Natural Computing manuscript No.
(will be inserted by the editor)

Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling

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

Received: date / Accepted: date

Abstract Combining metaheuristics is a common technique that may produce high quality solutions to complex problems. In this paper, we propose a combination of Genetic Programming (GP) and Genetic Algorithm (GA) to obtain ensembles of priority rules to solve a scheduling problem, denoted $(1, Cap(t)||\sum T_i)$, online. In this problem, a set of jobs must be scheduled on a single machine whose capacity varies over time. The proposed approach interleaves GP and GA so that a GP is in charge of evolving single priority rules and a GA evolves ensembles from the rules produced by the GP in each iteration. Therefore, the ensembles are produced in an anytime fashion. In the experimental study, we compare the proposed approach to a previous one in which the GP was firstly run to evolve a large pool of candidate priority rules, and then the GA was run to obtain ensembles from that pool of rules. The results of this study revealed that the ensembles produced by the interleaved combination of GP and GA are better than those obtained by the sequential combination of GP and GA. So, these results, together with the fact that the ensembles being available earlier, make this approach more appropriate to the online requirements of the scheduling problem. earlier.

Keywords One machine scheduling · hyper-heuristic · priority rules · ensemble learning · evolutionary algorithms

1 Introduction

This paper tackles the one machine scheduling problem with variable capacity, denoted $(1, Cap(t)||\sum T_i)$. In this problem, a number of jobs must be scheduled on a single machine, whose capacity varies over time, with the objective of minimizing the total tardiness objective function. This problem arose from the Electric Vehicle Charging Scheduling (EVCS) problem confronted in Hernández-Arauzo et al. (2015). Indeed, solving this problem requires solving many instances

Corresponding author: giljavier@uniovi.es

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

of the $(1, Cap(t)||\sum T_i)$ problem on-line. In Hernández-Arauzo et al. (2015), the $(1, Cap(t)||\sum T_i)$ problem is solved by means of the *Apparent Tardiness Cost* (ATC) priority rule.

Priority rules are of common use in on-line scheduling. They can be defined manually by experts on the problem domain, as it is the case of the ATC rule proposed by Koulamas (1994), although it is clear that automatic methods could capture some characteristics of the scheduling problem that are not clear to human experts. Under this assumption, in recent years, some hyper-heuristics as Genetic Programming (GP) was proposed to evolve priority rules for scheduling problems such as job shop (Park et al., 2015; Hart and Sim, 2016; Ingimundardottir and Runarsson, 2018; Nguyen et al., 2019), unrelated parallel machines (Durasević et al., 2016), bin packing (Burke et al., 2012), resource constrained project (Chand et al., 2018; Dumić et al., 2018), or the $(1, Cap(t)||\sum T_i)$ problem considered in Gil-Gala and Varela (2019).

As it may be expected, a single rule, even being very good in average for a large set of instances, may not be good for a number of them individually. For this reason, some researchers focused on calculating sets of rules that collaboratively solve the problem. This may be done in different ways. For example, in Hart and Sim (2016) the authors propose GP to obtain a set of rules that are applied in turn to schedule a single operation. Similar approaches were developed in Park et al. (2018) and Durasević and Jakobović (2018, 2019).

In Gil-Gala and Varela (2019), we took an alternative approach. Given the low time required to compute a solution to the instances of the $(1, Cap(t)||\sum T_i)$ problem generated when solving the EVSC in Hernández-Arauzo et al. (2015), we proposed to use a set of priority rules in parallel to obtain a number of solutions. The rationale of this is that if these rules are trained to solve instances with different characteristics, any of the rules will produce a good solution for each particular instance. The approach proposed in Gil-Gala and Varela (2019) consists of two steps. Firstly, a large pool of candidate priority rules is obtained by means of GP. Then, from this pool, a GA is used to evolve good ensembles of a given maximum size. This method produced very good ensembles, but the time taken to obtain them is too large due to the time required by GP to evolve the pool of candidate rules.

In this work we propose combining the GP and the GA in a different way. Given the long time required by GP to evolving from one generation, said T, to the next one $T + 1$, we propose to exploit the GA to evolve ensembles from the rules produced by the GP in generation T in parallel to the GP evolving generation $T+1$. In this way, we obtain a sort of *anytime* algorithm. As we will see, the evolved ensembles are similar to those obtained in Gil-Gala and Varela (2019), but they start to be available from much earlier.

The remainder of the paper is organized as follows. In the next section we introduce the $(1, Cap(t)||\sum T_i)$ problem and review the proposed solving methods. Then, in section 3 we describe the combined approach of the GP and the GA proposed to evolve rules and ensembles. In section 4 we report the results of the experimental study. Finally, in section 5, we summarize the main conclusions and outline some ideas for future work.

2 Background

In this section we review the state-of-the-art in the $(1, Cap(t)||\sum T_i)$ problem. We start from the formal definition of the problem and then describe the proposed methods to solve this problem on-line, i.e., schedule builders guided by priority rules. We also review the proposed methods to obtain good priority rules(Gil-Gala et al., 2019) and ensembles (Gil-Gala and Varela, 2019).

2.1 Definition of the $(1, Cap(t)||\sum T_i)$ problem

We are given a number of n jobs $\{1, \dots, n\}$, available at time $t = 0$, which have to be scheduled on a machine whose capacity varies over time. Job i has duration p_i and due date d_i. The goal is to allocate starting times $st_i, 1 \leq i \leq n$ to the jobs on the machine such that the number of jobs that are processed in parallel on the machine at any time t, $X(t)$, cannot exceed the capacity of the machine; i.e., $X(t) \leq Cap(t)$, where $Cap(t) \geq 0, t \geq 0$, is the capacity of the machine in the interval $[t, t+1)$; and the processing of jobs on the machine cannot be preempted, i.e., $C_i = st_i + p_i$, where C_i is the completion time of job i. The goal is minimizing the total tardiness, defined as:

$$\sum_{i=1,\dots,n} \max(0, C_i - d_i) \tag{1}$$

2.2 Building schedules on-line

A schedule builder, or schedule generation schema, is a non-deterministic algorithm that allows to enumerate schedules in a certain space. In Mencía et al. (2019), a schedule builder was proposed to the $(1, Cap(t)||\sum T_i)$ problem, which produce *left-shifted schedules*, a subspace of feasible schedules that contains at least one optimal schedule (see Algorithm 1). The algorithm maintains a set US with the unscheduled jobs, as well as the consumed capacity $X(t)$ due to the jobs scheduled so far. In each iteration, the algorithm builds the subset US^* containing the jobs in US that can be scheduled at the earliest possible starting time, denoted $\gamma(\alpha)$, and selects one of these jobs non-deterministically. This choice may be done either at random or by means of some priority rule so that the job having the highest priority in US^* is the one chosen to be scheduled next.

2.3 Priority rules for the $(1, Cap(t)||\sum T_i)$

In the literature there are a number of rules that could be adapted to the $(1, Cap(t)||\sum T_i)$ problem. Among them, we may consider the *Apparent Tardiness Cost* (ATC) rule, which was used with success to solve some scheduling problems with tardiness objectives (Sang-Oh Shim and Kim, 2007; Kaplan and Rabadi, 2012); with this rule, the priority of each unscheduled jobs is given by

$$\pi_j = \frac{1}{p_j} exp\left[\frac{-max(0, d_j - \gamma(\alpha) - p_j)}{g\bar{p}}\right] \tag{2}$$

Algorithm 1 Schedule Builder

```
Data: A $(1, Cap(t)||\sum T_i)$ problem instance $\mathcal{P}$.
Result: A feasible schedule $S$ for $\mathcal{P}$.
$US \leftarrow \{1, 2, ..., n\}$;
$X(t) \leftarrow 0; \forall t \geq 0$;
while $US \neq \emptyset$ do
    $\gamma(\alpha) = min\{t' | \exists u \in US; X(t) < Cap(t), t' \leq t < t' + p_u\}$;
    $US^* = \{u \in US | X(t) < Cap(t), \gamma(\alpha) \leq t < \gamma(\alpha) + p_u\}$;
    Non-deterministically pick job $u \in US^*$;
    Assign $st_u = \gamma(\alpha)$;
    Update $X(t) \leftarrow X(t) + 1; \forall t$ with $st_u \leq t < st_u + p_u$;
    $US \leftarrow US - \{u\}$;
end
return The schedule $S = (st_1, st_2, ..., st_n)$;
```

Functionals		
	Binary	- + / * *max min*
	Unitary	- pow_2 *sqrt exp ln*
Terminals		
	Data	p_i d_i $\gamma(\alpha)$ $\bar{p}$
	Constants	0.0 0.1 ... 0.9 1.0

Table 1: Functional and terminal symbols used to build expression trees.

where $\bar{p}$ is the average processing time of the jobs in US and g is a look-ahead parameter to be introduced by the user.

We may consider other priority rules simpler than ATC, for example *Earliest Due Date* (EDD) or *Shortest Processing Time* (SPT) rules, which calculate priorities for an eligible job j as $\pi_j = 1/d_j$ and $\pi_j = 1/p_j$ respectively. However, as these rules do not take into account some of the relevant attributes of the problem, it is expected that they perform worse than ATC. This fact was confirmed in some experiments reported in Gil-Gala et al. (2019), where both of them, specially SPT, produced much worse results than ATC with different values of g on a large set of instances of the $(1, Cap(t)||\sum T_i)$ problem.

2.4 Genetic programming to evolve priority rules

As pointed, a Genetic Programming (GP) approach was proposed in Gil-Gala et al. (2019) to evolve priority rules for the the $(1, Cap(t)||\sum T_i)$ problem. The objective is to come upon relations among attributes of the problem that are not evident to the human eye; and so to adapt the priority rules to the particular characteristics of the problem, or even to sets of instances having particular structure.

Table 1 shows the sets of terminals and symbols considered to build up expression trees representing priority rules. Figure 1 shows the expression tree corresponding to the ATC rule; in this figure, g should be substituted by some parameter in $0.1, \ldots, 1.0$.

The GP follows the design principles proposed in Koza (1992), but it was enhanced with some features to improve its efficiency and the rationality of the produced rules. The genetic operators are the same, although the search was re-

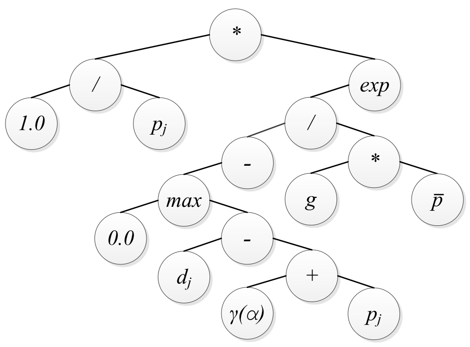

Fig. 1: Expression tree representing the ATC rule.

stricted to *dimensionaly aware* rules, which are more rational and easy to understand. Therefore, only expressions having the same dimension can be exchanged. Hence, the size and depth of the evolved trees are limited by two parameters, $\mathcal{S}$ and $\mathcal{D}$ respectively. To create initial chromosomes, we have used the ramped-half-and-half method proposed in Koza (1992).

Along the evolutionary process, each candidate rule is evaluated on a number of instances of the $(1, Cap(t)||\sum T_i)$ problem (the training set) and the inverse of the average tardiness taken as fitness value. In this way, it is expected that the GP is able to evolve rules that are good to the instances of the training set and others having similar structure. To assess that, the evolved rules must then be evaluated on a number of unseen instances (the test set).

As it may be expected, the evaluation of candidate rules is the most time consuming component of the GP; for this reason, we opted to keep stored the evaluated rules together with their results on the training set. This prevents GP for evaluating duplicate rules and at the same time makes available all evaluated rules when GP finishes, which will be useful in some variants of the proposed hybrid algorithm.

The experimental study reported in Gil-Gala et al. (2019) shows that the GP is able to obtain rules that are much more efficient than ATC, with any value of parameter g. For further details, we refer the reader to Gil-Gala et al. (2019).

2.5 Solving $(1, Cap(t)||\sum T_i)$ problem by ensembles of rules

Even though the rules obtained by GP are certainly good and may be specialized to sets or even particular instances of the $(1, Cap(t)||\sum T_i)$ problem, it is difficult to obtain a rule that is good in all instances of a given set, this is not surprising due to no free lunch theorem implications (Wolpert and Macready, 1997). From this reason, and taking into account the real-time requirements of the EVCS problem proposed in Hernández-Arauzo et al. (2015), we have considered in Gil-Gala and Varela (2019) the possibility of using ensembles of rules instead of just a single rule. As pointed, our proposal is to exploit in parallel a number of rules, i.e., an ensemble, to solve a given instance and select the best of the solutions as the solution produced by the ensemble. The rationale of this approach is that if these rules are well selected then there will be likely that at least one of them produce a good solution for any instance.

From the above assumption, in Gil-Gala and Varela (2019) we proposed a Genetic Algorithm (GA) to evolve ensembles from a large pool of rules obtained previously. This problem was formalized as a generalization of the maximum coverage problem. The main inconvenience of this approach is the time required to obtain the pool of rules, which are evolved from a number of runs of the GP. To solve this inconvenience, we propose herein an alternative way in which the GP and the GA are interleaved.

2.6 Solving $(1, Cap(t)||\sum T_i)$ problem by off-line algorithms

In Mencía et al. (2019) a memetic algorithm (MA) is proposed that, as far as we know, is the best method to solve the $(1, Cap(t)||\sum T_i)$ problem off-line. Thus, its solutions can be used as a reference in order to assess the quality of the on-line methods. The MA combines a generational genetic algorithm and a local search algorithm based on hill climbing. The evolutionary strategy is based on random selection and replacement by tournament among parents and offsprings, which confers its an implicit form of elitism. In this case, chromosomes are defined by permutations of the jobs. As in other works (González et al., 2012; Mencía et al., 2014), this encoding allows the MA to use the well-known Order Crossover (OX) or simple mutation operators. The decoder builds a schedule using Algorithm 1, scheduling the jobs in the order they appear in the chromosome. For further details, we refer the reader to Mencía et al. (2019).

3 The hybrid algorithm

In this paper, we propose interleaving the GP and the GA so that the GA calculates ensembles from the rules in the current generation of the GP at the same time as the GP evolves the next generation of rules from the current one. This method requires running the GA for a number of generations after each generation of the GP. In practice, this is not an inconvenience as one generation of the GP takes much more time than running the GA for, said, 100 generations or more.

In order to implement this method, the rules calculated by the GP in a generation are stored in a set, denoted $\mathcal{R}$, whose elements are tuples of the form $< Rule, Solutions >$. For a given rule *Rule*, *Solutions* is in turn another set with elements of the form $< Instance, Solution >$, where *Instance* is one of the instances of the training set and *Solution* is the schedule produced by *Rule* to that instance. Both sets may be efficiently implemented as hash tables.

The interleaved execution of the GP and the GA is shown in Figure 2. This schema expresses that the rules obtained in a generation of the GP can only be stored in the set $\mathcal{R}$ if the previous execution of the GA has finished, and that just after these rules are stored in $\mathcal{R}$ the generation of the initial population of the next execution of the GA can be done in parallel to the checking for the GP termination. Therefore, the next generation of the GP can be build at the same time as the GA evolves ensembles from the current one.

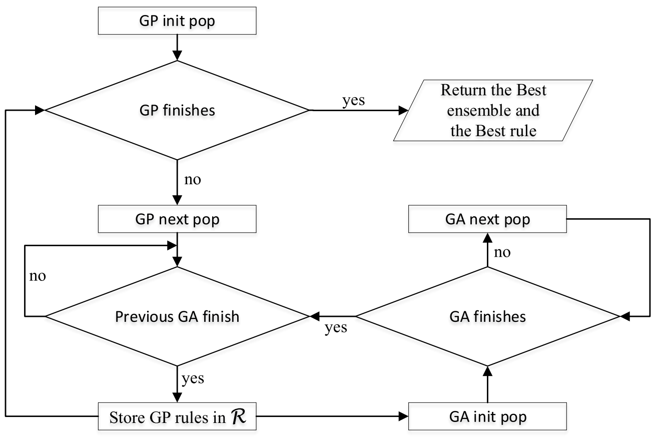

Fig. 2: Flow chart of the combined GP and GA algorithm.

Of course, different evolution schemas to that in Figure 2 could be considered. For example, we could take into account the ensembles evolved by the GA to identify the rules that most contribute to the best ensembles and maintain some of these rules in the next generation of the GP, what would be some kind of elitism; but this will require more detailed analysis to avoid other inconveniences as premature convergence. Besides, we could allow the GA to build ensembles from all different rules evolved by the GP in previous generations, but this schema would be more costly in both space and time. Therefore, we opted herein to keep the mentioned schema.

4 Experimental study

We have conducted an experimental study aimed at analyzing the behaviour of the proposed algorithm combining the GP and the GA. To this end, we implemented these algorithms from scratch in Java language and ran a series of experiments on a Linux cluster (Intel Xeon 2.26 GHz. 128 GB RAM). This cluster distributes the workload into 28 processing nodes, so the cluster is capable of executing up to 28 jobs in parallel. Making use of this feature and considering the stochastic nature of GP and GA, 28 independent runs were done for each input data. Then, best, average and standard deviation (SD) of the 28 solutions were recorded.

4.1 Test bed

We propose a new benchmark set that consists of instances more realistic than those considered in Gil-Gala et al. (2019) as they are much more similar to the instances that are expected to be generated from the system proposed in Hernández-Arauzo et al. (2015). The generation procedure works as follows, where MC denotes the maximum capacity of the machine, $U(a, b)$ refers to a random integer sampled from a uniform distribution in the interval $[a, b]$, and $N(\mu, \sigma)$ denotes a random integer from a normal distribution with mean μ and standard deviation σ:

1. For each operation i, its processing time is set as $p_i = U(20, 100)$. Based on these values, we define $min\_p_i = min\{p_i|i = 1, ..., n\}$ and $sum\_p_i = \sum_{i=1}^{n} p_i$.
2. The initial capacity of the machine is set as $IC = U(1, MC)$, whereas its final capacity is $FC = 2$. Then, the capacity of the machine is defined by different intervals, firstly increasing the capacity one by one from IC to MC, and then decreasing it one by one until FC.
3. The duration of each capacity interval is set as $max\{min\_p_i/4, N(R, 0.2 \times R)\}$, where $R = sum\_p_i/S$ and $S = \sum_{j=IC}^{MC-1} j + \sum_{j=FC}^{MC} j$. This aims at enforcing the operations to be distributed over all the capacity intervals.
4. Finally, for each operation i, its due date is set as $d_i = U(p_i, B)$, where $B = R \times (2 \times MC - IC - 1)$ approximates the completion time of all the operations.

With this procedure, we generated a total of $2,000$ instances with 60 jobs each. In order to avoid trivial instances, in this process we discarded instances for which at least one the rules EDD or ATC with $g \in \{0.25, 0.5, 0.75, 1.0\}$ produced schedules with total tardiness 0. Then these 2,000 instances were sorted by the total tardiness produced by the ATC rule with $g = 0.5$ and splited into two subsets of the same size: training and test. In order for these subsets to have instances with similar characteristics, the training set includes the 1,000 instances with odd indexes and the test set those with even indexes in the above ordering. In turn, the instances of the training set were numbered $0, \ldots, 999$ in the order they appear and the whole training set was divided into 20 subsets $(0 \ldots 19)$ of 50 instances each. Again for all of them being similar, the 20 subsets were defined so as the subset $i \in \{0, \ldots, 19\}$ includes the instances $j \in \{0, \ldots, 999\}$ such that $j\%i = 0$. In the following experiments we only used the subset $i = 10$ for training.

4.1.1 Results from classic rules and memetic algorithm on the test bed

Table 2 summarizes some preliminary results on the training set (50 instances) and the test set (1,000 instances) obtained by different methods, namely the EDD rule, the ATC rule with 10 values of parameter g, an ensemble composed by the 10 ATC rules and finally the best and average results from the memetic algorithm (MA) proposed by Mencía et al. (2019). As expected, the MA obtains the best results, but taking much more time (several orders of magnitude) than the single rules or the ensemble of 10 rules, which in turn obtains much better results that any of the single rules, at the cost of taking just one order of magnitude more time.

4.2 Analysis of the hybrid approach

We consider here the same parameters used in Gil-Gala et al. (2019) and Gil-Gala and Varela (2019) for the GA and the GP respectively. These values are shown in Table 3. The solution sizes in the GA and the GP are the cardinality of the

Method	Training		Test	
EDD	1922.44		1938.55	
ATC(0.1)	1655.18		1675.26	
ATC(0.2)	1654.54		1654.31	
ATC(0.3)	1611.72		1644.26	
ATC(0.4)	1644.64		1651.88	
ATC(0.5)	1661.52		1666.33	
ATC(0.6)	1682.68		1680.85	
ATC(0.7)	1703.52		1701.58	
ATC(0.8)	1714.30		1729.25	
ATC(0.9)	1753.98		1761.33	
ATC(1.0)	1780.26		1796.13	
ATC(avg)	1686.23		1696.12	
Ensemble ATC	1569.24		1578.69	
	Best	Avg.	Best	Avg.
MA	1399.90	1410.90	1408.80	1418.65

Table 2: Summary of results obtained by the rules ATC (with different values of the parameter g), EDD and MA.

Algorithm	Crossover rate	Mutation rate	Population size	Chromosome length
GP	1.0	0.02	200	$2^6 - 1$
GA	0.8	0.2	100	10

Table 3: Parameter values for the GP and the GA taken from Gil-Gala et al. (2019) and Gil-Gala and Varela (2019) respectively.

ensembles (P) and the maximum size of the rules ($\mathcal{S}$), respectively. The maximum depth of the rules ($\mathcal{D}$) is calculated as $\mathcal{S} = 2^{\mathcal{D}} - 1$.

As pointed in Section 3, the GA and the GP may be combined in different ways to obtain ensembles. We have analysed the following three different hybridizations:

1. The GA is executed after the GP finishes and so it evolves ensembles from the whole of different rules evaluated by the GP. In this case, there are a large variety of rules available for the GA.
2. The GA is interspersed with the GP and, after each generation of the GP, the GA evolves ensembles from only the rules in the current generation of the GP. In this case, the GA has much less rules available to be combined into ensembles.
3. A combination of both approaches. The GA is run after each generation of the GP, taking rules from the current population of the GP, and when the GP ends, the GA is executed once more using all the calculated rules.

Table 4 summarizes the results obtained by the ensembles calculated in the three ways considered. We can observe that the results are better when the GA and the GP are interleaved and that a final run of the GA from all rules evaluated by the GP may improve the results. The time taken is similar in the three cases and the small differences are due to the high stochastic nature of the GP. In any case, the time taken by the GA is negligible w.r.t. the time taken by the GP.

	Avg.	Training			Test		
Hybrid.	Time	Best	Avg.	SD	Best	Avg.	SD
1	172.83	1535.50	1574.84	41.50	1566.72	1622.58	46.00
2	174.01	1527.98	1562.97	39.73	1566.62	1613.52	28.40
3	173.48	**1523.84**	**1559.63**	34.64	**1565.68**	**1601.02**	36.96

Table 4: Summary of results obtained by the three hybridizations considered. The average time is given in minutes. The parameters are set as given in Table 3.

		Size	Training			Test		
$\mathcal{D}$	P	Avg.	Best	Avg.	SD	Best	Avg.	SD
4	3	12.86	1624.14	1723.35	48.21	1641.53	1744.36	52.99
4	5	12.93	1627.40	1736.79	57.12	1641.53	1759.67	63.73
4	10	12.71	1621.66	1722.20	57.23	1641.53	1745.08	62.45
4	20	12.79	1644.76	1742.25	45.72	1650.10	1763.91	54.49
4	50	12.89	1624.60	1735.53	52.70	1642.03	1761.78	59.51
6	3	29.18	1603.90	1637.35	32.65	1630.02	1669.09	37.62
6	5	29.36	1603.96	1637.80	39.04	1636.56	1667.48	42.74
6	10	30.39	1600.30	1644.61	40.42	1638.16	1679.03	45.57
6	20	28.68	1600.10	1652.36	42.34	1636.28	1686.71	48.29
6	50	29.64	1605.24	1640.04	42.70	1632.08	1671.55	46.99
8	3	43.36	1592.60	1641.96	49.42	1634.43	1682.69	54.88
8	5	43.89	1591.58	1626.11	31.37	1636.13	1665.60	33.91
8	10	48.00	1587.38	1625.98	27.20	1633.07	1664.65	31.05
8	20	44.96	1597.28	1627.01	32.06	1632.95	1669.53	41.77
8	50	45.79	1592.98	1625.99	31.71	1630.30	1665.51	36.30
10	3	51.00	1596.60	1629.23	34.41	1635.71	1671.23	43.69
10	5	54.46	1589.98	1619.02	25.25	1635.31	1661.04	30.54
10	10	53.93	1591.34	1639.84	43.80	1628.28	1684.76	47.54
10	20	55.25	1591.74	1632.06	40.85	1637.07	1675.87	42.88
10	50	54.75	1595.86	1628.66	29.93	1636.86	1671.57	34.97

Table 5: Detailed results obtained by the rules evolved by the GP from different combinations of $\mathcal{D}$ and P.

4.3 Detailed results from rules and ensembles evolved by the hybrid algorithm

In this section we report detailed results from the hybrid algorithm considering different depth limits ($\mathcal{D}$) of the rules evolved by the GP and also different maximum sizes of the ensembles (P) calculated by the GA. In accordance with the results in Table 4, we considered version 3 of the hybrid algorithm in these experiments.

Table 5 shows the results of the evolved rules on the training and test sets. Each row summarizes the results obtained by the hybrid algorithm for a pair ($\mathcal{D}$,P). Here is important to be aware that the value of P would not have any influence on the results (as we are not considering ensembles, but only individual rules). However, there are some differences between experiments with the same value of $\mathcal{D}$ and different P, which are due to the highly stochastic nature of the GP. In spite of that, we can observe that the quality of the solutions and the average size of the rules are in direct ratio with the value of $\mathcal{D}$, with the exception of the last two values, 8 and 10, which show similar quality of solutions. This may be better observed in Table 6 where the average results are in turn averaged for

	Avg.	Time	Average tardiness	
$\mathcal{D}$	size	(m)	Training	Test
4	12.84	24.80	1732.02	1754.96
6	29.45	90.38	1642.43	1674.77
8	45.20	179.69	**1629.41**	**1669.59**
10	53.88	230.64	1629.76	1672.89

Table 6: Summary of results from the rules evolved by the GP for different values of $\mathcal{D}$ on the training and test sets.

		Training			Test		
$\mathcal{D}$	P	Best	Avg.	SD	Best	Avg.	SD
4	3	1581.34	1675.31	50.27	1604.24	1698.11	52.35
6	3	1563.06	1590.34	31.55	1591.30	1624.98	34.44
8	3	1552.80	1595.47	46.81	1593.44	1634.11	48.33
10	3	1555.34	1585.43	31.55	1593.62	1624.22	35.79
4	5	1570.96	1670.78	52.80	1592.03	1695.37	56.25
6	5	1539.32	1571.81	34.22	1582.42	1607.63	37.74
8	5	1538.16	1563.55	24.86	1581.95	1604.02	28.62
10	5	1537.54	1558.17	22.37	1575.86	1599.17	26.26
4	10	1543.60	1642.16	52.99	1571.83	1668.48	56.73
6	10	1523.84	1559.63	34.64	1565.68	1601.02	36.96
8	10	1517.28	1543.89	23.35	1562.12	1586.52	24.11
10	10	1518.66	1558.99	39.93	1561.23	1605.02	42.03
4	20	1566.20	1652.31	39.33	1579.27	1677.24	42.85
6	20	1511.16	1553.80	38.76	1550.71	1594.34	42.44
8	20	1506.48	1532.95	29.86	1551.74	1576.96	31.19
10	20	1510.92	1540.73	37.11	1552.39	1584.71	37.41
4	50	1548.68	1640.51	45.27	1560.23	1666.75	49.49
6	50	1501.88	1535.23	38.70	1543.75	1573.52	39.85
8	50	1497.18	1521.76	27.27	1537.47	1564.57	29.49
10	50	1491.98	1521.03	23.84	1538.90	1567.06	28.83

Table 7: Detailed results obtained by the ensembles from different combinations of $\mathcal{D}$ and P.

each value of $\mathcal{D}$. Besides, we can see that $\mathcal{D} = 8$ produces the best rules in these experiments.

Table 7 shows detailed results from the ensembles produced in the same experiments. As it may be expected, in this case both $\mathcal{D}$ and P have strong influence on the results. This fact may be better observed in Table 8 and in Table 9 where the average results are also averaged for values of P and $\mathcal{D}$ respectively. From these results we can see that the quality of the solutions grows with the size of the ensemble as well as with the maximum depth of the rules, however in the last case the rate of improvement is lower, even it decreases from 8 to 10, maybe due to the fact that with $\mathcal{D} = 10$ the search space of rules is so huge that the GP can only visit a small portion of it. From all the above, and taking into account the cost of applying ensembles to solve the $(1, Cap(t)||\sum T_i)$ problem, we may consider that $P = 10$ and $\mathcal{D} = 8$ is a reasonable choice.

	Time	Average tardiness	
P	(m)	Training	Test
3	118.50	1611.64	1645.35
5	126.09	1591.08	1626.55
10	132.29	1576.17	1615.26
20	126.49	1569.95	1608.31
50	141.03	**1554.63**	**1592.98**

Table 8: Summary of results from the ensembles generated by the hybrid algorithm averaged for values of P.

	Time	Average tardiness	
$\mathcal{D}$	(m)	Training	Test
4	24.80	1656.22	1681.19
6	90.38	1562.16	1600.30
8	179.69	**1551.52**	**1593.24**
10	230.64	1552.87	1596.04

Table 9: Summary of results from the ensembles generated by the hybrid algorithm averaged for values of $\mathcal{D}$.

4.4 Analysis of the convergence of the hybrid algorithm

In order to gain insights into the hybrid algorithm that may help to improve its performance, it would be interesting to analyse its convergence along the generations. Due to the characteristics of our hybrid algorithm, we have to take into account how both the GP and the GA converge. As the second one is executed once for each generation of the first, in order to have a clear view about how they evolve together, we will visualize the best and average fitness for each generation of the GP, as it is usual, but for the GA we will just visualize the best and average fitness in the final generation for the execution issued from each generation of the GP. Figure 3 shows the results averaged for the 28 executions of the hybrid algorithm; for the sake of clarity, we show the first 50 generations of the GP, on the left, in different scale that the remaining 450, on the right. To strengthen the comparison, we include in both graphics the fitness value produced by the ensemble composed by the best P rules (Best Rules) evolved by the GP so far.

As we can observe, both the GP and the GA present proper convergence patterns these produce better and better rules or ensembles respectively along the generations. The performance of the best solution of the GP tends to be quite similar to that of the ensemble of the best P rules; this is due to the fact that the best rules get more and more similar, not only semantically but even in some cases syntactically as well. Regarding the GA, the evolved ensembles are much better (both the best and in average) than the ensemble of the best P rules. The improvement of the ensembles produced by the GA over the generations is a natural consequence of the GP evolution, which maintains a diversity of rules at the same time that the best and average of these rules improve.

It could also be worth to analyse the time taken by the GP and the GA, and whether or not the GP generates duplicated rules, these issues can be observed in Figure 4. In the left side, we can see that the difference between the total time and

Evolucion_tardinessReglas_c.pdf

Evolucion_tardinessReglas2_c.pdf

Evolucion_tardinessEnsembles_c.pdf

Evolucion_tardinessEnsembles2_c.pdf

Fig. 3: Evolution of the GP and the GA average for the 28 executions of the hybrid algorithm with $\mathcal{D}$=8 and P=10. For GA, only the values of the final population corresponding to each execution after a generation of the GP are considered.

the time taken by the GP, which corresponds to the time taken by the GA, is very small showing that we could leave the GA running for more time between each two consecutive generations of the GP. On the right side, we can observe that the number of rules generated after crossover and mutation by the GP grows linearly with the number of generations. Besides, a large portion of these rules are feasible, but only about half of these feasible rules are non-duplicated. These last results make it clear the utility of duplicate detection.

4.5 Comparison to other methods

Maybe, ATC rules and ensembles of them are the actual competitors of the rules and ensembles evolved by the hybrid algorithm. At the same time, the MA provides a realistic bound limit on the quality of the solutions that could be obtained by

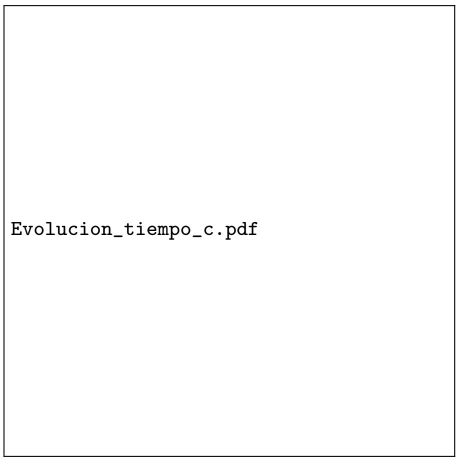

Fig. 4: Time taken and rules produced along the generations averaged for the 28 executions of the hybrid algorithm with $\mathcal{D}$=8 and P=10.

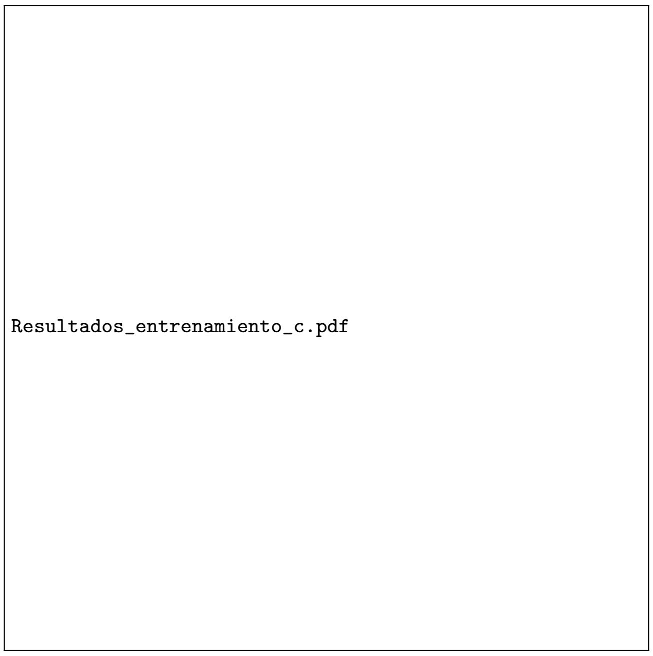

Fig. 5: The average tardiness is reported when solving the **training set** with the best rule and ensemble, and the ensemble formed by the P best rules. The average tardiness obtained by the ATC rule and the ensemble formed by the ATC rules are also reported.

the evolved ensembles. So, in this section we will try to establish a fair comparison with these methods. We will restrict the comparison to $\mathcal{D}$=8 and P=10.

Resultados_test_c.pdf

Fig. 6: The average tardiness is reported when solving the **test set** with the best rule and ensemble, and the ensemble formed by the P best rules. The average tardiness obtained by the ATC rule and the ensemble formed by the ATC rules are also reported.

Figures 5 and 6 summarize the results from rules and ensembles on the training and test set, respectively. Looking at Figure 5, we can see that there is an evident correlation between the best rule and the best evolved ensemble, showing that this rule is dominant in the construction of ensembles. Besides, the best rule and the ensemble of the best P rules is quite similar, which is due to the fact that the best evolved rules tend to be semantically similar. In average, they produce similar results to the best ATC rule, but in general they are clearly worse that the ensemble of ATC rules. On the other hand, the best ensemble evolved by the hybrid algorithm is better to the ensemble of ATC rules, not only in average but also in 22 out of the 28 independent runs of the hybrid algorithm.

Regarding the experiments on the test set (Figure 6), things are rather similar, but there are two exceptions. The first one is that the best rule and the ensemble of the P best rules is sometimes worse than the average ATC rule. But the most remarkable difference is that the evolved ensembles are better than the ensemble of ATC rules in 16 of the 28 runs and in a good number of the remaining 12 runs the evolved ensemble is clearly worse than the ATC ensemble. Therefore showing a certain over-fitting degree and so limited generalization capability, which may be due to the fact that the size of the training set is much lower than that of the test set.

Finally, we summarize in Table 10 the results from the above methods together with the results of the MA. We have to be aware of the differences in the time

	Training		Test	
Method	Best	Avg.	Best	Avg.
ATC rule	1611.72	1686.23	1644.26	1696.12
Rule GP	1587.38	1625.98	1633.07	1664.65
Ensemble ATC	1569.24		1578.69	
Ensemble best rules	1568.64	1622.43	1590.81	1660.31
Ensemble GA	1517.28	1543.89	1562.12	1586.52
MA	1399.90	1410.90	1408.80	1418.65

Table 10: Summary of results obtained by the rules ATC, rules evolved ($\mathcal{D}$=8), ensemble ATC, ensembles evolved (P=10) and MA.

taken: an ensemble takes P times that of a single rule and the MA takes the same time of a single rule to evaluate each chromosome plus the time taken by the local searches. All in all, we can see that the ensembles calculated by the proposed hybrid algorithm are better than those obtained from conventional rules, but at the same time, there is still room to improve given the difference between the solutions produced by the ensembles and the solutions provided by the MA.

5 Conclusions and future work

In this work we have proposed a hybrid algorithm that combines genetic algorithms (GA) and genetic programming (GP) to obtain on-line solvers for the $(1, Cap(t)||\sum T_i)$ problem. The GP is able to evolve priority rules showing superior performance that classic rules as ATC. In turn, the GA calculates ensembles from the rules in each generation of the GP. All these rules are exploited in parallel and the best solution from all rules taken as the solution produced by the ensemble. This schema may produce much better solutions than exploiting a single rule, provided that the rules in the ensemble have enough semantic diversity. Of course, this improvement comes at the cost of increasing the time taken in a factor proportional to the number of rules. Furthermore, the calculated ensembles outperform ensembles formed by classical rules, but their performance is still far from that of off-line algorithms such as memetic algorithms.

According to the taxonomy proposed in Burke et al. (2019), our approach can be classified as hyper-heuristic based on *heuristic generation*, due the fact that the ensembles do not use the rules in a coordinated way to build a single solution, as traditional hyper-heuristic based on *heuristic selection* do.

As future lines of research we will consider alternative schemas to exploit ensembles to build a single solution, as done in Hart and Sim (2016), Park et al. (2018) and Durasević and Jakobović (2018, 2019), and mechanisms for local improvement of the solutions reached by the GA and the GP. Furthermore, in order to dealing with some issues as the size of the rules or semantically duplicated rules, we plan to study alternative evolution methods and postprocessing mechanisms to simplify the structure of the evolved expression trees.

Acknowledgements This research has been supported by the Spanish Government under research project TIN2016-79190-R and by the Principality of Asturias under grant

IDI/2018/000176. Francisco J. Gil-Gala is supported by the scholarship FPI17 / BES-2017-08203.

References

Burke EK, Hyde MR, Kendall G, Woodward J (2012) Automating the packing heuristic design process with genetic programming. Evolutionary Computation 20(1):63–89

Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward JR (2019) A Classification of Hyper-Heuristic Approaches: Revisited. In: Gendreau M., Potvin JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science 272:453–477.

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

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

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(1):53–92

Durasević M, Jakobović D (2019) Creating dispatching rules by simple ensemble combination. Journal of Heuristics 25:959–1013

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

Gil-Gala FJ, 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. et al. (eds) Bioinspired Systems and Biomedical Applications to Machine Learning. Proceedings of IWINAC 2019.Lecture Notes in Computer Science 11487:223–233.

Gil-Gala FJ, Mencía C, Sierra MR, 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.

González MA, Vela CR, Varela R (2012) A competent memetic algorithm for complex scheduling. Natural Computing 11:151–160

Hart E, Sim K (2016) A hyper-heuristic ensemble method for static job-shop scheduling. Evolutionary Computation 24(4):609–635

Hernández-Arauzo A, Puente J, Varela R, Sedano J (2015) Electric vehicle charging under power and balance constraints as dynamic scheduling. Computers & Industrial Engineering 85:306–315

Ingimundardottir H, Runarsson TP (2018) Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. Journal of Scheduling 21(4):413–428

Kaplan S, Rabadi G (2012) Exact and heuristic algorithms for the aerial refueling parallel machine scheduling problem with due date-to-deadline window and ready times. Computers & Industrial Engineering 62(1):276–285

Koulamas C (1994) The total tardiness problem: Review and extensions. Operations Research 42:1025–1041

Koza JR (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press

Mencía C, Sierra MR, Mencía R, Varela R (2019) Evolutionary one-machine scheduling in the context of electric vehicles charging. Integrated Computer-Aided Engineering 26(1):1–15

Mencía R, Sierra MR, Mencía C, Varela R (2014) A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing. Natural Computing 13:179–192

Nguyen S, Mei Y, Xue B, Zhang M (2019) A hybrid genetic programming algorithm for automated design of dispatching rules. Evolutionary Computation 27(3):467–496

Park J, Nguyen S, Zhang M, Johnston M (2015) Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In: Machado P. et al. (eds) Genetic Programming. Proceedings of EuroGP 2015. Lecture Notes in Computer Science 9025:92–104

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

Sang-Oh Shim SO, Kim YD (2007) Scheduling on parallel identical machines to minimize total tardiness. European Journal of Operational Research 177(1):135–146

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1):67–82