elementi koji omogućuju procjenu isplativosti čekanja sredstva na neki budući posao, odnosno čekanja posla na neko sredstvo koje nije trenutno raspoloživo. Za dinamičku okolinu jednoga stroja definiran je i dodatni meta-algoritam koji se koristi u sprezi sa izvedenim pravilom raspoređivanja. Pokazano je da kombinacija meta-algoritma i odgovarajućeg pravila može iskoristiti informacije iz sustava i dodatne mogućnosti raspoređivanja u postizanju kvalitetnijeg rješenja. Osim vremena pripravnosti, promatrani su i uvjeti uz trajanja postavljanja i ograničenja u rasporedu, za koje su definirani odgovarajući građevni elementi na raspolaganju genetskom programiranju.

U radu su provedena ispitivanja u okruženju paralelnih strojeva za koja do danas ne postoje istraživanja uz uporabu genetskog programiranja. Pokazano je da se kvalitetna rješenja za jednolike strojeve mogu dobiti svođenjem problema na okruženje jednoga sredstva. Prilikom raspoređivanja za nesrodne strojeve, primijenjeno je raspoređivanje na zahtjev uz koje nisu poznati podaci o budućnosti sustava. Budući se problem ne može učinkovito riješiti svođenjem na jedno sredstvo, definiran je meta-algoritam koji uz odgovarajuće pravilo raspoređivanja može iskoristiti raspoložive informacije u postupku izrade rasporeda. Opisani meta-algoritam može se koristiti uz proizvoljno pravilo odnosno funkciju prioriteta u užem smislu.

Okruženje proizvoljne obrade nerijetko se promatra kao višeagentski sustav u kojemu je svakom stroju pridružen po jedan agent. Raspoređivanje je u ovome okruženju moguće definirati tako da svaki agent definira vlastito pravilo raspoređivanja, no kako su u promatranom problemu dozvoljeni primjeri sa različitim brojem strojeva (do 20 u ovome radu), ovaj pristup je neprikladan za učenje i neopćenit. Stoga se u radu predlaže struktura rješenja koja uzima u obzir trenutno stanje na promatranom stroju i primjenjuje odgovarajući postupak. Kriterij izbora 'odgovarajućeg' postupka također se izvodi s pomoću genetskog programiranja, za što su programu na raspolaganje dani prikladni podatkovni elementi. Prilikom izrade rasporeda dopuštena je uporaba umetnutog čekanja (na operacije koje će postati raspoložive u budućnosti), no postupak se može definirati i za slučaj u kojemu to zbog postojećih uvjeta nije omogućeno.

5 Poboljšavanje postupaka izvođenja pravila raspoređivanja

U prethodnom poglavlju pokazani su primjeri izvođenja pravila raspoređivanja uz pomoć genetskog programiranja, a njihova je učinkovitost uspoređena s postojećim pravilima za dotično okruženje. U postupku pronalaženja najboljeg rješenja, gdje rješenje predstavlja definiciju pravila raspoređivanja, korišteni su predefinirani parametri evolucijskog procesa i postojeći genetski operatori. Naglasak ovoga rada nije na ispitivanju utjecaja koji različite inačice evolucijskog procesa (npr. generacijski ili eliminacijski odabir) imaju na uspješnost pronalaženja zadovoljavajućeg rješenja, budući je to područje samo po sebi dostatno za samostalno istraživanje. Isto tako, kvaliteta dobivenih rješenja opravdava opisanu metodologiju bez obzira na odabrane parametre evolucijskog procesa genetskog programiranja. Prilikom provedbe pokusa i u postupku osmišljavanja rješenja za opisane probleme, pokazalo se međutim da neki elementi u sustavu genetskog programiranja zaslužuju posebnu pažnju jer mogu u velikoj mjeri utjecati na konačni rezultat. Slična zapažanja javljaju se i u drugim radovima na temu primjene genetskog programiranja, bez obzira na domenu dotičnog problema.

Problem koji je u primjeni najizraženiji je definiranje podatkovnih struktura, odnosno čvorova stabla za većinu modela, koji će genetskom programu biti na raspolaganju [Kin 93, Wan 03, Che 99]. Ovaj je problem, na ovaj ili onaj način, ugrađen u sve oblike strojnog učenja [Koz 92] i to je element rješenja koji u najvećoj mjeri zahtijeva znanje o specifičnom problemu. Odabir odgovarajućih podatkovnih elemenata ključan je zbog svojstva potpunosti koje je potrebno za pronalaženje rješenja nekoga problema. U većini slučajeva, međutim, potpunost podatkovnih struktura nije svojstvo koje može poprimiti samo dvije vrijednosti (ili postignuto ili ne), već se za određeni sustav može reći i da je djelomično ili nedovoljno potpun. Primjerice, za algoritam raspoređivanja informacija o trenutnom vremenu u sustavu ključna je kako bi algoritam mogao procijeniti žurnost ili kašnjenje nekoga posla. Iako je u ovom primjeru odluka o uključivanju takve informacije trivijalna, u općenitom slučaju nije jednostavno odrediti značaj nekog podatka u definiranju rješenja.

Drugi često oslovljavani problem je pitanje vrijednosti različitih parametara evolucijskog procesa, koji u znatnoj mjeri utječu na kvalitetu rješenja. Ovaj se problem prepoznaje i u drugim oblicima evolucijskog računanja, od kojih su genetski algoritmi najbliži promatranoj paradigmi [Hin 97, Nov 03]. Odabir vrijednosti većine parametara vođen je prethodnim iskustvom istraživača ili opažanjima iz literature, no kvalitetne vrijednosti parametara u jednoj domeni mogu dovesti do lošijih rezultata u nekoj drugoj okolini. Budući je genetsko programiranje jedna od novijih inačica evolucijskih postupaka, ovom se pitanju nije dosada poklanjalo previše pažnje.

Proces genetskog programiranja rukuje strukturama promjenjive veličine i oblika. Prilikom traženja rješenja nekog problema, veličina koju bi neko zadovoljavajuće rješenje trebalo imati nije unaprijed poznata. Mogućnost istraživanja rješenja različitih oblika i veličina ono je što daje prednost genetskom programiranju u primjeni. Međutim, upravo svojstvo promjenjive veličine može predstavljati jedan od problema u radu genetskog programa, kao što je spomenuto u 3. poglavlju. Nepostojanje ograničenja veličine rješenja, koje je s jedne strane korisno po pitanju općenitosti, može biti uzrokom ubrzanog rasta jedinki, usporenja procesa pretraživanja i smanjenja preglednosti dobivenih rješenja. Stoga se u praktičnoj primjeni javlja niz postupaka kojima je cilj ograničavanje nekontroliranog

rasta veličine rješenja, ali uz istovremeno ostavljanje dovoljno slobode evolucijskom procesu, što su gotovo uvijek oprečni zahtjevi.

U nastavku ovoga poglavlja pobliže su opisani neki od navedenih problema te je dan kratak pregled njihova dosadašnjeg rješavanja. Za pojedine probleme predložena su i moguća rješenja kojima je cilj postizanje veće učinkovitosti postupka bez obzira na područje primjene.

5.1 Odabir podatkovnih elemenata rješenja

5.1.1 Problem potpunosti

Uspješnost genetskog programiranja izravno je povezana sa oblikovanjem skupa funkcijskih i podatkovnih čvorova. U najgorem slučaju, ako funkcijski i podatkovni elementi nisu dovoljni za izražavanje rješenja promatranog problema, odnosno skup elemenata nema svojstvo potpunosti, genetskim programiranjem ne može se riješiti problem (isto vrijedi i za bilo koju drugu paradigmu strojnog učenja). U općenitom slučaju, pogotovo u problemima optimiranja, ne postoji jasno razgraničenje između 'potpunog' i 'nepotpunog' skupa. Uz 'manje potpuni' skup funkcija i podataka, dobiveno rješenje bit će lošije nego uz 'potpuniji' skup. S druge strane, skup u kojemu se nalaze podaci ili funkcije koje nisu nužne za pronalaženje rješenja može uzrokovati lošije ponašanje genetskog programiranja: manju mogućnost pronalaženja rješenja i općenito slabiju kvalitetu dobivenog rješenja. U većini primjena se ipak (opravdano) smatra kako je bolje u skup funkcija i podataka staviti i elemente za koje nismo posve sigurni da su nužni za rješenje problema: "Ukoliko nije poznato kakav je skup funkcija dovoljan za opis rješenja, općenito je bolje uključiti potencijalno nepotrebne funkcije nego uopće ne pronaći rješenje" ([Koz 92]).

Uvođenje 'nepotrebnih' funkcija i/ili podataka stvara puno veći prostor rješenja koji postupak učenja mora pretražiti i time uzrokuje slabije rezultate rada genetskog programa. U evolucijskom procesu brojnost čvorova koji ne utječu na vrijednost jedinke s vremenom se smanjuje, pa dobiveno rješenje obično ne uključuje elemente koji nisu relevantni za promatrani problem. No u velikom broju primjera, pogotovo u slučajevima relativno velikog broja različitih funkcijskih i podatkovnih elemenata, prepoznavanje bitnih elemenata za specifičnu primjenu predstavlja dodatni problem. Stoga je poželjno na neki način pomoći genetskom programu u prepoznavanju podskupa onih funkcija i podataka uz koje je predstavljanje rješenja moguće.

Dosadašnji napori u rješavanju toga problema na području genetskog programiranja nisu brojni; zapravo, tek se u jednom članku [Ok 00] obrađuje ova tematika. U navedenom radu autori predlažu postupak prepoznavanja bitnih podatkovnih čvorova u stablastom prikazu rješenja genetskog programiranja. Postupak se temelji na težinskom ocjenjivanju svojstava (engl. *feature weighting*) koje se i inače koristi u drugim paradigmama strojnog učenja. Osnovu rada predstavlja sljedeći izraz uz pomoć kojega se računaju težinske vrijednosti podatkovnih čvorova:

$$w_n(g) = \sum_{i \in S} \left(f_i(g) \cdot freq_{i,n}(g) \right) + w_n(g-1), \tag{5.1}$$

gdje je $w_n(g)$ težina čvora n u generaciji g, $f_i(g)$ je funkcija dobrote rješenja i u generaciji g, $freq_{i,n}(g)$ je učestalost čvora n u rješenju i, a S je podskup od 10% najboljih jedinki u trenutnoj generaciji. Težinske vrijednosti se računaju kumulativno budući da se nova vrijednost dodaje vrijednosti iz prethodne generacije. Računanje

težinske vrijednosti obavlja se samo za čvorove sadržane u 10 % najboljih jedinki populacije.

Izračunate težinske vrijednosti koriste se prilikom mutacije, gdje je vjerojatnost odabira pojedinih čvorova proporcionalna njihovoj težinskoj vrijednosti. Za potrebe izražavanja vjerojatnosti, težinske vrijednosti su normirane tako da je zbroj težina svih čvorova jednak 1. Osim toga, algoritam provjerava i pojavljivanje čvorova u najboljih 1% članova populacije: ako se unutar tih jedinki nalaze samo čvorovi iz definiranog podskupa čvorova sa najvećim težinskim vrijednostima, cijeli evolucijski postupak započinje ponovno (uz slučajno generiranje svih jedinki), ali korištenjem samo odabranih najboljih čvorova. Učinkovitost postupka prikazana je na dva problema usporedbom pokusa sa i bez težinskog ocjenjivanja čvorova. Autori su za potrebe ispitivanja rabili probleme minimiziranja u kojima je funkcija dobrote predstavljena odstupanjem od idealnog rješenja (manje je bolje), a izraz (5.1) definira veću težinu za jedinke sa većim iznosom funkcije dobrote, što je suprotno kvaliteti rješenja. U tekstu se, nažalost, nigdje ne definira očito korišteno preslikavanje originalne ocjene rješenja u funkciju dobrote uporabljenu u gornjem izrazu.

U nastavku je izložen komentar na opisani postupak i predložena alternativna metoda vrednovanja bitnih elemenata u skupu mogućih čvorova.

5.1.2 Metoda vrednovanja podatkovnih čvorova

Prilikom ocjenjivanja učinkovitosti prethodno opisanoga postupka, autori su uporabili probleme u kojima je u skup podatkovnih čvorova dodan veći broj elemenata koji ni na koji način ne utječu na funkciju dobrote jedinke. Drugim riječima, skup podataka bio je strogo podijeljen na bitne (one koji doprinose dobroti) i nebitne elemente. Genetskom je programu, naravno, ta informacija bila uskraćena. Ova vrsta problema posebna je upravo po tome što se za bilo koji podatkovni čvor može sa apsolutnom sigurnošću reći je li bitan za funkciju dobrote ili ne. Za veliki broj primjena, međutim, čak se i uz poznavanje prirode problema ne može za pojedini čvor reći je li apsolutno bitan ili posve nebitan u gradnji rješenja. Prikladnija ocjena značaja nekog elementa mogla bi biti da je manje bitan ili više bitan od nekoga drugoga. U većini slučajeva za tvorbu kvalitetnog rješenja važan je omjer značaja pojedinih čvorova više nego odluka o isključivanju ili uključivanju nekoga čvora u proces. Primjerice, u slučaju određivanja neke funkcije koja vraća brojčanu vrijednost, može se pretpostaviti da bi udio čvorova osnovnih računskih operacija trebao biti relativno veći od udjela čvorova odluke ('IF' funkcija). Isto tako, određeni čvor za kojega je poznato da bi trebao biti dijelom rješenja može u stohastičkom postupku evolucije 'izumrijeti', pa je ostatak procesa zapravo beskoristan [Lan 98]. Stoga nije uvijek najisplativije sasvim isključiti neki čvor iz skupa na temelju pokazatelja koji su, u osnovi, stohastički podaci.

Operator mutacije je najčešće jedini način kojim se može izvana utjecati na brojnost čvorova u populaciji. Budući je prosječan broj čvorova po jedinki u postupku evolucije uglavnom u stalnom porastu, utjecaj mutacije najveći je na samom početku procesa, kada je ukupan broj čvorova relativno mali. Iz tog je razloga važno već na samome početku pokušati pozitivno utjecati na postizanje 'dobrog' omjera čvorova u populaciji. Vrijednost 'dobrog' omjera vrlo je teško odrediti na početku kada je utjecaj slučajnosti najjači i kada i trenutno najbolje jedinke mogu dugoročno predstavljati potpuno krivi smjer pretraživanja. Zbog toga valja biti oprezan prilikom određivanja podskupa 'najboljih' jedinki, pogotovo ako se na temelju toga podskupa donosi globalna ocjena značaja pojedinih podatkovnih elemenata.

Na temelju iznesenih opažanja, kao i empirijskih pokazatelja dobivenih iz većeg broja pokusa, predložena je metoda vrednovanja podatkovnih čvorova kojoj je cilj

povećavanje vjerojatnosti pronalaženja kvalitetnog rješenja i ubrzanje postupka učenja. Osnovni princip vrednovanja je također težinsko ocjenjivanje čvorova i uspostavljanje njihovog povoljnog odnosa u populaciji. Metoda se može prikazati u nekoliko elemenata koji čine cjelinu (budući se operacije ne izvode na jednom mjestu u genetskom programu, postupak nije prikladno prikazivati u obliku jedinstvenog algoritma).

Veličina promatranog podskupa jedinki

Na početku rada genetskog programa jak je utjecaj rješenja dobivenih slučajnim pretraživanjem. Stoga se kao promatrani podskup jedinki, na temelju kojih se donosi ocjena kvalitete čvorova, u prvih 10 generacija postupka uzima cjelokupna populacija. Nakon 10. generacije, veličina podskupa prelazi na 10 % najboljih članova populacije.

Procjena težinskih vrijednosti čvorova

Težine čvorova određuju se na temelju jedinki u promatranom podskupu na sljedeći način:

$$w_n(g) = \sum_{i \in S} \left[freq_{i,n}(g) \cdot \left(A + B \frac{\left| f_i(g) - f_{LS} \right|}{\left| f_{HS} - f_{LS} \right|} \right) \right] + w_n(g-1), \qquad (5.2)$$

gdje je $w_n(g)$ težina čvora, S je promatrani podskup jedinki, $freq_{i,n}(g)$ je učestalost čvora *n* u jedinki i, $f_i(g)$ je funkcija dobrote jedinke, f_{LS} je dobrota najlošije a f_{HS} dobrota najbolje jedinke unutar podskupa S. Konstante A i B služe za određivanje omjera kvalitete pojedinih rješenja, a u ovom je radu empirijskim postupkom vrijednost za obje konstante postavljena na 1. Ocjenjivanje se može primijeniti na postupke u kojima je bolja dobrota prikazana ili manjom ili većom vrijednošću, uz odgovarajuću interpretaciju najbolje i najlošije jedinke u podskupu. Težinske vrijednosti čvorova također se računaju kumulativno, dodajući novu vrijednost na vrijednost iz prethodne generacije. Iznimka je trenutak promjene veličine promatranog podskupa, kada se dosadašnje težinske vrijednosti svih čvorova postavljaju na nulu. U suprotnom bi utjecaj akumuliranih težina, dobivenih promatranjem svih jedinki iz populacije, bio prevelik u odnosu na nove vrijednosti dobivene iz puno manjeg broja rješenja.

Prilagođeni operator mutacije

Prilikom primjene operatora mutacije na nekoj jedinki, ovisno o vrsti operatora, u jedinku se dodaju slučajno odabrani čvorovi. Kod prilagođene mutacije, vjerojatnost odabira određenog čvora proporcionalna je sljedećoj vrijednosti:

$$p_n(g) = C + D \cdot \frac{w_n(g)}{\sum_{t \in T} w_t(g)}, \qquad (5.3)$$

gdje je T skup svih promatranih čvorova. Konstante C i D određuju omjere vjerojatnosti odabira čvorova, a u ovom su radu postavljene na $C = 1$ i $D = 30$. Svi čvorovi imaju minimalnu vrijednost jednaku konstanti C, tako da nijednome čvoru nije posve uskraćena mogućnost pojavljivanja u mutaciji. Ukoliko evolucijski proces ne uključuje operator mutacije, što je u genetskom programiranju također moguće, bilo bi potrebno definirati posebni operator kojim bi bolji čvorovi dobivali veću brojnost u populaciji. Budući većina objavljenih radova uključuje mutaciju u pokusima [Lan 98, str. 34], u ovom se radu inačica genetskog programa bez mutacije ne razmatra.

Očuvanje i ograničavanje učestalosti čvorova

Pored promatranja učestalosti čvorova i računanja njihovih težinskih vrijednosti samo na temelju definiranog podskupa jedinki, u predloženom postupku također se promatraju relativni udjeli čvorova u cijeloj populaciji. Ukoliko postotak nekoga čvora u cjelokupnoj populaciji padne ispod 1 %, vrijednost $p_n(g)$ toga čvora postavlja se na polovinu vrijednosti za najbolje trenutno ocijenjeni čvor. Ukoliko pak postotak nekoga čvora prijeđe iznos od 10 %, vrijednost $p_n(g)$ toga čvora postavlja se na najmanji mogući iznos (vrijednost konstante C). Motivacija iza ovoga mehanizma je sprečavanje 'izumiranja' nekoga čvora iz populacije s jedne strane, a s druge strane prevelikog zagušenja nekim čvorom.

5.1.3 Ispitivanje učinkovitosti metode vrednovanja čvorova

Autori opisanog postupka odabira čvorova iz literature za ispitivanje učinkovitosti koriste dva problema. U oba problema definiran je skup funkcijskih čvorova sa osnovnim računskim operacijama, funkcijama sinus, kosinus i dekadski logaritam, te skup podatkovnih čvorova koji sadrži 30 elemenata označenih sa $x_1, x_2, ..., x_{30}$. Rješavanje problema sastoji se u otkrivanju simboličkog prikaza funkcije koja je u jednostavnijem slučaju zadana sa:

$$y_1 = x_1^3 + x_1^2 + x_1, \tag{5.4}$$

a u složenijem slučaju sa:

$$\begin{aligned} y_2 = {} & \sin(x_3) + \sin(2x_3) + \sin(3x_3) + \sin(4x_3) + \\ & \sin(x_2) + \sin(2x_2) + \sin(3x_2) + \\ & \sin(x_1) + \sin(2x_1) \end{aligned} \quad . \tag{5.5}$$

Svaka jedinka genetskog programiranja predstavlja neku funkciju koja bi za određene vrijednosti svih varijabli (x_1 - x_{30}) trebala dati rezultat što sličniji zadanoj funkciji. Prilikom ocjenjivanja rješenja koristi se slučajno generirani skup od 200 nizova vrijednosti za varijable x_1 - x_{30} u intervalu [-1, 1] (što čini 200 ispitnih primjera). Funkcija dobrote se u oba primjera definira kao zbroj apsolutnih odstupanja izraza predstavljenog stablom od prave vrijednosti dobivene funkcijama y_1 ili y_2. Broj članova populacije postavljen je na 2000, a evolucijski proces traje najviše 1000 generacija. Uspješnost algoritma ocjenjuje se brojem pokusa u kojemu je genetski program uspio naći točno rješenje, a točnim se smatra ono rješenje koje za dani skup ispitnih primjera daje rezultat koji od prave vrijednosti odstupa manje od 0.001. U prvom problemu algoritam treba otkriti samo jednu korisnu varijablu, što prilagođenoj inačici algoritma gotovo uvijek uspijeva. Za drugi problem postotak uspješnosti puno je manji, što je i očekivano, no ne i puno manji od postotka uspješnosti algoritma koji koristi samo bitne čvorove (x_1, x_2, x_3).

U ovom je radu načinjen pokušaj usporedbe predložene metode vrednovanja čvorova i opisanog postupka iz literature na istim primjerima, no tom je prilikom uočeno sljedeće: na jednostavnijem primjeru oba postupka pronalaze točno rješenje sa vrlo velikom vjerojatnošću (većom od 95 %) što ne omogućuje odgovarajuću usporedbu. Sličan problem javlja se pri pronalaženju složenije funkcije: oba algoritma već od početka evolucije gotovo u svakom pokusu uspijevaju prepoznati korisne čvorove (tako da je populacija u velikom dijelu načinjena samo od tih čvorova). S druge strane, točno rješenje nalazi se u vrlo malom broju pokusa (oko 5-6 % za oba postupka), što je uzrokovano

samom strukturom ciljne funkcije. U ovom slučaju problem ne predstavlja prepoznavanje korisnih čvorova, što oba postupka uspijevaju postići, nego naknadna gradnja ciljne funkcije. Za potrebe usporedbe stoga je definirana funkcija koja zahtijeva prepoznavanje istog broja bitnih čvorova:

$$y_3 = x_1^2 + x_2^2 + x_3^2 . \tag{5.6}$$

Iako po strukturi jednostavnija od funkcije opisane izrazom (5.5), definirana funkcija za oba algoritma predstavlja veći problem po pitanju prepoznavanja korisnih čvorova. Uzrok ovoj činjenici leži vjerojatno u tome da je u izrazu (5.6) algoritmu puno lakše zamijeniti vrijednosti neke od 'pravih' varijabli sa nekom 'lažnom' varijablom, što se često događa prilikom rada genetskog programa. Primjerice, algoritam bi prepoznao značaj dvije od tri potrebne varijable, ali bi umjesto treće varijable veliku težinsku vrijednost dobila neka druga varijabla. Budući se vrijednosti ulaznih varijabli, na temelju kojih se određuje vrijednost ciljne funkcije, stvaraju slučajno za svaki pokus, u pojedinom slučaju različita 'lažna' varijabla preuzela bi ulogu jedne od bitnih varijabli i time onemogućila dostizanje točnog rješenja. Jednom kada bi postupak pravilno prepoznao sva tri korisna čvora u zalihosnom skupu, genetski bi program gotovo uvijek pronašao pravo rješenje.

Ispitivanja su provedena za oba opisana postupka prilagodbe uz navedene parametre i veličinu populacije od 1000 i 2000 jedinki, kao i za osnovnu inačicu genetskog programiranja bez ocjenjivanja čvorova. Učinkovitost pojedinog postupka prikazana je postotnim udjelom broja pokusa u kojima je pronađeno točno rješenje, kao i srednjom vrijednošću najboljih rješenja za sve pokuse. Oba postupka prilagodbe ispitana su u 200 pokusa (za osnovnu inačicu provedeno je samo 100 pokusa), a rezultati su prikazani u tablici 5.1.

Tablica 5.1 Usporedni rezultati postupaka vrednovanja čvorova

Postupak	Postotak uspjeha	Srednja vrijednost najboljeg rješenja
osnovni genetski program, veličina populacije: 1000	0 %	37,49
prilagodba po [Ok 00], veličina populacije: 1000	10,5 %	28,65
predložena metoda vrednovanja, veličina populacije: 1000	17,5 %	24,53
osnovni genetski program, veličina populacije: 2000	0 %	22,62
prilagodba po [Ok 00], veličina populacije: 2000	36 %	13,59
predložena metoda vrednovanja, veličina populacije: 2000	47 %	9,87

Iz rezultata je vidljivo da predloženi postupak postiže nešto veću uspješnost iako razlika nije znatna. Također se može uočiti da povećanje veličine populacije znatno pomaže genetskom programu u pronalaženju točnog rješenja, a zbog veće populacije manja je i razlika u učinkovitosti postupaka. Korisnost obje metode potrebno je ispitati na većem broju problema i uz različite vrijednosti parametara kako bi se procijenila prava vrijednost prilagodbe evolucijskog procesa. Dodatno je ispitivanje provedeno u poglavlju 5.3.

5.2 Prilagodba genetskih operatora

5.2.1 Postupak prilagodbe operatora

Problem nadziranja vrijednosti različitih parametara evolucijskog procesa jedno je od najvažnijih i najisplativijih područja istraživanja metoda evolucijskog računanja. Osnovni cilj je postići prilagođavanje algoritma problemu za vrijeme rješavanja istoga problema. Motivacija iza ovih nastojanja je tvrdnja, manje ili više iskazana u različitim primjenama, da je bilo koji skup nepromjenjivih parametara nedovoljno kvalitetan za rješavanje zadanog skupa problema.

Prilagodba parametara u evolucijskim postupcima najčešće se odvija u dva koraka: prvi korak je prikupljanje informacija o trenutnom stanju evolucijskog procesa, a drugi korak predstavlja uporabu tih informacija mijenjanjem nekih od elemenata sustava. Mogućnosti promjene elemenata evolucijskog procesa su velike, u prvom redu zbog velikog broja međusobno povezanih operacija nad podacima i robusnosti samoga pristupa. Opširan pregled mogućnosti i podjela načina prilagodbe prikazani su u [Eib 00]. Većina postupaka prilagodbe temelji se na nekoj promjeni rada genetskih operatora, bilo operatora odabira, križanja ili mutacije, dok su nešto rjeđi postupci kojima se prilagođuju drugi elementi poput prikaza rješenja, funkcije dobrote, veličine populacije itd.

U ovom se radu predlaže postupak prilagodbe genetskih operatora na temelju promatranja stanja cjelokupne populacije. Ono što je važno kod opisanog postupka jest da je primjenjiv na više inačica evolucijskog računanja, a primarno je namijenjen uporabi za genetske algoritme i genetsko programiranje. Stoga će opis postupka prilagodbe u nastavku uzimati u obzir obje vrste evolucijskog procesa.

5.2.2 Određivanje stanja populacije

Osnovni preduvjet provedbe postupka prilagodbe je promatranje procesa evolucije za vrijeme rada algoritma. Ovakav pristup ograničava moguće načine promatranja na one koji ne zahtijevaju puno procesorskog vremena, iako je za većinu primjena u genetskim algoritmima (a pogotovo u genetskom programiranju) vrijeme evaluacije rješenja puno zahtjevnije od svih ostalih elemenata algoritma.

Jedan od jednostavnijih načina ostvarivanja uvida u stanje populacije je promatranje svojstava dobrote za cijelu populaciju. Najčešće promatrane svojstvene vrijednosti dobrote populacije su f_{MAX}, vrijednost dobrote najbolje jedinke i $\overline{f}$, prosječna dobrota svih jedinki. Pored toga, u analizu se može uvrstiti i dobrota najlošije jedinke f_{MIN} te standardno odstupanje od srednje vrijednosti, σ_{std}. Ovdje se pretpostavlja da veća vrijednost dobrote predstavlja i bolju vrijednost, a za potrebe minimiziranja potrebno je zamijeniti značenje oznaka f_{MAX} i f_{MIN}. Za sve se vrijednosti podrazumijeva ovisnost o trenutnoj generaciji, pa se može pisati npr. $f_{MIN}(g)$.

Procjena fenotipske raznolikosti može se dobiti promatrajući vrijednost izraza $\left(f_{MAX} - \overline{f}\right)$, koji će vjerojatnije poprimati manje vrijednosti za manje raznoliku populaciju, a veće vrijednosti za rastresit skup jedinki. Ovo je svojstvo već uočeno u prethodnim radovima [Sri 94, Jak 97, Jak 99] u smislu ocjene stanja populacije. Ispitivanje je pokazalo da je u primjeni povoljnije promatrati vrijednost sljedećeg izraza:

$$d(g) = \left(f_{MAX} - \overline{f}\right) \Big/ \left(f_{MAX} - f_{MIN}\right), \tag{5.7}$$

gdje je promatrana vrijednost podijeljena opsegom dobrota u trenutnoj generaciji. Ukoliko je vrijednost $d(g)$ 'mala', možemo pretpostaviti veći stupanj sličnosti jedinki u

populaciji, a ukoliko se vrijednost povećava, populacija postaje rastresitija. Međutim, ova vrijednost može ovisiti i o trenutnoj fazi evolucijskog procesa, odnosno o udjelu slučajno stvorenih rješenja u populaciji koji je na početku najveći. U stvarnosti, za svaki se provedeni pokus javljaju različite početne vrijednosti koje ovise o raspodjeli skupa početnih jedinki u prostoru rješenja. Stoga se vrijednost izraza (5.7) uspoređuje sa vrijednošću iz prethodne generacije kako bi se procijenio relativni pomak u stanju populacije. Uz pomoć dvije uzastopne vrijednosti definira se mjera kretanja populacije kao

$$w(g) = \min\left\{\left(\frac{d(g-1)}{d(g)}\right)^2, 1\right\}. \tag{5.8}$$

Mjera kretanja populacije interpretira se na sljedeći način: ako je vrijednost $d(g)$ u trenutnoj generaciji manja od vrijednosti u prethodnoj generaciji, pretpostavljamo da se raznolikost rješenja smanjuje, a potencija 2 dodana je radi veće osjetljivosti. Ako se vrijednost $d(g)$ povećava, pretpostavljamo da se raznolikost povećava, a mjeru kretanja zadržavamo na 1. Postupak prilagodbe operatora može koristiti opisanu mjeru i reagirati u slučaju smanjenja vrijednosti.

5.2.3 Prilagodljivi genetski operatori

Nakon dobivanja informacije o stanju populacije uz pomoć opisanih izraza, potrebno je na neki način prilagoditi ponašanje pojedinih operatora. Postupak prilagodbe operatora u velikoj je mjeri ovisan o vrsti primijenjenog evolucijskog procesa; na primjer, za određene vrste odabira ne postoji način utjecanja na učestalost operatora križanja, već je uporaba istoga uvjetovana procesom odabira [Bud 98, Jak 98]. U ovom radu će se postupak prilagodbe oslanjati na korištenu inačicu evolucijskog procesa koja uključuje turnirski odabir i ukupan broj mutacija određen zadanim vjerojatnostima (za sve oblike mutacije).

Prilagodba operatora križanja

Prilagodba operatora križanja ne uključuje mijenjanje načina rada samoga operatora, što ovisi o vrsti evolucijskog procesa i predstavljanju rješenja algoritma. Umjesto toga, postupak prilagodbe utječe na način odabira rješenja za križanje (ako to već nije određeno postupkom selekcije). Odabir jedinki za križanje te jedinke koja će biti zamijenjena rezultatom križanja najčešće se obavlja na dva načina:

- roditeljske jedinke se odabiru slučajno, a jedinka koja biva zamijenjena rezultatom križanja odabire se postupkom eliminacije,
- roditeljske jedinke odabiru se temeljem njihove dobrote (veću vjerojatnost odabira imaju bolje jedinke), a jedinka koja se zamjenjuje odabire se slučajno.

Opisana se prilagodba može uporabiti samo u inačicama algoritma u kojima se roditeljske jedinke odabiru slučajno, pa je na tom mjestu moguće utjecati na rad algoritma. Umjesto slučajnog odabira roditeljskih rješenja, za svaku se jedinku računa sljedeća svojstvena vrijednost:

$$v_i = (f_i - f_{MIN}) \cdot (2w(g) - 1) - (f_{MAX} - f_{MIN}) \cdot (w(g) - 1), \tag{5.9}$$

gdje je f_i dobrota pojedine jedinke u trenutnoj generaciji. Jedinke se tada biraju za križanje s vjerojatnošću proporcionalnom njihovim svojstvenim vrijednostima. Ovakvim načinom odabira jedinki postiže se sljedeće ponašanje: ukoliko je vrijednost $w(g)$ blizu 1

algoritam će preferirati jedinke s boljom vrijednošću dobrote i time ubrzavati konvergenciju. Ukoliko je vrijednost $w(g)$ manja, vjerojatnosti odabira se ujednačuju, a ako vrijednost mjere kretanja populacije padne ispod 0.5, jedinke s manjom dobrotom imat će veću vjerojatnost sudjelovanja u križanju.

Prilagodba operatora mutacije

Jednako kao i kod operatora križanja, prilagodba ovoga operatora ne podrazumijeva mijenjanje načina na koji se mutacija obavlja, nego najčešće promjenu učestalosti i odabira jedinki na koje se primjenjuje. U ovom se radu prilagodbom mijenja samo učestalost mutacije, dok odabir mutiranih jedinki ostaje potpuno slučajan. Parametar učestalosti mutacije može se definirati na više načina, od kojih su najuobičajeniji ili s pomoću vjerojatnosti ili ukupnog broja primjena operatora. Ukoliko je definirana vjerojatnost mutacije, najčešće se svaki put nakon stvaranja neke jedinke provjerava hoće li se nova jedinka mutirati ili ne. Ukoliko je definiran broj mutacija, nakon svake generacije obavlja se zadani broj mutacija, neovisno o drugim operatorima. Način primjene mutacije razlikuje se i ovisno o paradigmi evolucijskog procesa; kod genetskih algoritama uobičajena je primjena mutacije na novu jedinku dobivenu operatorom križanja. Kod genetskog programiranja u većini slučajeva oba se operatora ne primjenjuju uzastopno na jednu jedinku, već se mutacija obavlja usporedo sa operatorom križanja.

Ukoliko evolucijski proces provodi operator mutacije uz pomoć zadanog broja mutacija, tada se, koristeći vrijednosti opisane u prethodnom odjeljku, broj mutacija u jednoj generaciji definira sa

$$n_{mut} = N\left[2\cdot\left(1-w(g)\right)+0.1\right], \tag{5.10}$$

gdje je N ukupan broj jedinki u populaciji. Ako se u algoritmu mutacija provodi promatrajući vjerojatnost mutacije pojedine jedinke, tada se vjerojatnost mutacije definira kao

$$p_M = (A-B)\cdot\left(1-w(g)\right)+B, \tag{5.11}$$

gdje konstante A i B određuju najveću odnosno najmanju vjerojatnost mutacije. U pokusima u ovom radu te su vrijednosti postavljene na $A=0.5$ i $B=0.1$.

5.2.4 Ispitivanje učinkovitosti prilagodbe operatora

Učinkovitost opisanih prilagodbi ispitana je i za genetske algoritme i za genetsko programiranje. U oba slučaja uporabljeni su problemi prikladni odgovarajućoj metodi, dok su rezultati prikazani u obliku usporedbe promijenjenog algoritma sa osnovnom inačicom.

Ispitivanje učinkovitosti za genetske algoritme

Ispitivanje učinkovitosti provedeno je uporabom genetskog algoritma sa zadatkom optimiranja vrijednosti različitih funkcija. Genetski algoritam treba pronaći minimalne vrijednosti sljedećih funkcija [Gol 96]:

$$f_1 = 0.5+\frac{\sin^2\sqrt{\sum x_i^2}-0.5}{[1.0+0.001\cdot\sum x_i^2]^2},\quad x_i\in[-100,100], \tag{5.12}$$

$$f_2 = 1+(\sum x_i^2)^{0.25}\cdot[\sin^2(50(\sum x_i^2)^{0.1})+1.0],\quad x_i\in\left[-100,100\right], \tag{5.13}$$

$$f_3 = \sum\left(x_i^2-10\cos(2\pi\cdot x_i)\right),\quad x_i\in[-100,100], \tag{5.14}$$

$$f_4 = \sum x_i \sin(\sqrt{|x_i|}), \quad x_i \in [-512, 512]. \tag{5.15}$$

Ispitni algoritam koristi turnirski odabir uz veličinu turnira 3, binarni prikaz rješenja i primjenu mutacije uz zadani broj mutacija u jednoj generaciji prema izrazu (5.10). Algoritam sa prilagodbom uspoređen je sa genetskim algoritmom sa turnirskim i eliminacijskim odabirom. Parametri sve tri inačice prikazani su u tablici 5.2.

Tablica 5.2 Parametri promatranih genetskih algoritama

Parametar	GA s eliminacijskim odabirom	GA s turnirskim odabirom	GA s prilagodbom i turnirskim odabirom
postotak eliminacije	40 %	-	-
vjerojatnost mutacije jednoga bita	0.01	0.01	-
preciznost	10-5	10-5	10-5
veličina populacije	500	500	500

Uvjet zaustavljanja je dvojak: algoritam se zaustavlja ili nakon 10000 generacija ili nakon 2000 uzastopnih generacija bez poboljšanja vrijednosti najboljeg rješenja. Jednom generacijom se za sve tri inačice smatra stvaranje i evaluacija 500 novih jedinki. Ispitne funkcije su definirane uz proizvoljan broj varijabli, a pokusi su provedeni uz optimiranje ispitnih funkcija u 10 i u 30 dimenzija (broj elemenata vektora x). Rezultati su za sva tri algoritma prikazani u tablici 5.3 u obliku normiranog odstupanja najboljeg rješenja od vrijednosti minimuma funkcije.

Tablica 5.3 Rezultati optimiranja ispitnih funkcija

Ispitna funkcija	GA s eliminacijskim odabirom	GA s turnirskim odabirom	GA s prilagodbom i turnirskim odabirom
funkcija f_1, 10 dim.	4,456	0,125	0,098
funkcija f_1, 30 dim.	4,995	4,976	0,372
funkcija f_2, 10 dim.	5,754	0,036	0,068
funkcija f_2, 30 dim.	13,45	3,372	0,872
funkcija f_3, 10 dim.	16,12	0,64	0,85
funkcija f_3, 30 dim.	4,75	6,42	2,30
funkcija f_4, 10 dim.	0,448	0,024	0,028
funkcija f_4, 30 dim.	376,9	5,51	4,89

Ispitivanje učinkovitosti za genetsko programiranje

Provjera učinkovitosti prilagodbe operatora za genetsko programiranje provedena je na dva problema simboličke regresije, tj. pronalaženja najtočnijeg analitičkog prikaza nepoznate funkcije (primjer u odjeljku 3.1.3). U jednostavnijem problemu funkcija je zadana sa

$$f_1 = x_1 + x_2^2 + x_3^3, \tag{5.16}$$

a u složenijem kao

$$f_2 = \sin(x_3) + \sin(2x_3) + \sin(3x_3) + \sin(4x_3) + \sin(x_2) + \sin(2x_2) + \sin(3x_2) + \sin(x_1) + \sin(2x_1), \quad (5.17)$$

što je identično izrazu (5.5), tj. koristi se jednaka funkcija kao i kod ispitivanja metode vrednovanja čvorova. Razlika u primjeni u odnosu na prethodno poglavlje je u tome što su algoritmu na raspolaganju samo relevantni podatkovni čvorovi, odnosno x_1, x_2 i x_3. Skup funkcijskih čvorova je za oba problema sačinjen od osnovnih računskih operacija uz trigonometrijske funkcije sinus i kosinus. Dobrota rješenja gradi se na jednak način kao i u odjeljku 5.1.3: slučajno se odabire 200 nizova vrijednosti ulaznih varijabli x_1, x_2 i x_3 u intervalu [-1, 1] i računaju odgovarajuće vrijednosti funkcije cilja, a dobrota je definirana kao zbroj kvadrata odstupanja izraza predstavljenog stablom od prave vrijednosti dobivene funkcijama f_1 odnosno f_2. Broj članova populacije postavljen je na 500, a evolucijski proces ograničen je na 500 generacija. Najveća dubina stabla rješenja postavljena je na 14 razina (umjesto uobičajenih 17). U ostvarenju genetskog programiranja u ovom radu postupak odabira definira za svaku novu jedinku vjerojatnost primjene jednoga operatora kojim se dobiva nova jedinka (križanje, mutacija ili reprodukcija), a zbroj vjerojatnosti primjene svih operatora iznosi 100%. Postupak prilagodbe operatora mutacije u ovom primjeru ne mijenja izravno ukupan broj mutacija već vjerojatnost primjene mutacije na pojedinu jedinku, prema izrazu (5.11). Točan iznos vjerojatnosti mutacije izračunava se normiranjem vrijednosti dobivene izrazom (5.11) i iznosa vjerojatnosti ostalih operatora evolucijskog procesa, čije se vrijednosti ne mijenjaju.

Ispitivanje je provedeno za genetsko programiranje sa prilagodbom i za osnovnu inačicu algoritma. Uspješnost pojedinog postupka ocjenjuje se brojem pokusa u kojemu je genetski program uspio naći točno rješenje, a točnim se smatra ono rješenje koje za dani skup ispitnih primjera daje rezultat koji od prave vrijednosti odstupa za manje od 0.01. Za obje inačice algoritma i za svaku funkciju provedeno je po 200 pokusa, a rezultati su prikazani u tablici 5.4.

Tablica 5.4 Usporedni rezultati postupaka vrednovanja čvorova

Postupak	Postotak uspjeha	Srednja vrijednost najboljeg rješenja
osnovni genetski program, funkcija f_1	53 %	0,882
genetski program s prilagodbom, funkcija f_1	61,5 %	0,491
osnovni genetski program, funkcija f_2	1,5 %	8,79
genetski program s prilagodbom, funkcija f_2	2 %	6,34

Iz prikaza rezultata može se uočiti da genetski program s prilagodbom operatora postiže nešto bolja rješenja, no razlika u uspješnosti nije velika. Uzroci relativno slabije učinkovitosti prilagodbe kod genetskog programiranja u odnosu na primjenu u genetskom algoritmu mogu biti vrlo složeni. Budući su podatkovne strukture kojima se predstavlja

rješenje bitno različite za genetske algoritme i genetsko programiranje, definirana mjera kretanja populacije ne mora u jednakoj mjeri opisivati stanje evolucijskog procesa u obje metode. Isto tako, građa rješenja genetskog programiranja mijenja utjecaj koji genetski operatori imaju na stanje populacije. Dok je u genetskom algoritmu obnavljanje genetskog materijala i istraživanje novih područja pretežno zasluga operatora mutacije, u genetskom programiranju tu ulogu dijelom (ili čak u cijelosti, kao u nekim ostvarenjima) preuzima i operator križanja. Iz toga je razloga potrebno provesti više ispitivanja o utjecaju operatora na raznolikost populacije genetskog programiranja i korisnosti raznolikosti u postupku pronalaženja rješenja.

5.3 Primjena postupaka prilagodbe

5.3.1 Problem raspoređivanja u dinamičkim uvjetima

Postupci prilagodbe opisani u prethodnim poglavljima nisu vezani uz određeni problem ili način rješavanja, što im omogućuje općenitu primjenu. Naravno, poznato je da nijedan postupak ne može pokazivati bolju učinkovitost od nekog drugog postupka na svim problemima [Wol 97], pa rezultati pokazani u prethodnim poglavljima nisu nažalost jamstvo uspješnosti opisanih metoda na svakom području primjene. Budući nas u danim uvjetima obično zanimaju samo rješenja na određenom skupu problema, važno je utvrditi ponašanje postupaka u konkretnim situacijama. Na primjer, prilikom izvođenja pravila raspoređivanja za razna okruženja, najveći je problem određivanje neophodnih podatkovnih struktura i parametara evolucijskog procesa. U ovom je poglavlju stoga na primjeru pokazan utjecaj predloženih metoda na uspješnost pronalaženja rješenja u dotičnom problemu.

Kao primjer problema izvođenja pravila raspoređivanja uporabit će se raspoređivanje na jednom stroju u dinamičkim uvjetima, istovjetno okolini opisanoj u poglavlju 4.4.4. Pretpostavke problema su sljedeće: ukoliko ne posjedujemo opširnije znanje iz domene problema, kao podatkovne elemente na raspolaganju genetskom programu stavit ćemo u pravilu zalihosni skup elemenata, u nadi da će se u skupu naći dovoljno bitnih podataka kojima se može opisati traženo rješenje. Takav pristup nije nužno pogrešan, no obično će otežati pronalaženje rješenja i po pitanju utrošenog procesorskog vremena i po pitanju postignute kvalitete. Isto tako, priroda rješenja dobivenog genetskim programiranjem je takva da se u njemu mogu naći i elementi koji vrlo malo doprinose konačnom rezultatu, a mogu navesti na krive zaključke prilikom eventualnog interpretiranja rješenja.

U tablici 5.5 naveden je skup podatkovnih i funkcijskih čvorova korišten u obavljenim pokusima (skup funkcijskih čvorova istovjetan je onome iz poglavlja 4.4.4). Provedene su dvije skupine pokusa, jedna za osnovnu inačicu algoritma i druga za algoritam uz vrednovanje podatkovnih čvorova i prilagodbu operatora mutacije, a u svakoj skupini obavljeno je 20 pokusa. U pokusima su uporabljeni prethodno definirani ispitni primjeri iz skupa za učenje za dinamičku okolinu jednoga stroja. Ostali parametri genetskog programa jednaki su parametrima korištenim u 4. poglavlju.

Tablica 5.5 Popis podatkovnih čvorova za dinamički problem na jednom stroju

Oznaka funkcijskog čvora	Definicija
ADD, SUB, MUL, DIV, POS	kao u tablici 4.2
Oznaka podatkovnog čvora	Definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (d_j)

w	težina (w_j)
N	ukupni broj poslova
Nr	preostali broj poslova (koji još nisu raspoređeni)
SP	zbroj trajanja svih poslova
SPr	zbroj trajanja preostalih poslova
SD	zbroj željenih vremena završetka svih poslova
SL	pozitivna dopuštena odgoda, $\max\{d_j - p_j - time, 0\}$
CLK	trenutno vrijeme ($time$)
L	trenutno kašnjenje posla, $time - d_j$
TF	procjena očekivanog postotka zaostajanja (po engl. *tardiness factor*), $1 - \bar{d}/SP$, gdje je $\bar{d}$ srednje željeno vrijeme završetka
AR	vrijeme do pripravnosti posla, $\max\{r_j - time, 0\}$

Cilj je provedenih pokusa ustanoviti koliki utjecaj na ovome problemu ima metoda vrednovanja čvorova. U ovom je primjeru broj čvorova relativno mali, a među njima nema jasnog razgraničenja na 'bitne' i 'nebitne' čvorove. Stoga zadatak prilagodbe ovaj put nije ukloniti nebitne podatkovne elemente, nego pomoći genetskom programu prepoznati stupanj korisnosti pojedinih podataka. Rezultati su za obje skupine pokusa prikazani srednjom vrijednošću najboljih jedinki iz pojedinog pokusa te vrijednošću dobrote najbolje jedinke iz svih pokusa pojedine skupine. U tablici 5.6 navedeni su dobiveni rezultati.

Tablica 5.6 Usporedba izvođenja pravila raspoređivanja

Inačica algoritma	srednje najbolje rješenje	najbolje rješenje za sve pokuse
osnovni genetski program	53,76	52,59
genetski program uz vrednovanje čvorova	52,65	51,95

Budući su rješenja koja se dobivaju primjenom izvedenih pravila raspoređivanja već relativno blizu optimalnima (slika 4.12, str. 46), a i ograničena samim principom uporabe pravila raspoređivanja, eventualna poboljšanja postupka u ovom primjeru ne mogu biti značajna. Iz rezultata se može vidjeti da je postupak uz prilagodbu ostvario prosječno nešto bolje vrijednosti, no prednost u praktičnoj primjeni ostaje vrlo mala. Isto tako, poradi malog broja pokusa (a zbog velikih računalnih zahtjeva), statistički promatrano razlika između dvije inačice algoritma nije velika. Usprkos tome, ne može se zanemariti značaj postupaka prilagodbe i njihova uloga u problemima u kojima bi postupak pronalaženja rješenja inače bio znatno otežan.

5.4 Ubrzavanje genetskog programiranja

5.4.1 Određivanje dobrote rješenja

Pronalaženje zadovoljavajućeg rješenja genetskim programiranjem često je dugotrajan proces. Osnovni razlog toj pojavi je sama ideja genetskog programiranja, u kojoj jedinka predstavlja moguće rješenje problema u obliku računalnog programa.

Ispitivanje ispravnosti programa je problem koji u općenitom slučaju nije moguće egzaktno riješiti, pa se kvaliteta rješenja uvijek ocjenjuje većim ili manjim skupom primjera iz domene algoritma. Na osnovu učinkovitosti programa u zadanom skupu primjera, predviđamo ponašanje toga programa u problemima koji nisu obuhvaćeni postupkom ocjenjivanja. Iako ovakav pristup ne jamči ispravnost programa u svim slučajevima, opisani način ispitivanja rutinski se koristi u svim oblicima strojnog učenja, pa i u većini primjera razvoja komercijalne programske podrške.

Prilikom ocjenjivanja algoritama raspoređivanja u ovom radu koristi se skup ispitnih primjera za koje algoritam treba odrediti raspored. Na temelju kvalitete rasporeda svih ispitnih primjera određuje se konačan iznos dobrote jedinke. Algoritam raspoređivanja u obliku pravila predstavljen je funkcijom prioriteta koja se uzastopno primjenjuje u izradi rasporeda. Postupak je u općenitom slučaju sljedeći: na početku se računa funkcija prioriteta za sve poslove u ispitnom primjeru, na temelju čega se raspoređuje najbolje ocijenjeni posao. Potom se prioritet računa za sve preostale poslove, najbolji posao se ponovno raspoređuje, a postupak se ponavlja do raspoređivanja svih poslova u ispitnom primjeru. Nakon toga se dobiveni raspored ocjenjuje po jednom ili više kriterija, a ispitivanje se nastavlja za sve ostale primjere.

Najviše se procesorskog vremena troši upravo prilikom računanja vrijednosti funkcije prioriteta, budući je ona predstavljena stablom koje je potrebno interpretirati tijekom rada programa. U okruženju jednoga stroja, uz korišteni skup od 100 ispitnih primjera sa 12 do 100 poslova u svakome, za ocjenu dobrote jedinke stablo se mora interpretirati 4675 puta. Situacija je još teža u složenijim okruženjima gdje se prioritet računa za sve parove poslova i strojeva (ovisno o meta-algoritmu). Prilikom svakog novog računanja također se moraju promijeniti vrijednosti odgovarajućih varijabli ovisnih o pojedinom poslu, trenutnom vremenu i sl. Osim toga, prosječna veličina rješenja uglavnom raste tijekom evolucijskog procesa, što kao posljedicu ima znatno usporenje postupka nakon određenog broja generacija. Poradi svega navedenog, potrebno je osmisliti učinkovite načine predstavljanja i ocjenjivanja rješenja u evolucijskom procesu. Neka od ostvarenih poboljšanja opisana su u sljedećim odjeljcima.

5.4.2 Vektorsko ocjenjivanje rješenja

Po pretpostavci, podatkovni elementi stabla rješenja predstavljaju skalarni podatak čija je vrijednost ovisna o promatranom poslu ili stroju i trenutnom stanju u sustavu. Funkcijski su elementi također definirani samo za skalarni oblik podataka, međutim ne postoji zapreka definiranju operatora za složenije vrste podataka, kao što su liste, višedimenzijska polja i slično. U postupku ocjenjivanja pravila raspoređivanja posebno je pogodan vektorski oblik podatka, u kojemu se može zapisati niz vrijednosti za sve elemente sustava. Taj se pristup koristi i u ovom radu: umjesto skalarnih podatkovnih elemenata, svaki podatkovni čvor stabla predstavlja zapravo niz podataka određene duljine. Isto tako, svi funkcijski elementi prilagođeni su tako da rukuju sa nizovima podataka i kao rezultat vraćaju niz.

Ocjenjivanje jedinke provodi se na sljedeći način: za pojedini ispitni primjer veličina svakog podatkovnog elementa postavlja se na broj poslova u dotičnom primjeru. Izuzetak je računanje vrijednosti stabla odluke u raspoređivanju proizvoljne obrade, gdje se duljina niza postavlja na broj strojeva (svojstva poslova se u stablu odluke ne razmatraju). Vremenski neovisni podaci su predefinirani i izračunavaju se samo na početku rada evolucijskog procesa za sve primjere (npr. ukupni brojevi poslova, razina posla u grafu ovisnosti itd.). Vremenski ovisni podaci definiraju se po potrebi za sve poslove koji dolaze u obzir za raspoređivanje u određenom trenutku u sustavu. Podaci koji su zajednički za sve poslove jednostavno se kopiraju u sve elemente podatkovnog niza,

dok se podaci ovisni o poslu upisuju na odgovarajuće mjesto u nizu. Funkcijski elementi koji uključuju grananje (npr. 'IFGT' funkcija) za svaki posao računaju sve mogućnosti, a naknadno se provodi odluka o prosljeđivanju odgovarajućeg rezultata.

Budući nije unaprijed poznato koji će poslovi biti raspoređeni prije drugih, u postupku se koristi i posebni niz rednih brojeva poslova, u kojemu su odvojeno smješteni redni brojevi raspoređenih i neraspoređenih poslova. U svakoj iteraciji izračunavanje se obavlja samo za one dijelove podatkovnog niza koji predstavljaju neraspoređene poslove, a podacima se pristupa neizravno preko niza rednih brojeva poslova. Na taj se način u svakom sljedećem koraku smanjuje količina računanja.

U nekim okruženjima, na primjer u dinamičkoj okolini jednoga stroja, nije potrebno računati vrijednost prioriteta za sve neraspoređene poslove. Na početku rada sustava većina poslova još nije pripravna pa se ne moraju uzeti u obzir. Stoga je niz rednih brojeva poslova na početku poredan po rastućim vrijednostima vremena pripravnosti poslova, pa se prije računanja prioriteta može odrediti posljednji element niza poslova kojega je potrebno uključiti u računanje.

Nakon definiranja svih opisanih elemenata, računanje prioriteta provodi se paralelno za sve poslove koji mogu biti raspoređeni u sljedećem koraku izrade rasporeda. Ovim načinom ocjenjivanja rješenja postiže se otprilike 20-30% brži proces evolucije, što uz velike populacije može dovesti do znatnih ušteda u vremenu.

5.4.3 Uređivanje rješenja

Genetsko programiranje gradi rješenja primjenom operatora križanja i mutacije, dok se kvaliteta rješenja označava njegovom dobrotom. Za vrijeme evolucije genetski program rukuje jedinkom kao sa 'crnom kutijom', primjenjujući odgovarajuće operatore i uspoređujući kvalitetu pojedinih jedinki. Sama unutarnja građa rješenja nebitna je u procesu evolucije jer se rješenje gleda uvijek kao jedinstvena cjelina, najčešće uz samo jedno mjerilo vrednovanja. Zbog toga se unutar jedinke javljaju dijelovi koji imaju nikakav ili vrlo mali utjecaj na njenu funkcionalnost. Većinom se nekorisni dijelovi rješenja koji imaju mali utjecaj na njegovu kvalitetu sastoje od umetaka (poglavlje 3.2.6). Omjer korisnih i nekorisnih dijelova unutar rješenja ovisi o puno elemenata: funkciji dobrote, načinu prikaza rješenja, stupnju evolucijskog procesa itd. Nakupljanje nekorisnih dijelova dovodi do porasta prosječne veličine rješenja, usporavanja ocjenjivanja i nepreglednosti rezultata. U poglavlju 3.2.6 prikazane su neke tehnike sprječavanja nekontroliranog rasta rješenja genetskog programiranja. U ovom je radu ostvaren pristup koji pripada grupi postupaka otkrivanja nekorisnih dijelova rješenja.

Budući se primjenjuje prilikom ocjenjivanja, tehnika otkrivanja nekorisnih dijelova rješenja treba biti dovoljno jednostavna kako ne bi oduzimala previše vremena i time poništila eventualnu uštedu. Određivanje kvalitete pravila raspoređivanja obavlja se uzastopnim računanjem prioritetne funkcije koja je predstavljena stablom jedinke. Budući je broj izračunavanja vrijednosti funkcije relativno velik, isplativo je prije ocjenjivanja pojedine jedinke što više pojednostaviti građu rješenja, pa je opravdana uporaba neke metode otkrivanja nekorisnih dijelova.

Postupak ostvaren u ovom radu temelji se na ideji o uređivanju rješenja (engl. *editing*) [Koz 92, str. 108]. Za korištene funkcijske elemente definiran je skup aksioma koji služe pojednostavljivanju rješenja: aksiomi su odabrani tako da se njihovom primjenom složeniji izrazi mogu zapisati u kraćem obliku. Uzastopnom primjenom aksioma dolazi se, u većini slučajeva, do kraćeg zapisa rješenja, što izravno utječe na smanjenje trajanja evolucijskog procesa. Nedostatak ove tehnike je činjenica da nije općenita budući se koriste aksiomi ovisni o uporabljenim funkcijskim elementima. U većini se primjera, međutim, promatra određeni razred problema za koji se definiraju

zajednički funkcijski čvorovi, a neke se funkcije (npr. osnovne računske operacije) nalaze u gotovo svim primjenama genetskog programiranja.

Skup aksioma primijenjenih u pojednostavljivanju rješenja definiran je sljedećim elementima:

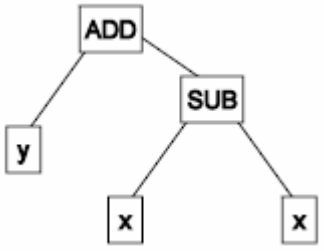
$$\left.\begin{array}{ll} A1. & SUB(a,a)=0 \\ A2. & DIV(a,a)=1 \\ A3. & ADD(S,0)=ADD(0,S)=S \\ A4. & SUB(S,0)=S \\ A5. & MUL(S,1)=MUL(1,S)=S \\ A6. & MUL(S,0)=MUL(0,S)=S \\ A7. & DIV(S,0)=1 \\ A8. & DIV(S,1)=S \\ A9. & DIV(0,S)=0 \end{array}\right\}. \qquad (5.18)$$

U navedenom skupu a predstavlja bilo koji podatkovni element, a S predstavlja bilo koji izraz odnosno dio rješenja. Budući u skupu podatkovnih elemenata nema numeričkih konstanti 0 i 1, preduvjet primjene aksioma $A3-A9$ je barem jedna prethodna primjena aksioma $A1$ ili $A2$. Aksiom $A7$ nije matematički ispravan, no u skladu je sa definicijom funkcije DIV koja predstavlja zaštićeni operator dijeljenja. Pored aksioma koji pojednostavljuju izraze s osnovnim računskim operacijama, definirani su i aksiomi za neke od ostalih korištenih funkcijskih elemenata:

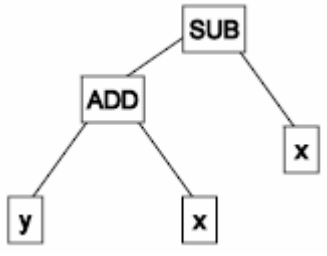
$$\left.\begin{array}{ll} A10. & POS(0)=0 \\ A11. & POS(1)=1 \\ A12. & POS(a)=a, \forall a>0 \\ A13. & SQR(0)=0 \\ A14. & SQR(1)=1 \end{array}\right\}. \qquad (5.19)$$

Uporaba aksioma $A12$ zahtijeva poznavanje definicije podatkovnih elemenata; na primjer, trajanje i željeno vrijeme završetka posla uvijek su pozitivne veličine.

Primjena aksioma obavlja se odmah nakon stvaranja nove jedinke, a prije postupka ocjenjivanja. Kako bi se aksiomi mogli jednostavnije primijeniti, rješenje je zapisano u prefiksnom obliku. Primjerice, matematički izraz $y+x-x$, predstavljen stablom na slici 5.1 zapisan je kao $ADD(y,SUB(x,x))$.



Slika 5.1 Stablo $ADD(y,SUB(x,x))$



Slika 5.2 Stablo $SUB(x,ADD(y,x))$

Aksiomi se primjenjuju pregledavanjem stabla u dubinu, pa će u ovom slučaju prvo biti primijenjen aksiom $A1$ a nakon toga aksiom $A3$, što kao rezultat daje vrijednost y. Ukoliko je, međutim, isti matematički izraz prikazan stablom na slici 5.2, primjena aksioma neće dovesti do pojednostavljenja zapisa, budući se radi o različitom redoslijedu operatora. Rješenje predstavljeno stablom na slici 5.2 može biti pojednostavljeno, no za to je potrebno definirati puno složenije mehanizme koji se u primjeni (za vrijeme evolucijskog procesa) obično ne koriste.

Ovako pojednostavljeni zapis rješenja *ne zapisuje* se umjesto postojećeg rješenja, već se koristi samo prilikom određivanja dobrote. Učinak zapisivanja pojednostavljenog oblika rješenja nije jednostavno ocijeniti i u većini se primjena izbjegava. Na primjer, nekorisni dio rješenja može uslijed djelovanja genetskih operatora postati koristan, tj. neka promjena u nekorisnom dijelu može uzrokovati promjenu utjecaja toga dijela na dobrotu rješenja. S druge strane, krajnje dobiveno rješenje može se nakon završetka rada genetskog programa pojednostaviti uporabom nekog od postojećih alata za rad sa simboličkim izrazima. Sva rješenja prikazana u ovom radu (prilog B1 - B4) prethodno su pojednostavljena primjenom opisanih aksioma.

Postotni udio na ovaj način otkrivenih nekorisnih dijelova rješenja bilježi se tijekom evolucijskog procesa. U pokusima provedenim u ovome radu taj se udio većinom kreće između 3 % i 9 %. Postotne vrijednosti nisu velike, no kako se gotovo cjelokupno trajanje genetskog programa sastoji od ocjenjivanja rješenja, navedeno se smanjenje veličine zapisa izravno preslikava u uštedu procesorskog vremena.

6 Zaključak

Genetsko programiranje je metoda stohastičkog pretraživanja koja predstavlja rješenja u obliku računalnih programa i na taj način omogućava rješavanje problema koji bi inače zahtijevao izradu programa. Zbog svoje strukture, genetsko programiranje može se smatrati najopćenitijim oblikom evolucijskog računanja te jednom od najopćenitijih paradigmi strojnog učenja.

U ovom je radu genetsko programiranje primijenjeno na problem definiranja pravila raspoređivanja u raznolikim okruženjima. Pravila raspoređivanja mogu se smatrati podvrstom heurističkih algoritama raspoređivanja koja raspored izrađuje u koracima uz definiranje sljedećeg stanja sustava. Pravila raspoređivanja u velikoj se mjeri koriste u raznim okolinama, no jedan od problema u primjeni je pronalaženje djelotvornog pravila za dano okruženje i uvjete raspoređivanja. Cilj ovoga rada je pokazati utemeljenost tvrdnje kako je korištenjem genetskog programiranja moguće stvaranje pravila raspoređivanja koja se po učinkovitosti mogu mjeriti s postojećim algoritmima iste namjene. Korisnost ovoga pristupa pokazuje se u slučajevima gdje su postojeća pravila neprikladna i time neučinkovita, a proces stvaranja pravila prilagođenih danoj okolini moguće je u određenoj mjeri automatizirati. Namjera rada nije proizvesti postupak koji dovodi do optimalnog rješenja nekoga problema, već daje kvalitetno rješenje za proizvoljnu okolinu i zahtjeve korisnika u dinamičkim uvjetima rada.

Pravilo raspoređivanja u užem smislu predstavlja samo način određivanja prioriteta elemenata u sustavu, kao što su poslovi ili sredstva. Primjena pravila raspoređivanja razlikuje se ovisno o zadanom okruženju, pa je za svaku okolinu potrebno definirati meta-algoritam koji nadzire uporabu odgovarajućeg pravila. Ovakav način odvajanja centralnog dijela algoritma, kojim se određuju prioriteti elemenata sustava, i upravljačkog dijela, koji raspoređuje aktivnosti u sustavu, posebno je pogodan za ugradnju, ocjenjivanje i automatsko sastavljanje algoritma.

Djelotvornost postupka izvođenja pravila raspoređivanja pokazana je na većem broju okruženja i mjerila vrednovanja. Za potrebe ispitivanja izgrađen je programski sustav koji podržava nekoliko različitih okruženja raspoređivanja, proizvoljna svojstva poslova i kriterije ocjene rasporeda te obuhvaća najpoznatija pravila i algoritme raspoređivanja za dotične okoline. Za svaki promatrani problem definirani su mnogobrojni ispitni primjeri čija je struktura u skladu sa ispitnim primjerima iz literature. Odgovarajuće pravilo raspoređivanja izvodi se za pojedinu okolinu te se nakon procesa učenja uspoređuje s algoritmima raspoređivanja iz literature. Definiranje kriterija za usporedbu uzima u obzir svojstva pojedinih ispitnih primjera, kao što su broj poslova, težinske vrijednosti itd., čime se vrednovanje postupka ujednačava za različita ispitna okruženja.

Velik broj primjera posvećen je okruženju jednoga stroja budući se rješavanje problema u složenijim okolinama nerijetko temelji na rješenjima iz ove domene. Na različitim inačicama osnovnog problema pokazano je stvaranje proizvoljnog pravila raspoređivanja. Za svaki primjer određeni su potrebni elementi rješenja genetskog programiranja i definiran način korištenja u pojedinoj okolini. U slučaju dinamičke okoline, predložen je meta-algoritam koji u sprezi s izvedenim pravilom postiže bolje rezultate od postojećeg pristupa. U ovom je okruženju također pokazana isplativost postupka stvaranja pravila za slučajeve u kojima je teško pronaći (ili ne postoji) odgovarajući algoritam.

Postupak izvođenja pravila raspoređivanja pokazan je i u okruženju paralelnih jednolikih i nesrodnih strojeva. Od velikog broja mogućih kombinacija uvjeta i kriterija

raspoređivanja odabrani su samo neki primjeri na kojima je ispitana učinkovitost pravila dobivenih genetskim programiranjem. Pravila raspoređivanja primijenjena su i u uvjetima dinamičkog raspoređivanja na zahtjev i uspoređena sa postojećim algoritmima raspoređivanja u toj okolini. Za potrebe raspoređivanja na nesrodnim strojevima predložen je meta-algoritam koji omogućuje djelotvornu uporabu sustava sa prioritetnim raspoređivanjem i postiže rezultate sumjerljive sa najboljim promatranim algoritmima. Važno je napomenuti da se predloženi algoritam i dalje može koristiti kao pravilo raspoređivanja budući u jednom koraku primjene definira samo sljedeće stanje sustava. U sprezi s meta-algoritmom moguće je iskoristiti bilo koji odgovarajući postupak određivanja prioriteta.

Okruženje proizvoljne obrade jedno je od najsloženijih u teoriji raspoređivanja pa je stoga i raspoloživ velik broj postojećih heuristika. Primjena istovjetnog modela raspoređivanja kao i kod jednostavnijih okruženja pokazuje sposobnost genetskog programiranja da i na ovaj način proizvede kvalitetno rješenje. U cilju poboljšavanja prilagodbe zahtjevnijim uvjetima rada, definirana je struktura pravila raspoređivanja sačinjena od tri neovisna elementa. Podjela uloga dijelova rješenja (prepoznavanje uskog grla sustava te računanje prioriteta za različite stupnjeve opterećenja) omogućuje iskorištavanje dodatne informacije o opterećenosti kako bi se poboljšao postupak raspoređivanja na trenutno promatranom stroju.

Opisani pristup izvođenja pravila raspoređivanja genetskim programiranjem vrlo je općenit i time primjenjiv na raznolike probleme. No, upravo se u općenitosti nalazi i najveći problem prilikom uporabe takvog sustava. Kao i kod svih ostalih metoda strojnog učenja, korisnik mora definirati skup podataka na osnovu kojih sustav može učiti i ugraditi ih u rješenje. Dodatni je problem postavljanje vrijednosti različitih parametara koji su svojstveni svim postupcima optimiranja, a o kojima uvelike ovisi uspješnost pronalaženja rješenja. Zadovoljavajuće postavke algoritma mogu se odrediti na više načina od kojih su u uporabi najčešći ili konzultiranje stručnog znanja za određeno područje ili opetovano ispitivanje uz različite početne uvjete. Napor koji bi bilo potrebno uložiti kako bismo odredili 'najbolje' postavke za rad algoritma često nije isplativo poduzimati, a i u suprotnosti je s principima strojnog učenja u kojemu teret odlučivanja prebacujemo na računalo. Stoga se u radu obrađuje i tematika općenite prilagodbe postupaka evolucijskog računanja, a specifično je obrađen problem odabira podatkovnih elemenata i prilagodbe genetskih operatora. Za oba problema predloženi su postupci kojima je cilj poboljšanje djelotvornosti algoritma. Posebnost ovih postupaka je mogućnost uporabe u rješavanju bilo kojega problema, a za postupak prilagodbe genetskih operatora primjena je moguća i u genetskim algoritmima i u genetskom programiranju. Uspješnost opisanih postupaka pokazana je na nekoliko primjera te na problemu izvođenja pravila raspoređivanja u dinamičkoj okolini. Iako su postupci na jednostavnijim primjerima pokazali poboljšanje u odnosu na ostale metode ili osnovnu inačicu algoritma, za problem raspoređivanja poboljšanja su relativno mala. To je djelomično uzrokovano i prirodom problema, za koji postoji realno ograničenje u čijoj blizini nema mogućnosti za velike razlike u kvaliteti (barem s gledišta pravila raspoređivanja).

Rezultati rada su višestruki: kao neposredni proizvod prikazano je nekoliko pravila raspoređivanja (u prilogu) koja se u neizmijenjenom obliku mogu primijeniti u odgovarajućim okruženjima. Izgrađeni programski sustav moguće je uporabiti za izvođenje pravila raspoređivanja za kombinacije okoline i kriterija koje u radu nisu eksplicitno navedene. Sâm način sastavljanja pravila moguće je primijeniti i na nekom drugom srodnom problemu, a predloženi meta-algoritmi mogu se koristiti u sprezi s proizvoljnim metodama prioritetnog raspoređivanja. Opisani postupci prilagodbe imaju

mogućnost primjene u rješavanju bilo kojeg problema genetskim programiranjem odnosno genetskim algoritmom.

Problem raspoređivanja uz pomoć genetskog programiranja već je bio predmetom istraživanja u literaturi. Ovdje opisani postupci unose sljedeće novosti u odnosu na dosadašnje radove:

- problem je rješavan za još neobrađena okruženja i uvjete raspoređivanja, kao što su raspoređivanje na zahtjev, ovisnosti u redoslijedu, dinamički dolasci poslova itd.;
- definirani su vlastiti podatkovni elementi i meta-algoritmi za pojedina okruženja;
- ostvarena je višestablena struktura rješenja koja omogućava bolju prilagodbu trenutnim uvjetima rada;
- rezultati su normirani prema svojstvima ispitnih primjera i uspoređeni sa većim brojem postojećih algoritama raspoređivanja.

Problemi uočeni prilikom izrade rada upućuju na mnoga područja mogućeg poboljšanja i dodatne smjerove razvoja. Buduća istraživanja mogu obuhvaćati sljedeće elemente:

- provedba iscrpnijih ispitivanja utjecaja postupaka prilagodbe na učinkovitost genetskog programiranja na različitim problemima, što predstavlja relativno neistraženo područje;
- uvrštavanje višestrukih mjerila vrednovanja u proces izvođenja pravila poradi ujednačavanja učinka;
- podrobnija ocjena ponašanja izvedenih algoritama u promjenjivim uvjetima rada;
- obogaćivanje građe rješenja genetskog programa dodavanjem elemenata kao što su spremnički prostor (za pohranjivanje međurezultata) ili druga pogodna podatkovna struktura;
- primjena postupka izvođenja pravila na okruženja u kojima postoji potreba za prikladnim algoritmima (neprecizno računanje, ugrađeni sustavi, Grid aplikacije);
- primjena opisanog pristupa u problemima izvan klasične teorije raspoređivanja, kao npr. stvaranje metode izbacivanja stranica iz priručnog spremničkog prostora (engl. *cache*), što se također može smatrati oblikom raspoređivanja [Pat 97, Nei 99, Nei 99a];
- primjena genetskog programiranja i u oblikovanju meta-algoritma, odnosno upravljačkog dijela pravila raspoređivanja, za što je potrebno pažljivo definirati odgovarajuću strukturu rješenja.

Iako se postupak izvođenja pravila raspoređivanja opisan u ovom radu može svesti na zajednički nazivnik, neophodni je element za rješavanje svakoga problema ipak ljudski čimbenik. Imajući to na umu, opisana metodologija može se shvatiti kao moguće olakšavanje procesa pronalaženja rješenja u koji bi inače bilo potrebno uložiti puno više vremena.

Literatura

[Ada 02] T. P. Adams, *Creation of Simple, Deadline, and Priority Scheduling Algorithms using Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2002, http://www.genetic-programming.org/sp2002/Adams.pdf

[And 94] D. Andre, *Automatically Defined Features: The Simultaneous Evolution of 2-dimensional Featrue Detectors and an Algorithm for Using Them*, Advances in Genetic Programming, pp. 477-494, MIT Press, 1994.

[Atl 94] Bonnet L. Atlan, J.B. Polack, *Learning distributed reactive strategies by genetic programming for the general job shop problem,* Proceedings 7th annual Florida Artificial Intelligence Research Symposium, Pensacola, Florida, IEEE Press, 1994.

[Auy 03] Andy Auyeung, Iker Gondra, H. K. Dai, *Multi-heuristic list scheduling genetic algorithm for task scheduling,* Symposium on Applied Computing, Proceedings of the 2003 ACM symposium on Applied computing,Melbourne, Florida,Pages: 721 - 724, http://portal.acm.org/citation.cfm?doid=952532.952673

[Ban 02] W. Banzhaf, W. B. Langdon, *Some considerations on the reason for bloat,* Genetic Programming and Evolvable Machines, 3(1), pp. 81-91, March 2002.

[Ban 98] W. Banzhaf, P. Nordin, R. E. Kellet, F. D. Francone, *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, 1998.

[Bea 05] Open BEAGLE: a versatile EC framework, http://beagle.gel.ulaval.ca/

[Bea 90] J. E. Beasley, *OR-Library: Weighted tardiness*, 1990, http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html

[Bli 94] T. Blickle, L. Thiele, *Genetic Programming and Redundancy,* Proc. Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94), Saarbrücken, Germany, 1994. , http://www.handshake.de/user/blickle/publications/GPandRedundancy.ps

[Bli 96] Tobias Blickle, *Evolving Compact Solutions in Genetic Programming: A Case Study,* Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation, LNCS, Vol. 1141, pp. 564-573, Springer-Verlag, 22-26 September 1996., http://www.blickle.handshake.de/publications/ppsn1.ps

[Bud 00] L. Budin, D. Jakobović, M. Golub, *Genetic Algorithms in Real-Time Imprecise Computing*, Journal of Computing and Information Technology CIT, Vol. 8, No. 3, September 2000, pp. 249-257.

[Bud 98] L. Budin, D. Jakobović, M. Golub, *Parallel Adaptive Genetic Algorithm*, Proc. Int. ICSC/IFAC Symposium on Neural Computation, NC'98, Vienna, September 23-25, 1998., pp. 157-163

[Bud 99] L. Budin, D. Jakobović, M. Golub, *Genetic Algorithms in Real-Time Imprecise Computing*, IEEE International Symposium on Industrial Electronics ISIE'99, Bled, 1999, Vol. 1, pp. 84-89.

[Bur 03] Edmund Burke, Steven Gustafson, Graham Kendall, *Ramped Half-n-Half Initialisation Bias in GP*, enetic and Evolutionary Computation -- GECCO-2003, LNCS, Vol. 2724, pp. 1800-1801, Springer-Verlag, 12-16 July 2003.

[Cha 04] Samarn Chantaravarapan, Jatinder N.D. Gupta, *Single Machine Group Scheduling with Setups to Minimize Total Tardiness,* 2004, http://www.pmcorp.com/PublishedPapers/Scheduling%20Publications/SingleMachineGroupSchedulingwithSetups.pdf

[Cha 96] Yih-Long Chang, Toshiyuki Sueyoshi, Robert Sullivan, *Ranking dispatching rules by data envelopment analysis in a job shop environment,* IIE Transactions, 28(8):631-642, 1996

[Che 99] V.H.L. Cheng, L.S. Crawford, P.K. Menon, *Air Traffic Control Using Genetic Search Techniques,* 1999 IEEE International Conference on Control Applications, August 22-27, Hawai'i, HA, http://www.optisyn.com/papers/1999/traffic_99.pdf

[Cic 01] Vincent A. Cicirello, Stephen F. Smith, *Ant Colony Control for Autonomous Decentralized Shop Floor Routing,* Fifth International Symposium on Autonomous Decentralized Systems March 26 - 28, 2001 Dallas, Texas p. 383, http://csdl.computer.org/comp/proceedings/isads/2001/1065/00/10650383abs.htm

[Cic 01a] Vincent Cicirello, Stephen Smith, *Randomizing Dispatch Scheduling Policies,* The 2001 AAAI Fall Symposium: Using Uncertainty Within Computation, November, 2001., http://www.ri.cmu.edu/pubs/pub_3789.html

[Cic 03] Vincent Cicirello, *Weighted Tardiness Scheduling with Sequence-Dependent Setups,* Technical report, The Robotics Institute, Carnegie Mellon Univeristy, 2003, http://www.cs.drexel.edu/~cicirello/benchmarks.html

[Cor 01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd ed., The MIT Press - McGraw-Hill, 2001.

[Cra 85] Nichael Lynn Cramer, *A Representation for the Adapative Generation of Simple Sequential Programs,* International Conference on Genetic Algorithms and their Applications [ICGA85], CMU, Pittsburgh, http://www.sover.net/~nichael/nlc-publications/icga85/index.html

[Dav 81] Ernest Davis, Jeffrey M. Jaffe, Algorithms for Scheduling Tasks on Unrelated Processors, Journal of the ACM, Volume 28 , Issue 4 (October 1981), pp. 721 – 736 http://portal.acm.org/citation.cfm?doid=322276.322284

[Dha 78] B. G. Dharan, T.E. Morton, Algoristics for Single Machine Sequencing with Precedence Constraints, Management Science 24, p. 1011-1020, 1978. http://www.ruf.rice.edu/~bala/files/dharan-morton-algoristics_for_sequencing-Mgt_Science_1978.pdf

[Dim 01] C. Dimopoulos, A. M. S. Zalzala, *Investigating the use of genetic programming for a classic one-machine scheduling problem,* Advances in Engineering Software, Volume 32, Issue 6 , June 2001, Pages 489-498, http://www.sciencedirect.com/

[Dim 99] Christos Dimopoulos, Ali M. S. Zalzala, *Evolving Scheduling Policies through a Genetic Programming Framework,* Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, p. 1231, Morgan Kaufmann, 13-17 July 1999.

[Dim 99a] Dimopoulos C., Zalzala A.M.S., *A genetic programming heuristic for the one-machine total tardiness problem,* Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on , Volume: 3 , 6-9 July 1999

[Eib 00] Ágoston E. Eiben, Robert Hinterding, Zbigniew Michalewicz, *Parameter Control in Evolutionary Algorithms,* IEEE Trans. on Evolutionary Computation, 2000. http://citeseer.ist.psu.edu/eiben00parameter.html

[Gag 02] C. Gagné, M. Parizeau, *Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computation*, Late Breaking papers at the Genetic and Evolutionary Computation Conference (GECCO-2002), New York, USA, 9-13 July 2002

[Gag 03] Christian Gagné, Marc Parizeau, Marc Dubreuil, *Distributed BEAGLE: An Environment for Parallel and Distributed Evolutionary Computations,* Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS) 2003, http://vision.gel.ulaval.ca/en/publications/Id_439/PublDetails.php

[Gat 97] C. Gathercole, P. Ross, *Tackling the Boolean Even N Parity Problem with Genetic Programming and Limited Error Fitness*, Genetic Programming 1997: Proceedings of the 2nd Annual Conference, pp. 119-127, San Francisco, 1997

[Gib 02] K. A. Gibbs, *Implementation and Evaluation of a Novel Branch Construct for Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2002, http://www.genetic-programming.org/sp2002/Gibbs.pdf

[Gol 00] M. Golub, D. Jakobović, *A New Model of Global Parallel Genetic Algorithm*, Proc. 22th Int. Conference ITI'00, Pula, June 13-16, 2000

[Gol 01] M. Golub, *Poboljšavanje djelotvornosti paralelnih genetskih algoritama*, doktorska disertacija, Fakulet elektrotehnike i računarstva, 2001.

[Gol 01a] M. Golub, D. Jakobović, L. Budin, *Parallelization of Elimination Tournament Selection without Synchronization*, Proc. 5th Int. Conf. on Intelligent Engineering Systems INES 2001, Helsinki, September 16-18., pp. 85-90., 2001.

[Gol 96] M. Golub, *Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova*, magistarski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 1996.

[Gol 98] M. Golub, D. Jakobović, *A Few Implementations Of Parallel Genetic Algorithm*, Proc. 20th Int. Conference ITI'98, Pula, June 14-17 1998, pp.332-337

[Gre 01] William A. Greene, *Dynamic Load-Balancing via a Genetic Algorithm,* 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01) November 07 - 09, 2001 Dallas, Texas, http://csdl.computer.org/comp/proceedings/ictai/2001/1417/00/14170121abs.htm

[Han 04] James V. Hansen, *Genetic search methods in air traffic control,* Computers and Operations Research, v 31, n 3, March, 2004, p 445-459, http://www.sciencedirect.com/

[Hay 96] T. D. Haynes, D. A. Schoenefeld, R. L. Wainwright, *Type Inheritance in Strongly Typed Genetic Programming*, Advances in Genetic Programming II, P. J. Angeline, K. E. Kinnear, (eds), MIT Press, 1996.

[He 03] Xiaoshan He, Xian-He Sun, Gregor Von Laszewski, *A QoS Guided Scheduling Model in Grid Environment*, Journal of Computer Science and Technology, Volume 18 , Issue 4 (July 2003), pp. 442 – 451 http://www.cs.iit.edu/~scs/psfiles/jcst_XHe-5-28.pdf

[Hel 02] Terry M. Helm, Steve W. Painter, Robert Oakes, *A comparison of three optimization methods for scheduling maintenance of high cost, long-lived capital assets,* Winter Simulation Conference Proceedings, v 2, 2002, p 1880-1884

[Hin 97] Robert Hinterding, Zbigniew Michalewicz, Agoston E. Eiben, *Adaptation in Evolutionary Computation: A Survey,* IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, 1997. http://citeseer.ist.psu.edu/hinterding97adaptation.html

[Iba 77] Oscar H. Ibarra, Chul E. Kim, *Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors*, Journal of the ACM, Volume 24 , Issue 2 (April 1977), pp. 280 – 289 http://portal.acm.org/citation.cfm?id=322011&jmp=abstract&dl=GUIDE&dl=ACM

[Jak 97] D. Jakobović, *Adaptive Genetic Operators in Elimination Genetic Algorithm*, Proc. 19th Int. Conference ITI'97, Pula, June 17-20 1997, pp.351-356

[Jak 98] D. Jakobović, M. Golub, *Adaptive Genetic Algorithm*, Proc. 20th Int. Conference ITI'98, Pula, June 14-17 1998, pp.351-356

[Jak 99] D. Jakobović, M. Golub, *Adaptive Genetic Algorithm*, Journal of Computing and Information Technology CIT, Vol. 7, No. 3, September 1999., pp. 229-236

[Jon 98] Albert Jones, Luis C. Rabelo, *Survey of Job Shop Scheduling Techniques,* NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, 1998., http://www.nist.gov/msidlibrary/summary/9820.html

[Kas 99] Joachim Käschel, Tobias Teich, Gunnar Köbernik, Bernd Meier, *Algorithms for the Job Shop Scheduling Problem - a comparison of different methods*, European Symposium on Intelligent Techniques ESIT ´99, June 3-4, 1999, Orthodox Academy of Crete, Greece http://www.erudit.de/erudit/events/esit99/12553_P.pdf

[Kin 93] Kenneth E. Kinnear, *Evolving a Sort: Lessons in Genetic Programming,* Proceedings of the 1993 International Conference on Neural Networks, Vol. 2, pp. 881-888, IEEE Press, 28 March -1 April 1993., ftp://cs.ucl.ac.uk/genetic/ftp.io.com/papers/kinnear.icnn93.ps.Z

[Koz 03] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Kluwer Academic Publishers, 2003.

[Koz 90] John R. Koza, *Genetically Breeding Populations of Computer Programs to Solve Problems in Artificial Intelligence ,* Proceedings of the Second International Conference on Tools for AI, Herndon, Virginia, USA, 1990 http://citeseer.ist.psu.edu/koza90genetically.html

[Koz 90a] John R. Koza, *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,* Stanford University Computer Science Department technical report STAN-CS-90-

1314. June 1990.,
http://www.genetic-programming.com/jkpubs72to93.html

[Koz 92] J. R. Koza, *Genetic Programming – On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[Koz 94] J. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, 1994.

[Koz 95] J. Koza, *Genetic Programming and Hill Climbing*, Machine Learning List, Vol. 7, No. 14, 18.9.1995.
http://www.ics.uci.edu/~mlearn/MLlist/v7/14.html

[Lan 00] W. B. Langdon, W. Banzhaf, *Genetic Programming Bloat without Semantics,* Parallel Problem Solving from Nature - PPSN VI 6th International Conference, LNCS, Vol. 1917, pp. 201-210, Springer Verlag, 16-20 September 2000.,
ftp://cs.ucl.ac.uk/genetic/papers/wbl_ppsn2000.ps.gz

[Lan 02] W. B. Langdon, R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, 2002.

[Lan 05] William Langdon, Steven Gustafson, John Koza, *The Genetic Programming Bibliography*, 2005
http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html

[Lan 97] W. B. Langdon, R. Poli, *Genetic Programming Bloat with Dynamic Fitness,* Technical Report, University of Birmingham, School of Computer Science, Number CSRP-97-29, 3 December 1997.,
ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1997/CSRP-97-29.ps.gz

[Lan 98] W. B. Langdon, *Genetic Programming and Data Structures*, Kluwer Academic Publishers, 1998.

[Lee 04] S. M. Lee, A.A. Asllani, *Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming,* Omega, v 32, n 2, April 2004, p 145-53

[Lee 97] Young Hoon Lee, Kumar Bhaskaran, Michael Pinedo, *A heuristic to minimize the total weighted tardiness with sequence-dependent setups,* IIE Transactions, 29, 45-52, 1997.

[Lek 03] Lekin®, Flexible Job Shop Scheduling System
http://www.stern.nyu.edu/om/software/lekin/

[Leu 04] J. Y-T. Leung (ed.), *Handbook of scheduling,* Chapman & Hall/CRC, 2004.

[Leu 95] J. Y-T. Leung, *A survey of scheduling results for imprecise computation tasks*, Imprecise and approximate computation, Kluwer Academic Publishers, pp. 35-42, 1995

[Lop 01] A. Lopez-Ortiz, *Computational Theory FAQ*,
http://db.uwaterloo.ca/~alopez-o/comp-faq/faq.html

[Mar 04] Goran Martinović, *Postupci raspoređivanja u raznorodnim računalnim sustavima*, doktorska disertacija, Fakultet elektrotehnike i računarstva, Zagreb, 2004.

[Meg 05] Nicole Megow, Marc Uetz, Tjark Vredeveld, *Stochastic Online Scheduling on Parallel Machines,* G. Persiano and R. Solis-Oba (eds): Approximation and Online Algorithms, Lecture Notes in Computer Science 3351, pages 167-180, Springer, 2005.,
http://www.math.tu-berlin.de/~nmegow/muv05sos.pdf

[Mic 92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer-Verlag, Berlin, 1992

[Miy 00] Kazuo Miyashita, *Job-Shop Scheduling with GP,* Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), pp. 505-512, Morgan Kaufmann, 10-12 July 2000.

[Moh 83] Ram Mohan, V. Rachamadugu, Thomas E. Morton, *Myopic Heuristics for the Weighted Tardiness Problem on Identical Parallel Machines,* Working Paper, The Robotics Institute, Carnegie-Mellon University, 1983.

[Mon 01] Patrick Monsieurs, Eddy Flerackers, *Detecting and Removing Inactive Code in Genetic Programs,* 2001, http://alpha.luc.ac.be/~lucp1089/DetectingAndRemovingInactiveCode.pdf

[Mon 95] D. J. Montana, *Strongly Typed Genetic Programming*, Evolutionary Computation, 3(2):199-230, 1995.

[Mor 93] Thomas E. Morton, David W. Pentico, *Heuristic Scheduling Systems,* John Wiley & Sons, Inc., 1993.

[Nei 99] Michael O'Neill, Conor Ryan, *Automatic Generation of Caching Algorithms,* Evolutionary Algorithms in Engineering and Computer Science, pp. 127-134, John Wiley & Sons, 30 May - 3 June 1999., http://www.mit.jyu.fi/eurogen99/papers/oneill.ps

[Nei 99a] Michael O'Neill, Conor Ryan, *Automatic Generation of Programs with Grammatical Evolution*, 1999, http://citeseer.ist.psu.edu/276159.html

[Nor 94] P. Nordin, *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*, Advances in Genetic Programming, pp. 311-331, MIT Press, 1994

[Nor 95] P. Nordin, W. Banzhaf, *Genetic Programming Controlling a Miniature Robot*, Working Notes for the AAAI Symposium on Genetic Programming, pp. 61-67, MIT, Cambridge, 1995.

[Nov 03] Sonja Novković, Davor Šverko, *A Genetic Algorithm With Self-Generated Random Parameters,* Journal of Computing and Information Technology - CIT, Vol. 11, No. 4, December 2003., pp. 271-284

[Ok 00] S. Ok, K. Miyashita, S. Nishihara, *Improving Performance of GP by Adaptive Terminal Selection,* Proc. of the Pacific Rim International Conference on Artificial Intelligence (PRICAI), pp.435-445, 2000, http://staff.aist.go.jp/k.miyashita/publications/PRICAI2000.ps

[Ok 01] S. Ok, K. Miyashita, K. Hase, *Evolving Bipedal Locomotion with Genetic Programming --- Preliminary Report,* Proc. of the Congress on Evolutionary Computation 2001, pp.1025-1032, 2001, http://staff.aist.go.jp/k.miyashita/publications/cec.ps

[Pat 97] Norman Paterson, Mike Livesey, *Evolving caching algorithms in C by genetic programming,* Genetic Programming 1997: Proceedings of the Second Annual Conference, pp. 262-267, Morgan Kaufmann, 13-16 July 1997., http://www.dcs.st-and.ac.uk/~norman/Pubs/cache.ps.gz

[Pen 00] Carlos Andrés Peña-Reyes, Moshe Sipper, *Evolutionary computation in medicine: an overview,* Artificial Intelligence in Medicine Volume 19, Issue 1, 1 May 2000, Pages 1-23, http://www.sciencedirect.com/

[Pin 04] M. Pinedo, *Offline Deterministic Scheduling, Stochastic Scheduling, and Online Deterministic Scheduling: A Comparative Overview*, Handbook of Scheduling, J. Y-T. Leung (ed.), Chapman & Hall/CRC, 2004.

[Pol 99] Riccardo Poli, *Parallel Distributed Genetic Programming,* New Ideas in Optimization, McGraw-Hill, 1999., http://citeseer.ist.psu.edu/328504.html

[Pru 04] K. Pruhs, J. Sgall, E. Torng, *Online scheduling*, Handbook of Scheduling, J. Y-T. Leung (ed.), Chapman & Hall/CRC, 2004.

[Rus 97] R. M. Russell, J. E. Holsenback, *Evaluation of greedy, myopic and less-greedy heuristics for the single machine, total tardiness problem,* Journal of the Operational Research Society (1997), 48, 640-646

[Sch 94] E. Schöneburg, F. Heinzmann, S. Feddersen, *Genetische Algorithmen und Evolutionsstrategien*, Addison-Wesley, 1994.

[Ser 01] F. Seredynski, J. Koronacki, C. Z. Janikow, *Distributed multiprocessor scheduling with decomposed optimization criterion,* Future Generation Computer Systems, Volume 17, Issue 4 , January 2001, Pages 387-396, http://www.sciencedirect.com/

[Ser 99] F. Seredynski, J. Koronacki, C. Z. Janikow, *Distributed Scheduling with Decomposed Optimization Criterion: Genetic Programming Approach,* Internationa Parallel and Distributed Processing Symposium, workshop: Bio-Inspired Solutions to Parallel Processing Problems, 1999., http://ipdps.eece.unm.edu/1999/biosp3/seredyns.pdf

[Sil 03] Sara Silva, Jonas Almeida, *Dynamic Maximum Tree Depth - A Simple Technique for Avoiding Bloat in Tree-Based GP ,* Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2003), pp. 1776-1787, Genetic and Evolutionary Computation Conference (GECCO-2003), Chicago, Illinois USA, July-2003, http://cisuc.dei.uc.pt/ecos/view_pub.php?id_p=109

[Sou 98] T. Soule, *Code Growth in Genetic Programming,* PhD Thesis, University of Idaho, 1998., http://www.cs.uidaho.edu/~tsoule/research/the3.ps

[Sri 94] M. Srinivas, L. M. Patnaik, *Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*, IEEE Trans. Systems, Man and Cybernetics, April 1994.

[Sri 94a] M. Srinivas, L. M. Patnaik, *Genetic Algorithms: A Survey*, IEEE Computer, June 1994.

[Tac 94] W. A. Tackett, *Recombination, Selection and the Genetic Construction of Computer Programs*, PhD thesis, University of Southern California, Department of Electrical Engineering Systems, 1994.

[Tai 03] E. Taillard, *Scheduling Instances*, 2003. http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html

[Tal 03] W. A. Talbott, *Automatic Creation of Team-Control Plans Using an Assignment Branch in Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2003, http://www.genetic-programming.org/sp2003/Talbott.pdf

[Tel 95] A. Teller, M. Veloso, *PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System*, Technical Report CMU-CS-95-101, Department of Computer Science, Carnegie Mellon University, 1995.

[Vaz 00] Manuel Vazquez, L. Darrell Whitley, *A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem,* Proceedings of the Genetic

and Evolutionary Computation Conference (GECCO '00), Las Vegas, Nevada, USA, July 8-12, 2000

[Wal 05] Scott S. Walker, Robert W. Brennan, Douglas H. Norrie, *Holonic Job Shop Scheduling Using a Multiagent System*, IEEE Intelligent Systems, 2/2005, pp. 50-57

[Wal 96] P. Walsh, C. Ryan, *Paragen: A Novel Technique for the Autoparallelisation of Sequential Programs Using Genetic Programming*, Genetic Programming 96: Proceedings of the 1st Annual Conference, pp. 406-409, MIT Press, 1996.

[Wan 03] J.-S. Wang, *Influences of Function Sets in Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2003, http://www.genetic-programming.org/sp2003/Wang.pdf

[Wol 97] D. H. Wolpert, W. G. Macready, *No Free Lunch Theorems for optimization*, IEEE Trans. on Evolutionary Computation, 1(1):67-82, 1997.

[Yin 03] Wen-Jun Yin, Min Liu, Cheng Wu, *Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming,* Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, pp. 1050-1055, IEEE Press, 8-12 December 2003.,

[Zha 96] B.-T. Zhang, H. Mühlenbein, *Adaptive Fitness Functions for Dynamic Growing/Pruning, of Program Trees*, Advances in Genetic Programming 2, pogl. 12, pp.241-256, MIT Press, 1996.

Prilog A: Parametri ispitnih primjera

U ovom prilogu dani su parametri ispitnih primjera za svako pojedino okruženje. Navedeni parametri koriste se za oblikovanje ispitnih primjera na način opisan u odgovarajućim poglavljima, gdje su opisana i njihova značenja. Za potrebe generiranja slučajnih brojeva po jednolikoj raspodjeli korišten je generator slučajnih brojeva ugrađen u MS Visual Studio .NET 2003 razvojni alat. Ispitni primjeri podijeljeni su na skup primjera za učenje i skup za ocjenjivanje u svakom okruženju. Vrijednosti su za svaki parametar dane u obliku niza, gdje pojedini element u nizu odgovara dotičnom ispitnom primjeru. Ukoliko je niz vrijednosti nekoga parametra jednak nizu u nekome drugome okruženju, to je napomenuto umjesto ponovnog navođenja cijeloga niza.

A1. Okruženje jednoga stroja

Skup primjera za učenje (100 primjera)

Broj poslova: 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100

Vrijednosti parametra R : .2 .4 .6 .8 1. .2 .4 .4 .6 .8 .2 .4 .8 .8 1. .2 .2 .4 .8 1. .4 .4 .4 .6 .8 .2 .4 .8 1. 1. .2 .4 .6 .6 .8 .2 .4 .4 .8 1. .2 .2 .4 .8 .8 .2 .2 .6 .8 1. .4 .4 .6 .8 1. .4 .4 .6 .6 .8 .2 .4 .6 .8 .8 .2 .2 .6 .8 1. .2 .2 .2 .4 .6 .2 .2 .4 .6 .8 .2 .4 .6 .8 .8 .2 .4 .4 .6 1. .2 .4 .6 .8 1. .2 .4 .4 .6 1.

Vrijednosti parametra T : .4 .8 .6 .2 .4 .8 .6 .8 .2 .4 .2 .2 .4 .6 .4 .4 .8 .6 .4 .6 .4 .6 .8 .2 .2 .2 .6 .2 .4 .6 .8 .4 .6 .8 .4 .4 .6 .8 .8 .2 .2 .4 .8 .6 .8 .2 .6 .8 .8 .2 .4 .6 .8 .6 .2 .6 .8 .2 .8 .8 .2 .4 .8 .2 .8 .6 .8 .6 .4 .4 .2 .4 .6 .8 .2 .2 .8 .8 .4 .8 .2 .4 .8 .2 .8 .2 .4 .8 .4 .2 .2 .8 .2 .6 .2 .4 .4 .6 .2 .6

Skup primjera za ocjenjivanje (600 primjera)

Broj poslova: 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100

100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100

Vrijednosti parametra R : .2 .2 .2 .2 .4 .4 .4 .4 .6 .6 .6 .6 .8 .8 .8 .8 1. 1. 1. 1.

Vrijednosti parametra T : .2 .4 .6 .8

A2. Okruženje paralelnih jednolikih strojeva

Skup primjera za učenje (120 primjera)

Broj poslova: 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 25 25 25 25 25 25 25 25 25 25 50 50 50 50 50 100 100 100 100

Broj strojeva: 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

Vrijednosti parametra R : .4 .4 .6 .8 1. .4 .4 .4 .6 .8 .4 .4 .8 .8 1. .4 .6 .4 .8 1. .4 .4 .4 .6 .8 .6 .4 .8 1. 1. .4 .4 .6 .6 .8 .4 .4 .4 .8 1. .4 .4 .4 .8 .8 .4 .4 .6 .8 1. .4 .4 .6 .8 1. .4 .4 .6 .6 .8 .4 .4 .6 .8 .8 .4 .4 .6 .8 1. .4 .6 .4 .4 .6 .4 .6 .4 .6 .8 .4 .4 .6 .8 .8 .6 .4 .4 .6 1. .6 .4 .6 .8 1. .6 .4 .4 .6 1. .4 .6 .4 .4 .6 .6 .8 .8 1. 1. .4 .4 .4 .6 .6 .6 .8 .8 1. 1.

Vrijednosti parametra T : .4 .8 .6 .2 .4 .8 .6 .8 .2 .4 .0 .2 .4 .6 .4 .4 .8 .6 .4 .6 .4 .6 .8 .0 .2 .2 .6 .2 .4 .6 .8 .4 .6 .8 .4 .4 .6 .8 .8 .0 .2 .4 .8 .6 .8 .2 .6 .8 .8 .2 .4 .6 .8 .6 .0 .6 .8 .2 .8 .8 .2 .4 .8 .2 .8 .6 .8 .6 .4 .4 .2 .4 .6 .8 .0 .2 .8 .8 .4 .8 .2 .4 .8 .2 .8 .2 .4 .8 .4 .0 .2 .8 .2 .6 .2 .4 .4 .6 .2 .6 .8 .6 .6 .4 .4 .2 .4 .0 .8 .6 .6 .4 .4 .2 .4 .2 .4 .6 .4 .2

Skup primjera za ocjenjivanje (600 primjera)

Broj poslova: jednaki kao i u skupu za ocjenjivanje za jedan stroj

Broj strojeva: 3 6 10 15 20

Vrijednosti parametra R : jednake kao i u skupu za ocjenjivanje za jedan stroj

Vrijednosti parametra T : jednake kao i u skupu za ocjenjivanje za jedan stroj

A3. Okruženje paralelnih nesrodnih strojeva

Skup primjera za učenje (120 primjera)

Broj poslova: jednaki kao i u skupu za učenje za paralelne jednolike strojeve

Broj strojeva: jednaki kao i u skupu za učenje za paralelne jednolike strojeve

Vrijednosti parametra R : .4 .4 .6 .8 1. .4 .4 .4 .6 .8 .4 .4 .8 .8 1. .4 .6 .4 .8 1. .4 .4 .4 .6 .8 .6 .4 .8 1. 1. .4 .4 .6 .6 .8 .4 .4 .4 .8 1. .4 .4 .4 .8 .8 .4 .4 .6 .8 1. .4 .4 .6 .8 1. .4 .4 .6 .6 .8 .4 .4 .6 .8 .8 .4 .4 .6 .8 1. .4 .6 .4 .4 .6 .4 .6 .4 .6 .8 .4 .4 .6 .8 .8 .6 .4 .4 .6 1. .6 .4 .6 .8 1. .6 .4 .4 .6 1. .4 .6 .4 .4 .6 .6 .8 .8 1. 1. .4 .4 .4 .6 .6 .6 .8 .8 1. 1.

Vrijednosti parametra T : .4 .8 .6 .2 .4 .8 .6 .8 .2 .4 .0 .2 .4 .6 .4 .4 .8 .6 .4 .6 .4 .6 .8 .0 .2 .2 .6 .2 .4 .6 .8 .4 .6 .8 .4 .4 .6 .8 .8 .0 .2 .4 .8 .6 .8 .2 .6 .8 .8 .2 .4 .6 .8 .6 .0 .6 .8 .2 .8 .8 .2 .4 .8 .2 .8 .6 .8 .6 .4 .4 .2 .4 .6 .8 .0 .2 .8 .8 .4 .8 .2 .4 .8 .2 .8 .2 .4 .8 .4 .0 .2 .8 .2 .6 .2 .4 .4 .6 .2 .6 .8 .6 .6 .4 .4 .2 .4 .0 .8 .6 .6 .4 .4 .2 .4 .2 .4 .6 .4 .2

Skup primjera za ocjenjivanje (600 primjera)

Broj poslova: jednaki kao i u skupu za ocjenjivanje za paralelne jednolike strojeve

Broj strojeva: jednaki kao i u skupu za ocjenjivanje za paralelne jednolike strojeve

Vrijednosti parametra R: jednake kao i u skupu za ocjenjivanje za paralelne jednolike strojeve

Vrijednosti parametra T: jednake kao i u skupu za ocjenjivanje za paralelne jednolike strojeve

A4. Okruženje proizvoljne obrade

Skup primjera za učenje (160 primjera)

Broj poslova: 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100

Broj strojeva: 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

Vrijednosti parametra R: .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8

Vrijednosti parametra T: .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1.

Skup primjera za ocjenjivanje (320 primjera)

Broj poslova: 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100 12 12 12 12 12 25 25 25 25 25 50 50 50 50 50 100 100 100 100 100

Broj strojeva: 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 20 20 20 20 20

20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

Vrijednosti parametra R : .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 .2 .2 .2 .2 .2 .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8 1. 1. 1. 1. 1. .4 .4 .4 .4 .4 .6 .6 .6 .6 .6 .8 .8 .8 .8 .8

Vrijednosti parametra T : .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1. .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .0 .2 .4 .6 .8 .2 .4 .6 .8 1. .2 .4 .6 .8 1. .2 .4 .6 .8 1.

Prilog B: Izvedena pravila raspoređivanja

U prilogu su navedena odabrana rješenja postupaka izvođenja pravila raspoređivanja. Svako je rješenje prikazano u obliku funkcije prioriteta koja se koristi u sprezi sa definiranim meta-algoritmom za odgovarajuće okruženje.

B1. Okruženje jednoga stroja

Problem: $1||\sum w_j T_j$

Funkcija prioriteta:

$\pi = (((pt/w)-pos(((pt+Nr)/pt)))+(((((((SL+pt)*(((((SL+SL)*SL)/w)-w)-w))/Nr)/(Nr/w))+pos(pos(((((pt+SL)+(((SL+(pt+SL))*(SL+w))/w))/(SD/pt))*(((((SL+SL)*SL)/w)-w)-((((SL+(pt/w))+(SL/((w+SD)/(Nr+SL))))*(Nr+SPr))+pos((((Nr+SD)/(Nr+SL))+(pt+((SL+((((Nr+pt)+(SD+Nr))+pt)/(Nr+SL)))+((pt/w)-pos((Nr/pt))))))))))))/SPr)+SL))$

Problem: $1||\sum w_j U_j$

Funkcija prioriteta:

$\pi = (((((pt+pt)+(SL+pos(pos(((w/SL)*(pt/w))))))/w)+(pos((pos(pos(((pt-((((w/SL)*((SD-dd)+(SL-dd)))*pos(((((N-SL)+(SP-dd))*SD)/w)))*(N/(((pt+(w/SL))+(SL+pos((SPr-SL))))/SD))))+SD)))+(pt+(1/w))))+pos(pos(((pt-((((w/SL)*((SD-dd)+(pt/SL)))*pos((pos((((w+(SD*pt))*SD)/w))/w)))*dd))+SD)))))-SPr)$

Problem: $1|r_j|\sum w_j T_j$

Funkcija prioriteta:

$\pi = ((SL+((SPr/(((((pt/w)+((Nr+(Nr/w))*AR))+((SPr/(N/((N-SPr)+(SPr+SL))))/(SPr*((SL+(pt+SL))/SPr))))/((pt-SPr)+(SPr+SL)))+w))/(N+N)))+((((pt/w)+((Nr+(Nr/w))*AR))+((SPr/((N/((pt-SPr)+(SPr+SL)))+w))/((pos((N*(dd+((N/(N+((N+N)*Nr)))/w))))*(N/(w/(SL+((SD-SPr)+SL)))))*((SL+(pt/N))/SPr))))-(((SL/(((N+N)*Nr)/((N*(dd+(AR/w)))-((SL+(Nr/w))+((SP+w)-pos(((pt+SL)*(dd+((N+N)/w)))))))))-(Nr/w))/((pt-SPr)+(((pt+SL)/pt)*w)))))$

Problem: $1|prec|\sum w_j T_j$

Funkcija prioriteta:

$\pi = ((((pt+(((SL-Nr)+(pt/w))+((SL-Nr)+((pt+SL)/w))))+((pos((pt*((SPr/LVL)*(((((SD/dd)/(pt*((SD*(Nr+(((SC*LVL)*(SPr/LVL))*(Nr*(SPr/LVL)))))*((SD*SPr)-(((SL-Nr)+((pt+SL)/w))*(dd*(SPr/LVL)))))))-(((SPr-LVL)+((SL-(SPr/Nr))+(pt/w)))*((SPr-SD)*(SL-(SPr/Nr)))))/(pt*((SD*SPr)*(((SL-Nr)+((SL-(SPr/Nr))+(SPr/Nr)))-(((SL-pt)+((SC*LVL)+((pt+SL)/w)))*((SPr-SD)*(SPr/LVL)))))))-(((SL-(SPr/Nr))+((SL-(SPr/Nr))+(pt/w)))*((SPr-SD)*(SPr/LVL)))))))+(pt+SL))-(SD/(SPr-w))))*((SPr-(SC*LVL))*(((pt*(((pt*dd)+((pt*((SD*SPr)*(((Nr/w)*LVL)-(((SC*LVL)+((SL-Nr)+((pt+SL)/w)))*((SPr-SD)*(SPr/LVL))))))-(SD/((SPr-SD)*(SPr/(SD/w))))))+((SL-pt)+SPr)))/w)/((pt+SL)/w))))/((Nr*((Nr/w)*LVL))+((SD/w)+(pt+SL))))$

Problem: $1|s_{ij}|\sum w_j T_j$

Funkcija prioriteta:

$\pi = (((STP*Nr)+((STP+(SPr+pos(((SL+(pt+((pt+((((((STP+pt)*Sav)/w)+pt)+((Sav+Sav)+((((w+w)*pt)/w)/(Nr/SL))))+((SPr+SPr)/SPr)))+(((SL+((pt/dd)+pt))+(STP*STP))+(((((S$

TP+pt)*Sav)/w)/w)/(pt/w))))))+(((Nr+((((SPr+(((SPr+SPr)+STP)+(STP*STP)))+(((SPr+(
STP*STP))+(Nr+((STP+pt)+(STP+pt))))+pt))+(w+SPr))+STP))+(Nr+(((w+((w+SL)*(((S
TP+pt)*Sav)/w)))+(w+((pt+(Sav*(((STP+pt)*Sav)*Sav)))+pos((STP*STP)))))/(SPr+(((w
+(pt/(pt/w)))+(w+((((STP+pt)*(pt/w))/w)+pt)))-w)))))+(w+SPr))))))-(w+pt)))+(STP*SL))

Problem: $1\left|s_{ij}, prec\right|\sum w_j T_j$

Funkcija prioriteta:

$\pi =$((pos((((Nr*(((SC*Nr)*(((LVL/(SC*(SD*pt)))+((STP/(pt/dd))*((pt+(STP*STP))/pt)))
/(((pt/w)-pt)/((LVL/(SC*w))/(((((((SD*STP)+(LVL/STP))/w)+SL)/dd)/(SD+pos((((STP/(
SC*Nr))*pt)*(SL*pt)))))+pt)))))/w))+(SD*((pt+((((STP*((Nr/pt)*(SD*((STP*STP)*SC)))
)+(((SC*Nr)+pos((((STP/Nr)*pt)*(SD*pt))))/w))/w)*(((pos((pt/w))*pos((STP*STP)))+(pt
/w))+(((pt+(LVL*Nr))/w)+SL))))-SC)))+(STP-(((STP-SC)/pt)+STP))))+((Nr+((((pt/w)-S
C)/w)+STP))*((((((STP+dd)/(SC*(SD*pt)))+(Sav/w))/w)+(((((SC*Nr)*(STP*(SD*((pt/w)
(STP-SC)))))/w)((STP+pt)+((STP*(SD*((pt/w)*(STP-SC))))/(SD+pos((((STP/(SC*Nr))
*pos(((STP+SL)+((STP/(SC*Nr))*SD))))*(SD*pt)))))))*((pos((Sav/w))*STP)+(pt/w))))+(
(((pt-SC)*(((STP*(STP*(SD*pos((w+STP)))))+(pos(((STP*STP)/w))*((Nr*STP)/(SC*Nr
))))/w))/w)*pt))))+((((((pt+(STP*STP))+w)*(((STP*w)+(((((SD+pos(((((Nr*STP)/(SC*Nr
))*pos(pos((dd+pt))))*(SD*STP))))/w)*Nr)*pos((Sav+STP)))*((pt*pos((w+STP)))+(pt/w)
)))+((((STP*STP)+(((((SD+pos((((STP/(SC*Nr))*pt)*(SD*pt))))/w)*Nr)*(STP+pt))*((pos
((Sav/w))*STP)+(pt/w))))+(((STP*(((STP*(STP*(SD*((STP*STP)*STP))))+(STP-SL))/w
))/w)*pt))*pt)))+STP)*(pt*((Nr/pt)*(STP+dd))))/((LVL/STP)/(Nr+(((((STP-SC)/w)+(STP
*Nr))+(((pt/w)-pt)/((LVL/(SC*w))/(((LVL/dd)/(SD+((STP-SC)/w)))+pt))))+STP)))))

Problem: $1\left|r_j, s_{ij}\right|\sum w_j T_j$

Funkcija prioriteta:

$\pi =$((((Sav*((STP/((((((w*(AR+STP))+AR)-w)+STP)/(AR/STP))+((pos((w+(((AR-w)/w)
/w)))*w)+(pt/SPr))))+(((pos(pos(pos(((w*STP)+(((STP*SL)-STP)-((w*(AR+STP))+AR))
))))-(((pos((((STP*SL)-Sav)-((w*Nr)+AR)))/((AR/w)+((w+1)-Nr)))+AR)/((w/(AR/w))/(A
R/STP))))-Nr)+((Sav*STP)/w))))/(((STP/Sav)+((SL+(AR+(((((AR-w)/((pt/w)+SL))/w)+((
(w*(w+SL))+((pt/w)-w))-Nr))+(pt/w))))+(Nr*pos((AR+pos(((((w-pt)-(Sav/((pt/w)-w)))-(
w+SL))+STP)))))))-(((SL+((STP/((pos((((STP*SL)-Sav)-((w*(AR+STP))+AR)))/((AR/w)
+((pt/w)-Nr)))+AR))+(AR+pos(((pos(((pt/w)-((w/SL)/Nr)))*w)/w)))))+Nr)-((w-(pos((((ST
P-(((pt-w)/Sav)+SPr))-Nr)+STP))/Nr))+SL))))+((STP*SL)-((SD+((pos(((((((w-pt)+((AR+
STP)+(w+SL)))+(((pt/w)+SL)/(pt/w)))/((AR/w)/w))+((pos((w+(pt-w)))*w)+(pt/w)))-(STP
*SL)))*w)+(pt/w)))-(STP*SL))))+(Sav*(((((SD/((AR+pos((((pt/w)-(w+SL))+STP)))/(AR
+(w+SL))))/(Nr+SD))+((SL+((((((w*(AR+STP))+AR)-w)+STP)/(AR/STP))+(((((AR-w)/
w)/w)+((AR+STP)+(pt/w)))+(pt/w))))+(Nr*(AR+pos(((AR-w)+STP))))))+(STP/w))+(pt/(
w-(pos(((((AR+STP)-pos((w+(pt-w))))-(pt/w))+STP))/Nr))))))

Problem: $1\left|r_j, prec\right|\sum w_j T_j$

Funkcija prioriteta:

$\pi =$((((pt/w)/LVL)*(((pt/((pt*((AR+pt)/LVL))-pt))+SL)+pt))*((((((LVL*pos((AR*w)))+(
pos(((pos((AR*(AR-SC)))-pt)*((SPr*(((((w-Nr)*SC)*((AR/LVL)*SD))*((dd/LVL)-SC))*
((((AR-pt)*dd)*AR)*w)))*pos(((((AR+pt)/LVL)-((pt/w)/LVL))*pt)))))+SL))+pt)+pt)+(((p
t/((((AR+pt)/LVL)-((pt/w)/((((((AR+pt)/LVL)-((pt/w)/LVL))*(pt*((AR+pt)/LVL)))+SL)+
pt)))*(AR*w)))+SL)+pt))+((((((pos((AR+((LVL-SC)-pt)))/dd)*AR)*AR)+((AR-SC)+AR)
)+((((((pt/(((((AR-SC)-pt)*(AR-SC))*AR)-((((AR+pt)/LVL)-((pt/w)/LVL))*(AR*((AR+pt
)/LVL)))))+SL)+pt)*w)/LVL)-((((((dd-LVL)-SC)*w)*(((w-Nr)*SC)*pos(pos((AR-SC)))))

+(((AR*(pt/AR))/((AR+(AR-SC))-pt))*(0-pt)))/((((((AR-((LVL*((pt/w)/LVL))-SC))*((w*(((w-Nr)*SC)*pt))-pt))*(pt*w))*AR)*AR)-pos(((AR-SC)*w))))))*(((((((AR-SC)-pt)*AR)*(AR+(dd*(AR-SC))))*(pt/w))*((((((AR-SC)-pt)*LVL)*AR)*(((dd/LVL)/(((AR-SC)-pt)*AR))+(LVL*((AR-SC)-SC))))*(pos(((((AR*pt)*(w-SC))*(((pos((AR*(AR-SC)))-pt)*(pt*((AR+pt)/LVL)))*(AR+pt)))*(AR+AR)))+pt)))*pos((((((AR+pt)/LVL)-((AR*(pt/dd))/((AR+((dd/LVL)-SC))-pt)))-(((LVL*pos((AR*((AR+pt)/LVL))))/((dd*((((((dd/LVL)-SC)*SC)*(((dd/LVL)-SC)+AR))*AR)-SC))-pt))*((((AR+w)/(((AR-SC)-pt)*AR))-((pt/w)/((pt-(LVL*pos((AR*w))))+pt)))*(AR*w))))*((((((AR-SC)-pt)*LVL)*AR)*(AR+(SPr*(AR-SC))))*(((AR+pt)/LVL)/LVL))))))))

B2. Okruženje paralelnih jednolikih strojeva

Problem: $Q|\ |\sum w_j T_j$

Funkcija prioriteta:

$\pi = (\text{pos}(\text{pos}(((((\text{SLs}+(\text{pt}+\text{w}))+\text{SLs})/\text{w})+\text{SLs})))+((\text{SLs}+(\text{SLs}/\text{w}))/\text{w}))$

Problem: $Q|s_{ij}|\sum w_j T_j$

Funkcija prioriteta:

$\pi = ((((\text{SD}/\text{w})+(\text{pos}(((\text{pos}(((((\text{SLs}*\text{SPD})+((\text{Nr}*\text{STP})/\text{w}))+(\text{Nr}+(\text{STP}*((\text{STP}*\text{SLs})/(\text{SLs}/\text{SPr})))))/((\text{STP}/\text{w})+(\text{w}+((\text{STP}*\text{Nr})/\text{STP})))))+(\text{STP}*\text{Nr}))+((\text{pt}+((\text{w}*\text{Nr})+(\text{pt}*\text{SPD})))*(((\text{SPD}*\text{SPD})/(\text{w}/\text{pt}))*\text{SPD}))))*(\text{SPD}*\text{Nr})))+\text{pos}(((((((((\text{STP}/(\text{w}*\text{Nr}))*\text{SPD})+(((\text{Nr}+(\text{STP}*((\text{STP}*\text{SLs})/(\text{SLs}/\text{SPr}))))/\text{STP})*\text{Nr}))+(\text{w}+(((\text{w}*((\text{SPr}*\text{Sav})/(\text{SLs}/\text{SPr})))+((\text{SPD}*((\text{SPD}-\text{w})/\text{Msm}))*\text{SPD}))+(\text{SLs}*((\text{STP}*\text{SLs})/(\text{SLs}/\text{SPr}))))))/\text{STP})+\text{pos}(((((\text{SPD}+((\text{STP}/\text{w})+(\text{SPD}+((\text{STP}*\text{Nr})/\text{STP}))))+((\text{Nr}+((\text{STP}*\text{Nr})*(((\text{SPD}*\text{SPD})/(\text{w}/\text{pt}))*\text{SPD})))+(\text{Nr}+(\text{STP}*(\text{STP}*\text{SLs})))))/\text{w})/((\text{SPr}*\text{w})+((\text{SPD}*\text{Sav})+((\text{w}*((\text{Nr}*\text{STP})/(\text{pt}+\text{w})))/(\text{w}*((\text{Nr}*\text{pt})/(\text{pt}+\text{w}))))))))+(((\text{SD}/\text{w})+(\text{pos}((\text{w}+((\text{STP}*\text{SLs})+(\text{w}+(\text{STP}*((\text{STP}*\text{SLs})/\text{pos}((\text{STP}/\text{dd}))))))))*(\text{SPD}*(\text{w}/\text{Msm}))))+((\text{w}+\text{pos}((\text{SPD}*((\text{Nr}*\text{STP})/(\text{pt}+\text{w})))))*\text{w})))/(\text{SPD}/(\text{w}/\text{pt})))))+((\text{pt}+\text{w})*((((((\text{w}*((\text{Nr}*\text{pt})/(\text{pt}+\text{w})))+(\text{SPD}/(((((\text{w}*((\text{Nr}*\text{STP})/(\text{pt}+\text{w})))/(\text{w}*((\text{Nr}*\text{pt})/(\text{pt}+\text{w}))))+\text{SLs})*\text{w})-(((\text{w}*((\text{STP}*\text{Nr})*(((\text{SPD}*\text{SPD})/(\text{w}/\text{pt}))*\text{SPD})))+(((\text{w}*((\text{Nr}*\text{pt})/(\text{pt}+\text{w})))/((\text{w}-(\text{SPD}*\text{SPD}))+(\text{SPD}/(\text{STP}*\text{SLs}))))*\text{SPD}))/(((((\text{SPD}*\text{SPD})/((\text{Nr}*\text{pt})/\text{SPr}))+(\text{SPr}*\text{w}))+(\text{Nr}+(\text{STP}*\text{SLs})))*\text{SPr})))))+((\text{SPD}+((\text{STP}/\text{w})+(\text{SPD}+((\text{STP}*\text{Nr})/\text{STP}))))/((((\text{w}-(\text{STP}/\text{w}))-\text{STP})+(\text{w}+\text{STP}))-(((\text{w}*(\text{pt}+\text{STP}))+\text{w})/(\text{Nr}*\text{pt})))))/\text{w})-\text{w})*\text{SPD})))$

Problem: $Q|r_j|\sum w_j T_j$

Funkcija prioriteta:

$\pi = (((((\text{SLs}+(\text{pt}*(\text{SPD}/\text{w})))+(((\text{SLs}+(\text{pt}*(\text{SPD}/\text{w})))+\text{w})+(((((\text{w}/(\text{w}*\text{SPD}))/\text{w})+(\text{Msm}/(((\text{SLs}-((\text{w}+\text{Msm})/\text{w}))/((((\text{SLs}-((\text{w}+\text{Msm})/\text{w}))/(\text{SPr}+(\text{SLs}-\text{SPD})))+(\text{w}/\text{SLs}))+(\text{SPr}/(\text{pt}*(\text{SPD}/\text{w})))))/(\text{pt}*((((((1+(\text{w}*\text{SPD}))+(\text{SLs}-\text{SPD}))/((\text{w}/((\text{w}*\text{SPr})*\text{SPD}))/\text{w}))+(\text{dd}/\text{w}))+\text{SLs})/((\text{dd}+\text{Msm})/(\text{Msm}*((\text{SPD}/\text{w})+\text{w})))))))+(\text{dd}/\text{Nr}))+(\text{SLs}-\text{SPD}))))+(((((\text{w}/(\text{w}*\text{SPD}))/\text{w})+(((((\text{w}/(\text{w}*\text{SPD}))/\text{w})/((\text{dd}+\text{Msm})/((\text{SLs}-((\text{w}/(\text{Msm}*(\text{w}*\text{w})))/((\text{Msm}*((\text{w}*\text{SPD})+\text{Nr}))/(\text{Msm}*((\text{w}+\text{w})+\text{w})))))+(\text{SLs}-((\text{Nr}-(\text{SPD}+\text{SLs}))-((\text{w}/(\text{w}*\text{SPD}))/\text{w})))))/\text{w})/(\text{w}*((((\text{w}/(\text{w}*\text{SPD}))/\text{w})+(\text{dd}/\text{Nr}))+\text{SLs})))+((\text{SLs}+((\text{SLs}+(\text{pt}*(\text{SPD}/\text{w})))/(((((\text{w}/(\text{w}*\text{SPD}))/\text{w})+((((((\text{Nr}-(\text{dd}/\text{Nr}))-\text{w})/(\text{w}*(\text{Nr}-(\text{SPD}+(\text{w}*\text{SPD})))))+(\text{dd}/\text{w}))+\text{SLs})/\text{pt}))+((\text{Msm}*(\text{SPr}/(\text{pt}*(\text{SPD}/\text{w}))))+\text{Msm}))+(\text{SLs}-\text{SPD}))))/((((\text{w}/((((\text{w}/(((\text{w}/(\text{w}*\text{SPD}))/\text{w})+(\text{SPD}/\text{w})))/\text{w})*(\text{dd}+\text{Msm}))+\text{SLs}))/(\text{Msm}/(\text{Nr}-(\text{dd}/\text{Nr}))))+((\text{Msm}*((\text{w}/(\text{w}/((\text{Nr}-(\text{w}*\text{SPr}))-((\text{w}/(\text{w}*\text{SPD}))/\text{w})))))/\text{w}))+(\text{SLs}*\text{SPD})))+(\text{SLs}-\text{SPD}))))+(\text{SLs}-\text{SPD})))+(\text{w}*\text{SPD}))+\text{pos}((\text{SLs}-((\text{Nr}/(\text{Nr}+\text{pt}))*(\text{pos}(((((\text{SPr}/(\text{pt}*(\text{SPD}/\text{w})))+(\text{w}/(\text{w}*\text{SPD})))+(\text{pt}+((\text{w}/((\text{SPD}/\text{w})*\text{SPD}))+\text{SPD})))/\text{w}))+\text{SPD})))))$

Problem: $Q|r_j|C_{\max}$

Funkcija prioriteta:
π = (sqr(((pos(pos(((pos(pos(((Nr/SPD)-(SPD*pt))))*((SPD*pt)/(SPD+pt)))))+pos(((Nr/SPD)-(SPD*pt))))-SPD))+((((pos(pos((Msm*SPr)))+pos((pos(((((pt-Msm)-pos(pos(((Nr/SPD)-(SPD*pt)))))*(pos((pt+Msm))*(Nr-AR)))+(pt*SPr)))-(Nr/SPD))))*(SPD/pt))-(AR+((SPr+pt)+(pt-((pt/SPD)-pos((pt-pos(((Msm-SPD)/pos((((pt*((SPr-Nr)-sqr((Msm-SPr))))/(SPD/pt))*sqr(sqr(SPr))))))))))))/(sqr(pt)+(SPr+pt))))

Problem: $Q\left|r_j, s_{ij}\right|\sum w_j T_j$
Funkcija prioriteta:
π = (((((((((pos((((SPD-w)*pos(((pt+pt)+pt)))/(w+SPD)))+((SPD-w)*pt))+pt)+((((SPD+SLs)+((SPD-w)+STP))+Nr)/((pt/Sav)+SPD)))+((Sav/w)+SLs))+SPD)+(((SPD-w)*((pt/w)+SPD))/(w+SPD)))-(((SPD+((SPD+SLs)*SPD))+((pos((SD/pt))-w)*pt))+Nr))+((((pt*pos(((pt/((SPD-w)*pt))+pos(((w+SPD)+(SPD-w))))))+pos(((((((((pt/w)+SPD)+SPD)+SLs)+(Nr/pt))-w)*pos((pt-SPD)))/(((pos((SD/pt))-SPD)*((Nr/pt)-SPD))+(((SPD+SLs)/((pos((SD/pt))-w)*((SPD-w)*pos(((pt+pt)+pt)))))+((pt+pt)+pt))))))/(((((((((SPD-w)+(SPD*w))+pt)+(SPD-(Sav/(((pt/Sav)-dd)/SLs))))+(((SPD*w)*pt)-SPD))+(SPD+SLs))-w)+SPD)+pos(((pt+(((SPD-w)*pos(((pt+pt)+pt)))/(w+SPD)))+((((pt+(((SPD*w)*pt)-SPD))*SPD)-SLs)+pos((w/pt)))))))*SPD))+((pt+(((((((((pos((pt+SPD))+SLs)+(Nr/pt))+((pt+(((SPD*w)*pt)-SPD))*SPD))+pos((pt+((SPD-w)-(SPD*w)))))+SPr)+pos(((pt/((STP-w)*((SPD*w)*pt)))+SPD)))+((pt+pt)+pt))*SPD))*SPD))

Problem: $Q\left|r_j, s_{ij}\right|C_{\max}$
Funkcija prioriteta:
π = ((((STP*((pt*SPD)/Nr))+pt)*((SPD+Msm)/(pos((((Nr+(((((((pt/Nr)-STP)+(((STP/pt)-(((pt/Nr)-STP)/(STP/Nr)))/(STP/pt)))-AR)/(pt*SPD))*STP)+(Nr+Nr)))*Nr)+(((pos((1/STP))-((pt/Nr)-STP))/AR)-STP)))+STP)))+(((Nr*STP)-pt)+(((STP-((SPD+Msm)/(((STP/pt)/((STP+((pos(((SPD-(((STP/pt)/SPD)*((pt+SPD)/pt)))+((sqr(AR)+(Nr+STP))+STP)))/pt)/Nr))+((pt+(Nr/SPD))/((AR+pt)/Nr)))+((Nr*STP)*(Nr*Nr))))+sqr((Msm+pt)))))/(Msm+pt))-(((pt/Nr)-STP)/(STP/Nr)))))

B3. Okruženje paralelnih nesrodnih strojeva

Problem: $R\left|r_j\right|\sum w_j F_j$
Funkcija prioriteta:
π = (((pmin-pt)+(PAT+pmin))-(((((MR-PAT)-((pt/(pmin/(((pmin-pt)+((pmin/((MR-PAT)/(((age/PAT)/pavg)+((pt-pavg)+(PAT*age)))))/(PAT+PAT)))-pt)))/pavg))-PAT)-(pmin/(PAT/pavg)))+((((MR/PAT)/((pmin/((MR-PAT)/((PAT+PAT)+(PAT*age))))+pavg))/pavg)*((PAT*((((MR-PAT)/((pmin/(pavg+PAT))+(PAT*age)))*pavg)/(pmin/((MR-PAT)/(((age/PAT)/(pavg/pmin))+(pmin/((MR-(PAT/pavg))*PAT)))))))*(PAT+(((((pt+pmin)-(pmin-pt))/((((age/PAT)/(pavg/pmin))/(((MR+pt)+((PAT*age)/(pavg/pmin)))-((PAT+PAT)*pt)))/pavg))/(MR-PAT))/(PAT-pavg)))))))

Problem: $R\left|r_j\right|C_{\max}$
Funkcija prioriteta:
π = ((((pavg/((pmin+(pt/((pt+pos((pt+pos((pt+PAT)))))+(((pos((pt+age))+MR)-((pos((pavg+MR))*(pmin-age))-pos((((PAT-age)/(PAT+(pmin/pt)))-pmin))))-(pmin-pt)))))-(((pavg/pt)+MR)-PAT)))-pavg)+((((pmin+(pt/((pt/(0-pt))+((pt-pavg)-(pavg*(pavg*pavg))))))-(pt-(pavg/(((pt/(((pavg*pavg)*((((pavg*pavg)*(pt+age))/(PAT+pmin))+MR))/(((pos((pt+age))-p

t)-pt)/((PAT-age)/(PAT+(pavg*pavg))))))-(((0-(pos((pos((pt+age))+MR))/((PAT+pmin)-pt)
))*((age*((pmin/MR)-age))-pos((pos(pos((PAT+pmin)))/MR)))))-(pt-PAT)))))-pavg)+(((((
0-((pos((PAT+pmin))*(pos((pavg+MR))-age))-(pt+age)))/(((pavg*pavg)-(pavg*age))-(((a
ge*pmin)-(pos((PAT+pmin))-pmin))+(PAT/((pavg*(pavg*age))-((pavg*pavg)-pt))))))-(((
pt/(((PAT-pt)*(PAT+MR))/pos((PAT+pmin))))-(pos((pavg+MR))-pos((pos((PAT-pt))-pm
in))))*(pmin-pt)))-(((pavg*pavg)*(pos((((pavg*pavg)*(pos((pt+age))+MR))/pos((PAT+p
min))))+MR))/pos((PAT+pmin))))-(pavg*(((pavg*pavg)-((pos((pos(pos((PAT+pmin)))*p
min))+MR)/PAT))-(((age*pmin)-pt)+(((pt+PAT)/((pavg*age)-((pt-pavg)-(pmin-pt))))-(pm
in-pavg)))))))+(pavg/(((pt/(((pavg*pavg)*(pos((pos(pos(pos((PAT+pmin))))/MR))+MR))
-pt))-((0-pos((pt+age)))*(0-pos(((pmin-pt)/MR)))))-((((((PAT+pmin)-age)-(pt+age))-((pav
g*age)/MR))*(((pmin/MR)-age)-pos((pos(pos((PAT+pmin)))/MR))))-PAT))))

Problem: $R|r_j|\sum w_j T_j$

Funkcija prioriteta:

$\pi =$ (((pt+pos(((MR+(pt+MR))+((pmin/SL)+w))))+MR)+pos((((((dd*((dd+pt)+MR))+MR
)+(((((pmin/SL)+(pmin/SL))+(((((PAT+(dd+1))+pos((pt+w)))+MR)+(((dd+pt)+(MR+pt))
+MR))+((pt+PAT)+(MR+(MR+(pt+w))))))*((pt/w)+pmin))*(((pt+pos(MR))+SL)+MR)))
((((dd((PAT+(dd+(dd/SL)))+pos(((MR+(pt+MR))+((pmin/SL)+w)))))/pavg)+((((pt+w)
+(pmin/SL))-SL)+(pt+(w/pos((((dd+pmin)+MR)+MR))))))/dd))*((((pt+MR)+(pmin/SL))+
MR)+((pmin+(MR/pos(((((w+PAT)+(((PAT+(dd+1))+pos((pt+w)))+(PAT/SL)))+(MR+pt
))+(pmin+(((dd*((dd+pt)+MR))/pavg)+(pt+pmin)))))))+pos((((dd*((dd+pt)+MR))/pavg)+(
pt+pmin))))))))

B4. Okruženje proizvoljne obrade

Problem: $J|\ |\sum w_j T_j$

Funkcija prioriteta:

$\pi =$ ((w+(((pt+AR)/w)*((TWKR/(((1-sqr(((pos(((TWK/w)+((CLK+((TWKR/w)+(dd+NO
Pr)))/TWK)))+TWKR)-((((dd+PTav)+(CLK*AR))+pos((TWK/w)))/pt))))+(TWKR/AR))+
((((CLK/w)+(dd+pt))+(TWKR/w))-((w+TWKR)/(PTav-w)))))+(((pt/CLK)+((CLK*AR)+(
(w/TWKR)/(PTav-w))))+(((pt+AR)/w)*(((pos((pos(pos((TWK/w)))+TWKR))+(TWKR/w
))+(TWKR/w))+(((TWK*AR)+(dd+pt))-(pos(pos(pos(((TWK*AR)+TWKR))))/TWKR)))
)))))+((sqr((dd-pos(((w+(TWKR+(CLK+(PTav-w))))-(pt/CLK)))))/w)*(((TWK*AR)+(pos
(pos((pt+TWKR)))/w))+((NOPr+((pos((TWK/w))+(((((dd+((PTav/w)+pt))+(pos(pos(((T
WK*AR)+TWKR)))/TWKR))+TWKR)*pos(pos((((dd+PTav)+(CLK*AR))+pos((TWK/w
))))))+(((((((TWK*AR)+TWKR)+(TWK-PTav))-(w-pt))/w)+TWKR)+((AR+TWK)/pt))))-
(TWKR/(TWK*w))))+w))))

Problem: $J|\ |\sum w_j T_j$

Funkcije za svako od 3 stabla rješenja genetskog programiranja:

$\pi_1 =$ (MNOPw-MTWKav), $\pi_3 =$ (TWK*NOPr)

$\pi_2 =$ ((dd-CLK)+((((AR+pt)+pos((((AR+PTav)*((dd/w)+((AR+PTav)*((((((TWK-HTR)+
TWK)*(((AR+pt)/w)/w))+(dd-CLK))+pos((AR+NOPr)))+((AR+PTav)*((AR+PTav)+(T
WK/w)))))))-((AR+PTav)*((AR+(dd-CLK))+(AR*((((AR+dd)+((dd+TWK)*sqr((dd-CLK
))))+((PTav+(AR+pt))*(TWK/w)))+sqr((dd-CLK)))))))))+(((((AR+PTav)*((dd+TWK)+T
WKR))+((AR*((pos((AR+pt))+(TWK/w))/w))+(((pos((((AR+pt)*((AR+(dd-CLK))+AR))
+NOPr))+(AR+pt))/NOPr)*((AR+PTav)+pos((dd-TWKR))))))*((AR+PTav)/w))-CLK))+
dd))

Sažetak

U radu se promatra problem definiranja prikladnih postupaka raspoređivanja za različita okruženja s obzirom na uvjete raspoređivanja i zadane kriterije. Predlaže se metodologija izvođenja algoritama raspoređivanja uz pomoć genetskog programiranja. Algoritmi raspoređivanja poprimaju oblik pravila u kojima se elementima u sustavu dodjeljuje prioritet na temelju kojega se aktivnosti pridružuju sredstvima.

Koristeći genetsko programiranje, izvode se pravila za različita okruženja raspoređivanja: raspoređivanje na jednom stroju, na paralelnim jednolikim strojevima, nesrodnim strojevima te u okruženju proizvoljne obrade. Za pojedino okruženje postupak raspoređivanja definiran je u dva dijela: jedan dio predstavlja meta-algoritam koji koristi prioritete elemenata u sustavu kako bi pridruživao aktivnosti sredstvima, a drugi dio predstavlja funkciju koja određuje prioritete elemenata. Prioritetna funkcija dobiva se primjenom genetskog programiranja.

Za svako okruženje definirani su skupovi ispitnih primjera za učenje i ocjenu, a predloženi algoritmi uspoređeni su sa postojećim algoritmima raspoređivanja. Algoritmi raspoređivanja izvedeni uz pomoć genetskog programiranja pokazuju sličnu ili bolju učinkovitost u usporedbi s postojećim algoritmima, a značajnu prednost ostvaruju u okolinama raspoređivanja za koje ne postoje prikladni postupci raspoređivanja.

U radu je također opisan postupak vrednovanja podatkovnih elemenata rješenja genetskog programiranja te postupak prilagodbe primjene genetskih operatora križanja i mutacije. Predloženi postupci prilagodbe olakšavaju pronalaženje kvalitetnog rješenja i povećavaju uspješnost evolucijskog procesa.

Ključne riječi: raspoređivanje, genetsko programiranje, prioritetno raspoređivanje, postupci prilagodbe genetskih operatora

Abstract

Title: Scheduling based on adaptive rules

In this work the problem of devising an appropriate scheduling policy for different environments is addressed. The methodology which uses genetic programming to evolve scheduling heuristics is described. The scheduling heuristics are developed in the form of scheduling rules which define dynamic priorities for the elements in the system.

Sheduling rules for different environments are devised using genetic programming: one machine, parallel proportional machines, unrelated machines and job shop environment. Scheduling algorithms are defined with two components: one component represents meta-algorithm which operates in scheduling environment, and the other represents an appropriate scheduling policy which derives job or machine priorities. The scheduling policy is evolved with genetic programming.

For each scheduling environment a set of learning and a set of evaluation scheduling instances is defined. Devised algorithms are compared with existing algorithms in each environment. The evolved algorithms exhibit similar or better efficiency in all cases, and a significant improvement is achieved in scheduling environments where there are no fitting algorithms.

Additionally, a method for evaluation of terminals in genetic programming solution and adaptive probabilities for genetic operators crossover and mutation are devised. The adaptive methods increase the probability of finding a good solution and may speed up the evolution process.

Keywords: scheduling heuristics, priority scheduling, genetic programming, adaptive genetic operators

Životopis

Domagoj Jakobović je rođen 11. srpnja 1973. u Našicama. Tijekom srednjoškolskog obrazovanja nekoliko je puta sudjelovao na natjecanjima iz informatike na republičkom nivou. Maturirao je 1992. godine kao jedan od učenika oslobođenih mature s odličnim uspjehom po programu matematičke gimnazije. Po završetku srednje škole upisuje studij na Fakultetu elektrotehnike i računarstva (FER) u Zagrebu. Za vrijeme studija obavljao je stručnu praksu u inozemstvu (Tunis, Slovačka) posredstvom organizacije IAESTE. Diplomirao je u redovnom roku u prosincu 1996. godine, po programu sa naglaskom na znanstveno-istraživački rad, sa temom optimizacije genetskih algoritama. Dobitnik je priznanja "Josip Lončar" za uspjeh na 4. godini studija.

U travnju 1997. godine započeo je s radom kao znanstveni novak na FER-u, Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave. Godine 2001. obranio je magistarski rad s temom "Ocjena učinkovitosti postupaka za rješavanje kinematike Stewartovih paralelnih mehanizama". U međuvremenu je objavio četrnaest znanstvenih radova u časopisima i konferencijama. Aktivno sudjeluje u održavanju nastave u nekoliko kolegija.

Biography

Domagoj Jakobović was born on 11. 07. 1973. in Našice, Croatia. After high school he enrolled Faculty of Electrical Engineering and Computing (FEEC) in Zagreb. During undergraduate studies he was an exchange student in several countries abroad. He received B. Sc. degreee in 1996. with scientific research based thesis topic and a "Josip Lončar" award for best students in generation.

In 1997. he started as a research assistant at FEEC and received M. Sc. degree with thesis "The Evaluation of Problem Solving Methods for Stewart Parallel Mechanism Kinematics" in 2001. In the meantime he published fourteen journal and conference articles and was involved in teaching of several undergraduate and graduate courses.