Using Priority Rules for Resource-Constrained Project Scheduling Problem in Static Environment

Mateja Đumić[a,*], Domagoj Jakobović[b]

[a] *Department of Mathematics, University of Osijek, Trg Ljudevita Gaja 6, Osijek, Croatia*
[b] *Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia*

Abstract

The resource-constrained project scheduling problem (RCPSP) is one of the scheduling problems that belong to the class of NP -hard problems. Therefore, heuristic approaches are usually used to solve it. One of the most commonly used heuristic approaches are priority rules (PRs). PRs are easy to use, fast and able to respond to system changes, which makes them applicable in a dynamic environment. The disadvantage of PRs is that when applied in a static environment, they do not achieve results of the same quality as heuristic approaches designed for a static environment. Moreover, a new PR must be evolved separately for each optimization criterion, which is a challenging process. Therefore, recently significant effort has been put into the automatic development of PRs. Although PRs are mainly used in a dynamic environment, they are also used in a static environment in situations where speed and simplicity are more important than the quality of the obtained solution. Since PRs evolved for the dynamic environment do not use all the information available in a static environment, this paper analyzes two adaptations for evolving PRs for the RCPSP - iterative priority rules and rollout approach. This paper shows that these approaches achieve better results than the PRs evolved and used without these adaptations. The results of the approaches presented in the paper were also compared with the results obtained with the genetic algorithm as a representative of the heuristic approaches used mainly in the static environment.

Keywords: genetic programming, resource constrained project scheduling problem, priority rules, iterative priority rules, rollout, static environment

1. Introduction

The resource-constrained project scheduling problem (RCPSP) is one of the complex constrained scheduling problems that occur in daily life and attract

*Corresponding author
Email addresses: mdjumic@mathos.hr (Mateja Đumić), domagoj.jakobovic@fer.hr (Domagoj Jakobović)

numerous researchers. In RCPSP, there are two types of constraints - precedences and resources. The precedence constraints specify which activities take place before and after each activity, while the resource constraints contain information about the resources needed to execute the activities and the amount of resources available in each time period.

Usually, solving methods in combinatorial optimization can be divided into two groups: exact algorithms and heuristics. The solution space for scheduling problems increases dramatically as the size of the problem increases, making exact methods impractical. Therefore, heuristic methods are mostly used to solve the RCPSP.

Heuristic methods used to solve RCPSP can be divided into two groups - constructive and improvement heuristics. Constructive heuristics start with an empty schedule and build a complete schedule through iterations by adding one or more activities in each iteration. The representatives of constructive heuristics are priority rules (PRs). Unlike constructive heuristics, improvement heuristics start with an initial solution that is then improved through iterations. Improvement heuristics include various evolutionary and population-based meta-heuristics such as genetic algorithm (GA) (Gargiulo & Quagliarella, 2012; Hindi et al., 2002; Alcaraz et al., 2003; Kadam & Kadam, 2014), ant colony optimization (ACO) (Merkle et al., 2002), and simulated annealing (SA) (Valls & Ballestín, 2004).

Heuristic methods may also differ in whether they are applicable in a static or dynamic environment, or both. A static environment is one in which all information about the activities in the project (duration, resource requirements, release time, due date) and the resources needed to schedule those activities are known in advance and cannot be changed during the creation of the schedule or the execution of the activities. In contrast, in a dynamic environment, not all information is available or known from the beginning and can change during planning or execution.

PRs are a heuristic approach that can respond to changes in dynamic environments and generally generate schedules very quickly. Therefore, they are very commonly used in a dynamic environment. The disadvantage of these methods is that they usually lead to solutions that are of worse quality than the solutions found by improvement heuristics. Moreover, a separate PR must be applied for each optimization criterion. Therefore, recent research has placed great emphasis on the automatic development of PRs. Evolutionary computation methods are most commonly used in the development of new PRs, especially genetic programming (GP). PRs evolved by GP have performed well in numerous scheduling environments (Jakobović & Budin, 2006; Jakobović & Marasović, 2012; Nguyen et al., 2013; Hunt et al., 2014; Đurasević et al., 2016). In most of the studies, PRs were designed for a dynamic environment. Although the results obtained with PRs in a static environment are worse than the results obtained with an improvement heuristic, PRs can also be used in static environments.

Typically, PRs are used in a static environment in situations where the time required to find a solution is more important than the quality of the solution.

Also, the ability to react to changes in the system is sometimes helpful and necessary in a static environment where all information is available and known, but changes can still occur, for example, when a machine fails. In such situations, the improvement heuristics must start from the beginning, while the PR continues from the moment of change. For this reason, the question arises whether there are ways to use more information available in a static environment to improve the quality of solutions while maintaining the speed of solution finding when using PRs. Such adaptations of PRs are found only in a small number of works (Hildebrandt et al., 2010; Nguyen et al., 2013; Đurasević et al., 2016). Recently, an adaptation for the unrelated machine environment has been analyzed in Đurasević & Jakobović (2020). In this paper, we analyze some adaptations of the evolving process of PRs for the resource-constrained project scheduling problems - RCPSP. So far, the adaptation of PRs for RCPSP in static environments has been analyzed only in a single paper (Chand et al., 2019), which leaves many questions unanswered. Accordingly, this paper analyzes previously unused approaches in RCPSP and static environments as well as existing approaches that are less time consuming than those used in Chand et al. (2019).

The main contributions of this paper are:

- two approaches for adaptation of automatically evolved priority rules for RCPSP for a static environment
- comparison of the proposed approaches with dynamic priority rules evolved with genetic programming and solutions obtained by the genetic algorithm as a representative of improvement metaheuristics
- time analysis of the compared approaches and methods.

The rest of the paper is organized as follows. Problem definition and related work, divided into three parts, are given in Section 2. In the first part of the section, the definition of RCPSP is given; in the second part, genetic programming is briefly explained, while in the third part, the evolution process of PRs for RCPSP is described. In Section 3, two adaptation approaches for PRs used in a static environment are described in detail. Section 4 describes the experimental setup and Section 5 presents and discusses the results of the experiments. Finally, Section 6 concludes the research conducted in this work and provides new directions for future investigations.

2. Problem Definition and Related Work

2.1. RCPSP

The resource-constrained scheduling problem (RCPSP) is an NP -hard combinatorial optimization problem (Blazewicz et al., 1983) given as follows: there are n activities that need to be scheduled and m resources available and needed for the execution of the given activities. The resources needed for execution can be renewable, nonrenewable, and doubly constrained (Słowinski, 1981). Resources are renewable if they are available in the same quantity in each time period, nonrenewable if their quantity is reduced by use, and doubly constrained if their quantity is limited in each time period and for the entire project. The time and quantity of each resource required to execute an activity are known, as well as the set of activities that must be completed before it starts with execution. Solving RCPSP means finding a schedule that satisfies all given constraints while optimizing one or more criteria. In the basic formulation, minimizing the completion time of the last activity (makepan) is usually used as the optimization criterion, i.e., the total duration of the project. The basic formulation implies no preemption, which means that the activity cannot be interrupted once it has started its execution. In addition, only renewable resources are considered.

Formally, the basic formulation of the RCPSP can be given with the tuple (Artigues et al., 2008):

$$RCPSP = (A, E, p, R, B, D, c). \tag{1}$$

In the above definition, $A = \{0, \dots, n+1\}$ represents the set of activities. The activities 0 and $n+1$ are usually called dummy activities and represent the beginning and the end of the project, respectively. The duration of the activities in the set A is given by the vector $p \in \mathbb{N}_0^{n+2}$. In particular, dummy activities have a duration of 0.

The set E consists of pairs of activities (i, j) representing precedence constraints. In each pair, the second activity is the successor of the first, which means that the second activity cannot be executed until the first is finished. By convention, the $n+1$ activity follows all other activities in the project.

The resources that are renewable in this formulation are given by the set $R = \{1, \dots, m\}$ and their available quantities in each time period are given by the vector $B \in \mathbb{N}^m$. The activities' demand for each available resource is given by the matrix $D \in \mathbb{N}^{(n+1)\times m}$. Dummy activities have no demand for resources. Consequently, the first and last rows of the matrix D consist of zeros.

The last member of the tuple is the objective function, denoted by $c : \chi \subset \mathbb{R}^{n+2} \to \mathbb{R}$, where χ is the set of all feasible schedules. If the time span is used as an objective function, it can be omitted in the tuple. The goal is to find the feasible schedule $S \in \chi$ with the best value of the objective function. The vector S consists of the start times of each activity in the project. More precisely, the i-th component S_i of S represents the start time of the activity i. The schedule is feasible if all constraints (precedence and resources) are satisfied, which can be formulated as follows:

$$S_j - S_i \geq p_i \quad \forall (i, j) \in E \tag{2}$$

$$\sum_{i \in A_t} D_{ij} \leq B_j \quad \forall t \geq 0, \forall j \in R \tag{3}$$

where $A_t = \{i \in A \mid S_i \leq t \leq S_i + p_i\}$ is the set of activities that are active at the given time t.

The basic formulation of this problem can be extended by introducing additional constraints, such as the time at which certain activities become available (not all activities may be available at the time the project starts), the possibility of interrupting an activity, and similiar . For more details on this problem, see Artigues et al. (2008); Blazewicz et al. (1983); Hartmann & Briskorn (2010) and Abdolshah (2014).

2.2. GP

Genetic programming (GP) is an evolutionary computing algorithm similar to the genetic algorithm (GA). The main difference between GP and GA is the representation of the individual. Unlike GA, the individuals in GP do not have to have a precise form known in advance (Poli et al.). The individuals can be programs, expressions, or mathematical functions that represent a solution to a particular problem (Poli et al.; Koza, 1992).

Based on foundation given by Koza (1990) GP is becoming a widely used technique in numerous optimization (Koza, 1992) and classification (Espejo et al., 2010) problems. The results obtained by GP are mostly competitive with human-made ones(Koza et al., 1996). The good performance and the ability to build a complex structure from simple structures make this technique ideal for the hyperheuristic approach (Burke et al., 2009, 2010, 2013). In the literature, GP has been used as a hyperheuristic in problems such as bin packing (Kumar et al., 2008; Özcan & Parkes, 2011; Burke et al., 2012), project scheduling (Frankola et al., 2008; Đumić et al., 2018; Chand et al., 2018), timetabling (Pillay, 2012; Bader-El-Den et al., 2009), and the vehicle routing problem (Oltean & Dumitrescu, 2004; Beham et al., 2009; Vonolfen et al., 2013).

GP uses a population of individuals where each individual represents a solution. The procedure of GP starts with initializing the population, which can be done randomly or with some other approach. After initialization, new individuals for the next generation are generated from the existing individuals and genetic operators. The goal is to improve the fitness of the individuals in the population in each generation. The GP procedure is described with the algorithm 1

Algorithm 1 genetic programming

```
1: initialize population
2: do
3:     select individuals based on fitness
4:     generate new individuals from the selected ones using genetic operators
5: while stopping criteria are not met
6: return best individual as solution
```

In GP it is important to choose a good solution representation. The solution representation must be able to build complex structures, and at the same time it must allow a genetic operator to easily make changes. Therefore, the tree representation is usually used as the solution representation. For the tree representation, it is important to define the primitive set from which the tree nodes are selected. The primitive set consists of two subsets: terminal set and function set. The terminal set consists of variables and constants, and elements from it can only be placed in the leaf nodes of the tree. The function set consists of expressions, operators and functions, and its elements are placed in non-leaf nodes. To achieve good results with GP, it is important that these two sets consist of different elements. It is also important that these sets are not too large (because the number of elements increases the solution space). At the same time, they must contain all the essential properties of the problem to form a structure good enough to represent the solution of the problem. For more details about GP and genetic operators, see Poli et al. and Koza (1992).

2.3. Evolving PRs for the RCPSP

The RCPSP is an NP-hard problem, and heuristic methods are usually used to solve it. Priority rules (PRs) are a representative of constructive heuristic methods that build schedules from scratch by iteration. This method is fast and straightforward, which makes it applicable in dynamic environments. PRs are created to solve multiple instances, which leads to the fact that heuristics that solve one problem instance at a time usually achieve better results than PRs. However, in real life, there are many situations where simplicity and speed are more important than quality, resulting in PRs being used in many situations. PRs are primarily used in dynamic environments, but they can also be used in a static environment.

PRs for scheduling problems, including RCPSP, are used in combination with the Scheduling Generation Scheme - SGS. Based on the priority assigned to each activity, the SGS decides which activity to schedule next, ensuring that all constraints are met. SGS builds the schedule through iterations. In RCPSP, we distinguish two versions of SGS: parallel and serial. Parallel SGS (PSGS) iterates by time intervals, while serial SGS (SSGS) iterates by activities. Therefore, the number of iterations may differ between the different types of SGS. For SSGS, the number of iterations corresponds to the number of activities that need to be scheduled, and for PSGS, it corresponds to the number of time periods in which some of the activities have been scheduled. In practice, the results obtained with PSGS are usually of better quality than those obtained with SSGS. For more on SSGS and PSGS, see Kolisch (1996) and Kolisch & Hartmann (1999).

The PR used in SGS can be defined manually or created using an automated development method. Due to the need to develop a different PR for each criterion and the complexity of creating new PRs, methods and procedures for automated development have recently been studied extensively. The most commonly used method for developing PRs is GP.

The development of PRs using GP for the RCPSP is first mentioned in the work of Frankola et al. (2008) and has been studied more extensively in recent years (Đumić et al., 2018; Chand et al., 2018; Chand, 2018; Chand et al., 2019; Đumić & Jakobović, 2021). Recently, PRs have been used for extended versions of RCPSP such as multi-skill RCPSP (Lin et al., 2020) and stochastic RCPSP

(Chen et al., 2020). The literature also contains research on how to select which PR to use (Guo et al., 2020; Chakrabortty et al., 2020).

To evolve PRs using GP, it is necessary to define a primitive set - a set of terminals and functions used in the evolving process. In Frankola et al. (2008), only a small set of terminals and functions was used for evolving PRs, while these two sets were extended in researches conducted in Đumić et al. (2018) and Chand et al. (2018). The elements of the terminal and functional sets were not identical in these two papers, and in Chand et al. (2018) the values of the attributes were normalized to a range from 0 to 1. The experiments conducted in Đumić (2020) have shown that there is no significant difference between the results obtained with a terminal set from Đumić et al. (2018) and a terminal set from Chand et al. (2018), while the results obtained with the function set proposed in Chand et al. (2018) are better than the results obtained with the function set from Đumić et al. (2018). Moreover, these experiments have shown that the results obtained with normalized and non-normalized attributes are not significantly different. Since normalization in the experiments adds effort and does not improve the results, it does not need to be used. In Chand et al. (2018), different representations of individuals were analyzed and the experiments performed recommend the use of an arithmetic representation. Moreover, in Đumić et al. (2018) different schemes for generating schedules were tested and PSGS seems to be the best one for evolving PRs with GP for RCPSP. Various optimization functions have been used in the literature for evolving priority rules. Generally, optimization functions based on profit margin or deviation from the lower bound have been used. However, it is important to note that research in the literature has shown that PRs evolved using GP can compete with existing rules for other criteria such as total weighted completion time, equal resource utilization, and net present value.

Although much research has been done recently on the automatic development of PRs for the RCPSP (Đumić et al., 2018; Chand et al., 2018, 2019), most of the focus is still on rules that are applicable in a dynamic environment, while there is only one work (Chand et al., 2019) for a static environment. In Chand et al. (2019), the rollout justification procedure is used. This procedure consists of two parts - rollout and justification. In the rollout part, when deciding which activity to schedule next, all options are considered and the best one is selected. This process repeats until a complete schedule is created. After a schedule is created, iterative forward-backward planning is performed, and this part is called justification.

3. Adaptation approaches for evolving PRs for a static environment

This paper will analyze iterative priority rules (IPR) and the rollout approach to further improve evolved priority rules that can be used in a static environment. IPR builds a schedule multiple times, in such a way that each iteration uses information from the previous iteration's schedule to improve results. Unlike IPR, the rollout approach does not build the entire schedule from scratch multiple times, but instead considers different options at the decision point. The rollout approach considers multiple activities in each decision point. It determines the quality of the created schedule when each of them is selected, and selects the one that has achieved the best complete schedule by scheduling it next. This process continues until the last activity is scheduled.

3.1. IPR

IPRs are an approach in which a schedule is built through iterations by using information from the previous iteration to build a better schedule in the current iteration. Building the schedule is repeated until its quality stops improving, which determines the number of iterations. The adapted SGS used in this approach is given by Algorithm 2. In IPR, the adapted SGS is used both in the development of new PRs and in their application to solve new instances of the problem. It can be seen from the Algorithm 2 that the final schedule is not the last created schedule, but the schedule created in the penultimate iteration, since this is a schedule whose fitness is not worse than the fitness of the last created schedule.

Algorithm 2 SGS for scheduling with IPR

```
 1: Input: initial value $P_0$ for the parameter set $P$ whose values are remem-
    bered through iterations
 2: $P \leftarrow P_0$
 3: fitness $\leftarrow \infty$
 4: schedule$\leftarrow \emptyset$
 5: best_schedule$\leftarrow \emptyset$
 6: do
 7:     best_schedule$\leftarrow$ schedule
 8:     schedule$\leftarrow$ schedule built by using SGS and PR
 9:     fitness*$\leftarrow$ fitness
10:     fitness $\leftarrow$ fitness of built schedule
11:     calculate new values for the parameter set $P$ based on the built schedule
        and save it in variable $P$
12: while fitness* $>$ fitness
13: return best_schedule
```

Since the information from the previous iteration must be used when creating the schedule, it is necessary to ensure that it can be transferred from one schedule to another. The easiest way to transfer information is to include it in the PR. For this reason, it is necessary to introduce new terminals that will be used in rule development. By putting information in terminals, the information from the previous iteration can be included in the rule in a way that can have the greatest impact on quality. Terminals must be introduced depending on the problem being solved and the optimization criterion being used. The values of these terminals are calculated based on the schedule built in the previous iteration, so they must be initialized somehow in the first iteration. When initializing these values, it is important to choose values that (depending on the optimization criterion) are higher or lower than the values that can be reached in order to avoid termination of the evolution process in the first iteration.

3.2. Rollout approach

A rollout approach is a simple approach that can be applied to different heuristic methods to obtain better results (Bertsekas & Castanon, 1998; Bertsekas & Castanon, 1999; Bertsekas, 2013). This approach combines exhaustive search and heuristic methods. The main idea is to apply exhaustive search only in some parts of the decision space and at the same time achieve better results than heuristic methods and reduce the required time for finding a solution. To achieve this, all possibilities at the time of the decision are considered. However, unlike an exhaustive search, the process of determining which of these choices is the best does not proceed with an exhaustive search; instead, the remaining choices are made using a given heuristic. After the best option is selected based on the results obtained by the heuristic, scheduling continues by moving to the next decision point and repeating the previous procedure.

A version of this approach was used for the RCPSP in Chand et al. (2019), where significant improvements were achieved. However, in this work, the time for evolving PRs and solving new problems has increased significantly, which means the loss of one of the main advantages of PRs over other heuristic methods. Therefore, in this work, the main goal will be to minimize the time needed to evolve PRs and solve new instances. As a result, the quality of the results is expected to deteriorate, but at the same time, the results obtained are expected to be significantly better than those obtained by using the standard procedure for evolving PRs used in a dynamic environment. The adaptations of the evolution of PRs using the rollout approach in this paper are made based on Đurasević & Jakobović (2020). In Đurasević & Jakobović (2020), the adaptation is made for an unrelated machine environment, and in this adaptation, the speed of the PRs is preserved, and at the same time the results have been improved.

The pseudocode of the adapted PSGS used in this work is given by the Algorithm 3. In this algorithm, a schedule is created starting from an empty schedule. The procedure is repeated as long as there are unscheduled activities. Algorithm 3 is an adaptation of PSGS, which means that the iterations are done by time periods. In the algorithm, the first time period in which an activity can be scheduled must be found. If multiple activities can be scheduled, they are prioritized based on the specified PR and then sorted. A predetermined number of activities with the highest priority are then considered and the same number of schedules are created. The schedules are created by selecting one of the observed activities to be scheduled next. After scheduling this activity, the remaining activities are scheduled based on a priority rule and standard PSGS. Then, the procedure is repeated for each observed activity. Among the observed activities, the one whose scheduling resulted in the best schedule is selected. Suppose that the fitness value of the schedule created by scheduling the selected activity is greater than the fitness value of the previous best schedule. In this case, the selected activity is scheduled, otherwise the activity with the highest priority is scheduled. During this whole procedure, the best schedule is saved and returned as the final solution. The greater the number of activities considered in the decision-making process, the greater is the possibility of achieving a better result. On the other hand, as the number of activities increases, so does the time needed to create a schedule.

Algorithm 3 roll-out approach

```
1:  Input: $n$ - number of activities considered in decision making
           $\pi$ - priority rule
2:  time ← 0
3:  previous_fitness← ∞
4:  best_fitness ← ∞
5:  best_schedule ← ∅  STATE while there is unscheduled activities do
6:     schedule ← best_schedule
7:     set time to the next time an activity is available
8:     use $\pi$ to calculate the priority of available activities $A_i$
9:     sort the activities by given priority
10:    textbffor activity $A_i, i = 1, \ldots, n$ do
11:       add activity $A_i$ in schedule
12:       use PR $\pi$ to create the remaining part of the schedule
13:       fitness← fitness of built schedule
14:       if fitness < best_fitness then
15:          best_fitness←fitness
16:          activity*← $A_i$
17:       end
18:    end
19:    if best_fitness < previous_fitness then
20:       previous_fitness ← best_fitness
21:       add activity* to best_schedule
22:    textbfelse
23:       add highest priority activity in best_schedule
24:    end  STATE textbfend  STATE back best_schedule
```

In Chand et al. (2019), the adapted SGS was used both in the evolution of PRs and in solving new instances. In contrast, in this work, the customized SGS is only used to solve new instances, while the PRs are evolved using the standard PSGS. Therefore, the time needed to evolve the PRs will be greatly reduced, which will unfortunately affect the quality of the solution. However, on the other hand, the time required for execution remains reasonable. It should also be noted that if the evolved PRs already exist, they do not need to be evolved again to be used in combination with adapted SGS for new instances of the problem. For completeness of results, a comparison of the approach used in this paper with the approach used in Chand et al. (2019) is provided.

4. Experimental setup

The C++ programming language in combination with the ECF (Jakobović & et al., 2020) was used to implement the hyperheuristic approach of designing PRs for the RCPSP.

4.1. Data set

PRs will be evolved using instances from the Project scheduling problem library (PSPLIB) (Wittemann, 2020). PSPLIB consists of 2040 problem instances divided into 4 groups depending on the number of activities in the project. We distinguish groups of problem instances with 30, 60, 90 and 120 activities. Within each of these groups, problem instances were generated using ProGen and various parameters that affect the hardness of the problem. More about the set of instances and their generation can be found in Kolisch & Sprecher (1997) and Kolisch et al. (2001).

In this work, the learning and validation sets are used for the development of PRs. These sets consist of 10% of different problem instances from the PSPLIB set, i.e. 204 problem instances. The additional validation set (Validation Set 2) was used to set the parameters of the approaches used. This set also consists of 10% of different problem instances from the PSPLIB set. The test set consists of the remaining instances from PSPLIB, 1428 problem instances that were used to compare the proposed approaches. Problem instances from all 4 groups were equally represented in all sets used (depending on the size of the set), and all four sets are mutually disjoint. This division was made based on the research conducted in Đumić (2020).

4.2. GP parameters

The values of the parameters used by GP are given in Table 1, which are taken from Đumić (2020).

Table 1: Parameters for the GP

Parameter	Value
Number of generations	25
Population size	1024
Mutation probability	0.3
Maximum tree depth	5
Tournament size	7

4.3. Fitness functions

Two different fitness functions are used as criteria for the optimization, based on the works Đumić et al. (2018) and Chand et al. (2018). The first, F_1, is given by the expression:

$$F_1 = \frac{\sum_{i=1}^{N} \frac{C_i}{p_i^{avg} \cdot \sqrt{n_i}}}{N}, \tag{4}$$

where C_i is the achieved makespan of the i-th problem instance, p_i^{avg} is the average activity duration, n_i is the number of activities in the i-th problem instance, and N is the number of project instances in the set for which the fitness value is calculated. The second fitness function used is F_2, which is given by the following expression:

$$F_2 = \frac{1}{N}\left(\sum_{i=1}^{N}\left(\frac{C_i - L_i}{L_i}\cdot 100\right)\right), \tag{5}$$

where L_i is the lower bound for the i-th problem instance computed by omitting the resource constraints in the original problem. The function F_2 is expressed as a percentage.

The fitness functions defined above are used in the GP evolving process. In each iteration of GP, the fitness of individuals is calculated using the fitness function F_1 or F_2 and the learning set described in 4.1. After the termination criteria are satisfied, the individual that achieves the best fitness value on the validation set also defined in 4.1 is returned as the final priority rule.

4.4. Terminal and function sets

The set of primitives used in this paper is given in Table 2 and Table 3. Terminal and function sets are taken from Đumić (2020). Based on tests performed in previous studies (mentioned in section 2.3), terminal values are not normalized, an arithmetic representation is used for individuals, and PSGS is used for the generation scheme.

Table 2: Function set

Function name	**Definition**
+, -, *	addition, subtraction and multiplication
/	protected division: $DIV(a,b) = \begin{cases} 1, & \lvert b\rvert < 0.000000001 \\ \frac{a}{b}, & \text{else.} \end{cases}$
MAX	$MAX(a,b) = \begin{cases} a, & a > b \\ b, & \text{else.} \end{cases}$
MIN	$MIN(a,b) = \begin{cases} a, & a < b \\ b, & \text{else.} \end{cases}$
APS	absolute value
NEG	$NEG(a) = (-1)\cdot a$

With respect to the fitness functions F_1 and F_2 for IPR, three additional terminals are introduced, listed in the Table 4. The introduced terminals contain simple information such as the time of completion of the activity, the waiting time for scheduling, and the distance of the project duration from the lower

Table 3: Terminal set

Terminal	Description
ARU	average resource usage
DPC	number of direct predecessors
GRPW	greatest rank positional weight all
LF	latest activity finish
LS	latest activity start
NSP	number of scheduled predecessors
NUA	number of unprocessed activities
TD	total project duration (horizon)
TNA	total number of activities (not including dummies)

bound of the problem. These specific terminals were chosen because they are related to fitness functions. The time of completion of the activities and the waiting time of the activities for scheduling affect the total duration of the project, which is the only variable in the function F_1, while the distance of the project duration from the lower boundary of the problem is contained in the function F_2.

Table 4: New terminals for IPR

Terminal	Description
ActFin	finish time of activity
ActWait	waiting time for scheduling activity
LBDist	the distance of the project duration from the lower bound

For all three introduced terminals, smaller values lead to a better solution. To avoid stopping the evolution process in the first iteration, their initial value must be set to a number higher than the values that can be reached for each terminal. In this paper, the project horizon is used as the initial value for all three terminals. In the experiments conducted, the project horizon is calculated by summing the duration of all activities in the project, i.e., by scheduling each activity alone without the possibility of another activity being processed at the same time. Since only 3 terminals were introduced, their effects on the outcome are studied separately and in all combinations with each other. The number of combinations to be tested in this case is 7.

4.5. Comparison and statistical test

In this paper, tests for IPR and rollout approach, mentioned in the previous sections, will be performed. The number of runs for both approaches will be 30. The significance of the obtained results will be examined using the Mann - Whitney - Wilcoxon (MWW) test at the 0.01 significance level. Since in this work adjusting the PRs for a static environment is considered in this paper, the results can be compared with metaheuristic methods applicable in a static environment and GA is used as representative. The values of the parameters used by GA are given in Table 5. The parameters were chosen based on experiments, and the maximum execution time of 10 seconds was chosen as the termination criterion. This time is still significantly longer than using priority rules, but on the other hand low enough to be fairly compared with other approaches.

Table 5: Parameters for the GA

Parameter	**Value**
representation	floating point
termination time	10 s
population size	200
mutation probability	0.7

5. Results

5.1. Results for IPR

For each of the 7 combinations determined by the additional terminals used in IPR, 30 rules were evolved. In evolving the PRs for IPR, the same parameter values were used as for GP. The only two differences in evolving PRs using this approach compared to the standard PRs are the use of newly added terminals and the adapted SGS. The schedule generation scheme used in IPR was given by the Algorithm 2. The same scheme is used when evolving PRs and solving new problem instances with evolved PRs. An additional validation set (validation set 2) was used to determine which terminal or combination is best for evolving PRs. The results obtained on validation set 2 for fitness function F_1 are shown in Table 6 and boxplot in the Figure 1, and for fitness function F_2 in Table 7 and boxplot in Figure 2.

Table 6: Results achieved by IPR using fitness function F_1

Combination	Added Terminals	min	med	max
K1	ActFin	2.04658	2.04997	2.05427
K2	ActWait	**2.04333**	2.04977	**2.05289**
K3	LBDist	2.04890	2.05265	2.05610
K4	ActFin, ActWait	2.04632	2.05011	2.05330
K5	LBDist, ActFin	2.04553	2.05031	2.05511
K6	LBDist, ActWait	2.04476	**2.04956**	2.05885
K7	LBDist, ActFin, ActWait	2.04453	2.05031	2.05423

The significance of the results obtained was examined using the Mann-Whitney-Wilcoxon (MWW) test with a significance level of 0.05. The MWW

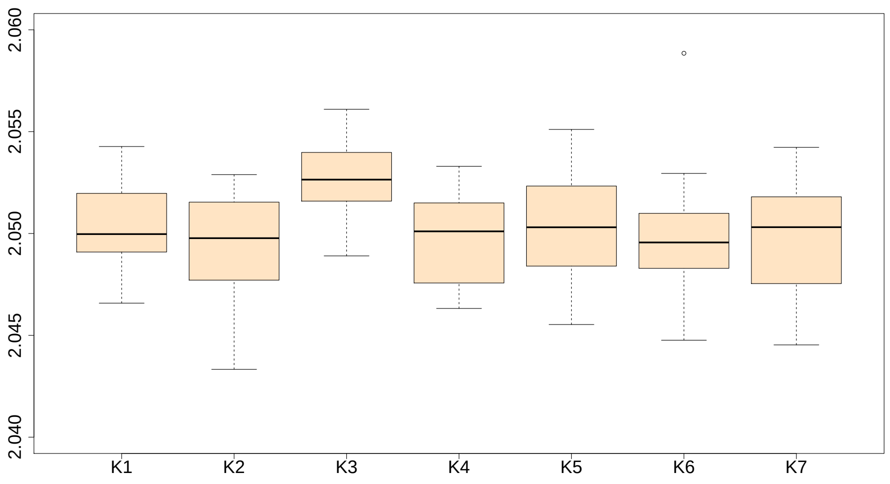

Figure 1: Results achieved by IPR using fitness function F_1

test did not discover a significant difference in the results when using different terminals or a combination of those. To decide which combination to use, the given boxplots were used. The boxplots can be used to determine the combination of terminals for which the results are less scattered.

The lowest minimum and maximum is obtained for the fitness function F_1 when only the terminal ActWait was added. On the other hand, if the terminal LBDist was added together with the terminal ActWait, a slightly smaller median was obtained than when only the terminal ActWait was added. It is interesting to note that in both cases the ActWait terminal was used, which means that the waiting time for the execution of the activity significantly affects the quality of the obtained solution when using the F_1 function. Since the difference in the obtained medians between these two cases is small and that between the minima and maxima is slightly larger, only the terminal ActWait is added to the terminal set when the function F_1 is used for IPR.

Table 7: Results achieved by IPR using fitness function F_2

Combination	Added Terminals	min	med	max
K1	ActFin	23.70156	23.91941	24.21187
K2	ActWait	23.64329	23.92800	**24.18997**
K3	LBDist	23.62028	24.09754	24.44826
K4	ActFin, ActWait	23.60090	23.90161	24.41879
K5	LBDist, ActFin	23.67911	23.98847	24.26429
K6	LBDist, ActWait	**23.51636**	23.89635	24.23332
K7	LBDist, ActFin, ActWait	23.54966	**23.83336**	24.19158

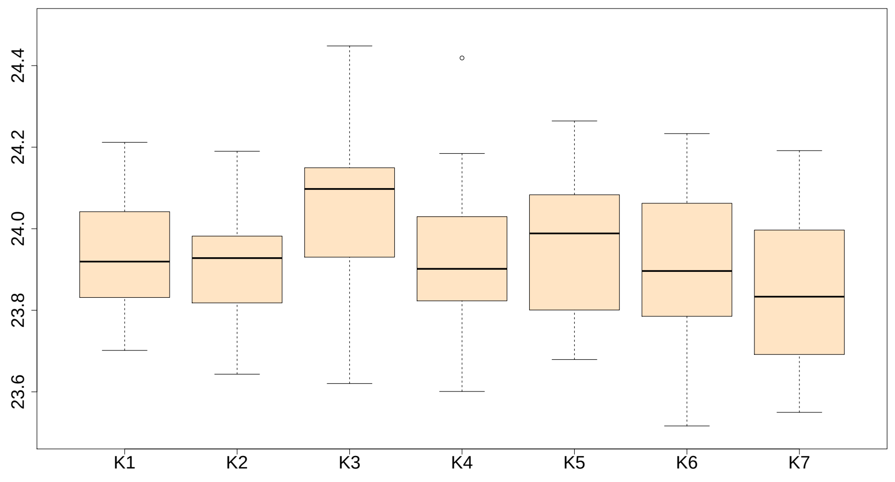

Figure 2: Results achieved by IPR using fitness function F_2

The decision of which terminal or combination to include in the terminal set when using function F_2 was also made based on the given boxplot. For function F_2, three combinations need to be closely observed: introducing only the terminal ActWait, introducing the terminals ActWait and LBDist simultaneously, and introducing all three terminals. By introducing only the terminal ActWait, the lowest maximum will be achieved. If the ActWait and LBDist terminals are introduced simultaneously, the lowest minimum is obtained, and the lowest median is obtained if all three terminals are used. The obtained results are less scattered when only the terminal ActWait is added, but the scatter of results when all three terminals are introduced comes from improved results. If all three terminals are introduced, the maximum will be slightly higher than the lowest achieved. At the same time, the minimum will be significantly lower than that obtained with the other combinations. Accordingly, all three features should be introduced for the function F_2.

Given the parts from which the function F_2 function consists, it is expected that the terminal LBDist should be added to the terminal set. However, it is interesting to note that adding only the terminal LBDist to the terminal set leads to worse results than any other combination of added terminals. This terminal produces better results only when combined with the ActWait terminal. It is interesting that when using the F_2 function, the ActWait terminal significantly impacts the result, as it did with the F_1 function.

5.2. Results for the RollOut Approach

In the rollout approach, if there are already evolved PRs, there is no need to evolved new PRs, which was not the case when IPR was used. This is one of the advantages of this approach. Before using this approach, you must not only choose which PR to use, but also determine the number of activities to consider in the decision process. Several options were explored to determine the optimal number of activities. The results obtained for validation set 2 can be found in Table 8 for function F_1 and in Table 9 for function F_2. The corresponding boxplots can be found in Figure 3 and Figure 4 for functions F_1 and F_2, respectively.

The number of activities considered in the decision process is indicated in the first column of the results shown. The case where all available activities are considered is indicated by ∞ in the tables and Inf in the figures.

Table 8: Results achieved by rollout approach using the fitness function F_1

No. of activities	min	med	max
3	2.02498	2.02766	2.03061
5	2.02203	2.02406	2.02650
7	2.02067	2.02293	2.02550
10	2.01996	2.02217	2.02524
15	2.01914	2.02123	2.02423
20	**2.01895**	2.02098	**2.02396**
30	2.01904	**2.02088**	**2.02396**
50	2.01904	**2.02088**	**2.02396**
∞	2.01904	**2.02088**	**2.02396**

Table 9: Results achieved by rollout approach using the fitness function F_2

No. of activities	min	med	max
3	22.42286	22.59419	22.73496
5	22.18898	22.36952	22.48702
7	22.12095	22.31179	22.4193
10	22.05667	22.24839	22.34631
15	22.01565	22.20643	22.30097
20	22.00902	22.18432	22.30097
30	**22.00222**	**22.18131**	**22.2884**
50	**22.00222**	**22.18131**	**22.2884**
∞	**22.00222**	**22.18131**	**22.2884**

The results obtained with the fitness function F_1 and the fitness function F_2 show similar behavior. The quality of the obtained results depends on the number of observed activities when making decisions. An increase in the number of activities considered at the time of the decision initially leads to significant improvements in the results, while the obtained value stabilizes after a certain number. One of the possible reasons for this behavior is the fact that activities that have a low priority and are not considered in the decision making process

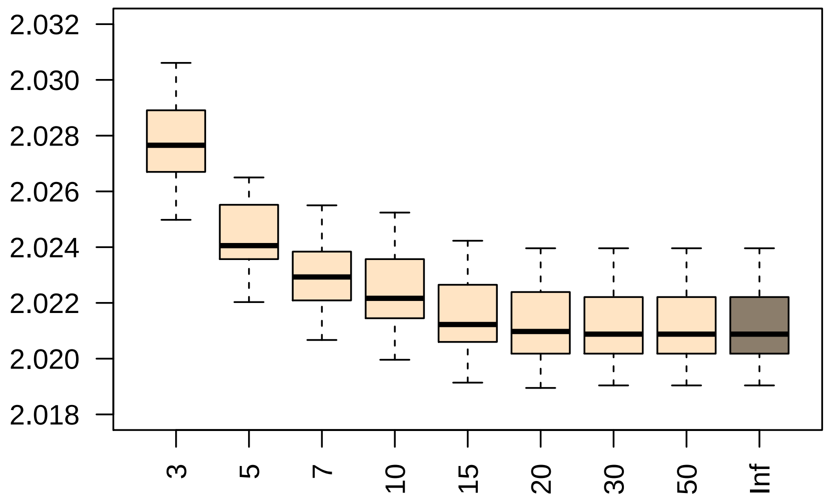

Figure 3: Results achieved by rollout approach using the fitness function F_1

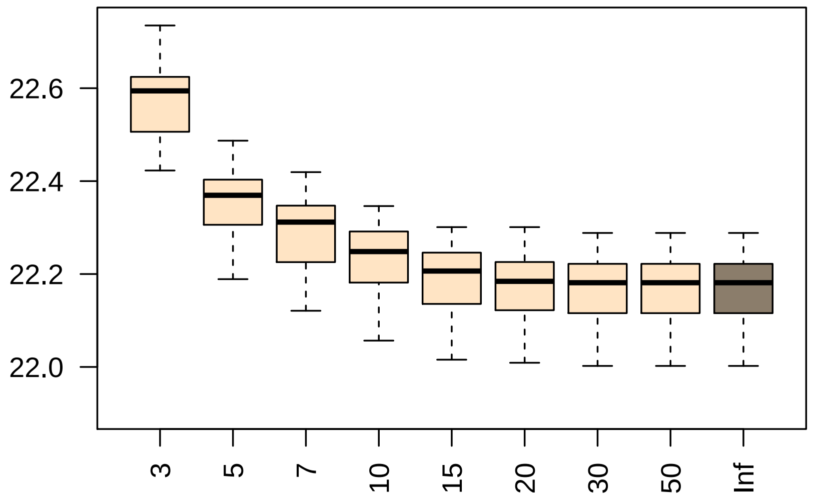

Figure 4: Results achieved by rollout approach using the fitness function F_2

are not those that need to be considered. The second possible reason is that the number of activities considered is greater than the number of activities available, and all available activities have already been considered. Although the best results are obtained when we consider all available activities at the time of decision, increasing the number of activities considered increases the time needed to find a solution.

The time needed for the evaluation depending on the number of activities considered at the time of the decision is given in Table 10 and the graph in Figure 5. The results for time are given in seconds and refer to the time required to evaluate all 204 instances included in the validation set used. As expected, the time required for evaluation increases as the number of activities considered increases. It is important to note that the time required for evaluation stops increasing after a certain number of activities. This cessation of the time increase occurs when the same number of activities is observed at which the performance improvements stop. This phenomenon was investigated in more detail by additional experiments. In the experiments, the number of available activities at each decision point was noted. Since the number of available activities may be different for different numbers of activities considered, the experiments were conducted for 20 and 30 activities considered, which was detected to be crucial in determining the stabilization time point. The results obtained in 30 runs are shown in Table 11 and are identical for functions F_1 and F_2.

Table 10: Time required to evaluate instances of a validation set depending on the number of activities considered, given in seconds

No. of activities	min	med	max
3	23	24.5	26
5	28	35.5	39
7	42	45.5	50
10	43	55	59
15	58	63	71
20	58	65	74
30	60	67	75
50	52	65	80
∞	59	66	79

Table 11: The number of available activities in the rollout approach when 20 and 30 activities were considered

No. of activities	min	med	max
20	1	3	31
30	1	3	30

The results given show that the maximum number of available activities at any decision point is 31 for 20 observed activities and 30 for 30 observed

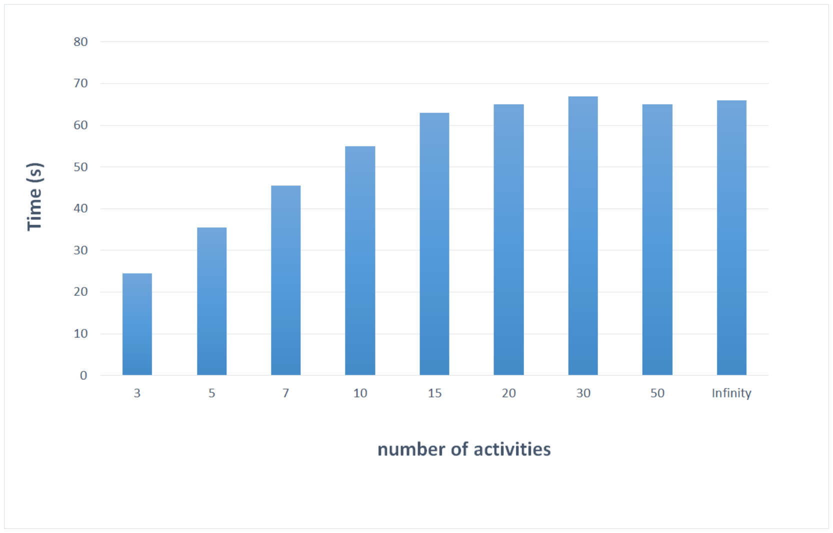

Figure 5: Time required to evaluate instances of a validation set depending on the number of activities considered

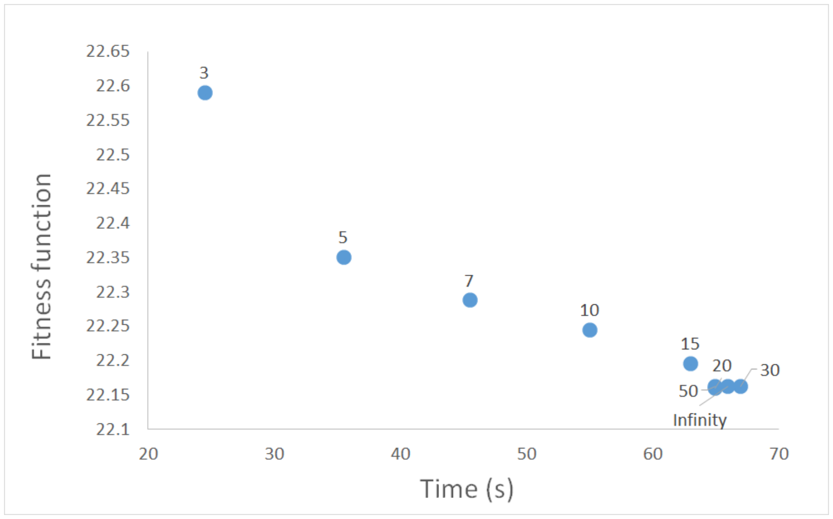

Figure 6: Relation between the value of the F_1 function and the time required to evaluate the instances of the test set depending on the number of activities considered

activities. At the same time, the median is much smaller and is 3 in both cases. It can be concluded that the results stop improving and the time stop increasing because when 30 activities are considered at any decision point, all available activities are considered. For this reason, it does not matter whether we set the number of activities to consider to 30 or more activities or consider all available activities.

Although the time required to evaluate instances from the validation set when all available activities are considered is no more than 1.5 minute, it is significantly longer than the time required to evaluate when a smaller number of activities are considered. For example, consider all available activities at each decision point. The time required for evaluation is on average almost 3 times greater than the time required when only 3 activities are considered.

To further analyze the number of activities available at the time of the decision, Table 12 shows the number of activities available for groups of instances within the validation set when using function F_1. The instances within the validation set are divided into groups of 30, 60, 90, and 120 activities within the project. The results show that the median is lower for instances with 30, 60, and 90 activities than for those with 120 activities. Further analysis of this approach considers two limiting cases, 3 and ∞. However, based on the results in Table 12, it is reasonable to consider the case where 7 activities are considered at each point in the decision for instances with 120 activities. The results for the F_2 function are similar and are not considered in this paper.

Table 12: The number of activities available when problem instances are grouped by the number of activities

No of considered activities	No of activities in project instance	30	60	90	120
20	min	1	1	1	1
	med	2	2	3	7
	max	8	13	21	31
30	min	1	1	1	1
	med	2	2	3	7
	max	8	13	21	30

5.3. Comparison of results

In this paper, two adaptations have been proposed for the development of PRs for a static environment. In a static environment, all the information is available and known, so it is expected that the PRs evolved using these approaches will produce significantly better results in such situations than the PRs that can only be applied in a dynamic environment (dynamic PRs). Since these approaches were evolved for a static environment, it is possible to compare the results with other metaheuristic approaches used to solve RCPSP. In this paper, the Genetic Algorithm (GA) was chosen as a representative of the metaheuristic approaches. When comparing the results obtained by different methods, it is necessary to compare the time required to find a solution. Accordingly, this section compares the results obtained by dynamic PRs, PRs evolved with the proposed adaptations, and GA as representatives of the metaheuristic methods. Given the need to consider the time required to find a solution, two extreme cases are considered for the rollout approach - when three activities are considered (RollOut-3) and when all available activities are considered at each decision point (RollOut-∞). In addition, a comparison is made for the approach where rollout was used in rule development (RollOut PR). Due to the long duration of rule development when using the rollout approach, the priority rules for rollout were evolved with only three activities considered in the decision points. In addition, the parameters from the Chand et al. (2019) were used for the evolution of the PRs, i.e., the population size was set to 200 and a maximum of 20 generations as the stopping criterion.

The results obtained for all these methods on the test set can be found in Table 13 for function F_1 and in Table 14 for function F_2. The corresponding boxplots can be found in the Figure 7 for F_1 and in the Figure 8 for F_2.

From the results presented, it is evident that the methods used behave quite similarly regardless of the fitness function. The results obtained with the IPR and the rollout approach achieve significantly better results than those achieved by dynamic PRs. This behavior is to be expected since the schedule is created multiple times in both adjustments proposed in this paper. With IPR, the initial schedule is improved through iterations. At the same time, in the rollout approach, several possible directions are considered and the best one is selected based on the created schedules.

Interestingly, the results of rollOut-3 and rollOut PR are not statistically different regardless of the fitness function, which means that PRs evolved with standard SGS can be as good as those evolved with the rollout part in SGS. Among the proposed approaches, rollout-∞ achieves the best results, and compared to the other approaches, this difference is statistically significant. Although IPR achieves statistically significantly worse results than the rollout approach, the results obtained with it are statistically significantly better than those obtained with dynamic PRs and GA.

Table 13: Comparison of results obtained by different methods using the F_1 function

Method	min	med	max
IPR	2.03211	2.03576	2.03806
RollOut-3	2.00910	2.01225	2.01477
RollOut-∞	2.00288	2.00522	2.00755
RollOut PR	2.01010	2.01285	2.0167
GP	2.05052	2.05261	2.05599
GA	2.05457	2.07406	2.08144

The time required to evolve a PR using the standard SGS, the SGS adapted for the IPR approach, and the SGS using the rollout approach is given in Table

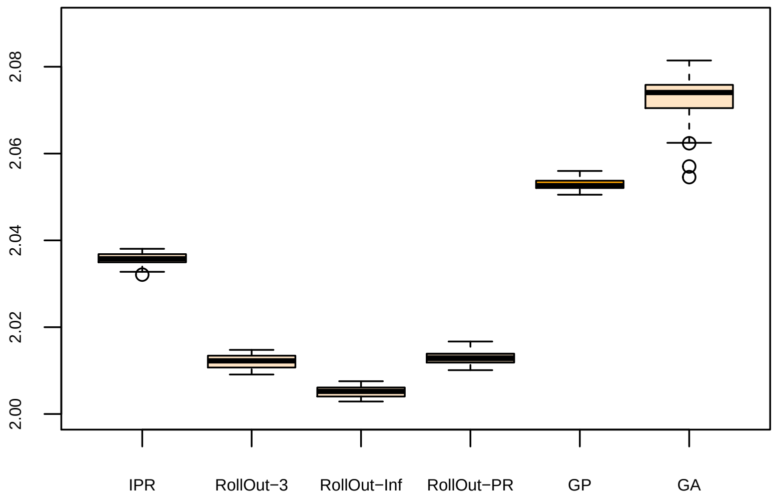

Figure 7: Comparison of results obtained by different methods using the F_1 function

Table 14: Comparison of results obtained by different methods using the F_2 function

Method	min	med	max
IPR	23.43823	23.65492	23.84060
RollOut-3	22.03836	22.24841	22.36618
RollOut-∞	21.62626	21.80197	21.88470
RollOut PR	22.07612	22.24631	22.38901
GP	24.55235	24.7202	24.85042
GA	24.97819	26.20198	26.65415

Table 15: Time required for evolving PRs depending on the method used

Method	duration (s) on average per generation	in total
GP	139.34	3483.43
IPR	458.54	11463.60
RollOut	6773.67	135473.30

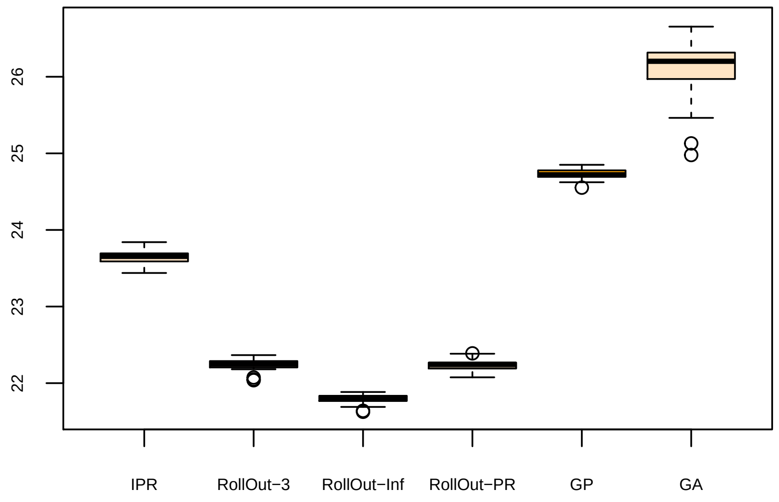

Figure 8: Comparison of results obtained by different methods using the F_2 function

Table 16: Time required for solving problem instances in test set depending on the method used

Method	duration (s) in total	on average per instance
IPR	79.33	0.06
RollOut-3	174.03	0.12
RollOut-∞	431.74	0.30
RollOut PR	321.00	0.22
GP	32.67	0.02
GA	14280	10.00

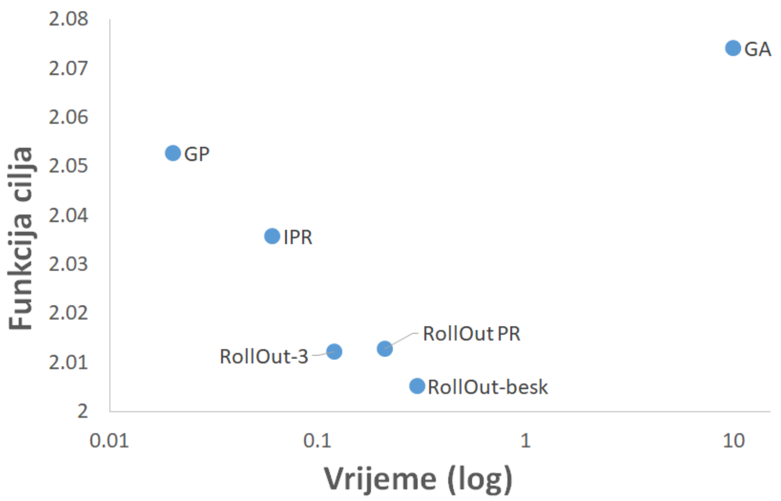

Figure 9: Time required for solving problem instances in test set depending on the method used in log scale

15. The time in the table represents the average value obtained in 30 runs. The average time required for one generation and the total time required to evolve a rule are given in the table. The results presented show that it takes less than an hour to evolve a rule using the standard SGS and slightly more than 3 hours using the IPR approach. The rollout approach used in SGS for evolving PRs takes almost 38 hours on average, which is a large increase over the first two approaches. Also, it is important to note that this approach uses a rollout that considers only 3 activities at each decision point, a smaller population size, and a smaller number of generations. Moreover, any increase in these parameters would lead to a significant increase in the duration of the evolution of the rule, and even this time is problematic to apply. When using PRs, the rules can be evolved in advance regardless of the method used to evolve them, so the time required to evolve the rules does not affect the time required to find a solution to new problem instances.

The time needed to solve new problem instances is more important to observe, and is shown in Table 16 and Figure 9. The table shows the times needed to solve all instances from the test set, which consists of 1428 instances.

When PR is used with the standard SGS, it takes about 33 seconds to solve the entire test set, which averages to 0.02 seconds per instance. When using the IPR approach, this time is about 3 times longer. From this, it can be seen that the ratio of the time required when using the dynamic PR and IPR is practically the same when solving new instances and evolving new PRs. Since the SGS used in IPR differs from the standard SGS only in the number of iterations, it can be concluded that IPR performs on average 3 iterations in each evaluation.

The rollout approach takes significantly more time to find a solution than the previous two approaches. However, the time required to solve an instance is less than half a second on average, regardless of the number of activities considered. The time taken to solve instances within the test set shows that the time taken to solve instances using GA is kept low, but is still longer than the time taken for the approaches proposed in this paper. It is important to note that while the execution time for GA is longer than for the other approaches, this does not lead to better results. It should be noted that parallelization was not used in these comparisons, which is possible in some parts and reduces the time required to find a solution. However, the ratio of time required for the approaches used in this work remains similar to that shown here.

5.4. Further analysis based on the number of activities in instances

The previous subsection presented the results obtained for the entire test set. The test set is defined in section 4.1 and consists of 336 instances with 30, 60 and 90 activities and 420 instances with 120 activities. To get a better insight into the results and quality of the presented approaches, in this section we show the results obtained with subsets of the test set created based on the number of activities. The first subset consists of instances of the test set with 30 activities (j30), the second of those with 60 activities (j60), the third of those with 90 activities (j90), and the fourth of those with 120 activities (j120).

In addition to the approaches compared in the previous subsection, for instances with 120 activities, we present the results obtained with the rollout approach, where 7 activities are considered at each decision point. This approach was added because 7 was identified as the median for this group of instances in Table 12.

The comparison of results explained earlier can be found in Table 17 for function F_1 and in Table 18 for function F_2. The results for the functions F_1 and F_2 show an interesting behavior of the compared approaches. For all approaches, it can be seen that the worst results were obtained for instances with 120 activities, while surprisingly the best results were obtained for instances with 90 activities. Also, the results for instances with 30 activities are worse than those for instances with 60 activities.

The MWW test shows a significant difference in the results between the IPR and the rollout approach with respect to the GP for all groups of instances for F_1 and F_2. In addition, all observed variants of the rollout approach are statistically significantly better than IPR for all groups of instances for both objective functions. For function F_1 and instance group j30, there is no statistically significant difference in the results obtained when 3 or all available activities are observed in the rollout, or when rollout with 3 activities was used in rule development. For groups j60, j90 and j120, rollOut-∞ achieves statistically significantly better results than rollOut-3 and rollOut PR, while rollOut-3 is better than rollOut PR for j60 and j90. For j120, note that rollOut-7 is statistically significantly better than rollOut-3 and statistically significantly worse than rollOut-∞. When we compare GA with rollOut-∞ as the best performing approach, we find that rollOut-∞ performs better than GA for all instance groups except j30. The differences in the results are statistically significant. GA compared to IPR shows statistically significantly worse results for groups j90 and j120.

Similarly, for F_2 IPR and RollOut-∞ perform statistically significantly better than GA only for j120 . From the given results, it is easy to see that GA performs better for sets with instances with fewer activities. Thus, we can conclude that 10 seconds is enough for GA to go in the right direction for instances with less activity, while this time is too short for instances with 120 activity. Thus, we can conclude that the approaches proposed in this paper perform significantly better for larger instances in a shorter time than GA.

Table 17: Results obtained on the instances of the test set grouped by the number of activities using the fitness function F_1

	IPR	RollOut-3	RollOut-7	RollOut-∞	RollOut PR	GP	GA
				j30			
min	2.01200	1.99440		1.99430	1.99660	2.03276	1.95690
med	2.01800	2.00030		1.99940	1.99940	2.03830	1.95898
max	2.02480	2.00490		2.00250	2.00560	2.04306	1.96195
				j60			
min	1.95140	1.93050		1.92590	1.93190	1.96458	1.91703
med	1.95525	1.93320		1.92880	1.93400	1.96734	1.93256
max	1.95820	1.93690		1.93180	1.93810	1.97099	1.94729
				j90			
min	1.89830	1.87950		1.87520	1.88210	1.90817	1.91776
med	1.90050	1.88260		1.87750	1.88360	1.91330	1.93220
max	1.90340	1.88610		1.88080	1.88690	1.91645	1.93912
				j120			
min	2.21480	2.18410	2.1758	2.16960	2.18430	2.24007	2.93773
med	2.22275	2.18920	2.18035	2.17360	2.18900	2.24543	2.99087
max	2.22770	2.19380	2.1848	2.17770	2.19800	2.25295	3.01025

6. Conclusion

Because of their speed, simplicity, and ability to respond to change, PRs are used primarily in a dynamic environment. Their use in a static environment is often absent because the quality of the solutions obtained is usually worse than the quality of the solutions obtained by other metaheuristic approaches. Nevertheless, there are situations where PRs are also used in a static environment. Mostly, these are situations where the speed with which a solution is found is much more important than the quality obtained, or in systems in which, regardless of available information, some changes can occur,e.g., the machine

Table 18: Results obtained on the instances of the test set grouped by the number of activities using the fitness function F_2

	IPR	RollOut-3	RollOut-7	RollOut-∞	RollOut PR	GP	GA
				j30			
min	17.19200	16.13600		16.13000	16.14200	18.34782	11.10473
med	17.50550	16.41950		16.33600	16.39300	18.64064	11.20617
max	18.08800	16.66600		16.52300	16.67700	18.95526	11.34601
				j60			
min	16.12400	14.93500		14.68900	14.95500	16.96477	11.28168
med	16.34100	15.04200		14.77550	15.11600	17.17010	12.05095
max	16.49500	15.15000		14.84300	15.36500	17.40501	12.76632
				j90			
min	14.51900	13.56800		13.22400	13.50100	15.30428	12.66450
med	14.72150	13.68700		13.36900	13.70900	15.50960	13.36472
max	14.88200	13.83100		13.54100	13.81100	15.67000	13.68257
				j120			
min	41.25300	38.97000	38.439	38.07700	39.11400	42.63614	49.75563
med	41.56550	39.56300	38.9525	38.54500	39.51400	43.02796	52.45914
max	42.00100	39.75400	39.174	38.78400	39.64000	43.32689	53.42624

may break down, the worker is on sick leave, and the similar. For this reason, this paper analyzes how PRs can be adjusted to achieve better results with the information available in a static environment.

Two adaptation approaches of the evolving process of PRs in a static environment are analyzed - iterative priority rules (IPR) and rollout approach. In IPR the schedule is improved by iterations using information from the previous iteration, so it is necessary to ensure the transfer of information from one iteration to the next. This transfer is accomplished by introducing new terminals into the terminal set. In this paper, three new terminals are presented and analyzed that contain helpful information from the previous iteration. Also, depending on the objective function used, it was analyzed which terminals should be introduced to achieve the best results. The rollout approach considers multiple activities at each decision point. The decision on which activity to choose is made by creating several schedules and selecting the best among them. The paper analyzes how the number of activities considered affects the quality of the solution and the time needed to find the solution, and whether it is important to use a rollout part in the development of PRs.

Moreover, the results for the proposed approaches are compared with those obtained with dynamic PRs and the GA as a representative of metaheuristic approaches that can be used in a static environment. To allow a fair comparison, GA was limited to 10 seconds for each instance. This time is more than 10 times longer than that required for the approaches presented in this paper. When GA is limited in this way, the approaches proposed in this work achieve significantly better results.

The results obtained with the proposed approaches show that the standard PRs can be improved by using additional information available in the static environment. These results open numerous new topics for further research. It needs to be investigated whether other modification could further improve the results. Next, it needs to be investigated whether the approaches used in this work can be further adapted, e.g., by using a different schedule generation scheme. Regardless of the direction of further research, it is necessary to figure out how to use as much information as possible from a static environment without significantly increasing the time needed to find a solution, which is currently the biggest advantage of PRs.

Acknowledgment

The research in this paper was supported by HRZZ project *Hyperheuristic Design of Dispatching Rules (HyDDRa)* No. **4333**. Any conclusions or recommendations stated herein are those of the authors and do not necessarily reflect the official positions of the organizations involved in this project.

References

Abdolshah, M. (2014). A review of resource-constrained project scheduling problems (rcpsp) approaches and solutions. *International Transaction Journal of Engineering, Management, Applied Sciences and Technologies*, .

Alcaraz, J., Maroto, C., & Ruiz, R. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *The Journal of the Operational Research Society*, *54*, 614–626.

Artigues, C., Demassey, S., & Neron, E. (2008). *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE/Wiley.

Bader-El-Den, M., Poli, R., & Fatima, S. (2009). Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, *1*, 205. URL: https://doi.org/10.1007/s12293-009-0022-y. doi:10.1007/s12293-009-0022-y.

Beham, A., Kofler, M., Wagner, S., & Affenzeller, M. (2009). Agent-based simulation of dispatching rules in dynamic pickup and delivery problems. In *2009 2nd International Symposium on Logistics and Industrial Informatics* (pp. 1–6). doi:10.1109/LINDI.2009.5258763.

Bertsekas, D. P. (2013). Rollout algorithms for discrete optimization: A survey. In P. M. Pardalos, D.-Z. Du, & R. L. Graham (Eds.), *Handbook of Combinatorial Optimization* (pp. 2989–3013). New York, NY: Springer New York.

Bertsekas, D. P., & Castanon, D. A. (1998). Rollout algorithms for stochastic scheduling problems. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)* (pp. 2143–2148). volume 2.

Bertsekas, D. P., & Castanon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, *5*, 89–108.

Blazewicz, J., Lenstra, J. K., & Kan, A. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, *5*, 11–24.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, *64*, 1695–1724. URL: https://doi.org/10.1057/jors.2013.71. doi:10.1057/jors.2013.71.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 449–468). Boston, MA: Springer US. URL: https://doi.org/10.1007/978-1-4419-1665-5_15. doi:10.1007/978-1-4419-1665-5_15.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., & Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In C. L. Mumford, & L. C. Jain (Eds.), *Computational Intelligence: Collaboration, Fusion and Emergence* (pp. 177–201). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: `https://doi.org/10.1007/978-3-642-01799-5_6`. doi:`10.1007/978-3-642-01799-5_6`.

Burke, E. K., Hyde, M. R., Kendall, G., & Woodward, J. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, *20*, 63–89. PMID: 21609273.

Chakrabortty, R. K., Rahman, H. F., & Ryan, M. J. (2020). Efficient priority rules for project scheduling under dynamic environments: A heuristic approach. *Computers and Industrial Engineering*, *140*, 106287. URL: `https://doi.org/10.1016/j.cie.2020.106287`. doi:`10.1016/j.cie.2020.106287`.

Chand, S. (2018). *Automated Design of Heuristics for the Resource Constrained Project Scheduling Problem*. Ph.D. thesis School of Engineering and Information Technology The University of New South Wales Australia.

Chand, S., Huynh, Q., Singh, H., Ray, T., & Wagner, M. (2018). On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems. *Information Sciences*, *432*, 146 – 163.

Chand, S., Singh, H., & Ray, T. (2019). Evolving rollout-justification based heuristics for resource constrained project scheduling problems. *Swarm and Evolutionary Computation*, *50*, 100556.

Chen, H. J., Ding, G., Qin, S., & Zhang, J. (2020). A hyper-heuristic based ensemble genetic programming approach for stochastic resource constrained project scheduling problem. *Expert Systems with Applications*, (p. 114174). URL: `https://doi.org/10.1016/j.eswa.2020.114174`. doi:`10.1016/j.eswa.2020.114174`.

Espejo, P. G., Ventura, S., & Herrera, F. (2010). A survey on the application of genetic programming to classification. *Trans. Sys. Man Cyber Part C*, *40*, 121–144. URL: `http://dx.doi.org/10.1109/TSMCC.2009.2033566`. doi:`10.1109/TSMCC.2009.2033566`.

Frankola, T., Golub, M., & Jaković, D. (2008). Evolutionary algorithms for the resource constrained scheduling problem. In *30th International Conference on Information Technology Interfaces*.

Gargiulo, F., & Quagliarella, D. (2012). Genetic algorithms for the resource constrained project scheduling problem. In *13th IEEE International Symposium on Computational Intelligence and Informatics* (pp. 39–47). Budapest, Hungary.

Guo, W., Vanhoucke, M., Coelho, J., & Luo, J. (2020). Automatic detection of the best performing priority rule for the resource-constrained project scheduling problem. *Expert Systems with Applications*, (p. 114116). URL: https://doi.org/10.1016/j.eswa.2020.114116. doi:10.1016/j.eswa.2020.114116.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *207*, 1–14.

Hildebrandt, T., Heger, J., & Scholz-Reiter, B. (2010). Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* GECCO '10 (p. 257–264). New York, NY, USA: Association for Computing Machinery.

Hindi, K. S., Yang, H., & Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on*, *6*, 512–518.

Hunt, R., Johnston, M., & Zhang, M. (2014). Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 618–625).

Jakobović, D., & et al. (2020). Evolutionary computation framework. URL: http://ecf.zemris.fer.hr/.

Jakobović, D., & Budin, L. (2006). Dynamic scheduling with genetic programming. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, & A. Ekárt (Eds.), *Genetic Programming* (pp. 73–84). Berlin, Heidelberg: Springer Berlin Heidelberg.

Jakobović, D., & Marasović, K. (2012). Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing*, *12*, 2781 – 2789.

Kadam, S., & Kadam, N. (2014). Solving resource-constrained project scheduling problem by genetic algorithms. *Business and Information Management (ICBIM)*, (pp. 159–164).

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, *90*, 320–333.

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz (Ed.), *Project Scheduling: Recent Models, Algorithms and Applications* (pp. 147–178). Boston, MA: Springer US.

Kolisch, R., & Sprecher, A. (1997). Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. *European Journal of Operational Research*, *96*, 205 – 216.

Kolisch, R., Sprecher, A., & Drexel, A. (2001). Characterization and generation of a general class of resource-constrained project scheduling problems, .

Koza, J. R. (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press.

Koza, J. R., III, F. H. B., Andre, D., & Keane, M. A. (1996). Four problems for which a computer program evolved by genetic programming is competitive with human performance. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation,* (pp. 1–10). IEEE Press volume 1.

Kumar, R., Joshi, A. H., Banka, K. K., & Rockett, P. I. (2008). Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* GECCO '08 (pp. 1227–1234). New York, NY, USA: ACM. URL: `http://doi.acm.org/10.1145/1389095.1389335`. doi:`10.1145/1389095.1389335`.

Lin, J., Zhu, L., & Gao, K. (2020). A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, *140*, 112915. URL: `https://doi.org/10.1016/j.eswa.2019.112915`. doi:`10.1016/j.eswa.2019.112915`.

Merkle, D., Middendorf, M., & Schmeck., H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, *6*, 333–346.

Nguyen, S., Zhang, M., Johnston, M., & Tan, K. C. (2013). A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, *17*, 621–639.

Nguyen, S., Zhang, M., Johnston, M., & Tan, K. C. (2013). Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology*, *67*, 85–100.

Oltean, M., & Dumitrescu, D. (2004). Evolving tsp heuristics using multi expression programming. In M. Bubak, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Eds.), *Computational Science - ICCS 2004* (pp. 670–673). Berlin, Heidelberg: Springer Berlin Heidelberg.

Čorić, R., Đumić, M., & Jakobović, D. (2017). Complexity comparison of integer programming and genetic algorithms for resource constrained scheduling problems. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1182–1188).

Özcan, E., & Parkes, A. J. (2011). Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* GECCO '11 (pp. 2011–2018). New York, NY, USA: ACM. URL: `http://doi.acm.org/10.1145/2001576.2001846`. doi:`10.1145/2001576.2001846`.

Pillay, N. (2012). Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society*, *63*, 47–58. URL: `https://doi.org/10.1057/jors.2011.12`. doi:`10.1057/jors.2011.12`.

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

Słowinski, R. (1981). Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. *European Journal of Operational Research*, *7*, 265–273.

Đumić, M. (2020). *Oblikovanje prioritetnih pravila za problem raspoređivanja s ograničenim sredstvima*. Ph.D. thesis Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia.

Đumić, M., Šišejković, D., Čorić, R., & Jakobović, D. (2018). Evolving priority rules for resource constrained project scheduling problem with genetic programming. *Future Generation Computer Systems*, *86*, 211 – 221.

Đumić, M., & Jakobović, D. (2021). Ensembles of priority rules for resource constrained project scheduling problem. *Applied Soft Computing*, *110*, 107606. URL: `https://www.sciencedirect.com/science/article/pii/S1568494621005275`. doi:`https://doi.org/10.1016/j.asoc.2021.107606`.

Đurasević, M., & Jakobović, D. (2020). Automatic design of dispatching rules for static scheduling conditions. *Neural Computing and Applications*, .

Đurasević, M., Jakobović, D., & Knežević, K. (2016). Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, *48*, 419 – 430.

Valls, V., & Ballestín, F. (2004). Population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, *131*, 305–324.

Vonolfen, S., Beham, A., Kommenda, M., & Affenzeller, M. (2013). Structural synthesis of dispatching rules for dynamic dial-a-ride problems. In R. Moreno-Díaz, F. Pichler, & A. Quesada-Arencibia (Eds.), *Computer Aided Systems Theory - EUROCAST 2013* (pp. 276–283). Berlin, Heidelberg: Springer Berlin Heidelberg.

Wittemann, F. (2020). Psplib resource constrained project scheduling problem. URL: `http://www.om-db.wi.tum.de/psplib`.