Ensembles of priority rules to solve scheduling problems 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*

Abstract

Priority rules are one of the most common and popular approaches to real-time scheduling. Over the last decades, a number of methods have been developed to automatically generate priority rules. Moreover, combining rules into ensembles showed to be better than using a single rule in many cases. In this paper, we analyze different ways to create and use ensembles of rules previously evolved by 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 build the same solution, while in the second case, each rule works independently on its own solution, and the best of them is taken as the solution of the ensemble. We observed that each method has its own strengths and weaknesses, which motivates their use in combination. From this hypothesis, we developed new methods to evolve and combine collaborative and coordinated ensembles and evaluated these methods on the One Machine Scheduling Problem with time-varying capacity and total tardiness minimization. The results of the experimental study provided interesting insights into the use of ensembles and show that our proposals outperform previous methods

*Corresponding author
Email addresses: giljavier@uniovi.es (Francisco J. Gil-Gala), Marko.Durasevic@fer.hr (Marko Đurasević), ramiro@uniovi.es (Ramiro Varela), Domagoj.Jakobovic@fer.hr (Domagoj Jakobović)

for the same problem.

Keywords: Scheduling, Priority Rules, Ensembles, Metaheuristics, Hyperheuristics.

---

1. Introduction

Scheduling problems arise in many real-world environments, such as injection moulding industry[38], bus driver scheduling [29], and electric vehicle charging [50]. In this paper, we focus on the One Machine Scheduling Problem (OMSP) with variable capacity over time and the objective of total tardiness, which is denoted as $(1, Cap(t)||\sum T_j)$ in the standard $\alpha|\beta|\gamma$ notation proposed in [21]. In this problem, the objective is to schedule a set of jobs without exceeding the capacity of the machine. Therefore, a priority must be computed for each of the jobs, and the machine processes them in the given order. The unique characteristic 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 or greater than 2.

The $(1, Cap(t)||\sum T_j)$ problem was introduced in [24] in the context of scheduling the charging times of a large fleet of electric vehicles. 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 creating a schedule builder. This framework is commonly referred to as *on-line scheduling* [23], as it can be used to create the schedule in parallel with its execution, in which case the 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 [34].

The performance of existing priority rules for various scheduling problems is still very limited, which makes it difficult to use an appropriate priority rule for the problem under consideration. Therefore, many studies put their focus

on the automatic design of priority rules using different methods, among which Genetic Programming (GP) stands out as the most commonly used [2].

Using GP, human competitive results have already been obtained in various fields such as hardware design, bioinformatics, symbolic regression and scheduling [31]. 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. [3], this paradigm is often referred to as hyper-heuristics based on *heuristic generation*. In recent years, many works have applied GP to evolve rules for various NP -hard problems, such as: bin packing [3], job shop scheduling [22, 36, 48], resource constrained project scheduling [4, 5, 8], scheduling unrelated parallel machines [12], one machine scheduling [6, 16] or travelling salesman problem [7], 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 called *the training set*. As in machine learning, the solution (the best rule evolved by GP) is evaluated on an unseen set of instances, which is in turn called *the test set*. Although the rules evolved by GP usually outperforms the ATC rule and other manually created rules, the results are still far from those obtained with evolutionary algorithms, leaving much room for improvement [34]. Since GP is a stochastic method, it is necessary to run it several times to evolve good rules. Usually, only the best evolved rule is used, which means that most of the evolved rules are simply discarded in the end. For these reasons, several papers studied the simultaneous application of a set of priority rules to generate schedules, i.e., ensembles [9, 18, 41].

In contrast to GP, which has long been used in on-line scheduling problems, ensemble methods have only recently gained attention and are being applied in the context of on-line scheduling problems [9, 41]. 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 [9, 41]. The rules collaborate by using a particular combination method that aggregates 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 building its own solution independently and then choosing the best solution among them [18].

In this paper, we are interested in creating ensembles specifically designed to solve the $(1, Cap(t)||\sum T_j)$ problem. In previous studies, only coordinated ensembles were considered for this problem. In this work, we develop collaborative ensembles; to do that we adapted the algorithms proposed in [18], which were used to design coordinated ensembles. Moreover, we study the influence of the cardinality and the composition of the problem set used by the algorithms for learning rules (GP [16]) and ensembles (mainly hybrid evolutionary algorithms [18]). Finally, we also propose to combine coordinated and collaborative ensembles to improve their performance. Experimental results show that collaborative ensembles achieve better results than a single rule. On the other hand, coordinated ensembles achieve much better results than collaborative ensembles when solving a large set of unseen instances.

The aims of this paper may be summarised as:

1. Adaptation of previous methods for constructing collaborative ensembles for the single machine scheduling problem.
2. Analyse ensemble cardinality and combination methods on the running time and performance of collaborative ensembles.
3. Combine collaborative and coordinated ensembles to further improve their efficiency.
4. Analyse and evaluate the effectiveness of combining collaborative and co-

ordinated ensembles that improves the current state of the art.

The main contributions of the paper may be summarized as follows

- Develop collaborative ensembles for the $(1, Cap(t)||\sum T_j)$ problem and establish a fair comparison between collaborative and coordinated ensembles on this problem.
- Analyze the strong and weak points of each class of ensembles and study different possibilities to exploit both of them in combination to obtain a synergistic effect.
- Establish a clear comparison of the best combination methods with the state-of-the-art proposals for the $(1, Cap(t)||\sum T_j)$ problem.

The remainder of the paper is organized as follows. In the next section we analyse the literature related to hyper-heuristics and scheduling under on-line conditions. In section 3 we provide the working hypotheses and purpose of the paper. Next we introduce the $(1, Cap(t)||\sum T_j)$ problem and review the proposed methods based on hyper-heuristics. Then, in section 5 we describe the proposed approach of creating collaborative and coordinated ensembles. In section, 6 we report the results of the experimental study. Finally, in sections 7 and 8, we summarize the main conclusions and outline some ideas for future work.

2. Preliminaries and literature review

As mentioned, the Genetic Programming framework proposed by John R. Koza [30] has proven to be one of the most successful methods to automatically generate priority rules; however, there are other emerging methods such as state-space search [17], Cartesian genetic programming [32] or gene expression

programming [39], which achieved good results as well. In addition to metaheuristics, machine learning techniques, such as neural networks [1, 13], have also been considered.

As for ensembles of priority rules, they have been created using different methods from different points of view. Park et al. [42] used the DeJong's cooperative coevolution framework [43] 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 [22] proposed an artificial immune network to evolve ensembles consisting of sequences of expression trees used sequentially to schedule each operation. These ensembles outperformed the ensembles developed in [42].

In their works about the Unrelated Parallel Machine Problem, Đurasević and Jakobović [9, 10] identified two key issues in constructing the ensembles: how the rules are combined and how they are selected for composing the ensemble. They proposed four learning approaches to create ensembles of priority rules for the unrelated machines environment: simple ensemble combination, BagGP, BoostGP, and cooperative coevolution. The simple ensemble combination method [9] was initially based on a simple random search, which generates up to 20 000 ensembles from random combinations of priority rules. They used exhaustive search for enumerating all possible solutions. BagGP evolves each rule on a different training set, while BoostGP applies the AdaBoost algorithm proposed in [14] for rule selection. The cooperative coevolution method is an evolutionary algorithm that divides the problem into several sub-problems, which are then solved by means of a sub-population each. The simple ensemble combination outperformed BagGP, BoostGP and cooperative coevolution. For this reason, they focused on this method and proposed five greedy methods:

- *Random selection* selects rules uniformly.

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

These methods were analyzed considering ensembles of sizes 3, 5 and 7. From an exhaustive experimental study, the authors concluded that ensembles made up of about 5 rules obtained better results than a single priority rule. The best methods being instance-based, grow and grow-destroy, even though ensembles calculated by random selection were sometimes better on average.

Regarding combination methods, Đurasević and Jakobović [9, 10] explored the two classic ways for combining rules into ensembles to take a decision, the sum and vote combination methods, which are common in the context of machine learning [28]. Park et al. [42, 40] and Hart and Sim [22] used majority voting. Besides, Park et al. [41] 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 exploit the ensembles where the rules are used in parallel to obtain a number of solutions to the problem instance. In this way, they obtain as many solutions as there are rules, whereas the previous approaches only obtain a single solution. They formulate the problem of computing ensembles of priority rules as the Optimal Ensemble

of Priority Rules Problem (OEPRP) and proposed three algorithms to solve it, namely an Iterated Greedy Algorithm (IGA), which is inspired by a similar algorithm for the Maximum Coverage Problem (MCP) [25], a Genetic Algorithm (GA) and a Local Search Algorithm (LSA).

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

3. Working hypotheses and purpose of the paper

As mentioned earlier, coordinated ensembles are able to outperform individual priority rules in the $(1, Cap(t)||\sum T_j)$ problem, and collaborative ensembles were good in unrelated parallel machines. But as far as we know, they have not yet been 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, and we will develop algorithms to evolve collaborative ensembles from the same set of rules and training sets. We will also try to highlight the weaknesses and strengths of each type of ensemble and explore the possibility of combining the two to achieve a synergistic positive effect.

4. 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 $t = 0$, that must be scheduled on a single

machine. The unique characteristic of this problem is that the capacity of the machine varies over time as $Cap(t) \geq 0, t \geq 0$. 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 p_j is the duration of the job j and C_j 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 the job j.

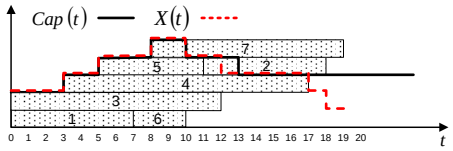


Figure 1: A feasible schedule to a $(1, Cap(t)||\sum T_j)$ instance with 7 jobs. $Cap(t)$ denotes the capacity of the machine over time and $X(t)$ is the capacity consumed by the jobs.

The $(1, Cap(t)||\sum T_j)$ problem stems from the Electric Vehicle Charging Scheduling Problem (EVCSP) described in [24]. The EVCSP is a dynamic problem motivated by the charging station designed in [46], in which a number of electric vehicles arrives to their parking lots over time, at times that are not known in advance, and their charging periods must be scheduled subject to some constraints, namely limited power, no interruption and balanced load on the three-phase feeder. In [24], the EVCSP is modeled as a sequence of static problems that are in turn decomposed into three instances of the $(1, Cap(t)||\sum T_j)$ problem each.

Due to the fact that the EVCSP must be solved online, each one of the $(1, Cap(t)||\sum T_j)$ instances must be solved in real-time. In [24], the authors proposed an effective solution by means of a stochastic schedule builder guided by the ATC rule. Other approaches have been proposed in the literature to solve

the EVCSP by means of metaheuristics such as genetic algorithm or ant colony optimization algorithm [15, 33], but neither of them fulfil the requirements to solve the problem online.

In [34], 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 6), but unfortunately it does not satisfy the real-time requirements arising from the online nature of the EVCSP. However, the schedule builder proposed in [34] 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 non-deterministically selects the job that can be scheduled at the earliest time from the partial schedule created 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 non-deterministic 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 is the classic ATC rule, which may 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 the job j as:

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

where p_j and d_j are the duration and the due date of job j respectively, $\gamma(\alpha)$ denotes the earliest starting time for a job in US^*, $\bar{p}$ is the average processing time of the jobs in US and g is a look-ahead parameter introduced by the user. Of course, a rule specifically tailored to the $(1, Cap(t)||\sum T_j)$ problem, or an ensemble of rules, might perform 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 may start at $\gamma(\alpha)$
    $US^* \leftarrow \{u \in US | X(t) < Cap(t), \gamma(\alpha) \leq t < \gamma(\alpha) + p_u\}$;
    Non-deterministically pick 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* $S = (st_1, st_2, ..., st_n)$;

5. Building rules and ensembles

In this section, we review the methods that have been proposed to evolve single rules and coordinated ensembles for the $(1, Cap(t)||\sum T_j)$ problem starting from a set of rules. Besides, we describe how the algorithms proposed to devise coordinate ensembles may be adapted to calculate collaborative ones starting from the same set of rules. Finally, we consider some ideas for combining both types of ensembles with the aim of taking advantage of their strong points and avoiding the weak ones.

5.1. Designing priority rules with GP

Given the success of Genetic Programming (GP) [30] in evolving priority rules for some problems as the job shop scheduling [37, 1], in [16] we proposed a GP to evolve rules for the $(1, Cap(t)||\sum T_j)$ problem. In this approach, the rules are built from the functions and terminal symbols given in Table 1. Besides, the used grammar restricts the expression trees to dimensionally compliant expressions and avoids generation of some equivalent expressions; in this way the search space is drastically reduced w.r.t. the full space of arithmetically correct expressions. Moreover, a maximum depth $\mathcal{D}$ of the expression tree with typical

values between 4 and 8 is considered to ensure both the readability of the rules and affordability of the search space. The rules evolved by GP demonstrated better performance than classic rules, such as the ATC rule. For further details of the GP approach, we refer the reader to [16].

Table 1: 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	-	+	/	×	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

5.2. 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 them already evaluated on a set of instances $\mathcal{P}$ of the $(1, Cap(t)||\sum T_j)$ problem denoted the training set. The objective (tardiness) values are recorded in a matrix $\mathcal{T}_{|\mathcal{R}|\times|\mathcal{P}|}$, so that $\mathcal{T}_{ij}$ is the tardiness of the schedule produced by the rule $\mathcal{R}_i$ on the instance $\mathcal{P}_j$. In principle, an ensemble is just a set of priority rules of a certain size, so it can be represented by a vector of single rules, whether they are used as collaborative or coordinated ensembles. Moreover, in this paper we consider the possibility of combining ensembles in such a way that one or more elements of an ensemble can in turn represent another ensemble. An example of such a combination is shown in Figure 2. In this case, we have a two-level ensemble with 3 elements in the first level, of which the last two are single rules, while the first one is an ensemble with only two single 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 multi-level 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 how to combine or modify ensembles, using the same algorithms proposed in [18] for creating coordinate ensembles.

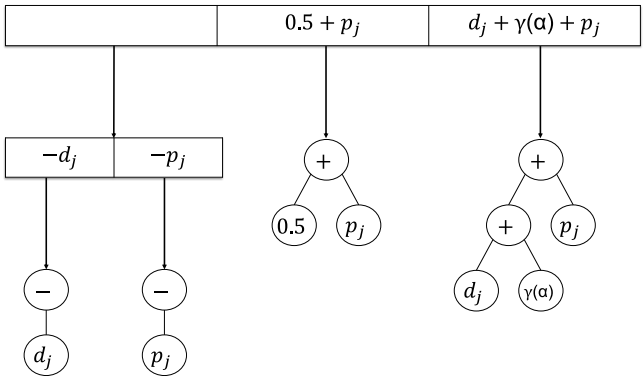


Figure 2: An example of ensemble composed of two rules and one ensemble, in turn composed of two single rules. Each rule is represented by the arithmetical expression in each array position; linked to each position is either the expression tree of a rule or the array of an ensemble.

5.2.1. Evaluation of ensembles

The evaluation of an ensemble, as for individual rules, consists in solving the instances of the training set $\mathcal{P}$. Then, the performance of the ensemble can be determined as the inverse of the accumulated tardiness of all $|\mathcal{P}|$ schedules obtained. 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. Figure 3 shows an example of a matrix $\mathcal{T}$ for a problem with 6 candidate rules and a training set with 7 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 performance measure of the ensemble, as they *cover* all the

instances of the test set, therefore the rule r_4 could be removed from the ensemble. Notice that even thought the rule r_0 has the worst average fitness on all instances, it achieves the best performance on most of the instances and thus it has a significant impact on the performance of the ensemble.

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

Figure 3: An example of a matrix of the tardiness values for a problem with 6 candidate rules and a training set with 7 instances.

On the other hand, evaluating a collaborative ensemble requires generating new solutions for all instances of $\mathcal{P}$ using the Algorithm 1, this time obtaining the priorities of the candidate jobs in US^* from aggregating the priorities of all the rules in the ensemble. In this way, the time required is greater than that for evaluating a coordinated ensemble, which is one of the drawbacks of collaborative ensembles. We consider here the two most typical aggregation methods: summation and voting, as considered in [9]. 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, the jobs are sorted from worst to best by each rule and each job is given 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 [41], the shortest turnaround time rule (SPT) [9], or even by chance. The voting method tends to lead to more draws,

while the summation method can introduce bias resulting from different scales in the priorities computed by the rules. Figure 4 shows an example of both aggregation methods for a situation with 4 jobs and 3 rules.

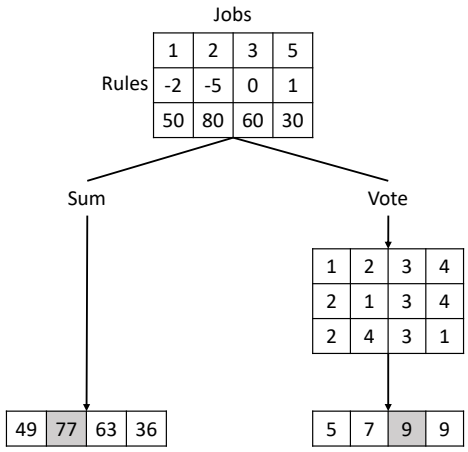


Figure 4: An example of summation and voting methods for an ensemble with 3 rules and 4 jobs. The first one will choose the second job as it presents the largest value of the summation of priorities, while the voting method would select the third one due to the largest sum of votes. In all cases the selected job is the one which has the highest value (priority or sum of votes).

As for the evaluation of multi-level ensembles, even if we restrict ourselves to two-level combinations, we must be aware that not all combinations are useful. For example, if the ensembles in the two levels are coordinated, the combined ensemble is equivalent to a coordinated ensemble of larger size. On the other hand, if we have a collaborative ensemble in the first level, it may not make sense to put a coordinated ensemble in the second level, and if we put collaborative ensembles in the second level the combined ensemble is in fact equivalent to a collaborative ensemble of a larger size. Therefore, the only reasonable combination of ensembles in two levels is to consider the ensemble in the first level as coordinated and the ensembles in the second level as collaborative, so this is the only combination we consider here.

In accordance with the above, the way to evaluate a multi-level ensemble

follows from the way in which the coordinated ensemble is evaluated at the first level. The only difference is in the contribution of the collaborative ensembles in the second level. If we only have the tardiness produced by the individual rules on the test 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.

5.2.2. Algorithms for building ensembles of priority rules

As mentioned before, when forming ensembles, we start from a set $\mathcal{R}$ of priority rules evolved by the GP proposed in [16]. From these rules, ensembles of a certain size P are built with the algorithms proposed in [18], namely an Iterated 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 they may be adapted for the creation of collaborative or multi-level 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 multi-level 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 produces the greatest improvement is chosen. 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, since only the instances in the training set $\mathcal{P}$ for which the rule is better than the ensemble before the rule addition need to be identified. Moreover, if none of the remaining rules is able to improve on 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 the possible repetitions. Since the order of the rules is not relevant, single point crossover and mutation operators are used and chromosomes are shuffled before mating.

LSA exploits a neighbourhood structure defined such that a single move exchanges one rule in the current ensemble for another one 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 ones.

LSA has been 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 [47, 44].

6. Experimental study

We have conducted an experimental study aimed to compare collaborative and coordinated ensembles and also to assess the performance of the proposed combined multi-level ensembles. To this end, we extended the algorithms proposed in [19] to build all types of ensembles. The algorithms are implemented in Java and were ran 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 single rules and coordinated ensembles. Then we analyze the results of collaborative ensembles and make a comparison with coordinated ensembles. Finally, we show the results of combined multi-level ensembles and illustrate their advantage over collaborative and coordinated ensembles.

6.1. The benchmark set and previous results

In this study, we used the set of $(1, Cap(t)||\sum T_j)$ problem instances and rules proposed in [18]. The first one includes 2000 instances with 60 jobs each and a machine whose capacity varies over time between 2 and 10, and were created to resemble the actual instances derived from the EVCSP [24]; 1000 instances are used for training and the remaining 1000 for testing. The set of rules is composed of 1000 *general* rules that were evolved by GP on the training instances. Specifically, the set of 1000 training instances was divided into 20 subsets with 50 instances each, then 50 rules were evolved on each subset. Another 1000 *specialized* rules were evolved for each one of the 1000 problem instances of the training set. After removing equivalent rules (those that calculate the same schedules to each of the 1000 instances of the training set), we obtained a set of 1930 diverse rules.

The results obtained in [18] with these rules and the coordinated ensembles of 10 rules each on the entire sets of 1000 training and 1000 test instances

are summarized in Table 2. In the first two rows, we include the results from the ATC rule ($g = 0.3$ being the best value from the 10 values $0.1; 0.2; ...; 1.0$ considered in these experiments) and the best from all 1930 rules evolved by GP. The next two rows show the values of the ensemble consisting of the 10 ATC rules and the ensemble consisting of the 10 best rules evolved by GP. The next 5 rows show the results of the coordinated ensembles obtained by the algorithms IGA, GA, LSA with random restarts, IGA followed by a single run of LSA (IGA-LSA) and MA respectively. In the cases of IGA and IGA-LSA only one ensemble was computed while for GA, LSA alone and MA 30 runs were performed and the best and average values are reported in the table. We can see that in each case the ensembles give much better results than single rules and that the ensembles created by the 5 proposed algorithms are better than the ensembles composed of 10 ATC rules or even the 10 best rules evolved 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 indeed a lower bound on the value of a coordinated ensemble that can be formed from these rules.

Table 2: 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 either one.

	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			

6.2. Collaborative versus coordinate ensembles

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

We start with a series of preliminary experiments in which collaborative ensembles are evolved from a training set of 50 instances. Specifically, each one is trained on a single instance from the training set, combining summation and voting methods, three cardinality values, 3, 5 and 10, and both algorithms IGA and GA. Only one run was done for each combination; so, we have 600 ensembles in all. 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 summarized in Table 3, where each value represents the average tardiness of 50 specialized ensembles, each of which solves the corresponding instance of the training set. The average running time 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 the 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 is 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 represents an advantage of this type

of ensembles over coordinated ensembles.

Table 3: 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, it reports the average run-time (seconds), the number of symbols and rules of each combination.

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	

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

Regarding the time taken by the ensembles to solve the 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. Figure 5 shows the boxplots of the times required to solve the test set using the 1930 rules and using 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 each rule. Moreover, in all cases, the ensembles are more time consuming than individual rules. We need to be aware that the ensembles evaluated in coordinated form require 3 or 10 times the time required for individual rules, depending on their cardinality.

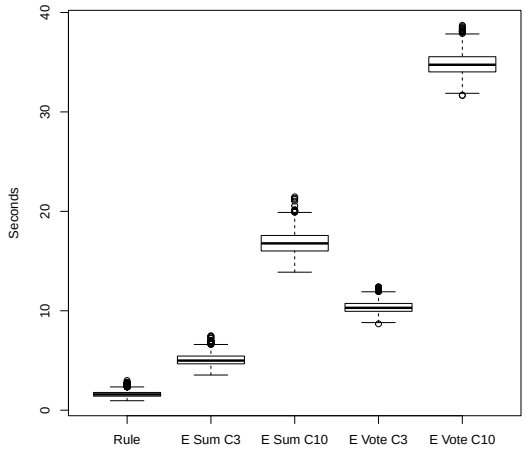


Figure 5: The run-time (seconds) taken by rules and collaborative ensembles when solving the test set (1000 instances).

Let us now consider the comparison between collaborative and coordinated ensembles. To this end, we created sets of ensembles that were trained using 50 instances. We obtained 20 ensembles of each class, collaborative and coordinated. Figure 6 shows the best and average convergence patterns of GA, averaged over the 20 runs. These experiments were performed in the aforementioned cluster, where GA takes about 3 minutes in a single run, while it would take only 3 seconds if the ensembles were trained on only one problem instance.

Table 4 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 the ensembles perform better than the

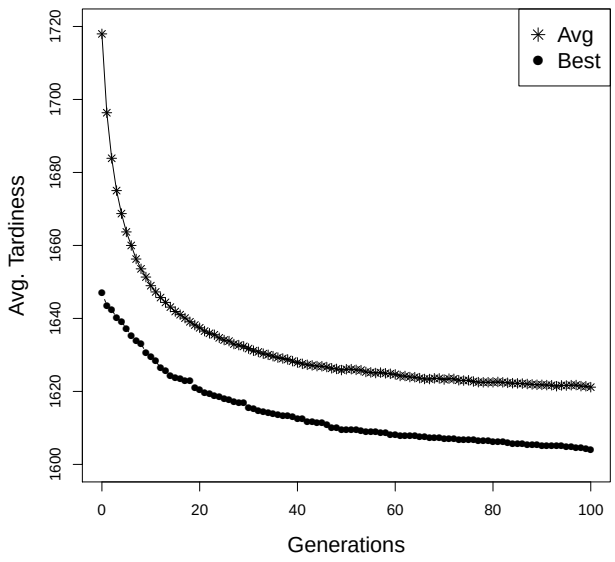


Figure 6: Convergence pattern of GA over 100 generations when calculates collaborative ensembles of 10 rules, selection by voting and trained with 50 problem instances. The values are averaged for 20 runs.

best rule in each case, and the coordinated ensembles perform better than the collaborative ensembles. These results, together with the fact that coordinated ensembles are much easier to compute, show a clear advantage of coordinated over collaborative ensembles.

Table 4: Summary of the results from ensembles on training (50 instances) and test (1000 instances) sets. Best Rule is the best of 1930 rules in average on the instances in each set. Collaborative and Coordinated are 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

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 5, where we can see that the specialized ensembles are absolutely dominant over 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.

Table 5: 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

From the above, 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 have high performance on subsets of instances of the training and testing sets, so they can contribute to the formation of powerful combined multi-level ensembles, which would definitely justify the interest in this type of ensembles.

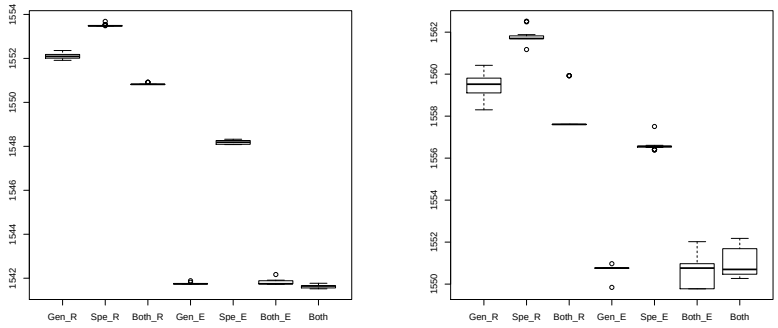
6.3. Evaluation of combined ensembles

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 2 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 neighbors and

5 iterations, as proposed in [18]. MA was run 30 times for each subset of 50 training instances. 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 ran 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 6 shows the tardiness values averaged for the 30 ensembles from each subset. The differences between the combined ensembles evolved from the 7 subsets of rules and the collaborative ensembles can be better seen from the boxplots in Figure 7. From these results, we can observe 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 composed by just specialized collaborative ensembles. This result shows that this type of ensembles is so specialized to 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 instances (see Table 4), 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 include 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.

Table 6: 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).

Figure 7: Box plots of the results in Table 6.

7. 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 analyses, we found that the way the rules and collaborative ensembles are evolved has a significant impact on the quality of the coordinated ensembles. As expected, when trained with a small set of instances, they tend to specialise in a particular type of instance. However, when a larger set of instances is used, they tend to generalise without over-fitting to the training set. Also, when creating smaller ensembles, it is preferable to build them from ensembles that generalise well across different instances, while for large ensemble sizes it is better to include ensembles or rules that specialise in a smaller number of instances.

8. Future work

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

1. The development of surrogate models to reduce the high computational cost of these approaches.
2. The use of local improvement mechanisms specifically designed for collaborative ensembles can help to achieve better results.
3. Try other combination schemes such as majority voting, linear combination, weighted majority voting, and weighted linear combination [41]. Other methods from machine learning can also be adapted, such as Borda Count [45].

On the other hand, priority rules can be used to improve the performance of other algorithms. For example, Vlasic et al. [49] improved evolutionary algorithms by population initialization with rules for the unrelated machine environment problem. Moreover, priority rules and ensembles can also be used as upper bounds for other algorithms such as Branch-and-Bound. They could

also be used to correct branch selection [35] or for guiding other algorithms such as the rollout algorithm [11].

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, such as job shop [26] or resource-constrained project scheduling problem [5], as well as considering additional constraints such as setup times and precedence constraints for the single-machine environment [27, 20].

Acknowledgements

This work has been supported in part by Spanish Government under research projects PID2019-106263RB-I00 and Croatian Science Foundation under the project IP-2019-04-4333.

References

[1] Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. Evolutionary Computation 23, 249–277. doi:10.1162/EVCO-a-00131.

[2] Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M., 2016. Automated design of production scheduling heuristics: A review. IEEE Transactions on Evolutionary Computation 20, 110–124. doi:10.1109/TEVC.2015.2429314.

[3] 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. doi:10.1162/EVCO_a_00044.

[4] Chand, S., Huynh, Q., Singh, H., Ray, T., Wagner, M., 2018. On the use of genetic programming to evolve priority rules for resource con-

strained project scheduling problems. Information Sciences 432, 146–163. doi:10.1016/j.ins.2017.12.013.

[5] Chand, S., Singh, H., Ray, T., 2019. Evolving heuristics for the resource constrained project scheduling problem with dynamic resource disruptions. Swarm and Evolutionary Computation 44, 897–912. doi:10.1016/j.swevo.2018.09.007.

[6] 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. doi:10.1016/S0965-9978(00)00109-5.

[7] 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. doi:10.1109/IPDPSW.2019.00094.

[8] 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. doi:10.1016/j.future.2018.04.029.

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

[10] Durasević, M., Jakobović, D., 2019. Creating dispatching rules by simple ensemble combination. Journal of Heuristics 25, 959–1013. doi:10.1007/s10732-019-09416-x.

[11] Durasević, M., Jakobović, D., 2020. Automatic design of dispatching

rules for static scheduling conditions. Neural Computing and Applications doi:10.1007/s00521-020-05292-w.

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

[13] Eguchi, T., Oba, F., Toyooka, S., 2008. A robust scheduling rule using a neural network in dynamically changing job-shop environments. International Journal of Machine Tools and Manufacture 14, 266–288. doi:10.1504/IJMTM.2008.017727.

[14] Freund, Y., Schapire, R.E., 1995. 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. p. 23–37.

[15] García-Álvarez, J., González, M.A., Vela, C.R., 2018. Metaheuristics for solving a real-world electric vehicle charging scheduling problem. Applied Soft Computing 65, 292–306. doi:10.1016/j.asoc.2018.01.010.

[16] 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, 105782. doi:10.1016/j.asoc.2019.105782.

[17] Gil-Gala, F.J., Mencía, C., Sierra, M.R., Varela, R., 2020. Exhaustive search of priority rules for on-line scheduling, in: Proceedings of the 2020 Conference on ECAI 2020: 24th European Conference on Artificial Intelligence. doi:10.3233/FAIA200365.

[18] Gil-Gala, F.J., Mencía, C., Sierra, M.R., Varela, R., 2021. Learning ensembles of priority rules for on-line scheduling by hybrid evolutionary algorithm. Integrated Computer-Aided Engineering 28, 65–80. doi:10.3233/ICA-200634.

[19] Gil-Gala, F.J., Varela, R., 2019. Genetic algorithm to evolve ensembles of rules for on-line scheduling on single machine with variable capacity, in: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Toledo Moreo, J., Adeli, H. (Eds.), Bioinspired Systems and Biomedical Applications to Machine Learning. Proceefingd, Springer International Publishing, Cham. pp. 223–233. Proceedings of IWINAC 2019.

[20] González, M.A., Palacios, J.J., Vela, C.R., Hernández-Arauzo, A., 2017. Scatter search for minimizing weighted tardiness in a single machine scheduling with setups. Journal of Heuristics 23, 81–110. doi:10.1007/s10732-017-9325-1.

[21] Graham, R., Lawler, E., Lenstra, J., Kan, A., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287 – 326. doi:10.1016/S0167-5060(08)70356-X.

[22] Hart, E., Sim, K., 2016. A hyper-heuristic ensemble method for static job-shop scheduling. Evolutionary Computation 24, 609–635. doi:10.1162/EVCO-a-00183.

[23] Hernández-Arauzo, A., Puente, J., González, M.A., Varela, R., Sedano, J., 2013. Dynamic scheduling of electric vehicle charging under limited power and phase balance constraints, in: ICAPS'13: Proceedings of SPARK'13. Scheduling and Planning Applications workshop, pp. 1–8.

[24] Hernández-Arauzo, A., Puente, J., Varela, R., Sedano, J., 2015. Electric vehicle charging under power and balance constraints as dy-

namic scheduling. Computers & Industrial Engineering 85, 306 – 315. doi:10.1016/j.cie.2015.04.002.

[25] Hochbaum, D.S., 1997. Approximation algorithms for np-hard problems, chapter Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems, pp. 94–143.

[26] Ingimundardottir, H., Runarsson, T.P., 2018. Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. Journal of Scheduling 21, 413–428. doi:10.1007/s10951-017-0534-0.

[27] Jakobović, D., Marasović, K., 2012. Evolving priority scheduling heuristics with genetic programming. Applied Soft Computing 12, 2781 – 2789. doi:doi.org/10.1016/j.asoc.2012.03.065.

[28] Kittler, J., Alkoot, F., 2003. Sum versus vote fusion in multiple classifier systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 110– 115. doi:10.1109/TPAMI.2003.1159950.

[29] Kletzander, L., Musliu, N., 2020. 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, pp. 421–429.

[30] Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.

[31] Koza, J.R., 2010. Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11, 251–284. URL: https://doi.org/10.1007/s10710-010-9112-3, doi:10.1007/s10710-010-9112-3.

[32] Manazir, A., Raza, K., 2019. Recent developments in cartesian genetic programming and its variants. ACM Computing Surveys 51, 1–29. doi:10.1145/3275518.

[33] Mavrovouniotis, M., Ellinas, G., Polycarpou, M., 2019. Electric vehicle charging scheduling using ant colony system, in: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 2581–2588. doi:10.1109/CEC.2019.8789989.

[34] 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. doi:10.3233/ICA-180582.

[35] Morikawa, K., Nagasawa, K., Takahashi, K., 2019. Job shop scheduling by branch and bound using genetic programming. Procedia Manufacturing 39, 1112 – 1118. doi:10.1016/j.promfg.2020.01.359.

[36] 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. doi:10.1162/evco_a_00230.

[37] Nguyen, S., Zhang, M., Johnston, M., Tan, K., 2013. Dynamic Multi-objective Job Shop Scheduling: A Genetic Programming Approach. volume 505. pp. 251–282. doi:10.1007/978-3-642-39304-4_10.

[38] Nicolò, G., Ferrer, S., Salido, M., Giret, A., Barber, F., 2019. 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, pp. 301–309.

[39] Nie, L., Shao, X., Gao, L., Li, W., 2010. Evolving scheduling rules with gene

expression programming for dynamic single-machine scheduling problems. The International Journal of Advanced Manufacturing Technology 50, 729–747.

[40] Park, J., Mei, Y., Nguyen, S., Chen, A., Johnston, M., Zhang, M., 2016. Genetic programming based hyper-heuristics for dynamic job shop scheduling: Cooperative coevolutionary approaches, in: GECCO'16: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, p. 109–110. doi:10.1145/2908961.2908985.

[41] Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M., 2018. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. Applied Soft Computing 63, 72–86. doi:10.1016/j.asoc.2017.11.020.

[42] Park, J., Nguyen, S., Zhang, M., Johnston, M., 2015. Evolving ensembles of dispatching rules using genetic programming for job shop scheduling, in: Machado, P., Heywood, M.I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K. (Eds.), Genetic Programming, Springer International Publishing, Cham. pp. 92–104.

[43] Potter, M., De Jong, K., 2000. Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evolutionary computation 8, 1—29. doi:10.1162/106365600568086.

[44] Raidl, G.R., Puchinger, J., Blum, C., 2019. Metaheuristic Hybrids. Springer International Publishing. pp. 385–417. doi:10.1007/978-3-319-91086-4_12.

[45] Rico, N., Pérez-Fernández, R., Díaz, I., 2020. The borda count as a tool for reducing the influence of the distance function on kmeans, in: de la

Cal, E.A., Villar Flecha, J.R., Quintián, H., Corchado, E. (Eds.), Hybrid Artificial Intelligent Systems, Springer International Publishing, Cham. pp. 450–461.

[46] Sedano, J., Portal, M., Hernández-Arauzo, A., Villar, J.R., Puente, J., Varela, R., 2013. Intelligent system for electric vehicle charging: Design and operation. DYNA 88, 640–647. doi:10.6036/5788.

[47] Talbi, E., 2009. Metaheuristics - From Design to Implementation. Wiley.

[48] 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. doi:10.1016/j.cie.2007.08.008.

[49] Vlasić, I., Durasević, M., Jakobović, D., 2019. Improving genetic algorithm performance by population initialisation with dispatching rules. Computers & Industrial Engineering 137, 106030. doi:10.1016/j.cie.2019.106030.

[50] de Weerdt, M., Albert, M., Conitzer, V., van der Linden, K., 2018. Complexity of scheduling charging in the smart grid, in: IJCAI'18: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, pp. 4736–4742.