Information Sciences 432 (2018) 146–163

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems

Shelvin Chand[a], Quang Huynh[a], Hemant Singh[a,*], Tapabrata Ray[a], Markus Wagner[b]

[a] School of Engineering & IT, University of New South Wales, Canberra ACT, Australia
[b] School of Computer Science, University of Adelaide, Adelaide SA, Australia

ARTICLE INFO

Article history:
Received 5 June 2017
Revised 4 December 2017
Accepted 7 December 2017
Available online 8 December 2017

Keywords:
Resource constrained project scheduling
Genetic programming
Heuristic evolution
Evolutionary computation
Generation hyper-heuristics

ABSTRACT

Resource constrained project scheduling is critical in logistic and planning operations across a range of industries. Most businesses rely on *priority rules* to determine the order in which the activities required for the project should be executed. However, the design of such rules is non-trivial. Even with significant knowledge and experience, human experts are understandably limited in terms of the possibilities they can consider. This paper introduces a genetic programming based hyper-heuristic (GPHH) for producing efficient priority rules targeting the resource constrained project scheduling problem (RCPSP). For performance analysis of the proposed approach, a series of experiments are conducted on the standard PSPLib instances with up to 120 activities. The evolved priority rules are then compared against the existing state-of-the-art priority rules to demonstrate the efficacy of our approach. The experimental results indicate that our GPHH is capable of producing reusable priority rules which significantly out-perform the best human designed priority rules.

1. Introduction

Classical project scheduling problem (PSP) [28] usually involves a set of non-preempt-able and precedence related activities that need to be scheduled. The structure of the project can be represented by an activity on node (AoN) graph which is a directed, acyclic and transitively reduced graph in which the nodes represent the activities and the arcs represent the precedence constraints. The precedence constraints impose that certain activities cannot start until all their predecessor activities have been completed. The classical PSP can be solved to optimality (makespan) in polynomial time using the Critical Path Method [18].

The resource constrained project scheduling problem (RCPSP) extends the classical project scheduling by taking into account constraints on the resources required to complete the activities. RCPSP is an important aspect of manufacturing and industrial operations, as it has direct impact on productivity. Therefore, it comes as no surprise that tools to handle this challenging problem are well sought after by practitioners for a wide range of applications, e.g. manufacturing systems [20], construction engineering and software development [5].

* Corresponding author.
E-mail addresses: shelvin.chand@student.adfa.edu.au (S. Chand), quang.huynh@student.adfa.edu.au (Q. Huynh), h.singh@adfa.edu.au (H. Singh), t.ray@adfa.edu.au (T. Ray), markus.wagner@adelaide.edu.au (M. Wagner).

https://doi.org/10.1016/j.ins.2017.12.013
0020-0255/© 2017 Elsevier Inc. All rights reserved.

Several approaches have been proposed for solving RCPSP, ranging from classical priority rules to stochastic search algorithms [20–22]. While optimization algorithms which operate on the solution space directly have the benefit of producing good solutions, they often lack the generalization, flexibility and intuitiveness needed for real-world problems. For this reason a number of businesses prefer to rely on simple priority rules for generating schedules. A priority rule, as the name suggests, assigns priority values to the activities within the project which then determine the order in which these activities will be processed/scheduled. Priority rules provide a scheduling approach which is more understandable and usable for the project participants with different roles, from the project manager to the shop floor operators. Another advantage of using priority rules is the good scalability which is often a weak point of optimization approaches [29]. In fact, a number of optimization methods within literature employ priority rules to generate good starting solutions [13]. Last but not the least, in highly dynamic environments where quick decisions need to be made, priority rules offer a more practical choice over time consuming optimization methods.

However, the design of good priority rules is non-trivial. Evidently, even with significant knowledge and experience, human experts are limited in terms of the possibilities they can consider; an issue that becomes increasingly aggravated as the number of activities grow. Therefore, a key open problem in the field is how to design rules that have good generalization capability when applied to unseen cases. Recently, the use of genetic programming (GP) [25] has gained traction for design of efficient priority and dispatch rules in the domain of job-shop scheduling [3,29]. A GP operates using the principles of recombination, mutation and natural selection to evolve programs. Through such systematic exploration of the large (unbound) search space of possible rules, it is able to produce efficient re-usable priority rules in contrast to the meta-heuristic approaches which only provide one disposable solution. Research on heuristic evolution has also been done in some other areas such as routing [27] and bin packing [6]. However, the potential of GP for evolving rules has been scarcely explored in the field of RCPSP, except for a few foundational studies discussed in the next section. In this study, we aim to address this gap by studying automated evolution of priority rules for RCPSP using a GP approach. In order to demonstrate the benefits of the proposed approach, we also compare the evolved rules against the existing priority rules from the literature. More specifically, the key intended contributions of this paper are:

- Development and study of a GP hyper-heuristic (GPHH) framework to evolve priority rules for the classical RCPSP, using a diverse set of attributes.
- Comparison of two representation schemes (arithmetic, decision-based) for priority rules and two schedule generation schemes (serial, parallel).
- Performance characterization of the evolved rules and benchmarking with the existing rules in the literature on a large set of standard PSPLib instances [23] with up to 120 activities.

The remainder of the paper is organized as follows. Section 2 gives background on the problem and the existing literature. Section 3 gives details on the proposed GP approach for evolving rules. Section 4 provides a discussion on the results obtained while Section 5 summarizes the findings of this paper and highlights some future research directions.

2. Background and related work

2.1. Problem definition

The RCPSP involves a project with M activities which need to be scheduled while considering two types of constraints:

- *Precedence constraints*: These represent the interdependence between the activities. If activity j is a successor of activity i then activity i must be completed before activity j can be started.
- *Resource constraints*: These represent realistic limitations on resources such as manpower, budget, etc. Each project is assigned a set of K renewable resources where each resource k is available in R_k units for the entire duration of the project. Each activity may require one or more of these resources to be completed. While scheduling the activities, the daily resource usage for resource k can not exceed R_k units.

Each activity j takes d_j time units to complete. The overall goal of the problem is to minimize the *makespan*, i.e., the total duration from the start of first to the end of last scheduled activity. The problem is NP-hard [2] and therefore the exact methods [7] are practical only for relatively small instances. For larger instances, heuristic/meta-heuristic methods such as genetic algorithm [37], particle swarm optimization [24], simulated annealing [34] and hybrid search [1] etc. are used to obtain good (but not necessarily optimal) solutions.

2.2. GPHH for evolving priority heuristics

Priority heuristics are often used for solving combinatorial optimization problems as they are simple and computationally efficient. They are used by decision makers to decide on which action to perform next based on a “priority value” among the candidates. Instead of relying on human experts to design new heuristics, researchers are now resorting to genetic programming based hyper-heuristics (GPHH) to automate this process. GPHH’s usually operate on a set of problem attributes and mathematical operators to construct heuristics. This can be both time saving as well as help in discovering rules which may not be easy to construct manually.

Table 1
Some of the prominent existing priority rules [a,b,c,d].

Rule	Abbrv.	Extr.	Description ($j \in \{1 \dots M\}$)
Early Start Time [9]	EST	Min	ES_j
Early Finish Time [9]	EFT	Min	EF_j
Late Start Time[9]	LST	Min	LS_j
Late Finish Time [9]	LFT	Min	LF_j
Most Total Successors [20]	MTS	Max	$\lvert\bar{S_j}\rvert$
First In First Out [9]	FIFO	Min	ID_j
Shortest Processing Time [9]	SPT	Min	d_j
Greatest Rank Position Weight[20]	GRPW	Max	$d_j + \sum_{i \in S_j} d_i$
Greatest Resource Demand [9]	GRD	Max	$d_j * \sum_{k=1}^{K} r_{kj}$
Improved Resource Scheduling Method [20]	IRSM	Min	$max\{0, E_{(j,i)} - LS_j \lvert (i, j) \in AP_n\}$
Worst Case Slack [20]	WCS	Min	$LS_j - max\{E_(i, j) \lvert (i, j) \in AP_n\}$
Average Case Slack [20]	ACS	Min	$LS_j - \frac{1}{\lvert D_n \rvert - 1} \sum_{(i,j) \in AP_n} E_{(i,j)}$

[a] $E(i, j)$ is the earliest time to schedule activity j if activity i is started at time t_n.
[b] D_n is the set of activities eligible for scheduling at stage n.
[c] AP_n is the set of all activity pairs in the decision set at stage n, i.e. $AP_n = \{(i, j) \lvert i, j \in D_n, i \neq j\}$
[d] $\bar{S_j}$ is the complete set of successor activities for activity j.

Recently, GPHH has been studied for production scheduling [4], routing [27] and project scheduling [33]. Nguyen et al. [29] studied the representation of rules within GP in evolving priority heuristics for the static job shop scheduling problem. Their study compared pure arithmetic representation against a mixed arithmetic-decision tree representation. The results indicated that GP operating on mixed representation performed better as it was able to utilize information on the machine and system status through the decision tree based rules. Interestingly, the rules evolved through GP with mixed representation operated on a hybrid scheduling procedure based on a non-delay factor as opposed to the rules evolved through GP with pure arithmetic structure which operated on pure non-delay scheduling. So it is not evident whether the improvement was due to the mixed representation or because of the hybrid scheduling.

Durasević et al. [11] compared a number of evolutionary approaches for evolving priority heuristics for scheduling on unrelated machines in a dynamic environment. Among these was the dimensionally aware genetic programming (DAGP) which restricted the search process to only semantically correct solutions. The results indicated that DAGP performed at par with standard GP.

Ensemble based GPHH approaches have also shown to be effective in evolving high performing heuristics. Park et al. [31] proposed a GPHH based on cooperative coevolution for evolving ensemble of rules for the static job shop scheduling problem. Each rule in the ensemble would vote on which job to schedule next and the one with the highest number of votes would be scheduled next. Djurasević et al. [11] considered more complex approaches for ensemble learning such as Bag GP, Boost GP and cooperative coevolution as well as a simple ensemble combination approach which tries to combine existing dispatching rules evolved beforehand. As expected ensemble approaches in general perform better in comparison to standard GPHH's evolving a single rule.

One of the disadvantages of GPHH is that it is often computationally expensive for evolving heuristics. This is especially true if the size of the training set is large. There are two ways to handle this. First is to find a way to reduce the size of the training set while still maintaining a reasonable level of diversity in the data. Some studies [17,30] have explored an alternative in which surrogate modeling is combined with the GPHH to efficiently utilize the computational budget. This has shown to improve results compared to standard GPHH when operating with limited number of function evaluations.

Apart from above advancements, there have also been problem specific developments such as introduction of new attributes, schedule generation schemes, etc. A review of such developments in the area of production scheduling could be found in [4]. Notably, however, the use of GPHH in RCPSP has been scarcely explored, and the relevant works are discussed in the next subsection.

2.3. Priority rules for RCPSP

A number of priority rules have been proposed in the literature for RCPSP based on different rationales. In-fact the authors of [19] indicated that they performed an initial evaluation of as many as 73 different priority rules for their study which focused on benchmarking priority rules. We list some of the most widely used and high performing rules in Table 1, which we also use later for comparisons in the numerical experiments.

While the number of rules available for RCPSP is by no means scarce, the automated evolution of such rules has received less attention. To the authors' knowledge, the first study towards this direction is by Frankola et al. [14] in which they used a GPHH to find an appropriate priority rule that can be used to calculate priorities for all activities in any project. Their GPHH used binary operators together with activity attributes to form arithmetic priority rules. They used duration, number of resources required, total successor count, total predecessor count, requested resource total and average resource usage as the attributes. While the overall idea was promising, the authors only considered a limited attribute set and did not really mention how competitive their evolved rules were in comparison to existing priority rules. Despite this, the work in [14] provides a good motivation for this research.

In a more recent work, Šišejković [33] explored a larger attribute set and used a greedy feature selection approach to identify the best performing attributes. It is interesting to note that attributes such as Latest Start Time (LST) and Latest Finish Time (LFT) were eliminated using this feature selection approach even though these attributes are part of some of the best performing priority rules designed by human experts. The selected attributes were later combined to form arithmetic priority rules. The resulting rules, while being competitive, were not able to outperform the existing LST and LFT rules. The author also conducted an extensive parameter tuning exercise in which the data from the test set was used. This is in contradiction to the standard paradigm where the test set is kept completely separate to gauge the generalization capability of the trained model.

The above discussions form the motivation for this study on evolving priority rules using GPHH, which introduces a number of methodological improvements supplemented with comprehensive numerical experiments. Some of the differences in this study compared to the existing literature are the following:

- We have considered two different forms of representation whereas in all of the previous studies for RCPSP the authors have only considered plain arithmetic representation. The decision tree based representation aims at improving not only the performance but also the understandability of the evolved rules. Further, we also present experiments with both serial and parallel schedule generation schemes.
- We propose a normalization procedure so that the evolved heuristics are less prone to get biased by the order of the values of the attributes, and improve the generalization capability.
- We have carefully constructed a diverse set of features to adequately capture different aspects of the problem, in contrast to previous studies where the choice of attributes was relatively ad-hoc.
- We use lower bounds as targets while training, so that no optimization runs are required for generating the training data (unlike previous approaches). Additionally, we use only smaller instances (J30, J60) for training and all instances (including J90, J120, RG300) for testing to more appropriately reflect the scalability of the proposed approach. Lastly, we treat the “test data” as completely unseen during the evolution, unlike previous work [33] where test set was used for tuning certain parameters.
- A detailed comparison and analysis of the results obtained using the proposed approach is performed with respect to the existing techniques. The proposed approach was also, for the first time reported, able to achieve better performance compared to the existing rules such as LST and LFT.

We further detail these aspects in the following sections.

3. Proposed GP framework for evolving priority rules

In order to build a framework for evolving priority rules, a few basic components are essential. First is the representation of the priority rule/function itself, which determines the forms in which one can construct it. We study two types of priority rules in this research – one being purely arithmetic and the other involving a combination of arithmetic and decision-based operators. Second is a set of features (problem attributes) that the priority rules will operate on. We consider a diverse range of attributes to capture the problem characteristics and help construct good priority rules. Furthermore, in order to evaluate any given priority rule to judge its fitness, a schedule needs to be generated from it. In this regard, we investigate two popular schedule generation schemes (SGS), the serial and the parallel SGS. Lastly, a mechanism is needed to manipulate, recombine and select good priority rules, for which we use a genetic programming approach. We discuss each of these aspects in more detail next.

3.1. Genetic programming framework

The GP framework used in this paper is the basic GP by Koza [25]. It starts with a random population of rules which are evaluated on a set of training instances. Each rule is evaluated on a given instance using either the parallel or the serial schedule generation scheme. Ties in priority values are broken using activity ids.

The overall fitness of an individual is determined by calculating the deviation of the achieved makespan(ms_i) against the instance lower bound (lb_i); That is to say, the lower bounds are used as the targets while evolving the rules using the GP. This is then summed across all instances (Q) and averaged, as shown in Eq. (1). A lower average percentage deviation is preferred. This process is different from [14] in which the authors use optimal or best known solutions for calculating deviation and fitness. The approach may not be always practical for two reasons. Firstly, in a real world scenario the optimal makespan will not always be available. Secondly, the best known solutions change over time making it difficult to make comparisons across different algorithms. For these reasons, as a more practical alternative, we have adopted lower bounds to calculate fitness in this study. The lower bounds used in this study are based on critical path, which is calculated by relaxing the resource constraints and generating a precedence feasible schedule. These lower-bounds are commonly used in

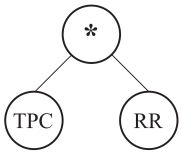

Fig. 1. An example of arithmetic priority rule.

Table 2
Operator set.

Add(a,b)	Mul(a,b)	Sub(a,b)	Div(a,b)	Max(a,b)	Min(a,b)	Neg1(a)
$a+b$	a^*b	$a-b$	$\begin{cases} a/b & if\ b>0 \\ 0 & otherwise \end{cases}$	$\begin{cases} a & if\ a>b \\ b & otherwise \end{cases}$	$\begin{cases} a & if\ a<b \\ b & otherwise \end{cases}$	$-1*a$

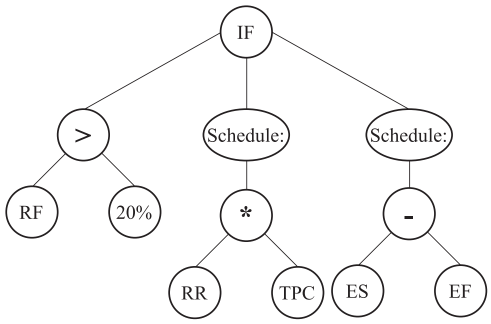

Fig. 2. An example of decision priority rule.

RCPSP studies [8,13,22] to measure the performance of algorithms.

$$\frac{1}{Q}\left(\sum_{i=1}^{Q}\left(\frac{ms_i - lb_i}{lb_i} * 100\right)\right) \tag{1}$$

Genetic operators of crossover and mutation are applied to evolve the population. Once the stopping criteria is reached, all the solutions within the final population are evaluated on a validation set. The best performing rule on the validation set is selected as the representative for that run. The validation set prevents the GP from selecting a solution which is over-fitted to the training set. The operators and settings used in this study for GP are discussed in the next section and are fairly standard. For the sake of brevity, we omit the detailed discussion of the GP components and instead refer the interested readers to [25].

3.2. Representation and GP approaches

In this study we consider three different GP approaches (namely GP-AR, GP-MRG, GP-MRL) using two types of representations, which are discussed next.

3.2.1. Arithmetic representation (GP-AR)

In GP-AR, the priority rules are represented as arithmetic expressions in which activity attributes are combined using mathematical operators. An example is shown in Fig. 1, which shows a simple multiplication of two attributes TPC and RR (these are discussed later). Six binary operators and one unary operators are considered, listed in Table 2. The benefit of an arithmetic representation is that it is quiet simple to understand and manipulate with the GP framework. The latter point is very important since even random recombinations will result in feasible trees.

3.2.2. Mixed representation (GP-MR)

The mixed representation consists of both pure arithmetic rules as well as rules based on a decision tree structure. The decision tree representation uses IF-Then-Else logic to construct rules. An example tree is given in Fig. 2 in which the first branch from the 'IF' represents the condition whereas the other two represent the 'Then' and 'Else' case, respectively. The decision rules utilize arithmetic as well as logical operators given in Tables 2 and 3.

Further, two types of decision rules, global and local are investigated using the mixed representations (corresponding approaches are termed GP-MRG and GP-MRL) respectively.

Table 3
Decision rule operators and terminals.

Type	Description
Comparison operators	$<$, $\leq$, $>$, $\geq$
Thresholds	10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%

Table 4
Attribute values and resulting priority ($d_j + r_j$).

	Scenario 1			Scenario 2		
	A_1	A_2	A_3	A_1	A_2	A_3
d_j	1	2	3	10	20	30
r_j	5	6	4	5	6	4
$Priority(Order)$	6(1)	8(3)	7(2)	15(1)	26(2)	34(3)

The **Global Decision Rules** operate on instance attributes to make decisions on which arithmetic sub-rule to use in determining activity priority. This allows us to generate rules specifically tailored for different instance difficulties and complexities. In this representation, the instance attributes are used for constructing the decision conditions whereas the activity attributes are used for constructing sub-rules to use in different conditions.

The **Local Decision Rules** operate on system and the *eligible set* attributes. At each time instant the state of the resources and the eligible set (activities eligible for scheduling) are considered, based on which a decision is made on the arithmetic sub-rule to be used. The system and eligible set attributes are used in formulating the decision conditions whereas the activity attributes are used for constructing sub-rules to use in different conditions. The eligible set attributes allow us to focus on the collective characteristics of the activities under consideration and device sub-rules to better handle different sets of activities. Similarly the system attributes allow us to focus on bottleneck cases.

The structure of the decision tree is inspired by the representation proposed by Nguyen et al. [29] for Job Shop Scheduling. The advantage of this representation is that the decision rules are more intuitive to understand in comparison to the plain arithmetic representation. Unlike the plain arithmetic rules, the size factor is not a big issue with the decision rules as the trees do not necessarily have to grow to improve. Condition and logic manipulation can also result in improvement. Another advantage of this representation is that the tree can easily transition into a pure arithmetic rule should the decision based elements prove to be not as useful. The disadvantage is that from a GP perspective this is a more complex form of representation to manipulate since the structural feasibility (with regards to the decision rules) can be easily violated through random crossover/mutation.

3.3. Attributes/terminal set

In this subsection we discuss all the attributes that we consider to be included in the priority rules. These are segregated into three types: those relating to the activities, instance and system/eligible set.

Before detailing the attributes, we wish to highlight that unlike previous works [14,33] we scale the range of the attributes between 0 and 1. This allows for better generalization and scalability with regards to unseen instances. Such normalization of the data is important since the attributes operate on different units, potentially with values in different orders of magnitude. Consider the simple rule $d_j + r_j$ (lower the better) which can be a candidate rule produced by GP during evolution. Here, d_j represents the duration of activity j while r_j represents the resource requirement of activity j considering there is only one resource. Table 4 shows two scenarios with three activities for which the priorities are computed using the aforementioned rule. For scenario 1, the resulting order in which tasks will be processed is $[A_1, A_3, A_2]$ since a lower priority value is preferred. Scenario 2 is constructed by simply multiplying the durations by a factor of 10. The problem is still essentially the same but the priorities are affected; the resulting order now being $[A_1, A_2, A_3]$. If however, all d_j and r_j values are normalized by dividing by $max(d_j)$ and $max(r_j)$ respectively, then the bias due to the magnitude of the values could be avoided and the relative priorities will be preserved.

3.3.1. Activity attributes

The activity attributes that we consider are listed below, and their calculations are detailed in Table 5.

- *TSC/TPC*: The total count of the (immediate & non-immediate) successors/predecessors of an activity.
- *ES/EF*: The earliest start/finish time for an activity in the precedence feasible schedule calculated by relaxing the resource constraints where each activity is scheduled as early as possible.
- *LS/LF*: The latest start/finish time for an activity in the precedence feasible schedule calculated by relaxing the resource constraints where each activity is scheduled as late as possible using backward recursion [12].
- *RR*: The total number of resources required by an activity.
- *AvgRReq*: The average resource requirement of an activity.

Table 5
Activity attribute set.

Attribute	Details
Early Start (ES_j)	$\frac{1}{max(EF)} ES_j$
Early Finish (EF_j)	$\frac{1}{max(EF)} EF_j$
Late Start (LS_j)	$\frac{1}{max(LF)} LS_j$
Late Finish (LF_j)	$\frac{1}{max(LF)} LF_j$
Total Predecessor Count (TPC_j)	$\frac{1}{M-1} \lvert\bar{P_j}\rvert$
Total Successor Count (TSC_j)	$\frac{1}{M-1} \lvert\bar{S_j}\rvert$
Resources Required (RR_j)	$\frac{1}{K} \sum_{k=1}^{K} \begin{cases} 1 & if\ r_{jk} > 0 \\ 0 & otherwise \end{cases}$
Average Resource Requirement ($AvgRReq_j$)	$\frac{1}{K} \sum_{k=1}^{K} \{ \frac{1}{R_k} r_{jk} \}$
Maximum Resource Requirement ($MaxRReq_j$)	$Min\left(\frac{1}{R_k} r_{jk},\ k \in 1 \ldots K \right)$
Minimum Resource Requirement ($MinRReq_j$)	$Max\left(\frac{1}{R_k} r_{jk},\ k \in 1 \ldots K \right)$

$\bar{P_j}$ is the complete set of predecessor activities for activity j.

- *MinRReq*: The minimum resource requirement of an activity.
- *MinRReq*: The maximum resource requirement of an activity.

The above attribute set has been chosen in order to be as diverse as possible in capturing different aspects of the problem. Attributes TPC and TSC capture the state of the precedence network. The early start/finish and late start/finish attributes capture the structure of the critical path. Finally the resource based attributes such as RR and AvgRReq capture the state of the activity resource requirements. For the resource based attributes, in our preliminary experiments, we had considered an alternative in which the requirement of each resource by an activity was considered as a separate attribute as opposed to the aggregated approach that is currently being used. This resulted in rules having poor performance and scalability since considering each one as a separate attribute makes it difficult to quantify the collective effect on the activity. Our attribute set is different from [33] in which the author focuses more on attributes representing the precedence network and resource constraints and less on the critical path.

Since the critical path attributes are also stand-alone rules we would like to be able to differentiate between situations in which we refer to the rule as opposed to the attribute within our evolved rules. For this reason we use two different notations (abbreviations). For example, if we refer to the Latest Finish Time as a rule, we use abbreviation LFT. If we refer to Latest Finish Time as an attribute we use simply LF. This approach is followed for all the remaining critical path attributes (LST, EST, EFT).

3.3.2. Instance attributes

For this study we consider three key instance attributes:

- Resource factor (*RF*): Average % of resources required per activity [16].
- Resource Strength (*RS*): The scarceness of the resource capacities [35].
- Resource Constraindness (*RC*): The demand of resources relative to overall availability [32].

Attributes such as network complexity and instance size are not included as prior studies [8] have shown that their impact on problem difficulty is quite limited. Despite not including size explicitly as an attribute our evolved rules scale very well with the instance size.

3.3.3. System and eligible set attributes

The system attributes we consider are the mean (*AvgRA*), minimum and maximum resource availability (*MinRA*, *MaxRA*) and system progress (*SP*).

- *AvgRA*: Averages the resource availability across all resources from the current time instant t_n to the planning horizon ($\mathscr{Y}$). Resource availability for resource k at time t_n is calculated as the difference between R_k and total resource utilization ($RU_{t_n k}$) by all the ongoing activities at time t_n. $\mathscr{Y}$ is the finish time of the longest activity within the eligible set if started now.
- *MinRA*: Averages the resource availability of the resource available in the least quantity from t_n to the planning horizon ($\mathscr{Y}$).
- *MaxRA*: Averages the resource availability of the resource available in the highest quantity from t_n to the planning horizon ($\mathscr{Y}$).
- *SP*: Measures the number of activities already scheduled, i.e. those assigned a start time, as a percentage of the total number of activities to be scheduled. Calculated as the percentage difference between M and $\mathscr{X}_{t_n}$ where $\mathscr{X}_{t_n}$ is the set of unscheduled activities at time t_n.

Table 6
System/eligible set attributes.

Attribute	Details
AvgRA	$\frac{1}{\mathscr{Y}-tn+1}\frac{1}{K}\sum_{p=t_n}^{\mathscr{Y}}\sum_{k=1}^{K}\frac{1}{R_k}(R_k - RU_{kp})$
MinRA	$Min(\frac{1}{\mathscr{Y}-tn+1}\sum_{p=t_n}^{\mathscr{Y}}\frac{1}{R_k}(R_k - RU_{kp}), k \in \mathscr{V})$
MaxRA	$Max(\frac{1}{\mathscr{Y}-tn+1}\sum_{p=t_n}^{\mathscr{Y}}\frac{1}{R_k}(R_k - RU_{kp}), k \in \mathscr{V})$
SP	$\frac{1}{M}(M - \mathscr{Z}_{t_n})$
AvgRF	$\frac{1}{\lvert Dn\rvert}\sum_{j \in D_n} RR_j$
AvgRU	$\frac{1}{\lvert Dn\rvert}\sum_{j \in D_n} AvgRReq_j$

These allow us to identify the state of the system and specifically focus on bottleneck situations. *MinRA* and *MaxRA* only operate on the resource set $\mathscr{V}$ which contains those resources that have a non-zero requirement from the activities in the eligible set. *AvgRA* is calculated over the complete resource set to give a more global picture.

The eligible set attributes we consider are the average resource factor (*avgRF*) and average resource utilization (*AvgRU*).

- *AvgRF*: Averages the number of resources required by the activities within the eligible set.
- *AvgRU*: Averages the resource utilization by the activities within the eligible set.

The calculation of the above system and eligible set attributes are outlined in Table 6.

3.4. Schedule generation scheme

The two commonly used schedule generation schemes (SGS) within literature are the serial and parallel SGS (detailed in [22]). Serial SGS generates a schedule in n stages where n is the number of activities. At any stage the set of eligible activities consists of unscheduled activities with all predecessor activities scheduled. In each of the n stages, one activity is selected from the eligible set and scheduled at the earliest precedence and resource feasible completion time [22]. The parallel SGS works based on time increment. At each time instant it identifies a set of eligible activities (D_n) from which one is selected to be scheduled. The eligible set is based on activities which can be precedence and resource feasibly started at the current time. A time increment is done once the eligible set at the current time is completely empty.

It was proven in [21] that the serial SGS generates *active* schedules. An active schedule is one in which none of the activities can be started earlier without delaying some other activity [22]. For makespan minimization, it has been proven that the optimal solution will always lie in the set of active schedules. Furthermore, it was also proven in [21] that the parallel SGS generates non-delay schedules. A non-delay schedule is one in which none of the activities can be started earlier without delaying some other activity even if activity preemption is allowed [22]. Since the set of non-delay schedules is only a subset of the set of active schedules it might not contain optimal schedules for a regular performance measure. However, despite generating non-delay schedules, parallel SGS has been shown to outperform serial SGS when operating on priority rules [21]. Nevertheless, the use of priority rules with serial SGS may still be of importance in cases where for example the priority rule is being used with an optimization algorithm which in most cases operate on serial SGS. For the sake of completeness, we present the results obtained using for both types of SGSs in Section 4.

We use the original Parallel and Serial SGS definitions presented in [22] which have been used in previous studies on priority rules [20] and are also commonly used in heuristic and meta-heuristic algorithms for RCPSP [8]. GP-AR and GP-MRG can operate on both the serial and parallel SGS but GP-MRL can only operate on the parallel SGS. This is because it tries to dynamically consider the system/eligible-set attributes at each time instant and the parallel SGS is the only one which can adequately support this behavior.

Unless and until stated otherwise, at each scheduling step, from the eligible set, we always pick the activity with minimum value of the priority function to be scheduled next. Ties are broken using the activity ID, with lower ID given higher preference. The calculated priority values are also rounded to 10 decimal places before the decision on which activity to be scheduled next is made, in order to avoid undesirable effects from floating point error.

4. Numerical experiments

4.1. Experimental setup

To evaluate the effectiveness of our GP system and the evolved rules, we used the problem instances from the project scheduling library (PSPLib) [23]. The PSPLib contains instances of different sizes and complexities. For this study we use instances with 30, 60, 90 and 120 activities, which are correspondingly referred to as J30, J60, J90 and J120 instances. The instances were originally constructed [23] using three complexity parameters, namely network complexity (NC), resource factor (RF) and resources strength (RS). Instances are constructed using 48 different combinations of the problem parameters for J30, J60 and J90, and 60 different combinations for J120. Each combination results in 10 instances. We have used combination of J30 and J60 instances for training and validation and combination of J30, J60, J90 and J120 for testing. We

Table 7
The parameters use for the numerical experiments.

Parameter	Value
Population size	1024
Stopping condition	25 Generations
Fitness function	Average Percentage Deviation from CPM Lower-Bound
Duplication retries	100
Initialization method	Ramped Half-and-Half algorithm, height range 3–5
Selection method	Tournament Selection
Selection pool size	7
Max program depth	6
Crossover and mutation probabilities	0.9 and 0.1 respectively
Elitism-fraction	0.1
Number of runs	31

have used 20% of the J30 and J60 instances for training and 10% of the same set for validation. These instances are carefully sampled to ensure that the training set contains diverse range of instances. As previously mentioned, each parameter combination results in 10 instances. We pick the first two instances from each combination to construct our training set. The third instance from each combination is picked to construct the validation set. For the test set we use the remaining 70% of the J30 and J60 instances together with the entire J90 and J120 set. This means we use around 15% (288) of the overall data for training and validation, and the remaining 85% (1752) for testing. Note that for partitioning the instance set we relied on the parameter data and combinations used in [23]. While NC and RF are directly replicable to the data given in [23], reverse engineering for RS does not always lead to the same exact value. For this reason, in GP-MRG we relied on explicitly computing RS using the formula given in [35]. For partitioning the instance set we still relied on the parameter data by Kolisch since it was already organized into discrete combination sets.

The GP has been coded using the ECJ Framework [26]. All codes and data used for this research are available on request from the first author. To compare with the existing rules on the test set we choose two different rules from our 31 runs. As previously mentioned, the solutions from the final generation are all evaluated on the validation set and the best performing rule is selected as the representative for that run. Across all runs we pick the two best rules based on validation performance. These two rules are then compared against the existing human-designed priority rules on a completely unseen test set (i.e. the 85% instances mentioned earlier). Since the performance on the test set is not known in advance it prevents us from biasing the selection of the rule or in any way affecting the training process. For ease of reference, the evolved rules are denoted as $< GPApproach >_{<schedule\ generation\ scheme><validation-rank>}$. For example, a rule denoted as AR_{psgs2} indicates that the rule was evolved using GP-AR, operates on the parallel schedule generation scheme and is ranked second overall on validation performance. Also to be noted that our training and validation set does not contain any instances from the J90 and J120 set. This allows us to evaluate how well our evolved rules scale with an increase in problem size.

In the results (tables) presented in the next sub-sections, the best performance values are highlighted in bold. Furthermore, we use Wilcoxon Signed Rank Test to establish the statistical significance of the performance improvement offered by our evolved rules. We have utilized the standard two-tailed Matlab implementation of the test which operates at the 5% significance level. The observations for each pair of rules being compared include the percentage deviation on each corresponding test instance.

4.2. Priority rules used for benchmarking

To demonstrate the effectiveness of rules evolved by GP, we compare the performance with notable existing priority rules mentioned earlier in Table 1. The CPM (Critical Path Method) based rules (EST, EFT, LST, LFT) are calculated based on the precedence network by relaxing the resource constraints. These rules are commonly used in the traditional unconstrained project management domain and generalize very well to the constrained domain. In fact, as one can see, the critical path attributes are also used as building blocks for many other rules (e.g. WCS and ACS). The more sophisticated rules such as WCS and ACS operate on the principle of slack imposed if an activity is not scheduled at the current time. WCS has been ranked as the best rule for RCPSP in different independent studies [19,20]. The comparison set also includes other rules which focus on simple activity attributes such as the number of successors (MTS) and the duration of an activity (SPT). For a more detailed description of these rules we refer the interested reader to [9,20]. Finally, the comparisons with two other GP based strategies [14,33] are also presented.

4.3. GP settings

For this study we have used the standard Koza GP [25], although extension to some of the more advanced GPs could be an interesting subject for future research. The parameters for the study are detailed in Table 7. Nearly all of these setting and parameter values are standard in the GPHH literature [29].

Table 8
Comparison of test performance for the different GP approaches: Serial SGS.

	GP-AR	GP-MRG
Mean	**27.44**	27.70
Median	**27.40**	27.63
Best	**27.11**	27.37
Worst	28.32	**28.20**
Standard Dev.	0.24	**0.20**

Table 9
Comparison with existing priority rules: Serial SGS.

Rule	Test Perf.	Rule	Test Perf.	Rule	Test Perf.
Random	45.63	*EST*	36.40	AR_{SSGS_1}	27.37
EFT	39.45	*LST*	27.63	AR_{SSGS_2}	**27.20**
LFT	28.59	*FIFO*	35.21	MRG_{SSGS_1}	27.37
SPT	48.76	*MTS*	29.95	MRG_{SSGS_2}	27.39
GRD	41.56	*GRPW*	39.24		

4.4. Results using serial SGS

4.4.1. Performance

Comparison on the test performance for the two different GP approaches (AR and MRG) is given in Table 8. GP-AR outperformed GP-MRG on both mean and median performance. The poor performance of GP-MRG can be attributed to the complicated structure of the decision rules constructed using GP-MRG. Random crossover and mutation might be insufficient to produce high quality offspring since the structure is strongly typed. When we consider the structure of the decision rules, essentially we are expecting the GP to produce several different sub-rules to be used in different cases. This may be a difficult task to achieve and in essence it may be easier to find a single arithmetic rule which gives decent performance rather than to find a set of rules which perform well for specific scenarios. Nevertheless, GP-MRG has the ability to transition into pure arithmetic rules if need be, so in theory it should have the benefit of both the normal arithmetic as well as the decision rules. However, it seems the basic GP scheme used here was not able to exploit this benefit.

4.4.2. Comparison with existing priority rules

Next we compare the performance of the two best rules on validation set[1] obtained using each of the GP approaches with the existing priority rules, shown in Table 9. The performance of these rules are compared on the test set. WCS, ACS and IRSM are not included in this comparison since they were designed specifically for parallel SGS. It can be seen that all of the four evolved rules outperform the existing priority rules. The best performing rule from literature is LST. In Table 8 we can see that the best generalization performance among all the evolved rules for GP-AR within the 31 runs is better than those of the selected rules in Table 9.

4.4.3. Performance breakdown

A further breakdown of performance based on average deviation and makespan is given in Table 10. This shows how the selected rules and the existing human designed rules perform based on instances of different sizes (number of activities). The biggest performance gap can be seen on J120 instances where our evolved rules perform extremely well. Correspondingly, it may be expected that GP evolved rules will generally perform better as the size of the problem increases making them more suitable for large scale dynamic conditions. It is also interesting to note that even though the evolved rules were not explicitly trained on J90 and J120 instances, they are still able to generalize really well. This indicates the effectiveness of the evolved rules in adapting well to new unseen problem instances. It must also be noted that in general J120 instances are constructed using a relatively lower set of RS values. For J120, RS ranges from 0.1 to 0.5. For J30, J60 and J90 the range used is 0.2 to 1.0. Previous research has shown that instances with low RS values are generally harder to solve. Hence, this also shows that our rules perform better as the instance set becomes harder. In terms of raw makespan values, AR_{ssgs2} gives an overall improvement of over 600 time-units over LST on the test set. Even the worst performing rule out of the 4, MRG_{ssgs2}, gives an improvement of over 300 time-units over LST. Among the existing rules, LST and LFT seem to be robust in the sense that their performance does not show a dramatic deterioration as the instance size increases. With all other existing rules, a reasonable performance gap is already visible on the smaller instances, which further increases with problem size.

We also do an instance-wise comparison of performance. Table 11 shows the number of instances in which our evolved rules achieve better/worse/equivalent makespan in comparison to LST. In every case the instance count favours our evolved

[1] Note: test set is not used during this selection, to keep it completely unseen. It is possible that some of the other evolved rules (not selected) had even better performance on test set.

Table 10
Performance breakdown on test set using serial SGS.

Rule	Percentage deviation				Makespan			
	J30	J60	J90	J120	J30	J60	J90	J120
EST	24.32	24.16	23.24	60.55	21720	29929	51300	91164
EFT	27.17	26.74	25.79	64.38	22212	30547	52371	93341
LST	19.88	17.78	16.07	46.74	20944	28368	48299	83274
LFT	20.86	18.52	16.67	48.11	21080	28549	48533	84039
SPT	34.56	34.56	32.16	77.94	23448	32376	54962	100942
FIFO	25.45	23.83	21.81	57.79	21890	29805	50656	89496
MTS	21.78	19.41	17.59	50.32	21255	28747	48912	85239
GRPW	25.88	26.46	24.99	65.28	21970	30413	51971	93693
GRD	27.62	28.41	26.96	68.42	22254	30868	52765	95515
AR_{SSGS_1}	19.83	17.63	15.99	46.15	20933	28342	48258	82940
AR_{SSGS_2}	**19.64**	**17.36**	**15.93**	**45.96**	**20893**	**28277**	**48214**	**82816**
MRG_{SSGS_1}	19.83	17.62	16.06	46.09	20930	28346	48283	82910
MRG_{SSGS_2}	19.91	17.59	16.03	46.16	20943	28341	48272	82947

Table 11
Comparing the number of instances in which the evolved rules are better/worse/equivalent in performance when compared against LST on the test set.

	Better	Worse	Equivalent
AR_{SSGS_1}	406	332	1014
AR_{SSGS_2}	410	314	1028
MRG_{SSGS_1}	**412**	362	978
MRG_{SSGS_2}	367	**307**	1078

Table 12
Deviation on instances with RS<= 0.25 and RF = 1 within the test set.

Rule	% Deviation	Makespan
LST	113.51	23292
AR_{SSGS_1}	**112.62**	**23196**
AR_{SSGS_2}	113.29	23249
MRG_{SSGS_1}	112.98	23230
MRG_{SSGS_2}	113.23	23264

rules. Furthermore, through Wilcoxon Two-Sided Signed Rank Test it was verified that the differences obtained using the evolved rules over LST on the test set are statistically significant.

4.4.4. Comparison on difficult instances

Within the PSPLib instance set there exist a subset of instances that are in general considered more difficult to solve. These difficult instances are characterized by high demand for resources ($RF = 1$) coupled with low availability ($RS <= 0.25$). In Table 12 we compare the performance of the evolved rules on these set of difficult instances. Once again, the evolved rules always outperform LST. Biggest improvement both in terms of makespan and percentage deviation is given by AR_{ssgs1}. This reinforces the strength of the evolved rules for not only the larger instances but also more difficult instances.

4.4.5. Further generalization analysis

To further test the generalizability of the evolved rules, we have used the complete RG300 instance set proposed in [10]. This set contains 480 instances, each having 300 activities with 4 resources. These instances are constructed using three key parameters, namely order strength, resource usage and resource constrainedness. It is also worth noting that these instances have been constructed using a different parameter set and the GP-evolved rules have not touched this set at all during the training phase. The RG300 instance set has not been used in any of the previous studies [14,33] on GPHH for RCPSP.

For this analysis we have focused on two key aspects. First is the cumulative makespan, which is summarized in Table 13. The best performing rule on this new instance set is MRG_{SSGS_1} which gives an improvement of 2339 time units over LST across the 480 instances. Interesting to note that AR_{SSGS_2} which was the best on the PSPLib test set is now the worst among the 4 selected rules. Nevertheless AR_{SSGS_2} also gives a significant improvement of over 1000 time units. Secondly we focus on the instance count which is summarized in Table 14. In all cases the 'better' count exceeds the 'worse' count substantially.

Table 13
Cumulative makespan comparison on RG300 set.

Rule	Makespan
LST	355605
AR_{SSGS_1}	353556
AR_{SSGS_2}	354197
MRG_{SSGS_1}	**353266**
MRG_{SSGS_2}	353639

Table 14
Comparing the number of instances in which the evolved rules are better/worse/equal in performance when compared against LST on RG300 set.

	Better	Worse	Equal
AR_{SSGS_1}	273	129	78
AR_{SSGS_2}	249	159	72
MRG_{SSGS_1}	**276**	129	75
MRG_{SSGS_2}	273	**125**	82

Table 15
Comparison of test performance for the different GP approaches: parallel SGS.

Rule	AR	MRG	MRL
Mean	25.83	25.83	**25.79**
Median	**25.72**	**25.72**	25.77
Best	25.59	**25.50**	25.62
Worst	26.68	26.67	**26.26**
Standard Dev.	0.29	0.27	**0.14**

Table 16
Comparison with existing priority rules: parallel SGS.

Rule	Test Perf.	Rule	Test Perf.	Rule	Test Perf.
Random	37.37	*EST*	33.57	MRG_{PSGS_1}	**25.57**
LST	26.42	*LFT*	26.44	MRL_{PSGS_2}	25.78
SPT	36.64	*MTS*	27.62	AR_{PSGS_1}	25.66
GRD	36.79	*IRSM*	27.38	MRG_{PSGS_2}	25.64
WCS	25.95	*GRPW*	34.54	AR_{PSGS_2}	25.78
EFT	33.88	*ACS*	26.10	MRL_{PSGS_1}	25.79
FIFO	31.19				

4.5. Results: parallel SGS

4.5.1. Performance

As mentioned in the study of Kolisch [20], priority heuristics perform better with parallel SGS. This is apparent when one compares the performance of the GP approaches on serial SGS (Table 8) vs. parallel SGS (Table 15). The rules evolved using parallel SGS on average perform significantly better.

The three GP approaches (GP-AR, GP-MRG and GP-MRL) have nearly equal median performance using Parallel SGS as shown in Table 15. In general, the mixed representation based GP approaches failed to show any major benefit. The rule with the best overall generalization performance was produced by GP-MRG which means there is definitely potential for discovering better quality rules with the mixed representation but further strategies may need to be sort to allow for consistently good performance. Investigation into specialized crossover/mutation, repair techniques and better ways of selecting diverse instances for training could form interesting directions for future research in an effort to improve GP-MRG/MRL.

4.5.2. Comparison with existing priority rules

Two best rules based on validation performance from each GP approach are selected for evaluation on the test set. Table 16 gives a comparison of the selected rules with those from literature. The best existing rule from the comparison list turns out to be WCS for this case. It can be seen that all of the selected evolved rules outperform the existing priority rules. The biggest improvement is shown by MRG_{psgs1}. Once again this shows the strength of the different GP approaches in producing rules that are able to generalize well to new unseen instances. The performance of the rules from literature is mostly in-line with the results produced by previous studies such as [20].

Table 17
Performance breakdown on test set using parallel SGS.

Rule	Percentage deviation				Makespan			
	J30	J60	J90	J120	J30	J60	J90	J120
EST	22.75	22.06	21.45	55.78	21452	29424	50554	88471
EFT	23.40	22.49	21.82	55.77	21566	29539	50720	88472
LST	18.93	17.60	15.80	44.04	20787	28338	48191	81753
LFT	18.78	18.05	15.90	43.86	20758	28455	48238	81653
SPT	25.48	24.14	23.60	60.33	21905	29918	51431	91012
FIFO	21.86	20.86	19.47	51.57	21296	29126	49689	86008
MTS	19.35	18.62	16.70	46.03	20872	28589	48563	82863
GRPW	22.76	22.82	21.83	57.87	21447	29598	50660	89546
GRD	24.71	24.58	23.18	61.30	21786	29986	51223	91464
IRSM	18.91	18.12	16.44	46.06	20791	28471	48454	82910
ACS	18.34	17.21	15.77	43.69	20693	28251	48185	81559
WCS	18.13	17.38	15.40	43.57	20656	28292	48029	81492
AR_{PSGS_1}	18.40	17.42	15.29	**42.63**	20704	28298	47978	**80961**
AR_{PSGS_2}	18.34	17.38	15.35	42.98	20689	28288	48008	81163
MRG_{PSGS_1}	18.16	**17.12**	15.26	42.72	20660	**28228**	47963	81002
MRG_{PSGS_2}	**18.11**	17.43	15.27	42.76	**20650**	28300	47974	81025
MRL_{PSGS_1}	18.54	17.29	**15.23**	43.05	20730	28270	**47950**	81197
MRL_{PSGS_2}	18.19	17.42	15.43	42.98	20661	28298	48042	81142

Table 18
Comparing the number of instances in which the evolved rules are better/worse/equal in performance when compared against WCS on the test set.

	Better	Worse	Equal
AR_{PSGS_1}	478	450	824
AR_{PSGS_2}	446	417	889
MRG_{PSGS_1}	478	**397**	877
MRG_{PSGS_2}	446	402	904
MRL_{PSGS_1}	**483**	428	841
MRL_{PSGS_2}	448	463	841

4.5.3. Performance breakdown

Next, we perform a further breakdown on the performance of the evolved rules and compare it with WCS. Table 17 shows how the performance is affected by instance size for all the existing as well as selected evolved rules. As observed previously, the evolved rules perform better as the size of the problem increases, with a big chunk of the improvement visible on larger J120 instance set. One of the main reasons for this big gap in performance on the J120 instances is the fact that the J120 instances in general can be classified as harder to solve because of the relatively smaller values of RS used to construct these instances. MRG_{psgs1}, which is the best in terms of cumulative makespan, gives an improvement of 616 time-units over WCS. AR_{psgs2}, which is the worst among the selected rules, gives an improvement of 321 time units over WCS. The benefit of our evolved rules is clearly visible over the large and diverse test set. In terms of the existing rules, LST, LFT, WCS and ACS exhibit robust performance. While their performance is significantly worse of in comparison to our selected rules, their performance can still be considered mostly stable across instances of different sizes and there is a clear separation between these four rules and the other existing rules within the comparison set.

The instance-wise comparison between the evolved rules and WCS is shown in Table 18. Except for MRL_{PSGS_2}, all other selected rules dominate WCS in terms of instance count. However, as mentioned previously, all of the selected rules outperform WCS in terms of overall percentage deviation (Table 16). This indicates that in the cases where the evolved rules are worse-off, the difference is relatively small; whereas in cases where they are better off, the relative improvement is significant. Once again, Wilcoxon Signed Rank test was performed in order to check the statistical significance of the performance of the six rules on test set compared to WCS. The test concluded that all differences were statistically significant, except for the case of MRL_{PSGS_2} which turned out to be statistically equivalent to WCS.

4.5.4. Comparison on difficult instances

Comparison against WCS on the more difficult instances (characterized by high resource demand ($RF = 1.0$) and low availability ($RS <= 0.25$)) is given in Table 19. Once again, all of our evolved rules outperform WCS on these instances. Rule AR_{psgs1} gives the biggest improvement over WCS in terms of makespan and percentage deviation.

4.5.5. Further generalization analysis

We now compare the performance of our selected rules against WCS on the RG300 set. As mentioned previously, we have not considered this set during the training or validation phase. Table 20 indicates the cumulative makespan performance.

Table 19
Deviation on instances with RS<= 0.25 and RF = 1 within the test set.

Rule	% Deviation	Makespan
WCS	104.06	22237
AR_{PSGS_1}	**101.79**	**21986**
AR_{PSGS_2}	102.95	22118
MRG_{PSGS_1}	102.52	22060
MRG_{PSGS_2}	102.08	22022
MRL_{PSGS_1}	102.73	22083
MRL_{PSGS_2}	102.57	22076

Table 20
Cumulative makespan on RG300 set.

Rule	Makespan
WCS	347856
AR_{PSGS_1}	342618
AR_{PSGS_2}	343596
MRG_{PSGS_1}	**342576**
MRG_{PSGS_2}	342833
MRL_{PSGS_1}	345282
MRL_{PSGS_2}	342955

Table 21
Comparing the number of instances in which the evolved rules are better/worse/equal in performance when compared against WCS on RG300 set.

	Better	Worse	Equal
AR_{PSGS_1}	364	61	55
AR_{PSGS_2}	354	68	58
MRG_{PSGS_1}	370	60	50
MRG_{PSGS_2}	**376**	**55**	49
MRL_{PSGS_1}	298	120	62
MRL_{PSGS_2}	336	87	57

One can immediately notice the significant improvement provided by our evolved rules over the 480 instances. The best rule on this set is MRG_{PSGS_1}, which gives an improvement of 5280 time units. Even in the worst case (MRL_{PSGS_1}), the improvement is greater than 2500 time units. This is also reflected in Table 21 where the instance count is heavily skewed in favor of the evolved rules.

4.6. Rule analysis

In this section we only focus on the rules evolved on the parallel SGS because of their superior performance in comparison to rules evolved on serial SGS. Given in Table 22 are the trees obtained for the best performing (selected) rules for each GP approach on the parallel SGS. For GP-MRL although MRL_{psgs2} was better on test performance, we pick MRL_{psgs1} to be included in the table below since MRL_{psgs1} proved to be statistically better on the test set in comparison to WCS.

One can also notice that the rules are dominated by four attributes, namely *LF, LS, AvgRReq* and *TSC*. It is interesting to note that these attributes actually represent three key aspects of the problem. *LF* and *LS* represent the critical path, *TSC* captures the state of the precedence network and *AvgRReq* captures the resource constraints. This further reinforces our earlier justification on the selected attributes which in a diverse manner covered different aspects of the problem, namely the critical path, project network and the resource constraints. Another interesting aspect is the fact the rules are mostly composed of linear components and sub-expressions.

4.7. Comparison with other GP-based approaches

Previous studies [14,33] on this subject have used different attributes, operators, data sets and GP algorithms. In order to compare our technique with theirs, we incorporated the attributes and operators used by them in our GP-AR framework. We used the same settings and data sets as all of our proposed GPHH variants using the parallel SGS. The only difference is that, at each scheduling step, the activity with the highest priority value is picked from the eligible set to be scheduled next since this is the approach the authors of these respect works adopted. The results obtained across 31 runs are given in Table 23. A comparison with the results in Table 15 shows a significant improvement offered by using our attribute and operator sets. The attribute and operator sets used in [33] (based on a feature selection exercise) performed better in comparison to

Table 22
Selected evolved rules: GP trees.

AR_{psgs1}
(+ (Min (+ LS (− LF TSC)) LF) (+ (* LF (+ (* LS LF) (− LF AvgRReq))) (− (− (− LF TSC) AvgRReq) AvgRReq)))
MRG_{psgs1}
(if (>= RC 30%) (schedule: (+ (− (− LS TSC) AvgRReq) (+ (− 1 AvgRReq) (+ LF LF)))) (schedule: (+ (− (− LS TSC) AvgRReq) (+ (− LS TSC) LF))))
MRL_{psgs1}
(if (<= MinRA 90%) (schedule: (− (+ (− LS maxRReq) (+ LF LF)) (* (* AvgRReq maxRReq) maxRReq))) (schedule: (− (− (Max AvgRReq LF) TSC) (* (* AvgRReq LS) maxRReq))))

Table 23
Performance on the test set (Parallel SGS) using the attributes and operator sets from previous studies [14,33].

	Frankola et al. [14]	Šišejković [33]
Mean	27.44	26.96
Median	27.49	26.93
Best	27.04	26.6
Worst	28.13	27.65
Standard Dev.	0.24	0.26

those in[14] but both of them showed inferior performance compared to the proposed approach. The average and the best case performance of both these approaches are worse than LST and LFT on the test set.The results clearly show the benefit of using our attribute and operator set which allow us to generate rules that outperform more sophisticated rules such as WCS/ACS.

4.8. Further discussion and concluding remarks

The results presented above highlight the strengths and weaknesses of the GP evolved rules for solving RCPSP. For all cases the evolved rules outperformed the best performing rules in the literature. Rules evolved on parallel SGS significantly outperformed those evolved on serial SGS. GP-AR consistently showed better average performance across different schedule generation schemes. The GP-MRG/GP-MRL approaches also showed promise in evolving high quality rules, although their consistency needs further improvement. The relatively inferior performance (in average/median sense) of GP-MR compared to GP-AR can be attributed to some of the possible factors mentioned earlier, e.g. (a) the random crossover/mutation may not be as conducive to decision trees in MR as it is for pure arithmetic rules; (b) the set of decision attributes used to build the decision logic may not have been too effective in capturing instance, system or eligible set processes which can be used to derive good results, and (c) the procedure of selecting training and validation sets could also play a role in overall performance.

We would also like to reiterate that the priority rules developed above are not meant to compete with heuristic/metaheuristic optimization algorithms in terms of *optimality*. A solution obtained through an optimization algorithm would typically achieve better makespan specific to the given instance; whereas the priority rules are intended for understandability and quick deployment for generic cases and dynamic scenarios. Nevertheless, for the sake of completeness, we conducted studies comparing the above rules with existing optimizers. The effect of mutation rate was also studied (with different values between 0 and 0.3), and the performance of the proposed approach is found to be fairly robust to the mutation rate. Due to space limitations, the full results of these two extended studies are provided as an online supplementary document.

To conclude the study, we would like to reflect on the overall trade-off between the performance and complexity of the priority rules considered. We quantify complexity by counting the number of nodes in the GP Tree. For our evolved rules, this is a straight forward process while for the existing human designed rules, we went about converting the rules into a GP tree representation and then counting the nodes. More details on how the existing human designed rules were converted into GP tress is given in the online supplementary document. The overall summary is presented in Fig. 3, based on the parallel SGS results (since these are consistently better than serial SGS). We additionally include a 'Rand' rule in the plot to verify the intuition that if the activities are randomly scheduled, then the performance would suffer significantly, though the complexity is the least.

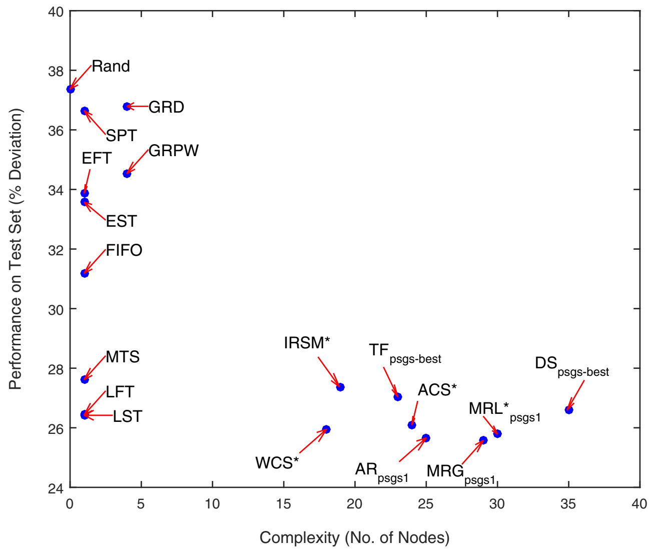

Fig. 3. Comparison of evolved and existing rules. The rules marked with a '*' are calculated at every time instant for the eligible set while the others are computed only once.

As can be seen in the plot, the evolved rules are better in terms of performance while the existing human-designed rules are in general simpler. A possible extension in this regard could be the use of a multi-objective GP [36] to more actively attempt to reduce the complexity of the evolving rules.

The rules AR_{psgs1} and MRG_{psgs1} have higher complexity in comparison to rules such as WCS and ACS. However, for these two rules, in a static environment such as the problems considered in this study, the priorities are pre-computed *only once*, before the scheduling process begins. On the other hand, the dynamic rules (highlighted using *) such as ACS, WCS, IRSM and even MRL_{psgs1} have to recompute the priority values of the eligible set at each time instant. Therefore the overall "operational" complexity of AR_{psgs1} and MRG_{psgs1} can in a way be considered competitive to these rules. It is worth highlighting that even for the makespan alone, the most recent study [33] did not report better solutions than LST, while using more complex expressions than those obtained in this study. The best rule (on the test set) evolved using the attribute and operator set given in [14] is represented on the plot as $TF_{psgs-best}$ while the best rule (on the test set) evolved using the attribute and operator set given in [33] is represented as $DS_{psgs-best}$. As expected, these two rules are dominated both in terms of performance and complexity by rules such as WCS and ACS and also some of the rules evolved using our approach. For the rules evolved using the attribute and operator set given in the previous studies, we have picked the best rule on the test set as opposed to our usual validation approach to give an indication of the best case scenario for these approaches.

Another interesting aspect worth considering is the arithmetic sub-rules within the decision rules. It is expected that the worst case complexity of any given arithmetic sub-rule within the decision rules will be less than the worst case complexity of any pure arithmetic rule since both are allowed the same depth yet a part of the decision rule is occupied by decision logic. In this sense, decision rules are more preferred since they can match the performance of pure arithmetic rules while offering better understandability and the likelihood of shorter arithmetic sub-rules to utilize in different scenarios. Shorter sub-rules will result in faster processing time. This is visible in MRL_{psgs1} and MRG_{psgs1} where majority of the branches result in significantly short arithmetic sub-rules. In this sense, one can argue that the complexity measure used in the plot above is not appropriately capturing the complexity of the decision rules or rather portraying them as more complex than they actually are. Nevertheless, since nearly all other rules in the plot are more or less purely-arithmetic the complexity measure used seems to be the best one in capturing or at-least providing a brief picture of the trade-offs that exist. Consider the comparison between MRG_{psgs1} and AR_{psgs1}. All of the sub-rules within MRG_{psgs1} are less complex than the stand-alone long arithmetic expression given by AR_{psgs1}. Between these two rules, MRG_{psgs1} will actually have a faster processing time. This is a strong reason to further explore the use of decision rules within this domain since they offer competitive performance with the possibility of better understandability (low complexity). In our case GP-MR was not able to fully exploit this benefit but the potential is definitely promising.

While our evolved priority rules show robust behavior across different instances and significantly outperform the existing priority rules, it should also be noted that existing rules such as LST, LFT, ACS and WCS also deliver robust performance. In general, these four rules are observed to perform better than the other existing human designed rules in our experiments. The PSPLib benchmarks seem to lack a certain level of diversity that prevents one rule or the other from completely standing out above all others. Future research could be focused towards a more detailed performance characterisation on these instances or the generation of new instances based on hardness and complexity with respect to existing and new heuristics.

5. Summary and future work

In this study we explored the use of GP for evolving priority rules for the classical RCPSP. Towards this end, we performed a systematic and comprehensive investigation using a basic GP framework. The study introduces a number of methodological and experimental developments over the existing literature, for example (a) use of a diverse set of attributes for constructing rules, (b) experimentation with both arithmetic and decision-based rules, (c) experimentation with both serial and parallel SGS, (d) use of lower bounds for target, so that no optimization runs are required for generating the training data; (e) normalization of the attributes to make the rules more generalizable while dealing with different orders of magnitudes, (f) use of smaller instances (J30, J60) only for training and all instances (including J90, J120, RG300) for testing to more appropriately reflect the scalability. Through extensive numerical experiments, the performance of the proposed approach was characterized. We hope that the discussion and the results mentioned in this paper will stimulate further interest in this research direction, eventually leading towards more advanced and more specialized evolution techniques for RCPSP.

For future research we will consider improving the quality of the evolved rules by incorporating better logic structures and exploring other system and activity attributes. We will also consider improving the GP framework itself by exploring the use of or designing better recombination operators to handle more complex rule structures, and use of a multi-objective framework by using complexity as one of the objectives explicitly. Some of the other interesting directions could be a more comprehensive investigation on the selection of appropriate attributes and training instances that can lead to improved generalization capability. Another long term goal would be to develop hyper-heuristics that are able to generate heuristics that are applicable across instances, problem variants and problem types. A possible direction in this regard could be identification of appropriate attributes and characteristics that maintain relevance across domains. Some inspiration can perhaps be taken from the research on evolutionary multi-tasking [15] which aims to develop algorithms that can solve problems of different sizes and types simultaneously.

Acknowledgments

The first author acknowledges the *2016 Graduate Student Research Grant* by the IEEE Computational Intelligence Society (CIS) and Prof. Frank Neumann from the University of Adelaide for discussions related to this work.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ins.2017.12.013.

References

[1] S. Asta, D. Karapetyan, A. Kheiri, E. Özcan, A.J. Parkes, Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem, Inf. Sci. (Ny) 373 (2016) 476–498.
[2] J. Blazewicz, J.K. Lenstra, A.R. Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Appl. Math. 5 (1) (1983) 11–24.
[3] J. Branke, S. Nguyen, C.W. Pickardt, M. Zhang, Automated design of production scheduling heuristics: a review, IEEE Trans. Evol. Comput. 20 (1) (2016) 110–124.
[4] J. Branke, S. Nguyen, C.W. Pickardt, M. Zhang, Automated design of production scheduling heuristics: a review, IEEE Trans. Evol. Comput. 20 (1) (2016) 110–124.
[5] P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: notation, classification, models, and methods, Eur. J. Oper. Res. 112 (1) (1999) 3–41.
[6] E.K. Burke, M.R. Hyde, G. Kendall, J. Woodward, Automating the packing heuristic design process with genetic programming, Evol. Comput. 20 (1) (2012) 63–89.
[7] J.A. Carruthers, A. Battersby, Advances in critical path methods, J. Oper. Res. Soc. 17 (4) (1966) 359–380.
[8] S. Chand, H.K. Singh, T. Ray, A heuristic algorithm for solving resource constrained project scheduling problems, in: 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 225–232.
[9] E.W. Davis, J.H. Patterson, A comparison of heuristic and optimum solutions in resource-constrained project scheduling, Manage. Sci. 21 (8) (1975) 944–955.
[10] D. Debels, M. Vanhoucke, A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem, Oper. Res. 55 (3) (2007) 457–469.
[11] M. Durasević, D. Jakobović, K. Knežević, Adaptive scheduling on unrelated machines with genetic programming, Appl. Soft Comput. 48 (2016) 419–430.
[12] S.E. Elmaghraby, Activity Networks : Project Planning and Control by Network Models, A Wiley-Interscience publication, Wiley New York London, 1977.
[13] C. Fang, L. Wang, An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem, Comput. Oper. Res. 39 (5) (2012) 890–901.
[14] T. Frankola, M. Golub, D. Jakobovic, Evolutionary algorithms for the resource constrained scheduling problem, in: ITI 2008 – 30th International Conference on Information Technology Interfaces, 2008, pp. 715–722.
[15] A. Gupta, Y.S. Ong, L. Feng, Multifactorial evolution: toward evolutionary multitasking, IEEE Trans. Evol. Comput. 20 (3) (2016) 343–357.
[16] S. Hartmann, Project Scheduling under Limited Resources, Springer, 1999.
[17] T. Hildebrandt, J. Branke, On using surrogates with genetic programming, Evol. Comput. 23 (3) (2015) 343–367.
[18] J. James E. Kelley, Critical-path planning and scheduling: mathematical basis, Oper. Res. 9 (3) (1961) 296–320.
[19] R. Klein, Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects, Eur. J. Oper. Res. 127 (3) (2000) 619–638.
[20] R. Kolisch, Efficient priority rules for the resource-constrained project scheduling problem, J. Oper. Manage. 14 (3) (1996) 179–192.
[21] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: theory and computation, Eur. J. Oper. Res. 90 (2) (1996) 320–333.
[22] R. Kolisch, S. Hartmann, Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, Springer US, Boston, MA, pp. 147–178.

[23] R. Kolisch, A. Sprecher, PSPLIB – a project scheduling problem library: OR software - ORSEP operations research software exchange program, Eur. J. Oper. Res. 96 (1) (1997) 205–216.
[24] G. Koulinas, L. Kotsikas, K. Anagnostopoulos, A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem, Inf. Sci. (Ny) 277 (2014) 680–693.
[25] J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press, 1992.
[26] S. Luke, Essentials of metaheuristics, 2009. URL: http://cs.gmu.edu/~sean/book/metaheuristics/.
[27] Y. Mei, X. Li, F. Salim, X. Yao, Heuristic evolution with genetic programming for traveling thief problem, in: 2015 IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 2753–2760.
[28] M. Mika, G. Waligóra, J. Weglarz, Modelling Setup Times in Project Scheduling, Springer US, Boston, MA, pp. 131–163.
[29] 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 Trans. Evol. Comput. 17 (5) (2013) 621–639.
[30] S. Nguyen, M. Zhang, M. Johnston, K.C. Tan, Selection Schemes in Surrogate-Assisted Genetic Programming for Job Shop Scheduling, Springer International Publishing, Cham, pp. 656–667.
[31] J. Park, S. Nguyen, M. Zhang, M. Johnston, Evolving Ensembles of Dispatching Rules Using Genetic Programming for Job Shop Scheduling, Springer International Publishing, Cham, pp. 92–104.
[32] B.D. Reyck, W. Herroelen, On the use of the complexity index as a measure of complexity in activity networks, Eur. J. Oper. Res. 91 (2) (1996) 347–366.
[33] D. Šišejković, Evolution of scheduling heuristics for the resource constrained scheduling problem, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2016 Master's thesis.
[34] V. Valls, F. Ballestin, S. Quintanilla, Justification and rcpsp: a technique that pays, Eur. J. Oper. Res. 165 (2) (2005) 375–386.
[35] M. Vanhoucke, J. Coelho, J. Batselier, An overview of project data for integrated project management and control, J. Mod. Project Manage. 3 (3) (2016).
[36] B. Wang, H.K. Singh, T. Ray, A multi-objective genetic programming approach to uncover explicit and implicit equations from data, in: Evolutionary Computation (CEC), 2015 IEEE Congress on, IEEE, 2015, pp. 1129–1136.
[37] R. Zamani, A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem, Eur. J. Oper. Res. 229 (2) (2013) 552–559.