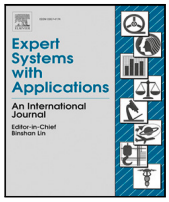
Expert Systems With Applications 233 (2023) 120916

Contents lists available at ScienceDirect

Expert Systems With Applications

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



Surrogate model for memetic genetic programming with application to the one machine scheduling problem with time-varying capacity

Francisco J. Gil-Gala *, María R. Sierra, Carlos Mencía, Ramiro Varela

Department of Computer Science, University of Oviedo, Campus de Viesques s/n, Gijón, 33271, Spain

ARTICLE INFO

Keywords:
Evolutionary computation
Surrogate model
Scheduling
Hyper-heuristics

ABSTRACT

Surrogate evaluation is a useful, if not the unique, technique in population-based evolutionary algorithms where exact fitness calculation is too expensive. This situation arises, for example, in Genetic Programming (GP) applied to evolve scheduling priority rules, since the evaluation of a candidate rule amounts to solve a large number of problem instances acting as training set. In this paper, a simplified model is proposed that relies on finding and then exploiting a small set of small problem instances, termed filter, such that the evaluation of a rule on the filter may help to estimate the performance of the same rule in solving the training set. The problem of finding the best filter is formulated as a variant of the optimal subset problem, which is solved by means of a Genetic Algorithm (GA). The surrogate evaluation of a new candidate rule consist in solving the instances of the filter. This model is exploited in combination with a Memetic Genetic Program (MGP); the resulting algorithm is termed Surrogate Model MGP (SM-MGP). An experimental study was performed on the problem of scheduling a set of jobs on a machine with varying capacity over time, denoted $(1, Cap(t)||\sum T_i)$. The results of this study provided interesting insights into the problems of filter and rules calculation, and showcase that the priority rules evolved by SM-MGP outperform those evolved by MGP.

1. Introduction

In many population-based evolutionary algorithms, where fitness evaluation is the most time-consuming part, surrogate or simplified models are often used to improve their performance (Bhattacharya, 2008; Branke & Schmidt, 2005; Hussein & Deb, 2016; Zeiträg et al., 2022; Zhou et al., 2007). In the literature, these models have been learned by means of different techniques, for example, regression analysis or neural networks (He et al., 2023). Some of these methods are based on predicting the performance of an individual without the need for evaluating it, whereas others use a simplification of the real fitness function.

For example, in Hildebrandt and Branke (2015), the authors present a technique for evaluating fitness in GP. Instead of directly assessing the fitness, they propose a strategy where the fitness of the new rule is determined based on the most similar rule found in previous generations of the GP. To facilitate this approach, the authors introduce a phenotype characterisation method that efficiently computes a numerical representation of the phenotype. The paper demonstrates the effectiveness of this approach by achieving the elimination of duplicates and reducing the number of expensive fitness evaluations required. Consequently, the proposed technique improves the speed and quality of GP, particularly in the context of the dynamic Job Shop Scheduling Problem.

An alternative approach is taken in Nguyen et al. (2017), where the authors propose using simplified models of the problem. They consider the dynamic Job Shop Scheduling Problem and analyse four simplified models under the assumption that "if rule a is better than rule b based on the absolute performance, the simplified model should also show that a is better than b". Therefore, they checked the rank-correlation between performances of rules on the original and simplified models using the Spearman's rank coefficient (Kendall, 1938, 1970; Spearman, 1904), and showed that the simplified model was the best one.

Building on these ideas, in Zeiträg et al. (2022), the authors extended these surrogate models to enhance the optimisation process in the multi-objective variant of the DJSSP. They proposed two approaches: one based on a simplified problem and the other based on machine learning techniques using samples of fully evaluated individuals based on the proposals by Hildebrandt and Branke (2015) and Nguyen et al. (2017).

In Gil-Gala et al. (2020b) proposed an exhaustive method incorporating simplified models called "filters" to eliminate evaluations for low-performing rules. This approach competes favourably with GP,

* Corresponding author.
E-mail addresses: giljavier@uniovi.es (F.J. Gil-Gala), sierramaria@uniovi.es (M.R. Sierra), menciacarlos@uniovi.es (C. Mencía), ramiro@uniovi.es (R. Varela).

https://doi.org/10.1016/j.eswa.2023.120916
Received 14 March 2023; Received in revised form 7 June 2023; Accepted 26 June 2023
Available online 1 July 2023

mainly when applied to limited-depth expression trees. These studies collectively emphasise the importance of surrogate evaluation in tackling the challenges posed by expensive fitness evaluations and enhancing the efficiency and effectiveness of GP in various problem domains.

In this work, we consider the Memetic Genetic Programming (MGP) approach proposed in Gil-Gala et al. (2021c) to evolve priority rules for the problem of scheduling jobs on a single machine with variable capacity over time, denoted $(1, Cap(t) \| \sum T_i)$. MGP extends the GP approach developed in Gil-Gala et al. (2019) with a Local Search Algorithm (LSA) specifically designed to compute dimensionally-aware priority rules (Keijzer & Babovic, 1999). In these algorithms, the evaluation of a candidate rule, coming either from recombination in GP or from a neighbourhood structure in LSA, requires solving a set of instances of the $(1, Cap(t) \| \sum T_i)$ problem, which is termed the *training set*. For the evaluation to be accurate, the number of instances in the training set and their size must be sufficiently large; therefore, fitness evaluation may be a very time-consuming process.

In order to reduce this computation time, we propose in this work a surrogate model (SM) relying on the hypothesis that, given a training set, it is possible to calculate another set, containing just a few instances of smaller size than those in the training set, so that the performance of a priority rule on this reduced set may estimate the performance of the rule on the training set. More specifically, given two candidate rules a and b, if a is better than b on the reduced set, then it is likely that the rule a will be better than b on the training set as well. These reduced sets of instances are called *filters* herein. The calculation of filters is formulated as the Optimal Filtering Set Problem (OFSP). This problem is solved by a Genetic Algorithm (GA); the first version of this algorithm and some preliminary results on the application of filters in GP were discussed in Gil-Gala et al. (2021b). The purpose of this method is to balance the use of exact and surrogate evaluation. From these previous experiments, it is clear that the best method consists in generating a number of offspring from each pair of mated chromosomes, which are evaluated on the filter. Then, only the offspring with the best estimation on the filter is evaluated on the training set. The resulting algorithm, termed SM-GP (Surrogate Model GP), outperformed the original GP.

Additionally, this paper presents a significant contribution by proposing a range of strategies to enhance the use of filters within the local search (LSA) component of the MGP algorithm. The use of filters allows LSA to discard many unpromising candidate neighbours. The resulting algorithms, collectively referred to as SM-MGP, exhibit remarkable improvements over existing state-of-the-art methods. Specifically, the extensive analysis conducted to explore different combinations of LSA and SM, showcases that the evolved priority rules obtained are generally smaller and more effective than the rules evolved by GP and MGP.

The remaining sections of the paper are organised as follows. In the next section provides the necessary background information, including the definition of the $(1, Cap(t) \| \sum T_i)$ problem and detailed explanations of the MGP approach introduced in Gil-Gala et al. (2021c) to evolve priority rules for this problem. In Section 3, we describe the proposed surrogate model, which is based on notion of filter; besides, we formulate the calculation of filters as a variant of the Optimal Subset Problem (OSP), and propose a Genetic Algorithm (GA) for its resolution. Section 4 presents different combinations of the surrogate model with MGP and the proposed general Genetic Programming (GP) framework. We explain how the surrogate model is integrated into the MGP approach and discuss the modifications made to the GP framework to incorporate the surrogate model. The experimental design is outlined in Section 5, where we provide details on the setup and parameters used for conducting the experiments. The results and analyses of these experiments are reported in Sections 6 and 7, respectively. We present the obtained results and compare them with the state-of-the-art methods. Additionally, we conduct in-depth analyses to gain insights into the performance of the proposed algorithms. Finally, in Section 8, we summarise the main conclusions of this work and propose potential avenues for future research.

2. Background and previous results

In this section we give the background necessary to understand the surrogate model proposed in this paper. We start with the definition of the sequencing problem of jobs on one machine with time-varying capacity, and then describe how to solve this problem by means of priority rules. Finally, we review the methods proposed to learn these rules, namely Genetic Programming (GP) and Memetic GP.

2.1. The $(1, Cap(t) \| \sum T_i)$ problem

In the $(1, Cap(t) \| \sum T_j)$ problem, n jobs, all of them available at time $t = 0$, must be scheduled on a single machine whose capacity varies over time as $Cap(t) \geq 0$, with $t \geq 0$.

The goal is to allocate starting times $st_j, 1 \leq j \leq n$, to the jobs such that:

1. at any time $t \geq 0$ the number of jobs that are processed in parallel on the machine, $X(t)$, cannot exceed the capacity of the machine, i.e., $X(t) \leq Cap(t)$, and
2. the processing of jobs on the machine cannot be preempted, i.e., $C_j = st_j + p_j$, where C_j is the completion time of job j and p_j is its duration.

The objective function is the total tardiness, which must be minimised. It is defined as $\sum_{j=1,\ldots,n} \max(0, C_j - d_j)$, where d_j is the due date of job j. Due to the fact that the $(1, Cap(t) \| \sum T_j)$ problem is an extension of the Parallel Identical Machines Problem (Koulamas, 1994), which is NP-hard, it follows that the $(1, Cap(t) \| \sum T_i)$ problem is NP-hard as well (Mencía et al., 2019).

The $(1, Cap(t) \| \sum T_i)$ problem was introduced in Hernández-Arauzo et al. (2015) in the context of scheduling the charging times of a large fleet of Electric Vehicles. Solving the above Electric Vehicle Charging Scheduling Problem (EVCSP) amounts to solving a number of instances of the $(1, Cap(t) \| \sum T_i)$ problem over time. Due to the computational intractability and the tight real-time requirements of the EVCSP, on-line scheduling represents the only suitable approach to the problem.

2.2. Using priority rules for solving the $(1, Cap(t) \| \sum T_i)$

Algorithm 1 shows a schedule builder proposed in Hernández-Arauzo et al. (2015) for the $(1, Cap(t) \| \sum T_i)$ problem. It is a greedy algorithm that in each step has to chose non-deterministically one among of a set of candidate jobs to be scheduled next. This choice may be done by means of a priority rule: the rule is evaluated for all options and the option taking the highest priority is chosen. In Hernández-Arauzo et al. (2015), the authors exploited the well-known Apparent Tardiness Cost (ATC) rule. However, many other rules could be exploited instead.

2.3. Evolving scheduling priority rules

The above ATC rule was designed from the intuition of experts in scheduling with due date objectives, and as with any other single rule, it may or may not be the best choice for a particular problem. According to the No-Free-Lunch theorem (Wolpert & Macready, 1997), it is expected that the same rule is not the best for any possible situation. For instance, in work by Sels et al. (2012), the simple Earliest Due Date (EDD) rule achieves better results than the ATC rule in some well-known instances of the Job Shop Scheduling Problem. To deal with the above issue, priority rules may be automatically designed and adapted to the structure of a given problem by means of some learning mechanism, which will hopefully be able to capture some problem characteristics that may not be intuitive for experts. In this context, Genetic Programming (GP) (Koza, 1992; Poli et al., 2008) stands out as a suitable approach due to the fact that priority rules may be naturally represented as expression trees, which may be evolved by GP outperforming other learning techniques, such as Neural Net-

Algorithm 1 Schedule Builder.

Data: A $(1, Cap(t)||\sum T_j)$ problem instance $\mathcal{P}$.
Result: A feasible schedule S for $\mathcal{P}$.
$US \leftarrow \{1, 2, ..., n\}$;
$X(t) \leftarrow 0, t \geq 0$;
while $US \neq \emptyset$ **do**
  // Calculate $\gamma(\alpha)$ as the earliest starting time for the next job
  $\gamma(\alpha) \leftarrow min\{t' | \exists u \in US; X(t) < Cap(t), t' \leq t < t' + p_u\}$;
  // Determine all jobs that can start at $\gamma(\alpha)$
  $US^* \leftarrow \{u \in US | X(t) < Cap(t), \gamma(\alpha) \leq t < \gamma(\alpha) + p_u$;
  Select a job $u \in US^*$ using a priority rule;
  // Schedule job u at $\gamma(\alpha)$
  $st_u \leftarrow \gamma(\alpha)$;
  $X(t) \leftarrow X(t) + 1, st_u \leq t < st_u + p_u$;
  $US \leftarrow US - \{u\}$;
end
return *The schedule* $S = (st_1, st_2, ..., st_n)$;

works (Branke et al., 2015). When GP is used to perform that task, it is classified as a hyper-heuristic based on heuristic generation, according to the taxonomy proposed by Burke et al. (2019).

The use of GP requires establishing a grammar to build priority rules, which includes a set of terminal and function symbols and some grammar rules. The terminal symbols represent attributes of the problem, such as job durations or due dates, and usually some numeric constants. The function symbols usually include basic arithmetic operators and some unary and binary functions. The grammar rules restrict the combinations of symbols to form feasible expressions. Additionally, a number of constraints are commonly considered to limit the size of the expression trees or to prevent the use of some partial expressions, relying on knowledge from the problem domain.

The use of GP also has its drawbacks, the most relevant one being the time required to evaluate the candidate rules. This procedure usually requires solving a large number of real instances of the problem at hand to assign a fitness value to each rule, which will be proportional to the performance shown by the rule on these instances. This approach was taken to deal with a number of optimisation problems, such as the Job Shop Scheduling Problem (Tay & Ho, 2008; Zhang et al., 2022), some variants of the Single Machine Scheduling Problem (Dimopoulos & Zalzala, 2001; Gil-Gala et al., 2019; Jakobović & Marasović, 2012), Unrelated Parallel Machines Scheduling (Durasević & Jakobović, 2018; Durasević et al., 2016; Jaklinović et al., 2021), Resource Constrained Project Scheduling (Chand et al., 2018; Dumić et al., 2018; Guo et al., 2021; Lin et al., 2020; Luo et al., 2022), Travelling Salesman (Duflo et al., 2019; Gil-Gala et al., 2022b), Arc Routing (MacLachlan et al., 2020; Wang et al., 2021), Electric Vehicle Routing (Gil-Gala et al., 2022a) or Bin Packing (Burke et al., 2012) problems, among others.

2.3.1. The search space of rules

The way in which GP based on hyper-heuristic generation is specifically applied to solve a particular optimisation problem is via an alphabet of symbols that are used to compose rules. Thus, the key element in GP is to define a grammar that establishes the constraints between the symbols in the alphabet; therefore, the grammar is in charge of defining the search space of rules. The other component that defines the search space is the maximum size of rules, i.e., the maximum number of nodes in the expression tree associated with the rule. In this work, we define it as $S = 2^D - 1$, where D is the maximum depth of expression trees.

In Gil-Gala et al. (2019), a GP approach was proposed to evolve priority rules for the $(1, Cap(t)||\sum T_j)$ problem. The grammar explored in Gil-Gala et al. (2019) generates dimensionally compliant expressions under the assumption that they are more rational than just arithmetically correct expressions. Two examples of rules are $p_j + 0.5$ and $p_j + d_j$, where only the second one is a dimensionally compliant expression. In addition, it drastically reduces the search space, making it possible to use alternative paradigms to GP, such as local search or state-space search (Gil-Gala et al., 2020a), and the classical rules are usually dimensionally compliant expressions as well. One of these rules is the ATC rule; one possible expression tree for encoding this rule is represented in Fig. 1, where g is a parameter that has to be superseded by any non-null numeric constant between 0.0 and 1.0.

Table 1
The alphabet of symbols composed by the functional and terminal sets used to build expression trees.

Binary functions	–	+	/	×	max	min	
Unary functions	–	pow_2	$sqrt$	exp	ln	max_0	min_0
Terminal symbols	p_j	d_j	$\gamma(\alpha)$	$\bar{p}$	0.1	…	0.9

Table 1 shows the terminal and function symbols used in Gil-Gala et al. (2019, 2021c) that compose the alphabet proposed in this work. On the one hand, the terminal symbols are the processing time p_j and due date d_j of each unscheduled job, while $\bar{p}$ is the average duration of the unscheduled jobs at a given time and $\gamma(\alpha)$ denotes the earliest possible starting time of the next job to be scheduled. Furthermore, some numeric constants are considered. On the other hand, the function symbols are restricted to the most basic ones in order to reduce the search space. Furthermore, some of them were selected because they are also contained in the ATC rule, and the rest due to being the opposite of them, such as exp is of ln, and max_0 is of min_0. Note that the symbol "-" is considered in unary and binary versions, and max_0 and min_0 return the maximum and minimum between any given expression and 0.

2.4. Memetic Genetic Programming

As far as we know, the Memetic Genetic Programming (MGP) approach proposed in Gil-Gala et al. (2021c), shown in Algorithm 2, represents the state-of-the-art in evolving priority rules for the $(1, Cap(t)||\sum T_i)$ problem. It is a generational evolutionary algorithm that starts from an initial population of $\#popsize$ random individuals. MGP iterates over a number of $\#gen$ generations using a selection procedure (line 4) in which the individuals are randomly organised into pairs of parents. The crossover and mutation operators (line 5), with probabilities p_c and p_m respectively, are applied over each pair of parents undergoing crossover and the resulting offspring are mutated. These genetic operators are the well-known *one point crossover* and *sub-tree mutation* that were adapted from the classical ones described in Koza (1992) to always generate feasible expression trees, i.e., they always generate dimensional compliant expressions. Then, the generated offspring are evaluated (line 6), assigning a fitness value to each of them. In the next step (line 7), each individual is improved by local search with probability p_{LS}, which is the main difference between GP and MGP. Finally, the replacement operator (line 8) passes the best two individuals from each pair of parents and their offspring to the next generation, but it always includes, at least, one of the offspring in order to avoid premature convergence as was proposed in Gil-Gala et al. (2019). In that way, it always keeps the best individual in each generation, which represents a form of elitism. Additionally, it forces that the next population is always composed of, at least, 25% of individuals from the previous population.

MGP uses two classical local search algorithms: Hill Climbing (HC) and Gradient Descent (GD). The difference between both is that, in each iteration, HC explores the neighbourhood until if finds a neighbour better than the current solution (priority rule), whereas GD evaluates all the neighbours and selects the best one of them. In both cases, the algorithm terminates when there is not any neighbour that improves the current solution. In Gil-Gala et al. (2021c) it was concluded that there were clear differences between HC and GD in favour of the second one, so in this work, we focus only on GD. The union of two

$$\frac{1}{p_j} exp\left[\frac{-max_0(d_j - (\gamma(\alpha) + p_j))}{g\bar{p}}\right]$$

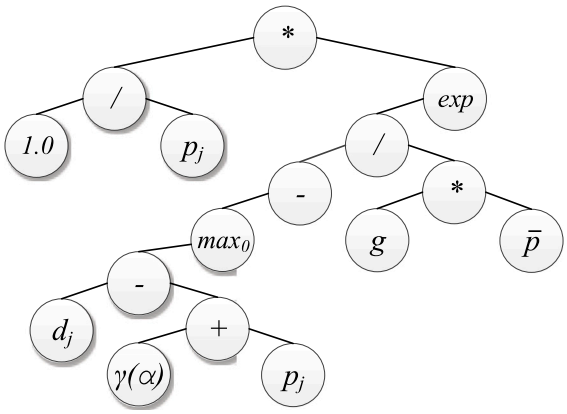


Fig. 1. Expression tree representing the ATC rule.

Algorithm 2 Memetic Genetic Program

Data: A set $\mathcal{M}$ of instances of the $(1, Cap(t)||\sum T_i)$ problem. Parameters: crossover probability p_c, mutation probability p_m, number of generations $\#gen$, population size $\#popsize$, local search probability p_{LS}.

Result: An expression tree representing a priority rule for the $(1, Cap(t)||\sum T_i)$ problem.

1: Generate the initial population $\mathcal{P}(0)$ with #popsize expression trees;
2: Evaluate $\mathcal{P}(0)$ on the set $\mathcal{M}$;
3: **for all** t=1 to #gen-1 **do**
4: **Selection**: organise the expression trees in $\mathcal{P}(t-1)$ into pairs of parents at random;
5: **Recombination**: mate each pair of parent expression trees and mutate the two offspring in accordance with p_c and p_m;
6: **Evaluation**: evaluate the resulting expression trees on the set $\mathcal{M}$;
7: **LocalSearch**: apply local search algorithm to the offspring in accordance with p_{LS};
8: **Replacement**: make a tournament selection of two expression trees from every two parents and their offspring to build the population in the next generation $\mathcal{P}(t)$;
9: **return** The best expression tree reached;

neighbourhood structures generates neighbours. The first structure is based on changing a single symbol in the rule, whereas the second structure changes a sub-tree in a similar way to what was proposed in Nguyen et al. (2019). However, MGP limits the neighbourhood only to generate dimensionally aware neighbouring rules. Additionally, the candidate expression to be changed in the second structure is limited to only those with a maximum depth of 2 and 3 symbols, such that they are also dimensionally compliant expressions. Despite using these constraints, the number of neighbours may be too high to be fully explored. Consequently, another version of MGP was proposed based on limiting the number of evaluated neighbours, which is termed MGPl. An exhaustive experimental analysis, in Gil-Gala et al. (2021c), concluded that the best configuration proposed was based on the union of both structures and limiting the number of neighbours to about 50, which are taken randomly and processed with GD. In this work, our hypothesis is that a suitable alternative approach is to use surrogate models to evaluate the whole neighbourhood, and only the best neighbouring rules are fully evaluated.

2.4.1. The learning model

When GP is used to perform the task of evolving priority rules, it is actually used as a machine learning algorithm, where the fitness function is the implementation of a learning model. Specifically, individual evaluation needs to solve a large pool of instances of the $(1, Cap(t)||\sum T_i)$ problem, the *training set*, and then the fitness value is calculated as the inverse of the sum of the total tardiness values obtained for all the instances, breaking ties in favour of rules with smaller size. Then, as usual, another set of unseen instances, the *test set*, is used to assess the performance of the best rule on the training set, in order to assess whether the rule generalises well or it is over-fitted to training data.

In the experimental studies reported in Gil-Gala et al. (2019, 2021c), the training set included 50 instances with 60 jobs each, which required large amounts of evaluation time, being quite similar to other works where GP has been used to calculate rules in other settings, such as Unrelated Machine Environment (Durasević & Jakobović, 2020) or Job Shop Scheduling (Nguyen et al., 2019).

3. Surrogate evaluation

As mentioned in Section 1, the use of surrogate models is a common and effective strategy to address the computational challenges posed by expensive fitness evaluations in evolutionary computation. These models may provide computationally efficient approximations of the fitness function, enabling faster optimisation and the exploration of more complex problem domains.

In Gil-Gala et al. (2020a), an exhaustive method was proposed to enumerate expression trees from the grammar described in Gil-Gala et al. (2019) for the $(1, Cap(t)||\sum T_j)$ problem. In order to reduce the number of actual evaluations on the training set, this method uses a simplified model that consists of 5 instances of 10 jobs each (instead of the 50 instances with 60 jobs of the training set). These small sets of instances were called *filters*, as they were used to filter each candidate priority rule so that if a rule is not better than another reference rule on at least 4 of the 5 instances, it is discarded. Otherwise, the rule is fully evaluated on the training set. In this way, most of the low-performing rules are not evaluated, and, as a result, the enumeration method is quite competitive with GP when the search is restricted to expression trees of limited depth. In this way, they were able to prevent their algorithm from evaluating most of the low performing rules so that the enumeration method is quite competitive with GP when the search is restricted to expression trees of limited depth.

In this work, we build on the idea presented in Gil-Gala et al. (2020a) and Nguyen et al. (2017) about the correlation between performance measures between real and simplified models and propose a method to obtain a good filter in the following terms: given a set of priority rules and their performances on both the training set and another set of simplified instances, the objective is to come up with a small set of simplified instances (that is, a filter), such that the correlation between the performances of the rules in the training set and the filter is maximised. The problem is referred to as the Optimal Filtering Set Problem, and it is formulated in the next section. To solve this problem, we propose a genetic algorithm, also described in the next sections.

3.1. The optimal filtering set problem

This section provides the formal definition of the Optimal Filtering Set Problem (OFSP), and how to use genetic algorithms to evolve filters. The Optimal Filtering Set Problem (OFSP) may be defined as a variant of the Subset Selection Problem as follows.

Given are:

- A problem $\mathcal{P}$.
- An ordered set of heuristics $H = \{h_1, \dots, h_n\}$ to solve $\mathcal{P}$.
- Two ordered sets $R = \{R_1, \dots, R_r\}$ and $S = \{S_1, \dots, S_s\}$ of instances of $\mathcal{P}$.
- X_{ij}, $1 \le i \le n$, $1 \le j \le r$, the performance measure of heuristic h_i on the instance R_j.
- Y_{ij}, $1 \le i \le n$, $1 \le j \le s$, the performance measure of heuristic h_i on the instance S_j.
- A parameter $k > 0$.

The goal is to find a subset $F \subset S$, $F = \{S_{[1]}, \dots, S_{[k']}\}$, $k' \le k$, where $[i]$, $1 \le [i] \le s$, $1 \le i \le k'$, is the index of the ith instance of F in S, such that

1. The correlation between the paired observations $X = \{X_1, \dots, X_n\}$ and $Y = \{Y_1, \dots, Y_n\}$, where
$$X_i = \sum_{j=1,\dots,r} X_{ij}, \qquad Y_i = \sum_{[j]=1,\dots,k'} Y_{i[j]}, \tag{1}$$
is maximised.
2. k' is minimised.

These two objective functions are considered hierarchically in the order they are declared, i.e., we will try to maximise the correlation, and only in the case of ties will we prefer the set F with the lowest cardinality.

Even though the Spearman's coefficient could be used as it was done in Nguyen et al. (2017), we opted to use the Kendall Tau-b (τ_b) coefficient (Kendall, 1938) to measure the correlation. The reason for this is that it is generally accepted that Kendall's coefficient is more robust than Spearman's (Colwell & Gillett, 1982), and that τ_b has a direct interpretation in terms of the probability of observing concordant or discordant pairs of observations; however, both tests usually yield a nearly identical result (Agresti, 2010; Croux & Dehon, 2010). Given two observations X and Y, the n pairs of values, (X_i, Y_i) and (X_j, Y_j), $1 \le i < j \le n$, are concordant iff the expressions $(X_i - X_j)$ and $(Y_i - Y_j)$ are both greater than 0 or lower than 0; they are discordant iff one of them is greater than 0, and the other is lower than 0, and they have tied iff at least one of them is 0. The Kendall Tau-b coefficient is defined as:

$$\tau_b = \frac{(n_c - n_d)}{\sqrt{(n_0 - n_1) * (n_0 - n_2)}} \tag{2}$$

where

- $n_0 = n * (n - 1) / 2$ is the number of pairs of observations.
- n_c is the number of concordant pairs.
- n_d is the number of discordant pairs.
- n_1 is the number of tied pairs on X.
- n_2 is the number of tied pairs on Y.

Values of τ_b range from −1 to +1, where −1 is 100% negative association, +1 is 100% positive association, and a value of zero indicates the absence of association. In this way, a set F maximising τ_b may be expected to be a good filter for the set of heuristics H solving the set of instances R, in the sense that if we consider two heuristics h_i and h_j in H such that h_i is better than h_j in solving the set of instances F, then it may be expected that h_i will be better than h_j in solving the instances in R as well. Besides, if we develop a new heuristic h following a similar method to that used to develop the heuristics in H, then a similar behaviour may be expected for h, and so the set F could be used for surrogate evaluation of new heuristics on the set R, which would be particularly useful if the instances in the set S are (much) smaller than those instances in R and the cardinality of F is (much) lower than that of R.

As pointed in Hamo and Markovitch (2005), decision problems associated with subset selection are typically NP-hard or NP-complete. The decision version of the OFSP, given a correlation threshold and a value of k, is in NP. Besides, the search space is exponential in the size of S; therefore, in spite of the absence of a formal proof that some NP-hard problem may be polynomially reduced to OFSP, we will treat it as if it was NP-complete, and so its optimisation version NP-hard.

3.1.1. The genetic algorithm for the OFSP

Genetic algorithms have been recently applied to solve some variants of the optimal subset problem (Kromer et al., 2018). In this work, we propose a genetic algorithm (GA) to solve the OFSP. A preliminary version of this algorithm was presented in Gil-Gala et al. (2021a). The main components of GA were chosen as follows.

Coding scheme. Chromosomes are given by variations with repetition of instances from S taken k by k; so, there are s^k different chromosomes in all. Allowing for repetitions, a chromosome actually represents a subset $F \subset S$ with $k' \le k$ instances, namely, the set of instances in the chromosome. In this way, F is a candidate solution to the OFSP.

Initial population. Initial chromosomes are random variations taken uniformly. However, other strategies could be used as well, for example, giving each rule a probability proportional to the average value of the solutions obtained by that rule on the training set, i.e., the fitness value of the rule in the GP proposed in Gil-Gala et al. (2019).

Crossover. Given two parent chromosomes and assuming that their k elements are sorted in each chromosome as they are in S, a random binary string of length k is generated, taking each bit uniformly in $\{0, 1\}$. Then, one offspring is generated, taking the elements of the first parent in the positions where the binary string has 0s and the elements of the second parent in the positions where the string has 1s; analogously, a second offspring is generated, swapping the roles of 0s and 1s. In this way, the offspring has the same chance to inherit characteristics from both parents.

Mutation. This operator has an important role as it is in charge of including new instances from the set S in the chromosomes, namely, the new genetic material in the population. The mutation operator we use here changes a number of instances, between 1 and $k/2$, of the chromosome by new instances chosen uniformly from S.

Evaluation. The fitness value of the chromosome is the τ_b value calculated by Eq. (2) between two arrays X and Y, which in turn are calculated by Eq. (1). In this case, X and Y represent the performances of the rules on the sets of instances R and F, respectively.

Evolutionary scheme. We use a generational GA with a random selection of parents, conventional mating and mutation, and replacement by tournament among every two mated parents and their two offspring, which confers the GA an implicit form of elitism, and which was initially proposed in Vela et al. (2010). A similar evolutionary scheme is summarised in Algorithm 2, however, GA does not use a local search algorithm as another operator. Additionally, it does not always enforce to include one of the offspring; thus, both parents could pass to the next population if both offsprings are worse than them.

4. Using filters for surrogate evaluation in Genetic Programming

As a first approximation, we have used the Surrogate Model (SM) based on the use of filters in combination with the GP devised in Gil-Gala et al. (2019) to evolve priority rules for the $(1, Cap(t) \| \sum T_i)$ problem. The resulting algorithm is termed SM-GP. To assess the viability of this approach, we start considering the following setting: first, a filter is calculated in a preprocessing step, and then this filter is exploited along the evolution of GP to evaluate chromosomes. To obtain the filter, we start from a set of instances $R = \{R_1, \ldots, R_r\}$ of current interest, another set of smaller instances $S = \{S_1, \ldots, S_s\}$, a set of rules $\Delta = \{\Delta_1, \ldots, \Delta_n\}$ gathered, for example, from previous executions of the GP, and a parameter $k > 0$. We also assume that the performance of these rules on the instances in R and S is known. From these data, a filter $F = \{S_{[1]}, \ldots, S_{[k']}\}, k' \leq k$ is calculated.

In some preliminary experiments discussed in Gil-Gala et al. (2021b), the filters were exploited for surrogate evaluation in accordance with one of the following three strategies:

- S_1. The whole evolution process is guided by surrogate evaluation. So, the evaluation of every chromosome, namely expression tree representing a priority rule, is performed by SM. In other words, the fitness of a candidate rule is assigned from the tardiness values produced by the rule on the instances of the filter F. In this setting, only the best individual in each generation will be fully evaluated on the training set to visualise the evolution of the algorithm and establish the final solution.
- S_2. The same as S_1, but with a different mating strategy: each pair of selected chromosomes are mated a number of times to produce a set of N feasible new candidates, which SM evaluates. Then, the candidate with the best estimation is selected to make a tournament among the two parents and another child generated in the same way.
- S_3. Similar to S_2, but the selected chromosome is fully evaluated on the set R. So, only in this case, the evolution is guided by actual evaluation.

Since S_1 and S_2 generate a number of N candidate rules, it is likely that one of them may be syntactically identical to one of its parents or at least semantically equivalent. For this reason, when a candidate rule solves the instances of F producing the same tardiness as one of its parents, we opted to select the next best candidate rule in solving the instances of F. The experiments carried out in Gil-Gala et al. (2021b) indicate that only S_3 is able to achieve better results than GP.

In the present study, we consider a number of previous approaches for the purpose of comparison, specifically:

1. GP, which is based on the framework proposed by Koza (1992). Still, it is delimited to the search space of dimensionally aware rules (Keijzer & Babovic, 1999) and using an alternative evolutionary scheme (Vela et al., 2010), which is based on a selection between each pair of parents and their offspring. Classical genetic operators (the punctual mutation and one-point crossover) are adapted only to generate dimensionally compliant expressions. More details are provided in Gil-Gala et al. (2019).
2. MGP, which combines GP with LSA as done in Algorithm 2. In this approach, all the neighbouring rules are fully evaluated; in this way, MGP spends most of the time doing local search, so it hardly evolves for a few generations. More details of this algorithm are provided in Gil-Gala et al. (2021c).
3. MGP-N, an extension of MGP with a limit of N neighbouring solutions, is termed MGPl in Gil-Gala et al. (2021c). This allows MGP-N to evolve for a larger number of generations than MGP, taking both at the same time. The exhaustive experimental analysis reported in Gil-Gala et al. (2021c) showed that about 50 to 200 neighbours (without statistical difference in this range) are good values of N to balance the execution time between LSA and the other genetic operators.
4. SM-N-GP, which is an extension of GP that uses a crossover operator implementing the strategy S_3, as in Gil-Gala et al. (2021b).

And we are considering the following new proposals:

5. MGP-SM-N, which is as MGP-N, but the N neighbours are selected exploiting the SM estimated fitness.
6. SM-N-MGP-N is combined with LSA considering only N neighbours chosen at random.
7. SM-N-MGP-SM-N, which enhances the above with a heuristic selection of the N neighbours by SM.

Furthermore, the last four strategies will be considered taking random filters (R) or filters calculated by GA (see Section 3.1.1). So, we have 11 variants in all.

Fig. 2 illustrates a flowchart outlining the main phases of the proposed GP framework. Initially, a population of rules is randomly generated using the Ramped Half-n-Half method introduced by Koza (1992). Next, the population is randomly organised, and two parents are selected. These parents can either undergo the crossover operator, resulting in two offspring, or they can be cloned to create two new individuals. During the crossover operator, the surrogate evaluation can be invoked to generate a set of N individuals, from which the one with the highest estimated fitness in the surrogate model is selected. Following that, the mutation operator may be applied to the two new individuals, after which they are evaluated. After that, the local search operator can be used further to improve the two new individuals in the next step. In the local search operator, the surrogate evaluation can be invoked again to discard unpromising neighbouring solutions, and only the top N individuals are fully evaluated. Finally, a tournament selection is performed using the resulting two individuals and the two parents. The best new individual automatically becomes part of the new population, while the second one is determined by comparing the other new individual with the two parents. This entire process is repeated until the new population is created and the stopping condition is met.

5. Experimental design

In this study, we aim to analyse the performance of GA in solving the OFSP and then to study how the calculated filters may improve the performance of the proposed GP variants. All the algorithms were implemented in Java 8, and the target machine was a Linux cluster (Intel Xeon 2.26 GHz. 128 GB RAM) that is able to perform up to 28 executions in parallel. To ensure statistical significance, a number of independent runs were performed for each algorithm and instance considered and when required, the Kruskal–Wallis test was used with the post-hoc Dunn test to assess statistically significant differences in the results.

5.1. The benchmark set

We consider the training and test sets of instances of the $(1, Cap(t) \| \sum T_i)$ problem proposed in Gil-Gala et al. (2020b), which include 50 and 1000 instances, respectively. These are large instances with 60 jobs each and a maximum machine capacity of 10 jobs. The generation procedure is described in Gil-Gala et al. (2020b). We also consider another set of 1000 small instances (10 jobs and maximum machine capacity of 3) generated by the same procedure. The training set plays the role of the set R in the OFSP, while the small instances give the set S. The test set is only used to evaluate the rules evolved on unseen instances to assess if they are over-fitted to the training set. The maximum size of the filters is taken as $k = 5$; in this way, we observed that the ratio between the time taken to evaluate a candidate rule on the set R and a filter is about 60/1.

The set of rules Δ is composed of 600 rules with different characteristics (100 for each group), as described in Table 2. We consider random rules (R) and the best rules evolved by Genetic Programming (GP) in 100 runs. In both cases, rules with a maximum depth of 4, 6 and 8 are considered.

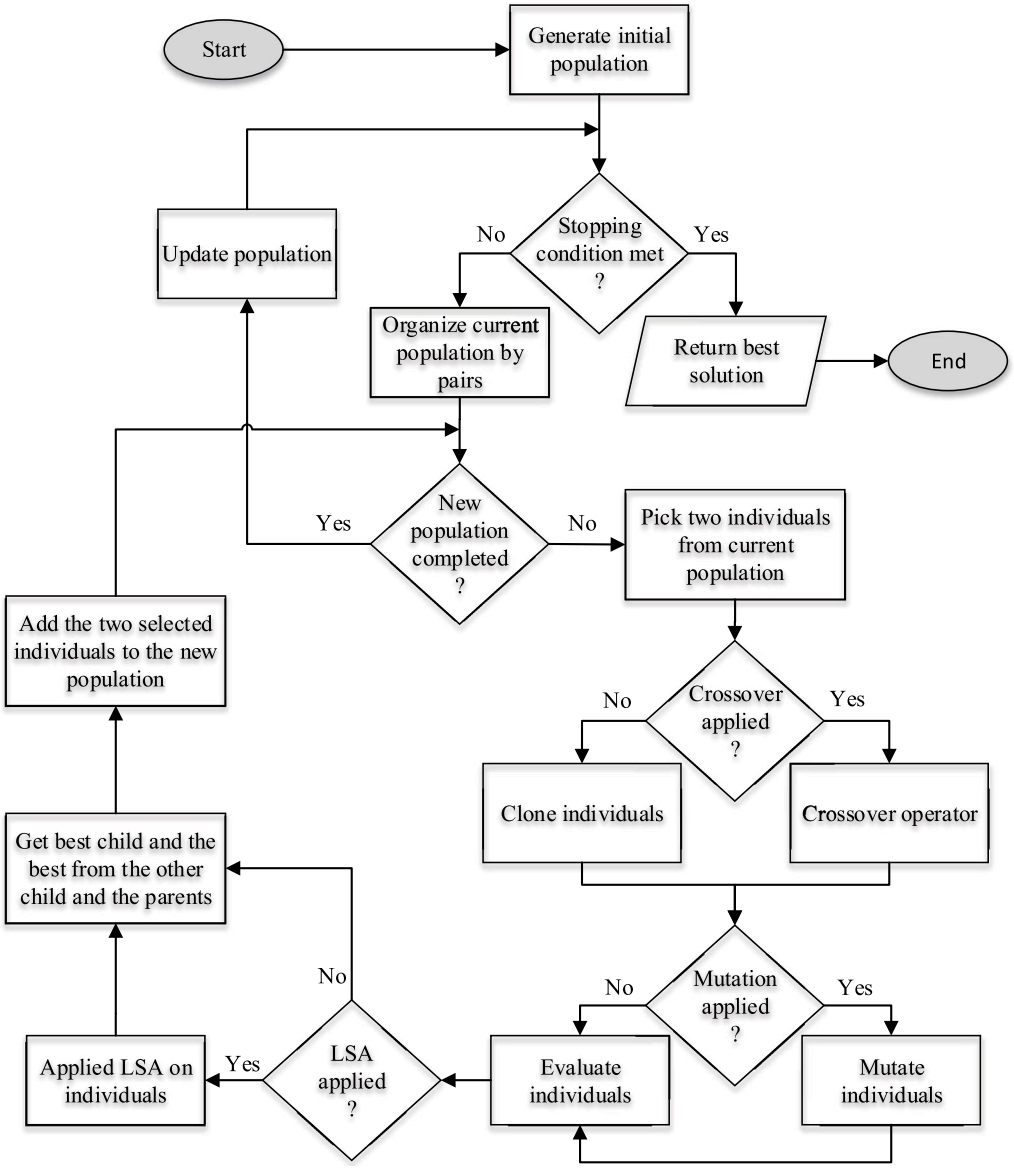


Fig. 2. Flowchart of the proposed GP framework.

Table 2
Characteristics of the 600 rules in the set Δ: generation method, maximum depth and average performance (tardiness values) on the training and test sets.

gen. Method	max. Depth	Training	Test
R	4	6859.43	6828.84
	6	6375.84	6340.91
	8	5448.26	5395.74
GP	4	1755.07	1732.57
	6	1679.36	1646.11
	8	1671.72	1630.91

Table 3
Parameters values for GA and GP.

Parameter	GA	GP
Population size	500	200
Crossover rate	0.8	1.0
Mutation rate	0.2	0.02
Chromosome length	5	$2^D - 1$
Stopping condition	500 gen.	1 h

5.2. GP and GA parameters

In this study, the parameters of GP and GA were set to the values reported in Table 3. These are in fact, rather conventional values, taken from some previous experiments reported in Gil-Gala et al. (2021b, 2021c). The short length of the chromosomes in GA is due to the maximum size of the filters being $k = 5$. But this fact does not make the problem easy, as there can be 1000^5 different chromosomes (given that we consider 1000 small instances in the set S), so the search space is very large. As the length of the chromosomes in GP is $2^D - 1$, the maximum depth of the expression trees is D, which is tested for values 4, 6 and 8. A time limit gives the stopping condition, instead of a number of generations to obtain comparable results, that was established in 1 h per run. Besides, GP uses the same filter in each run, which is chosen randomly from the 30 filters calculated by GA in 30 independent runs. Apart from comparing the different versions of GP, we also performed some preliminary experiments, in which SM-GP is considered with the three strategies, S_1, S_2 and S_3, described in Section 4.

Table 4
Fitness values of random filters and filters generated by GA. 30 filters were considered in each case with $k = 5$.

	Best	Avg.	Worst	SD
R	0.7862	0.7037	0.5884	0.0502
GA	0.8839	0.8834	0.8828	0.0004

Table 5
Results from GP and SM-GP with different F and N new feasible candidate solutions produced by the crossover operator. For each, 28 independent runs were performed with a time limit of 1 h. $Best_T$ is the result of the best in training on the test.

Method	N	F	Training			Test				
			Best	Avg	SD	Best	$Best_T$	Avg	SD	Size
GP			1617.22	1692.27	50.17	1637.63	1642.55	1722.18	52.17	42.30
S_1	1	R	1736.84	1840.21	55.61	1758.79	1758.79	1866.21	58.96	22.10
		GA	1625.92	1732.13	78.14	1642.48	1642.48	1746.16	79.98	24.60
S_2	10	R	1649.48	1771.16	85.15	1656.13	1656.13	1789.65	93.18	22.60
		GA	1621.22	1726.08	98.91	1638.33	1641.28	1759.11	103.12	21.80
	50	R	1643.76	1714.86	53.94	1661.93	1668.90	1730.10	58.91	21.80
		GA	1633.50	1660.61	21.63	1642.01	1662.30	1675.93	19.17	20.50
	200	R	1643.76	1729.49	71.78	1661.45	1668.90	1742.81	72.23	16.20
		GA	1648.08	1668.38	44.96	1641.43	1659.04	1678.94	42.05	16.60
	500	R	1712.68	1777.63	56.16	1728.72	1728.72	1787.44	49.92	20.60
		GA	1697.14	1792.12	62.04	1700.51	1700.51	1786.74	65.18	23.00
S_3	10	R	1611.60	1624.16	11.98	1637.96	1643.02	1652.97	14.99	31.96
		GA	1600.13	1620.05	23.47	1640.06	1640.06	1652.68	21.17	37.90
	50	R	1618.62	1639.04	24.32	1637.01	1637.01	1660.58	28.11	21.40
		GA	1610.87	1617.64	4.13	1637.74	1645.37	1643.98	4.67	31.40
	200	R	1649.13	1698.12	26.10	1670.29	1670.29	1722.13	25.98	22.00
		GA	1641.66	1672.87	22.12	1672.98	1674.13	1698.10	24.96	15.10
	500	R	1753.46	1804.19	33.54	1763.16	1772.19	1811.31	38.20	26.10
		GA	1658.17	1788.87	67.29	1673.51	1674.68	1798.13	71.58	22.20

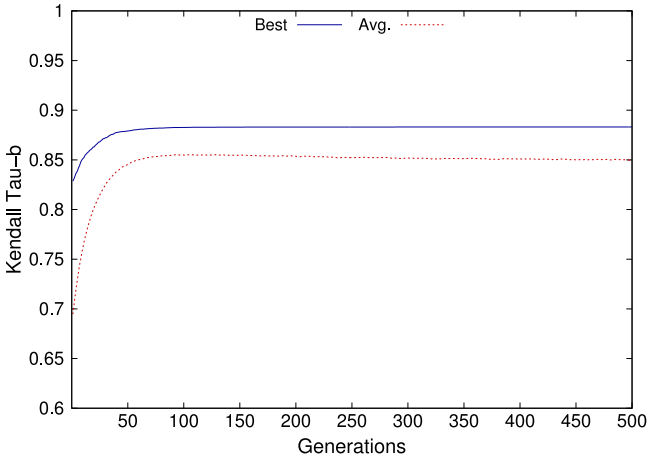


Fig. 3. Evolution of the average and best solution averaged for 30 independent runs of GA.

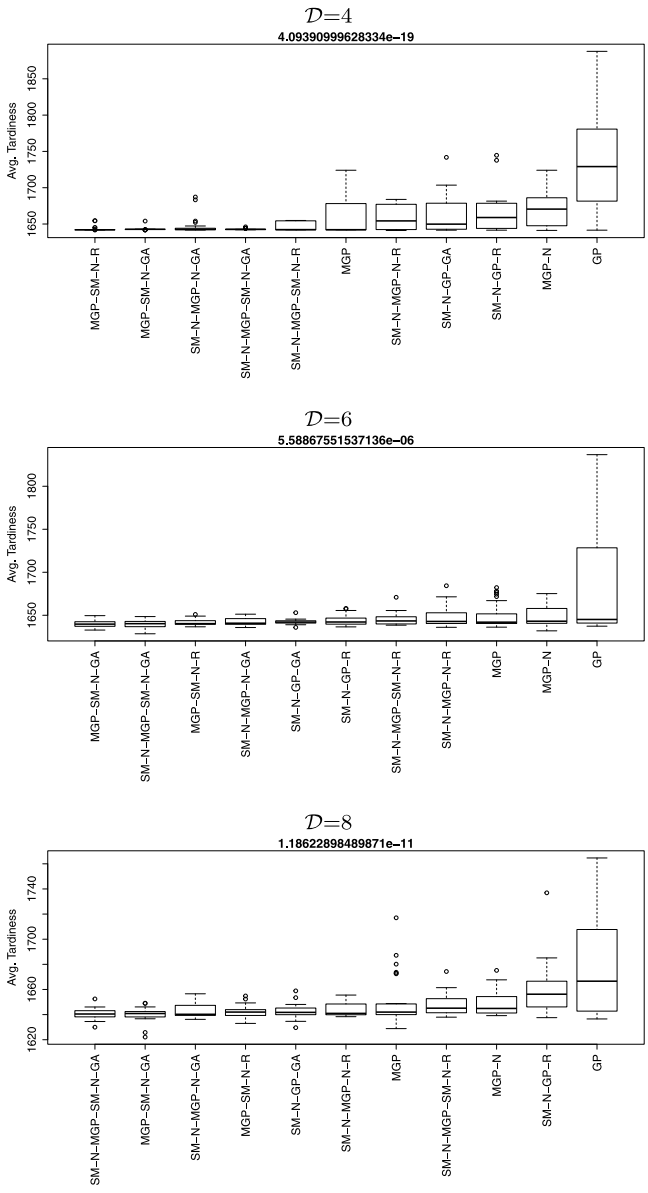


Fig. 4. Box-plots for each maximum depth tested (4, 6 and 8) with the 11 proposed GP variants solving the test set. They are sorted by their position on the Rank Mean for the results obtained by the Kruskal–Wallis Test. The number on the top of each box-plot is the p-value produced by the Kruskal–Wallis test.

6. Analysis of the results

In this section, we first analyse the results achieved by GA and then the performance of the GP variants executed under the conditions above.

6.1. Results of GA on the OFSP

Table 4 shows the best, average, worst and standard deviation (SD) of the fitness of the solutions, i.e. filters, reached by GA over 30 independent runs, together with the values obtained from a set of 30 random filters, which represent a baseline for comparison with GA.[1]

Fig. 3 shows the evolution of the best and average fitness values averaged for the 30 runs. We can observe that random filters produce a large fitness (about 0.7 on average), showing that the performance of the rules in Δ on the instances of even a random filter offers a high correlation with the performance of the rules on the training set. However, the filters evolved by GA show better fitness values, and they are much more stable, as indicated by the SD values. We conjecture that so low SD values may be due to the fact that the solutions are close to the optimal ones. At the same time, we can observe a good convergence pattern showing a low improvement after the first 50 generations. The most time-consuming part is solving the R and S instances sets with all rules in Δ. When it is done, the execution of GA is negligible (just needs some seconds), in contrast with GP, which needs a large number of minutes, in some cases even though hours (Durasević et al., 2016; Gil-Gala et al., 2019; Nguyen et al., 2019), to calculate the rules.

[1] Remember that the fitness values in GA range in the interval [−1,1].

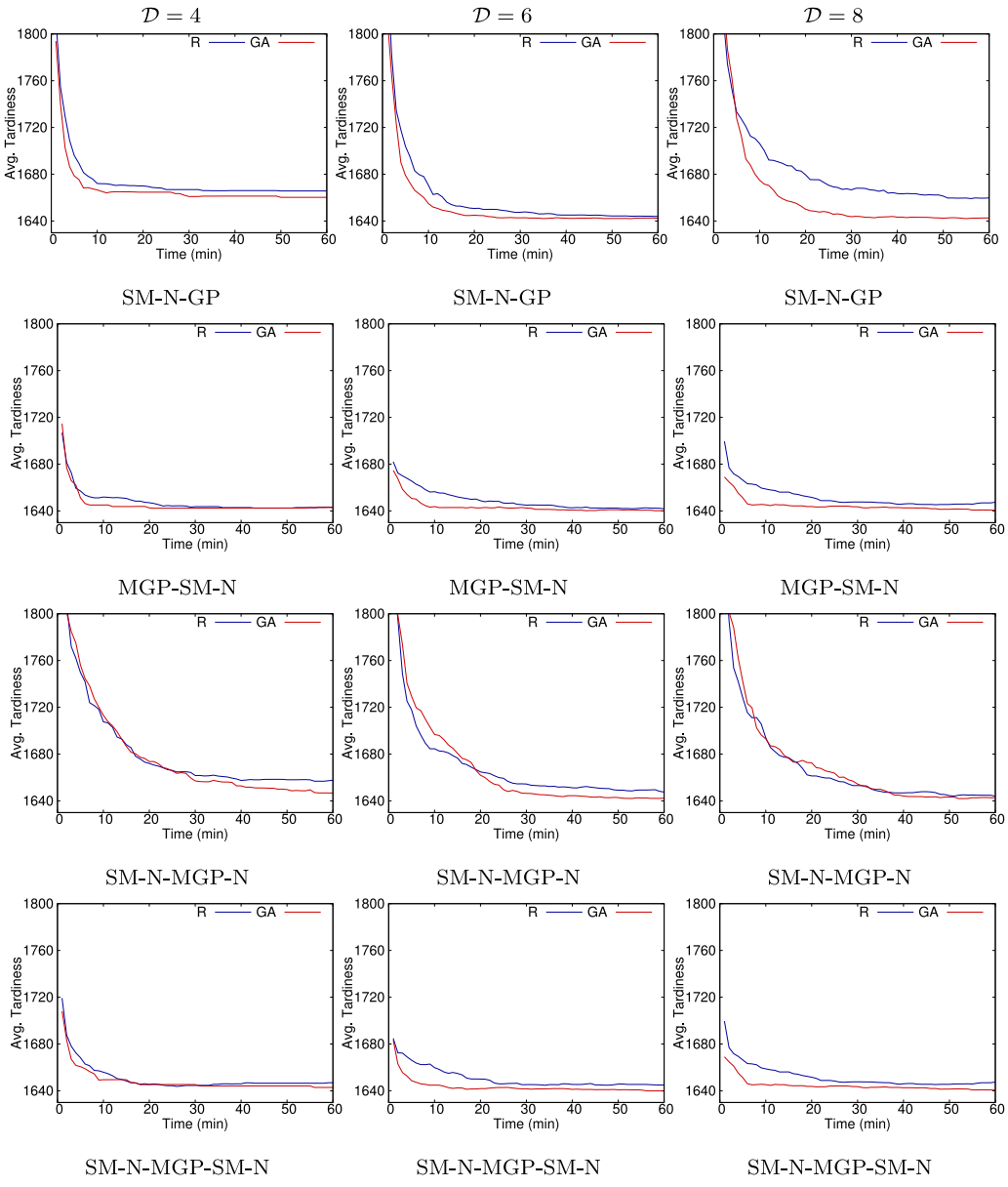


Fig. 5. Evolution of the best rule (averaged from 28 runs) on the test set over time (0–60 min) using a filter calculated by the Genetic Algorithm (GA) or randomly generated (R) in the search space defined by the values of D (4, 6 and 8).

6.2. Analysis of the SM strategies S_1, S_2 and S_3

To assess the viability of the proposed surrogate evaluation model, we conducted experiments with SM-GP exploiting each of the strategies S_1, S_2 and S_3. We considered a time limit of 60 min, different values of N in S_2 and S_3 (10, 50, 200 and 500) and rules with depth D=8. The SM model was applied with a random filter (R) and with a filter calculated by GA (in this case, one of the 30 filters evolved by GA was chosen at random). For each combination, 28 independent runs were performed.

The results are summarised in Table 5. From these results, we may draw the following conclusions:

- In every case, a filter evolved by GA performs better than a random one.
- The only combinations that are able to outperform GP are S_2 and S_3, with N taking the values 50 and 200. For larger values of N, i.e. 500, SM-GP only completes a small number of generations in the given time, and so it is unable to converge to good solutions. On the other side, choosing the best estimated of just 10 offspring does not produce enough selective pressure for a proper convergence, in particular when using S_2, which relies on surrogate evaluation only.
- S_3 is better than S_2, which is reasonable due to the exact evaluation of the best estimated offspring. It seems clear that SM-GP exploiting S_3 with a filter evolved by GA and $N = 50$ is the best one of the evaluated options.

6.3. Results of the GP variants

This section reports the results achieved by the 11 variants of GP described in Section 4. The stopping condition was established in 60 min, and the maximum depth D was set to the values 4, 6 and 8. Each experiment was performed considering both a random filter (R)

Table 6
Adjusted p-values for N ×1 comparisons of test control algorithm, for different values of D (4, 6 and 8), with the Bonferroni algorithm for the Dunn's post-hoc test on test executions. Cells with "✓" express that there are significant statistical differences, empty cells mean that there are not.

Algorithm	Bonferroni adjusted		
	$D = 4$	$D = 6$	$D = 8$
	MGP-SM-N-R	MGP-SM-N-GA	SM-N-MGP-SM-N-GA
GP	✓	✓	✓
MGP	✓	✓	
MGP-N	✓	✓	✓
SM-N-GP-R	✓		✓
SM-N-GP-GA	✓		
MGP-SM-N-R	–		
MGP-SM-N-GA		–	
SM-N-MGP-N-R	✓	✓	
SM-N-MGP-N-GA			
SM-N-MGP-SM-N-R			✓
SM-N-MGP-SM-N-GA			–

and a filter evolved by the genetic algorithm (GA). When appropriate, N=50 for both the number of neighbours and the number of chromosomes generated by the crossover operator. For each configuration, 28 independent runs were performed, and the best rule from each run was registered.

The results are summarised in Fig. 4, which shows the box-plots produced by the 28 rules on the test set for each value of D. In each figure, the methods are sorted from left to right in accordance with the ranking established by the Kruskal–Wallis test. At a first glance, we may observe that the methods that combine the surrogate and exact evaluation of some chromosomes are the best ones. To assess the statistical differences between the methods, we performed a post-hoc analysis using the Dunn test with Bonferroni correction for each value of D with the best algorithm in the ranking as the test control algorithm. The result of this analysis is shown in Table 6, where we can observe that, in general, there are no significant differences between the methods that exploit the surrogate model for neighbouring estimation and offspring selection. In particular, none of the analysed methods is statistically better than SM-N-MGP-SM-N-GA, which combines all the above options. So, the results confirm the usefulness of the proposed surrogate model to improve the performance of a genetic program in evolving scheduling rules.

7. Further analyses

In order to gain a better understanding of the differences among the proposed GP variants, a time-based analysis of algorithm evolution was conducted. At intervals of 1 min, ranging from 0 to 60 min, the best-performing rule in the training set from the current population was recorded. Subsequently, these rules were evaluated on the test set. Furthermore, a tuning parameter analysis was carried out, and the runtime required for applying the surrogate model (SM) was compared to that of the full fitness evaluation.

7.1. Evolution with random and GA filters

Fig. 5 shows the convergence patterns averaged for the three values of D (4, 6 and 8) over the time points 0 to 60 min, for each algorithm and value of D, with both random (R) and GA filters. In general, we can observe similar convergence patterns. However, the final values reached using GA filters are always better than those achieved using R filters. Besides, in most cases, the convergence produced by GA filters is faster than that produced by R filters without causing premature convergence. These results confirm a slight superiority of GA filters over random ones.

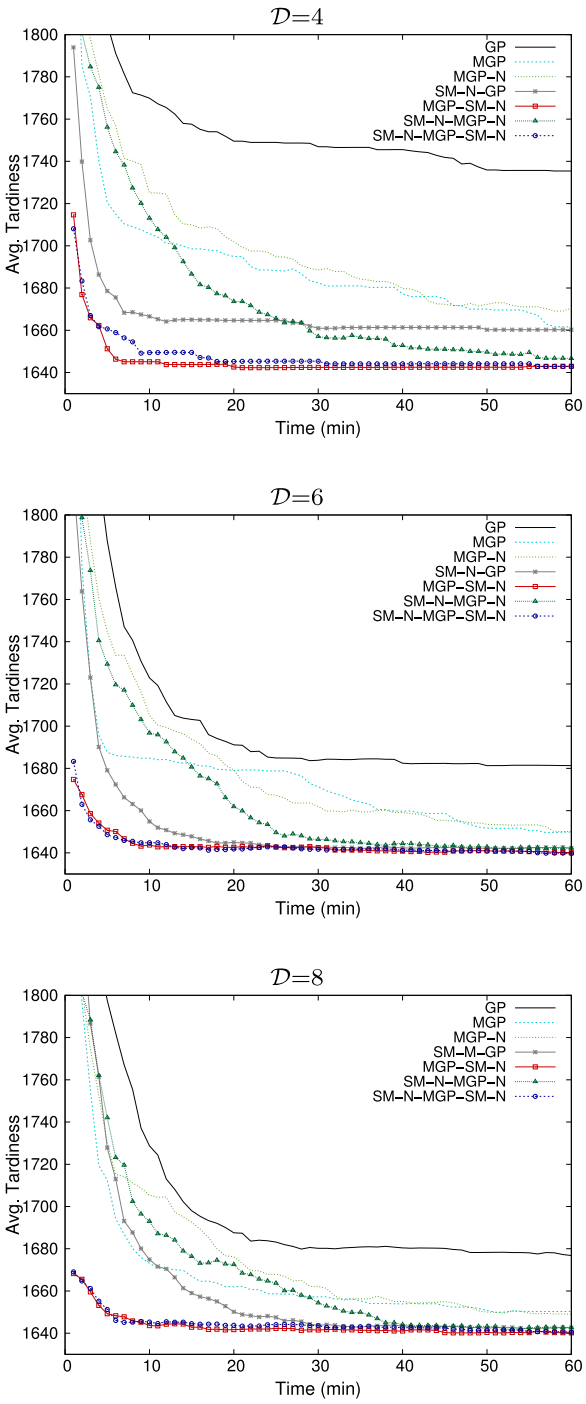


Fig. 6. Convergence patterns of all versions of GP, with GA filters, averaged for each value of D.

7.2. Evolution of GP variants

Fig. 6 shows the convergence patterns of all versions of GP (restricted to those with GA filters when they are used in any way) averaged for each value of D (4, 6 and 8). We can observe that GP alone is the worst method in all three cases. The methods that exploit local search without neighbouring estimation outperform GP. And finally, the methods exploiting surrogate evaluation and neighbouring estimation are those that perform the best. All of them reach rather similar average results at the end but show significant differences over the first half of the time limit.

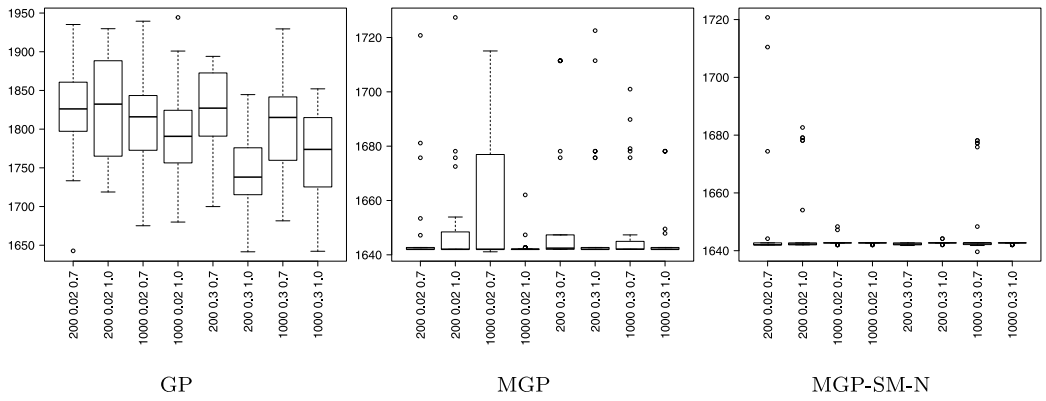


Fig. 7. Results obtained by GP, MGP and MGP-SM-N on the test set using different values of population size (200 or 1000), crossover ratio (1.0 and 0.7) and mutation ratio (0.02 and 0.3).

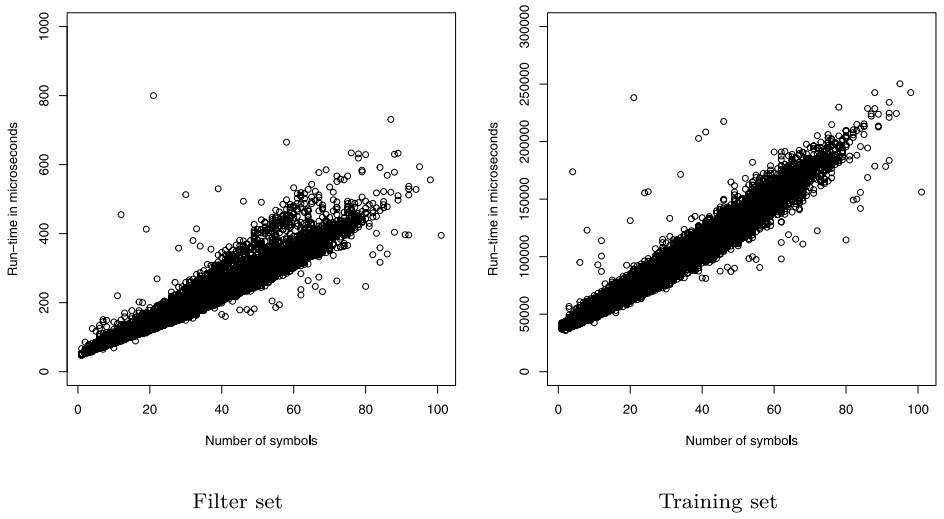


Fig. 8. Comparison of time in microseconds required to solve the filter and training sets using 10 000 priority rules with varying numbers of symbols.

7.3. Parameter tuning analysis

Fig. 7 showcases the results obtained from GP, MGP, and MGP-SM-N on the test set. The experiments were conducted with a fixed D=4, and variations were made in the population size, mutation ratio, and crossover ratio. The results reveal that while there are variations in performance based on different parameter values, the observed differences do not attain statistical significance. Additionally, MGP-SM-N emerges as the standout approach, consistently demonstrating improved performance and generating solutions with reduced dispersion.

7.4. Run-time analysis

The run-time (in microseconds) required by 10 000 rules with different sizes (number of symbols) when solving the filter and training sets is shown in Fig. 8. Notably, the computation time needed to solve the filter set is approximately 25 times shorter than that required for full evaluation.

7.5. Discussion

The above results highlight the effectiveness of the local search in MGP for achieving superior performance compared to GP, which only uses the crossover and mutation operators for search space exploration. Moreover, the introduction of surrogate evaluation, aimed at eliminating the worst neighbouring solutions, enables MGP-SM-N to converge faster than MGP while maintaining the quality of the solutions obtained. When considering the other two variants that incorporate surrogate evaluation, namely SM-N-MGP-N and SM-N-MGP-SM-N, it becomes evident that combining all the aforementioned options does not yield any advantages. Furthermore, the effectiveness of surrogate evaluation is more pronounced in the local search operator than the crossover operator, as demonstrated by the superior results obtained by MGP-SM-N compared to SM-N-GP and SM-N-MGP-N. In relation to the filter employed, it was observed that the use of filters evolved through GA results in faster convergence of MGP-SM-N in comparison to random filters. However, there are no statistically significant differences in the final outcomes.

To sum up, the integration of filters into GP can significantly enhance its efficiency. This approach involves discarding the least promising individuals during surrogate evaluation and prioritising the evaluation of the most potential candidates. By utilising a surrogate model, the time required to estimate fitness is drastically reduced, approximately 25 times faster than real fitness estimation. As a result, a considerable amount of computational time is allocated to evaluating the most promising solutions, leading to improved convergence of the algorithm while maintaining solution quality.

8. Conclusions and future work

We have seen that the proposed surrogate model (SM) based on small subsets of simple instances, which are called filters herein, may be effective to evolve scheduling priority rules via Genetic Programming

(GP). The key point is to come up with a filter so that the performance of the candidate rules on the filter resembles their performance on the actual training set, which commonly contains a (much) larger number of (much) larger instances of the problem.

The problem of calculating the best filter was formulated as a variant of the optimal subset problem and solved by a Genetic Algorithm (GA), which requires the availability of a pool of candidate instances for the filters and a pool of rules previously evaluated on these instances and on the test set. However, the method could be applied in settings where the pool of rules is not previously available. For example, the filters could be evolved by GA in coevolution with GP so that the pool of rules in each generation of GA may be taken from the rules previously evaluated by GP on the training set and the candidate instances for the filters. Besides, these instances may be obtained as simplifications of those in the training set.

Another key point is how to exploit the filters as a surrogate model in GP. In this regard, the best of the strategies we studied consists in using the filter to select the best among a series of offspring and then evaluate only the selected ones on the training set, and select the most promising neighbours generated with a local search algorithm, which was integrated into GP. The resulting algorithm clearly outperforms GP in evolving priority rules for the one machine scheduling problem with variable capacity, denoted $(1, Cap(t) \| \sum T_i)$.

This work leaves open some interesting lines of research, in particular, how to obtain simplified instances of the $(1, Cap(t) \| \sum T_i)$ problem bearing resemblance with the training set or, more generally, how to apply the proposed method to other scheduling problems (Zhou et al., 2022, 2021).

CRediT authorship contribution statement

Francisco J. Gil-Gala: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Visualization. **María R. Sierra:** Visualization, Validation, Data curation, Software. **Carlos Mencía:** Conceptualization, Writing – review & editing. **Ramiro Varela:** Conceptualization, Investigation, Analysis, Methodology, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This research has been supported by the Spanish State Agency for Research (AEI) under research project PID2019-106263RB-I00.

References

Agresti, A. (2010). *Statistical methods and applications, Analysis of ordinal categorical data*. New York: John Wiley & Sons.

Bhattacharya, M. (2008). Reduced computation for evolutionary optimization in noisy environment. In *GECCO'08: Proceedings of the 10th annual conference on genetic and evolutionary computation 2008* (pp. 2117–2122). http://dx.doi.org/10.1145/1388969.1389033.

Branke, J., Hildebrandt, T., & Scholz-Reiter, B. (2015). Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary Computation*, *23*, 249–277. http://dx.doi.org/10.1162/EVCO-a-00131.

Branke, J., & Schmidt, C. (2005). Faster convergence by means of fitness estimation. *Soft Computing*, *9*, 13–20. http://dx.doi.org/10.1007/s00500-003-0329-4.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., & Woodward, J. R. (2019). A classification of hyper-heuristic approaches: Revisited. In *Handbook of metaheuristics* (pp. 453–477). Springer International Publishing Volume 272 of International Series in Operations Research & Management Science, http://dx.doi.org/10.1007/978-3-319-91086-4_14.

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. http://dx.doi.org/10.1162/EVCO_a_00044.

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. http://dx.doi.org/10.1016/j.ins.2017.12.013.

Colwell, D. J., & Gillett, J. R. (1982). Spearman versus kendall. *The Mathematical Gazette*, *66*, 307–309. http://dx.doi.org/10.2307/3615525.

Croux, C., & Dehon, C. (2010). Influence functions of the spearman and kendall correlation measures. *Statistical Methods & Applications*, *19*, 497–515.

Dimopoulos, C., & Zalzala, A. (2001). Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, *32*, 489–498. http://dx.doi.org/10.1016/S0965-9978(00)00109-5.

Duflo, G., Kieffer, E., Brust, M. R., Danoy, G., & Bouvry, P. (2019). A gp hyper-heuristic approach for generating tsp heuristics. In *IPDPSW'19: IEEE international parallel and distributed processing symposium workshops* (pp. 521–529). http://dx.doi.org/10.1109/IPDPSW.2019.00094.

Dumić, M., Šišejkovic, 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. http://dx.doi.org/10.1016/j.future.2018.04.029.

Durasević, M., & Jakobović, D. (2018). Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. *Genetic Programming and Evolvable Machines*, *19*, 9–51. http://dx.doi.org/10.1007/s10710-017-9310-3.

Durasević, M., & Jakobović, D. (2020). Automatic design of dispatching rules for static scheduling conditions. *Neural Computing and Applications*, http://dx.doi.org/10.1007/s00521-020-05292-w.

Durasević, M., Jakobović, D., & Knežević, K. (2016). Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, *48*, 419–430. http://dx.doi.org/10.1016/j.asoc.2016.07.025.

Gil-Gala, F. J., Durasević, M., & Jakobović, D. (2022a). Genetic programming for electric vehicle routing problem with soft time windows. In *Proceedings of the genetic and evolutionary computation conference companion GECCO '22* (pp. 542–545). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3520304.3528994.

Gil-Gala, F. J., Durasević, M., Sierra, M. R., & Varela, R. (2022b). Building heuristics and ensembles for the travel salesman problem. In J. M. Ferrández Vicente, J. R. Álvarez-Sánchez, F. de la Paz López, & H. Adeli (Eds.), *Bio-inspired systems and applications: From robotics to ambient intelligence* (pp. 130–139). Cham: Springer International Publishing..

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

Gil-Gala, F. J., Mencía, C., Sierra, M. R., & Varela, R. (2020a). Exhaustive search of priority rules for on-line scheduling. In *Proceedings of the 2020 conference on ECAI 2020: 24th European conference on artificial intelligence*. http://dx.doi.org/10.3233/FAIA200365.

Gil-Gala, F. J., Mencía, C., Sierra, M. R., & Varela, R. (2021a). Learning ensembles of priority rules for on-line scheduling by hybrid evolutionary algorithm. *Integrated Computer-Aided Engineering*, *28*, 65–80. http://dx.doi.org/10.3233/ICA-200634.

Gil-Gala, F. J., Sierra, M. R., Mencía, C., & Varela, R. (2020b). Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling. *Natural Computing*, http://dx.doi.org/10.1007/s11047-020-09793-4.

Gil-Gala, F. J., Sierra, C., Mencía, María R., & Varela, R. (2021b). The optimal filtering set problem with application to surrogate evaluation in genetic programming. In *GECCO'21: Proceedings of the 2021 on genetic and evolutionary computation conference*.

Gil-Gala, F. J., Sierra, M. R., Mencía, C., & Varela, R. (2021c). Genetic programming with local search to evolve priority rules for scheduling jobs on a machine with time-varying capacity. *Swarm and Evolutionary Computation*.

Guo, W., Vanhoucke, M., Coelho, J., & Luo, J. (2021). Automatic detection of the best performing priority rule for the resource-constrained project scheduling problem. *Expert Systems with Applications*, *167*, http://dx.doi.org/10.1016/j.eswa.2020.114116.

Hamo, Y., & Markovitch, S. (2005). The compset algorithm for subset selection. In *IJCAI'05: Proceedings of the 19th international joint conference on artificial intelligence* (pp. 728–733).

He, C., Zhang, Y., Gong, D., & Ji, X. (2023). A review of surrogate-assisted evolutionary algorithms for expensive optimization problems. *Expert Systems with Applications*, *217*, http://dx.doi.org/10.1016/j.eswa.2022.119495.

Hernández-Arauzo, A., Puente, J., Varela, R., & Sedano, J. (2015). Electric vehicle charging under power and balance constraints as dynamic scheduling. *Computers & Industrial Engineering*, *85*, 306–315. http://dx.doi.org/10.1016/j.cie.2015.04.002.

Hildebrandt, T., & Branke, J. (2015). On using surrogates with genetic programming. *Evolutionary Computation*, *23*, 343–367. http://dx.doi.org/10.1162/EVCO_a_00133.

Hussein, R., & Deb, K. (2016). A generative kriging surrogate model for constrained and unconstrained multi-objective optimization. In *GECCO'16: Proceedings of the genetic and evolutionary computation conference 2016* (pp. 573–580). http://dx.doi.org/10.1145/2908812.2908866.

Jaklinović, K., Durasević, M., & Jakobović, D. (2021). Designing dispatching rules with genetic programming for the unrelated machines environment with constraints. *Expert Systems with Applications*, *172*, Article 114548. http://dx.doi.org/10.1016/j.eswa.2020.114548.

Jakobović, D., & Marasović, K. (2012). Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing*, *12*, 2781–2789. http://dx.doi.org/10.1016/j.asoc.2012.03.065.

Keijzer, M., & Babovic, V. (1999). Dimensionally aware genetic programming. In *GECCO'99: Proceedings of the 1st annual conference on genetic and evolutionary computation. Vol. 2* (pp. 1069–1076).

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, *30*, 81–93. http://dx.doi.org/10.1093/biomet/30.1-2.81.

Kendall, M. G. (1970). *Rank correlation methods*. London: Griffin.

Koulamas, C. (1994). The total tardiness problem: Review and extensions. *Operations Research*, *42*, 1025–1041. http://dx.doi.org/10.1287/opre.42.6.1025.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

Kromer, P., Platos, J., Nowakova, J., & Snasel, V. (2018). Optimal column subset selection for image classification by genetic algorithms. *Annals of Operations Research*, *265*, 205–222.

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*, http://dx.doi.org/10.1016/j.eswa.2019.112915.

Luo, Jingyu, Vanhoucke, Mario, Fernandes da Silva Coelho, José, & Guo, Weikang (2022). An efficient genetic programming approach to design priority rules for resource-constrained project scheduling problem. *Expert Systems with Applications*, *198*, 20.

MacLachlan, J., Mei, Y., Branke, J., & Zhang, M., Vehicle Collaboration (2020). Genetic programming hyper-heuristics with for uncertain capacitated arc routing problems. *Evolutionary Computation*, *28*, 563–593.

Mencía, C., Sierra, M. R., Mencía, R., & Varela, R. (2019). Evolutionary one-machine scheduling in the context of electric vehicles charging. *Integrated Computer-Aided Engineering*, *26*, 1–15. http://dx.doi.org/10.3233/ICA-180582.

Nguyen, S., Mei, Y., Xue, B., & Zhang, M. (2019). A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation*, *27*, 467–496. http://dx.doi.org/10.1162/evco_a_00230.

Nguyen, S., Zhang, M., & Tan, K. C. (2017). Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics*, *47*, 2951–2965. http://dx.doi.org/10.1109/TCYB.2016.2562674.

Poli, R., Langdon, W. B., & McPhee, N. F. (2008). A field guide to genetic programming.

Sels, V., Gheysen, N., & Vanhoucke, M. (2012). A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research*, *50*, 4255–4270. http://dx.doi.org/10.1080/00207543.2011.611539.

Spearman, C. (1904). The proof and measurement of association between two things. *American Journal of Psychology*, *15*, 72–101. http://dx.doi.org/10.2307/1412159.

Tay, J. C., & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, *54*, 453–473. http://dx.doi.org/10.1016/j.cie.2007.08.008.

Vela, C. R., Varela, R., & González, M. A. (2010). Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, *16*, 139–165. http://dx.doi.org/10.1007/s10732-008-9094-y.

Wang, S., Mei, Y., Zhang, M., & Yao, X. (2021). Genetic programming with niching for uncertain capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 1. http://dx.doi.org/10.1109/TEVC.2021.3095261.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*, 67–82. http://dx.doi.org/10.1109/4235.585893.

Zeiträg, Y., Figueira, J. R., Horta, N., & Neves, R. (2022). Surrogate-assisted automatic evolving of dispatching rules for multi-objective dynamic job shop scheduling using genetic programming. *Expert Systems with Applications*, 209. http://dx.doi.org/10.1016/j.eswa.2022.118194.

Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., & Zhang, M. (2022). Instance rotation based surrogate in genetic programming with brood recombination for dynamic job shop scheduling. *IEEE Transactions on Evolutionary Computation*, http://dx.doi.org/10.1109/TEVC.2022.3180693.

Zhou, S., Jin, M., Liu, C., Zheng, X., & Chen, H. (2022). Scheduling a single batch processing machine with non-identical two-dimensional job sizes. *Expert Systems with Applications*, *201*, http://dx.doi.org/10.1016/j.eswa.2022.116907.

Zhou, Z., Ong, Y. S., Nair, P. B., Keane, A. J., & Lum, K. Y. (2007). Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *37*, 66–76. http://dx.doi.org/10.1109/TSMCC.2005.855506.

Zhou, S., Xing, L., Zheng, X., Du, N., Wang, L., & Zhang, Q. (2021). A self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times. *IEEE Transactions on Cybernetics*, *51*, 1430–1442. http://dx.doi.org/10.1109/TCYB.2019.2939219.