Genetic Algorithm to evolve ensembles of rules for on-line scheduling on single machine with variable capacity

Francisco J. Gil-Gala and Ramiro Varela

Department of Computer Science,
University of Oviedo, Campus of Gijón, Gijón 33204, Spain
{giljavier,ramiro}@uniovi.es
http://www.di.uniovi.es/iscop

Abstract. On-line scheduling is often required in real life situations. This is the case of the one machine scheduling with variable capacity and tardiness minimization problem, denoted $(1, Cap(t)||\sum T_i)$. This problem arose from a charging station where the charging periods for large fleets of electric vehicles (EV) must be scheduled under limited power and other technological constraints. The control system of the charging station requires solving many instances of this problem on-line. The characteristics of these instances being strongly dependent on the load and restrictions of the charging station at a given time. In this paper, the goal is to evolve small ensembles of priority rules such that for any instance of the problem at least one of the rules in the ensemble has high chance to produce a good solution. To do that, we propose a Genetic Algorithm (GA) that evolves ensembles of rules from a large set of rules previously calculated by a Genetic Program (GP). We conducted an experimental study showing that the GA is able to evolve ensembles or rules covering instances with different characteristics and so they outperform ensembles of both classic priority rules and the best rules obtained by GP.

Keywords: One machine scheduling, Priority Rules, Ensembles of Rules, Genetic Algorithm, Hyperheuristics, Electric Vehicle Charging Scheduling

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 [5]. Indeed, solving this problem requires solving many instances of the $(1, Cap(t)||\sum T_i)$ problem on-line.

Priority rules are of common use in on-line scheduling. For example, in [5], the $(1, Cap(t)||\sum T_i)$ problem is solved by means of the *Apparent Tardiness*

Cost (ATC) priority rule. Priority rules can be defined manually by experts on the problem domain, as it is the case of the ATC rule [8], although it is clear that automatic methods could capture some characteristics of the scheduling problem that are not clear to human experts. For this purpose, hyper-heuristics as Genetic Programming (GP) are a good choice.

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. For this reason, some researchers focus in calculating sets of rules that collaboratively solve the problem. This may be done in different ways. For example, in [4] the authors propose GP to obtain a set of rules that are applied in turn to schedule a single operation. In this paper, we take 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 [5], we propose to use a set of priority rules in parallel to obtain a number of solutions. Our working hypotheses 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.

So, it is our goal to devise ensembles of rules from a large pool of rules obtained by other method; in our study, we will consider rules evolved by the GP proposed in [3]. To this end, we propose to use a Genetic Algorithm (GA). The results produced by the evolved ensembles are compared with those from the best on-line and off-line methods in the literature to solve the $(1, Cap(t)||\sum T_i)$ problem. As far as we known, these methods are the GP proposed in [3] and the genetic algorithm proposed in [9]. The results of the experimental study show that the solutions obtained from the evolved ensembles are better that those produced by the sets of best rules obtained in [3] and that these solutions are actually close to those obtained off-line in [9].

The remainder of the paper is organized as follows. In the next section we give a formal definition of the $(1, Cap(t)||\sum T_i)$ problem. Section 2 introduces the solving method used for the $(1, Cap(t)||\sum T_i)$ problem on-line, which consists of two main components: schedule builder and priority rules. In Section 3 we present the GA devised to evolve ensembles of rules. 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.

1.1 Problem Definition

The $(1, Cap(t)||\sum T_i)$ problem may be defined as follows. We are given a number of n jobs $\{1, \ldots, n\}$, 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.,

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

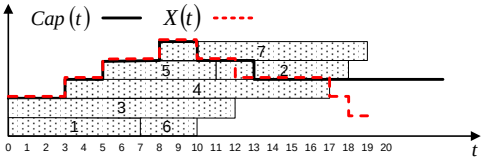

Fig. 1. One schedule for an instance of the $(1, Cap(t)||\sum T_i)$ problem with 7 tasks and a machine with capacity varying between 2 and 5.

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 1 shows a schedule 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, $X(t) \leq Cap(t)$ for all $t \geq 0$.

One particular case of this problem is the parallel identical machines problem [8], denoted $(P||\sum T_i)$, which is NP-hard. Thus, the $(1, Cap(t)||\sum T_i)$ problem is NP-hard as well.

2 Solving Method

Given the real time requirements of the EVCSP, to solve the $(1, Cap(t)||\sum T_i)$ problem, we consider a schedule builder guided by priority rules.

2.1 Schedule builder

We use here the schedule builder proposed in [9], which produces *left-shifted schedules*, a subspace of feasible schedules that includes at least one optimal solution. This 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. 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 of these jobs non-deterministically to be scheduled. For example, the sequence of choices $(1, 3, 4, 5, 6, 7, 2)$ would lead to building the schedule in Figure1(b). The schedule builder may be instantiated by using any priority rule or heuristic, as we will see in the next section. Besides, it could be embedded as a decoder in a genetic algorithm, as done in [9].

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

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

The non-deterministic choice in each iteration of Algorithm 1 may be done with a priodrity so that the job having the highest priority in US^* is chosen to be scheduled. 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 (e.g. [10, 7]); with this rule, the priority of each job $j \in US^*$ is given by

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

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. 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.

As pointed, an alternative to the priority rules defined by experts are rules created automatically for systems as GP [4, 6, 2, 1, 3]. These systems are used to evolve priority rules that are trained to solve a set of given instances for different scheduling problem, and then evaluated over a set of unseen instances. In this way, they are able to evolve rules that are good in average on the training set, but the best of these rules may be not so good for each single instance.

2.3 Considering ensembles of rules and some previous results

From the above observations, it seems clear that it is not easy to evolve a rule that may generalize well over a large set of unseen instances so that it can compete with off-line algorithms such as GA introduced in [9]. Therefore, we propose considering ensembles or rules in the following way: each instance is

Table 1. Summary of results obtained from ensembles of rules evolved by GP, an ensemble of ATC rules, the best rule evolved by GP and the best ATC rule. For each method, average tardiness values on the training and testing sets of instances considered in [3] are reported.

Set of rules	Size	Training	Testing
ATC(0.5)	1	1000.52	1011.33
ATC(0.1,...,1.0)	10	984.18	989.57
GP best rule	1	986.32	992.14
GP best rules	10	974.24	975.97

solved by a set of rules (10 rules or so) and then the best of the 10 solutions is taken. This method would be feasible due to the execution time being still suitable for the real time requirements of the EVCSP. To estimate the viability of this method, we may analyse the results in Table 1, which shows solutions from two single rules, ATC(0.5) and the best rule evolved by GP in [3]; and two ensembles with 10 instances of the ATC rule, g varying in $0.1 \dots 1.0$, and the ensemble formed by the 10 best of the 30 rules obtained by GP. We can observe that putting 10 learned rules to work together reduces significantly the average tardiness. These results strongly suggest that learning ensembles of rules is an interesting line of research.

3 Evolving ensembles of rules with genetic algorithms

We are given a large set $\mathcal{N}$ of N priority rules and the results of these rules over a set of M instances; i.e., the tardiness values obtained by Algorithm 1 guided by each one of these N rules. The goal is to obtain a small ensemble of P rules such that they cover the M instances; namely, such that the average value of the best solutions to the M instances obtained by the rules in P is minimized.

To solve this problem we propose a genetic algorithm (GA). The main elements of this algorithm are chosen as follows:

Coding scheme. Individuals are given by variations with repetition of the N rules taken P by P; so, there are M^N different chromosomes in all. This encoding allows for efficient genetic operators and, having duplicated rules in the chromosome, the GA may keep good rules (those covering many instances) more easily after mating and mutation. Furthermore, it gives GA the chance to reach ensembles with less than P different rules.

Initial population. Initial chromosomes are random variations of the N rules. In principle, every rule has the same probability to be chosen. However, other strategies could be better, for example giving each rule a probability proportional to the average value of the solutions obtained by that rule on the M instances; i.e., the fitness value of this rule in the GP.

Crossover. In this problem, the order of rules in the chromosome is not relevant, so it is enough to make that each offspring inherits some rules from each one of the parents. So, a simple scheme as the following may be appropriate. Given two parents, two offspring are obtained. Firstly, a binary string of length P is generated, each bit chosen uniformly in $\{0, 1\}$. Then, the first offspring includes in each position the rules from the first parent with 0 in the same position in the bit string and those with 1 in the second parent. Analogously for the second offspring swapping 0 and 1.

Mutation. Mutation plays a very important role in this GA as it is in charge of including in the population new rules from the set $\mathcal{N}$. It changes the rules in a number of positions, between 1 and $P/2$, of the chromosome by random rules chosen uniformly from $\mathcal{N}$.

Evolutionary scheme. The main structure of the algorithm is given in Algorithm 2. It is a generational GA with random selection and replacement by tournament among every two mated parents and their two offspring, which confers the GA an implicit form of elitism.

Evaluation. For each of the M instances, the solution given by the best of the P rules in the ensemble is considered. Then the fitness value of the chromosome is the average value of the M solutions.

4 Experimental study

We have conducted an experimental study aimed at analysing the behaviour of the proposed GA. To this end, we implemented a prototype in Java, the target machine was Windows on top of Intel i5 @4.2 GHz. 15.5 GB RAM.

Algorithm 2: Genetic Algorithm.

Data: A set of M instances of the $(1, Cap(t)||\sum T_i)$ problem. A set $\mathcal{N}$ of N priority rules and the average tardiness of the solutions to the N instances obtained by the N rules. Parameters: crossover probability p_c, mutation probability p_m, number of generations $\#gen$, population size $\#popsize$, chromosome length P (the size of the ensemble).

Result: A ensemble of P priority rules

Generate and evaluate the initial population $\mathcal{P}(0)$;

for *t=1 to #gen-1* **do**

- **Selection**: organize the chromosomes in $\mathcal{P}(t-1)$ into pairs at random;
- **Recombination**: mate each pair of chromosomes and mutate the two offsprings in accordance with p_c and p_m;
- **Evaluation**: evaluate the resulting chromosomes;
- **Replacement**: make a tournament selection among every two parents and their offsprings to complete $\mathcal{P}(t)$;

end

return *the best ensemble of P priority rules reached*;

4.1 Test bed and previous results

The test bed considered in this study is set $\mathcal{N}$ of N=10000 different priority rules obtained by the GP proposed in [3]. The GP exploited a set of M=50 instances of the $(1, Cap(t)||\sum T_i)$ problem as training set to evolve the rules. The same M instances are considered here. It is important to remark that the N rules correspond to different generations of the GP, therefore their qualities are quite different. The set of M=50 instances is a subset of the set of 1000 instances proposed in [9]. The remaining 950 instances in this set are considered here to evaluate the best evolved ensembles.

Before going to the experiments and results from GA, it is worth analysing some previous results. Table 2 shows the average results obtained by three classic priority rules, SPT, EDD and ATC, taking the parameter g=0.5 that is usually the best one, and the best of the N=10000 rules evolved by GP, on all the instances. As we can see, SPT is really bad as it seems natural due to the fact that it only considers processing times but not due dates. EDD is better, but it is ATC the classic rule that produces the best results of the classical rules as it considers both processing times and due dates to calculate jobs' priorities. Besides, the best rule evolved by GP is even better than ATC. As can be expected, these values are far from the results obtained by the GA proposed in [9], which of course takes much more time, about 40s per run, versus a few ms taken by the schedule builder combined with a single priority rule. If we look at the last column of the table, we can see that the quotient of values on the training and testing sets are quite similar and closed to 1.0, showing that all rules perform similarly on both sets, what in particular means that the rules evolved by GP generalize quite well.

Table 2. Summary of results obtained by the classical rules and the best rule obtained by GP on the instances of the $(1, Cap(t)||\sum T_i)$ problem considered in this study.

	Avg. Tardiness		
Rule	Training (50 inst.)	Testing (950 inst.)	Test./Train.
SPT	5473.18	5319.81	1.03
EDD	1256.84	1291.66	0.97
ATC(0.5)	1000.52	1011.33	0.99
Best rule(GP)	988.02	997.65	0.99
GA (30 runs) Best/Avg.	950.18/950.63	959.06/959.51	0.99/0.99

It is also worth to consider the tardiness values obtained by some subsets of the N rules on the M instances. This provides some reference on the results one may expect from the ensembles evolved by GA. Table 3 shows results from all rules in $\mathcal{N}$ and from some subsets. $Better_R$ denotes the subset of $\mathcal{N}$ such that each rule in the subset produces better or at least equal result to R. The second column in the table shows the number of rules of the set ($\#rules$) and the third

one shows the average tardiness from all rules on all 50 instances (so each value is the average of $\#rules \times 50$ single tardiness values). These results show that the rules in $\mathcal{N}$ produce quite different tardiness values and that restricting to subsets of rules that individually perform better than a given threshold, these subsets perform better and better. This is natural and suggests that varying $\mathcal{N}$ over these subsets the GA could obtain better and better ensembles as well.

4.2 Results from GA

The GA was run 30 times on each instance with the following parameters: $\#popsize$=100, $\#gen$=1000, p_c=0.8, p_m=0.2. Besides, four values of the ensemble size P (or chromosome size) were considered: 3, 5, 10 and 50. With these values, the time taken in each single run of GA varies between 2s and 360s.

Table 4 reports the results from GA starting from different sets of rules $\mathcal{N}$. For each value of P and set $\mathcal{N}$, the best ensemble of each of the 30 runs is recorded. Then we show the average fitness and the fitness of the best ensemble reached. Remember that to obtain the fitness of an ensemble we consider for each of the M instances the best solution from the P rules and then take the average value over the M instances. We can observe that these values clearly depend on the value of P, as expected, being better and better as long as P increases. At the same time, the results depend on the set $\mathcal{N}$ of initial solutions, even though to a lesser extent. So, the results are rather similar for the sets *All*, $Better_{SPT}$ and $Better_{EDD}$, showing that GA is robust. However, for values of P larger than 3, the results get worse if the candidate rules are restricted to the set $Better_{ATC(0.5)}$, which includes only about the best 10% of the 10000 rules considered. This fact shows that to obtain good ensembles it is not enough to count only on the best rules.

To assess the performance of the ensembles evolved by the GA, we consider the best ensembles obtained for each value of P and other ensembles obtained from classical rules or those obtained from the best rules of the initial set. Specifically, we consider the ATC rule with 10 values of the parameter g (0.1,...,1.0), which in practice perform quite well, and the ensembles given by the 3, 5, 10 and 50 best rules of the initial set. Table 5 summarizes the results of these ensembles on the training set and also on the testing set that includes 950 unseen instances.

As we can see, the ATC(0.1,...,1.0) ensemble is better or at least similar to the ensembles of the 10 or less best rules. Only from the 50 best rules is obtained

Table 3. Summary of results obtained from different subsets of $\mathcal{N}$. $Better_R$ is the subset of rules of $\mathcal{N}$ that produces a result better or at least equal to the rule R.

Set of rules	#rules	Avg. Tard. (50 inst.)
All	10000	1769.50
$Better_{SPT}$	9302	1306.53
$Better_{EDD}$	7654	1108.88
$Better_{ATC(0.5)}$	784	993.63

Table 4. Summary of results from GA starting from different sets of rules $\mathcal{N}$ (*All*, $Better_{SPT}$, $Better_{EDD}$, $Better_{ATC(0.5)}$) and for different ensemble sizes P (3, 5, 10, 50). Values in **bold** are the best ones

P	*All*		$Better_{SPT}$		$Better_{EDD}$		$Better_{ATC(0.5)}$	
	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best
3	978.52	**978.14**	978.46	**978.14**	978.31	**978.14**	978.14	**978.14**
5	975.64	974.70	975.66	**974.64**	975.49	**974.64**	975.32	975.20
10	972.74	971.90	972.55	971.48	972.55	**971.32**	972.89	972.66
50	968.14	967.56	968.03	967.62	967.91	**967.42**	971.86	971.86

a better ensemble than ATC(0.1,...,1.0). However, the ensembles evolved by GA are clearly better than those composed by the best rules. In particular, if we take the ensembles with 10 rules, which are reasonable considering the real time requirements of the EVCSP, the evolved ensemble improves in more than 10 units the values produced by the others. This value is actually relevant if we consider that the average value of the best solutions reached to the 50 instances of the test set by all the 10000 rules is 967.28. In terms of relative error we can tell that the evolved ensemble of 10 rules reduces this value in about 75%. Furthermore, for P=50, GA almost reaches the optimal solution; i.e., the best ensemble that can be built from the initial rules.

From these results, we may conclude that the proposed GA combined with the GP proposed in [3] is a good approach to obtain ensembles of rules to solve the $(1, Cap(t)||\sum T_i)$ problem.

5 Conclusions

We have demonstrated that an ensemble of priority rules is a good option to solve scheduling problems on-line. From a large set of priority rules with different characteristics, it is possible to evolve small ensembles such that for any

Table 5. Summary of results from different ensembles.

Ensemble	Size	Training	Testing
ATC(0.1,...,1.0)	10	984.18	989.57
Best rules	3	985.64	994.97
	5	985.64	994.90
	10	983.96	992.40
	50	975.58	978.56
Evolved	3	978.14	985.94
	5	974.64	981.69
	10	971.32	978.26
	50	967.42	974.68

unseen instance of the scheduling problem any of the rules in the ensemble has a high chance of getting a good solution. We have experimented with the one machine scheduling problem with variable capacity, denoted $(1, Cap(t)||\sum T_i)$, and the pool of priority rules evolved by the genetic progam proposed in [3], which evolves rules that minimize the total tardiness on a set of instances. Clearly, the same method could be applied to other scheduling problems. Besides, pools of rules evolved by other methods could also be considered. Indeed, one of the lines of research we propose is aimed to evolve rules covering instances of the $(1, Cap(t)||\sum T_i)$ problem having quite different structure.

Acknowledgements

This research has been supported by the Spanish Government under research project TIN2016-79190-R and by Principality of Asturias under grant IDI/2018/000176.

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. Durasevic, M., Jakobovi, D., Kneevi, K.: Adaptive scheduling on unrelated machines with genetic programming. Applied Soft Computing 48, 419 – 430 (2016)
3. Gil-Gala, F., Mencía, C., Sierra, M., Varela, R.: Genetic programming to evolve priority rules for on-line scheduling on single machine with variable capacity. XVIII Conferencia de la Asociación Española para la Inteligencia Artificial, MAEB (2018)
4. Hart, E., Sim, K.: A hyper-heuristic ensemble method for static job-shop scheduling. Evolutionary Computation 24(4), 609–635 (2016)
5. 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)
6. Jakobovi, D., Marasovi, K.: Evolving priority scheduling heuristics with genetic programming. Applied Soft Computing 12(9), 2781 – 2789 (2012)
7. 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)
8. Koulamas, C.: The total tardiness problem: Review and extensions. Operations Research 42, 1025–1041 (1994)
9. Mencía, C., Sierra, M., Mencía, R., Varela, R.: Evolutionary one-machine scheduling in the context of electric vehicles charging. Integrated Computer-Aided Engineering 26(1), 1–15 (2019)
10. 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)