Noname manuscript No.
(will be inserted by the editor)

Automatic design of dispatching rules for static scheduling conditions

Marko Đurasević · Domagoj Jakobović

Received: date / Accepted: date

Abstract Dispatching rules (DRs) represent heuristic methods designed for solving various scheduling problems. Since it is hard to manually design new DRs, genetic programming is used to design them automatically. Most DRs are designed in a way that they can be applied under dynamic conditions. On the other hand, static problems are usually solved using various metaheuristic methods. However, situations exist in which metaheuristics might not be the best choice for static problems. Such situations can occur when the schedule needs to be constructed quickly so that the system starts executing as soon as possible, or when it is feasible that certain changes happen during the execution of the system. For these cases DRs are more suitable, since they execute faster and can adapt to possible changes in the system. However, as most research is focused on developing DRs for dynamic conditions, they would perform poorly under static conditions, since they would not use all the information that is available. Therefore, there is the need to enable automatic development of DRs suitable for static and off-line conditions. The objective of this paper is to analyse several methods by which automatically generated DRs can be adapted for static and off-line scheduling conditions. In addition to look-ahead and iterative DRs which were studied previously, this paper proposes new terminal nodes, as well as the application of the rollout algorithm to adapt DRs for static conditions. The performance and execution time of all methods is compared with the results achieved by automatically generated DRs for dynamic conditions and genetic algorithms. The tested methods obtain a wide range of results, and prove to be competitive both in their performance and execution speed with other approaches. As such, they are a viable alternative to metaheuristics since they can be used in situations where metaheuristics could not, but can offer either a better execution time or even competitive results.

Marko Đurasević
Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, 10000, Croatia
E-mail: marko.duraevic@fer.hr

Domagoj Jakobović
Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, 10000, Croatia

Keywords Genetic programming · dispatching rules · look-ahead · rollout heuristic · iterative dispatching rules · unrelated machines environment · static conditions

1 Introduction

Scheduling represents an important decision making process which has found its application in various areas, like scheduling planes on runways [10,20], scheduling in manufacturing and assembly lines [11], scheduling in wafer fabrication [55,60] and production plants [31], scheduling resources in clouds [61], staff scheduling [14], multiprocessor scheduling [24], or scheduling for radiotherapy pre-treatment [52]. The goal of scheduling is to perform a mapping between available activities and a set of resources, to obtain a schedule which optimises certain user defined criteria [58]. Since most scheduling problem instances are NP-hard, various heuristic algorithms are most often used to find solutions to scheduling problems. These heuristic algorithms are divided into two groups, *improvement heuristics* and *constructive heuristics*.

Improvement heuristics start with a complete schedule and improve its quality by performing different modifications. Various evolutionary and population based metaheuristics belong to this category, like genetic algorithms, particle swarm optimisation, etc. Metaheuristic methods have become immensely popular for solving various kinds of real world optimisation problems like river flow forecast [68], adaptation of parameters in biodiesel production [44], big flood management [16], discharge prediction in hydropower [7], hydrological time series forecasting [66], evaporation prediction [42], application in cryptography [56,57], and many others [19]. These metaheuristic algorithms can easily be adapted for solving different kinds of scheduling problems, and have therefore been extensively used for solving various scheduling problems [8,9,69,26,21,17,37,34]. Since these heuristics start with an existing schedule, it is necessary to have all the relevant information about the scheduling problem available prior to the execution of the system, meaning it is presumed that scheduling is performed under static or off-line conditions. Additionally, because of the relatively long execution time of these heuristics, they need to be executed before the start of the system.

Constructive heuristics start with an empty schedule, which is incrementally constructed. They mostly appear in the form of dispatching rules (DRs) [38,6,67]. When creating schedules, DRs start with an empty schedule and each time a scheduling decision needs to be performed, they use the currently available information from the system to determine which job should be scheduled on which machine at the current moment in time. Thus, DRs can quickly react to the changing conditions in the environment and be applied under dynamic conditions. However, manually designing DRs for different conditions and criteria is time consuming.

To deal with the problem of manually designing new DRs, recent studies propose the application of different evolutionary computation methods to create new DRs for various scheduling problems. The most widely used method for the creation of new DRs is genetic programming (GP) [32,59], which achieved good results for many problems [33]. By using GP new DRs can be created for a wide variety of different scheduling conditions and criteria [41,18,27,50,28]. Automatically generated DRs achieve a better performance than most manually designed DRs. In recent years a lot of research has been performed in different areas of automatic design of DRs, such as designing DRs for optimising multi-objective problems [47,49,39,29,63], creating ensembles of DRs [51,22,62], comparing different methods for designing DRs [46,4,65], etc. Aside from GP, other machine learning methods [25] or data mining techniques [36] were used to design DRs. In the case of the unrelated machines environment, recent research focused on parameter optimisation and testing various methods for the evolution of DRs [65], optimising multiple objectives simultaneously with different multi and many ovjective methods [63], and using ensemble learning algorithms to improve the performance of DRs [62,64]. Unfortunately most of the previously mentioned works were focused almost exclusively on dynamic conditions, and no customisations were done to adapt DRs to static conditions (except for iterative DRs in [65]). Therefore, the adaptation of DRs to static conditions is still an open issues in the unrelated machines environment, but also in other environments as well. Two recent surveys give an overview of previously conducted research in the field od automatic design of DRs [5,45].

A lot of recent research is focused on dynamic scheduling problems, where information about the problem becomes available during the execution of the system. However, static problems in which the entire schedule has to be constructed before the execution of the system, or off-line problems in which the information is available prior to the execution of the system but the schedule can be constructed during its execution are still widely researched. This is due to the fact that for certain real world problems, like aeroplane scheduling, staff scheduling, or therapy treatment scheduling all the information is available up front and the schedules can be constructed beforehand [14,52]. Although in those cases standard metaheuristics are commonly used to deal with such problems, there are occasions in which they cannot not be applied. For example, in off-line dynamic scheduling, although the data is available beforehand, the schedule still needs to be executed in parallel with the execution of the system [53]. An example of this situation can be in systems where all the information about the scheduling problem becomes available at the start of the system, and therefore the schedule needs to be created as soon as possible in order to start the execution of jobs [1]. In this situations metaheuristic improvement algorithms are not applicable, since they need to create the entire schedule before it can be executed on the system. Thus the system would be required to wait before the metaheuristic method executes and then execute its schedule. Furthermore, even if static scheduling is considered, it is still possible that certain changes appear during the execution of the system (new jobs that are released into the system, changes in job processing times, late arrivals of certain jobs, machine breakdowns and similar) [43]. Those cases are also difficult for metaheuristic methods, since they construct the schedule up front, and thus if any changes happen in the system, the schedule constructed by the improvement metaheuristics is invalid. Thus, the metaheuristics methods would need to be executed from the start, or some correction methods would need to be executed on the existing schedule to amend the problems which would arise from changes that happen in the system.

As a consequence metaheuristics might not always be the method of choice in situations when all the information about the scheduling problem is known beforehand. In some cases DRs would still be a better choice, since they can construct schedules in a much lower time and also due to the fact that they do not construct the entire schedule immediately, but rather only select the next job to be executed. By default these methods solve all the previous problems that were denoted for improvement metaheuristic methods. However, the disadvantage of DRs is that by default they do not use any static information, and thus construct the schedule only with a limited view on the problem. As such, they cannot compete with any improvement methods, and achieve poor results. As a result, efforts were made to adapt DRs for static scheduling conditions, so that they can use the additional information for creating better schedules [67]. However, the problems associated with DRs still remain, since it is difficult to manually design new DRs for static conditions as well. Although, several studies have also dealt with the problem of automatically evolving DRs that are adapted for solving static scheduling problem, the research in this area is still sparse. Hildebrannt et al. [23] have used GP to generate DRs which use look-ahead to consider jobs that have not yet been released into the system. Although the method achieved the best results it demonstrated a quite large sensitivity and dispersal of the obtained results. Nguyen et al. [48] proposed a GP procedure for the generation of iterative DRs (IDRs), which create the schedule several times, extracting certain information from previously generated schedules to create better schedules in subsequent iterations. Đurasević et al. [65] applied IDRs in the unrelated machines environment, and demonstrated that the method achieves better results than DRs designed for dynamic conditions.

Although the previous studies apply methods for adapting DRs to static scheduling conditions, many questions still remain open. For example, the tested methods are not compared to each other or to any improvement metaheuristic to demonstrate their speed and performance. Therefore, it might not be clear what the benefits of these methods compared to improvement heuristics really are, and whether the results obtained by such DRs can even compare to those of improvement metaheuristics. Furthermore, there are also alternative ways to adapt DRs for static conditions which have not been considered in previous studies, like the application of specialised static terminal nodes or the rollout algorithm. As a consequence, there is not enough information available to determine advantages and disadvantages of such methods and if the results that can be obtained by each the methods are satisfactory.

The focus of this paper is to analyse the existing methods that can be used for adapting automatically designed DRs for scheduling under static conditions. To achieve this, four methods are selected and adjusted for adapting automatically designed DRs for static conditions. Out of these the rollout algorithm is for the first time proposed to solve scheduling problems with automatically designed DRs, whereas the other three methods were adapted to the unrelated machines environment. Furthermore, all the methods have been combined with each other, to analyse whether it is possible to obtain improved results with their combinations. The selected methods are tested on a set of scheduling problem instances and the quality of the obtained results as well as the time requited to construct the schedule are compared. Furthermore, the results of the tested methods are also compared to a manually designed DR for static problems, the Apparent Tardiness Cost (ATC) rule, and to a genetic algorithm. In that way it is possible to rank all methods with regards to their solution quality and execution time, which allows the user to select the appropriate method which offers the required trade off between the execution speed and solution quality. Finally, the solution construction process of all four methods is compared to obtain a deeper insight in the way that each method performs the decisions and denote situations which can be problematic for certain methods. Therefore, the key objectives of this study can be summarized as:

1. Four methods for adaptation of automatically designed DRs for solving static and off-line scheduling problems, out of which the rollout algorithm is adapted for the first time to solve scheduling problems in unison with automatically designed DRs
2. Comparison of the four tested methods and their combinations with each other, a manually designed static DR and an improvement metaheuristic
3. Analysis of the advantages and disadvantages of each method, as well as the decision process performed by each of them.

The rest of this paper is organised as follows. Section 2 gives an overview of the unrelated machines environment which will be used for performing the experiments, and the GP procedure that will be used to automatically generate new DRs. Section 3 provides a description of the different methods which will be used to adapt DRs for scheduling under static conditions. The benchmark setup is described in Section 4. In Section 5 the results obtained by the tested methods will be presented. A short analysis of the different methods will be presented in Section 6, while Section 7 concludes this study and gives a short outlook on future research directions.

2 Background

2.1 Scheduling in the unrelated machines environment

The unrelated machines environment consists of n jobs which need to be scheduled on any of the m available machines. Each machine has a different processing speed for each job, meaning that the processing speeds can vary freely across all machines. This paper focuses on solving the scheduling problem defined as $Rm|r_j|Twt$ using the $\alpha|\beta|\gamma$ notation of scheduling problems [58], where the α field represents the machine environment, the β field denotes additional constraints placed upon the scheduling problem, and the γ field denotes the criteria which are optimised. In the considered problem r_j denotes that for each job a release time is defined, which determines the moment when the job enters the system. The Twt criterion represents the total weighted tardiness of the schedule, which is defined as $Twt = \sum_j w_j(\max\{C_j - d_j, 0\})$, where w_j represents the weight (importance) of job j, C_j the point in time when job j finished with its execution, and d_j the due date which determines the point in time until when job j should finish with its execution. Additionally, for each machine and job a processing time p_{ij} is defined, which determines the amount of time needed for machine i to process job j. The scheduling process will be performed under static conditions, where all the information about jobs and their characteristics is known beforehand.

2.2 Generating DRs with GP

DRs which are evolved by GP usually consist of two parts: the schedule generation scheme (SGS) and the priority function (PF). The PF determines a priority value for a certain job-machine pair, which is calculated based on certain properties of jobs and machines. These priority values are used by the SGS to determine which job should be scheduled on which machine and in what order. Algorithm 1 represents the SGS which is used to generate schedules for the unrelated machines environment under dynamic conditions. The procedure first waits until at least one job and one machine are available. Then the priority values are calculated for scheduling each available job (those which are released, but not yet scheduled) on each of the machines (even those which are executing another job). Based on the calculated priority values, for each job the best machine, the one for which the job achieves the best priority value, is determined. Out of all jobs whose best machine is available, the one with the best priority value is selected and scheduled on the appropriate machine. This part is repeated until there are no more jobs whose best machine is available. If there is no job whose best machine is available, then the scheduling decision is postponed to a later moment in time, when another job or machine becomes available. The entire procedure is repeated until there are no more jobs to be scheduled.

Algorithm 1 Schedule generation scheme used by DRs generated by GP

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

Unlike the SGS, which is defined manually, the PFs are generated automatically by using GP. The objective of GP is to evolve a PF which is appropriate for optimising a certain scheduling criterion, and which can be used by the aforementioned SGS. To do so the primitive set of nodes needs to be defined. The initial terminal set that is used by GP is given in Table 1. The *time* variable, used in some terminal node definitions, represents the current time of the system when the value of the nodes is calculated. The set of function nodes includes the binary addition, subtraction, multiplication, secure division (returns 1 if divisor is close to zero) operators, and the unary operator $POS(a) = \max(a, 0)$. Although other function nodes (like minimum, maximum, absolute values, if-else branches) can be used by GP to evolve the PF, it was shown that no significant improvements can be achieved by using them [65]. The PF is represented in the form of a tree in which the inner nodes represent functions, and leaf nodes represent the system parameters from Table 1. These trees are then modified by using various crossover and mutation operators outlined in the appendix.

Aside from GP, many other soft computing methods can be used to design new DRs, like neural networks. Although neural networks have demonstrated to achieve a similar results as DRs evolved by GP, their disadvantage lies in the fact that DRs evolved by neural networks cannot be interpreted as easily as can symbolic expressions that are evolved by GP [4]. Aside from neural networks, there are also many methods that are similar to GP, and just use different solution representations or introduce additional constraints to those expressions like Gene Expression Programming (GEP) [15], Dimensionally Aware GP (DAGP) [30], Cartesian GP (CGP) [40]. Some of these methods have also been tested for the evolution of DRs, and it was found that they all achieve similar results with GP achieving slightly better results than other methods [65]. Therefore, using methods with alternative representations can have a slight influence on the achieved results, but in general these methods should be capable of obtaining good DRs as well.

Table 1: Terminal nodes used by GP

Node	Description
pt	processing time of job j on the machine i (p_{ij})
pmin	the minimal job processing time on all machines: $\min_i(p_{ij})$
pavg	the average processing time on all machines
PAT	time until the machine with the minimal processing time for the current job is available
MR	the amount of time until the current machine becomes available
age	the time that the job spent in the system: $time - r_j$
dd	due date (d_j)
SL	positive slack: $\max(d_j - p_{ij} - time, 0)$
w	weight of the job

3 DR based methods for static scheduling

3.1 Terminal nodes with static information

An easy way of generating DRs for static conditions is to define additional terminal nodes which provide static information about the system. With these terminals GP can design PFs that use information about the future of the system when making scheduling decisions. The additional static terminals are presented in Table 2. The first seven terminals do not depend on the choice of which job would be scheduled next. This means that at the current decision point these terminals will have the same value for all considered jobs. However, some of the terminals depend on the currently considered machine, which means that they will have different values for them. These terminals represent general information about the future of the system, like the time until the next job arrives or the slack of the next job released into the system.

The remaining terminals represent how the scheduling of one job could affect the future of the system. The first four terminals in this group (*NREL*, *NRELM*, *SLAVGD* and *MLOADD*) extract information about jobs which would be released during the execution of job j. The other twelve terminals approximate how much the scheduling of job j influences the tardiness of jobs which become available during the execution of job j. The first six terminals (from *FUTLATES* until *WLATEL*) approximate the tardiness of jobs which become available during the execution of the currently considered job j. The other six terminals approximate the difference between the tardiness of job j, if it would be delayed and other jobs would be scheduled before it, and the tardiness of other jobs which become available during the execution of job j, if job j would be executed immediately.

Table 2: List of static terminal nodes

Node name	Node description
NSHORT	Number of unreleased jobs which have the shortest processing time for a machine
SLNXT	Slack of the next job that is released into the system
SLNXTM	Slack of the next job that is released into the system, and has the shortest processing time for the given machine
SLAVG	Average slack of jobs with the shortest processing time for the given machine
TTAR	Time until the next job arrives into the system
TTARM	Time until the next job, which has the shortest processing time for the given machine, arrives into the system
MLOAD	Sum of processing times of unreleased jobs, which have the shortest processing time for the given machine
NREL	Number of jobs which become available during the execution of the selected job
NRELM	Number of jobs which become available during the execution of the selected job, and have the shortest processing time for the given machine

SLAVGD	Slack of jobs which become available during the execution of the selected job, and have the shortest processing time for the given machine
MLOADD	Sum of processing times of jobs which become available during the execution of the selected job, and which have the shortest processing time for the given machine
FUTLATES	Approximation of the weighted tardiness of the job which has the fastest execution time for the considered machine, and is released first during the execution of job j. The approximation is performed as if the considered job would be executed immediately after the completion of job j
WLATES	Approximation of the unit penalty of the job which has the fastest execution time for the considered machine, and is released first during the execution of job j. The approximation is performed as if the considered job would be executed immediately after the completion of job j
FUTLATE	Approximation of the weighted tardiness of all jobs which become available during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed right after the completion of job j
WLATE	Approximation of the weighted number of tardy jobs, of all jobs which become available during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed right after the completion of job j
FUTLATEL	Approximation of the weighted tardiness of all jobs which become available during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed sequentially after the completion of job j
WLATEL	Approximation of the weighted number of tardy jobs, of all jobs which become available during the execution of job j, and have the smallest processing time for the given machine. The approximation is performed as if each of the jobs would be executed sequentially after the completion of job j
FLDS	Difference between the approximation of the weighted tardiness of job j and the value of the *FUTLATES* terminal. The weighted tardiness is approximated as if job j is executed after the job that has the fastest execution time for the considered machine, and is released first during the execution of job j, finishes with its execution
WLDS	Difference between the approximation of the tardiness weight of job j and the value of the *WLATES* terminal. The tardiness weight is approximated as if job j is executed after the job that has the fastest execution time for the considered machine, and is released first during the execution of job j, finishes with its execution

FLD	Difference between the approximation of the weighted tardiness of job j and the value of the *FUTLATE* terminal. The approximation is performed as if job j is executed after all the jobs with the fastest execution time for the considered machine, which become available during the execution of job j, finish with their execution
WLD	Difference between the approximation of the tardiness weight of job j and the value of the *WLATE* terminal. The approximation is performed as if job j is executed after all jobs with the fastest execution time for the considered machine, which become available during the execution of job j, finish with their execution
FLDL	Difference between the approximation of the weighted tardiness of job j and the value of the *FUTLATEL* terminal. The approximation is performed as if job j is executed after all jobs with the fastest execution time for the considered machine, which become available during the execution of job j, finish with their execution
WLDL	Difference between the approximation of the tardiness weight of job j and the value of the *WLATEL* terminal. The approximation is performed as if job j is executed after all jobs with the fastest execution time for the considered machine, which become available during the execution of job j, finish with their execution

The terminals *FUTLATES*, *FUTLATEL*, and *FUTLATE* approximate the tardiness of jobs that would first be released into the system during the execution of the currently considered job j. The *FUTLATES* terminal approximates the tardiness of the next job which would be released into the system during the execution of job j and executes the fastest on machine m on which job j is executing. This is done in a way that the tardiness of this job is calculated as if it would be scheduled on machine m after job j finished with its execution. The *FUTLATE* terminal considers all jobs which would be released during the execution of job j, and which have the fastest processing time on machine m. The tardiness of all these jobs is approximated as if each job would be executed on machine m after completion of job j, independently of each other. Since this is a quite optimistic approximation, the *FUTLATEL* represents a more pessimistic approximation. This terminal calculates the tardiness as if all the jobs were scheduled sequentially on machine m, in order of their release times, after job j finishes with its execution. The terminals *WLATES*, *WLATE* and *WLATEL* perform the approximation of the weighted number of tardy jobs in the same manner as *FUTLATES*, *FUTLATE*, and *FUTLATEL*, respectively.

The previously described terminals approximate only the tardiness of jobs that would be released during the execution of job j, but do not take into account the tardiness of job j if its execution would be delayed. Thus, six additional terminals are defined. The *FLDS* terminal approximates the tardiness of job j in a way that it delays its execution until the job which would be first released into the system during the execution of job j, and has the smallest processing time on machine m, finishes with its execution. The value of the *FUTLATE* terminal is additionally subtracted to determine which decision would lead to a greater tardiness. The *WLDS* terminal uses the same concept, just for approximating the weighted number of tardy jobs, and subtracting its value with the value of the *WLATES* terminal. For the *FLD* and *FLDL* terminals, the approximation of the tardiness of job j is performed somewhat differently. For those two terminals, the tardiness of job j is approximated as if job j would be executed after all jobs which were released during its execution and have the smallest processing time on machine m, would be executed sequentially on the considered machine m. For the *FLD* terminal, the approximation is additionally decreased by the the *FUTLATE* terminal, while for the $FLDL$ terminal the approximated tardiness value is subtracted by the *FUTLATEL* terminal. The last two terminals, *WLD* and *WLDL*, also use the same concepts as *FLD* and *FLDL* to approximate the number of tardy jobs, and subtract the approximation by the *WLATE* and *WLATEL* terminals, respectively.

3.2 Look-ahead

In the dynamic scheduling environment, DRs use the PF to calculate the priorities only of those jobs which were already released into the system. Since in the static scheduling environment the information about all jobs is known in advance, DRs can be extended so that they calculate the priorities for jobs which are not yet released into the system. This property is called *look-ahead*. When using look-ahead the SGS is similar to the one in Algorithm 1, with the only difference being that in line 3 the algorithm will iterate through all jobs for which the following condition is true: $r_j < (time + (\max_j(r_j) - time) * \alpha)$. The look-ahead factor α determines the amount of unreleased jobs that will be considered when calculating the priority values, and which will be denoted as the *look-ahead horizon*. The look-ahead parameter is sensitive to the size of the problem instance, meaning that more unreleased jobs will be considered in each iteration if there are more jobs in the scheduling problem. Instead of using the look-ahead factor, a fixed number of unreleased jobs, which should be considered in each iteration, can be used. This has the advantage that regardless of the problem size the same number of jobs will always be considered. Furthermore, jobs which are not yet released will not be scheduled until they are actually released into the system. Thus, there is a possibility that in meantime another job is scheduled on the machine for which previously an unreleased job achieved the largest priority. When using look-ahead, a new terminal node *AR* is introduced, which determines the amount of time until the job is released into the system. Without this terminal DRs could not take into account when the jobs become ready, and would be unable to prioritise jobs which are released sooner. Look-ahead allows the SGS to introduce idle times into the schedule, if it determines that in the near future a job of high priority will be released.

3.3 Iterative dispatching rules

Iterative dispatching rules (IDRs) construct the schedule several times, and each time a new schedule is constructed they use information from previously generated schedules to improve the newly constructed schedules [48]. The schedule is reconstructed until the fitness of the newly constructed schedule stops improving. The motivation behind this approach is that by using information from previously created schedules, IDRs could correct mistakes made in previous iterations. The steps of the SGS used by IDRs are shown in Algorithm 2. The procedure does not return the schedule which was created in the last iteration, but the previous one, since that schedule achieved the best result. For better understandability, the algorithm is also presented in Figure 1 as a flowchart. Since this method recreates the schedule several times, it is only applicable under static conditions, and cannot be used for dynamic off-line scheduling, unlike the other described methods.

Algorithm 2 Schedule generation scheme used by IDRs

```
 1: Let $R$ represent the set of parameters extracted from the previous schedule which
    are used by the PF, and let $R_0$ represent their initial values
 2: $R \leftarrow R_0$
 3: $Fitness^* \leftarrow \infty$
 4: Let $S$ represent the current schedule (empty at the beginning), and $bestS$ the best
    created schedule
 5: do
 6:     $bestS \leftarrow S$
 7:     Generate the schedule using the standard schedule generation scheme and the
        PF $\pi$
 8:     $S \leftarrow$ generated schedule
 9:     $Fitness^* \leftarrow Fitness$
10:     $Fitness \leftarrow$ fitness value of the generated schedule $S$
11:     Calculate new values for schedule dependant nodes, based on the constructed
        schedule $S$, and store the calculated values in $R$
12: while ($Fitness^* > Fitness$)
13: Return $bestS$ as the result
```

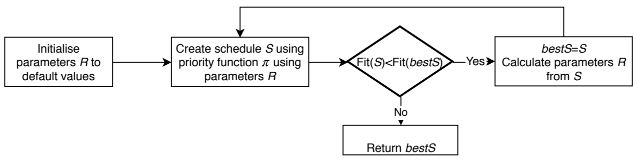

Initialise parameters R to default values
Create schedule S using priority function π using parameters R
Fit(S)<Fit(bestS)
Yes
bestS=S Calculate parameters R from S
No
Return bestS

Fig. 1: Flowchart of creating schedules with IDRs

In order for the PF to use information about previously built schedules, additional nodes are defined, the values of which are calculated based on the schedule created previously by the IDR, and are updated every time a new schedule is created. Table 3 represents additional nodes which use information from previously created schedules, four of which (*NLATE*, *LATENESS*, *INDLATE*, *ISLATE*) are taken from a previ-

Table 3: Additional nodes used by IDRs

Node name	Node description
NLATE	number of tardy jobs in the previous schedule
LATENESS	total lateness of the entire previously built schedule
INDLATE	lateness of a concrete job in the previous schedule
TARDINESS	total weighted tardiness of the entire previously built schedule
INDTARD	tardiness of a concrete job in the previous schedule
INDWTARD	weighted tardiness of a concrete job in the previous schedule
ISLATE	if the job was late in the previous schedule executes the left branch, otherwise the right branch
JOBFINISH	completion time of a concrete job in the previous schedule
FLOWTIME	flowtime of a concrete job in the previous schedule

ous study [65]. All nodes represent terminals, apart from the *ISLATE* node, which is a function that executes one branch if the currently considered job was late in the previous schedule, and the other branch if not. The aim of this node is to create PFs where one part of the function is appropriate for scheduling jobs which were late in the previous schedule, making it possible to apply a different scheduling strategy for those jobs. The *NLATE*, *LATENESS*, and *TARDINESS* nodes provide information about the entire previously built schedule, in the form of the total number of tardy jobs, total lateness of the schedule, and total weighted tardiness of the schedule. Nodes *INDLATE*, *INDWTARD*, and *INDTARD* provide information about the lateness, weighted tardiness, and tardiness of a job in the previous schedule. By using these nodes the PF can put more emphasis on jobs which were tardy in the previous schedule. Finally, nodes *JOBFINISH* and *FLOWTIME* provide information about the completion time and flowtime of jobs in the previous schedule. Although these nodes do not provide any due date related information, they are included to analyse whether they provide useful information for IDRs. An additional thing which needs to be defined for these nodes are the initial values used in the first iteration, when there is no previous schedule from which the information could be extracted. In that case, all nodes are initialised to large values which cannot be achieved by any schedule constructed by the SGS, while for the $ISLATE$ node all jobs are denoted as late.

3.4 Rollout algorithm

The rollout algorithm is an approach that can improve the results of different heuristic methods [3,2]. The algorithm balances between exhaustive and heuristic search to perform better than heuristic methods, but to obtain solutions faster than exhaustive search. The algorithm considers all possible decisions at each decision moment. However, to determine the best decision, it does not perform an exhaustive search, but uses a heuristic method to construct the rest of the solution for each possible decision. The algorithm performs the decision which leads to the best solution after applying the heuristic. In that way the algorithm has a better overview on the problem in comparison with the heuristic methods on their own. These steps are repeated for each decision moment until the solution is constructed.

The rollout algorithm can be combined with DRs so that at each decision moment and for each possible decision at that point the rollout algorithm uses a DR to construct the rest of the schedule. After constructing a schedule for each decision, the algorithm performs the decision for which the DR obtained the best schedule. Algorithm 3 denotes the steps of the rollout algorithm for solving scheduling problems with DRs. In the first part the algorithm tries out all possible scheduling decisions at the current moment in time, and uses a predefined DR to construct the rest of the schedule from that decision onwards. Applying the rollout algorithm in this way will lead to bad solutions, since it creates schedules in which jobs are scheduled immediately on a machine if it is free. On the other hand, DRs can introduce idle times on machines even if there are available jobs. Therefore, DRs can construct a schedule approximation that cannot be obtained by the rollout algorithm, since it does not introduce idle times in the schedule. Instead of improving the fitness of solutions during the execution of the rollout algorithm, the fitness will oscillate and will sometimes be worse than the one obtained by the DR. To solve this problem, if the fitness of the best decision in the current iteration would be worse than that of the best decision in the previous iteration, instead of performing the selected decision the rollout algorithm will use the DR to determine which job should be scheduled next, and at what time. Thus, if the rollout algorithm will in itself not be able to perform the best decision, it will delegate this task to the underlying DR, which can then introduce idle times into the schedule.

The execution time of the rollout algorithm can be improved if not all jobs are considered at each decision moment. For example, jobs that are released far in the future will have a small probability of being scheduled at the current decision moment. Thus, it makes sense to consider a smaller number of unreleased jobs that have a closer release time. For that purpose either a rollout factor γ or the number of unreleased jobs which will be considered, can be defined. The set of jobs that is considered in each iteration will be denoted as the *rollout horizon*.

3.5 Combination of static methods

The benefit of the previous four methods is that they can be combined in various ways. All the methods can easily be combined with each other except for two cases. The first case is when combining static terminals with look-ahead. The problem here is that static terminals are calculated based on all unreleased jobs, however, in look-ahead the priorities are calculated even for some unreleased jobs. This would mean that the properties of unreleased jobs which are considered by the DRs would be used in the calculation of static terminal nodes. Therefore, it could be better to calculate static terminals based only on jobs which are currently outside the look-ahead horizon. Both static terminal calculation methods will be tested to determine their influence on the quality of the look-ahead method.

The second problematic case is when combining IDRs with the rollout algorithm. The problem arises from the fact that IDRs need to reconstruct the entire schedule,

Algorithm 3 Rollout algorithm for scheduling with DRs

```
1: $time \leftarrow 0$
2: $previousFitness \leftarrow \infty$
3: $bestFitness \leftarrow \infty$
4: while unscheduled jobs are available do
5:     Set $time$ to the next point in time where there is at least one released job and
       one available machine
6:     for each unscheduled job $j$ where $r_j < (time + (\max_j(r_j) - time) * \gamma)$ do
7:         for each machine $m$ do
8:             Use a DR to construct the rest of the schedule when job $j$ would be
               scheduled on machine $m$.
9:             Let $fitness$ denote the fitness of the constructed schedule.
10:            if $fitness < bestFitness$ then
11:                $bestFitness \leftarrow fitness$
12:                Let $bestPair$ denote the selected job-machine pair
13:            end if
14:        end for
15:    end for
16:    if $previousFitness > bestFitness$ then
17:        $previousFitness \leftarrow bestFitness$
18:        Schedule the job from $bestPair$ on the machine from $bestPair$
19:    else
20:        Execute the DR to perform the next scheduling decision
21:    end if
22: end while
```

meaning that the schedule constructed by the rollout algorithm would be lost and a schedule of inferior quality would probably be constructed. This could be fixed by not recreating the entire schedule with IDRs, but only those parts which were not constructed by the rollout algorithm. However, because IDRs use information from previously generated schedules, the performance of IDRs is highly dependent on them. This leads to a great instability of the approach, since in each iteration the approximation of IDRs would be very different. Thus, the entire rollout algorithm would perform poorly since the IDRs would not properly guide the algorithm. As a consequence, the combination of the rollout algorithm and IDRs will not be considered.

4 Benchmark setup

To test the performance of the afore described methods, a set of 120 problem instances was generated similar as in many other studies dealing with this problem [13, 12, 35, 54]. These problem instances were divided into two independent problem sets, the *training set* and *test set*, each consisting out of 60 problem instances. The training set was used by GP to design a new DR, while the test set was used to evaluate the performance of the evolved DRs. In each iteration the same training set was used and the rules were evolved for the entire set. Both problem sets contained problem instances of various sizes, consisting of 12, 25, 50, and 100 jobs, and 3, 6, and 10 machines. Details about problem instances generation are given in the appendix and can be downloaded from: `http://gp.zemris.fer.hr/scheduling/probleminstances.zip`.

Since the performance of GP depends on the selected parameter values, all parameters underwent an in depth optimisation as described in a previous study [65]. More details about the parameter tuning and selected parameter values can be found in the appendix. In the appendix only the part concerning the general GP parameters, whereas the parameters that are specific for each of the considered methods have also been optimised, and the results are presented in more detail in the next section. For the benchmark results to be statistically significant, each experiment was executed at least 30 times, while preserving the best solution from each run. The resulting best solutions from these runs were used to calculate quantitative information such as the median of the fitness, as well as the minimum and the maximum fitness value. The Mann-Whitney statistical test was used to determine if there is a statistically significant difference between the results obtained by two different experiments. The difference between two results will be considered significant if a p value smaller than 0.05 was achieved. Additionally, Spearman's rho test will be performed in several cases to analyse if there is a correlation between two variables.

5 Results

This section gives an overview of the results achieved for the four selected methods and their combinations. The first four sections will analyse the performance of a single approach with regards to different parameter choices. The fifth section will compare the results obtained by these approaches and their combinations to those obtained by the ATC rule and genetic algorithm.

5.1 Results for DRs with static terminals

In this section the results of using additional static terminal nodes will be presented. All the proposed static nodes will be used in addition to the terminal nodes from Table 1. Since 23 static nodes were proposed, it is hardly possible to test all node combinations to obtain the best one. For that reason, two greedy heuristics are used to guide the selection of static nodes, the *constructive* and *destructive* heuristic. The constructive heuristic will start with a set of terminal nodes from Table 1, and add only those static terminal nodes which happen to increase the performance of the generated DRs. On the other hand, the destructive heuristic starts with a set containing all nodes, and removes those static nodes whose removal from the set leads to the largest improvement of the automatically generated DRs.

By using the two aforementioned greedy heuristics for selecting static terminal nodes, more than 500 experiments and node combinations were tested. Since the

Table 4: Results achieved by using additional static terminal nodes

	Static terminals	min	med	max
1	*FLD, FLDS, FUTLATE, FUTLATEL, FUTLATES, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDL, WLDS*	12.20	13.83	15.37
2	*FLD, FLDS, FUTLATE, MLOADD, NREL, NSHORT, SLAVGD, SLNEXT, SLNEXTM, TTARM, WLATE, WLATEL, WLATES, WLD, WLDS*	12.27	**13.21**	15.10
3	*FLD, FUTLATE, MLOADD, NREL, SLAVGD, SLNEXT, WLATES, WLD*	12.08	13.30	**14.62**
4	*FLD, FUTLATE, MLOADD, NREL, NSHORT, SLNEXT, WLATES, WLD*	12.08	13.42	16.06
5	*FUTLATES, NREL, SLAVGD, SLNEXT, WLD*	**12.06**	13.30	16.59

number of tested combinations of static nodes is vast, only the 5 combinations which achieved the best minimum values on the test set are selected and presented in Table 4. The best obtained values are denoted in bold. The best overall DR was generated when using the static terminal node combination denoted with index 5. From the table it is evident that the best minimum values are mostly achieved by experiments which use smaller terminal node sets, such as those used by experiments 3, 4, and 5. In all terminal node sets different terminal node types were used, which leads to the conclusion that better results are achieved by simultaneously using nodes which provide different kind of information to the DR. Nodes *NREL*, *SLNEXT*, and *WLD* were used in all five experiments, thus outlining that they might be the most informative of the nodes which were proposed. From the node definitions, ti can be seen that each of these three nodes provides quite different information, from the simple number of jobs that become available during the execution of the current job, to the slack of the next job and the approximation of tardy jobs. On the other hand, nodes *FLD*, *FUTLATE*, *MLOADD*, *SLNEXT*, and *WLATES* were used in four experiments. Here it is evident that most of them provide approximations of the tardiness during the execution of the next job. Thus, these tardiness approximations seem to be informative to the generated DRs. Figure 2 shows the box plot representation of the results obtained from different static terminal combinations. The box plot shows that by using the static terminal nodes GP obtains dispersed results. For all the static terminal node combinations similar minimum values are obtained, meaning that most of the tested node combinations have a similar expressiveness.

5.2 Results for DRs with look-ahead

This section will present the results achieved by using look-ahead in automatically designed DRs. Look-ahead will be tested with both the look-ahead factor and a fixed

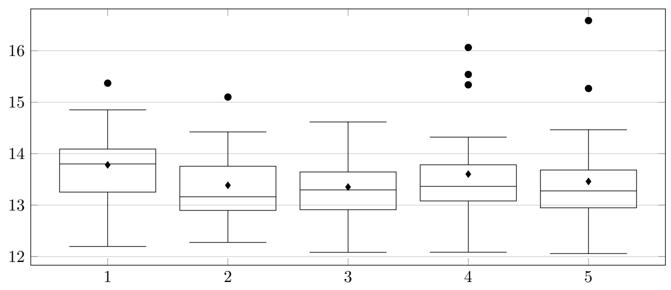


Fig. 2: Box plot representation of results for DRs with static terminal nodes

Table 5: Results for DRs with look-ahead using only nodes for dynamic scheduling

Look-ahead factor	min	med	max	Number of jobs	min	med	max
0.03	11.88	13.45	15.56	3	11.24	12.25	15.06
0.05	12.20	13.53	15.16	5	11.31	12.06	14.32
0.1	12.12	12.78	14.55	10	**10.82**	11.83	14.60
0.2	11.67	12.44	**13.89**	20	10.96	11.83	13.60
0.5	11.30	**11.99**	14.86	50	11.17	11.69	15.30
1	**11.17**	12.02	15.35	100	11.02	**11.64**	**13.53**

number of jobs in the look-ahead horizon, to determine how this parameter influences the overall procedure.

Table 5 represents the results achieved by DRs which additionally use look-ahead. In this case, DRs will use PFs generated by using the terminal set consisting only of nodes used for the dynamic scheduling environment, and the *AR* node. The table shows that the size of the look-ahead horizon significantly influences the performance of the method. When using the look-ahead factor the results gradually improve as the value of the factor is increased up until 0.5, since with that value the DRs already seem to have a good overview on the problem. The same behaviour can be noticed when using the fixed number of look-ahead jobs, however, the increase in the performance is not that drastic as when using the look-ahead factor. An additional interesting behaviour is that the increase in the number of jobs in the look-ahead horizon will not always lead to better results. This is evident by comparing results achieved when using 10 and 20, or 50, and 100 jobs in the look-ahead horizon, since the median values in those experiments are similar. Thus, it is possible to conclude that already by considering only a small number of jobs in the look-ahead horizon the performance of DRs can be increased significantly. However, the results also show that it is not necessary to consider too many jobs in the future, which is due to the fact that jobs that are too far in the future anyhow have a small probability of being scheduled at the current time.

The results display that DRs achieve better performance when using a fixed number of jobs in the look-ahead horizon, than when using the look-ahead factor. The main reason for such behaviour is that by using a constant number of jobs in the horizon the procedure is more stable since it will always consider the same number of unreleased jobs. On the other hand, when using the look-ahead factor, the number of unreleased jobs which are considered depends not only on the number of jobs in the problem instance, but also on the distribution of the release times of jobs. The reason for this is that the look-ahead factor is used only to define a time window, and jobs that are released during that time window belong to the look-ahead horizon. However, it is possible that in certain time windows no jobs are released, and therefore no unreleased jobs would be considered. For example, if 10 jobs in the look-ahead horizon are used, this means that the DR will always considered additional 10 jobs that are not yet released. However, if the look-ahead factor of 0.1 is used, this does not means that 10% of jobs will be considered at each moment, but rather that next 10% of the total time interval will be considered. However, the number of jobs that are released during that period can vary during system execution. In some cases it can happen that not even one job is released, thus the DR will not have any benefits from it and will at that moment work as normal DRs without look-ahead. Therefore, the behaviour of DRs that use the look-ahead factor can be more volatile than when using a fixed number of jobs in the look-ahead horizon. Because of this variability the performance of DRs is not as good as when using a constant number of jobs in the look-ahead horizon.

Figure 3 shows the box plot representation of the results. The figure demonstrates that designing DRs with look-ahead can lead to the appearance of outliers, but this does not significantly affect the performance of the methods. Additionally, the figure depicts that the solution distributions for look-ahead factors of 0.5 and 1, and for certain numbers of jobs in the look-ahead horizon (from 10 to 100) are quite similar with smaller variations in the values of the median and minimum values. This means that it is possible to use smaller values of the parameters without a great deterioration in the results. Based on all the observations it is evident that the look-ahead method should preferably be used with a fixed number of jobs in the look-ahead horizon.

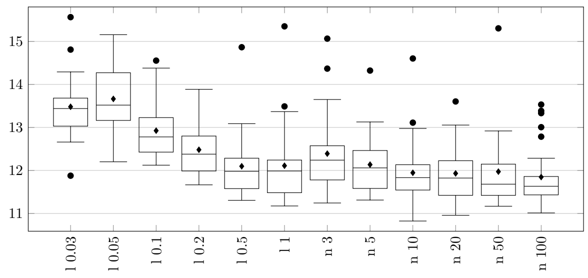


Fig. 3: Box plot representation of results for dispatching rules with lookahead

Table 6: Results for IDRs by using various IDR nodes

	IDR terminals	min	med	max
1	*INDTARD*	12.09	13.19	14.77
2	*INDWTARD*	12.09	**13.07**	14.40
3	*INDWTARD, NLATE*	11.87	13.08	**13.94**
4	*INDTARD, INDWTARD, NLATE*	**11.82**	13.18	14.39
5	*INDWTARD, NLATE, INDTARD, INDLATE, LATE, TARDINESS*	12.06	13.21	14.51

5.3 Results for IDRs

In this section the results for IDRs with different combinations of IDR nodes will be presented. Because testing all possible combinations of those nodes would be too time consuming, the same destructive and constructive heuristics, which were used for creating the sets of static terminal nodes, will also be used here to create sets of IDR nodes. In total around 80 combinations of IDR nodes were tested, however, only the five best were selected and are presented in this section.

Table 6 represents the results achieved by the five best combinations of IDR nodes on the test set obtained by the constructive and destructive heuristics. It is evident that for IDRs to work well not many additional nodes are needed, but rather it is important to select those which hold useful information. The experiments show that for optimising the *Twt* criterion the best results are achieved when using nodes which contain information about the tardiness of the jobs. IDRs usually achieved the best results when using the *INDTARD*, *INDWTARD* and *NLATE* nodes. The first two nodes are important since they denote the tardiness and weighted tardiness for each of the jobs, while the third node gives a notion about the number of jobs which were late in the previous schedule. Nodes that do not contain tardiness information, like *FLOWTIME* or *JOBFINISH*, are not useful, which can be seen from the fact that they do not appear at all in the five best IDR node combinations. Furthermore, the results also demonstrate that by using the *LATENESS* and *TARDINESS* nodes, which represent the tardiness and lateness of the entire schedule, IDRs usually do not achieve good results. This can be seen from the fact that by including these nodes the results start to deteriorate. Therefore, nodes which provide information about the tardiness of individual jobs lead to a much better performance of IDRs. This makes sense since nodes that provide information about the individual tardiness or lateness of jobs can help the DR to identify critical jobs, whereas the information about the total tardiness or lateness of the schedule can not be used for that purpose. Figure 4 shows the box plot representation of IDRs for the different combinations of IDR nodes.

5.4 Results for the rollout algorithm

In this section the results for the rollout algorithm by using DRs generated by DGP will be presented. Table 7 denotes the results achieved by the rollout algorithm. It can

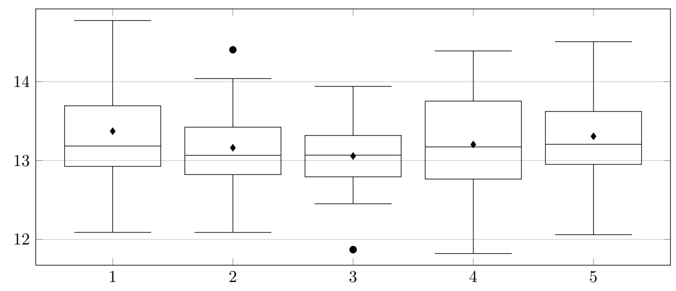


Fig. 4: Box plot representation of results for IDRs

be noticed that the performance of the rollout algorithm depends heavily on whether the rollout factor or number of jobs in the rollout horizon is be used. The rollout algorithm achieved much better performance when a constant number of jobs in the rollout horizon was used. An additional thing which can be observed is that after a certain value for the number of jobs in the rollout horizon the results do not improve further, which means that considering more jobs is not beneficial.

Table 7: Results for the rollout algorithm when using DRs generated by DGP

Rollout factor	min	med	max	Number of jobs	min	med	max
0	10.51	10.87	11.21	0	10.51	10.87	11.21
0.03	10.07	10.63	10.98	3	9.956	10.37	**10.65**
0.05	10.16	10.56	11.07	5	9.883	10.24	11.18
0.1	10.07	10.41	**10.77**	10	9.956	10.22	11.09
0.2	9.956	10.36	10.79	20	**9.790**	**10.08**	10.88
0.5	10.05	10.27	11.12	50	**9.790**	**10.08**	10.88
1	**9.790**	**10.08**	10.88	100	**9.790**	**10.08**	10.88

Figure 5 shows the box plot representation of the results. The figure illustrates how different values of the rollout parameters influence the performance of the rollout algorithm. The experiments which use the rollout factor are denoted with "r" and the value of the parameter, while the experiments which use a fixed number of jobs in the rollout horizon are denoted with "n" and the value of the parameter. The figure denotes that good results are achieved even by small values for the number of jobs in the rollout horizon, and that those results slowly improve as the value of the parameter rises, up until the value of 20, after which no improvement is achieved. On the other hand, when using smaller values for the rollout factor, the results that are quite bad, and only for larger values does the rollout algorithm achieve good results. Based on the outlined results it can be concluded that the rollout should preferably be used with a constant number of jobs in the rollout horizon. Furthermore, there is no need to use a large value for the number of jobs in the rollout horizon, because it will usually not lead to significant improvements beyond a certain point, which can nicely be seen when the fixed number of jobs is used with the rollout algorithm is used. Namely, the number of jobs in the rollout horizon larger or equal to 20 there is no difference in the obtained results. The reason is that jobs which are released much later in time are hardly going to be scheduled at the current moment in time, since other jobs can be scheduled and executed in the meantime. Waiting for jobs which are released too much in the future would just lead to a poor schedule and significantly increase the tardiness of the entire schedule.

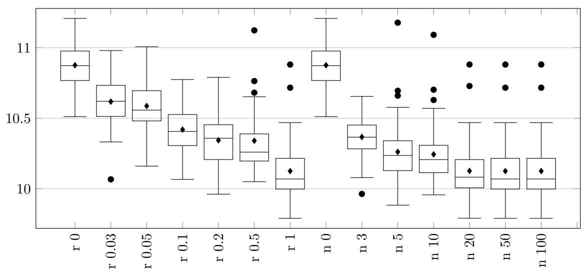


Fig. 5: Box plot representation of results for the rollout algorithm

5.5 Comparison of all static scheduling methods

In this section the best results from all the methods will be compared with each other. Since experiments from different methods will be used, a proper nomenclature needs to be defined. If static terminal nodes were used, this will be denoted with "S" and the index of the combination of static nodes (from Table 4) that was used. The experiments which use look-ahead will be denoted with "L", with "l" denoting that the look-ahead factor is used, and "n" that a constant number of jobs in the look-ahead horizon is used. The value of the look-ahead parameter will be denoted immediately after the "l" or "n" flag. If the static terminal nodes are used together with look-ahead, then the flag "u" will be used to denote that the static terminals are calculated based on all unreleased jobs, while the "ul" flag will denote that the static terminals are calculated based on all unreleased jobs outside the look-ahead horizon. The use of IDRs will be denoted with "I", after which the index of the applied IDR node combination (from Table 6) will be denoted. The experiments which use the rollout algorithm will be denoted with "R", followed by either "r", if the rollout factor is used, or "n", if the number of jobs in the rollout-horizon is used. The value of the rollout parameter will follow immediately after the "r" or "n" flag.

To measure the performance of the tested approaches, their results will be compared to several other values and methods. The value denoted as *baseline* represents the union of the best solutions for each problem instance in the test set, which were obtained by various methods (including all methods mentioned in this paper). Although this does not represent the true lower bound for the problem, it should be close to it since a wide variety of algorithms and optimisation methods could not obtain better results. The results obtained by automatically designed DRs generated by GP, but without using any static information, are also included and denoted as DGP. They will serve for outlining the improvements that each static method can obtain in comparison when no static information is used. Furthermore, the results achieved by the static version of the ATC rule [67] are also included. This version of the ATC rule will calculate the priorities of jobs using the following PF: $\pi_{i,j} = \frac{w_T}{p_{ij}} \exp\left[-\frac{\max(d_j - p_{i,j} - \max(r_j, time), 0)}{k_1 \bar{p}}\right] \exp\left[-\frac{\max(r_j - time, 0)}{k_2 \bar{p}}\right]$, where *time* denotes the current time of the system, $\bar{p}$ the average processing time of all jobs waiting to be scheduled, k_1 and k_2 the scaling parameters. The parameters which were used by the ATC rule were 0.7 for k_1 and 0.2 for k_2, since for those values the rule achieved the best results on the training set.

The results will also be compared to two genetic algorithms, floating point GA (GA-FP) and permutation GA (GA-PERM), with the difference being in the encoding of the solutions. The reason why genetic algorithms were selected is because in several preliminary experiments they obtained among the best results in comparison with several other methods. GA-FP encodes the solutions as an array of n numbers between 0 and 1. These numbers determine the order in which the jobs should be scheduled. In addition, they are also used to determine the machine on which each job should be scheduled in a way that the interval from 0 to 1 is split into m subintervals, and depending to which subinterval the value of the job belongs it will be scheduled on the corresponding machine. The GA-PERM algorithm represents the solution as a permutation of n numbers. This permutation array denotes the sequence in which the jobs have to be scheduled, but do not determine the allocation of jobs to machines. This is determined by a heuristic which schedules the current job on the machine on which it would finish with its execution the soonest. The reason why both algorithm were included is because GA-FP represents the most computationally efficient representation with regards to the evaluation of the fitness function, while GA-PERM achieves the best possible results, although with a higher execution time. More details about the genetic algorithms and the parameter values can be found in the appendix.

Table 8 contains the best results of all methods and their combinations, as well as the average values of execution times in seconds calculated based on all experiment runs. The results are additionally represented using box plots in Figure 6. It should be stressed out that the table and figure denote only the few bes results for all the methods and their combinations. However, for each combination of methods the parameters were optimised similarly as it was denoted in the last sections for each parameter individually. Therefore, it is ensured that the results for each method, and combination of methods are representable. DRs with static terminal nodes (indices 6 and 7) achieve the worst results among all the tested methods. Although the two results denoted in the table achieve significantly better results than DGP, the improvement in performance is too small to justify their use, since the increased complexity did not have a large effect on the results. Furthermore, it is evident that they are also inferior to the ATC rule, which signalises that they are not very competitive. The same holds for IDRs (indices 12 and 14). They achieve significantly better results than DGP and also perform better than DRs with static terminal nodes, but still fall behind the ATC rule. Thus, these two methods do not seem to bring any merit on their own, since the ATC method outperforms both of them. The look-ahead method (indices 8 and 9) achieved more competitive results. It does not only significantly outperform DGP, DRs with static nodes, and IDRs, but performs much better than the ATC rule. Although it might seem that the look-ahead methods is a bit unstable due to the fact that it obtains several outliers, it is still evident that most of the obtained results are better than the best solution found by DGP. A further benefit of this method is that it results in only a slight increase in the execution time compared to DGP. The results are still far away from the baseline by around 20%. When comparing the results achieved by look-ahead with GA-PERM, which achieved the best results, it can be seen that although DRs with look-ahead achieve around 18% worse results, they construct the schedule 4000 times faster.

The combinations of static nodes with look-ahead (indices 10, 11, and 12) and static nodes with IDRs (index 15) do not bring any benefits, or improve the results only slightly. Look-ahead and IDRs seem to already have a good overview on the problem, so the additional information that is provided to them by static terminal nodes does not improve the results. However, combining IDRs with look-ahead (indices 16 and 17) does significantly improve the performance when compared to both methods individually. This demonstrates that these two methods complement each other well, with look-ahead providing information about the future in the current schedule, and IDRs providing information about previously created schedules which can serve to correct certain mistakes which would be done only by using look-ahead. Thus, look-ahead will ensure that in each iteration a better schedule is constructed, whereas the iterative part will ensure that in the next iteration the DRs fixes possible problems in the schedule. The downside is that the combination is to a certain extent slower than DRs with look-ahead. However, this is still negligible when compared to the execution time of the GAs, since the method is still around 2000 times faster and achieves results which are worse of those obtained by GA-PERM by around 14% on average. Due to the fast execution speed of this method, it would also be possible to try out all the generated DRs and select the best solution. In this case the method would obtain results which are only by 9% worse, but it would still be 70 times faster. Adding static terminal nodes to the combination of IDRs and look-ahead (indices 18 and 19) did not improve the obtained results, making this combination redundant. Again, this is due to the reason that IDRs and look-ahead already have a good overview of the entire problem.

Out of the tested GP methods the rollout algorithm obtains the best results, which comes at the price that the execution times are significantly larger in comparison to the other three methods. By using the rollout algorithm with dynamic DRs (indices 20 and 21) it is already evident that the method achieves results which are significantly better than those obtained by any of the other static DR methods. The methods in this case are even comparable with the results obtained by GP-FP, although still falling behind GA-PERM by around 5%. Combining rollout with DRs with static nodes (index 22) does not improve the results and only prolongs the execution time. This is expected since the rollout algorithm relies on a good DR to create a good approximation of the remaining schedule. Since DRs with static terminals have not

Table 8: Results for the execution times of the different methods

Index	Method	min	med	max	execution time
1	Baseline		9.419		-
2	DGP	12.96	13.60	14.62	0.091
3	ATC	12.45	12.45	12.45	1.899
4	GA-FP	9.917	10.27	10.90	339.1
5	GA-PERM	9.521	9.584	9.695	681.0
6	S-2	12.27	13.21	15.10	0.105
7	S-5	12.06	13.30	16.59	0.105
8	L-n-10	10.82	11.83	14.60	0.113
9	L-n-100	11.02	11.64	13.53	0.159
10	L-n-10 S-u-5	11.22	12.46	14.21	0.144
11	L-n-10 S-ul-5	10.90	12.24	14.11	0.144
12	L-n-20 S-ul-5	11.10	11.64	14.54	0.156
13	I-3	11.87	13.08	13.94	0.155
14	I-4	11.82	13.18	14.39	0.155
15	I-4 S-5	12.11	12.88	14.31	0.180
16	I-4 L-n-5	10.75	11.17	14.41	0.207
17	I-4 L-n-100	10.53	11.10	12.44	0.321
18	I-4 L-n-10 S-ul-5	10.77	11.42	14.29	0.350
19	I-4 L-n-100 S-ul-5	10.52	11.85	13.07	0.456
20	R-n-3	9.956	10.37	10.65	101.0
21	R-n-20	9.790	10.08	10.88	375.5
22	R-n-20 S-5	9.769	10.08	10.74	495.6
23	R-n-10 L-n-10	9.713	9.914	10.62	204.9
24	R-n-20 L-n-10	9.744	9.903	10.74	356.3
25	R-n-3 L-n-100	9.773	10.05	10.69	98.11
26	R-n-10 L-n-20 S-u-5	9.731	9.999	10.94	352.9
27	R-n-3 L-n-50 S-u-5	9.965	10.27	11.11	179.3
28	R-n-3 L-n-20 S-ul-5	9.810	10.10	10.64	175.9
29	R-n-10 L-n-20 S-ul-5	9.674	9.898	10.25	372.8
30	R-n-10 L-n-100 S-ul-5	9.750	9.917	10.27	1603

achieved a good improvement over dynamic DRs on their own, they are also unable to improve the performance of the rollout algorithm. However, by combining rollout with DRs that use look-ahead (indices 23, 24, 25) leads not only to improvement in the results, but also smaller execution time. The rollout algorithm can now be applied with a smaller number of jobs in the rollout horizon and still achieve similar results as when a larger horizon would be used with dynamic DRs. For example, in experiment 25 the rollout algorithm still achieves results that are worse than those of GA-PERM by 4.6%, but is 7 times faster. In this case the method would be inferior to the baseline by only around 6% on the average.

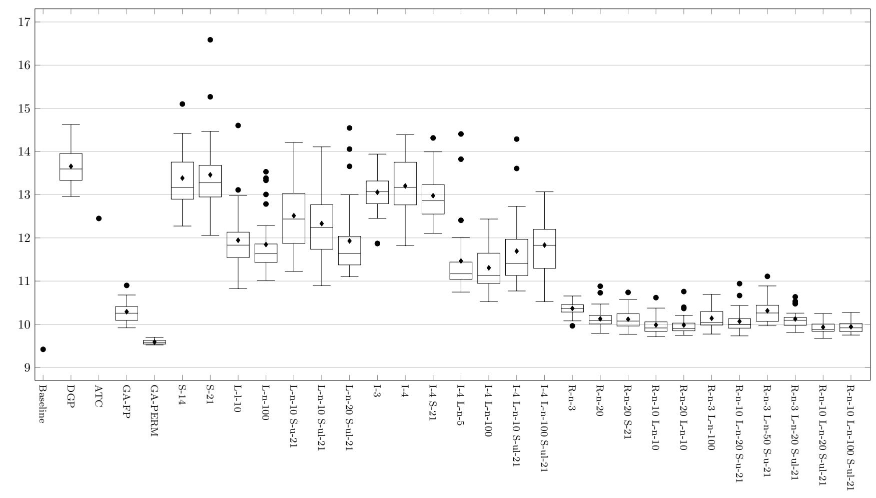


Fig. 6: Box plot representation of the results for all methods

Although one could think that by decreasing the execution time of GA-PERM it could be made more competitive to the rollout algorithm, this is not the case. For example, if GA-PERM is given roughly the same time as the rollout required for experiment 25, it achieved a result of around 11.71. This shows that the algorithm has a quite slow convergence, and that it cannot compete with the rollout algorithm when speed is also considered. By additionally using static terminal nodes it is possible to further increase the performance of the rollout algorithm (indices 26 to 30), but again with an increase in the execution times. However, with these combinations the rollout algorithm obtains its best results (index 29). In that case the rollout algorithm is still inferior to GA-PERM by 4% on the average, but the best result which rollout obtained was only 1.6% worse than the best solution found by GA-PERM, and 2.6% worse than the baseline solution. This proves that the rollout algorithm is expressive enough to obtain high quality solutions similar to results of GAs or other metaheuristic methods.

To better present the correlation of the execution times and the obtained results, Figure 7 represents the dependence of the achieved minimum and median values with the execution times. Each point is denoted with the index of the experiment it represents from Table 8. Furthermore, the points which represent the Pareto front of solutions are denoted in red, while the rest of solutions are represented with blue points. Since the execution times for different methods have vastly different values, a logarithmic scale was used for the axis which represents the execution times. The figure shows how the different approaches are grouped together based on their performance and execution times. The shortest execution times, but also the worst results, are achieved by DRs evolved by DGP which is denoted with the index 2. The group of results which use static terminal nodes, look-ahead, IDRs, or any combinations of those methods are represented by points with indices from 6 to 19. These methods can be seen to cover quite a large part of the values for the *Twt* criterion, with only small differences in the execution times. Therefore, by slight increase of the execution time, large variation in the achieved results can be obtained. The static ATC rule (index 3) achieved better results than DGP, but due to its slow execution time it is not competitive to the other methods. The second group of results, those achieved by the rollout algorithm and combinations of it with other methods, are denoted by indices from 20 to 30. The results of these methods are centred around the result achieved by the GAs (indices 4 and 5). This group of results shows to cover a small range of the values for the Twt criterion, but a large part of the execution times achieved by the procedures. This means that to improve the results even by a small extent, it is required to significantly prolong the execution times. Nevertheless, in some cases the rollout algorithm can achieve results close to those of GA-PERM, but in a much shorter time.

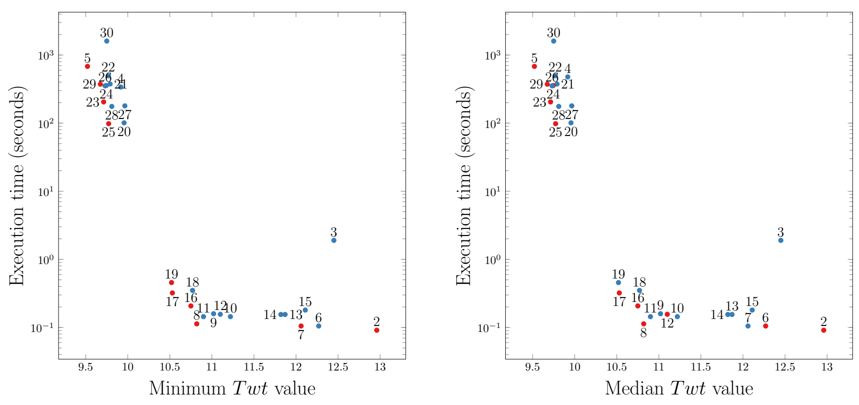


Fig. 7: Dependency between the execution times and fitness achieved by the methods

Based on the presented results, several conclusions can be drawn. Static terminal nodes, look-ahead, IDRs, and various combinations of those methods have achieved better results than DGP, with only a small increase in the execution time, which was at most 5 times larger than that of DGP. Out of these methods static terminals and IDRs achieve inferior results than look-ahead, and also the static ATC. Thus, these are not methods which should be used by themselves. However, the combination of IDRs and look-ahead leads to better results that are still relatively far from the baseline, but can be obtained in an almost negligible amount of time. This makes these methods preferable if the execution time is as important as the quality of the schedule. Additionally, because of their small execution times, it is possible to execute several DRs for a certain problem, and select the best obtained solution. On the other hand, the rollout algorithm achieved the best results, however, with much larger execution times. By combining the rollout algorithm with look-ahead and static terminal nodes it was possible to improve its results and decrease its execution time. Such combinations achieved results which are to a smaller extent worse than the baseline and the GA-PERM methods, with an execution time that is several times smaller. This makes the rollout algorithm a better choice if the quality of the obtained results is of primary

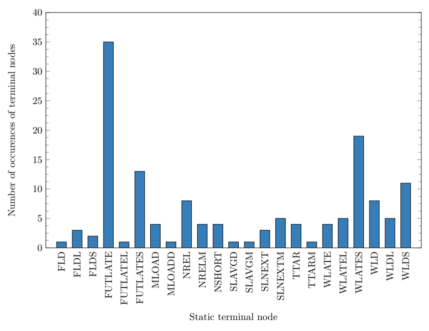


Fig. 8: Frequency of static terminal nodes

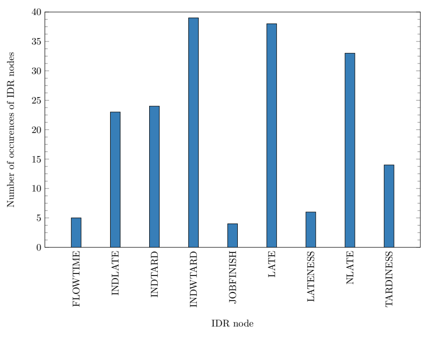


Fig. 9: Frequency of IDR nodes in the DRs

importance, and the execution time is of less concern. Furthermore, unlike the GAs, this method could also be applied for dynamic off-line scheduling, which makes it applicable for a larger class of problems.

6 Analysis of static scheduling methods

6.1 Analysis of DRs with static terminal nodes

In this section the frequency with which the static terminal nodes appear in the individuals will be analysed. Thirty DRs which achieved the best performance on the test set were selected, and the occurrence frequency of each static node in those individuals was calculated.

Figure 8 shows a histogram which represents the number of occurrences of each static node. The $FUTLATE$ node appears by far most frequently in the DRs, which would mean that it is the most informative of the nodes. The $FUTLATES$ node also appears quite often, which denotes that the information about the possible tardiness of jobs which would be released during the execution of the current job is useful to the DRs. The other nodes which also take into account the possible tardiness of the current job if it would not be scheduled right away were used seldom. This leads to the conclusion that the more simple approximations are already informative enough. The nodes which approximate the weighted number of tardy jobs, if the current job would not be scheduled, also appear quite often in the individuals. Therefore, nodes which try to approximate the tardiness of jobs appear most often and seem to be the most informative.

6.2 Analysis of IDRs

The occurrence frequency of IDR nodes will be analysed in the same way, meaning that the 30 IDRs, which achieved the best results on the test set, were collected and the number of occurrences of each IDR node was calculated. Figure 9 represents the histogram of the occurrence number of each IDR node. The *INDWTARD* terminal appears most often, and seems to be useful since it directly supplies the information about the weighted tardiness of a job in the previous schedule. The *LATE* function node also appears very often, meaning that allowing for parts of the DR to specialise for tardy jobs, while others specialise for jobs which are not tardy, provides to be beneficial for the performance of the IDRs. The *NLATE* node is also frequently used, thus the IDRs seem to benefit from the information on how many jobs were tardy in the previous schedule. The *INDLATE* and *INDTARD* nodes are used to a certain extent less than the previously mentioned nodes. The reason for this seems to be the absence of the job weight in their calculation. Finally, the *TARDINESS* node appears in even less occasions, meaning that the information about the total weighted tardiness of the previous schedule is not important to the IDRs. Based on the these observations the *INDWTARD*, *INDTARD*, *INDLATE*, *NLATE*, and *LATE* nodes seem to contain the most useful information for the IDRs.

For the IDRs it is also interesting to analyse the number of times IDRs recreate the schedule. This analysis will use the same 30 best IDRs which were used for the previous analysis. When all problem instances and all IDRs are considered the IDRs created the schedule 2.465 times on average. The last schedule which is created by the IDR, but not returned as the result, was also included in the calculations. The value shows that IDRs usually create the schedule two or three times. By analysing the number of created schedules for each problem instance independently, it was shown that for the easier instances the IDRs can find the best schedule immediately in the first iteration. However, the IDRs still need to validate that they cannot improve the schedule, which in the end means that for the easier problem instances they will create the schedule two times. For more difficult problem instances the schedule is usually created three or four times, with the maximum number of created schedules being seven. This shows that the IDRs do not have the tendency to recreate the schedule many times, but that they are rather capable of performing all improvements in only a few iterations. During the analysis it was also observed that a few of the better IDRs usually recreated the schedule only a few times, therefore it was tested whether there is a correlation between the fitness of the IDRs and the number of times IDRs recreate the schedule. To test the correlation between those two variables, the Spearman's rho test was performed, and the values of $\rho = -0.097$ and $p = 0.612$ were obtained. Based on these two values it is not possible to conclude whether a correlation between those two variables exists.

6.3 Analysis of the rollout algorithm

For the rollout algorithm it will be analysed whether there is a connection between the quality of the results produced by the rollout algorithm and the quality of the DR which was used for the approximations. The rollout algorithm with dynamic DRs was used to test this, and the values $\rho = 0.487$ and $p = 0.00034$ were obtained. Therefore it can be seen that there is a positive correlation between the quality of the rollout algorithm and the DR it uses. However, the correlation is not quite strong to accept it as a general rule. This can be also seen from the results, since several better results for the rollout algorithm are achieved when better DRs are being used, however there

Table 9: Details about the problem instance used for analysis

Job j index	0	1	2	3	4	5	6	7	8	9	10	11
r_j	15	98	1	25	47	31	59	42	56	3	21	19
dd_j	15	104	43	76	96	70	84	78	102	64	27	43
w_j	0.9	0.07	0.87	0.06	0.06	0.26	0.78	0.94	0.63	0.33	0.49	0.95
p_{0j}	73	62	47	58	65	38	12	33	92	99	1	18
p_{1j}	35	64	89	31	75	82	93	24	62	70	80	15
p_{2j}	45	5	9	24	47	19	92	62	31	92	18	96

are also cases where the rollout algorithm achieves quite good results even if the DR it uses performs poorly by itself.

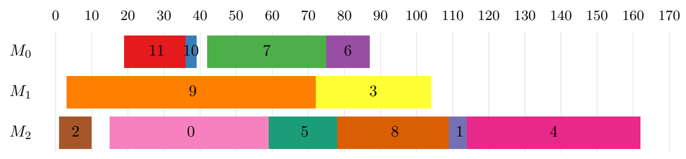
6.4 Comparison of schedules generated by different methods

To illustrate the behaviour of the different methods, the schedules they create for a simple scheduling problem instance will be analysed. Details about the problem instance used for the analysis are presented in Table 9.

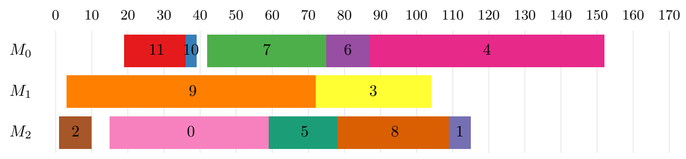
The schedules created by the dynamic DR and various static DR methods are presented in Figure 10. Subfigure 10a represents the schedule which is created by the best DR which was created by DGP. From the schedule it can be seen that the DR makes two bad decisions, which lead to an increased value of the total weighted tardiness criterion. The first bad decision is that job j_{11} was scheduled before j_{10}. Since the execution time of job j_{10} would be only one time unit, it would be better to first execute this job and delay the execution of job j_{11} until job j_{10} is finished. Unfortunately, job j_{11} is released prior to job j_{10}, and the DR has no means of detecting that very soon a new job will be released, which should have a higher priority of being scheduled. Thus, job j_{10} will have to wait until job j_{11} finishes with its execution, so that it can execute on machine M_0. This will consequentially lead to a larger tardiness value. The second bad decision can be observed at the end of the schedule, where the DR scheduled job j_4 on machine M_2. This job could have been scheduled at an earlier time on M_0 to reduce the tardiness. However, the DR chose to schedule job j_4 on the machine for which it has the minimum processing time, and keep the machine M_0 free in case that other jobs would arrive into the system. In the end, the schedule generated by the dynamic DR achieves the total weighted tardiness value of 0.596.

Subfigure 10b represents the schedule created by a selected DR with static terminal nodes. The figure shows that the DR with static terminal nodes fixes only one of the bad decisions which were performed by the dynamic DR. This DR performed a better decision at the end of the schedule, by scheduling job j_4 on machine M_0 rather than on machine M_2. The total weighted tardiness of this schedule is 0.590.

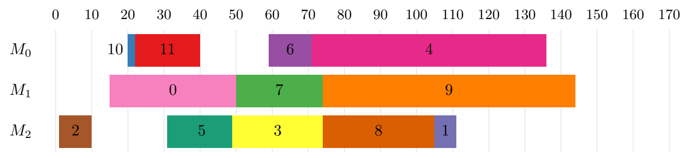
The selected IDR created the schedule three times. The first created schedule is presented in Subfigure 10a. The figure shows that the schedule constructed by this IDR is the same as the one created by the dynamic DR. It is expected that in the first iteration the IDR will perform equally well or worse than the dynamic DR, since the IDR did not yet have access to any additional static information. Subfigure 10b



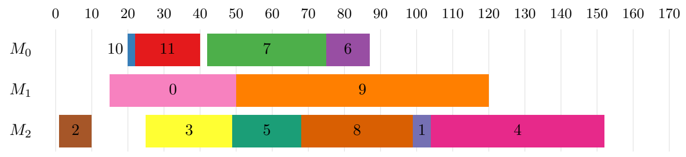
(a) Schedule generated by the best dynamic DR, and the IDR in the first iteration



(b) Schedule generated by a DR with static terminal nodes, and the IDR in the second iteration



(c) Schedule generated by the best DR with look-ahead



(d) Schedule generated by the rollout algorithm

Fig. 10: Schedules generated by the static DR methods

denotes the schedule which was created in the second iteration by the IDR. The figure shows that the IDR created the same schedule as the DR with static terminal nodes. It seems that the IDR was able to capture the tardiness of job j_4 in the last schedule, and use this information to schedule it to another machine in the second iteration. In the third iteration the IDR obtained a schedule of the same fitness value, and therefore the procedure was terminated. In the end, the IDR created a schedule with a *Twt* value of 0.590.

Subfigure 10c shows the schedule which was created by a DR which uses look-ahead. The figure shows that the DR was able to fix both bad decisions which the dynamic DR made. First it was able to determine that job j_{11} should not be scheduled on machine M_0 the very moment it arrives into the system, but with the help of look-ahead it was able to determine that a job which has a higher priority will very soon arrive into the system, and thus it delayed the scheduling of job j_{11}. When job j_{10} arrives into the system, it is immediately scheduled on machine M_0, and after it has finished with its execution, job j_{11} is executed. At the end of the schedule, the DR was also able to determine that it would be better to schedule job j_4 immediately on machine M_0. Look-ahead can also be helpful in this situation, since the DR will have a clear oversight of the look-ahead horizon, and will be aware that no new jobs will be released during it, therefore it can prioritise the available jobs and schedule them without any drawbacks. The schedule also shows that the DR introduced several idle times into the schedule to keep the machines free for high priority jobs which arrive in the near future. Therefore, the DR kept machine M_1 free until job j_0 comes into the system, since this job has a high weight and executes the fastest on machine M_1. The same happens for machine M_0, where the DR does not schedule job j_7, but rather waits for job j_6, and schedules job j_7 on machine M_1 on which it achieves the fastest processing time. For machine M_2 the same thing can be observed when scheduling jobs j_3 and j_5. Therefore, it can be concluded that the look-ahead provides DRs with more information than the previous two methods, and allows them to create better schedules. Even though the DR introduced a lot of idle times into the system, it nevertheless achieved a better performance than the previous two methods, with the *Twt* value being 0.574.

Finally, Subfigure 10d represents the schedule created by the rollout algorithm. The schedule is very similar to the one obtained by the DR with look-ahead, however, the rollout algorithm tries to reduce the amount of idle times which are introduced into the schedule. Once again, for machine M_0, it was shown to be more beneficial if job j_{10} is executed prior to job j_{11}. The rollout algorithm also determined that a certain amount of idle time should be introduced on machine M_1 so that job j_0 can start immediately with its execution, to minimise the tardiness caused by it. However, rollout determined that it was better to execute job j_7 on machine M_0 as soon as it arrives, since this will allow for job j_9 to be executed immediately on machine M_1. This will lead to a very small increase in the tardiness value for job j_6, but will allow for job j_9 to finish significantly earlier, and thus greatly reduce its tardiness value. Also, for machine M_2 the algorithm determined that it is better to immediately start executing job j_3, since it will not only prevent job j_5 of being late, but will additionally reduce the tardiness of jobs j_8 and j_1. In this schedule, job j_4 does finish at a latter time with its execution, than it was in the case when a DR with look-ahead was used. However, since the job has a very small weight, it will not have a large effect on the *Twt* value of the entire schedule. In the end, the *Twt* value of this schedule amounts to 0.510, which is the best value achieved by any of the methods.

Based on all previous observations it can be concluded that the rollout algorithm has the best overlook on the problem. This can best be seen in the comparison with look-ahead. The DR with look-ahead gives a higher priority to jobs which had a shorter execution time on the current machine, and thus introduced several idle times to keep the machines free for those jobs. However, this had a negative effect on the later parts of the schedule, where these decisions lead to an increased tardiness of some other jobs. Even with look-ahead it is hard for the DR to predict all the effects a scheduling decision could have on the future of the system. However, the rollout algorithm can try out all the combinations at each scheduling decision, and approximate the influence of this decision on the rest of the schedule with a good DR. This gives rollout an unparalleled overview of the problem, and allows it perform decisions at the beginning of the schedule, which will not have a negative influence on the later parts of the schedule.

7 Conclusion

The objective of this paper was to analyse different ways of adapting automatically generated DRs to improve their performance for the static and off-line scheduling problems. The paper analysed how additional static terminal nodes, look-ahead, IDRs, and the rollout algorithm improve the performance of DRs for scheduling problems under static conditions. The results demonstrate that the various methods offer different levels of improvement over dynamic DRs, and obtain vastly different execution times. The rollout algorithm achieved the overall best results out of the previous four methods. Although the method was unable to outperform the results of an effective GA, it obtained results that came quite close but in a smaller execution time. The other methods did not achieve better results than the GA. However, their execution time is almost negligible when compared to that of the GA, and in most cases they achieved better results than dynamic DRs. Out of these three methods, look-ahead achieved the best results and has demonstrated to offer the best results in the least amount of time required to construct them. Additional tests demonstrated that that combining various methods mostly leads to even better performance of the DRs.

Based on the previously outlined observations, it can be concluded that the tested static methods achieved improved results over the DRs generated for dynamic environments. Even more, the proposed rollout algorithm is demonstrated to be very competitive with GAs, obtaining results worse by only a few percent but in a much smaller amount of time. Since the tested methods have different execution times, it is possible to select the one which offers the best trade-off between the quality of the obtained results and the time needed to create the schedule. Although this study did consider four methods for adapting DRs to static conditions, it is still evident from the execution times that a huge gap exists which was not filled by any of the considered methods. Therefore, it should still be possible to cover that area by using other methods or adaptations of the methods considered in this study. Furthermore, this study considered only the total weighted tardiness criterion. Therefore, in the future this study could be extended by considering other scheduling criteria like the makespan and flowtime, or multi-objective optimisation could also be coupled together with these methods. Another possibility to improve this research would be to combine these methods with other methods that can improve the performance of DRs, but which are not exclusively applicable to static scheduling conditions (like ensemble learning). This would enable the methods to probably achieve even better results than those denoted in this paper. This research considers only four methods to adapt DRs for static conditions, but it would be possible to develop novel methods of adapting DRs for static scheduling problems. Another line of research would be to improve the used methods to obtain better results in less time. For example, it would be interesting that in each step the rollout heuristic does not create the entire schedule by using a DR, but rather a part of the schedule, so that the scheduling decision is based only on that partially created schedule. This should improve the execution time of the method, however, the question is how it would effect the performance. Thus, it can be seen that many open questions and possibilities for improvement remain in this topic.

Conflict of interest and funding

Funding: None.
Conflict of Interest: The authors declare that they have no conflict of interest.

References

1. Adyanthaya, S., Geilen, M., Basten, T., Schiffelers, R., Theelen, B., Voeten, J.: Fast multiprocessor scheduling with fixed task binding of large scale industrial cyber physical systems. In: Proceedings of the Euromicro Conference on Digital System Design 2013, 4-6 September 2013, Los Alamitos, California, pp. 979–988. Institute of Electrical and Electronics Engineers, United States (2013). DOI 10.1109/DSD.2013.111
2. Bertsekas, D.P.: Rollout Algorithms for Discrete Optimization: A Survey, pp. 2989–3013. Springer New York, New York, NY (2013). DOI 10.1007/978-1-4419-7997-1_8. URL `http://dx.doi.org/10.1007/978-1-4419-7997-1_8`
3. Bertsekas, D.P., Castanon, D.A.: Rollout Algorithms for Stochastic Scheduling Problems. Journal of Heuristics **5**(1), 89–108 (1999). DOI 10.1023/A:1009634810396. URL `http://link.springer.com/10.1023/A:1009634810396`
4. Branke, J., Hildebrandt, T., Scholz-Reiter, B.: Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations. Evolutionary Computation **23**(2), 249–277 (2015). DOI 10.1162/EVCO_a_00131. URL `http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00131`
5. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated Design of Production Scheduling Heuristics: A Review. IEEE Transactions on Evolutionary Computation **20**(1), 110–124 (2016). DOI 10.1109/TEVC.2015.2429314. URL `http://ieeexplore.ieee.org/document/7101236/`
6. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing **61**(6), 810–837 (2001). DOI 10.1006/jpdc.2000.1714. URL `http://linkinghub.elsevier.com/retrieve/pii/S0743731500917143`
7. Cheng, C.T., Lin, J.Y., Sun, Y.G., Chau, K.: Long-term prediction of discharges in manwan hydropower using adaptive-network-based fuzzy inference systems models. In: L. Wang, K. Chen, Y.S. Ong (eds.) Advances in Natural Computation, pp. 1152–1161. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
8. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. Computers & Industrial Engineering **30**(4), 983–997 (1996). DOI 10.1016/0360-8352(96)00047-2. URL `https://doi.org/10.1016/0360-8352(96)00047-2`
9. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Computers & Industrial Engineering **36**(2), 343–364 (1999). DOI 10.1016/s0360-8352(99)00136-9. URL `https://doi.org/10.1016/s0360-8352(99)00136-9`
10. Cheng, V., Crawford, L., Menon, P.: Air traffic control using genetic search techniques. In: Proceedings of the 1999 IEEE International Conference on Control Applications, vol. 1, pp. 249–254 (1999). DOI 10.1109/CCA.1999.806209. URL `http://ieeexplore.ieee.org/document/806209/`
11. Dimopoulos, C., Zalzala, A.: Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. IEEE Transactions on Evolutionary Computation **4**(2), 93–113 (2000). DOI 10.1109/4235.850651. URL `http://ieeexplore.ieee.org/document/850651/`
12. Dimopoulos, C., Zalzala, A.M.: A genetic programming heuristic for the one-machine total tardiness problem. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol. 3. IEEE (1999)
13. Dimopoulos, C., Zalzala, A.M.: Investigating the use of genetic programming for a classic one-machine scheduling problem. Advances in Engineering Software **32**(6), 489–498 (2001)
14. Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research **153**(1), 3–27 (2004). DOI 10.1016/s0377-2217(03)00095-x. URL `https://doi.org/10.1016/s0377-2217(03)00095-x`

15. Ferreira, C.: Gene expression programming: a new adaptive algorithm for solving problems. Complex Systems **13**(2), 87–129 (2001). URL `http://arxiv.org/abs/cs/0102027`
16. Fotovatikhah, F., Herrera, M., Shamshirband, S., wing Chau, K., Ardabili, S.F., Piran, M.J.: Survey of computational intelligence as basis to big flood management: challenges, research directions and future work. Engineering Applications of Computational Fluid Mechanics **12**(1), 411–437 (2018). DOI 10.1080/19942060.2018.1448896
17. Gao, J., Gen, M., Sun, L., Zhao, X.: A hybrid of genetic algorithm and bottleneck shifting for multi-objective flexible job shop scheduling problems. Computers & Industrial Engineering **53**(1), 149–162 (2007). DOI 10.1016/j.cie.2007.04.010. URL `https://doi.org/10.1016/j.cie.2007.04.010`
18. Geiger, C.D., Uzsoy, R., Aytuğ, H.: Rapid Modeling and Discovery of Priority Dispatching Rules: An Autonomous Learning Approach. Journal of Scheduling **9**(1), 7–34 (2006). DOI 10.1007/s10951-006-5591-8. URL `http://link.springer.com/10.1007/s10951-006-5591-8`
19. Gogna, A., Tayal, A.: Metaheuristics: review and application. Journal of Experimental & Theoretical Artificial Intelligence **25**(4), 503–526 (2013). DOI 10.1080/0952813X.2013.782347
20. Hansen, J.V.: Genetic search methods in air traffic control. Computers & Operations Research **31**(3), 445–459 (2004). DOI 10.1016/S0305-0548(02)00228-9. URL `http://linkinghub.elsevier.com/retrieve/pii/S0305054802002289`
21. Hart, E., Ross, P., Corne, D.: Evolutionary Scheduling: A Review. Genetic Programming and Evolvable Machines **6**(2), 191–220 (2005). DOI 10.1007/s10710-005-7580-7. URL `http://link.springer.com/10.1007/s10710-005-7580-7`
22. Hart, E., Sim, K.: A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling. Evolutionary Computation **24**(4), 609–635 (2016). DOI 10.1162/EVCO_a_00183. URL `http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00183`
23. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10, p. 257. ACM Press, New York, New York, USA (2010). DOI 10.1145/1830483.1830530. URL `http://portal.acm.org/citation.cfm?doid=1830483.1830530`
24. Hou, E., Ansari, N., Ren, H.: A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems **5**(2), 113–120 (1994). DOI 10.1109/71.265940. URL `https://doi.org/10.1109/71.265940`
25. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: C.A.C. Coello (ed.) Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers, pp. 263–277. Springer (2011). DOI 10.1007/978-3-642-25566-3_20. URL `https://doi.org/10.1007/978-3-642-25566-3_20`
26. Ishibuchi, H., Yoshida, T., Murata, T.: Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Transactions on Evolutionary Computation **7**(2), 204–223 (2003). DOI 10.1109/tevc.2003.810752. URL `https://doi.org/10.1109/tevc.2003.810752`
27. Jaković, D., Budin, L.: Dynamic scheduling with genetic programming. In: Genetic Programming: 9th European Conference, EuroGP 2006, Budapest, Hungary, April 10-12, 2006. Proceedings, pp. 73–84. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). DOI 10.1007/11729976_7. URL `https://doi.org/10.1007/11729976_7`
28. Jaković, D., Marasović, K.: Evolving priority scheduling heuristics with genetic programming. Applied Soft Computing **12**(9), 2781–2789 (2012). DOI 10.1016/j.asoc.2012.03.065. URL `http://linkinghub.elsevier.com/retrieve/pii/S1568494612001780`
29. Karunakaran, D., Chen, G., Zhang, M.: Parallel Multi-objective Job Shop Scheduling Using Genetic Programming. In: Artificial Life and Computational Intelligence: Second Australasian Conference, ACALCI 2016, Canberra, Australia, February 2-5, 2016, Proceedings, pp. 234–245. Springer (2016). DOI 10.1007/978-3-319-28270-1_20. URL `http://link.springer.com/10.1007/978-3-319-28270-1_20`
30. Keijzer, M., Babovic, V.: Dimensionally Aware Genetic Programming. Proceedings of the Genetic and Evolutionary Computation Conference **2**, 1069–1076 (1999). URL `http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GP-420.pdf`
31. Kofler, M., Wagner, S., Beham, A., Kronberger, G., Affenzeller, M.: Priority Rule Generation with a Genetic Algorithm to Minimize Sequence Dependent Setup Costs. In: Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, pp. 817–824. Springer (2009). DOI 10.1007/978-3-642-04772-5_105. URL `http://link.springer.com/10.1007/978-3-642-04772-5_105`

32. Koza, J.R.: Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Tech. rep., Stanford, CA, USA (1990)
33. Koza, J.R.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines **11**(3-4), 251–284 (2010). DOI 10.1007/s10710-010-9112-3. URL `http://link.springer.com/10.1007/s10710-010-9112-3`
34. Lee, J.H., Yu, J.M., Lee, D.H.: A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness. The International Journal of Advanced Manufacturing Technology **69**(9-12), 2081–2089 (2013). DOI 10.1007/s00170-013-5192-6. URL `http://link.springer.com/10.1007/s00170-013-5192-6`
35. Lee, Y.H., Bhaskaran, K., Pinedo, M.: A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE transactions **29**(1), 45–52 (1997)
36. Li, X., Olafsson, S.: Discovering Dispatching Rules Using Data Mining. Journal of Scheduling **8**(6), 515–527 (2005). DOI 10.1007/s10951-005-4781-0. URL `http://link.springer.com/10.1007/s10951-005-4781-0`
37. Lin, C.W., Lin, Y.K., Hsieh, H.T.: Ant colony optimization for unrelated parallel machine scheduling. The International Journal of Advanced Manufacturing Technology **67**(1-4), 35–45 (2013). DOI 10.1007/s00170-013-4766-7. URL `http://link.springer.com/10.1007/s00170-013-4766-7`
38. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing **59**(2), 107–131 (1999). DOI 10.1006/jpdc.1999.1581. URL `http://linkinghub.elsevier.com/retrieve/pii/S0743731599915812`
39. Masood, A., Mei, Y., Chen, G., Zhang, M.: Many-objective genetic programming for job-shop scheduling. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 209–216. IEEE (2016). DOI 10.1109/CEC.2016.7743797. URL `http://ieeexplore.ieee.org/document/7743797/`
40. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 1802, pp. 121–132 (2000). DOI 10.1007/978-3-540-46239-2_9
41. Miyashita, K.: Job-shop scheduling with genetic programming. In: Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation, GECCO'00, pp. 505–512. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000). URL `http://dl.acm.org/citation.cfm?id=2933718.2933809`
42. Moazenzadeh, R., Mohammadi, B., Shamshirband, S., wing Chau, K.: Coupling a firefly algorithm with support vector regression to predict evaporation in northern iran. Engineering Applications of Computational Fluid Mechanics **12**(1), 584–597 (2018). DOI 10.1080/19942060.2018.1482476
43. Morton, T.E., Pentico, D.W.: Heuristic Scheduling Systems. John Wiley And Sons, Inc. (1993)
44. Najafi, B., Ardabili, S.F., Shamshirband, S., wing Chau, K., Rabczuk, T.: Application of anns, anfis and rsm to estimating and optimizing the parameters that affect the yield and cost of biodiesel production. Engineering Applications of Computational Fluid Mechanics **12**(1), 611–624 (2018). DOI 10.1080/19942060.2018.1502688
45. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems **3**(1), 41–66 (2017). DOI 10.1007/s40747-017-0036-x. URL `http://link.springer.com/10.1007/s40747-017-0036-x`
46. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem. IEEE Transactions on Evolutionary Computation **17**(5), 621–639 (2013). DOI 10.1109/TEVC.2012.2227326. URL `http://ieeexplore.ieee.org/document/6353198/`
47. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Dynamic multi-objective job shop scheduling: A genetic programming approach. In: Automated Scheduling and Planning: From Theory to Practice, pp. 251–282. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-39304-4_10. URL `https://doi.org/10.1007/978-3-642-39304-4_10`
48. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Learning iterative dispatching rules for job shop scheduling with genetic programming. The International Journal of Advanced Manufacturing Technology **67**(1-4), 85–100 (2013). DOI 10.1007/s00170-013-4756-9. URL `http://link.springer.com/10.1007/s00170-013-4756-9`
49. Nguyen, S., Zhang, M., Tan, K.C.: Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 2781–2788. IEEE (2015). DOI 10.1109/CEC.2015.7257234. URL `http://ieeexplore.ieee.org/document/7257234/`

50. Nie, L., Gao, L., Li, P., Zhang, L.: Application of gene expression programming on dynamic job shop scheduling problem. In: Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 291–295. IEEE (2011). DOI 10.1109/CSCWD.2011.5960088. URL `http://ieeexplore.ieee.org/document/5960088/`
51. Park, J., Nguyen, S., Zhang, M., Johnston, M.: Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In: Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, pp. 92–104. Springer (2015). DOI 10.1007/978-3-319-16501-1_8. URL `https://doi.org/10.1007/978-3-319-16501-1_8`
52. Petrovic, S., Castro, E.: A genetic algorithm for radiotherapy pre-treatment scheduling. In: Applications of Evolutionary Computation: EvoApplications 2011, Torino, Italy, April 27-29, 2011, pp. 454–463. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). DOI 10.1007/978-3-642-20520-0_46. URL `https://doi.org/10.1007/978-3-642-20520-0_46`
53. Pfund, M., Fowler, J.W., Gadkari, A., Chen, Y.: Scheduling jobs on parallel machines with setup times and ready times. Computers & Industrial Engineering **54**(4), 764–782 (2008). DOI 10.1016/j.cie.2007.08.011. URL `http://linkinghub.elsevier.com/retrieve/pii/S036083520700229X`
54. Pfund, M., Fowler, J.W., Gadkari, A., Chen, Y.: Scheduling jobs on parallel machines with setup times and ready times. Computers & Industrial Engineering **54**(4), 764–782 (2008)
55. Pfund, M.E., Mason, S.J., Fowler, J.W.: Semiconductor Manufacturing Scheduling and Dispatching. In: Handbook of Production Scheduling, pp. 213–241. Kluwer, Boston (2006). DOI 10.1007/0-387-33117-4_9. URL `http://link.springer.com/10.1007/0-387-33117-4_9`
56. Picek, S., Cupic, M., Rotim, L.: A new cost function for evolution of s-boxes. Evolutionary Computation **24**(4), 695–718 (2016). DOI 10.1162/EVCO_a_00191. PMID: 27482748
57. Picek, S., Jakobovic, D., Miller, J.F., Batina, L., Cupic, M.: Cryptographic boolean functions: One output, many design criteria. Applied Soft Computing **40**, 635 – 653 (2016). DOI https://doi.org/10.1016/j.asoc.2015.10.066. URL `http://www.sciencedirect.com/science/article/pii/S1568494615007103`
58. Pinedo, M.L.: Scheduling: Theory, algorithms, and systems: Fourth edition, vol. 9781461423614. Springer US, Boston, MA (2012). DOI 10.1007/978-1-4614-2361-4. URL `http://link.springer.com/10.1007/978-1-4614-2361-4`
59. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk` (2008). URL `http://www.gp-field-guide.org.uk`
60. Sarin, S.C., Varadarajan, A., Wang, L.: A survey of dispatching rules for operational control in wafer fabrication. Production Planning & Control **22**(1), 4–24 (2011). DOI 10.1080/09537287.2010.490014. URL `http://www.tandfonline.com/doi/abs/10.1080/09537287.2010.490014`
61. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing: Issues and challenges. Journal of Grid Computing **14**(2), 217–264 (2016). DOI 10.1007/s10723-015-9359-2. URL `https://doi.org/10.1007/s10723-015-9359-2`
62. Đurasević, M., Jakobović, D.: Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. Genetic Programming and Evolvable Machines **19**(1), 53–92 (2018). DOI 10.1007/s10710-017-9302-3. URL `https://doi.org/10.1007/s10710-017-9302-3`
63. Đurasević, M., Jakobović, D.: Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. Genetic Programming and Evolvable Machines **19**(1), 9–51 (2018). DOI 10.1007/s10710-017-9310-3. URL `https://doi.org/10.1007/s10710-017-9310-3`
64. Đurasević, M., Jakobović, D.: Creating dispatching rules by simple ensemble combination. Journal of Heuristics **25**(6), 959–1013 (2019). DOI 10.1007/s10732-019-09416-x. URL `https://doi.org/10.1007%2Fs10732-019-09416-x`
65. Đurasević, M., Jakobović, D., Knežević, K.: Adaptive scheduling on unrelated machines with genetic programming. Applied Soft Computing **48**, 419–430 (2016). DOI 10.1016/j.asoc.2016.07.025. URL `http://linkinghub.elsevier.com/retrieve/pii/S1568494616303519`
66. chuan Wang, W., wing Chau, K., Qiu, L., bo Chen, Y.: Improving forecasting accuracy of medium and long-term runoff using artificial neural network based on eemd decomposition. Environmental Research **139**, 46 – 54 (2015). DOI https://doi.org/10.1016/j.envres.2015.02.002. URL `http://www.sciencedirect.com/science/article/pii/S0013935115000298`. Environmental Research on Hydrology and Water Resources
67. Yang-Kuei, L., Chi-Wei, L.: Dispatching rules for unrelated parallel machine scheduling with release dates. The International Journal of Advanced Manufacturing Technology **67**(1-4), 269–

279 (2013). DOI 10.1007/s00170-013-4773-8. URL http://link.springer.com/10.1007/s00170-013-4773-8

68. Yaseen, Z.M., Sulaiman, S.O., Deo, R.C., Chau, K.W.: An enhanced extreme learning machine model for river flow forecasting: State-of-the-art, practical applications in water resource engineering area and future research direction. Journal of Hydrology **569**, 387 – 408 (2019). DOI https://doi.org/10.1016/j.jhydrol.2018.11.069. URL http://www.sciencedirect.com/science/article/pii/S0022169418309545
69. Zhou, H., Feng, Y., Han, L.: The hybrid heuristic genetic algorithm for job shop scheduling. Computers & Industrial Engineering **40**(3), 191–200 (2001). DOI 10.1016/s0360-8352(01)00017-1. URL https://doi.org/10.1016/s0360-8352(01)00017-1

A Problem instance design

The processing times of jobs are generated from the interval $p_{ij} \in [0,100]$, by using either the uniform, normal (Gaussian), or quasi-bimodal distribution. The normal distribution uses a mean of 51 and a standard deviation value of 20. If a value is generated from outside of the allowed interval, then it is set to the nearest boundary. For each processing time it will be randomly selected which one of the three aforementioned distributions will be used for its generation. All job weights are generated uniformly from the interval $w_T \in < 0,1]$. A higher value of the weight denotes that the job has a higher priority. The release times of the jobs are generated by a uniform distribution from the interval $r_j \in \left[0, \frac{\hat{p}}{2}\right]$, where $\hat{p}$ is defined as $\hat{p} = \frac{\sum_{j=1}^{n}\sum_{i=1}^{m} p_{ij}}{m^2}$, and p_{ij} denotes the duration of job j on machine i, while m denotes the total number of machines. The due dates of the jobs are also defined using a uniform distribution from the interval $d_j \in \left[r_j + (\hat{p} - r_j) * \left(1 - T - \frac{R}{2}\right), r_j + (\hat{p} - r_j) * \left(1 - T + \frac{R}{2}\right)\right]$, where parameter T represents the due date tightness, while the parameter R represents the due date range. The due date range parameter defines the dispersion of the due date values, while the due date tightness adjusts the amount of jobs that will be late. Both of those parameters assumed values of 0.2, 0.4, 0.6, 0.8 and 1 in various combinations while generating the problem set. 1

The fitness of a single individual on the entire problem set is calculated so that for each problem instance in the set the individual is used to create a schedule independently from the other instances, and the *Twt* value for the obtained schedule is calculated. Since problem instances in the sets come in different sizes, the obtained *Twt* value for each instance is first normalised. The normalisation is performed by using the following expression: $f_i = \frac{Twt}{n\bar{w}\bar{p}}$, where n represents the number of jobs, $\bar{w}$ represents the average of all job weights, while $\bar{p}$ the average processing times of jobs across machines and is calculated as $\bar{p} = \frac{\hat{p}}{n}$. The normalised *Twt* values are then summed up, and the obtained value represents the total fitness value for he problem set.

B Parameter settings and tuning

In order to obtain the best possible results, the parameters for all tested methods have been thoroughly optimised. This appendix provides the information on how the parameter optimisation process was conducted.

B.1 GP parameters

The parameters which were optimised were the number of iterations, population size (sizes of 200, 500, 1000, and 2000), mutation probability (values of 0.1, 0.3, 0.5, 0.7, 0.9), different tree depths (depths of 3, 5, 7, 9, 11, and 13), various function nodes (basic arithmetic operators, minimum, maximum, if else branches, etc.), different crossover and mutation operators. In the cases of the function nodes and the genetic operators the same constructive and destructive heuristics have been applied as described for the selection of static terminal nodes. The parameters were optimised independently from each other. This means that all parameters except one were fixed to certain values, while several different values were tested for the remaining parameter. The parameter for which the best average values over 50 runs were obtained was selected and the next parameter was optimised. Table 10 lists all the parameter values which were in the end used by GP. As can be seen, several operators are used for crossover and mutation. This means that in each iteration of the algorithm one operator from the given set is randomly selected for crossover and one for mutation, and then those two selected operators are applied in the given iteration.

Table 10: Parameter values used by GP

Parameter	**Value**
Population size	1000
Termination criterion	80 000 iterations
Selection	steady state GP using tournament selection
Tournament size	3
Initialisation	*ramped half-and-half*
Mutation probability	0.3
Maximum tree depth	5
Crossover operators	subtree, uniform, context-preserving, size-fair
Mutation operators	subtree, Gauss, hoist, node complement, node replacement, permutation, shrink

B.2 GA parameters

The parameters used by the GA-PERM are represented in Table 11, while the parameters used by the GA-FP algorithm are shown in Table 12. The parameter values for these two algorithms were optimised similarly as it was performed for GP parameters. The values that were tested for the population size were 30, 100, 200, and 1000, whereas for the mutation probability the values 0.05, 0.1, 0.3, 0.5, and 0.7 were tested. Since in this case more than one operator is again defined for each representation, the procedure to select the operators is the same as described for GP.

Table 11: Parameter values used by GA-PERM

Parameter	**Value**
Population size	1000
Termination criterion	1 000 000 function evaluations
Selection	steady state GP using tournament selection
Tournament size	3
Mutation probability	0.7
Crossover operators	COSA, DPX, OBX, OPX, OX, OX2, PBX, PMX, SPX, ULX, UPMX, cyclic
Mutation operators	toggle, inverse, insert

Table 12: Parameter values used by GA-FP

Parameter	**Value**
Population size	30
Termination criterion	1 000 000 function evaluations
Selection	steady state GP using tournament selection
Tournament size	3
Mutation probability	0.3
Crossover operators	arithmetic, average, bga, blxalpha, discrete, flat, heuristic, local, onepoint, sbx
Mutation operators	simple