Genetic programming to evolve priority rules for on-line scheduling on single machine with variable capacity

Francisco Gil-Gala, Carlos Mencía, María R. Sierra, Ramiro Varela
Department of Computer Science, University of Oviedo,
Campus of Gijón, Gijón 33204, Spain
{*giljavier, menciacarlos, sierramaria, ramiro*}*@uniovi.es*
http://www.di.uniovi.es/iscop

***Abstract*—On-line scheduling is often required in a number of real-life settings. This is the case of distributing charging times for a large fleet of electric vehicles arriving stochastically to a charging station under power constraints. In this paper, we consider a scheduling problem derived from a situation of this type: one machine scheduling with variable capacity and tardiness minimization, denoted $(1, Cap(t)||\sum T_i)$. The goal is to develop new priority rules to improve the results from some classical ones as Earliest Due Date (EDD) or Apparent Tardiness Cost (ATC). To this end, we developed a Genetic Programming (GP) approach. We conducted an experimental study showing that it is possible to evolve new efficient rules that outperform ATC and EDD using the same problem attributes and operations.**

***Index Terms*—Scheduling, One machine scheduling, Priority Rules, Genetic Programming, Hyperheuristics, Electric Vehicle Charging Scheduling**

I. Introduction

One machine scheduling problems have attracted an ever increasing body of research over the last decades, due to both their usual high computational complexity as well as for acting as building blocks in the development of solutions to more complex scheduling problems. This paper focuses on a problem in this class in which 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 was introduced in [6] in the context of scheduling the charging times of a large fleet of Electric Vehicles (EVs), and it is denoted $(1, Cap(t)||\sum T_i)$.

Solving the Electric Vehicle Charging Scheduling Problem (EVCSP) tackled in [6] amounts to solving a number of instances of the $(1, Cap(t)||\sum T_i)$ problem. Due to the computational intractability of this problem and the tight real-time requirements of the EVCSP, *on-line scheduling* represents the most (if not the only) suitable approach to the $(1, Cap(t)||\sum T_i)$ problem. In [6], it is solved by means of the *Apparent Tardiness Cost* (ATC) priority rule, commonly used in the context of scheduling with tardiness objectives.

The aim of this paper is the automated development of new, efficient, priority rules specifically adapted to address the $(1, Cap(t)||\sum T_i)$ problem. A natural way to cope with this task is the use of hyper-heuristics, as search needs to be conducted in a space of heuristics rather than in a space of solutions to the scheduling problem. Since priority rules are arithmetic expressions that can be naturally represented by trees, we opted to investigate a Genetic Programming (GP) approach, which is proposed in this paper. Experimental results indicate that GP is capable of evolving effective priority rules for the $(1, Cap(t)||\sum T_i)$ problem, outperforming ATC and other classical priority rules. The results also provide insights of practical interest that motivate further research.

The remainder of the paper is organized as follows. Section II reviews some GP approaches to evolve priority rules for scheduling problems. In Section III, we describe the EVCSP and show how the solving procedure proposed in [6] decomposes an instance of this problem into a number of instances of the $(1, Cap(t)||\sum T_i)$ problem. In Section IV, we give the formal definition of the $(1, Cap(t)||\sum T_i)$ problem. Section V introduces a schedule builder for this problem and describes its main properties. In Section VI, we present some classical priority rules used to solve the problem. Section VII describes the GP approach propose to evolve new priority rules. In Section VIII, we report the results of the experimental study conducted to evaluate the proposed GP approach. Finally, in Section IX we summarize the main conclusions and outline some ideas for future work.

II. Evolving priority rules for scheduling problems

The terms *Dispatching Rule* (DR) and *Priority Rule* (PR) are commonly used in the scheduling literature to refer to "a simple heuristic that derives a priority index of a job from its attributes" [1]. Due to their low computational cost, PRs are well suited for *on-line scheduling*: the job with the highest priority among those available at a given time is scheduled next. In this section, we review some existing GP approaches proposed to discovering dispatching or priority rules for scheduling problems, such as job shop (JSSP), one machine or unrelated parallel machines scheduling problems, among others. In some cases, the purpose is just to find a *good* priority rule which is then embodied into a schedule builder.

Other works notice that a single rule may not suffice and focus on finding sets of rules to be applied collaboratively to solve instances with different characteristics.

Branke, Schols-Reiter and Hildeblant analyze in [1] three representation models for priority rules for the dynamic JSSP: expression trees commonly used in GP, Artificial Neural Networks (ANNs) and weighted linear combination of job properties. Their results show that expression trees evolved by GP perform slightly better than the other approaches.

In [5], the authors propose evolving sets of rules that are used collaboratively to solve problems. They use GP to evolve a set of PRs for the static JSSP. They consider single and composite dispatching rules as terminal nodes, as for example SPT or ATC, in addition to some parameters. The rules are used in combination with various schedule builders, as for example the well-known Giffler and Thompson algorithm [4]. These rules are sequenced into heuristics. To produce a solution to the problem, each rule in the heuristic is applied in turn to schedule a single operation.

In [8], the authors consider composite PRs for the JSSP given by linear combinations of 16 problem features, as for example total remaining work for a job or total idle time for all machines. The weights in the linear function are learned from a set of optimal solutions obtained by a MILP solver. Preference and imitation learning were used for this purpose.

In [13], GP is used to learn DRs for the Order Acceptance and Scheduling problem (OAS) directly from optimal scheduling decisions. Instead of evolving just a single rule, a set of rules is evolved that is used in a Forward Construction Heuristic (FCH): at each step the rule that produces the best local improvement is applied. One of the novelties of this model is that the fitness of a rule depends on how well the rule performs at each decision point (i.e., whether or not it takes the optimal decision) rather than the final objective values of the schedule.

In [3], the authors consider on-line scheduling for multiple unrelated parallel machines. They also propose evolving new priority rules with GP, incorporating some enhancements as dimension awareness to guarantee semantically correct rules and some GP variant as gene expression.

In [2], PRs are evolved by GP for the Resource Constrained Project Scheduling Problem (RCPSP) that outperform many of the existing ones for this problem.

III. Where the $(1, Cap(t)||\sum T_i)$ problem comes from

As pointed out, the $(1, Cap(t)||\sum T_i)$ comes from the EV Charging Scheduling Problem (EVCSP) considered in [6]. In turn, the EVCSP is motivated by the charging station designed in [15] to be installed in a community park where each user has its own space. Figure 1 shows the general structure and the main components of this charging station. Each space has a charging point which is connected to one of the three lines of a three phase feeder. The system is controlled by a central server and a number of masters and slaves. Each slave takes control of two charging points and each master controls up to eight slaves in the same line. The control system registers events as EVs arrivals and sends activation/deactivation signals to the charging points in accordance with a schedule.

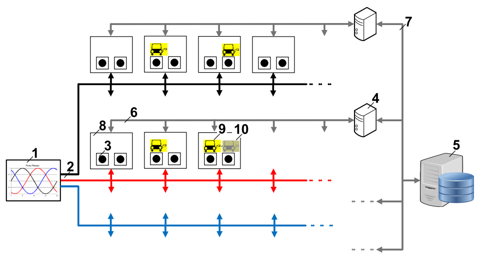


Fig. 1. General structure of the charging station. (1) Three-phase electric power 400v AC, (2) lines, (3) charging points Type 2/AC IEC 62196-2 with V2G communication interface ISO 15118, (4) masters, (5) server, (6) communication Rs 485, (7) communication TCP/IP, (8) slaves, (9) active vehicles, (10) inactive vehicles.

Due to the EVs arrivals being not known in advance, the EVCSP is dynamic and so schedules must be computed at different points over time. Furthermore, the physical characteristics and the operating mode of the charging station impose some restrictions to the EVCSP that make it hard to solve. In particular, the contracted power is limited and so there is a maximum load in each line. Besides, the load in the three lines must be similar to avoid an excessive imbalance among the three phases. Here, we assume two simplifications of the model: (1) the contracted power is constant over time, and (2) the EVs charge at constant rate in the so called Mode 1 in accordance with the regulation UNE-EN 61851-1 [7]. Therefore, there is a maximum number N of EVs that can be charging in each line simultaneously.

Figure 2 shows a feasible schedule for the situation represented in Figure 1; dark bars represent the EVs that are charging at time T_k and light bars represent EVs that are scheduled at a later time. In this example, we consider that the maximum number of active EVs in a line is 4 and that the maximum difference in the number of active EVs in every two lines is 2. For these reasons, none of the tasks 12 and 13 can be scheduled at T_k because if some of them were scheduled at T_k, lines 2 and 3 would be imbalanced after completion of task 8, as there would be 4 EVs charging in line 2 and only one (number 9) charging in line 3, so exceeding the maximum difference of 2. The schedule built at T_k allows the EVs in the system to complete their charging periods without violating the constraints of the system. However, if new EVs arrive the charging station after T_k, a new schedule must be built to accommodate them.

To solve the EVCSP, in [6] the authors proposed an algorithm that considers at each scheduling time T_k the active EVs in each line (which cannot be rescheduled), the demanding EVs (which have not yet started to charge), the maximum number of active EVs in a line, N, and a profile of maximum load in each line $N_i^{max}(t), i = 1, 2, 3$, which is iteratively adapted to keep the imbalance among the lines under control. The objective is to schedule all the EVs in the three lines such that all the constraints are satisfied and the total tardiness, i.e., the delay w.r.t. to the times the users want to take their

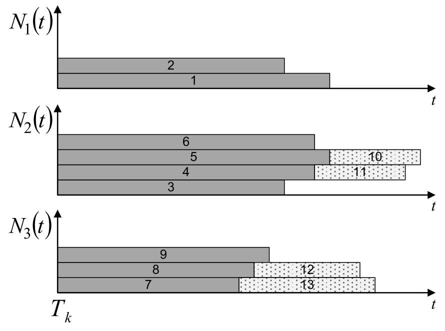


Fig. 2. A feasible schedule for the problem in Figure 1. Tasks 1 - 10 are the active EVs at time T_k in the lines 1, 2 and 3 respectively, while tasks 10 - 13 correspond to inactive EVs in the lines 2 and 3, which could be rescheduled if it were neccesary to accomodate new EVs arriving after T_k.

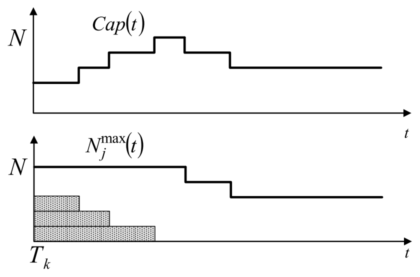
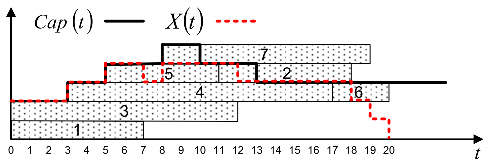
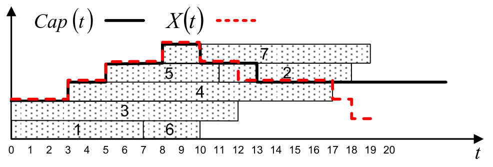


Fig. 3. Definition of the capacity of the machine $Cap(t)$ from the maximum profile $N_j^{max}(t)$ and the active EVs at time T_k.



(a) Non left-shifted schedule



(b) Left-shifted schedule

Fig. 4. Two feasible schedules for an instance of the $(1, Cap(t)||\sum T_i)$ problem with 7 jobs and a machine with capacity varying between 2 and 5.

EVs away, is minimized. If two of the obtained schedules are imbalanced at some time point, some of the maximum profiles $N_i(t)$ must be recalculated and a new schedule obtained for the line i. The details of this process are given in [6].

Therefore, scheduling the EVs in each line, subject to the maximum load and taking into account the active EVs, may be viewed as the problem of scheduling a set of jobs on a machine with variable capacity over time. The calculation of the capacity of the machine from the active EVs and the maximum load profile is illustrated in Figure 3. In this example, we consider that at the scheduling time T_k there are three EVs charging in line j as they were scheduled before T_k, and that the maximum load of line j, $N_j^{max}(t)$, undergone two adjustments due to lower load in the other two lines. So, the capacity to accommodate new charging EVs, $Cap(t)$, for $t \geq T_k$, is variable as shown in the upper part of the figure.

IV. Problem Definition

The $(1, Cap(t)||\sum T_i)$ problem may be defined as follows. We are given a number of n jobs $\{1, \ldots, n\}$, all of them available at time $t = 0$, which have to be scheduled on a machine whose capacity varies over time, such that $Cap(t) \geq 0, t \geq 0$, is the capacity of the machine in the interval $[t, t+1)$. 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 following constraints are satisfied:

i. 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.,

$$X(t) \leq Cap(t). \tag{1}$$

ii. The processing of jobs on the machine cannot be preempted; i.e.,

$$C_i = st_i + p_i, \tag{2}$$

where C_i is the completion time of job i.

The objective function is the total tardiness, defined as:

$$\sum_{i=1,\ldots,n} \max(0, C_i - d_i) \tag{3}$$

which should be minimized.

Figure 4 shows an example of two feasible schedules for a problem with 7 jobs; the capacity of the machine varies between 2 and 5 over time. Due dates are not represented for the sake of clarity. As we can observe, in both schedules $X(t) \leq Cap(t)$ for all $t \geq 0$.

One particular case of this problem is when the capacity of the machine is constant over time. This is the parallel identical machines problem [10], denoted $(P||\sum T_i)$, which is NP-hard. Thus, it follows that the $(1, Cap(t)||\sum T_i)$ problem is NP-hard as well.

V. Schedule Builder

Schedule builders constitute an essential component for designing efficient scheduling algorithms. Also known as *schedule generation schemes*, these methods provide a way for computing and enumerating a subset of the feasible schedules, thus enabling the definition of a search space to look for solutions to the problem. We use here the schedule builder proposed in [12], which produces *left-shifted schedules*, in which no job can be scheduled earlier without changing the starting time of some other job. Figure 4 shows two feasible schedules for a problem instance with 7 jobs, one is not left-shifted (a), while the other one is left-shifted (b).

The schedule builder is depicted in Algorithm 1; it maintains a set US with the unscheduled jobs, as well as the consumed capacity $X(t)$ due to the jobs scheduled so far. US is initialized with all the jobs. In each iteration, the algorithm

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 $S = (st_1, st_2, ..., st_n)$;
```

selects one unscheduled job among the ones that can start at the earliest time $\gamma(\alpha)$.

Note that the selection of a job to be scheduled at each iteration is non-deterministic. Regardless of this, we can guarantee that the application of Algorithm 1 always results in a feasible left-shifted schedule; for example, the sequence of choices $(1, 3, 4, 5, 6, 7, 2)$ would lead to building the schedule in Figure 4(b). Furthermore, any left-shifted schedule may be obtained considering the appropriate choice in each iteration. In other words, the scheduler searches in the whole space of left-shifted schedules, which is *dominant*; i.e., it contains at least one optimal schedule.

The schedule builder may be instantiated by using any priority rule or heuristic, as we will see in the next section. In [12] it was embedded as a decoder in a genetic algorithm.

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

A schedule builder, as the one shown in Algorithm 1, may be used in combination with some priority rule to make the non-deterministic choice in each iteration: the job having the highest priority is chosen to be scheduled. This paradigm is called *priority scheduling*, which is particularly appropriate for *on-line scheduling*, where decisions must be made quickly. In the literature there are a number of rules that could be adapted to the $(1, Cap(t)||\sum T_i)$ problem. Among the simplest ones, we may consider *Earliest Due Date* (EDD) or *Shortest Processing Time* (SPT); the first one picks the operation with the smallest due date, while SPT selects the one with the least duration. These two rules are often used for objective functions that are non decreasing with the completion time of the jobs, as for example the makespan, the lateness or even the tardiness. As they are quite simple rules, it often happens that they produce rather moderate results. In contrast, more sophisticated rules are usually able to produce (much) better results as they take into account more knowledge on the problem. This is the case of the *Apparent Tardiness Cost* (ATC) rule, which was used with success to solve some scheduling problems with tardiness objectives (e.g. [14], [9]); with this rule, the priority of each job $j \in US$ is give by Equation (4).

$$\pi_j = \frac{1}{p_j} exp\left[\frac{-max(0, d_j - \gamma(\alpha) - p_j)}{g\bar{p}}\right] \tag{4}$$

TABLE I
Summary of results from [12]. Average total tardiness.

			ATC				GA		
n	EDD	SPT	0.25	0.5	0.75	1.0	Best	Avg	Time(s)
15	8.28	14.03	7.35	7.17	7.27	7.50	6.53	6.53	13.13
30	26.56	55.84	19.36	18.93	18.81	19.09	17.73	17.76	21.49
45	46.52	137.74	36.25	36.20	35.83	36.75	33.31	33.45	30.22
60	131.31	262.54	90.59	89.86	89.03	89.23	86.84	87.19	38.48
Avg	53.17	117.93	38.39	38.04	37.74	38.14	36.10	36.23	25.83

In Equation (4), $\bar{p}$ is the average processing time of the jobs in US and g is a look-ahead parameter to be introduced by the user. As we can see, the ATC rule combines the information exploited by SPT and EDD as the priority of a job i is in inverse ratio with its duration p_i and it is decreasing with the slack time to its due date $d_j - \gamma(\alpha) - p_j$.

Table I reproduces some results reported in [12] obtained by the rules EDD, SPT and ATC with four values of the parameter g, and by a genetic algorithm proposed therein (GA), over a set of 120 instances distributed in four sets having different number of jobs (15, 30, 45, 60) with 30 instances each. As we can see, ATC produces much better results than both EDD and SPT, the results of the latter being actually poor, as can be expected due to the fact that this rule does not consider any information related to the tardiness objective. Besides, the performance of ATC depends on the value of the parameter g; the best value of g depending on the the size of the instances n. Furthermore, the ATC rule produces worse results than the GA, which of course takes much longer time than the priority rules. These facts lead us to formulate the following hypotheses:

1) The ATC rule may be outperformed by new rules having a different structure or more detailed information of the problem domain, or just considering other parameters.
2) Given a benchmark containing instances with a similar structure, there may exist priority rules that are well adapted to this particular benchmark.

VII. Evolving new priority rules with Genetic Programming

From the hypotheses above, our purpose is to devise new dispatching rules for the $(1, Cap(t)||\sum T_i)$ problem. To this end, we propose using hyper-heuristics, as they provide a natural way of searching over a (sub)space of the heuristics that solve a given problem. As we are interested in devising some arithmetic expression, as that of the ATC rule given in Equation (4), Genetic Programming (GP) [11] is a good choice as it provides a way of evolving tree structures.

The first step in the design of a GP solution is selecting the sets of terminal and function nodes of the candidate trees. Terminal symbols represent the elementary properties that are considered relevant to establish jobs' priorities as, for example, processing times, due dates, etc., as well as some constants. Function symbols are the elementary arithmetic operations and some other unary and binary functions.

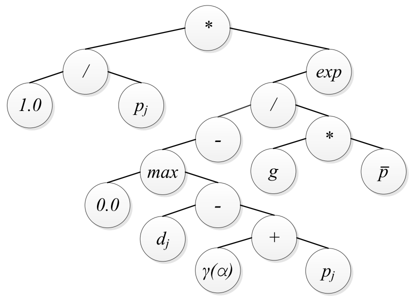


Fig. 5. Tree representing the ATC rule.

Looking at conventional rules as ATC, EDD or SPT, we have chosen the functional and terminal sets of symbols showed in Table II. The considered grammar generates any feasible expression in accordance with the arithmetic rules, without any other restrictions. For this reason, we could have inconsistencies as division by 0. To dealing with that, we make use of the EDD rule in the following way: when a rule produces a division by 0 for some job, we assume that this rule does not decide anything for that job. Then, we consider the job having the largest priority together with all jobs for which the rule did not decide and apply to these jobs the EDD rule. This way, a rule producing many divisions by 0 gets penalized as many decisions are taken by using EDD. Figure 5 shows the tree representing the ATC rule.

We remark that the set of terminal symbols only include information regarding jobs, but no information about the machine, such as its capacity $Cap(t)$, which is relevant in this problem. The main reason to not include it is that in a first stage we plan to analyze the extent to which it is possible to improve the conventional rules without considering new information. Besides, including new terminal symbols augments the size of the search space.

TABLE II
FUNCTIONAL AND TERMINAL SETS USED TO BUILD EXPRESSION TREES. SYMBOL "-" IS CONSIDERED IN UNARY AND BINARY VERSIONS.

Function	-	+	/	*	pow	max	min	exp	
Terminal	p_i	d_i	$\gamma(\alpha)$	$\bar{p}$	0.00	0.01	...	0.99	1.0

We used a rather conventional GP as proposed in [11] with only some small changes.

The evaluation of chromosomes is the most time consuming component of the algorithm and consists in solving a battery of instances of the $(1, Cap(t)||\sum T_i)$ problem. The fitness of each individual being the inverse of the average cost (total tardiness) of the solutions obtained. Therefore, if the instances had a similar structure, for example if their data were generated from the same probability distributions, we could expect the GP to evolve rules well adapted to those instances.

The evolutionary schema used in our approach is generational, with a selection phase in which all chromosomes are organized into pairs and tournament replacement is done between every two parents and their two offsprings. The strategy is combined with elitism.

TABLE III
GP PARAMETERS' SETTING.

Cross., Mut., Init. pop. gen.	Standard [11]
Cross. and Mutation ratio	0.8 and 0.02 resp.
Population size	500
Number of generations	500
Max. init. chrom. depth	6
Max. chromosome size	32
Elitism	1
Number of runs	30

To control the size of chromosomes, we limit the size of the initial candidate solutions and also after crossover. In this case, if an offspring exceeded the size limit, it is discarded and the parents are mated again choosing different points until some valid offspring is reached.

VIII. Experimental Study

We have conducted an experimental study aimed at assessing the quality of the rules obtanied by the proposed approach (GP). To this aim, we implemented a prototype in Java, and ran a series of experiments on a Linux cluster (Intel Xeon 2.26 GHz. 128 GB RAM).

The experiments were carried out over a benchmark set of 2000 instances, generated by means of the procedure introduced in [12]. Each instance is characterized by the number of jobs (n) and the maximum capacity of the machine (MC). Given fixed n and MC, a random instance is generated using uniform distributions as follows (all sampled values are integers):

1) Each job $i \in \mathcal{J} = \{1, ..., n\}$ is assigned a random processing time $p_i \in \{1, ..., 100\}$.
2) Once all jobs have a processing time, they are assigned a random due date $d_i \in [p_i, \max(p_i + 2, \sum_{j \in \mathcal{J}} p_j/2)]$.
3) The capacity of the machine ($Cap(t)$) is generated as a unimodal function, with each constant interval taking a random duration in the range $[1, \sum p_j/MC]$. Both the initial and the final capacity of the machine is a random integer in $\{1, 2\}$.

This procedure aims at avoiding the generation of underconstrained instances, which can be easily solved. All the 2000 instances considered in this experimental study have been generated with $n = 60$ and $MC = 10$.

We ran GP considering a training subset of 50 instances to evaluate each candidate solution and used the remaining 1950 instances for testing. The parameters of GP are given in Table III. These parameters were chosen from a large set of preliminary experiments. In these experiments we ran GP 30 times and recorded the best priority rule evolved in each run. Then, we report the average results of the 30 rules and the results from the best and worst rules (over the training and test sets). These results are summarized in Table IV, which includes the results from EDD, ATC (with $g \in \{0.25, 0.5, 0.75, 1.0\}$) and the solutions by the genetic algorithm from [12] (GA).

The results over the test set show that the best priority rule obtained in the training phase (which in this case is the second

TABLE IV
SUMMARY OF RESULTS. AVERAGE VALUES OF THE TOTAL TARDINESS OBTAINED FOR THE TRAINING AND TEST SETS BY THE BEST RULES EVOLVED IN EACH ONE OF THE 30 RUNS OF GP, ATC, EDD AND GA.

Priority Rules	Train.	Test.
Best(in training)	636.0	548.0
Average(30 rules)	640.4	553.1
Worst(in training)	678.0	580.0
ATC(g=0.25)	660.2	565.0
ATC(g=0.50)	650.9	552.8
ATC(g=0.75)	650.0	557.1
ATC(g=1.00)	659.0	570.2
ATC(avg. 4 g-values)	655.0	561.3
EDD	833.3	695.7
GA	616.1	520.4

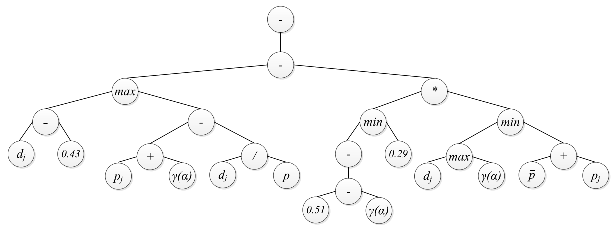


Fig. 6. Tree representing the best rule obtained.

best performing rule on the test set) performs better than the four versions of the ATC rule. In addition, the average value of the 30 evolved rules outperforms the average value of the four ATC rules. We conducted Wilcoxon paired tests confirming statistically significant differences in favor of the new best and average results obtained by the proposed GP approach. The worst performing rule evolved by GP lags behind the ATC rules, although it yields better average total tardiness than the EDD rule, which performs worst overall. The GA produces the best results, at the expense of taking much longer time (note that priority rules take negligible time).

A close look at the actual priority rules generated reveal some interesting insights. Figure 6 shows the tree representation of the best priority rule obtained in the experiments. Interestingly, we can observe that this rule contains some odd expressions, such as $(d_j - 0.43)$ on the left-most part of the tree. This may be due to using a grammar allowing for any valid arithmetic expression, and motivates further research, e.g., using a restricted grammar instead to deal with these situations.

IX. CONCLUSIONS

This paper studies the one machine scheduling problem with variable capacity, denoted $(1, Cap(t)||\sum T_i)$, and shows that Genetic Programming is a suitable approach to generate new priority rules, improving the best-performing classical ones for total tardiness minimization such as EDD and ATC. In order to make a fair comparison, we considered the same problem attributes and operations as in these rules. At the same time, we have seen that there is still room for improvement, as a genetic algorithm running for much longer time than a schedule builder guided by priority rules is able to obtain even better solutions. Therefore, we conjecture that by using more attributes of the problem, in particular some related to the capacity of the machine, better rules may be evolved. Besides, it seems clear that no single rule can be the best one in every problem instance. Therefore, it may be more appropriate to try to evolve sets of rules to cover different subsets of instances, or to take decisions at different stages in the schedule construction. These are some promising lines of research we plan to explore in the future.

ACKNOWLEDGEMENTS

This research has been supported by the Spanish Government under research project TIN2016-79190-R.

REFERENCES

[1] Branke, J., Hildebrandt, T., Scholz-Reiter, B.: Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. Evolutionary Computation 23(2), 249–277 (2015)
[2] Chand, S., Huynh, Q., Singh, H., Ray, T., Wagner, M.: On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems. Information Sciences 432, 146 – 163 (2018)
[3] Durasevic, M., Jakobovi, D., Kneevi, K.: Adaptive scheduling on unrelated machines with genetic programming. Applied Soft Computing 48, 419 – 430 (2016)
[4] Giffler, B., Thompson, G.L.: Algorithms for solving production scheduling problems. Operations Research 8, 487–503 (1960)
[5] Hart, E., Sim, K.: A hyper-heuristic ensemble method for static job-shop scheduling. Evolutionary Computation 24(4), 609–635 (2016)
[6] Hernández-Arauzo, A., Puente, J., Varela, R., Sedano, J.: Electric vehicle charging under power and balance constraints as dynamic scheduling. Computers & Industrial Engineering 85, 306 – 315 (2015)
[7] (IEC), T.I.E.C.: Electric Vehicle Conductive Charging System?Part 1: General Requirement; IEC 61851-1. London, UK (2001)
[8] Ingimundardottir, H., Runarsson, T.P.: Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. Journal of Scheduling First Online: 13 June 2017 (2017)
[9] Kaplan, S., Rabadi, G.: 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 (2012)
[10] Koulamas, C.: The total tardiness problem: Review and extensions. Operations Research 42, 1025–1041 (1994)
[11] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
[12] Mencía, C., Sierra, M.R., Mencía, R., Varela, R.: Genetic algorithm for scheduling charging times of electric vehicles subject to time dependent power availability. In: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Toledo Moreo, J., Adeli, H. (eds.) Natural and Artificial Computation for Biomedicine and Neuroscience. pp. 160–169. Springer International Publishing, Cham (2017)
[13] Nguyen, S., Zhang, M., Johnston, M.: A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014 (2014)
[14] Sang-Oh Shim, S.O., Kim, Y.D.: Scheduling on parallel identical machines to minimize total tardiness. European Journal of Operational Research 177(1), 135–146 (2007)
[15] Sedano, J., Portal, M., Hernández-Arauzo, A., Villar, J.R., Puente, J., Varela, R.: Intelligent system for electric vehicle charging: Design and operation. DYNA 88(6), 640–647 (2013)