Information Sciences 634 (2023) 340–358

Contents lists available at ScienceDirect

Information Sciences

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

Ensembles of priority rules to solve one machine scheduling problem in real-time

Francisco J. Gil-Gala [a,*], Marko Đurasević [b], Ramiro Varela [a], Domagoj Jakobović [b]

[a] Department of Computer Science, University of Oviedo, 33271, Gijón, Spain
[b] Faculty of Electrical Engineering and Computing, University of Zagreb, 10000, Zagreb, Croatia

ARTICLE INFO

Keywords:
Scheduling
Priority rules
Ensembles
Metaheuristics
Hyperheuristics

ABSTRACT

Priority rules are one of the most common and popular approaches to real-time scheduling. Over the last decades, several methods have been developed to generate rules automatically. In addition, it has been shown that combining rules into ensembles is better than using a single rule in many cases. In this paper, we analyze different ways to create and use ensembles previously developed through genetic programming. In our study, we classify ensembles as either collaborative or coordinated, depending on how the rules are used. In the first case, all the rules contribute to the creation of the same solution, while in the second case, each rule works independently on its own solution, and the best of them is selected as the solution of the ensemble. We found that each method has its own strengths and weaknesses, which leads us to use them in combination. Based on this hypothesis, we developed new methods to design and combine collaborative and coordinated ensembles and evaluated these methods for the One Machine Scheduling Problem with time-varying capacity and minimization of total tardiness. The results of the experimental study provided interesting insights into the use of ensembles and showed that our proposals outperform previous methods.

1. Introduction

Scheduling problems arise in many real-world environments, such as injection moulding [31], bus driver scheduling [27], and electric vehicle charging [43]. In this work, we focus on the One Machine Scheduling Problem (OMSP), also known as the Single Machine Problem [34], with variable capacity over time and the objective of minimizing the total tardiness, denoted as $(1, Cap(t)||\sum T_j)$ in the standard $\alpha|\beta|\gamma$ notation proposed in [22]. In this problem, the goal is to schedule a set of jobs without exceeding the capacity of the machine. Therefore, a priority must be computed for each job, and the machine processes them in the given order. The peculiarity of the $(1, Cap(t)||\sum T_j)$ problem is that some of the jobs can be processed in parallel since the machine has a capacity that varies over time and is usually equal to or greater than 2.

The one machine scheduling problem has applications in many real-world environments, such as in the steel making industry [50], communication systems [48], and others. In this study, we consider the problem of scheduling the charging times of a large fleet of electric vehicles, which was introduced and modelled in [24] as a $(1, Cap(t)||\sum T_j)$ problem. Hernandez-Arauzo et al. [24] proposed to solve the $(1, Cap(t)||\sum T_j)$ problem using the Apparent Tardiness Cost (ATC) rule, which was used as a guideline for

[*] Corresponding author.
E-mail addresses: giljavier@uniovi.es (F.J. Gil-Gala), Marko.Durasevic@fer.hr (M. Đurasević), ramiro@uniovi.es (R. Varela), Domagoj.Jakobovic@fer.hr (D. Jakobović).

https://doi.org/10.1016/j.ins.2023.03.114
Received 5 January 2022; Received in revised form 7 March 2023; Accepted 19 March 2023

Available online 22 March 2023
0020-0255/© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

creating a schedule builder. This framework is commonly referred to as *on-line scheduling* because it can be used to create the schedule in parallel with its execution, where decisions must be made quickly. The reason why ATC and similar rules can be used for on-line scheduling is their low time complexity compared to classical metaheuristics such as genetic algorithms [30].

The performance of existing priority rules for various scheduling problems is still very limited, making it challenging to use an appropriate priority rule for the problem. Therefore, many studies focus on the automatic design of priority rules using various hyper-heuristic methods, among which Genetic Programming (GP) emerges as the most commonly used [5].

Competitive results have been obtained using GP in various fields such as hardware design, bioinformatics, symbolic regression, and scheduling [29]. GP is usually interpreted as a *hyper-heuristic* since it can be applied to obtain heuristics or rules that can solve any number of problems. According to the taxonomy proposed by Burke et al. [6], this paradigm is often called hyper-heuristics based on *heuristic generation*. In recent years, many works have used GP as a hyper-heuristic to develop rules for various NP-hard problems, such as job shop scheduling [23], resource constrained project scheduling [7,10], scheduling unrelated parallel machines [13] or one machine scheduling [17], among others.

In general, GP is used as a learning algorithm in which the fitness of individuals (rules) is computed from the results obtained by the rules in solving a set of instances of a given problem, usually referred to as the *training set*. As in machine learning, the solution (the best rule developed by GP) is evaluated on an unseen set of instances called the *test set*. Although the rules developed by GP generally outperform the ATC rule and other manually created rules, the results are still far from those obtained with evolutionary algorithms, leaving much room for improvement [30]. Since GP is a stochastic method, it is necessary to run it several times to develop good rules. Usually, only the best developed rule is used, meaning that most of the developed rules are discarded in the end. For these reasons, several works have investigated the simultaneous application of a set of priority rules to generate schedules, i.e., ensembles [11,18,32].

Recently, ensemble strategies used in combination with various metaheuristic methods have received greater attention [45]. One way ensembles can be used with metaheuristics is to allow the simultaneous application of different strategies or operators in the algorithm. For example, they can be used to combine multiple mutation operators in differential evolution [44], use multiple population sorting strategies in multi-objective algorithms [35], or incorporate multiple constraint solving techniques into the algorithm [46]. However, in the context of hyper-heuristics, ensemble learning is used in such a way that multiple generated heuristics are used in synergy to improve their results and obtain more stable behaviour [11,32]. In on-line scheduling problems, ensembles of priority rules are constructed and used to solve a scheduling problem. We define an ensemble as a set of rules that can be classified as *coordinated* and/or *collaborative* depending on how the rules construct the solution. On the one hand, collaborative approaches use the rules together to construct a single solution [11,32]. The rules work together using a particular combination method that combines the decisions of all the rules in the ensemble into a single decision. The coordinated approach, on the other hand, is based on each rule creating its own solution independently and then selecting the best solution among them [18].

In this paper, we are interested in constructing ensembles specifically designed to solve the $(1, Cap(t)||\sum T_j)$ problem. In previous studies, only coordinated ensembles capable of outperforming single priority rules have been considered for this problem. On the other hand, collaborative ensembles have shown good performance on other problems, such as the unrelated parallel machines [11]. Nevertheless, as far as we know, they have yet to be systematically compared on the same problem. Therefore, one of the goals of this paper is to make a fair comparison between the two classes of ensembles in solving the $(1, Cap(t)||\sum T_j)$ problem. To this end, we consider the method proposed in [18] to obtain coordinated ensembles from a pool of priority rules evaluated on a set of training instances. We will develop algorithms to evolve collaborative ensembles from the same set of rules and training sets. Furthermore, we will highlight the weaknesses and strengths of each type of ensemble and explore the possibility of combining the two to achieve a positive synergistic effect. In addition, we study the influence of the cardinality and the composition of the problem set used by the algorithms to learn rules (GP [17]) and ensembles (mainly hybrid evolutionary algorithms [18]). Experimental results show that collaborative ensembles perform better than a single rule, while coordinated ensembles perform much better than collaborative ensembles when solving a large set of unseen instances. In summary, the combination of collaborative and coordinated ensembles leads to the best overall results, which motivates us to continue research in this direction.

The main contributions of the paper may be summarized as follows

- Develop collaborative ensembles for the $(1, Cap(t)||\sum T_j)$ problem and compare collaborative and coordinated ensembles for this problem.
- Analyze the strengths and weaknesses of each class of ensembles and explore different ways to use both in combination to achieve a synergistic effect.
- Compare the best combination methods with state-of-the-art proposals for the $(1, Cap(t)||\sum T_j)$ problem.

The rest of the paper is organized as follows. In the next section, we analyze the literature on hyper-heuristics and scheduling under on-line conditions. Next, we introduce the $(1, Cap(t)||\sum T_j)$ problem and provide an overview the proposed methods based on hyper-heuristics. Then, in section 4, we describe the proposed approach for creating collaborative and coordinated ensembles. In section 5, we report the results of the experimental study. Finally, in sections 6 and 7, we summarize the main conclusions and outline some ideas for future work.

2. Preliminaries and literature review

As mentioned above, genetic programming proposed by John R. Koza [28] has proven to be one of the most successful hyper-heuristic methods for automatically generating priority rules. As for ensembles of priority rules, they have been created using different methods and from different points of view. Park et al. [33] used DeJong's cooperative coevolution framework for the job shop scheduling problem. The evolved ensembles are used in a conventional cooperative manner with a voting method to schedule the next operation. Hart and Sim [23] proposed an artificial immune network to evolve ensembles consisting of sequences of expression trees used sequentially to schedule each operation. These ensembles outperformed those developed in [33].

In their works about the Unrelated Parallel Machine Problem, Đurasević and Jakobović [11,12] identified two key issues in ensemble construction: how rules are combined and how they are chosen to compose the ensemble. They proposed four learning approaches to construct ensembles of priority rules for the unrelated machines environment: simple ensemble combination, BagGP, BoostGP, and cooperative coevolution. The simple ensemble combination (SEC) method [11] was initially based on a simple random search that generates up to 20 000 ensembles of random combinations of priority rules. BagGP evolves each rule on a different training set, while BoostGP uses the AdaBoost algorithm proposed in [15] for rule selection. Cooperative coevolution is an evolutionary algorithm that splits the problem into several subproblems, each of which is then solved using a subpopulation. The simple ensemble combination performed better than BagGP, BoostGP, and cooperative coevolution, which was also observed when constructing ensembles for the resource constrained project scheduling problem in a later study [39]. For this reason, the authors focused on the SEC method and proposed five greedy methods to construct the ensembles [12]:

- *Random selection* selects rules uniformly.
- *Probabilistic selection* selects rules based on their fitness.
- *Grow* builds the ensemble incrementally, adding the rule that provides the greatest improvement in the quality of the ensemble.
- *Grow-destroy* uses the grow method to build a large ensemble and then removes the rules from the ensemble whose removal results in the best improvement in the quality of the ensemble.
- *Instance-based* is similar to the grow method, but the quality of the rules is interpreted as the number of problem instances for which the ensemble performs best.

These methods were analyzed using ensembles of sizes 3, 5, and 7. In a comprehensive experimental study, the authors concluded that ensembles comprising about five rules produced better results than a single priority rule. The best methods were instance-based, grow, and grow-destroy, although ensembles constructed by random selection were sometimes better on average.

As for combination methods, Đurasević and Jakobović [11,12] have studied the two classical methods for combining rules into ensembles to make a decision, the sum and vote combination methods, which are common in the machine learning context [26]. Park et al. [33], and Hart and Sim [23] used majority voting. In addition, Park et al. [32] studied four popular combination methods: majority voting, linear combination, weighted majority voting, and weighted linear combination.

Gil-Gala et al. [18] proposed an alternative approach to apply ensembles, where the rules are used in parallel to obtain a set of solutions to the problem instance. In this way, they obtain as many solutions as there are rules, while the previous approaches obtain only a single solution. Out of the obtained set of solutions, the best one is selected as the final result. They formulate the problem of computing ensembles of priority rules as the Optimal Ensemble of Priority Rules Problem (OEPRP) and propose three algorithms to solve it, namely an Iterative Greedy Algorithm (IGA) inspired by a similar algorithm for the Maximum Coverage Problem (MCP), a Genetic Algorithm (GA), and a Local Search Algorithm (LSA). These types of ensembles are designed for static scheduling problems where all information about the problem is known in advance. However, a new method of ensemble collaboration for dynamic scheduling problems, inspired by the ensemble approach described earlier, was proposed in [14] and showed good performance in dynamic environments.

In our study, we classify ensembles as either *collaborative* or *coordinated*, depending on which of the above methods is used to generate a solution. In the first case, all rules contribute to creating the same solution. In contrast, in the coordinated ensembles, each rule searches for a solution, and the best of these solutions is considered the solution generated by the ensemble.

Table 1 provides an overview of the literature on using ensembles in hyper-heuristics. All studies were categorized into several groups, including the problem considered, the method used to construct the ensembles, how the ensembles are aggregated to arrive at a unified decision, and finally, whether the rules used to construct the ensembles were evolved simultaneously with them (evolved) or whether previously developed rules were used (pre-evolved). Most of the studies focused on solving various scheduling problems. However, some also demonstrated the successful application of ensembles to other problems, such as the capacitated arc routing problem [42,41] and the travelling salesman problem [20]. As for the methods used to construct the ensembles, we find a plethora of different methods in the literature, ranging from evolutionary algorithms [33,20] to greedy methods [12] to methods inspired by machine learning algorithms [11,42]. Based on the overview, we see that no single method is dominantly used in the literature. Therefore, it is impossible to point to a single method as the most appropriate for ensemble construction, mainly since different performances of a single method have been observed in different studies. Moreover, the choice of ensemble construction method is tied to whether to use existing rules or develop them simultaneously with ensemble construction. In the former case, simple greedy methods such as SEC or more sophisticated optimization methods such as GAs and LSA can be used. In the latter case, many methods have been used, ranging from standard cooperative coevolution to methods inspired by the machine learning field (BagGP and BoostGP) to entirely new methods (NELLI-GP).

Table 1
Literature overview of ensemble application for hyper-heuristic methods.

Reference	Problem	Ensemble construction	DRs in ensembles	Ensemble aggregation	Esnemble type
[33]	job shop	cooperative coevolution	evolved	vote	collaborative
[23]	job shop	NELLI-GP	evolved	vote	collaborative
[11]	unrelated machines	cooperative coevolution, BagGP, BoostGP, SEC	evolved, preevolved	vote, sum	collaborative
[32]	job shop	cooperative coevolution	evolved	vote, sum, weighted vote, weighted sum	collaborative
[41]	arc routing	cooperative coevolution, BoostGP, BagGP	preevolved	sum	collaborative
[42]	arc routing	BagGP, NicheGP	preevolved	sum	collaborative
[12]	unrelated machines	SEC	preevolved	vote, sum	collaborative
[21]	one machine	GA	preevolved	individual execution	coordinated
[19]	one machine	GA	preevolved	individual execution	coordinated
[18]	one machine	GA, LSA, IGA, MA	preevolved	individual execution	coordinated
[39]	resource constrained scheduling	cooperative coevolution, BagGP, BoostGP, SEC	evolved, preevolved	vote, sum	collaborative
[20]	travelling salesman	GA	preevolved	vote, sum	collaborative
[14]	unrelated machines	SEC	preevolved	simulation based	collaborative

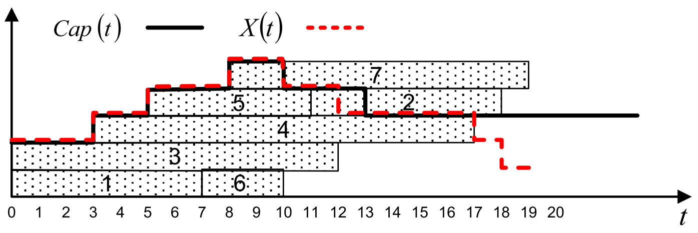


Fig. 1. A feasible schedule to a $(1, Cap(t)||\sum T_j)$ instance with 7 jobs. $Cap(t)$ denotes the capacity of the machine over time and $X(t)$ is the capacity consumed by the jobs.

The literature covers the evolution of rules and the application of pre-evolved rules in ensembles with equal share. The way rules are aggregated usually depends on the type of ensemble. In the case of collaborative ensembles, the sum or vote methods are usually used to aggregate the decisions in the ensemble. On the other hand, in the case of coordinated ensembles, the rules are executed individually, and the best schedule is then selected. Most of the research papers dealt with collaborative ensembles. However, this is because coordinated ensembles have only recently been proposed and have received less attention. As we can see from the literature review, these two types of ensembles have not yet been applied to the same problem or compared with each other. Therefore, in this study, we attempt to fill this gap in the literature by applying both ensembles to a common problem and comparing their performance.

3. The $(1, Cap(t)||\sum T_j)$ problem

In the $(1, Cap(t)||\sum T_j)$ problem, we are given a number of n jobs $\{1, \ldots, n\}$, all of which are available at time $t = 0$ and must be scheduled on a single machine. The feature of this problem is that the capacity of the machine varies over time with $Cap(t) \geq 0, t \geq 0$. The goal is to assign such starting times $st_j, 1 \leq j \leq n$ to the jobs that (1) at any time $t \geq 0$, the number of jobs 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 p_j is the duration of job j, and C_j is its completion time. The objective is to minimize the total tardiness defined as $\sum_{j=1,\ldots,n} \max(0, C_j - d_j)$, where d_j is the due date of job j (see Fig. 1).

The $(1, Cap(t)||\sum T_j)$ problem comes from the Electric Vehicle Charging Scheduling Problem (EVCSP) described in [24]. The EVCSP is a dynamic problem motivated by the charging station designed in [36], where a number of electric vehicles arrive at the parking lots overtime at times not known in advance, and their charging times must be scheduled considering some constraints, namely limited power, no interruption, and balanced load on the three-phase feeder. In [24], the EVCSP is modelled as a sequence of static problems, which in turn are decomposed into three instances each of the $(1, Cap(t)||\sum T_j)$ problem.

Since the EVCSP must be solved online, each of the $(1, Cap(t)||\sum T_j)$ instances must be solved in real-time. In [24], the authors proposed an effective solution using a stochastic schedule builder guided by the ATC rule. Other approaches to solve the EVCSP using metaheuristics have also been proposed in the literature, such as genetic algorithm [16], but none meet the requirements to solve the problem online.

Table 2
Functional and terminal sets used to build expression trees. Symbol "-" is considered in unary and binary versions. max_0 and min_0 return the maximum and the minimum of an expression and 0, respectively.

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

In [30], the authors propose a memetic algorithm to solve the $(1, Cap(t)||\sum T_j)$ problem. This approach provided the best-known solutions for the benchmark set considered in our experimental study (see Section 5). Unfortunately, it does not satisfy the real-time requirements arising from the online nature of EVCSP. However, the schedule builder proposed in [30] can be easily adapted for chromosome decoding to obtain schedules in real-time. This schedule builder is shown in Algorithm 1; it builds a schedule iteratively, so that at each step it selects the job that can be scheduled at the earliest time from the partial schedule built so far. This schedule builder has some interesting properties. For example, that it can generate any schedule in the space of left-shifted schedules that is dominant, i.e., contains at least one optimal schedule. It is clear that the selection of the next job can be done using a priority rule: the rule computes a priority for each candidate job j in US^*, and the job with the highest priority is selected.

One candidate to be used to calculate the job priorities is the classical ATC rule, which can be easily adapted to many scheduling problems with due date objectives. For the $(1, Cap(t)||\sum T_j)$, problem, this rule estimates the priority of job j as follows:

$$\frac{1}{p_j} exp\left[\frac{-max(0, d_j - \gamma(\alpha) - p_j)}{g\bar{p}}\right] \tag{1}$$

where p_j and d_j are the duration and due date of job j, respectively, $\gamma(\alpha)$ denotes the earliest start time for a job in US^*, $\bar{p}$ is the average processing time of jobs in US, and g is a user-introduced look-ahead parameter. Of course, a rule specific to the $(1, Cap(t)||\sum T_j)$ problem, or an ensemble of rules might work better.

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 staring 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 $u \in US^*$;
    // 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 $S = (st_1, st_2, ..., st_n)$;
```

4. Building rules and ensembles

In this section, we give an overview of the methods that have been proposed to evolve individual rules, and coordinated ensembles for the $(1, Cap(t)||\sum T_j)$ problem starting from a set of rules. We also describe how the algorithms proposed for developing coordinated ensembles can be adapted to construct collaborative ensembles starting from the same set of rules. Finally, we consider some ideas for combining both types of ensembles to exploit their strengths and avoid their weaknesses.

4.1. Designing priority rules with GP

Given the success of Genetic Programming (GP) [28] in developing priority rules for some problems such as job shop scheduling [4], we proposed in [17] a GP to develop rules for the $(1, Cap(t)||\sum T_j)$ problem. In this approach, the rules are formed from the functions and terminal symbols in Table 2. The grammar used restricts the expression trees to dimensionally correct expressions and avoids generating some equivalent expressions; in this way, the search space is drastically reduced compared to the whole space of arithmetically correct expressions. Moreover, a maximum depth D of the expression tree with typical values between 4 and 8 is considered to ensure both the readability of the rules and the affordances of the search space. The rules developed by GP performed better than classical rules, such as the ATC rule. For more details of the GP approach, we refer the reader to [17].

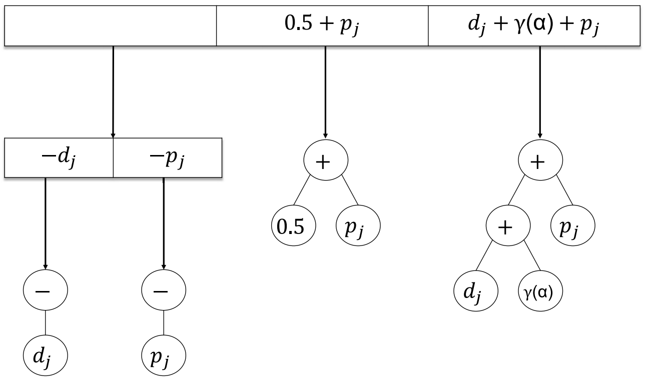
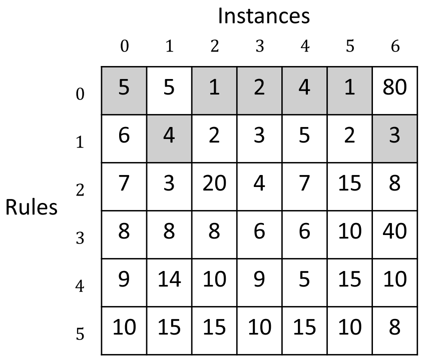


Fig. 2. An example of an ensemble composed of two rules and one ensemble, which is in turn composed of two individual rules. Each rule is represented by an arithmetical expression.



Rules \ Instances	0	1	2	3	4	5	6
0	5	5	1	2	4	1	80
1	6	4	2	3	5	2	3
2	7	3	20	4	7	15	8
3	8	8	8	6	6	10	40
4	9	14	10	9	5	15	10
5	10	15	15	10	15	10	8

Fig. 3. An example of a matrix of the tardiness values for a problem with six candidate rules and a training set with seven instances.

4.2. Building ensembles from priority rules

In this section, we consider the construction of ensembles from an existing set of priority rules $\mathcal{R}$, each of which has already been evaluated on a set of instances $\mathcal{P}$ of the $(1, Cap(t)|| \sum T_j)$ problem, called the training set. The objective (tardiness) values are recorded in a matrix $\mathcal{T}_{|\mathcal{R}|\times|\mathcal{P}|}$ such that $\mathcal{T}_{ij}$ is the tardiness of the schedule generated by the rule $\mathcal{R}_i$ for the instance $\mathcal{P}_j$. In principle, an ensemble is just a set of priority rules of a certain size. Therefore, it can be represented by a vector of individual rules, whether they are used as collaborative or coordinated ensembles. Also, in this paper, we consider combining ensembles in such a way that one or more elements of an ensemble can represent another ensemble. An example of such a combination is shown in Fig. 2. In this case, we have a two-level ensemble with three elements in the first level, the last two of which are individual rules, while the first is an ensemble with two individual rules in the second level. In this way, we could design ensembles with many levels, but we will limit our study to ensembles with only two levels. In the following sections, we will justify the actual usefulness of combined multilevel ensembles and show how they can be used. We will also address the most important aspects of creating ensembles, namely how to evaluate a candidate ensemble and combine or modify ensembles. In doing so, we will use the same algorithms proposed in [18] for creating coordinate ensembles.

4.2.1. Evaluation of ensembles

As with individual rules, the evaluation of an ensemble consists of solving the instances of the training set $\mathcal{P}$. Then, the ensemble's performance can be determined as the inverse of the cumulative tardiness of all obtained $|\mathcal{P}|$ schedules. Let us first consider single-level coordinated or collaborative ensembles. A coordinated ensemble is evaluated based on the tardiness values generated by each rule for the instances of the training set. More precisely, for each instance of $\mathcal{P}$, the best tardiness from all rules in the ensemble is considered. In this way, the evaluation of a coordinated ensemble is not very time consuming, since the tardiness value that each rule $\mathcal{R}_i$ generates for the instance $\mathcal{P}_j$, $\mathcal{T}_{ij}$, is known in advance. This is one of the advantages of coordinated ensembles. Fig. 3 shows an example matrix $\mathcal{T}$ for a problem with six candidate rules and a training set with seven instances. The performance of the ensemble $\{r_0, r_1, r_4\}$ is given by the values in the grey cells. In this case, only the rules r_0 and r_1 contribute to the ensemble performance measure since they cover all training set instances, so the rule r_4 could be removed from the ensemble. Note that while rule r_0 has the worst average fitness in all instances, it performs best in most instances and thus significantly impacts the ensemble's performance.

On the other hand, evaluating a collaborative ensemble requires generating new solutions for all instances of $\mathcal{P}$ using Algorithm 1, this time obtaining the priorities of the candidate jobs in US^* from aggregating the priorities of all rules in the ensemble. In this way,

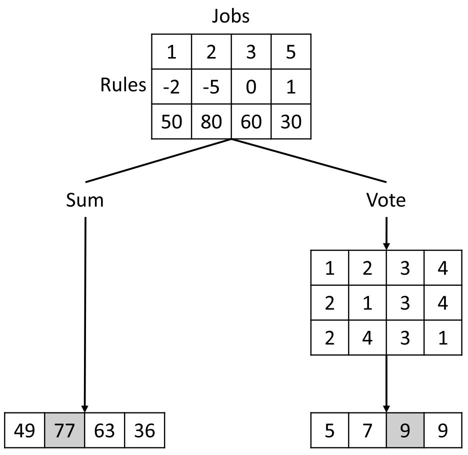


Fig. 4. An example of summation and voting methods for an ensemble with three rules and four jobs. The first one will choose the second job as it presents the largest value of the summed priorities, while the voting method would select the third one due to the largest sum of votes. In all cases the selected job is the one which has the highest value (priority or sum of votes).

the time required is more significant than when evaluating a coordinated ensemble, which is one of the drawbacks of collaborative ensembles. We consider the two most typical aggregation methods: summation and voting, as considered in [11]. In the first method, the priorities of each rule are summed, and the job with the largest summed value is selected. In the voting method, which is similar to Borda's voting method [49], the jobs are sorted from worst to best according to each rule, and each job receives a number of votes equal to its position in the sorted list. Then the votes received by each job are added together, and the job in US^* with the largest number of votes is selected. In both cases, ties can be resolved by other criteria, such as the ATC rule [32], the shortest processing time rule (SPT) [11], or even by chance. The voting method tends to lead to more draws, while the summation method can lead to biases resulting from different scales in the priorities calculated by the rules. Fig. 4 shows an example of both aggregation methods for a situation with four jobs and three rules.

Regarding the evaluation of multilevel ensembles, we consider only two-level ensembles, where the first level is interpreted as a coordinated ensemble and the second level as a collaborative ensemble. This combination is considered because it is the only reasonable combination of the two types of ensembles. First of all, the combination where the first level is interpreted as a collaborative ensemble and the second level is interpreted as a coordinated ensemble is not feasible. This is because coordinated ensembles produce an entire schedule as a result, while the collaborative ensembles require that the elements from which they are constructed produce a priority value at each decision point. For this reason, ensembles cannot be used in such a combination. On the other hand, it is possible to construct ensembles where both levels consist of the same ensemble type. However, such a two-level ensemble can be reduced to a single-level ensemble by combining all rules into one ensemble. This ensemble would make decisions in the same way as the original two-level ensemble. Since any multilevel ensemble of the same type can be reduced to a single level and it only makes sense to consider ensembles where coordinated ensembles consist of collaborative ensembles, this means that it is sufficient to consider ensembles with two levels, since any multilevel ensemble where the higher levels are represented as collaborative ensembles can be reduced to a two-level ensemble without changing its behaviour. Thus, the only combination of ensemble types that is useful to study is a two-stage ensemble in which the first stage is interpreted as a coordinated ensemble and the second stage as a collaborative ensemble.

Consistent with the above, the way to evaluate a multilevel ensemble follows from the way to evaluate the coordinated ensemble at the first level. The only difference is in the contribution of the collaborative ensembles at the second level. If we have only the tardiness generated by each rule on the training set, i.e., the matrix $\mathcal{T}_{|\mathcal{R}|\times|\mathcal{P}|}$, each collaborative ensemble must compute its own solution, as indicated above. However, we could start the process not only with the set of evaluated rules $\mathcal{R}$, but also with a set of pre-evaluated collaborative ensembles $\mathcal{E}$; in this case, the evaluation would be similar to that for the coordinated ensembles. We will consider this option in our experimental study.

4.2.2. Algorithms for building ensembles of priority rules

As mentioned before, when forming ensembles we start from a set $\mathcal{R}$ of priority rules developed by the GP proposed in [17]. From these rules, ensembles of a certain size P are formed using the algorithms proposed in [18], namely an Iterative Greedy Algorithm (IGA), a Genetic Algorithm (GA), a Local Search Algorithm (LSA), and a Memetic Algorithm (MA) combining GA and LSA. These algorithms were developed for the creation of coordinated ensembles, but can also be adapted for the creation of collaborative or multilevel ensembles by simply choosing the right evaluation operators and input data, i.e., the set of rules $\mathcal{R}$ for coordinated and collaborative ensembles, and additionally a set of collaborative ensembles $\mathcal{E}$ for multilevel ensembles. We will discuss these algorithms in the following paragraphs.

IGA starts with an empty ensemble and in each iteration adds a new rule to the partial ensemble created so far, provided that this rule improves the quality of the ensemble. More precisely, the rule that causes the largest improvement is selected. The process continues until P rules are selected or none of the remaining rules can improve the ensemble. IGA is particularly suitable for creating coordinated ensembles. In this case, the evaluation of an ensemble after adding a new rule is fast because only the instances in the training set $\mathcal{P}$ for which the rule is better than the ensemble before adding the rule need to be identified. Furthermore, if none of the remaining rules are able to improve the ensemble formed so far, then this ensemble is optimal in the sense that it is the best that can be generated from the set of rules $\mathcal{R}$, and its size is thus an upper bound on the lowest size of an optimal ensemble. Unfortunately, none of these properties apply to the formation of collaborative ensembles.

GA implements a generational evolutionary strategy with random selection, conventional recombination, and tournament replacement between every two parents and their two offspring. It uses an encoding scheme based on variations of rules in $\mathcal{R}$ taken P at a time. Thus, a chromosome represents an ensemble of size $K \leq P$ due to possible repetitions. Since the order of rules is not relevant, single point crossover and mutation operators are used and chromosomes are shuffled before mating.

LSA uses a neighbourhood structure defined such that a single move exchanges a rule in the current ensemble for another in $\mathcal{R}$. We consider both hill climbing (HC) and gradient descent (GD) as selection strategies. The stopping condition is satisfied if no improvement is obtained in the current iteration. As in the case of IGA, LSA is very suitable for coordinated ensembles but too time consuming for collaborative ensembles. For this reason, we used this method only for the former ensemble type.

LSA was used in combination with both IGA and GA. In the first case, we used it in two ways: to improve only the final solution of IGA and also to improve each partial ensemble. When combined with GA, LSA is applied to a set of chromosomes after they have been evaluated. An improved ensemble replaces the original one in the population; in this way we have a memetic algorithm (MA) with Lamarckian evolution.

The pseudocodes and descriptions of the above algorithms can be found in Appendix A.

4.3. Application for real-time problems

Although evolving new priority rules and building ensembles is computationally quite expensive, this is not a limitation when applied to real-time problems. The reason is that the evolution of new rules or ensembles is performed offline before they are applied to solve a concrete problem. After evolution, a set of priority rules or ensembles is obtained, which can be used to solve new scheduling problems as needed. Since priority rules and ensembles are fairly simple constructive heuristic methods, they can be used to build the schedule in real-time and in parallel with the execution of the schedule. Splitting this method in this way allows it to be used for real-time scheduling problems since the more expensive part of the method is executed at any time before a scheduling problem is solved, and the generated rules or ensembles are then used to construct the solution to a scheduling problem as needed. In contrast, standard search-based metaheuristic methods that directly solve a concrete problem would have to be executed while solving the problem in question, which prevents the use of such methods for solving real-time problems due to their complexity.

5. Experimental study

We conducted an experimental study to compare collaborative and coordinated ensembles and also to evaluate the performance of the proposed combined multilevel ensembles. For this purpose, we have extended the algorithms proposed in [21] to create all types of ensembles. The algorithms are implemented in Java and were run on a Linux cluster (Intel Xeon 2.26 GHz, 128 GB RAM).

We first describe the benchmark rule sets and $(1, Cap(t)||\sum T_j)$ problem instances used in the experiments and summarize some previous results from individual rules and coordinated ensembles. Then we analyze the results of collaborative ensembles and draw a comparison with coordinated ensembles. Finally, we show the results of combined multilevel ensembles and illustrate their advantage over collaborative and coordinated ensembles.

5.1. The benchmark set and previous results

In this study, we used the set of $(1, Cap(t)||\sum T_j)$ problem instances and the rules proposed in [18]. The first one includes 2000 instances, each with 60 jobs and a machine whose capacity varies between 2 and 10 over time, and was created to resemble the actual instances from the EVCSP [24]; 1000 instances are used for training and the remaining 1000 for testing. The problem instances are distributed between the two sets by first solving each instance with the ATC priority rule and sorting them by the total tardiness generated by the rule for each instance. Then, the instances are alternately placed in the training set or the test set in the order of their tardiness values. By constructing the sets in this way, we hope that both sets contain instances with similar characteristics and difficulty. The rule set consists of 1000 *general* rules developed by GP on the training instances. Specifically, the set of 1000 training instances was divided into 20 subsets of 50 instances each, and then 50 rules were developed on each subset. Another 1000 *specialized* rules were evolved for each of the 1000 problem instances in the training set. After removing equivalent rules (i.e., rules that compute the same schedules for each of the 1000 instances of the training set), we obtained a set of 1930 distinct rules. The testbed used can be downloaded from [1].

The results obtained in [18] with these rules, and the coordinated ensembles of 10 rules each on the training and test sets are summarized in Table 3. In the first two rows, we include the results of the ATC rule ($g = 0.3$ is the best value out of the ten values considered in these experiments: $0.1; 0.2; \ldots; 1.0$) and the best of all 1930 rules developed by GP. The next two rows show the values of the ensemble consisting of the 10 ATC rules and the ensemble consisting of the ten best rules developed by GP. The next five rows show the results of the coordinated ensembles obtained by IGA, GA, LSA with random restarts, IGA followed by a single run of LSA (IGA-LSA) and MA, respectively. In the case of IGA and IGA-LSA only one ensemble was computed, while for GA, LSA, and MA, 30 runs were performed, and the best and average values are shown in the table. We can see that in each case, the ensembles provide

Table 3
Summary of the previous results from coordinated ensembles and comparison to the results from single rules. The training and test sets are composed of 1000 instances each.

	Training		Test	
Method	Best	Avg.	Best	Avg.
Best ATC rule ($g = 0.3$)	1645.60		1644.26	
Best GP rule	1632.92		1637.29	
Ensemble 10 ATC rules	1576.07		1578.69	
Ensemble 10 best GP rules	1570.14		1573.92	
IGA	1551.50		1559.74	
GA	1550.82	1551.24	1557.61	1558.95
LSA	1550.89	1552.69	1558.15	1560.13
IGA-LSA	1551.38		1559.19	
MA	1550.82	1550.83	1557.61	1557.92
Best rule for each instance	1498.32		1505.06	

Table 4
Summary of the results (tardiness) obtained by specialized collaborative ensembles evolved using IGA and GA with the sum and vote combination methods on the 50 instances of the training set. Each instance was solved by the corresponding specialized ensemble, and the average from the 50 instances is shown for each algorithm. The last row represents the result achieved by the best rules for each instance. Additionally, the average run-time (seconds), the number of symbols and rules of each combination are also outlined.

Configuration			Results			
Algorithm	Method	Cardinality	Tardiness	Cardinality	Symbols	Time (s)
IGA	Sum	3	1478.64	2.04	47.62	15.70
		5	1477.54	2.22	51.92	22.84
		10	1471.38	2.54	60.82	62.66
	Vote	3	1473.60	2.16	51.64	35.04
		5	1471.70	2.46	58.80	59.60
		10	1477.54	2.22	51.92	22.28
GA	Sum	3	1467.76	3.00	83.88	109.82
		5	1460.78	5.00	136.58	175.42
		10	1470.94	10.00	277.88	161.26
	Vote	3	1486.56	3.00	79.44	49.48
		5	1476.04	5.00	131.08	80.38
		10	1455.18	10.00	270.92	334.34
Single rule			1489.46		23.10	

much better results than single rules and that the ensembles produced by the 5 proposed algorithms are better than the ensembles composed of 10 ATC rules or even the 10 best rules developed by GP. MA is the best method overall. To better compare the above results, the last row of the table shows the value of the coordinated ensemble composed of all 1930 rules, which is in fact, a lower bound on the value of a coordinated ensemble that can be formed from these rules.

5.2. Collaborative versus coordinate ensembles

In this section, we first address the creation and evaluation of collaborative ensembles and then compare collaborative and coordinated ensembles. As mentioned earlier, creating collaborative ensembles is much more time consuming than creating coordinated ensembles, especially for some of the algorithms described. For this reason, we only considered IGA and GA in this study.

We begin with a series of preliminary experiments in which collaborative ensembles are developed from a training set of 50 instances. Specifically, each ensemble is trained on a single instance from the training set, combining summation and voting methods, three cardinality values, 3, 5, and 10, and the IGA and GA algorithms. Only one run was performed for each combination, giving us a total of 600 ensembles. GA was parameterized with fairly standard values, namely 100 chromosomes, 100 generations, and crossover and mutation probabilities of 0.8 and 0.2, respectively, while IGA was run until it completed the formation of the ensemble. The results of these ensembles are shown in Table 4, where each value represents the average tardiness of 50 specialized ensembles, each of which solves the corresponding instance of the training set. The average runtime of each combination in seconds and the average size of the ensembles are also given. The purpose of this experiment is to analyze the performance limit of collaborative ensembles. The first observation we can draw from the table is that in all cases, the average tardiness produced by the ensembles is lower than the average tardiness produced by the best rule for each instance, which is shown in the last row of the table. This indeed represents a difference with the coordinated ensembles, for which this value represents a lower bound. Thus, it is clear that a collaborative ensemble can perform better in solving an instance than the best rule for that instance, which is an advantage of this

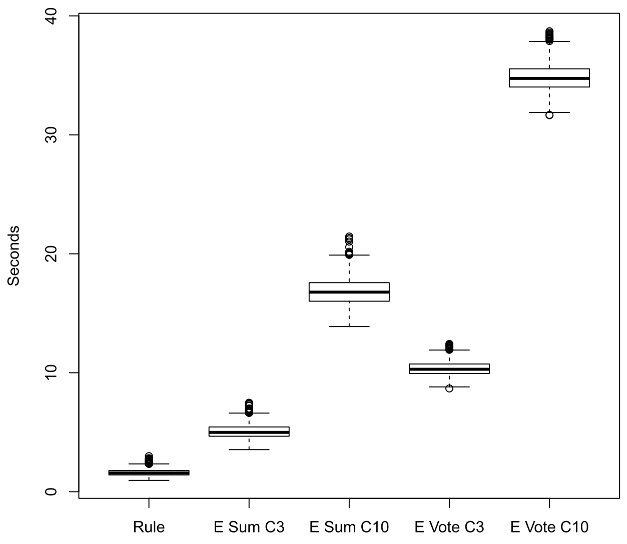


Fig. 5. The run-time (seconds) taken by rules and collaborative ensembles when solving the test set (1000 instances).

Table 5
Summary of the results from ensembles on training (50 instances) and test (1000 instances) sets. Best Rule is the best of 1930 rules on average on the instances in each set. Collaborative and Coordinated represent the mean values of the 20 ensembles of each class averaged for all instances in the sets.

Set	Best Rule	Collaborative	Coordinated
Training	1608.96	1592.45	1527.35
Test	1642.87	1631.56	1565.72

type of ensemble over coordinated ensembles. The reason why ensembles of PRs perform better than individual rules is that they are less prone to perform suboptimal scheduling decisions than individual PRs. This is because the ensemble makes its decision based on the collective decisions of all PRs in the ensemble. This means that even if a particular PR performs poorly in certain situations, this will not affect the ensemble's decision unless the majority of rules in the ensemble perform poorly. Therefore, ensembles tend to be more stable than individual PRs, which can easily make quite poor decisions in certain situations.

Moreover, we can see that GA is better than IGA, and voting is better than summation in both cases, especially when combined with GA. These results are reasonable since GA takes more time than IGA. Moreover, the quality of the ensembles improves in direct proportion to the cardinality of the ensemble. Overall, GA with voting is the best option. Ensembles with cardinality greater than ten would give better results, but in our study, we keep a limit of 10 rules as this is reasonable for the online requirements of EVCSP.

Regarding the time taken by ensembles to solve instances of the $(1, Cap(t)||\sum T_j)$ problem, there are significant differences depending on both the selection method and the cardinality of the ensemble. Fig. 5 shows the boxplots of the times required to solve the test set with the 1930 rules and with the 1930 collaborative ensembles consisting of 3 or 10 random rules. It can be seen that the voting method takes more time due to the normalization and aggregation of the priority values of the individual rules. Also, in all cases, the ensembles are more time consuming than individual rules. We must be aware that the ensembles evaluated in coordinated form require 3 or 10 times the time required for individual rules, depending on their cardinality.

Let us now consider the comparison between collaborative and coordinated ensembles. For this purpose, we created ensembles trained with 50 instances. We obtained 20 ensembles for each class, collaborative and coordinated. Fig. 6 shows the best and average convergence patterns from GA, averaged over the 20 runs. These experiments were conducted in the cluster mentioned above, where GA takes about 3 minutes for a single run, while it would take only 3 seconds if the ensembles were trained on only one problem instance.

Table 5 summarizes the results of both types of ensembles on the training set and the test set, along with the tardiness values obtained by the best of the rules in each set. We can see that in each case the ensembles perform better than the best rule, and that the coordinated ensembles perform better than the collaborative ensembles. These results, and the fact that coordinated ensembles are much easier to compute, show a clear advantage of coordinated over collaborative ensembles.

To gain better insight into collaborative ensembles, GA was run to generate one specialized ensemble for each of the 1000 instances of the entire training set and 20 general ensembles for each of the 20 subsets of 50 instances. Then, the entire test and training sets were solved by all 1400 ensembles (1000 specialized and 400 general). Using the results of these experiments, we analyzed the *dominance* of the rules and ensembles in terms of the number of times a rule or ensemble provides the best solution for an instance. The results are summarized in Table 6, where we can see that the specialized ensembles are absolutely dominant over

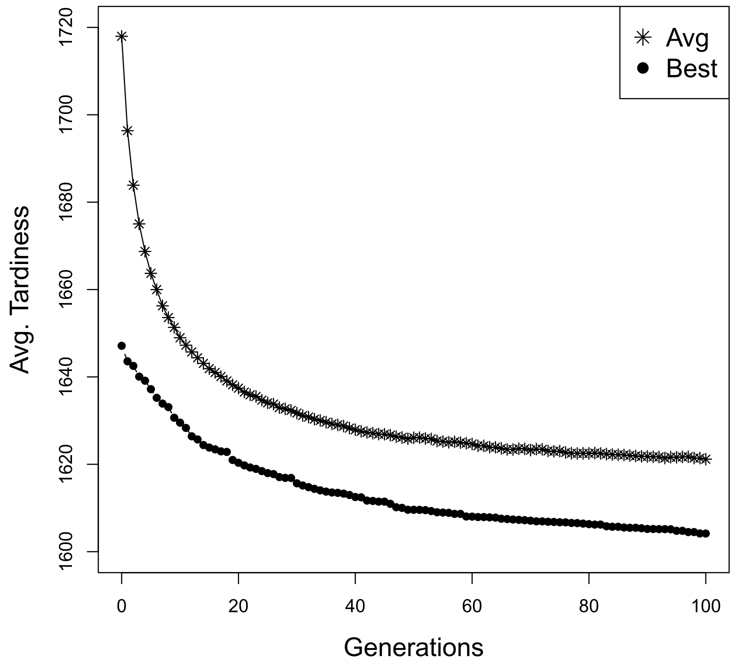


Fig. 6. Convergence pattern of GA over 100 generations using collaborative ensembles of 10 rules, the voting combination method, and trained with 50 problem instances. The values are averaged over 20 independent runs.

Table 6
Dominance of rules and ensembles on the training and test set, measured as the number of instances for which a type of rules or ensembles produce the best solution.

Method		Dominance	
		Training	Test
Rule	General	6	70
	Specialized	41	172
Ensemble	General	22	242
	Specialized	931	516

all methods on the training set, but they dominate only in about half of the instances of the test set, where again general ensembles dominate over rules and specialized rules dominate over general rules.

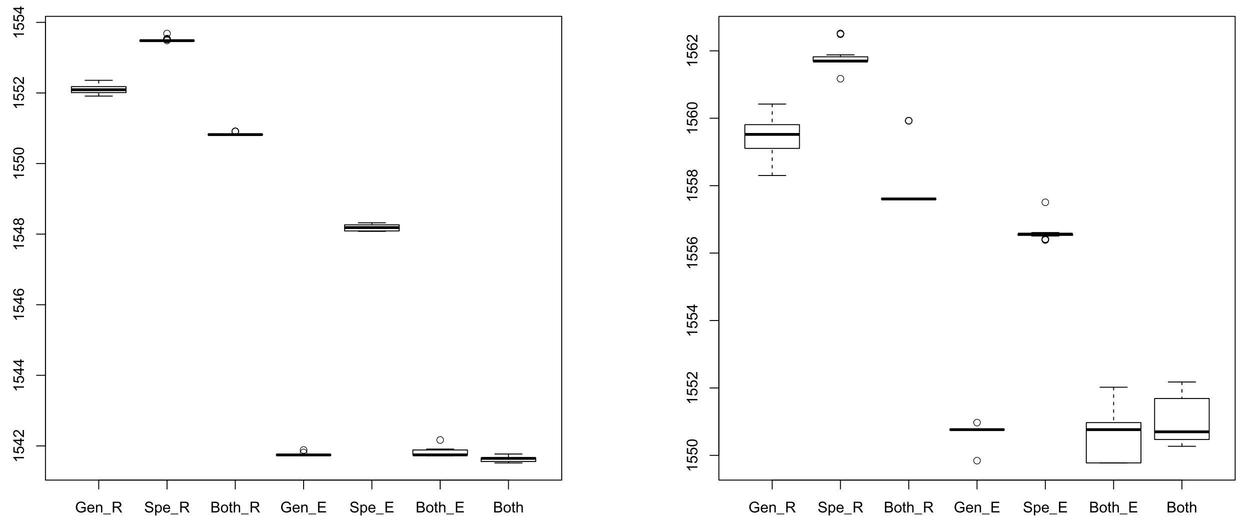
It follows that collaborative ensembles are still interesting because they are able to give results for certain instances that can be better than the results of the best rules for those instances, which, as mentioned, is a lower bound for coordinated ensembles. Therefore, collaborative ensembles can perform well on subsets of instances of the training and testing sets, so they can contribute to the formation of powerful combined multilevel ensembles, which would definitely justify the interest in this type of ensembles.

5.3. Evaluation of combined ensembles

To evaluate the performance of the combined ensembles, i.e. the coordinated ensembles consisting of rules and collaborative ensembles, we used the MA described in Section 4 with the following parameters: 100 individuals, 500 generations, crossover and mutation probabilities of 0.8 and 0.2, LSA was applied to 20% of individuals in each generation, limited to 100 neighbours and 5 iterations, as proposed in [18]. MA was run 30 times with the entire training set as described in Table 3. We also used the 1930 rules and the 1400 collaborative ensembles created in previous experiments as building blocks. To analyze the contribution of each type of rules and ensembles, we performed experiments starting from 7 different subsets: Rules (General, Specialized or Both types), Ensembles (General, Specialized or Both types), and Both (Rules and Ensembles of each type together). Table 7 shows the average tardiness values for the 30 ensembles from each subset. The differences between the combined ensembles developed from the 7 subsets of rules and the collaborative ensembles can be observed using the boxplots in Fig. 7 to better identify them. From these results, we can see that the combined ensembles formed only from rules of any type, which are in fact single collaborative ensembles, are clearly inferior to those formed from collaborative ensembles, with the sole exception of the combined ensembles formed only from specialized collaborative ensembles. This result shows that this type of ensembles is so specialized for certain instances that it cannot cover other instances. It is also clear that general collaborative ensembles are the best building blocks for building combined ensembles. Even though collaborative ensembles perform worse than coordinated ensembles when applied to the whole set of

Table 7
Tardiness values of the combined ensembles generated by MA from different subsets or rules and collaborative ensembles.

Set		Training		Test	
		Best	Avg	Best	Avg
Rules	Gen.	1551.91	1552.10	1558.30	1559.44
	Spe.	1553.48	1553.50	1561.17	1561.81
	Both	1550.82	1550.83	1557.61	1557.92
Ensembles	Gen.	1541.75	1541.75	1549.84	1550.74
	Spe.	1548.08	1548.17	1556.39	1556.56
	Both	1541.72	1541.81	1549.78	1550.79
Rules + Ens.	Both	1541.52	1541.62	1550.27	1551.06



(a) Training set (1000 instances).

(b) Test set (1000 instances).

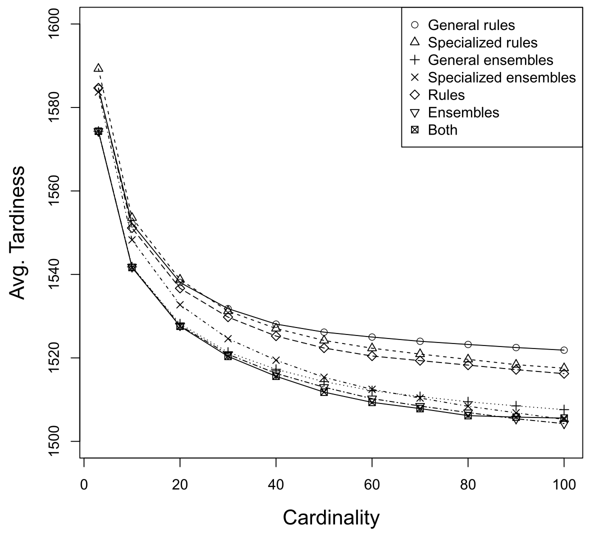
Fig. 7. Box plots of the results from Table 7.

instances (see Table 5), they show high performance on some subsets, so their combination gives the best overall performance for the whole set of instances. Therefore, combined ensembles are the best option as long as they contain general collaborative ensembles as a building block and are independent of considered rules and specialized ensembles. In fact, Kruskal-Wallis tests showed no statistical differences between these three combinations.

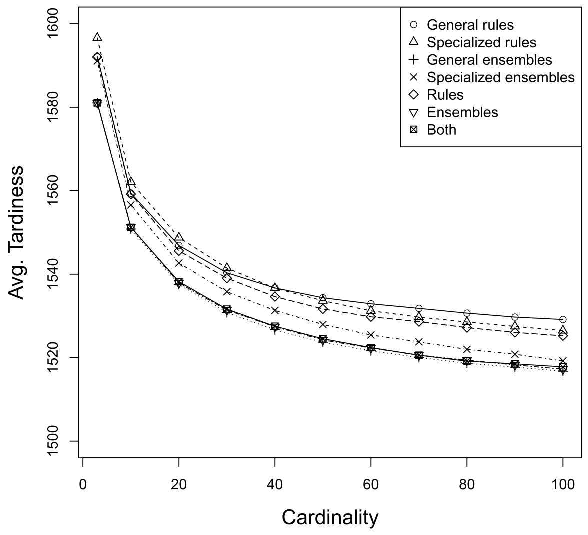
To further investigate the effect of ensemble size on performance and convergence, MA is run 30 times for different ensemble sizes ranging from 3 to 100. The average tardiness obtained for solving the training and test sets with the obtained ensembles is shown in Figs. 8 (a) and (b), respectively. The best results are always obtained with collaborative ensembles for each ensemble size. Moreover, it is obvious that the combination of specialized and general ensembles yields the best overall results. This shows that the ensembles that combine both have a higher chance of achieving good results. To better understand the structure of the evolved ensembles, in Fig. 8 (c) we plot the distribution of each building block in the ensemble as a function of the cardinality of the ensembles. We see that for smaller cardinality, general ensembles are the most frequently used building blocks. However, as cardinality increases, the proportion of specialized ensembles gradually increases, while the proportion of general ensembles decreases. The proportion of both types of rules increases only at the beginning and then remains constant regardless of the increase in cardinality.

From these observations we conclude that for smaller ensembles, general ensembles are the most appropriate building blocks. This is to be expected, since they usually perform better than individual rules, and hence ensembles may yield the best results when used. However, as cardinality increases, there is more opportunity to include specialized ensembles that can perform well for specific instances because they are better adapted to specific situations than general ensembles. However, specialized ensembles are not suitable for a wide range of instances. Therefore, it is still necessary to include a certain number of general ensembles that are suitable for the instances that are not covered by the specialized ensembles.

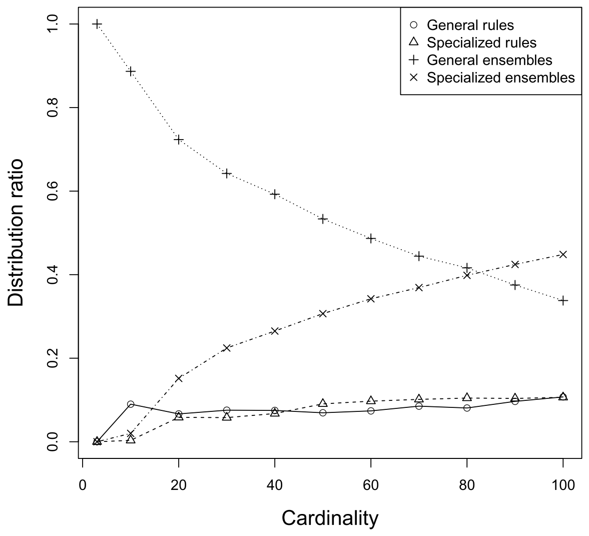
One of the most important conclusions we can draw from the experimental study is that coordinated ensembles are generally better than collaborative ensembles. A natural explanation for this is that coordinated ensembles create many schedules simultaneously, and thus it is likely that at least one of the rules can find a good schedule. In contrast, collaborative ensembles create only one schedule. Even though each decision is more reasonable because it results from aggregating the recommendations of a set of rules,



(a) Training set (1000 instances).



(b) Test set (1000 instances).



(c) Test set (1000 instances).

Fig. 8. The average tardiness (in (a) and (b)) obtained with combined ensembles with sizes 3, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 built by the MA using the seven sets of building blocks average from 30 executions. The distribution of the building blocks that compose the combined ensembles is outlined in (c).

the final solution may not be optimal if it makes a bad decision in one step. At the same time, a collaborative ensemble may be better than all of its rules. Therefore, the effect of including collaborative ensembles as building blocks in addition to individual rules may result in better coordinated ensembles than those consisting only of individual rules.

6. Conclusions

In this paper, we show that the performance of schedule builders for the $(1, Cap(t)|| \sum T_j)$ problem can be improved by using new ensemble approaches. On the one hand, ensembles can be used in a collaborative or coordinated manner, but both approaches can also be combined. In this paper, we proposed an approach where coordinated ensembles consist not only of rules, but also of other ensembles that create a single solution collaboratively (through combination methods).

The experiments show that better results can be obtained by using collaborative ensembles than by using single priority rules. Moreover, a combination of collaborative and coordinated ensembles led to the best results on the considered problem. Through various analyzes, we found that the way the rules and collaborative ensembles are developed has a significant impact on the quality of the coordinated ensembles. As expected, when trained with a small set of instances, they tend to specialize in a particular type of instance. However, when a larger set of instances is used, they tend to generalize without adapting too much to the training set. Also, when creating smaller ensembles, it is better to build them from ensembles that generalize well across different instances, while for large ensembles it is better to include ensembles or rules that are specialized to a smaller number of instances.

7. Future work

This work leaves several lines open for future research. First, the methodology proposed in this work can be extended from several points of view:

1. The development of surrogate models to reduce the high computational cost of these approaches [37,47].
2. The use of local improvement mechanisms specifically designed for collaborative ensembles can help to achieve better results.
3. Testing other combination methods such as majority voting, linear combination, weighted majority voting, and weighted linear combination [32]. Other machine learning methods can also be adapted, such as other variants of Borda Count or alternative voting methods [2,8,9].
4. Using other recent optimizers to create the ensembles [38,3].

On the other hand, priority rules can be used to improve the performance of other algorithms. For example, Vlasic et al. [40] improved evolutionary algorithms by population initialization with rules for the unrelated machine environment. Finally, the same methodology could be applied to develop online methods for other scheduling problems, which would allow comparison with methods proposed in the literature, as well as consideration of additional constraints such as setup times and precedence constraints for the single-machine environment [25].

CRediT authorship contribution statement

Francisco J. Gil-Gala: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft. **Marko Đurasević:** Conceptualization, Investigation, Methodology, Supervision, Writing – review & editing. **Ramiro Varela:** Funding acquisition, Methodology, Project administration, Supervision, Writing – review & editing. **Domagoj Jakobović:** Funding acquisition, Project administration, Supervision, Writing – review & editing.

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 work has been supported in part by the Spanish State Agency for Research (AEI) under research project PID2019-106263RB-I00 and the Croatian Science Foundation under the project IP-2019-04-4333.

Appendix A. Algorithms for building ensembles

All proposed algorithms encode ensembles as variations with repetitions of size P, where P is the size of the ensemble to be constructed. The elements of the solution are selected from a set of N available building blocks (rules or ensembles), resulting in a total number of P^N distinct ensembles in the entire search space. A set of N building blocks and a size of ensembles P is required to run all algorithms, as well as a set of M instances of the $(1, Cap(t)||\sum T_i)$ problem to evaluate the ensembles.

The Local Search Algorithm (LSA) is summarized in Algorithm 2. The algorithm starts with an initial ensemble K_0, which it iteratively tries to improve. In each iteration, the algorithm determines the worst component of the ensemble, i.e., the component that contributes the least to the quality of the ensemble (whose removal leads to the greatest degradation of the ensemble's performance), and removes it from the ensemble. Then, the algorithm selects n potential building blocks that could potentially be added to the ensemble to replace the removed building block. Which of the n building blocks is selected is determined by either the Hill Climbing (HC) or Gradient Descent (GD) strategies. In HC, the first building block that improves the current solution is selected, while in GD all n building blocks are evaluated and the one that led to the greatest improvement in the solution is inserted into the ensemble. In summary, LSA uses a unique neighbourhood structure in which each neighbour of an ensemble is created by replacing the worst building block of the ensemble with a building block from a subset $\mathcal{N}'$ of $\mathcal{N}$ of n elements that are uniformly selected. The complexity of the LSA is on the order of $\mathcal{O}(nPI)$, where I is the maximum number of iterations.

Algorithm 3 shows the applied Iterative Greedy Algorithm (IGA). At each iteration, in the design phase, the algorithm determines the building block to be added to the ensemble that will lead to the greatest improvement in the ensemble's performance. This is done by adding each potential building block to the current ensemble and evaluating the performance of this new ensemble to determine

Algorithm 2 Local Search Algorithm.

```
Data: An initial ensemble $K_0$. A set $\mathcal{N}$ of $N$ candidate building blocks. Acceptance Criterion: Hill Climbing ($HC$) or Gradient Descent ($GD$). The maximum number of iterations $I$
Result: An (hopefully) improved ensemble $K$.
 1: $K = K_0$
 2: $i = 0$
 3: $stopping\_condition \leftarrow$ false
 4: while not $stopping\_condition$ and $i < I$ do
 5:     $R_w \leftarrow$ the worst building block of $K$
 6:     $K' \leftarrow K$
 7:     $\mathcal{N}' \leftarrow n$ building blocks selected from $\mathcal{N}$
 8:     for all $R_q \in \mathcal{N}'$ do
 9:         $K'' \leftarrow K \setminus \{R_w\} \cup \{R_q\}$
10:         if $K''$ is better than $K'$ then
11:             $K' \leftarrow K''$
12:             if $improving\_condition$ = HC then
13:                 break
14:     if $K'$ is better than $K$ then
15:         $K = K'$
16:     else
17:         $stopping\_condition \leftarrow$ true
18:     $i = i + 1$
19: return $K$
```

how the addition of the new building block would affect the performance of the ensemble. The algorithm terminates when either an ensemble of the maximum allowed size has been created or when the ensemble cannot be further improved by adding building blocks. Therefore, the complexity of the IGA is on the order of $\mathcal{O}(NP)$. As shown in [18], it has an approximation ratio of $1 - \frac{1}{e}$. Moreover, the solution generated by IGA is optimal since it has the smallest tardiness for each ensemble that can be formed from N when $K < P$.

In [18], IGA is combined with LSA in two different ways. In the first, LSA is used to improve the final ensemble generated by IGA, which means that the solution generated by IGA is used as the initial solution for LSA. In the second, LSA is applied in a destruction phase to each partial solution obtained by IGA. However, both combinations did not improve the results, so they were discarded.

Algorithm 3 Iterative Greedy Algorithm.

```
Data: The maximum size of the ensemble $P$. A set $\mathcal{N}$ of $N$ candidate building blocks.
Result: The ensemble $K$.
 1: $K = \emptyset$
 2: $I = 0$
 3: while $I < P \wedge may\_improve$ do
 4:     $K' = K$
 5:     for all $R_q \in \mathcal{N}$ do
 6:         $K'' = K \cup \{R_q\}$
 7:         if $K''$ is better than $K'$ then
 8:             $K' = K''$;
 9:     if $K = K'$ then
10:         $may\_improve = false$
11:     else
12:         $K = K'$
13:         $I = I + 1$
14: return the ensemble $K$
```

The general structure of both the Genetic Algorithm (GA) and the Memetic Algorithm (MA) is the same and is presented in Algorithm 4. The main difference between the algorithms is that MA uses the LSA procedure to potentially improve the solutions, while GA does not. The evolutionary process starts with an initial population of $\#popsize$ individuals generated randomly, and then runs a number of generations ($\#gen$). Each iteration begins with a selection procedure in which individuals are randomly selected in pairs. In the next step, recombination, each pair is crossed according to the mating and mutation probabilities p_c and p_m, and the resulting ensembles are mutated. Crossover uses a single-point operator, whereas mutation replaces a number of building blocks between 1 and $P/2$ of the chromosome with random building blocks chosen uniformly from N. An example of crossover and mutation is shown in Figs. 9 and 10, respectively. After these steps, each ensemble is additionally improved by LSA (Algorithm 2) with probability p_{LS} in the case of MA. Finally, in the replacement step, the two best ensembles are selected from each pair of parents and offspring and passed on to the next generation, introducing an implicit form of elitism into the algorithm. The complexity of GA is on the order of $\mathcal{O}(gsP)$, where g is the number of generations and s is the population size, since the genetic operators are on the order of $\mathcal{O}(s)$. The complexity of MA is increased by that of LSA compared to GA.

Algorithm 4 Evolutionary algorithm.

```
Data: A set $\mathcal{N}$ of $N$ building blocks, crossover probability $p_c$, mutation probability $p_m$, LSA probability $p_{LS}$, number of generations $\#gen$, population size $\#popsize$,
      chromosome length $P$ (the size of the ensemble).
Result: A ensemble of $P$ building blocks
1: Generate and evaluate the initial population $\mathcal{P}(0)$ with $\#popsize$
2: for t=1 to $\#gen$-1 do
3:     Selection: organize the ensembles in $\mathcal{P}(t-1)$ into pairs of parents at random
4:     Recombination: mate each pair of parent ensemble and mutate the two offsprings in accordance with $p_c$ and $p_m$
5:     Evaluation: evaluate the resulting ensemble
6:     LocalSearch: apply LSA (Algorithm 2) to the offspring in accordance with $p_{LS}$
7:     Replacement: make a tournament selection of two ensembles from every two parents and their offsprings to build the population in the next generation $\mathcal{P}(t)$
8: return the best ensemble reached
```

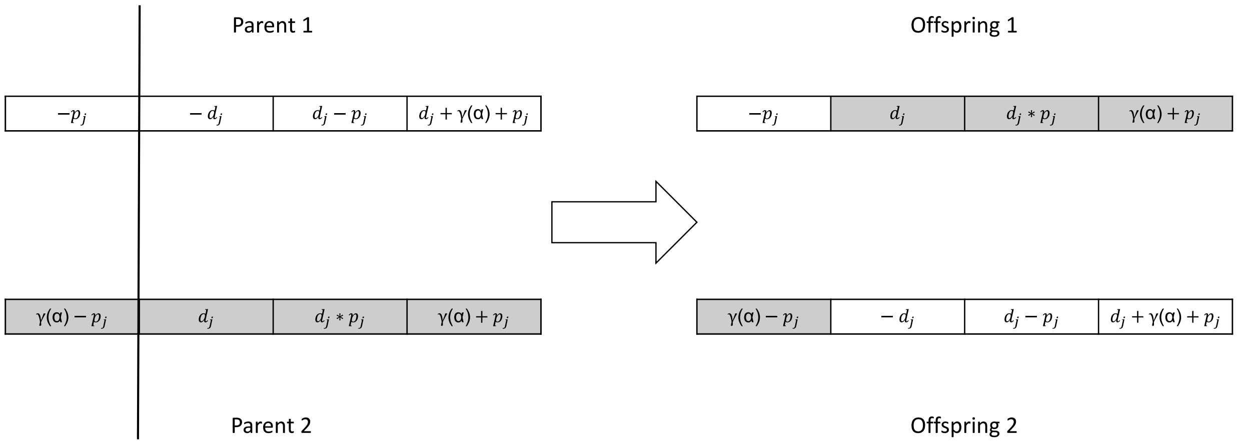


Fig. 9. Example of crossover of ensembles.

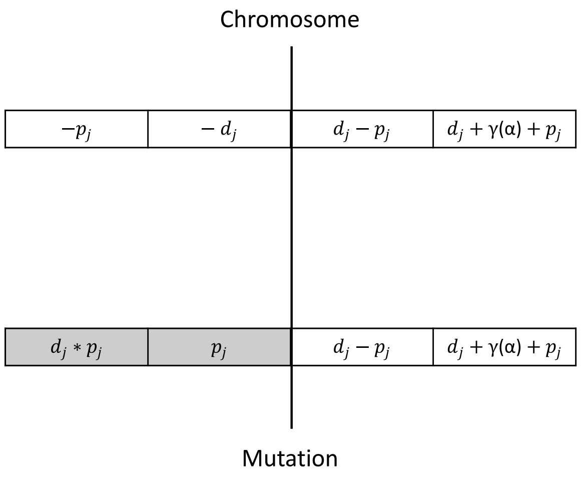


Fig. 10. Example of mutation of ensembles.

Appendix B. Run-time analysis

To illustrate the differences in the execution time of the various algorithms used to construct the ensembles, a brief runtime analysis of all the algorithms is performed. Tables 8 and 9 report the execution time (in seconds), the number of fitness evaluations, and the performance of the evolved ensembles for both the training and test sets. In both experiments, the maximum execution time was set to 10 seconds, while the ensemble size was set to 10 in the first experiment (results in Table 8) and 100 in the second experiment (results in Table 9).

The results in Table 8 show that IGA is by far the fastest at forming ensembles and in some cases terminates before the time limit (since it has exhausted all the combinations it had to try). Of course, it performed fewer fitness evaluations compared to the other algorithms. The other three algorithms perform a similar number of fitness evaluations. The most inefficient of the methods is LSA, since it performs the least number of fitness evaluations due to the time it takes to build neighbours.

Table 8
Summary of the results (tardiness) obtained when combined ensembles are evolved using IGA, GA, LSA and MA from 30 runs with ensemble size 10 and time limit of 10 seconds.

		Time	Fitness	Training		Test	
Set	Algorithm	(seconds)	evaluations	Best	Avg.	Best	Avg.
Rules	IGA	0.90	19300.00	1551.50	1551.50	1559.74	1559.74
	GA	10.00	179114.33	1551.00	1551.75	1558.06	1559.55
	LSA	10.00	167255.33	1552.05	1552.80	1557.72	1559.30
	MA	10.00	185284.70	1550.91	1551.23	1557.56	1559.24
Ensembles	IGA	0.67	13900.00	1542.12	1542.12	1550.21	1550.21
	GA	10.00	176904.27	1541.90	1542.36	1549.37	1550.83
	LSA	10.00	156064.77	1542.29	1542.83	1549.65	1550.99
	MA	10.00	187221.37	1541.72	1541.91	1549.76	1551.00
Both	IGA	1.33	33200.00	1542.12	1542.12	1550.21	1550.21
	GA	10.00	198856.20	1541.54	1542.65	1549.75	1551.32
	LSA	10.00	167015.43	1542.76	1543.38	1549.66	1551.42
	MA	10.00	182303.83	1541.56	1541.86	1549.72	1551.30

Table 9
Summary of the results (tardiness) obtained when combined ensembles are evolved using IGA, GA, LSA and MA from 30 runs with ensemble size 100 and time limit of 10 seconds.

		Time	Fitness	Training		Test	
Set	Algorithm	(seconds)	evaluations	Best	Avg.	Best	Avg.
Rules	IGA	10.00	68720.37	1522.43	1525.03	1534.59	1536.54
	GA	10.00	28385.13	1516.23	1517.50	1524.42	1526.58
	LSA	10.00	20170.40	1518.15	1519.11	1525.47	1526.42
	MA	10.00	17455.70	1517.73	1519.13	1525.66	1527.08
Ensembles	IGA	10.00	60643.23	1511.75	1513.27	1524.06	1525.74
	GA	10.00	33887.07	1504.44	1505.27	1516.31	1517.98
	LSA	10.00	27206.63	1504.96	1505.73	1517.06	1517.91
	MA	10.00	30188.93	1504.71	1505.75	1515.92	1517.76
Both	IGA	10.00	94107.73	1518.53	1520.56	1531.62	1533.36
	GA	10.00	34008.60	1506.21	1507.43	1517.94	1519.23
	LSA	10.00	21041.43	1507.58	1508.27	1517.96	1518.88
	MA	10.00	25559.17	1506.37	1507.37	1517.31	1518.28

Table 9 shows the results for the same analysis, but considering ensembles of size 100. In this case, we see that IGA was unable to terminate early, even though it had a significantly larger number of fitness evaluations than the other methods. This is because it is a fairly simple method compared to the other methods tested. For the other three methods, we can observe that there are slightly larger differences in the number of fitness evaluations between each algorithm. This time GA performs the largest number of function evaluations, while the other two methods perform a similar number of evaluations. This difference in runtime is likely due to the added complexity of LSA (which is also used in MA). It is also interesting to note that increasing the size of the ensemble had the largest effect on the number of evaluations performed by the LSA algorithm, implying that its runtime is quite sensitive to the size of the evolved ensembles.

From the point of the algorithm performance, we can see that for the ensemble size of 10 all algorithms performed equally well. However, as the size of the ensembles increased to 100, the performance of the IGA was inferior to the other three methods, which again performed equally well.

References

[1] Test bed data, http://di002.edv.uniovi.es/iscop/index.php/repository/summary/2-benchmarks/31-single-machine-scheduling-ensembles. (Accessed 28 September 2022) (Online).

[2] J.A. Aledo, J.A. Gámez, A. Rosete, A highly scalable algorithm for weak rankings aggregation, Inf. Sci. 570 (2021) 144–171, https://doi.org/10.1016/j.ins.2021.04.034.

[3] J. Bi, H. Yuan, J. Zhai, M. Zhou, H.V. Poor, Self-adaptive bat algorithm with genetic operations, IEEE/CAA J. Autom. Sin. 9 (2022) 1284–1294, https://doi.org/10.1109/JAS.2022.105695.

[4] J. Branke, T. Hildebrandt, B. Scholz-Reiter, Hyper-heuristic evolution of dispatching rules: a comparison of rule representations, Evol. Comput. 23 (2015) 249–277, https://doi.org/10.1162/EVCO-a-00131.

[5] J. Branke, S. Nguyen, C.W. Pickardt, M. Zhang, Automated design of production scheduling heuristics: a review, IEEE Trans. Evol. Comput. 20 (2016) 110–124, https://doi.org/10.1109/TEVC.2015.2429314.

[6] E.K. Burke, M.R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J.R. Woodward, A Classification of Hyper-Heuristic Approaches: Revisited, International Series in Operations Research & Management Science, vol. 272, Springer International Publishing, 2019, pp. 453–477.

[7] S. Chand, Q. Huynh, H. Singh, T. Ray, M. Wagner, On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems, Inf. Sci. 432 (2018) 146–163, https://doi.org/10.1016/j.ins.2017.12.013.
[8] D. Chen, Y. Xiao, H. Zhu, Y. Deng, Robustness of rank aggregation methods for malicious disturbance, Inf. Sci. 624 (Jun 2023) 639–651, https://doi.org/10.1016/j.ins.2023.01.008.
[9] P. Drotár, M. Gazda, L. Vokorokos, Ensemble feature selection using election methods and ranker clustering, Inf. Sci. 480 (2019) 365–380, https://doi.org/10.1016/j.ins.2018.12.033.
[10] M. Dumić, D. Šišejkovic, R. Čorić, D. Jakobović, Evolving priority rules for resource constrained project scheduling problem with genetic programming, Future Gener. Comput. Syst. 86 (2018) 211–221, https://doi.org/10.1016/j.future.2018.04.029.
[11] M. Durasević, D. Jakobović, Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment, Genet. Program. Evol. Mach. 19 (2018) 53–92, https://doi.org/10.1007/s10710-017-9302-3.
[12] M. Durasević, D. Jakobović, Creating dispatching rules by simple ensemble combination, J. Heuristics 25 (2019) 959–1013, https://doi.org/10.1007/s10732-019-09416-x.
[13] M. Durasević, D. Jakobović, K. Knežević, Adaptive scheduling on unrelated machines with genetic programming, Appl. Soft Comput. 48 (2016) 419–430, https://doi.org/10.1016/j.asoc.2016.07.025.
[14] M. Durasević, L. Planinić, F.J.G. Gala, D. Jakobović, Novel ensemble collaboration method for dynamic scheduling problems, in: Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, 2022, pp. 893–901.
[15] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory, Springer-Verlag, 1995, pp. 23–37.
[16] J. García-Álvarez, M.A. González, C.R. Vela, Metaheuristics for solving a real-world electric vehicle charging scheduling problem, Appl. Soft Comput. 65 (2018) 292–306, https://doi.org/10.1016/j.asoc.2018.01.010.
[17] F.J. Gil-Gala, C. Mencía, M.R. Sierra, R. Varela, Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time, Appl. Soft Comput. 85 (2019) 105782, https://doi.org/10.1016/j.asoc.2019.105782.
[18] F.J. Gil-Gala, C. Mencía, M.R. Sierra, R. Varela, Learning ensembles of priority rules for on-line scheduling by hybrid evolutionary algorithm, Integr. Comput.-Aided Eng. 28 (2021) 65–80, https://doi.org/10.3233/ICA-200634.
[19] F.J. Gil-Gala, M.R. Sierra, C. Mencía, R. Varela, Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling, Nat. Comput. (2020), https://doi.org/10.1007/s11047-020-09793-4.
[20] F.J. Gil-Gala, M. Ðurasević, M.R. Sierra, R. Varela, 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, Springer International Publishing, Cham, 2022, pp. 130–139.
[21] F.J. Gil-Gala, R. Varela, Genetic algorithm to evolve ensembles of rules for on-line scheduling on single machine with variable capacity, in: J.M. Ferrández Vicente, J.R. Álvarez-Sánchez, F. de la Paz López, J. Toledo Moreo, H. Adeli (Eds.), Bioinspired Systems and Biomedical Applications to Machine Learning, Springer International Publishing, Cham, 2019, pp. 223–233.
[22] R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326, https://doi.org/10.1016/S0167-5060(08)70356-X.
[23] E. Hart, K. Sim, A hyper-heuristic ensemble method for static job-shop scheduling, Evol. Comput. 24 (2016) 609–635, https://doi.org/10.1162/EVCO-a-00183.
[24] A. Hernández-Arauzo, J. Puente, R. Varela, J. Sedano, Electric vehicle charging under power and balance constraints as dynamic scheduling, Comput. Ind. Eng. 85 (2015) 306–315, https://doi.org/10.1016/j.cie.2015.04.002.
[25] D. Jakobović, K. Marasović, Evolving priority scheduling heuristics with genetic programming, Appl. Soft Comput. 12 (2012) 2781–2789, https://doi.org/10.1016/j.asoc.2012.03.065.
[26] J. Kittler, F. Alkoot, Sum versus vote fusion in multiple classifier systems, IEEE Trans. Pattern Anal. Mach. Intell. 25 (2003) 110–115, https://doi.org/10.1109/TPAMI.2003.1159950.
[27] L. Kletzander, N. Musliu, Solving large real-life bus driver scheduling problems with complex break constraints, in: ICAPS'20: Proceedings of the International Conference on Automated Planning and Scheduling, 2020, pp. 421–429.
[28] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
[29] J.R. Koza, Human-competitive results produced by genetic programming, Genet. Program. Evol. Mach. 11 (2010) 251–284, https://doi.org/10.1007/s10710-010-9112-3.
[30] C. Mencía, M.R. Sierra, R. Mencía, R. Varela, Evolutionary one-machine scheduling in the context of electric vehicles charging, Integr. Comput.-Aided Eng. 26 (2019) 1–15, https://doi.org/10.3233/ICA-180582.
[31] G. Nicolò, S. Ferrer, M. Salido, A. Giret, F. Barber, A multi-agent framework to solve energy-aware unrelated parallel machine scheduling problems with machine-dependent energy consumption and sequence-dependent setup time, in: ICAPS'19: Proceedings of the International Conference on Automated Planning and Scheduling, 2019, pp. 301–309.
[32] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Zhang, An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling, Appl. Soft Comput. 63 (2018) 72–86, https://doi.org/10.1016/j.asoc.2017.11.020.
[33] J. Park, S. Nguyen, M. Zhang, M. Johnston, Evolving ensembles of dispatching rules using genetic programming for job shop scheduling, in: P. Machado, M.I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, K. Sim (Eds.), Genetic Programming, Springer International Publishing, Cham, 2015, pp. 92–104.
[34] M.L. Pinedo, Scheduling, Springer, US, 2012.
[35] W. Qiu, J. Zhu, G. Wu, H. Chen, W. Pedrycz, P.N. Suganthan, Ensemble many-objective optimization algorithm based on voting mechanism, IEEE Trans. Syst. Man Cybern. Syst. 52 (2022) 1716–1730, https://doi.org/10.1109/TSMC.2020.3034180.
[36] J. Sedano, M. Portal, A. Hernández-Arauzo, J.R. Villar, J. Puente, R. Varela, Intelligent system for electric vehicle charging: design and operation, DYNA 88 (2013) 640–647, https://doi.org/10.6036/5788.
[37] M. Sun, C. Sun, X. Li, G. Zhang, F. Akhtar, Surrogate ensemble assisted large-scale expensive optimization with random grouping, Inf. Sci. 615 (2022) 226–237, https://doi.org/10.1016/j.ins.2022.09.063.
[38] J. Tang, G. Liu, Q. Pan, A review on representative swarm intelligence algorithms for solving optimization problems: applications and trends, IEEE/CAA J. Autom. Sin. 8 (2021) 1627–1643, https://doi.org/10.1109/JAS.2021.1004129.
[39] M. Đumić, D. Jakobović, Ensembles of priority rules for resource constrained project scheduling problem, Appl. Soft Comput. 110 (2021) 107606, https://doi.org/10.1016/j.asoc.2021.107606.
[40] I. Vlasić, M. Durasević, D. Jakobović, Improving genetic algorithm performance by population initialisation with dispatching rules, Comput. Ind. Eng. 137 (2019) 106030, https://doi.org/10.1016/j.cie.2019.106030.
[41] S. Wang, Y. Mei, J. Park, M. Zhang, Evolving ensembles of routing policies using genetic programming for uncertain capacitated arc routing problem, in: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), 2019, pp. 1628–1635.
[42] S. Wang, Y. Mei, M. Zhang, Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1093–1101.
[43] M. de Weerdt, M. Albert, V. Conitzer, K. van der Linden, Complexity of scheduling charging in the smart grid, in: IJCAI'18: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, 2018, pp. 4736–4742.

[44] G. Wu, R. Mallipeddi, P. Suganthan, R. Wang, H. Chen, Differential evolution with multi-population based ensemble of mutation strategies, Inf. Sci. 329 (2016) 329–345, https://doi.org/10.1016/j.ins.2015.09.009.
[45] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms – a survey, Swarm Evol. Comput. 44 (2019) 695–711, https://doi.org/10.1016/j.swevo.2018.08.015.
[46] G. Wu, X. Wen, L. Wang, W. Pedrycz, P.N. Suganthan, A voting-mechanism-based ensemble framework for constraint handling techniques, IEEE Trans. Evol. Comput. 26 (2022) 646–660, https://doi.org/10.1109/TEVC.2021.3110130.
[47] Z. Yang, H. Qiu, L. Gao, D. Xu, Y. Liu, A general framework of surrogate-assisted evolutionary algorithms for solving computationally expensive constrained optimization problems, Inf. Sci. 619 (2023) 491–508, https://doi.org/10.1016/j.ins.2022.11.021.
[48] Y. Yin, M. Liu, J. Hao, M. Zhou, Single-machine scheduling with job-position-dependent learning and time-dependent deterioration, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 42 (2012) 192–200, https://doi.org/10.1109/TSMCA.2011.2147305.
[49] M.A. Zahid, H. de Swart, The borda majority count, Inf. Sci. 295 (2015) 429–440, https://doi.org/10.1016/j.ins.2014.10.044.
[50] Z. Zhao, S. Liu, M. Zhou, X. Guo, L. Qi, Decomposition method for new single-machine scheduling problems from steel production systems, IEEE Trans. Autom. Sci. Eng. 17 (2020) 1376–1387, https://doi.org/10.1109/TASE.2019.2953669.