Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment

Marko Đurasević[a,*], Domagoj Jakobović[a]

[a] *Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia*

Abstract

Automatically designing new dispatching rules (DRs) by genetic programming has become an increasingly researched topic. Such an approach enables that DRs can be designed efficiently for various scheduling problems. Furthermore, most automatically designed DRs outperform existing manually designed DRs. Most research focused solely on designing priority functions that were used to determine the order in which jobs should be scheduled. However, in some scheduling environments, besides only determining the order of the jobs, one has to additionally determine the allocation of jobs to machines. For that purpose, a schedule generation scheme (SGS), which constructs the schedule, has to be applied. Until now the influence of different choices in the design of the SGS has not been extensively researched, which could lead to the application of a SGS which would obtain inferior results. The main goal of this paper is to perform an analysis of different SGS variants. For that purpose, three SGS variants are tested, two of which are proposed in this paper. They are tested in several variations which differ in details like whether they insert idle times in the schedule, or if they select the job with the highest or lowest priority values. The obtained results demonstrate that the automatically designed DRs with the tested SGS variants perform better than manually designed DRs, but also that there is a significant difference in the performance between the different SGS types and variants. The best DRs are analysed and show that the main reason that they performed well was due to the more sophisticated decisions they made when selecting the appropriate machine for a job. The results suggest that it is best to apply SGS variants which use the evolved priority functions to choose both the next job and the appropriate machine for that job.

Keywords: genetic programming, dispatching rules, schedule generation scheme, unrelated machines environment, hyper-heuristics, scheduling

*Corresponding author
[1] marko.durasevic@fer.hr
[2] domagoj.jakobovic@fer.hr

1. Introduction

Scheduling problems have been extensively researched in the literature due to their complexity and applicability in different areas. In general, scheduling is defined as the process in which a certain set of jobs (activities) has to be scheduled on a scarce set of machines (resources) to optimise some user-specified criteria [1]. Scheduling problems are quite widespread and have applications in different areas from the medical sector [2, 3], workforce scheduling [4], biopharmaceutical companies [5], and many others. Since most scheduling problems are NP-hard, the majority of researchers focused on either designing new problem-specific heuristic methods or applying existing metaheuristic methods. However, the type of methods that can be applied depends on the conditions under which the schedule has to be constructed. For example, if all the information about the schedule is known beforehand, then the entire schedule can first be constructed and then executed. In such static conditions, it is possible to use a great number of heuristic and metaheuristic methods to construct the schedules [6, 7, 8].

Metaheuristics have become one of the most popular methods used for solving not only scheduling but also other continuous or combinatorial optimisation problems. Among them the most prevalently used are some of the older and most acclaimed methods like genetic algorithms [9], particle swarm optimisation [10], ant colony optimisation [11]. However, in recent years a plethora of new metaheuristic methods, which usually mimic various natural phenomena or animal behaviour, have been designed by different researchers. Some of these newer metaheuristic algorithms include the sine-cosine optimisation method [12], grey wolf optimisation [13], whale inspired optimisation algorithms [14, 15], moth flame algorithm [16], salp chain optimisation methods [17, 18], and many others.

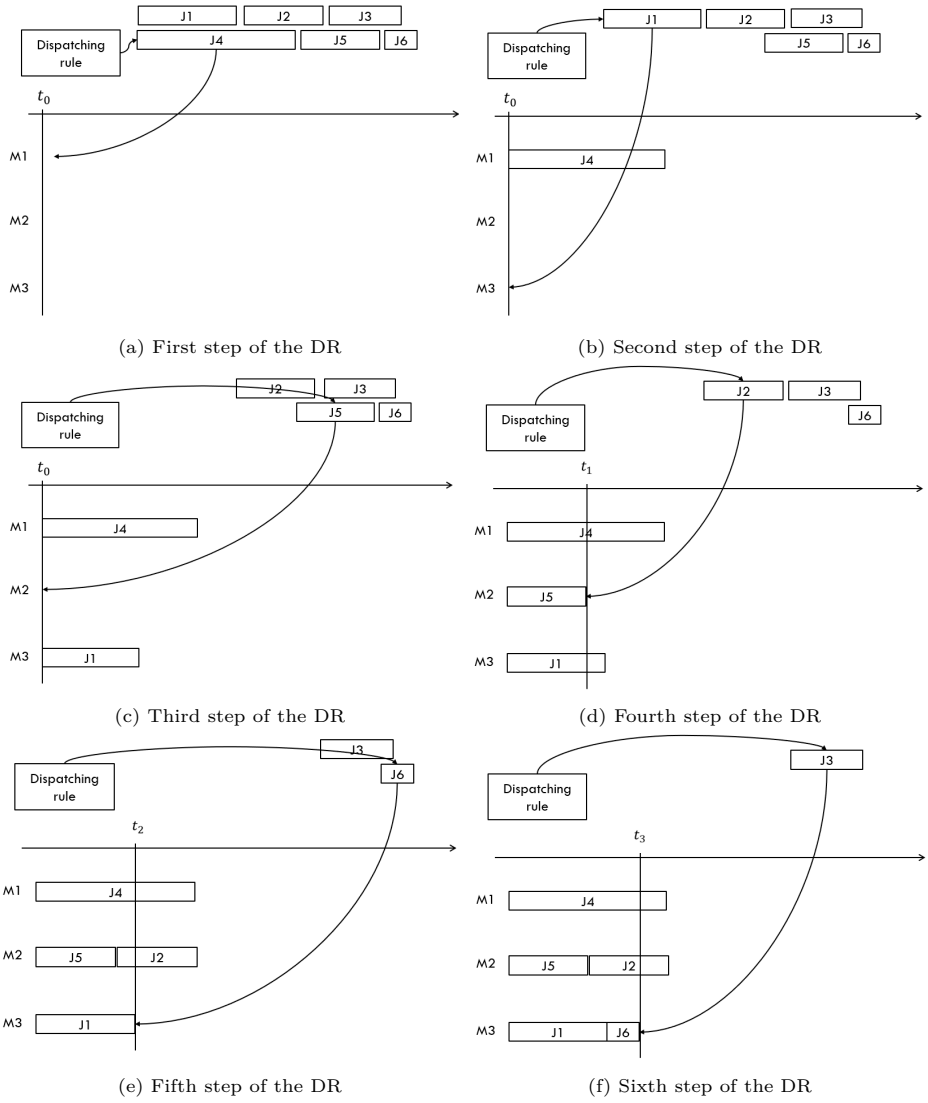
Aside from the aforementioned methods that search for the optimal solution of only one problem instance, methods like genetic programming (GP) [19], gene expression programming (GEP) [20], Cartesian genetic programming (CGP) [21] and others, have also been proposed. These methods differ in the sense that they can actually be used to evolve a strategy or heuristic that can solve a set of problems, rather than only one problem instance. GP has not only demonstrated the ability to obtain good solutions for many different problems [22, 23], but also that it is highly appropriate of being applied as a hyperheuristic method, i.e. a method the for automatic design of novel heuristics [24, 25, 26, 27, 28, 29]. In recent years, GP and similar methods were applied for automatic generation of heuristics for various problems like the capacitated arch routing problem [30, 31, 32], timetabling problems [33, 34, 35], vehicle routing problems [36], design of pipeline networks [37], and multidimensional knapsack problem [38].

In the case when scheduling is performed under dynamic conditions, the number of methods that can be used to construct the schedule is limited. This is because the characteristics of jobs become available during the execution of the system and are not known at the beginning. Thus, algorithms which search the solution space, as most metaheuristics do, cannot be applied to these problems without modifications, since they would not have all the necessary information

to construct valid solutions. Such scheduling problems are usually solved by using *constructive heuristics*, which iteratively construct the schedule during the execution of the system. Constructive heuristics are most often designed in the form of *dispatching rules* (DRs), which rank all the jobs by a certain priority calculated based on selected job and system parameters, and then schedule the job with the highest priority [39, 40, 41]. Therefore, DRs do not need all the information about the problem, but rather use only the information that is currently available to construct the schedule. However, DRs are problem-specific heuristics, which means that different DRs need to be designed for optimising different criteria and solving different kinds of scheduling problems. Designing DRs is not a trivial task, and a lot of domain knowledge and experience is necessary to design effective DRs. This is a serious drawback since it would mean that DRs would need to be manually designed for all possible scheduling problems that could appear, which is not feasible.

Figure 1 illustrates how a DR could schedule a certain set of jobs in a simple scheduling problem. In this example, six jobs need to be scheduled on three available machines starting from a certain time point t_0. The DR is invoked each time a machine is free and decides which job to schedule on which machine. At the start of the system, the DR selects and schedules one job on each of the machines as all three machines are free at the start. Based on the properties of jobs, the DR ranks all the jobs and selects the best one, which would in this case be job $J4$. The DR then schedules this job on one of the machines that are free. In this case it would schedule the job on machine $M1$ as denoted in Subfigure 1a. Since some machines are still available, the remaining jobs are ranked again and the best job is selected and scheduled on a free machine. This is illustrated in Subfigure 1b where job $J1$ is selected and scheduled on machine $M3$. Machine $M2$ is still available, therefore the entire procedure is repeated once again, after which job 5 is selected and scheduled on machine $M2$. At this point, there are no free machines and the system executes those jobs that are currently scheduled on the machines until a certain machine becomes available again. This happens at time moment t_1, when machine M_2 finishes executing its job. At that point the DR is invoked again, and it determines that job $J2$ should be executed on that machine. This is shown in Subfigure 1d. After this, the machines continue executing until machine $M3$ becomes available at time moment t_2. The entire procedure is repeated, and the DR selects and schedules job $J6$ on the machine $M3$, as show in Subfigure 1e. Finally, the DR is invoked one last time at moment t_3 when machine $M3$ becomes available once again. Since only job $J3$ is left the DR selects it and schedules it on machine $M3$, as illustrated in Subfigure 1f.

In recent years, a great deal of research focused on applying GP and similar methods for the automatic creation of DRs. In this approach, the DR is decomposed into two parts, a *priority function* (PF) used to calculate the priorities of jobs, and a *schedule generation scheme* (SGS) which uses the PF to construct the schedule. Such an approach of designing DRs has demonstrated to be quite efficient, not only because new DRs can be designed in a smaller amount of time, but also since the automatically designed DRs perform equally



(a) First step of the DR

(b) Second step of the DR

(c) Third step of the DR

(d) Fourth step of the DR

(e) Fifth step of the DR

(f) Sixth step of the DR

Figure 1: Scheduling strategy of a DR

well or even better than existing manually designed DRs. As a consequence, the topic became extensively researched [42, 43]. However, the entirety of research focused almost exclusively on designing PFs, while the SGS part did not receive a lot of attention. One of the main reasons for this is that most research focused on the job shop environment, in which the only decision that had to be made was the ordering of jobs. Therefore, the SGS only had to rank all the jobs by using the PF and then schedule them in that order. However, research in different scheduling environments has already demonstrated that even subtle

differences in the design of the SGS can lead to a significant difference in the obtained results [44]. Therefore, in environments in which ordering the jobs is not the sole decision that has to be made, the SGS also has a more significant influence on the quality of the constructed schedule. For example, in some environments, it is additionally required to determine to which machine the selected job will be allocated, like in the parallel machines environment. Therefore, the SGS does not only have to rank the jobs but also in a certain way determine how to allocate them to different machines. Thus, they are required to perform an additional decision that can have a significant effect on the quality of the constructed schedules. Even though DRs have already been generated for the unrelated machines environment, the design and choice of an appropriate SGS have still not received much attention, and thus it is not known whether better results could be obtained by performing different design choices.

Because of the above reasons, the goal of this paper is to analyse how different designs of the SGS can affect the performance of automatically generated DRs in the unrelated machines environment. Therefore, in addition to the existing SGS used in the literature for the unrelated machines environment, two additional schemes are proposed. One proposed SGS is modelled to more closely resemble the schemes used in manually designed DRs. By using this scheme it will be possible to determine if the existing manually designed DRs already present the best that can be obtained or whether it is possible to fine-tune the PF even further. The second SGS is modelled with the motivation to separate the choice of selecting the next job and the machine it will execute on into two independent PFs, which was not the case until now. This division should hopefully lead to the design of more interpretable PFs. Furthermore, the paper also analyses the influence of several design choices, like selecting whether to schedule jobs with the smallest or largest priority value, or whether to take the absolute value of the priorities, which have also not been considered until now. The influence of using an SGS which allows idle times or not will also be analysed, as in the unrelated machines environment this decision largely affects the construction process of the schedule. Although it is intuitively expected that using idle times should lead to better schedules, we were unable to find any research in designing DRs to support this claim. A detailed analysis of the evolved DRs is also performed, both based on the interpretability of the evolved PFs, and the decision process performed by the different DRs. The analysis shows that the main reason for the superior performance of the automatically designed DRs lies in their ability to evolve the strategy by which to allocate jobs on machines. This observation can influence how DRs are designed, especially the manually designed ones, as it demonstrates the significance of the decision of allocating jobs to machine, and the limitation of the scheme used by manually designed DRs. Therefore, instead of putting most focus on designing the PF which ranks the jobs, equal attention should be placed also on designing better schemes for allocating jobs on machines. The contributions of the paper can be summarised through the following points:

1. Comparison of three SGS variants for the unrelated machines environment,

two of which are proposed in this paper
2. Analysis of the influence of selecting jobs with highest or lowest priorities, taking absolute values of priorities, and inserting idle times in the schedule
3. A detailed analysis of the scheduling decisions performed by the SGS variants showing that a superior performance can be obtained by those SGS variants which also use a PF for the allocation of jobs to machines

The rest of the paper is organised as follows. Section 2 gives a short overview of the existing researched performed in the area of automatic design of DRs by GP. Section 3 provides background information about scheduling in the unrelated machines environment and designing new DRs with GP. The SGS types and their variants are described in Section 4. Section 5 describes the experimental design and outlines the selected parameter values for performing the experiments. The results obtained from the performed experiments are presented in Section 6. Section 7 provides an analysis of several PFs that were designed for different SGS variants. The discussion about the obtained results and the performed analysis is given in Section 8. Finally, the conclusion and directions for further research are given in Section 9.

2. Literature overview

The topic of automatically designing DRs for various scheduling problems has become quite extensively researched in the literature. In the first attempts, GP was applied to generate DRs for the single machine and job shop environments [45, 46]. Even in those first studies, it became evident that GP holds great potential for the automated design of DRs. Further research on this area has focused on many different topics, mostly to improve the results of the generated DRs, or to apply them to new and specialised scheduling problems, like the flexible job shop environment [47]. In subsequent studies scheduling problems with various additional criteria like setup times [48], precedence constraints [48], and machine breakdowns [49, 50] have also been considered. Furthermore, a lot of research has focused on using different methods for constructing the DRs like gene expression programming [51, 52], artificial neural networks [53], and different rule representations in GP [54]. An alternative representation for DRs was proposed in [55]. The proposed representation can effectively select the relevant attributes, which leads to simpler and more interpretable expressions. Generating DRs that are appropriate for generating schedules which optimise several criteria simultaneously has also been researched for the job shop [56, 57, 58, 59], and unrelated machines environments [60]. The great potential of GP for automatic construction of entire scheduling policies was demonstrated in [61], where GP was used to design due date assignment rules. This study demonstrated that these automatically designed scheduling policies perform better than many of the tested manually designed ones. In [62] the authors proposed a feature selection method which obtains better feature subsets and determines the most important features which should be used when constructing the DR. In [63] the authors propose an adaptive search strategy which uses the information on the

frequency of features obtained in good DRs to guide the search to promising areas in the search space. Feature construction was also investigated in [64] to improve the performance of the designed rules. The ability to evolve both the sequencing and routing rules simultaneously has been investigated in [65], where a multi-tree representation was applied for that purpose. The proposed representation demonstrated that it performs better than the cooperative co-evolution procedure when simultaneously evolving both sequencing and routing rules. Another representation that considers the contributions of the different features and combines them in a more sophisticated way is studied in [66].

Another researched area is to perform scheduling under uncertainties [67, 68, 69] which means that some parameters, like processing times, are not completely known when creating schedules, but rather just estimated and their values become known only after the schedule has completed with its execution. In [70] the authors propose a two-stage approach for scheduling in a work centre. In this study, GP is used to design novel DRs, while an evolutionary algorithm is applied to assign the evolved rules to different work centres. To increase the performance of the generated DRs, several studies used various ensemble learning methods with great success. Some of the methods that were applied or proposed for constructing ensembles of DRs include cooperative cooevolution [71], NELLI-GP [72], SEC [73], BoostGP and BagGP [41]. Additionally, in [74] the authors have investigated several ensemble combination schemes, and demonstrate that the linear combination scheme obtains the best results. The generation of DRs has proven to be a computationally intensive process, due to the reason that a lot of problem instances have to be used for training to ensure that rules with good generalisation properties are obtained. As a consequence various surrogate models [75, 76], or directions for constructing smaller problem instances [77] have been proposed to improve the evolution speed and obtain better results with the same or smaller computational effort. Since the underlying evolution of DRs is quite difficult to follow, new methods for better visualisation and understanding of the evolutionary process have been proposed and applied for the job shop scheduling problem [78]. The work in the aforementioned study is expanded in [79] by using surrogate models to allow decision-makers to interact and guide the evolution process. In [80] the authors consider a method that guides GP to unexplored areas by using the growing neural gas method. To additionally improve the quality of the obtained DRs, in [81] a new strategy is proposed for selecting subtrees in crossover and mutation, where the probability of selecting the subtree is based on its importance and the type of the operators. The topic of selecting the appropriate problem instances to evolve DRs is considered in [82]. The study proposes an active sampling method that selects good instances during the evolutionary process. Another recent research direction is directed towards evolving DRs for the single machine environment with variable capacity [83]. Since the initial results demonstrated that for this variant of problem automatically evolved DRs obtained better results than manually designed rules, the authors further improved the results by using ensemble methods to construct DRs [84, 85]. In [86] it was demonstrated that automatically designed DRs can also be applied to initialise the starting

population of a genetic algorithm, which significantly improves its performance.

Although it is evident that most of the research has focused predominantly on the generation of PFs, in some studies new SGS variants have been defined, although usually with the intent of adapting DRs to some special conditions. One of the earliest studies where some kind of SGS was used was in [87], where a GP method that evolved three separate expressions was used. In this case, one expression represented a decision function that decided which of the other two evolved PFs to use for scheduling based on the current system parameters. Therefore, the general SGS had to be slightly adapted to conform to this new scheduling logic. Another example where a new SGS for DRs was defined was when trying to generate iterative DRs that are specialised for static scheduling conditions [88]. In this SGS the schedule is constructed several times and each time the schedule is reconstructed additional information is used from previously constructed schedules. An area in which the design of different SGS variants was studied in more detail is the order acceptance and scheduling problem (OAS) in which it is not only required to schedule the job, but also to decide which jobs to select for scheduling, since now there is a possibility that jobs do not have to be accepted if deemed that they could not bring enough profit [89, 90]. Since in this problem it is required to perform two decisions, it was required to specify an appropriate SGS. In these studies, several SGS variants were tested, most notably a variant in which a single PF performs both decisions, and a variant where the acceptance decision is performed by one PF and the scheduling decision is performed by another PF. Although a bit surprising, the results showed that the variant in which the PF performs both decisions obtains better results. In [44] an analysis of different SGS variants for the resource constrained scheduling problem was performed. In this study, it was also demonstrated that the selection of different SGS variants can influence the performance of the generated DRs. These few studies demonstrate the importance of designing a good SGS, since the have a significant influence on the results.

Several studies focused on generating DRs for the unrelated and parallel machines environment [91, 92, 60, 41, 73], in which a single SGS was used, but none of them has investigated the design of appropriate SGS variants and their influence on the performance of the generated DRs. Therefore, there still exists a large gap in this area, and many open questions, which have not been answered, still remain. Furthermore, the design choices of the existing SGS have not been explained and thus the question remains whether different SGS types or variants could possibly lead to a better performance of the automatically designed DRs. Since the studies dealing with SGS variants for other scheduling problems demonstrated that different choices in the design of SGS variants can have a significant influence on the obtained results, it is likely that the situation would be similar for the unrelated machines environment as well.

A more detailed overview of the research that has been done in the area of automated design of DRs is presented in [42, 43, 93].

3. Background

3.1. Unrelated machines environment

The unrelated machines environment is a scheduling environment in which several machines are executing in parallel that can process jobs. This environment belongs to the class of single stage environments, which means that each job needs to be executed on a single machine to be completed. The unrelated machines environment is a specialised type of the parallel machine environment, in which each job has a different execution time on each of the machines. These execution times are completely provisional, which means that it is not possible to determine relations between different machines (for example that one machine executes all of the jobs two times faster than another machine). Scheduling in the unrelated machines environment can be found in many practical real-world examples, such as in multiprocessor computers, landing lanes in airports, semiconductor manufacturing, painting and plastic industries [94, 95, 96, 1].

The unrelated machines scheduling problem consists of n jobs that have to be allocated to one of the m available machines. In the standard definition of the problem each machine can execute only a single job at each moment in time, and once a machine starts executing a job it has to execute it until the end before it can start executing another job. A job in the problem is usually denoted with the index j, while a concrete machine is denoted with the index i. For each job in the problem instance several properties are defined, which are used to calculate values of different scheduling criteria. The first and most important property is the processing time of job j on machine i which is denoted as p_{ij}. Jobs are usually not available from the start of system execution but are rather released during the execution of the system. The release times of jobs are denoted as r_j. Furthermore, not all jobs have the same importance. Usually, some jobs are more important since they will either lead to a larger profit if finished sooner, or cause a larger penalty if not completed on time. This is modelled by defining a weight w_j for each job, which has a value between 0 and 1. A larger value of this parameter denotes that the job has higher importance. Finally, jobs usually also have a due date d_j defined, which is the time until jobs should be completed or otherwise a certain penalty will be caused. Based on the aforementioned properties the schedule is constructed in a way that a certain criterion is optimised, which can be the total duration of the schedule, the number of late jobs, the maximum lateness of a job, and similar criteria. This paper will focus on minimising the *total weighted tardiness* (TWT) criterion, which is defined as $TWT = \sum_j w_j T_j$, where T_j denotes the tardiness of job j. The tardiness of job j is defined as $T_j = \max\{C_j - d_j, 0\}$ where C_j denotes the completion time of job j. In this criterion jobs that are completed after their due date incur a certain penalty or loss, which has to be minimised. The considered problem can also be more formally defined using a mixed integer programming formulation [97]. In this formulation the time horizon is discretized into time periods $1, \ldots, l$, where l denotes the largest completion time of all jobs. The binary variable χ_{ij}^t is equal to 1 if job $j \in J$ starts executing on machine $i \in M$ at time t, otherwise the variable is equal to 0.

$$\min \quad \sum_j w_j T_j \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in M} \sum_{t=0}^{l} \chi_{ij}^t = 1 \qquad j \in J \tag{2}$$

$$\sum_{j \in J} \sum_{h=\max(0, t-p_{ij})}^{i-1} \chi_{ij}^h \leq 1 \qquad i \in M, t = 1, \ldots, l \tag{3}$$

$$\sum_{i \in M} \sum_{t=0}^{r_j - 1} \chi_{ij}^t = 0 \qquad j \in J \tag{4}$$

$$C_j \geq \sum_{i \in M} \sum_{t=0}^{l-1} (t + p_{ij}) \chi_{ij}^t \qquad j \in J \tag{5}$$

$$T_j \geq C_j - d_j \qquad j \in J \tag{6}$$

Constraint (2) ensures that each job starts processing on only one machine at only one point in time. Constraint (3) allows for only a single job to be processed at any time on any of the machines. That no job starts executing prior its release time is ensured by constraint (4). Constraints (5) and (6) represent the constraints imposed on the completion time C_j and tardiness T_j of job j.

The final part which needs to be specified for scheduling problems is whether they are solved under static or dynamic conditions. This paper focuses on dynamic conditions, which means that jobs are released into the system during its execution and that no job properties are known before the job is released into the system (not even the release time). Under such conditions, DRs are the most appropriate methods to solve scheduling problems. As described in the introduction, DRs consist of a PF and an SGS. For example, a PF can be defined as $\frac{1}{p_{ij}}$, which would mean that the priorities of jobs are inversely proportional to their processing times. The SGS then needs to define how it will schedule jobs to machines by using the PF. For example, the SGS can be defined so that it will select the job with the highest priority value and schedule it. However, the SGS still needs to determine to which machine the selected job will be allocated. One way to determine this would be to allocate the job to the machine on which it would be completed the soonest. As can be seen from the example, both the PF and the SGS are of great importance when constructing the schedule. Thus, both parts of the DR need to be designed carefully to ensure that it will perform well.

3.2. Designing dispatching rules with genetic programming

Designing a DR automatically can be viewed as a hyper-heuristic optimisation problem in which, by using a metaheuristic method, a heuristic that can

be used to solve various scheduling problems can be obtained. Formally, this problem could be defined as [98]:

$$F(h^*|h* \rightarrow s^*, h^* \in H) \leftarrow f(s^*, s^* \in S) = \min(f(s), s \in S),$$

where h represents a heuristic, in this case a DR, from the heuristic space H, s represents a solution, in this case a schedule, in the solution space S for some scheduling problem under consideration, and $f(s)$ the function for evaluating the quality of solution s which is mapped to the function $F(h)$ which evaluates the quality of the obtained heuristic. The quality of the heuristic h is evaluated by executing it on a certain problem and generating the solution s which is then evaluated using the function $f(s)$. In other words, we are searching for a DR which obtains the best results for the considered problem. As denoted previously, the DR consist out of a PF and an SGS. Leaving GP to evolve the entire DR would be too difficult, as the SGS needs to have a certain structure to create feasible schedules. Therefore, this part is designed manually, and the evolution of PFs is left to GP. This entire process is also illustrated in Figure 2. The flowchart shows that the DR consists out of the SGS and PF. As illustrated, the GP is used to automatically design a new PF which is used in synergy with a manually designed SGS to construct a schedule for a given scheduling problem. Based on the input parameters of the scheduling problem, which represent various job and machine properties (like the processing times or due dates), the PF calculates the priorities for jobs and the DR constructs the schedule for that problem.

For GP to evolve new PFs, the set of primitive nodes needs to be defined. The nodes which will be used depend largely on the machine environment, optimised criteria, and additional constraints imposed on the schedule. Therefore, the set of primitive nodes has to be adapted depending on the concrete problem. For the unrelated machine environment the terminal set denoted in Table 1 has been found to work best in previous studies and will also be used in this study [92]. The terminals are divided into three groups depending on whether their values will be different for various jobs, machines, or the combination of both. The set of function nodes is even smaller consisting only of the addition, subtraction, multiplication, protected division (returns 1 if the denominator is close to 0), and positive operator ($POS(a) = \max(a, 0)$). With these primitive nodes, GP constructs the PF of the DR. The PF evolved by GP is used by a manually designed SGS to construct the schedule for a certain problem instance.

The benefit of this approach is that DRs can be designed for various criteria, and also that it is possible to design DRs that are specifically tailored for solving a certain kind of scheduling problem. As a consequence, they can perform much better than manually designed DRs, which are often tailored to be very general. However, this methodology also has certain drawbacks. To design new DRs it is required to evolve a new PF, which can take some time depending on the algorithm parameters and training instances. Furthermore, the quality of the evolved PFs depends on the problems that were used to evolve them. Thus, if the evolved DRs are used for solving problems that differ significantly from

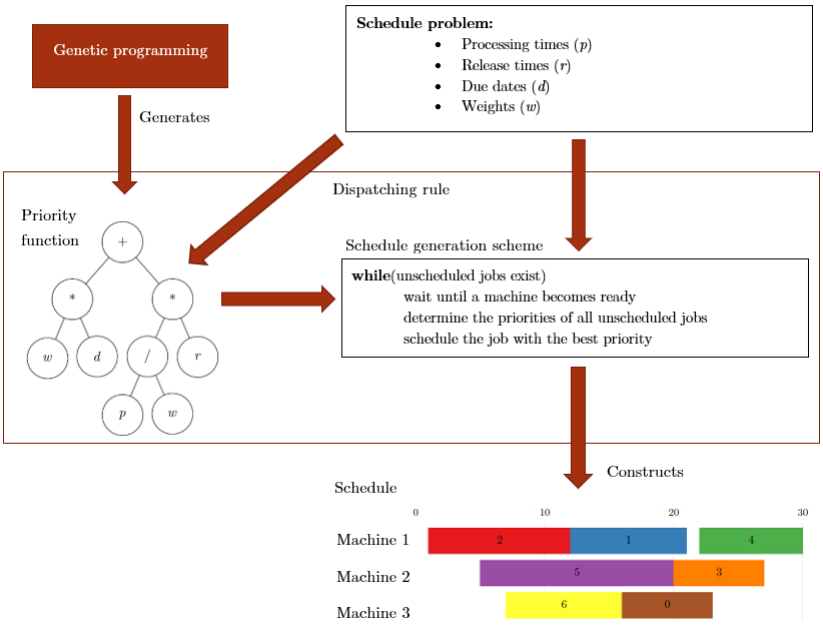


Figure 2: Structure of automatically generated DRs

those on which the GP trained them, the DRs will likely obtain poor results. An additional thing that needs to be outlined is that the approach needs to be adapted when additional different conditions are considered, by providing more terminal nodes to GP and also adapting the SGS to prevent the DR from creating unfeasible schedules.

Aside from GP, there are also other alternatives that can be used to design PFs. For example, one could simply use a weighted sum of different job and system properties, and then by using a genetic algorithm, or other metaheuristic methods, evolve the weights for each property [53]. However, this method has demonstrated to be quite limited and when compared to others it did not achieve competitive results. Another alternative is to use neural networks to design PFs [53]. Although such DRs can achieve almost equal performance as those designed by GP, they suffer from a different problem. Namely, such PFs are not interpretable, and thus no knowledge can be extracted from them, and neither can their decision process be explained. Another model that was used for generating PFs was GEP [99, 100], which is similar to GP and achieved competitive results when compared to it.

Since the PF is used in synergy with the SGS to represent the DR, this means that the design choices that are made in the SGS will also affect the evolutionary process of GP. Even though GP evolves the PF and tries to obtain those which are close to optimal, the quality of the entire DR is also restricted

Table 1: Terminal nodes used by GP

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

by the SGS. Therefore, even if GP obtains the optimal PF, the entire DR might still perform poorly due to a bad SGS design. As the SGS is designed manually, the designer needs to ensure that the choices performed in the design will not negatively affect the entire DR. Instead of focusing only on techniques which improve the quality of the evolved PF, it is also required to improve the SGS as much as possible. Therefore, one could say that the SGS also has to be optimised to a certain degree, albeit manually by considering the influences of different design decisions in the SGS. Only when both parts of the DRs, namely the SGS and PF, are optimised can we be sure that a good DR was obtained, and not just a suboptimal DR limited by the poor design of either the PF or SGS. The next section outlines different SGS variants that can be used for the unrelated scheduling problem and outlines various decision which can be made during the design process.

4. Schedule generation schemes

This section gives details about some variants of SGS that can be defined for the unrelated machines environment.

4.1. Variants of SGS

For the unrelated machines environment it is possible to use various SGS variants, depending on how the schedule should be constructed by using the generated PF. The first type, denoted as *simple* SGS, is shown in Algorithm 1. The reasoning behind this SGS is to use a single PF to determine both the sequence of jobs and their allocation to machines. The evolved PF is first

applied to determine the best machine for each available job. This is performed so that for each job the machine with the best PF value is associated with it. Then, out of all job-machine associations, the one for which the PF achieves the best value is selected and the job is allocated to its associated machine. In this way, both the sequencing and scheduling decisions are performed by a single PF, which is a benefit since only one PF needs to be evolved. On the other hand, it might be difficult for GP to design a PF that performs both of these decisions at the same time.

Algorithm 1 Simple SGS

```
1: while unscheduled jobs are available do
2:     Wait until at least one job and one machine are available
3:     for all jobs where r_j < currentTime and each machine i in m
4:         Obtain the priority π_ij of scheduling job j on machine i
5:     end for
6:     for all available jobs
7:         Determine the best machine (the one with the best π_ij value)
8:     end for
9:     while jobs whose best machine is available exist do
10:        Determine the best priority of all such jobs
11:        Schedule the job with best priority
12:    end while
13: end while
```

Most of the manually designed DRs in the unrelated machines environment do not use a single PF to also determine the allocation of jobs to machines. The main reason for this is that PFs could easily cause unbalanced schedules where most jobs would be allocated to only a few machines. To avoid such a problem, the allocation of jobs to machines is usually performed independently of the PF. The most common way is to allocate the job on the machine on which the job would be completed the soonest (taking into account the time when the machine will become available). In that way, it is easier to ensure that jobs are allocated more evenly across all the machines. Based on that idea, an SGS that selects jobs based on the PF and allocates them using some kind of a predefined heuristic is proposed, and will be called the *heuristic* SGS. Algorithm 2 represents such an SGS. In this SGS, the priority value is calculated for all released jobs, and the one with the best priority value is selected. For the selected job the completion times on each of the machines are calculated, as the sum between the processing time of job j on machine i and the maximum value between the release time of the job and the time when the selected machine becomes available. The job is allocated to the machine on which it has the smallest completion time. In that way, GP is tasked only with obtaining a good PF for selecting the jobs, while their allocation to machines is left for the SGS to decide.

It is also important to note that there are two variants in which the priority

values can be calculated. The first one is that the PF calculates the priorities for jobs by also taking into account the machines, which means that it uses terminal nodes that depend on the machine on which the job would be scheduled. The SGS variant that works in that way will be denoted as *heuristic1*. However, these priority values are only used to select the job, and the allocation to the machine is still done based on the minimum completion time of the job. The motivation for this variant is to see whether considering information about the machines can lead the DR to select the appropriate job since in that case, it can have a better overview of the entire problem.

Since the PF is not used to make any decisions about the allocation to a concrete machine, it could be possible that using information about machines could be misleading. Therefore, the second variant of this SGS uses the PF to calculate priority values for jobs based solely on job properties, meaning that the PF does not use any terminals that depend on a concrete machine like the *pt*, *MR*, and *SL* nodes. However, since the *SL* node which represents the slack of the job is quite important, it has been redefined in this case as $\max(d_j - pavg - time, 0)$. In that way, it is at least possible to approximate the slack time of the job based the average processing time of the job on all machines. The motivation for this variant is to analyse whether constraining the information that is used only to the information about jobs can improve the performance of the generated DRs. This SGS variant will be denoted as *heuristic2*.

Algorithm 2 Heuristic SGS

```
while unscheduled jobs are available do
    Wait until at least one job and one machine are available
    for all jobs where r_j < currentTime
        Calculate priority π_j for job j
    end for
    Determine job j with the best π_j value
    Determine the machine on which job j would be completed the soonest
    Schedule job j on the selected machine
end while
```

The previous SGS variants use only a single PF, which is applied just for selecting the job that should be scheduled (in the heuristic SGS), or for both selecting the job and the machine on which it should be scheduled (in the simple SGS). Another way to create the schedule would be to use two independent PFs, one to determine which job should be selected, and the other for allocating the selected job to a concrete machine. Algorithm 3 represents the SGS which uses two priority functions to construct the schedule. The proposed SGS first calculates the priority values for all available jobs by using one PF. It selects the job with the best priority value and then by using another PF it calculates priority values for all machines and schedules the selected job on the machine with the best priority value. In this case it is evident that GP has to evolve

two PFs. However, each PF is only tasked with selecting either only jobs or machines, which could possibly lead to better results and maybe simpler and more interpretable PF. This SGS variant will be denoted as *twotrees*. Since this SGS uses two PFs, for each one of them it is required to define which terminals are used to construct them. The PF used to select the job uses the same terminals as the *heuristic2* SGS, since it does not consider the machines in the calculation of priorities. However, the PF that is used to determine the appropriate machine for the selected job has to take into account both the information about the job and the machine, which means that all the terminals will be used just like for the PFs generated for the *simple* and *heuristic1* SGS variants.

Algorithm 3 Twotrees SGS

1: **while** unscheduled jobs are available **do**
2: Wait until at least one job and one machine are available
3: **for** all jobs where $r_j < currentTime$
4: Calculate the priority π_j of job j by using the first PF
5: **end for**
6: Determine job j with the best priority value
7: **for** each machine i in m
8: Calculate the priority π_{ij} for allocating job j on machine i, by using the second PF
9: **end for**
10: Schedule job j on machine i
11: **end while**

4.2. Priority calculation

In the previous section, only the main logic of each SGS was defined. However, there are still details concerning the PFs which were not specified. One such detail is in which way the SGS variants will use the PF to select jobs and machines. The value of the PF is used to determine the priorities of individual jobs. This means that it is possible to define two variants, one in which the job or machine with the highest PF value has the largest priority ("best priority" in the algorithm denotes the largest value), and one in which the job and machine with the lowest PF value have the largest priority ("best priority" in the algorithm denotes the lowest value). Therefore, the PF value can be interpreted in different ways to determine which job or machine has the highest priority during scheduling and has to be scheduled first. Although it might seem strange to prioritise a job with the lowest priority value, the reason why this is examined is to analyse whether any of the SGS variants has a bias towards any of the two ways of prioritising jobs and machines. Knowing if such a bias exists could then help to improve the performance of GP by including this knowledge into the algorithm. Most often researchers opted for the variant in which the element

with the highest priority value was selected. In parts of the previous three algorithms, it was only specified that the job or machine with the "best" priority should be selected. Thus, in all those places it is possible to either select the element with the highest or lowest priority value. This means that in the case of the simple and twotrees SGS one element may be selected based on the lowest priority, while the other is selected based on the highest priority of the same PF. Therefore, these two variants will also be tested in the experiments. However, it must be noted that this choice can also be delegated to GP by introducing a function node that would simply perform a negation of a subtree.

Finally, one additional thing which can be considered with the PFs is whether it makes sense to just use their absolute values. In that way, the GP could completely ignore the sign of the expression and focus on evolving an expression which achieves either the highest or the lowest value for the jobs it should schedule first. This variant of PFs will also be tested to see whether it affects the results.

4.3. Idle times

An important part in the construction of schedules and thus in each SGS is the choice whether it is allowed to insert idle times on machines, even though there are jobs in the system waiting to be scheduled. Although it might seem that inserting idle times in the schedule might have a negative effect on the optimised criteria, not having idle times can have a much larger impact, especially in the unrelated machines environment. The reason for this is that if no idle times would be used, it would mean that jobs have to be immediately scheduled on machines. However, at later stages, when only a small number of machines are free, this can have a negative effect on the schedule. The reason for this is that a job could be scheduled on a machine which is completely inappropriate for executing it. This would mean that the job would be executed for a longer period, which would, in turn, affect the entirety of the remaining schedule. However, if idle times could be inserted, then it could be determined that the machine is not appropriate for scheduling the job, and the machine could remain idle until another job enters the system. In that way, the DR can try to schedule jobs on the machines which are more appropriate for them.

All the proposed SGS variants are by default defined to introduce idle times similarly. For example, when a job is selected, the priority or minimum completion time of scheduling that job on all machines, whether they are free or not, is calculated. If the machine with the best priority value or minimum completion time is free, then the job is scheduled immediately on it. However, if the selected machine is currently executing another job, then the job is not scheduled to any machine, and its scheduling will be postponed to the next decision point. In that way, it is possible to postpone scheduling certain jobs if it is determined that the appropriate machine is not free. To analyse the importance of idle times, all SGS variants are also adapted to create schedules without using idle times. In this case the logic of the SGS remains the same, except for the point that now they would only consider free machines when calculating the priority values or minimum completion times. In that way, if there are free machines,

Table 2: Time and space complexity of the proposed SGS methods

	Simple	Heuristic1	Heuristic2	Twotrees
Time (one iteration)	$\mathcal{O}(nm)$	$\mathcal{O}(nm)$	$\mathcal{O}(n)$	$\mathcal{O}(n+m)$
Time (total)	$\mathcal{O}(n^2m)$	$\mathcal{O}(n^2m)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2+nm)$
Space (naive)	$\mathcal{O}(nm)$	$\mathcal{O}(nm)$	$\mathcal{O}(n)$	$\mathcal{O}(n+m)$
Space (best)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

the job will necessarily be scheduled on one of those machines. Both variants of SGS will be tested in the experiments.

4.4. Time and space complexity

The time and space complexity of the applied SGS variants is denoted in Table 2 using the big O notation. Only the worst-case scenario is considered in which it is presumed that all jobs are available from the start of execution. In one iteration it can be presumed that n jobs need to be allocated on m machines. The time complexity of the SGS is mostly tied to the number of evaluations of the PF that have to be performed in each iteration. Since the complexity of the PF is always the same, its complexity can be considered constant for each evaluation, thus its complexity is $\mathcal{O}(1)$. The simple and heuristic1 SGS variants calculate the priority for each combination of a job and machine. Therefore they will perform nm evaluations of the PF function, and thus the time complexity of one iteration is $\mathcal{O}(nm)$. The heuristics2 SGS needs to evaluate the PF only for each job, thus making the complexity of each iteration equal to $\mathcal{O}(n)$. Finally, the twotrees SGS first calculates the PF for each job, and then for the job with the best priority it calculates the PF for each machine, which in total makes the complexity equal to $\mathcal{O}(n+m)$.

The aforementioned complexities are only for one iteration of the SGS. It is impossible to estimate the number of iterations the DR will perform since in some iterations no job might be scheduled, while in others more than one job can be scheduled (especially in the beginning when the system is empty). However, suppose that one job is scheduled per iteration then the number of iterations would be equal to n. It is possible to calculate the total number of PF evaluations for jobs during the execution for the entire system as $n + n - 1 + n - 2 + \ldots + 2 + 1$, as in each subsequent iteration there would be one job less to schedule. This sum is equal to $\frac{n(n+1)}{2}$, or rather to $\mathcal{O}(n^2)$ in the big O notation. Therefore, it can be seen that the complexity of all methods would be larger by the order of n in comparison the complexity in one iteration.

Regarding the space complexity, it depends heavily on the implementation details. In the naive implementation, which would also represent the worst-case scenario, each calculation of the PF in an iteration would be stored separately. Thus, the space required would be equal to the number of calculations that are required in one iteration, which is then equal to the time complexity of one iteration. However, it is never required to keep track of all values obtained by

the PF, since only the one with the best PF value is scheduled. Therefore, it is only required to store the best PF value so far, and the currently calculated PF so that they can be compared and the best one updated if necessary. In that way the space complexity can be considered constant since only a constant number of values need to be stored regardless of the number of jobs and machines.

5. Parameters and experimental setup

For the PF design, the standard GP algorithm based on the steady state tournament selection process is used. The outline of this algorithm is denoted by the flowchart in Figure 3. At the start, the algorithm randomly initialises the population of individuals. After that the algorithm randomly selects 3 individuals from the population which form the tournament. The two best individuals in the tournament are selected to participate in the crossover. Since several crossover operators can be available, one of them is randomly selected in each iteration and applied on the two individuals to generate the child individual. The child individual is mutated by using one randomly selected mutation operator from the set of available operators. The mutation operator is performed with the given probability, thus it will not be performed in each iteration. After the genetic operators were performed, the child individual is evaluated. The worst individual that was selected in the tournament is removed from the population, and the child individual is inserted in the population. The algorithm checks whether a predefined number of iterations were performed, and if not the entire process of selection, crossover and mutation is repeated. On the other hand, if the predefined number of function iterations were performed, then the algorithm stops with execution and returns the best evolved individual as the final solution.

The parameters of GP used for evolving the DRs are given in Table 3. The values for these parameters were set during preliminary experiments on a separate problem instance set [92]. The parameter fine tuning was done in a way that for each parameter several values were tested, while the values of the remaining parameters were fixed. The population was set to 1000 individuals since the fine tuning process demonstrated that by using smaller population sizes the results deteriorated, whereas using a larger population size did not improve the overall results and as such it does not bring any advantage to use a larger population. The mutation probability was set to 0.3, as for this value GP obtained the best results. For both, smaller and larger mutation values that were tested, the results slightly deteriorated. The termination criterion was optimised by setting a large number of iterations and monitoring the value of the fitness function of the best individual on a separate validation set. After around 80 000 iterations it was noticed that the fitness of the best individual started to stagnate on the validation set, and even started to worsen after a while. Therefore, it was concluded that at around this point the algorithm started to overfit on the training set. Therefore it was decided to set the termination criterion to 80 000 iterations, which gives the algorithm enough time to obtain good solutions, but prevents it from overfitting. The tree depth parameter is one of the most

Table 3: Parameters for GP

Parameter name	Parameter value
Population size	1000 individuals
Termination criterion	80 000 iterations
Selection	Steady state tournament GP
Tournament size	Three individuals
Initialisation	Ramped half-and-half
Maximum tree depth	5
Crossover operators	Subtree, uniform, context-preserving, size-fair
Mutation operators	Subtree, Gauss, hoist, node complement, node replacement, permutation, shrink
Mutation probability	0.3

important parameters, as it influences the complexity of the evolved PFs and the size of the search space. Therefore, several three depths ranging from 3 until 15 were tested. The results demonstrated that the tree depth of 5 lead to the best results. Smaller tree depths do not allow GP to evolve complex enough DRs, whereas for the larger tree depths the search space is too large and it is more difficult for GP to obtain good PFs. Since several genetic operators are used for crossover and mutation, in each iteration a random operator is selected and applied to recombine the parents and mutate the child. The best set of genetic operators used in GP was also determined in a previous study by testing out the effectiveness of different genetic operator combinations [92]. This was done in a way that the set of genetic operators is constructed from scratch by adding those operators that increase the performance of the generated DRs. The operators were added until no performance increase was observed. Detailed descriptions of the applied genetic operators can be found in [19]. For the primitive set, the nodes described in Section 3.2 were used, but the concrete nodes that will be selected depend on the SGS that is applied in the DR to construct the schedule. Since the twotrees SGS uses two PFs to construct the schedule, GP evolves two PFs simultaneously. This is done in a way that each individual in GP consists out of two independent expression trees where one represents the PF for ordering jobs while the other denotes the PF used for selecting the machine. Genetic operators are performed independently on each of the two expressions. The two trees contained in the individual are evaluated together to determine how well they work when used for constructing the schedule with the twotrees SGS.

To test the performance of the proposed SGS variants, GP is used for the generation of PFs for each SGS variant. GP evolves the PFs using a training set consisting of 60 problem instances, in which the number of jobs ranges from 12 to 100, while the number of machines ranges from 3 to 10 [92]. After the termination criterion is satisfied, the best individual obtained during the evolution on the training set is saved. The obtained DR is tested on an independent test

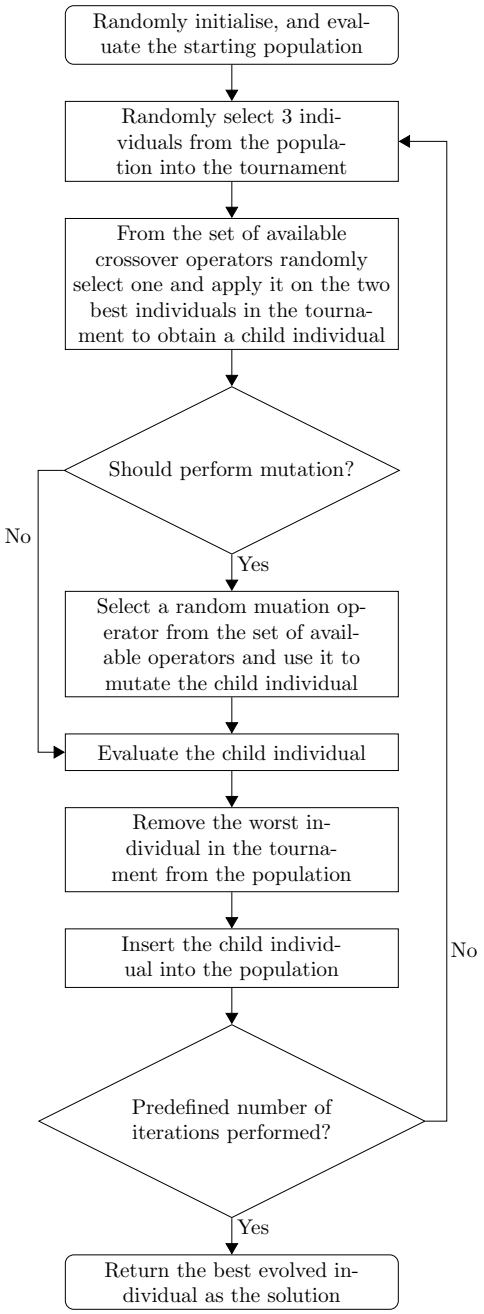


Figure 3: Flowchart of the applied GP algorithm

set, which also contains 60 problem instances. The rule is evaluated on the set by independently solving each problem instance in it. The total performance of the DR is calculated as the sum of the criterion values, in this case, the TWT values, for each of the problem instances in the set. Due to the different sizes and characteristics of the problem instances, the criterion values are normalised to ensure that each instance has a similar influence [92]. For each experiment 30 independent runs are performed to obtain statistically relevant results. The best individual in each of these 30 runs is saved, and based on their performance on the test set the minimum, median, average, maximum, and standard deviation values of the results are calculated, which will be denoted as *min*, *med*, *avg*, *max* and *std* in the tables displaying the results. When comparing different types and variants of SGS, the Kruskal-Wallis rank sum statistical test was used to determine whether there is a significant difference between the two experiments. The difference are considered significant if the obtained *p-value* is less than 0.05. Furthermore, for additional post hoc analysis, the Canover method further adjusted with the Benjamini-Hochberger false discovery rate method was used.

To further test the performance of the considered SGS variants, they will also be compared to three manually designed DRs which obtained the best results for the TWT criterion. The DRs that will be used are the ATC, COVERT, and EDD [41]. The reason why these three DRs were selected as baselines is because out of the 26 manually designed DRs that were tested, those three rules obtained the best results on the training set. The k parameter for the ATC rule is set to 0.5, and that of COVERT to 0.2, based on preliminary experiments on the training set.

Furthermore, to reduce the amount of text in figures, tables, and paragraphs when referring to the different tested variants of SGS, the notation denoted in Table 4 will be used.

The source code that was used in the experiments can be obtained via the web site of the ECF framework available at `http://ecf.zemris.fer.hr/`, whereas the problem instances and additional descriptions about them, can be obtained from the project site available at `http://gp.zemris.fer.hr/scheduling/`.

6. Experimental results

The results for all the tested SGS variants that use idle times are collected in Table 5, with the best results denoted in bold for each SGS. The priority columns denote whether the maximum or minimum value of the PF is used for selecting machines and jobs. Furthermore, the results for the three previously outlined manually designed DRs are also included in the table. Since all three DRs are deterministic, only a single value is denoted for them. It can immediately be seen that all the proposed SGS variants achieve a better median value than any of the manually designed DRs. The magnitude of the improvements over the best manually designed DR, namely the ATC rule, depends on the SGS that was used, but ranges from around 2% to 6%. However, the best DRs which

Table 4: Notations for the variants of SGS

SGS	Priority		Notation
	job	machine	
Simple	min	min	s-mm
	min	max	s-mx
	max	min	s-xm
	max	max	s-xx
Heuristic1	min	-	h1-m
	max	-	h1-x
Heuristic2	min	-	h2-m
	max	-	h2-x
Twotrees	min	min	t-mm
	min	max	t-mx
	max	min	t-xm
	max	max	t-xx

were evolved by GP can achieve an improvement of up to 16%, which shows the superiority of automatically generated DRs over manually generated DRs. When compared to other DRs, the improvements are even more evident.

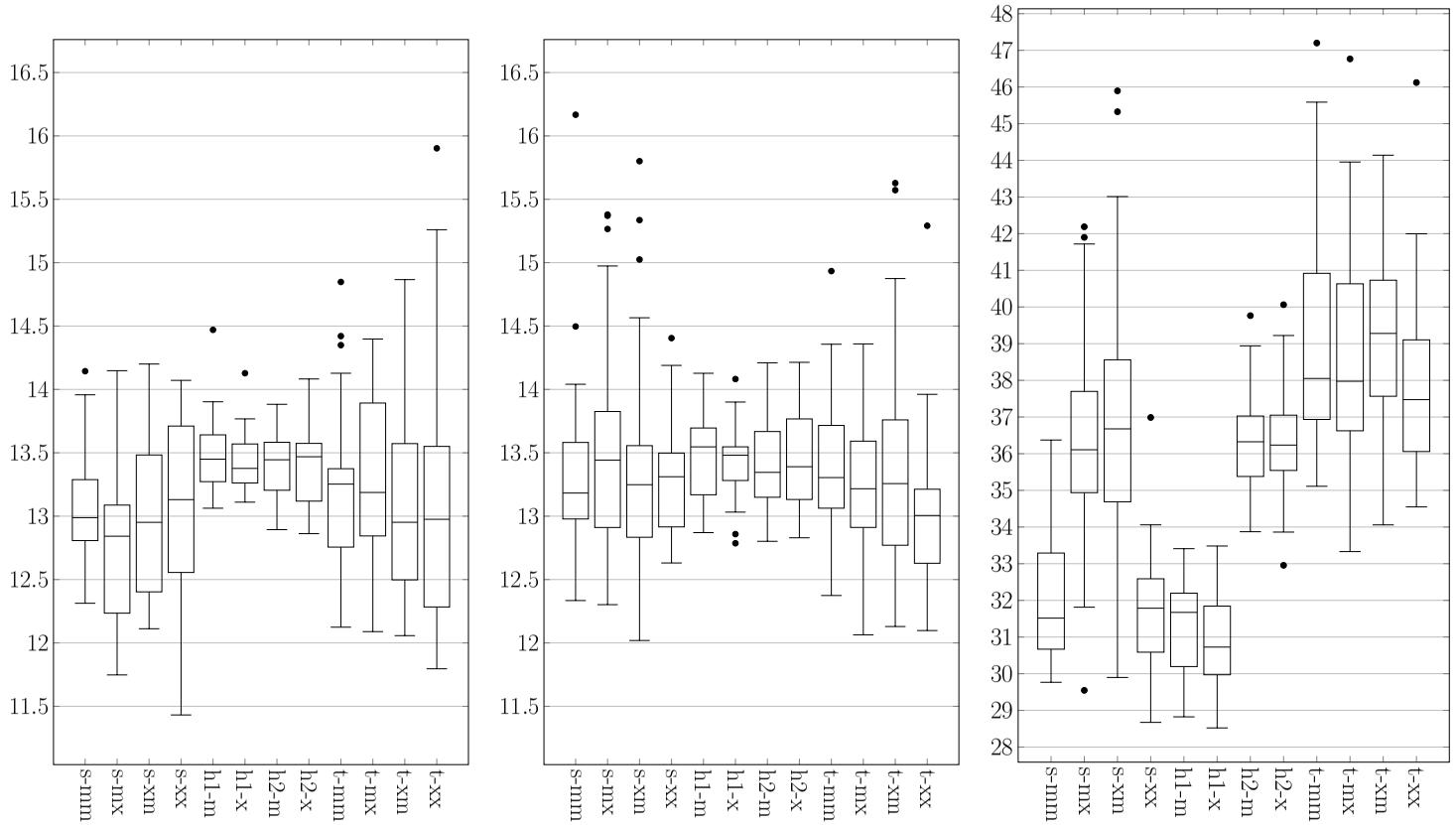
To determine whether significant differences exist between the SGS variants, the Kruskal-Wallis test was performed, and the p-value of $9.5 * 10^{-8}$ was obtained. This means that the null hypothesis, which was that all the results are from the same distribution, must be rejected. The p-values of the post hoc analysis (using the Canover method) are represented in Table 6. The values which denote that a statistically significant difference exists between the SGS variants denoted in the row and column are presented in bold. The results in Table 5 show that out of the three types of SGS, the heuristic SGS obtained the worst median values. The statistical tests prove that the results obtained by the heuristic SGS are significantly worse than the results of all other SGS variants (except for t-mx). From these results it is clear that restricting the choices that are delegated to the PF also limits the quality of the evolved DRs. Even though it might be harder for GP to evolve DRs which consider both the sequencing and allocation of jobs, allowing it to consider both decisions enables it to evolve PFs with a better synergy, which easily outperform the heuristic SGS. Nevertheless, the DRs generated by using the heuristic SGS still achieve a better median performance than the manually designed DRs. Since most manually designed DRs allocate jobs to machines using the same strategy as the heuristic SGS, this means that the PFs used in those DRs can be further improved. However, the results could be enhanced further by designing a better allocation scheme of jobs to machine, as by automatically designing this part has lead to significantly better results. Therefore, one can conclude that the allocation of jobs to machines used in the manually designed DRs could in many cases represent the

Table 5: Results of different SGS variants (with idle times allowed)

SGS	Priority job	Priority machine	Min	Avg	Med	Max	Std
ATC	max	-	-	13.73	-	-	-
COVERT	max	-	-	14.20	-	-	-
EDD	max	-	-	14.53	-	-	-
Simple	min	min	12.31	13.08	12.99	14.14	**0.428**
	min	max	11.75	**12.77**	**12.86**	14.15	0.594
	max	min	12.11	12.99	13.00	14.20	0.586
	max	max	**11.43**	13.11	13.15	**14.07**	0.667
Heuristic1	min	-	**13.06**	13.49	13.45	14.47	0.281
	max	-	13.11	**13.43**	**13.38**	14.13	**0.233**
Heuristic2	min	-	12.89	**13.41**	**13.46**	**13.88**	**0.274**
	max	-	**12.86**	**13.41**	13.48	14.08	0.327
Twotrees	min	min	12.12	13.18	13.25	14.85	**0.646**
	min	max	12.09	13.31	13.19	**14.40**	0.647
	max	min	12.06	**13.10**	**12.98**	14.87	0.716
	max	max	**11.80**	13.16	13.05	15.90	0.966

bottleneck which limits the performance of the DRs. Between the two variants of the heuristic SGS, it can be seen that the heuristic1 variant, which calculates the priority values for jobs by additionally using machine properties, achieves a slightly better median value. However, the statistical tests show that there is no significant difference between the two variants of this SGS. Furthermore, the results show that there is no significant difference when selecting the job with the highest or lowest priority. Although the heuristic SGS achieves significantly worse results when compared to the other two variants, it still does have certain benefits. One benefit can be seen from the box plot representation of the results in Figure 4a, from which it is evident that out of all SGS variants this one obtains the least dispersed results. This means that the method is quite stable and that upon several invocations of GP one is very likely to obtain a DR which will perform quite similarly.

The DRs that use the simple SGS obtain the best median and minimum values out of all three tested SGS variants. However, the results show that the performance of the SGS also depends to a certain degree on whether elements with the minimum or maximum value are selected. The simple SGS achieves the worst median values when both the job and machine with the highest priorities are selected for scheduling (s-xx). On the other hand, the best median value is obtained when the job with the lowest priority and the machine with the highest priority value is selected (s-mx). The statistical tests show that the s-mx variant achieves significantly better results only when compared to the variant which selects jobs and machines with the maximum priority value. In



(a) SGS results (b) SGS results when taking absolute values of PFs (c) SGS results without allowing idle times

Figure 4: Box plot representations of the obtained results

Table 6: Statistical tests for the different variants of SGS

	s-mm	s-mx	s-xm	s-xx	h1-m	h1-x	h2-m	h2-x	t-mm	t-mx	t-xm	t-xx
s-mm	-	0.15	0.94	0.54	**0.00**	**0.01**	**0.01**	**0.01**	0.94	0.49	0.77	0.77
s-mx	0.15	-	0.20	**0.03**	**0.00**	**0.00**	**0.00**	**0.00**	**0.04**	**0.00**	0.06	0.06
s-xm	0.94	0.20	-	0.47	**0.00**	**0.01**	**0.01**	**0.01**	0.54	0.10	0.72	0.72
s-xx	0.54	**0.03**	0.47	-	**0.02**	**0.04**	0.05	0.06	0.94	0.49	0.77	0.77
h1-m	**0.00**	**0.00**	**0.00**	**0.02**	-	0.76	0.72	0.68	**0.01**	0.11	**0.01**	**0.01**
h1-x	**0.01**	**0.00**	**0.01**	**0.04**	0.76	-	0.96	0.94	**0.04**	0.27	**0.02**	**0.02**
h2-m	**0.01**	**0.00**	**0.01**	0.05	0.72	0.96	-	0.96	**0.04**	0.29	**0.02**	**0.02**
h2-x	**0.01**	**0.00**	**0.01**	0.06	0.68	0.94	0.96	-	**0.04**	0.33	**0.03**	**0.03**
t-mm	0.94	**0.04**	0.54	0.94	**0.01**	**0.04**	**0.04**	**0.04**	-	0.42	0.85	0.85
t-mx	0.49	**0.00**	0.10	0.49	0.11	0.27	0.29	0.33	0.42	-	0.27	0.27
t-xm	0.77	0.06	0.72	0.77	**0.01**	**0.02**	**0.01**	**0.03**	0.85	0.27	-	0.99
t-xx	0.77	0.06	0.72	0.77	**0.01**	**0.02**	**0.02**	**0.03**	0.85	0.27	0.99	-

addition, from the box plot it is evident that this variant obtained the best distribution of the solutions. In all other cases there is no significant difference between the tested variants. Therefore, although it cannot be stated that this variant achieves significantly better results than the other three, it still seems to perform slightly better. Such behaviour is very interesting, but it seems that using the PF differently for selecting jobs and machines allows GP to better design it. This could be because the logic of the PF when selecting jobs and machines is to a certain degree inverted, and thus it works better by selecting jobs in one way and the machines in another. Although the benefit of this SGS is that only a single PF needs to be evolved, this also has a downside in that the PF will probably be much harder to interpret since it is tasked with determining the priorities for both jobs and machines. When compared to the twotrees SGS, it is evident that the simple SGS usually achieves slightly better median values. However, the statistical tests demonstrate that there is no significant difference between these two SGS variants in most cases.

As previously outlined, the twotrees SGS achieves median values that are usually slightly worse than those of the simple SGS, but better than the heuristic SGS. The statistical tests show that there is no significant difference in the results obtained by the different twootrees SGS variants. Nevertheless, it is still evident that this SGS also has a bias towards certain ways of selecting jobs and machines. In this case, the SGS prefers to select the jobs by highest priority, while the way in which the machines are selected does not seem to have that much of an influence. Dividing the responsibility of selecting jobs and machines into two PFs can be beneficial since each PF now has to concern with a smaller problem and should also be more understandable and interpretable.

An additional thing to observe about the tested variants of SGS is the difference in the time that is required to evolve PFs for each of them, and the time required to construct the schedule. Table 7 represents the minimum, average, median, maximum and standard deviation of the times required to evolve and execute a PF for each SGS based on 30 executions. The results demonstrate that in both cases the heuristic2 SGS is the most time-efficient. The reason why

Table 7: Training and execution times of each SGS

	Training time (minutes)					Execution time (miliseconds)				
	min	avg	med	max	std	min	avg	med	max	std
Simple	72	85	85	103	7.1	56	72	68	109	13.2
Heuristic1	57	70	70	84	7.3	42	54	55	68	6.6
Heuristic2	11	13	13	15	1.0	11	15	13	33	5.1
Twotrees	75	94	92	128	14	44	65	64	92	10.6

this SGS is so efficient is because it calculates the priorities only for jobs, which means that it calculates the priorities the least number of times. The heuristic1 SGS, although significantly slower than the heuristic2 SGS, is still slightly faster than the other two SGS variants. The twotrees SGS has the slowest training time, which is expected since it consists out of two trees that need to be evaluated and evolved. However, the execution time is in the end similar to the one of the simple SGS, which demonstrates that using an additional PF to calculate the priority values does not have a negative effect on the execution time.

To test the effects of using only the absolute value of the PFs, all SGS variants were additionally tested in a way that the absolute value of the evolved PF is used to select jobs and machines. Table 8 and Figure 4b represents the results obtained for this PF version. The table includes an additional column denoted as "delta" which represents the difference of the median value obtained by the SGS when taking the absolute value of all PFs, and the median value of the same SGS from Table 5. The results show that in most cases the difference of median values with and without absolute values is almost negligible for most SGS variants, although the results slightly deteriorate when using the absolute values of PFs. Nevertheless, even when taking absolute values of PFs the median values of automatically generated DRs are still better than by the three best manually designed DRs. Another thing which is evident from Figure 4b is that the obtained results are more dispersed and contain much more outliers. Because of these reasons taking the absolute values of PFs does not seem to bring any evident benefit to the SGS, and thus should probably not be used. When comparing the variants of SGS which use the absolute value against the ones which do not, the statistical tests demonstrated that there is no significant difference in their performance except for s-mx where the variant with the absolute value obtained significantly worse results.

In all the tests until now the SGS variants were allowed to insert idle times when creating schedules. In the next experiment, all the SGS variants are executed again but this time without being allowed to insert idle times in the schedule. The results of SGS variants without using idle times are shown in Table 9. The three DRs that were used in previous comparisons were also adapted to also construct schedules without idle times. This was done in the same way as for automatically designed DRs, meaning that when selecting to which machine a job should be allocated, only the machines which currently do

Table 8: Results of different SGS variants when taking absolute values of PFs

SGS	Priority job	Priority machine	Min	Avg	Med	Max	Std	Delta
Simple	min	min	12.33	13.37	**13.20**	16.17	0.696	0.21
	min	max	12.30	13.62	13.56	15.38	0.871	0.70
	max	min	**12.02**	13.38	13.26	15.80	0.865	0.26
	max	max	12.63	**13.30**	13.33	**14.40**	**0.420**	0.18
heuristic1	min	-	12.87	13.47	13.55	14.13	0.339	0.13
	max	-	**12.79**	**13.43**	**13.49**	**14.08**	**0.266**	0.11
heuristic2	min	-	**12.80**	**13.46**	13.44	**14.21**	0.395	-0.02
	max	-	12.83	**13.46**	**13.40**	**14.21**	**0.373**	-0.08
twotrees	min	min	12.37	13.39	13.32	14.93	**0.531**	0.07
	min	max	**12.06**	13.28	13.22	**14.36**	0.542	0.03
	max	min	12.13	13.46	13.26	15.63	0.878	0.28
	max	max	12.10	**13.04**	**13.03**	15.29	0.613	-0.02

not execute any job are considered. The parameter values for these adapted DRs were optimised again, and this time the parameter value of 1.2 is used for ATC and 5.9 for COVERT.

The results in Table 9 and Figure 4c show that by not allowing idle times in the schedule the value for the TWT criterion deteriorated significantly. For example, the value of the criterion increased by around 2.5 times when compared to the results obtained when allowing the insertion of idle times. This shows that allowing DRs to insert idle times into the schedule, and having the possibility of postponing the scheduling of jobs is important when constructing schedules. The statistical tests clearly demonstrated that this variant is significantly worse than the one which uses idle times. Therefore, the results obtained when not using idle times will not be discussed further, since it is clear that no benefit can be achieved by not using them.

7. Result analysis

7.1. Analysis of the generated PFs

In this section, the different PFs which were generated by GP for the tested SGS variants are analysed to gain knowledge about their behaviour and mutual differences. For each SGS the corresponding PF with the best performance on the test set were selected. All the PFs were additionally simplified by removing parts which do not significantly influence the performance of the DR (meaning that the performance of the DR in question did not change more than by 1% on the entire test set), so that the analysis is easier.

The first PF that will be analysed is the one obtained for the s-xx SGS, which means that the job and machine with the highest priority values are selected.

Table 9: Results of different SGS variants without allowing idle times

SGS	Priority job	Priority machine	Min	Abs	Med	Max	Std
ATC	max	-	-	36.07	36.07	-	-
COVERT	max	-	-	38.24	38.24	-	-
EDD	max	-	-	148.5	148.5	-	-
Simple	min	min	29.76	32.01	**31.59**	**36.37**	**1.694**
	min	max	29.55	36.56	36.35	42.19	2.786
	max	min	29.90	36.95	36.77	45.89	3.835
	max	max	**28.67**	**31.80**	31.79	36.99	1.705
heuristic1	min	-	28.82	31.35	31.72	**33.41**	**1.204**
	max	-	**28.52**	**30.91**	**30.73**	33.48	1.264
heuristic2	min	-	33.87	36.43	36.41	**39.76**	**1.271**
	max	-	**32.96**	**36.26**	**36.28**	40.06	1.472
twotrees	min	min	35.11	39.33	38.70	47.20	3.117
	min	max	**33.33**	38.88	38.31	46.77	3.004
	max	min	34.06	39.24	39.57	**44.14**	2.554
	max	max	34.55	**38.17**	**37.93**	46.12	**2.538**

The evolved PF is defined with the following expression

$$SL * \frac{dd}{MR} * \frac{dd+pt}{MR} - (dd - PAT) * (dd + pt) - \frac{dd + MR + pt}{dd * w + \frac{pt}{MR}} * dd * (MR + pt).$$

The PF can be seen to consist out of three parts which are summed up. The first part focuses on selecting jobs and machines which would cause a larger slack and processing time. This is because the expression contains both SL and pt in the nominator, and thus the value of the expression increases with the increate of the processing times and slack of jobs. Furthermore, the importance of this part increases as the machine becomes available sooner, since the MR value is located in the denominator. This part seems very unintuitive since if used only by itself it would cause a lot of jobs to be tardy. However, the rest of the expression tries to reduce the influence of this element, thus the PF balances between jobs with a higher and lower slack and processing time values. The second subexpression, namely $-(dd-PAT)*(dd+pt)$, selects those jobs which have a smaller due date and processing time values, which is due to the negative sign that appears in front of it. The final and third part of the PF is quite complicated and difficult to interpret. Still, it seems that the main focus of this part is to select jobs with a smaller processing time and machines which become available sooner, since the pt and MR values appear in the nominator of the expression several times, and thus have the main influence in this subexpression. Additionally, jobs with a smaller weight are preferred, due to the expression $dd * w$ which appears in the denominator. Since the entire subexpression needs to be minimised (due to

the negative sign before it), the denominator needs to be as large as possible, which happens for jobs with larger priority values. In the end, the PF tries to balance among the first expression in the PF and the other two to determine which job and machine should be selected. However, for this PF it has proven to be quite difficult to understand its underlying logic in more detail.

For the simple SGS, the PF which selects the job with the lowest and the machine with the highest priority value will also be analysed. The PF evolved for this SGS is defined by the following expression

$$2 * \frac{SL}{MR} + \frac{PAT}{w} + 2 * SL + \frac{pavg}{MR} + pmin - pt - MR + \frac{pmin}{w}.$$

It is immediately evident that this PF is much simpler than the last one, and thus easier to interpret. When selecting the appropriate machine for a job, the most relevant part of the PF is $\frac{2*SL}{MR} + 2 * SL - pt - MR$. Since the machine with the maximum priority value is selected, this means that this part of the PF would favour machines with a larger slack value for the job, due to the SL variable appearing two times in the expression. In addition, it also favours machines for which the selected job has a smaller execution time, as a consequence of the $-pt$ part that appears. Finally, machines which become available sooner are also preferred, since the MR variable appears both in the denominator part of the subexpression and also with a negative sign, which means that a lower value of MR will increase the value of the overall expression. The reason why the PF selects machines on which the job would have a higher slack is that a zero slack value would mean that the job cannot finish in time on the given machine since it is already tardy. Nevertheless, selecting the machine for which the job would have the largest slack does not seem like a good decision, because that job could likely be scheduled at a later moment without causing an additional penalty. However, once a machine is selected for every job, the PF is used to select the job where this part of the PF is now minimised instead of being maximised. This means that amongst all the jobs the one with the smallest slack, but the largest processing time is preferred. In that way, the SGS tries to schedule jobs with smaller slack values so that they finish as soon as possible. In addition, the $\frac{PAT}{w} + \frac{pavg}{MR} + pmin + \frac{pmin}{w}$ part forces the PF to select jobs with a smaller minimum and average processing time values, as the $pmin$ and $pavg$ variables appear several times in the expression. Additionally, since the w variable is located in the denominator several times, the value of the expression will be larger for jobs with a higher weight. The value of the subexpression becomes larger as the value of the PAT variable increases. This means that the PF will select those hobs whose machine on which they would execute the fastest becomes available the latests. The reason why those jobs are preferred is because it might be better to schedule those jobs on an alternative machine which might not be the best choice for them, rather than to wait a long time for the appropriate machine to become available. When the machine is fixed the MR variable is constant and thus does not have an influence on the value of the subexpression. This example demonstrated that constructing PFs in this way does have its benefits since the logic for selecting jobs and machines is slightly

inverted. Thus, the same parts of the expression are useful both for selecting jobs and machines, which in turn leads to more simple expressions. Therefore the PF constructed for this SGS variant is much easier to interpret than the one which was constructed for the variant in which both the machines and jobs with the highest priorities were selected.

The best PF evolved for the heuristic SGS which calculates the priorities for jobs by taking into account machines is defined as

$$2 * pt - pavg + dd + \frac{pavg}{PAT} + \frac{pt}{dd} * w * SL * PAT.$$

The job with the lowest priority value is selected and scheduled on the machine with the earliest completion time. Most of the PF consists out of terminals like pt, dd, SL, and PAT, which are summed up or multiplied. Since all of them appear with a positive sign or in the nominators of the expression, the PF focuses on selecting the job with the smallest processing time and slack values. This is expected since the TWT criterion needs to be optimised. However, the $pavg$ terminal appears with a negative sign, while some other terminals like PAT and dd appear as denominators, which means that the priority value will be smaller when these terminals have a larger value. Therefore, these two terminals appear in the expression in places where they need to be both maximised nad minimised. As such, these terminals are probably included in the PF to avoid selecting the jobs with the smallest due dates and average processing time values, but rather to find a balance between jobs with higher and lower due dates and average processing times. In such a way, the PF avoids a bias towards jobs with with the smallest processing times, and gives a chance to also select other jobs as well. A significant benefit of this PF is that is not only quite simple but it is also easy to interpret its behaviour since it consists of a smaller number of elements.

The PF for the heuristic SGS which does not consider machines when selecting the job is given with the following expression

$$\frac{age * pmin}{w} + dd * \frac{SL + pmin}{w} + pavg * w * \left(PAT + \frac{SL}{w}\right) * (pavg + pmin).$$

This SGS selects the job which has the smallest priority value. Since this SGS uses only terminals that represent job characteristics, it has to make its decision solely by using that information. This PF bases its decision mostly on the minimum processing time of the job, its slack value, and the weight, as the terminals denoting these values appear most often in the expression. Since the w terminal appears quite often in the denominator parts of the expression, it can be concluded that this rule puts a heavy emphasis on selecting jobs with a higher weight. All other terminals appear with a positive sign or in the nominator parts of the expression. The most common terminals that appear in the expression are $pmin$, SL, and $pavg$. This means that the PF prioritises those jobs which are tardy or near tardy and have smaller processing times. It tries to reduce the number of tardy jobs but also selects those which are more likely

to be executed faster. Although the PF did not use any information about the machines, it nevertheless obtained a similar performance as the variant which also used information about the machines. Therefore, it seems that not using information about the jobs can lead to slightly better median values, since including machine properties might be misleading due to the fact that the PF does not influence the selection of the machine on which the job will be executed in the end.

Finally, the PFs constructed for the twotrees SGS will also be analysed. The SGS uses the PF in a way that it selects the job and machine which obtain the smallest priority value. The PF used to rank the jobs is defined as

$$\frac{pavg + dd}{w} * \left(pmin + \frac{SL}{w} \right) + \left(\frac{SL}{pavg} * pmin * age + \frac{SL * pmin}{w * w} - pavg * pavg \right),$$

while the PF used to select the machine is defined as

$$2 * pt + 2 * MR - \frac{pmin * SL * pavg}{pt * MR}.$$

From the first expression it can be seen that when calculating the priority for jobs the PF prioritises jobs with a larger weight, since the w terminal appears several times in the denominator parts of the expression. Additionally, the expression mostly consists out of the $pmin$, $pavg$ and SL terminals, all of which appear in the nominator part. Thus, the PF tries to schedule those jobs that are already tardy or close to being tardy and have a smaller processing time. The PF can also be seen to use the $pavg$ terminal to prioritise jobs with higher average processing times. In that way, the PF selects jobs with a smaller minimum processing time, but that have a larger average value of processing times, since this means that there is a high imbalance between the processing times on different machines. This means that if such a job would be allocated to an inappropriate machine its execution could be significantly prolonged, which would affect other jobs as well. As a result, the PF prioritises such jobs. The dd and age terminals also appear in the expression, which means that the PF prioritises jobs with closer due dates and jobs which have not been too long in the system. However, as those two nodes appear only once, they have a smaller influence on the the entire expression.

The PF used for selecting the machine on which the job has to be scheduled is quite easy to interpret. From the expression, it is evident that the PF gives a greater priority to machines that become available sooner and execute the selected job faster, which is evident from the expression $2 * pt2 * MR$. This part of the PF would allocate the jobs in the same way as the heuristic SGS, meaning it would schedule the job on the machine on which it would be completed the soonest. It is interesting to note how GP was able to learn the very same rule which was defined by human experts to select the machine on which to schedule a job. However, GP expanded this rule with additional terminals to further enhance it. The additional expression which is used to select the machines is $-\frac{pmin * SL * pavg}{pt * MR}$. As can be seen, in this part the machine with a larger slack is

Table 10: Details of the problem instance used for analysis

Job j index	0	1	2	3	4	5	6	7	8	9	10	11
r_j	68	85	79	42	12	66	44	68	62	43	0	51
dd_j	114	114	115	88	60	94	52	103	91	65	70	64
w_j	0.97	0.65	0.24	0.26	0.58	0.26	0.56	0.87	0.15	0.95	0.40	0.73
p_{0j}	68	95	23	65	54	45	38	42	17	43	41	49
p_{1j}	100	4	34	42	69	63	60	30	36	89	43	47
p_{2j}	36	12	74	12	48	58	24	79	91	7	65	6

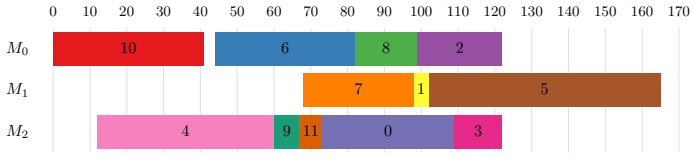
preferred (due to the minus sign of the entire expression), which means that the PF tries to select the machine on which the job will not miss its due date. The slack is additionally divided with the pt and MR terminals, which makes the slack more important when the job has a smaller execution time and the machine becomes available sooner, otherwise the slack has less influence in the expression. In that way, instead of relying solely on the completion time of the jobs, the PF uses additional information to allocate the job to the most appropriate machine. However, even such small additions lead to good improvements in the results, which additionally seems to prove that allocating jobs to machines only by using the completion times seems to be very limiting and has a significant influence on the performance of the SGS.

7.2. Analysis of scheduling decisions

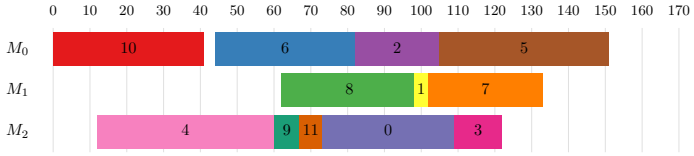
To better understand how the proposed variants of SGS work and to outline their similarities and differences, a small scheduling problem will be solved with three selected DRs for the following three variants: s-mm, h1-m, t-mm. For each SGS the DR which achieved the best results on the test set was selected and used to construct the solution for the considered scheduling problem. The properties of that problem are denoted in table 10.

Figure 5 shows the schedules that were generated by the three selected DRs. From the figure it is evident that at the start of the schedule all three DRs performed the same decisions. This can best be seen from the fact that they all scheduled the first five jobs that are released (jobs 10, 6, 4, 9, and 11) in the same way. With a closer look, one can see that each of these jobs is scheduled on the machine on which it would complete the soonest. Thus, the DRs which use the standard and twotrees variants work in the same way as the heuristic SGS at this first part of the schedule. This means that the strategies which the GP learned in the PFs are similar to the minimum completion time strategy that is used by different manually designed DRs.

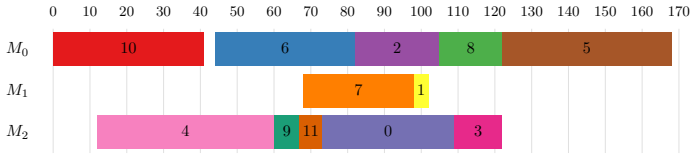
The first difference between the DRs occurs at the time moment 62, at which the DR with the heuristic SGS schedules job 8, whereas the other two DRs do not schedule any job at this moment. Here, the fundamental difference between the heuristic SGS and the other two can be described. At this moment, the DR with the heuristic SGS has no other choice than to schedule job 8 on machine 1. The reason is that on this machine the job would complete the soonest, at the moment 98, in comparison to moments 99 and 156 if it were scheduled on



(a) The schedule generated by the DR which uses the standard SGS



(b) The schedule generated by the DR which uses the heuristic SGS



(c) The schedule generated by the DR which uses the twotrees SGS

Figure 5: Schedules generated by the different SGS variants

machines 0 or 2 respectively. The other two DRs do not perform this decision, but rather determine that at this moment it would be better to schedule job 8 on machine 0. This is due to the reason that the other two variants of SGS can use some additional information when selecting the appropriate machine. In this case, those DRs determined that it would be better to schedule job 8 on machine 0 since it has a much shorter execution time on that machine. As a result, machine 1 remained free and job 7, which was released next, could immediately have been scheduled on that machine. In the end, this decision has demonstrated to be beneficial since job 7 has a higher weight, and by not scheduling job 8 on machine 1, it was possible to complete job 7 before its due date.

The next point in which the DRs disagree is whether to schedule job 2 or 8 first on machine 0. This decision mostly depends on the generated PF, since it needs to determine which of the two jobs it would be better to schedule first. The DR with the simple SGS decided to schedule job 8 first, which in the end did lead to a slightly smaller TWT value. Finally, some DRs also disagree on the placement of job 5. For the DR with the heuristic SGS it is clear why it scheduled this job on machine 0, because job 7 already started with its execution on machine 1. However, the situation is not as evident for the DR with the twotrees SGS. The question is why this DR did not schedule job 5 on machine 1 after job 1 finished with its execution, just like the DR with the simple SGS did. The reason is that at the moment when job 1 finished with its execution, the DR obtained the best priority for scheduling jobs 8 and 5 on

machine 0. However, only one of those jobs can be scheduled at that moment, and the other one will have to wait. In this case, job 8 had a better priority value and was scheduled at the moment when job 2 finished with its execution. At that moment, the DR also tried to see whether it would be better to schedule job 5 on machine 1 instead, but the rule determined that it would once again be better to wait and schedule it on machine 0. Therefore, the situation happened because the DR considers each job individually, and did not take into account that another job would be scheduled on the considered machine and thus that job 5 could not be scheduled on machine 0 at the next possible time. However, such scheduling strategy also has its benefits since it tries to not schedule jobs too greedily, which could cause a problem if quite soon other jobs with large weights would be released into the system.

In general, the largest difference occurs between the DR using the heuristic SGS and the other two DRs. The difference is a result of the fact that the heuristic SGS has a fixed strategy of allocating jobs to machines, while the other two evolve such a strategy. The schedules show that these two SGS variants also evolved strategies that work similarly as the strategy of the heuristic SGS since these DRs also tended to schedule jobs on the machines on which they would complete the soonest. Between the DRs which use the simple and twotrees variants, the difference is only in the PFs they evaluate, and not so much in the SGS. The generated schedules also show that at the start of the schedule all the DRs did perform the same decisions, which shows that in the case of low utilisation all DRs work similarly. However, the real differences start to appear later in the schedule as more jobs are released and fewer machines are available. In those cases, the DRs heavily depend on the PF to determine which job needs to be scheduled and as a consequence more differences start to appear.

8. Discussion

The experimental results and analysis which were presented in the previous two sections show that the performance of the generated DRs significantly depends on the variant of the SGS that is used. Out of the three tested variants of SGS, the heuristic SGS has achieved the worst results, while the twotrees and simple SGS achieved similar results, with the simple SGS being usually slightly better. The general conclusion which can be made based on these results is that it seems that the main strength of automatically designed DRs comes from the fact that they can design more efficient strategies of selecting the appropriate machine. Although the heuristic SGS did improve the results over the manually designed DRs, which means that job selection still leaves room for improvement, the largest improvements over the manually designed DRs were achieved by the other two variants of SGS which allow GP to also design the strategy of selecting the machines. This just demonstrates that the selection of the appropriate machine in the unrelated machines environment represents an equally important decision as selecting the appropriate job since it has a significant influence on the obtained results. A deeper analysis of the DRs showed that the automatically generated DRs did schedule the jobs on machines in

a similar manner as the manually designed DRs at the start of the schedule. An additional evidence of this is the fact that the PF which was analysed for the twotrees SGS did include a subexpression that would select the machines in the same way as the manually designed DRs would. This is an important proof which shows that the minimum completion time strategy is a good baseline for selecting machines. However, the automatically generated DRs also did use some other element based on which they selected the appropriate machine. These elements have resulted in better decisions in situations when more jobs were available for scheduling and less there were fewer available machines. In those situations the twotrees and simple SGS started to make better selections of machines thanks to additional properties.

The experiments which tested the different ways of selecting jobs and machines using either the highest or lowest values showed some interesting results. For the heuristic SGS, both ways of selecting jobs did not significantly influence the obtained results. For the twotrees SGS this choice had more influence on the obtained results since in some cases the DRs achieved slightly lower minimum fitness values than in others. However, the statistical tests did not show that there is any significant difference between the two variants, therefore it is not possible to claim that this would generally be true. The most interesting situation happened for the simple SGS, where it was demonstrated that by selecting jobs with the lowest and machines with the highest priority function value did lead to results that are significantly better than most of the other variants. In this case, where only one PF is used for selecting both jobs and machines, it seems to be more beneficial to use the PF in different ways for selecting jobs and machines. This was demonstrated also though the analysis of two selected DRs which use the simple SGS, in which one selected both job and machines with the highest value, and the other which selected the job with the lowest and machine with the highest values. The analysis showed that the first DR was quite complicated and included a sophisticated logic to select jobs and machines. On the other hand, the second DR was much simpler, and it was demonstrated that the logic in which jobs and machines are selected is inverted. This means that when selecting a machine it is preferred that parts of the PF are higher, but when selecting the jobs these parts have to be minimised. However, when both jobs and machines are selected with either the highest or lowest priority, then this complicates the PF and the logic it encapsulates.

The variant which used absolute values of PFs to select jobs and machines did not lead to improved results, rather it even leads to significantly worse results in several cases. Therefore, it seems that for GP it is easier to develop good PFs if they can have negative values. Furthermore, the additional experiment with idle times also demonstrated that introducing idle times in the schedule is invaluable. The reason for this is that in that way the developed DRs act less greedy, and allow machines to remain free in case that in the future important jobs enter the system.

9. Conclusion

This paper analysed several SGS variants for the automatic generation of DRs in the unrelated machines environment. The proposed SGS variants were tested in different variants for optimising the TWT criterion. Generally, it was demonstrated that the SGS in which the PF is used to only select the jobs, while the machine on which it will be executed is selected by a predefined heuristic, achieves the worst performance out of the tested SGS variants. The other two SGS variants achieved a similar performance, although the one which uses the same PF for selecting jobs and machines does achieve slightly, although not significantly, better results. All the variants of SGS achieved better results than any of the existing DRs. Furthermore, the results also prove that the logic of choosing elements of either the highest or lowest priority value can have a significant influence on the result, and thus also has to be carefully selected. An especially interesting observation that was made is that the best overall results obtained by any of the tested DRs were achieved when using an SGS variant in which the job with the smallest priority value, and the machine with the highest priority value are selected. The presented analysis showed that using the PF in such a way is beneficial since in that way the PF can more easily balance between the jobs and machines that it wants to select. Finally, it was also demonstrated that allowing the SGS variants to insert idle times in the schedule is extremely important and that without it the schedules that are generated are significantly worse. Based on all the outlined conclusions it is evident that the choices made in the design of the SGS can have a significant influence on the performance of the generated DRs.

In future studies, it is planned to extend the research further on the topic of designing SGS variants. One possible future research would be to analyse SGS variants for problems with additional constraints, like setup times or precedence constraints. Here it would be interesting to analyse whether the inclusion of such constraints has an influence on which of the SGS variants performs best, or even if it would be better to design specialised SGS variants for such problems. Another topic that has not been extensively examined is whether it would be possible to automatically generate an SGS similarly as the PF is generated. This would make it possible to automatically design the entire DR and not just the PF. An additional topic would also be to analyse if certain expert knowledge could be inserted into GP before the evolution process, based on which GP could evolve better DRs, but also take less time to evolve them. Finally, as it was seen that the evolved PFs can sometimes be hard to interpret, one of the next steps would also be to analyse and include some simplification procedures, both exact and inexact, which could simplify the expressions during the evolution of PFs.

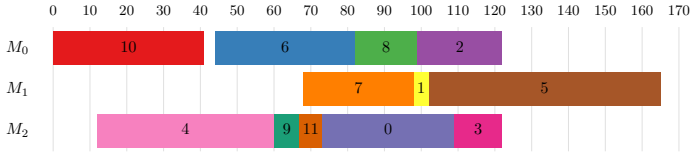
Appendix A. Analysis of different PFs generated for the SGS variants

Since the process of designing DRs is of stochastic nature, this section will outline the differences in schedules that can occur between different PFs designed for a certain SGS. This section will thus illustrate how for the same SGS different PFs can have an influence on the quality of the schedule. For each SGS variant, three PFs were selected and used to construct the the schedules. In addition to the PFs that were used in Section 7.2, the PFs which lead to the best and worst result for the considered problem instance were additionally used to construct schedules.

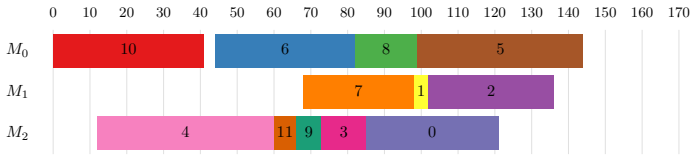
Figure A.6 shows three schedules obtained for the s-m SGS variant, denoted as S1, S2, and S3. The fitness values denote that there is a quite big difference in the quality of the 3 sample schedules, however, the median value of all the 30 schedules constructed by the evolved DRs is equal to 0.798, which demonstrates that most DRs did not perform well on this problem instance. The figure shows that in all three schedules jobs 4, 10, 7, and 1, are placed in the same positions. Between the two better schedules, S1 and S2, there are only slight differences at the end of the schedule. The first difference is in the placement of jobs 2 and 5, namely which of these two jobs should be scheduled when machine M_0 becomes free after executing job 8. DR2 that created schedule S2 selected that job 5 should be scheduled due to it already being tardy, and having a slightly larger priority value. This has proven to be a better choice since job 5 was completed much sooner than in schedule S2. Although job 2 did take much more time to finish than in schedule S1, it caused a smaller weighted tardiness penalty. The second difference appears in the order of jobs 11 and 9 on machine M_2. The better schedule S2 scheduled job 11 first, since it would sooner become tardy. However, this was a worse choice than the one in S1 where job 9 was scheduled first, since job 9 has a higher priority value and thus causes a larger weighed tardiness value if not scheduled first. However, this decision is quite hard to make, as both jobs have a short execution time and a close due date. The third difference is whether to schedule job 0 or 3 first on machine M_2. DR2 performed a better decision by selecting job 3 first as it can finish it before its due date, and it will make job 0 miss its due date only slightly. On the other hand, by selecting job 0 first in S1 job 0 could finish before its due date, but this resulted in job 3 missing its due date by a large amount of time, which lead to a large weighted tardiness.

Schedule S3 is the worst one out of all the schedules that were obtained for the s-m SGS variant. Although this schedule has many similarities to the best schedule S2, one poor decision by the DR was enough to construct a schedule with a poor quality. This decision was not to schedule job 6 on machine M_0 as was the case in the other two schedules. This was due to the fact that job 6 executes the fastest on machine M_2 so DR3 decided to not schedule it on any other machine. This allowed jobs 8 and 2 to finish sooner. However, job 6 was not executed immediately after executing job 4, since the other jobs that were released in the meantime had a larger priority value due to their shorter

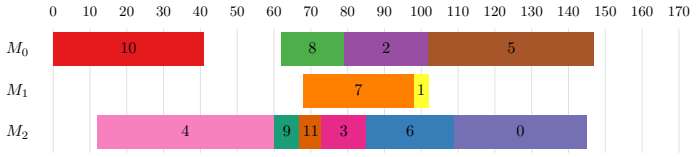
execution times. When these jobs finished executing, job 6 was finally executed and after it job 0. This resulted in a high weighted tardiness value as both jobs ended up being quite late, but also had high priority values, especially job 0. The problem with the DR that created this schedule was that it focused too much on the execution times of the jobs, and tried to schedule jobs on the machines on which they have the shortest execution time. However, as can be seen, this has proven to be quite a bad strategy.



(a) Schedule S1 generated by the first DR (DR1), with fitness value of 0.523



(b) Schedule S2 generated by the second DR (DR2), with fitness value of 0.496

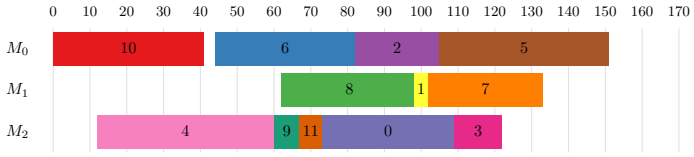


(c) Schedule S3 generated by the third DR (DR3), with fitness value of 0.804

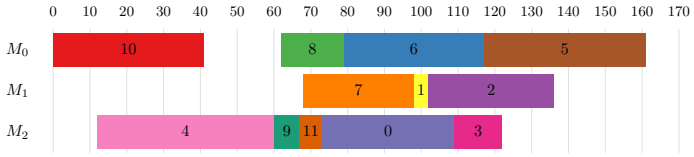
Figure A.6: Schedules generated by DRs using the s-m SGS variant

Figure A.7 shows the three schedules created by three different DRs which use the h1-m SGS variant. The fitness values denote that all the differences between the best schedule S1 and the worst schedule S3 are quite small. This shows that the variance between the different DRs was smaller compared to the standard or twotrees SGS variants. The median value of all 30 rules that were generated was equal to 0.804, which shows that most DRs did not perform well for this problem. Although schedules S1 and S2 have a similar fitness, there are several key differences between them. DR2 which constructed schedule S2 decided that job 6 should not be scheduled immediately on machine M_0, but rather wanted to schedule it on another machine later on. However, as other new jobs were released into the system that were more appropriate to be scheduled on other machines, the DR eventually scheduled the job on machine M_0 after job 8 finished with its execution. Additionally DR1 scheduled job 8 on M_1 rather than on M_0, as it would have sooner executed on that machine than if it would wait for job 6 to finish on machine M_0. Therefore, once again a poor decision for job 6 lead to a deterioration of the results. Although this difference

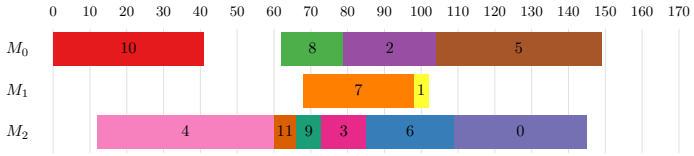
was quite small between schedules S1 and S2, as DR2 did manage to make some good choices when creating schedule S2. However, this cannot be said for S3, which is almost completely the same as S3 that was created using the s-m SGS variant. The only difference is that the positions of jobs 11 and 9 are switched, which leads to an even worse result.



(a) Schedule S1 generated by the first DR (DR1), with fitness value of 0.717



(b) Schedule S2 generated by the first DR (DR2), with fitness value of 0.727

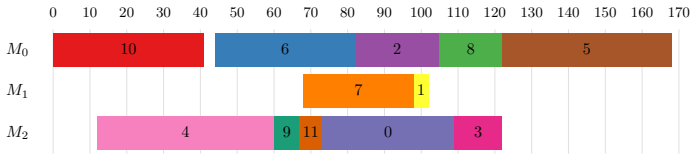


(c) Schedule S3 generated by the first DR (DR3), with fitness value of 0.810

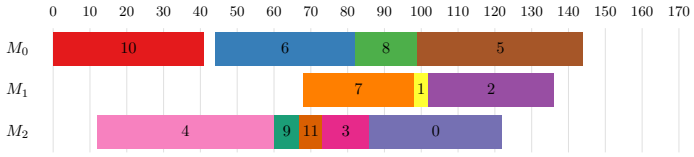
Figure A.7: Schedules generated by DRs using the h1-m SGS variant

Finally, Figure A.8 shows the schedules generated by the DRs that use the t-mm SGS. The fitness values of the three outlined schedules show that it was possible to obtain DRs which perform quite differently, however, the median value of all the 30 DRs was 0.792, which is due to the fact that most generated DRs did not perform well on this instance. Schedule S2 achieved the overall best performance. It is the same as the schedule S2 generated by the S-m SGS variant, except that it reversed the position of jobs 9 and 11 and thus slightly improved the quality of the schedule. In comparison to S2, schedule S1 has several key differences. Namely when constructing this schedule the DR tried to execute jobs 2 and 0 sooner. The reason why the DR did this is that both jobs have a larger weight than the jobs 3 and 8 that were also considered at that time. And although it might seem like a logical decision to reduce the tardiness of those jobs that have a larger weight, in this situation it was a poor decision as jobs 3 and 8 had a much closer due date than the other two jobs. Therefore, by postponing the scheduling of those two jobs to a later moment caused had a large influence on the weighted tardiness of the schedule, regardless of their weight. The worst schedule S3 for this SGS also incorporates the problem with the poor scheduling decision for job 6. However, in this case the only improvement was

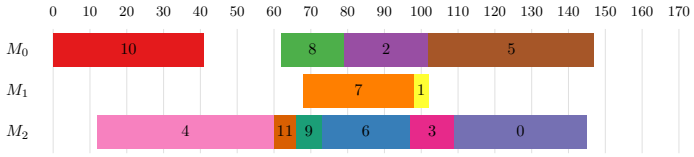
that job 6 was scheduled prior to job 3, which in the end did lead to a slight improvement in the fitness, but still the quality of this schedule is poor when compared to others.



(a) Schedule S1 generated by the first DR (DR1), with fitness value of 0.553



(b) Schedule S2 generated by the first DR (DR2), with fitness value of 0.490



(c) Schedule S3 generated by the first DR (DR3), with fitness value of 0.798

Figure A.8: The schedule generated by the DR which uses the t-mm SGS variant

Based on the performed analysis it is evident that the simple and twotrees SGS variants have similar expressiveness, as they can obtain good DRs for the considered problem instance, both of which perform almost equally well. On the other hand, the performance of the heuristic SGS variant is more limited due to the way in which the jobs are allocated to the machines. Because of that, the DRs using the heuristic SGS variant are unable to perform some good decisions as those DRs that use the other two SGS variants.

References

[1] M. L. Pinedo, Scheduling: Theory, algorithms, and systems: Fourth edition, Vol. 9781461423614, Springer US, Boston, MA, 2012. `arXiv:arXiv:1011.1669v3`, `doi:10.1007/978-1-4614-2361-4`.

[2] S. Petrovic, E. Castro, A genetic algorithm for radiotherapy pre-treatment scheduling, in: Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 454–463. `doi:10.1007/978-3-642-20520-0_46`.

[3] N. Pillay, R. Qu, Nurse Rostering Problems, Springer International Publishing, Cham, 2018, pp. 61–66. doi:10.1007/978-3-319-96514-7_8.
URL https://doi.org/10.1007/978-3-319-96514-7_8

[4] J. A. Castillo-Salazar, D. Landa-Silva, R. Qu, Workforce scheduling and routing problems: literature survey and computational study, Annals of Operations Research 239 (1) (2014) 39–67. doi:10.1007/s10479-014-1687-2.
URL https://doi.org/10.1007/s10479-014-1687-2

[5] F. B. Oyebolu, R. Allmendinger, S. S. Farid, J. Branke, Dynamic scheduling of multi-product continuous biopharmaceutical facilities: A hyper-heuristic framework, Computers and Chemical Engineering 125 (2019) 71 – 88. doi:https://doi.org/10.1016/j.compchemeng.2019.03.002.
URL http://www.sciencedirect.com/science/article/pii/S009813541831189X

[6] E. Hart, P. Ross, D. Corne, Evolutionary Scheduling: A Review, Genetic Programming and Evolvable Machines 6 (2) (2005) 191–220. doi:10.1007/s10710-005-7580-7.

[7] S. Nguyen, D. Thiruvady, A. T. Ernst, D. Alahakoon, A hybrid differential evolution algorithm with column generation for resource constrained job scheduling, Computers and Operations Research 109 (2019) 273 – 287. doi:https://doi.org/10.1016/j.cor.2019.05.009.
URL http://www.sciencedirect.com/science/article/pii/S0305054819301224

[8] I. Vlašić, M. Đurasević, D. Jakobović, A comparative study of solution representations for the unrelated machines environment, Computers & Operations Research 123 (2020) 105005. doi:https://doi.org/10.1016/j.cor.2020.105005.

[9] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st Edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[10] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, IEEE, 1995, pp. 1942–1948. doi:10.1109/ICNN.1995.488968.
URL http://ieeexplore.ieee.org/document/488968/

[11] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 26 (1) (1996) 29–41. doi:10.1109/3477.484436.
URL http://ieeexplore.ieee.org/document/484436/

[12] H. Chen, S. Jiao, A. A. Heidari, M. Wang, X. Chen, X. Zhao, An opposition-based sine cosine approach with local search for parameter estimation of photovoltaic models, Energy Conversion and Management 195 (2019) 927 – 942. doi:https://doi.org/10.1016/j.enconman.2019.05.057.
URL http://www.sciencedirect.com/science/article/pii/S0196890419306120

[13] A. A. Heidari, R. A. Abbaspour, H. Chen, Efficient boosted grey wolf optimizers for global search and kernel extreme learning machine training, Applied Soft Computing 81 (2019) 105521. doi:https://doi.org/10.1016/j.asoc.2019.105521.
URL http://www.sciencedirect.com/science/article/pii/S1568494619302911

[14] J. Luo, H. Chen, A. A. Heidari, Y. Xu, Q. Zhang, C. Li, Multi-strategy boosted mutative whale-inspired optimization approaches, Applied Mathematical Modelling 73 (2019) 109 – 123. doi:https://doi.org/10.1016/j.apm.2019.03.046.
URL http://www.sciencedirect.com/science/article/pii/S0307904X19301908

[15] A. A. Heidari, I. Aljarah, H. Faris, H. Chen, J. Luo, S. Mirjalili, An enhanced associative learning-based exploratory whale optimizer for global optimization, Neural Computing and Applicationsdoi:10.1007/s00521-019-04015-0.
URL https://doi.org/10.1007/s00521-019-04015-0

[16] Y. Xu, H. Chen, A. A. Heidari, J. Luo, Q. Zhang, X. Zhao, C. Li, An efficient chaotic mutative moth-flame-inspired optimizer for global optimization tasks, Expert Systems with Applications 129 (2019) 135 – 155. doi:https://doi.org/10.1016/j.eswa.2019.03.043.
URL http://www.sciencedirect.com/science/article/pii/S0957417419302179

[17] I. Aljarah, M. Mafarja, A. A. Heidari, H. Faris, Y. Zhang, S. Mirjalili, Asynchronous accelerating multi-leader salp chains for feature selection, Applied Soft Computing 71 (2018) 964–979. doi:10.1016/j.asoc.2018.07.040.

[18] Q. Zhang, H. Chen, A. A. Heidari, X. Zhao, Y. Xu, P. Wang, Y. Li, C. Li, Chaos-induced and mutation-driven schemes boosting salp chains-inspired optimizers, IEEE Access 7 (2019) 31243–31261. doi:10.1109/ACCESS.2019.2902306.

[19] R. Poli, W. B. Langdon, N. F. McPhee, A field guide to genetic programming, Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008, (With contributions by J.

R. Koza).
URL http://www.gp-field-guide.org.uk

[20] C. Ferreira, Gene expression programming: a new adaptive algorithm for solving problems, Complex Systems 13 (2) (2001) 87–129.
URL http://arxiv.org/abs/cs/0102027

[21] J. F. Miller, P. Thomson, Cartesian genetic programming, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 1802, 2000, pp. 121–132. arXiv:arXiv:1011.1669v3, doi:10.1007/978-3-540-46239-2_9.

[22] P. Espejo, S. Ventura, F. Herrera, A survey on the application of genetic programming to classification, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 40 (2) (2010) 121–144. doi:10.1109/tsmcc.2009.2033566.
URL https://doi.org/10.1109/tsmcc.2009.2033566

[23] J. R. Koza, Human-competitive results produced by genetic programming, Genetic Programming and Evolvable Machines 11 (3-4) (2010) 251–284. doi:10.1007/s10710-010-9112-3.
URL http://link.springer.com/10.1007/s10710-010-9112-3

[24] E. K. Burke, M. R. Hyde, G. Kendall, J. Woodward, Automatic heuristic generation with genetic programming, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07, ACM Press, New York, New York, USA, 2007, p. 1559. doi:10.1145/1276958.1277273.
URL http://portal.acm.org/citation.cfm?doid=1276958.1277273

[25] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. R. Woodward, Exploring Hyper-heuristic Methodologies with Genetic Programming, Computational Intelligence 1 (2009) 177–201. doi:doi:10.1007/978-3-642-01799-5_6.
URL http://www.cs.nott.ac.uk/~gxo/papers/ChapterGPasHH09.pdf

[26] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724. doi:10.1057/jors.2013.71.
URL http://link.springer.com/10.1057/jors.2013.71

[27] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. R. Woodward, A classification of hyper-heuristic approaches: Revisited, in: Handbook of Metaheuristics, Springer International Publishing, 2018, pp. 453–477. doi:10.1007/978-3-319-91086-4_14.
URL https://doi.org/10.1007/978-3-319-91086-4_14

[28] J. H. Drake, A. Kheiri, E. Özcan, E. K. Burke, Recent advances in selection hyper-heuristics, European Journal of Operational Research 285 (2) (2020) 405 – 428. doi:https://doi.org/10.1016/j.ejor.2019.07.073.
URL http://www.sciencedirect.com/science/article/pii/S0377221719306526

[29] R. Qu, G. Kendall, N. Pillay, The general combinatorial optimization problem: Towards automated algorithm design, IEEE Computational Intelligence Magazine 15 (2) (2020) 14–23. doi:10.1109/mci.2020.2976182.
URL https://doi.org/10.1109/mci.2020.2976182

[30] J. MacLachlan, Y. Mei, J. Branke, M. Zhang, Genetic programming hyper-heuristics with vehicle collaboration for uncertain capacitated arc routing problems, Evolutionary Computation 0 (0) (2019) 1–31, pMID: 31730372. doi:10.1162/evco_a_00267.
URL https://doi.org/10.1162/evco_a_00267

[31] S. Wang, Y. Mei, M. Zhang, Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2019. doi:10.1145/3321707.3321797.
URL https://doi.org/10.1145/3321707.3321797

[32] Y. Liu, Y. Mei, M. Zhang, Z. Zhang, A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem, Evolutionary Computation 28 (2) (2020) 289–316.

[33] N. Pillay, W. Banzhaf, A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem, European Journal of Operational Research 197 (2) (2009) 482 – 491. doi:https://doi.org/10.1016/j.ejor.2008.07.023.
URL http://www.sciencedirect.com/science/article/pii/S0377221708005638

[34] N. Pillay, A review of hyper-heuristics for educational timetabling, Annals of Operations Research 239 (1) (2014) 3–38. doi:10.1007/s10479-014-1688-1.
URL https://doi.org/10.1007/s10479-014-1688-1

[35] X. Hao, R. Qu, J. Liu, A unified framework of graph-based evolutionary multitasking hyper-heuristic, IEEE Transactions on Evolutionary Computation (2020) 1–1.

[36] B. Chen, R. Qu, R. Bai, W. Laesanklang, A hyper-heuristic with two guidance indicators for bi-objective mixed-shift vehicle routing problem with time windows, Applied Intelligence 48 (12) (2018) 4937–4959. doi:

10.1007/s10489-018-1250-y.
URL https://doi.org/10.1007/s10489-018-1250-y

[37] P. P. Oteiza, D. A. Rodríguez, N. B. Brignole, Parallel hyperheuristic algorithm for the design of pipeline networks, Industrial & Engineering Chemistry Research 57 (42) (2018) 14307–14314. doi:10.1021/acs.iecr.8b02818.
URL https://doi.org/10.1021/acs.iecr.8b02818

[38] J. H. Drake, M. Hyde, K. Ibrahim, E. Ozcan, A genetic programming hyper-heuristic for the multidimensional knapsack problem, Kybernetes 43 (9/10) (2014) 1500–1511. doi:10.1108/k-09-2013-0201.
URL https://doi.org/10.1108/k-09-2013-0201

[39] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, Journal of Parallel and Distributed Computing 59 (2) (1999) 107–131. doi:10.1006/jpdc.1999.1581.

[40] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund, 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) (2001) 810–837. doi:10.1006/jpdc.2000.1714.

[41] M. Durasević, D. Jakobović, Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment, Genetic Programming and Evolvable Machines 19 (1) (2018) 53–92. doi:10.1007/s10710-017-9302-3.
URL https://doi.org/10.1007/s10710-017-9302-3

[42] J. Branke, S. Nguyen, C. W. Pickardt, M. Zhang, Automated Design of Production Scheduling Heuristics: A Review, IEEE Transactions on Evolutionary Computation 20 (1) (2016) 110–124. doi:10.1109/TEVC.2015.2429314.
URL http://ieeexplore.ieee.org/document/7101236/

[43] S. Nguyen, Y. Mei, M. Zhang, Genetic programming for production scheduling: a survey with a unified framework, Complex & Intelligent Systems 3 (1) (2017) 41–66. doi:10.1007/s40747-017-0036-x.

[44] M. Đumić, D. Šišejković, R. Čorić, D. Jakobović, Evolving priority rules for resource constrained project scheduling problem with genetic programming, Future Generation Computer Systems 86 (2018) 211 – 221. doi:https://doi.org/10.1016/j.future.2018.04.029.
URL http://www.sciencedirect.com/science/article/pii/S0167739X1732441X

[45] C. Dimopoulos, A. Zalzala, A genetic programming heuristic for the one-machine total tardiness problem, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), IEEE, 1999, pp. 2207–2214. doi:10.1109/CEC.1999.785549.
URL http://ieeexplore.ieee.org/document/785549/

[46] K. Miyashita, Job-shop scheduling with genetic programming, in: Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation, GECCO'00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp. 505–512.
URL http://dl.acm.org/citation.cfm?id=2933718.2933809

[47] J. C. Tay, N. B. Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, Computers & Industrial Engineering 54 (3) (2008) 453–473. doi:10.1016/j.cie.2007.08.008.
URL http://linkinghub.elsevier.com/retrieve/pii/S0360835207002008

[48] D. Jakobović, K. Marasović, Evolving priority scheduling heuristics with genetic programming, Applied Soft Computing 12 (9) (2012) 2781–2789. doi:10.1016/j.asoc.2012.03.065.
URL http://linkinghub.elsevier.com/retrieve/pii/S1568494612001780

[49] Wen-Jun Yin, Min Liu, Cheng Wu, Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming, in: The 2003 Congress on Evolutionary Computation, 2003. CEC '03., Vol. 2, IEEE, 2003, pp. 1050–1055. doi:10.1109/CEC.2003.1299784.
URL http://ieeexplore.ieee.org/document/1299784/

[50] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Zhang, Investigating the generality of genetic programming based hyper-heuristic approach to dynamic job shop scheduling with machine breakdown, in: M. Wagner, X. Li, T. Hendtlass (Eds.), Artificial Life and Computational Intelligence: Third Australasian Conference, ACALCI 2017, Geelong, VIC, Australia, January 31 – February 2, 2017, Proceedings, Springer International Publishing, Cham, 2017, pp. 301–313. doi:10.1007/978-3-319-51691-2_26.
URL https://doi.org/10.1007/978-3-319-51691-2_26

[51] L. Nie, X. Shao, L. Gao, W. Li, Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems, The International Journal of Advanced Manufacturing Technology 50 (5-8) (2010) 729–747. doi:10.1007/s00170-010-2518-5.
URL http://link.springer.com/10.1007/s00170-010-2518-5

[52] L. Nie, L. Gao, P. Li, L. Zhang, Application of gene expression programming on dynamic job shop scheduling problem, in: Proceedings

of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD), IEEE, 2011, pp. 291–295. doi:10.1109/CSCWD.2011.5960088.
URL http://ieeexplore.ieee.org/document/5960088/

[53] J. Branke, T. Hildebrandt, B. Scholz-Reiter, Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations, Evolutionary Computation 23 (2) (2015) 249–277. doi:10.1162/EVCO_a_00131.

[54] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, 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) (2013) 621–639. doi:10.1109/TEVC.2012.2227326.
URL http://ieeexplore.ieee.org/document/6353198/

[55] S. Nguyen, Y. Mei, B. Xue, M. Zhang, A hybrid genetic programming algorithm for automated design of dispatching rules, Evolutionary Computation 27 (3) (2019) 467–496. doi:10.1162/evco_a_00230.

[56] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Dynamic multi-objective job shop scheduling: A genetic programming approach, in: A. S. Uyar, E. Ozcan, N. Urquhart (Eds.), Automated Scheduling and Planning: From Theory to Practice, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 251–282. doi:10.1007/978-3-642-39304-4_10.
URL https://doi.org/10.1007/978-3-642-39304-4_10

[57] S. Nguyen, M. Zhang, K. C. Tan, Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 2781–2788. doi:10.1109/CEC.2015.7257234.
URL http://ieeexplore.ieee.org/document/7257234/

[58] F. Zhang, Y. Mei, M. Zhang, Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics, in: 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 1366–1373.

[59] A. Masood, G. Chen, Y. Mei, H. Al-Sahaf, M. Zhang, Genetic programming with pareto local search for many-objective job shop scheduling, in: J. Liu, J. Bailey (Eds.), AI 2019: Advances in Artificial Intelligence, Springer International Publishing, Cham, 2019, pp. 536–548.

[60] M. Durasević, D. Jakobović, Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment, Genetic Programming and Evolvable Machinesdoi:10.1007/s10710-017-9310-3.
URL https://doi.org/10.1007/s10710-017-9310-3

[61] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Automatic Design of Scheduling Policies for Dynamic Multi-objective Job Shop Scheduling via Cooperative Coevolution Genetic Programming, IEEE Transactions on Evolutionary Computation 18 (2) (2014) 193–208. doi:10.1109/TEVC.2013.2248159.
URL http://ieeexplore.ieee.org/document/6468087/

[62] Y. Mei, S. Nguyen, M. Zhang, Evolving time-invariant dispatching rules in job shop scheduling with genetic programming, in: J. McDermott, M. Castelli, L. Sekanina, E. Haasdijk, P. García-Sánchez (Eds.), Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Springer International Publishing, Cham, 2017, pp. 147–163. doi:10.1007/978-3-319-55696-3_10.
URL https://doi.org/10.1007/978-3-319-55696-3_10

[63] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling, in: L. Paquete, C. Zarges (Eds.), Evolutionary Computation in Combinatorial Optimization, Springer International Publishing, Cham, 2020, pp. 214–230.

[64] D. Yska, Y. Mei, M. Zhang, Feature construction in genetic programming hyper-heuristic for dynamic flexible job shop scheduling, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 149–150. doi:10.1145/3205651.3205741.
URL https://doi.org/10.1145/3205651.3205741

[65] F. Zhang, Y. Mei, M. Zhang, Genetic programming with multi-tree representation for dynamic flexible job shop scheduling, in: T. Mitrovic, B. Xue, X. Li (Eds.), AI 2018: Advances in Artificial Intelligence, Springer International Publishing, Cham, 2018, pp. 472–484.

[66] F. Zhang, Y. Mei, M. Zhang, A new representation in genetic programming for evolving dispatching rules for dynamic flexible job shop scheduling, in: A. Liefooghe, L. Paquete (Eds.), Evolutionary Computation in Combinatorial Optimization, Springer International Publishing, Cham, 2019, pp. 33–49.

[67] D. Karunakaran, Y. Mei, G. Chen, M. Zhang, Dynamic Job Shop Scheduling Under Uncertainty Using Genetic Programming, in: G. Leu, H. K. Singh, S. Elsayed (Eds.), Intelligent and Evolutionary Systems: The 20th Asia Pacific Symposium, IES 2016, Canberra, Australia, November 2016, Proceedings, Springer International Publishing, Cham, 2017, pp. 195–210. doi:10.1007/978-3-319-49049-6_14.
URL http://link.springer.com/10.1007/978-3-319-49049-6_14

[68] D. Karunakaran, Yi Mei, Gang Chen, Mengjie Zhang, Evolving dispatching rules for dynamic Job shop scheduling with uncertain processing times, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 364–371. doi:10.1109/CEC.2017.7969335.
URL http://ieeexplore.ieee.org/document/7969335/

[69] D. Karunakaran, Y. Mei, G. Chen, M. Zhang, Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty, in: Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '17, ACM Press, New York, New York, USA, 2017, pp. 282–289. doi:10.1145/3071178.3071202.
URL http://dl.acm.org/citation.cfm?doid=3071178.3071202

[70] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, B. Scholz-Reiter, Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems, International Journal of Production Economics 145 (1) (2013) 67–77. doi:10.1016/j.ijpe.2012.10.016.

[71] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Johnston, M. Zhang, Genetic Programming Based Hyper-heuristics for Dynamic Job Shop Scheduling: Cooperative Coevolutionary Approaches, in: M. I. Heywood, J. McDermott, M. Castelli, E. Costa, K. Sim (Eds.), Genetic Programming: 19th European Conference, EuroGP 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Springer International Publishing, Cham, 2016, pp. 115–132. doi:10.1007/978-3-319-30668-1_8.
URL http://link.springer.com/10.1007/978-3-319-30668-1_8

[72] E. Hart, K. Sim, A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling, Evolutionary Computation 24 (4) (2016) 609–635. doi:10.1162/EVCO_a_00183.
URL http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00183

[73] M. Djurasević, D. Jakobović, Creating dispatching rules by simple ensemble combination, Journal of Heuristicsdoi:10.1007/s10732-019-09416-x.
URL https://doi.org/10.1007/s10732-019-09416-x

[74] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Zhang, An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling, Applied Soft Computing 63 (2018) 72 – 86. doi:https://doi.org/10.1016/j.asoc.2017.11.020.
URL http://www.sciencedirect.com/science/article/pii/S156849461730683X

[75] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Selection Schemes in Surrogate-Assisted Genetic Programming for Job Shop Scheduling, in: G. Dick, W. N. Browne, P. Whigham, M. Zhang, L. T. Bui,

H. Ishibuchi, Y. Jin, X. Li, Y. Shi, P. Singh, K. C. Tan, K. Tang (Eds.), Simulated Evolution and Learning: 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings, Springer International Publishing, Cham, 2014, pp. 656–667. doi:10.1007/978-3-319-13563-2_55.
URL http://link.springer.com/10.1007/978-3-319-13563-2_55

[76] S. Nguyen, M. Zhang, K. C. Tan, Surrogate-Assisted Genetic Programming With Simplified Models for Automated Design of Dispatching Rules, IEEE Transactions on Cybernetics (2016) 1–15doi:10.1109/TCYB.2016.2562674.
URL http://ieeexplore.ieee.org/document/7473913/

[77] Y. Mei, M. Zhang, A comprehensive analysis on reusability of GP-evolved job shop dispatching rules, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 3590–3597. doi:10.1109/CEC.2016.7744244.
URL http://ieeexplore.ieee.org/document/7744244/

[78] S. Nguyen, M. Zhang, D. Alahakoon, K. C. Tan, Visualizing the evolution of computer programs for genetic programming [research frontier], IEEE Computational Intelligence Magazine 13 (4) (2018) 77–94. doi:10.1109/MCI.2018.2866731.

[79] S. Nguyen, M. Zhang, D. Alahakoon, K. C. Tan, People-centric evolutionary system for dynamic production scheduling, IEEE Transactions on Cybernetics (2019) 1–14.

[80] S. Nguyen, M. Zhang, K. C. Tan, Adaptive charting genetic programming for dynamic flexible job shop scheduling, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 1159–1166. doi:10.1145/3205455.3205531.
URL https://doi.org/10.1145/3205455.3205531

[81] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Guided subtree selection for genetic operators in genetic programming for dynamic flexible job shop scheduling, in: T. Hu, N. Lourenço, E. Medvet, F. Divina (Eds.), Genetic Programming, Springer International Publishing, Cham, 2020, pp. 262–278.

[82] D. Karunakaran, Y. Mei, G. Chen, M. Zhang, Active sampling for dynamic job shop scheduling using genetic programming, in: 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 434–441.

[83] F. J. Gil-Gala, C. Mencía, M. R. Sierra, R. Varela, Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time, Applied Soft Computing 85 (2019) 105782. doi:https://doi.org/10.1016/j.asoc.2019.105782.

[84] F. J. Gil-Gala, C. Mencía, M. R. Sierra, R. Varela, Learning ensembles of priority rules for online scheduling by hybrid evolutionary algorithms, Integrated Computer-Aided Engineering (2020) 1–16doi:10.3233/ICA-200634.
URL http://doi.org/10.3233/ICA-200634

[85] F. J. Gil-Gala, M. R. Sierra, C. Mencía, R. Varela, Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling, Natural Computingdoi:10.1007/s11047-020-09793-4.
URL https://doi.org/10.1007/s11047-020-09793-4

[86] I. Vlašić, M. Đurasević, D. Jakobović, Improving genetic algorithm performance by population initialisation with dispatching rules, Computers & Industrial Engineering 137 (2019) 106030. doi:https://doi.org/10.1016/j.cie.2019.106030.

[87] D. Jakobović, L. Budin, Dynamic scheduling with genetic programming, in: P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.), Genetic Programming: 9th European Conference, EuroGP 2006, Budapest, Hungary, April 10-12, 2006. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 73–84. doi:10.1007/11729976_7.
URL https://doi.org/10.1007/11729976_7

[88] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Learning iterative dispatching rules for job shop scheduling with genetic programming, The International Journal of Advanced Manufacturing Technology 67 (1-4) (2013) 85–100. doi:10.1007/s00170-013-4756-9.
URL http://link.springer.com/10.1007/s00170-013-4756-9

[89] J. Park, S. Nguyen, M. Zhang, M. Johnston, Genetic programming for order acceptance and scheduling, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 1005–1012. doi:10.1109/CEC.2013.6557677.
URL http://ieeexplore.ieee.org/document/6557677/

[90] S. Nguyen, M. Zhang, M. Johnston, A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1824–1831. doi:10.1109/CEC.2014.6900347.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6900347

[91] D. Jakobović, L. Jelenković, L. Budin, Genetic Programming Heuristics for Multiple Machine Scheduling, in: Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 321–330. doi:10.1007/978-3-540-71605-1_30.
URL http://link.springer.com/10.1007/978-3-540-71605-1_30

[92] M. Đurasević, D. Jakobović, K. Knežević, Adaptive scheduling on unrelated machines with genetic programming, Applied Soft Computing 48 (2016) 419–430. doi:10.1016/j.asoc.2016.07.025.

[93] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Genetic Programming for Job Shop Scheduling, Springer International Publishing, Cham, 2019, pp. 143–167. doi:10.1007/978-3-319-91341-4_8.
URL https://doi.org/10.1007/978-3-319-91341-4_8

[94] L. Fanjul-Peyro, R. Ruiz, Scheduling unrelated parallel machines with optional machines and jobs selection, Computers & Operations Research 39 (7) (2012) 1745–1753. doi:10.1016/j.cor.2011.10.012.

[95] J.-H. Lee, J.-M. Yu, D.-H. Lee, 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) (2013) 2081–2089. doi:10.1007/s00170-013-5192-6.

[96] I.-L. Wang, Y.-C. Wang, C.-W. Chen, Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics, Flexible Services and Manufacturing Journal 25 (3) (2013) 343–366. doi:10.1007/s10696-012-9150-7.
URL http://link.springer.com/10.1007/s10696-012-9150-7

[97] Y. Unlu, S. J. Mason, Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems, Computers & Industrial Engineering 58 (4) (2010) 785 – 800. doi:https://doi.org/10.1016/j.cie.2010.02.012.
URL http://www.sciencedirect.com/science/article/pii/S0360835210000483

[98] N. Pillay, R. Qu, Hyper-Heuristics: Theory and Applications, Springer International Publishing, 2018. doi:10.1007/978-3-319-96514-7.
URL https://doi.org/10.1007/978-3-319-96514-7

[99] L. Nie, Y. Bai, X. Wang, K. Liu, Discover Scheduling Strategies with Gene Expression Programming for Dynamic Flexible Job Shop Scheduling Problem, in: Y. Tan, Y. Shi, Z. Ji (Eds.), Advances in Swarm Intelligence: Third International Conference, ICSI 2012, Shenzhen, China, June 17-20, 2012 Proceedings, Part II, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 383–390. doi:10.1007/978-3-642-31020-1_45.
URL http://link.springer.com/10.1007/978-3-642-31020-1_45

[100] L. Nie, L. Gao, P. Li, X. Li, A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates, Journal of Intelligent Manufacturing 24 (4) (2013) 763–774. doi:10.1007/s10845-012-0626-9.
URL http://link.springer.com/10.1007/s10845-012-0626-9