XIII Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (XIII MAEB)

MAEB 3: Heurísticas, Hiperheurísticas y Metaheurísticas

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

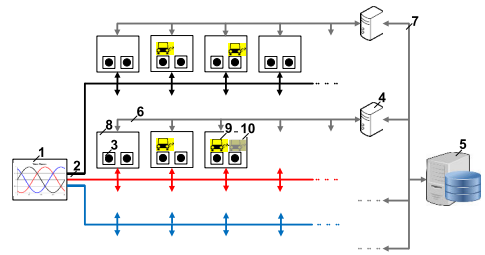


Fig. 1. General structure of the charging station. (1) Three-phase electric power 400v AC, (2) lines, (3) charging points Type 2/AC IEC 62196-2 with V2G communication interface ISO 15118, (4) masters, (5) server, (6) communication Rs 485, (7) communication TCP/IP, (8) slaves, (9) active vehicles, (10)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

events as EVs arrivals and sends activation/deactivation signals to the charging points in accordance with a schedule.

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

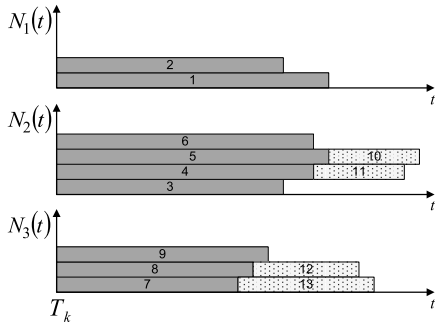


Fig. 2. A feasible schedule for the problem in Figure 1. Tasks 1 - 10 are the active EVs at time T_k in the lines 1, 2 and 3 respectively, while tasks 10 - 13 correspond to inactive EVs in the lines 2 and 3, which could be rescheduled if it were neccesary to accomodate new EVs arriving after T_k.

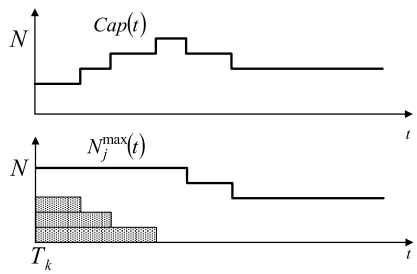


Fig. 3. Definition of the capacity of the machine $Cap(t)$ from the maximum profile $N_j^{max}(t)$ and the active EVs at time T_k.

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 pre-empted; i.e.,

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

where C_i is the completion time of job i.

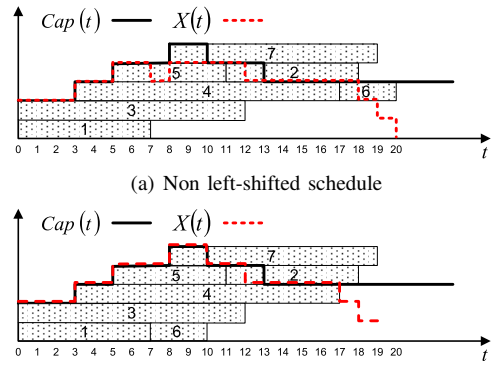
The objective function is the total tardiness, defined as:

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

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.



(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.

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).

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

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.

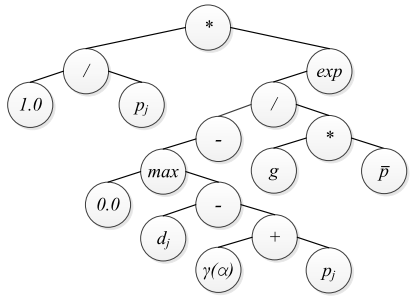


Fig. 5. Tree representing the ATC rule.

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 under-constrained 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

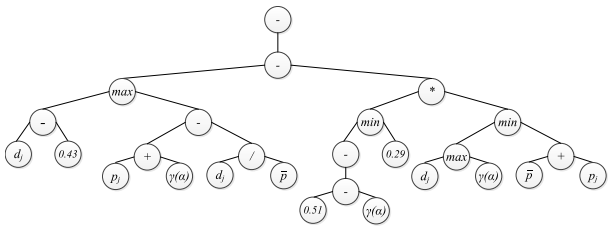


Fig. 6. Tree representing the best rule obtained.

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)

On the Landscape of a Problem of Finding Satisfactory Metaheuristics

José M. Cecilia, Baldomero Imbernón
Bioinformatics and HPC Research Group (BIO-HPC)
Polytechnic School, Univ. Católica San Antonio of Murcia (UCAM)
Murcia, Spain
{jmcecilia,bimbernon}@ucam.edu

José-Matías Cutillas-Lozano, Domingo Giménez
Department of Computing and Systems
University of Murcia
Murcia, Spain
{josematias.cutillas,domingo}@um.es

***Abstract*—Fitness landscape analysis is used to determine the distribution and shape of the fitness function for the space of solutions of optimization problems. It is a hard task which can help in the design of efficient metaheuristics to approach the solution of the problem in hand. This paper studies the landscape of a problem of finding satisfactory metaheuristics. A molecule-docking problem and a problem of determination of kinetic constants in a chemical reaction are used as case studies. We are not working at the level of the case study problems, but at a higher abstraction level, which makes the landscape analysis an even harder task. The metaheuristics being analyzed are developed with a parameterized schema, with each combination of parameters in the schema being an element of the solutions space and a metaheuristic to be applied to the base problem. The fitness for a metaheuristic is measured by applying it to some training instance of that problem. Some highlights are given on how fitness landscape analysis can be conducted for this type of problems. The ideas analyzed with the two case studies can be extended for the development of hyperheuristics searching for satisfactory metaheuristics for other optimization problems.**

***Index Terms*—fitness landscape analysis, metaheuristics, hyperheuristics, molecule-docking, reaction kinetics**

I. Introduction

Metaheuristics are used to approach the solution of complex optimization problems, but the selection of a satisfactory metaheuristic for a particular problem is a difficult task. Fitness landscape analysis has been applied to the field of metaheuristics since the 1980s [1]–[3]. It studies the shape of the fitness in the space of solutions, and gives guidelines for the design of metaheuristics for the problem in hand.

Here we are interested in the landscape of an optimization problem built on top of another optimization problem. Two optimization problems are used at the bottom as case studies: a molecule-docking problem [4] and determining the kinetic constants of a chemical reaction [5]. Several metaheuristics can be applied, and a hyperheuristic (a metaheuristic in itself) searching in the space of metaheuristics is the optimization problem at the top of our hierarchy.

A parameterized schema of metaheuristics is used. It was initially used for the application of several metaheuristics to some optimization problems [6]. The next step led us to the development of hyperheuristics on top of the metaheuristic schema [7]. These hyperheuristics has high computational costs due to the repeated application of numerous metaheuristic configurations to the problem in hand. The base problem may in turn be computationally demanding, and so the combination of the high computational cost of metaheuristics with that of the hyperheuristics working on top of metaheuristics advises the exploitation of parallelism at both levels [8]. But even the efficient exploitation of parallelism is not enough to reduce the search to an acceptable time, and so fitness landscape analysis can guide the search for satisfactory metaheuristics. The conclusions drawn here can be applied to the problem of searching for satisfactory metaheuristics for other optimization problems.

The rest of the paper is organized as follows. Section II briefly describes the two case studies. The general ideas of the parameterized metaheuristic schema and the hyperheuristic developed on top of it are presented in Section III. The fitness landscape analysis is shown in Section IV. Finally, Section V summarizes the conclusions and gives some directions for future work.

II. Case Study Problems

A. The Molecule-Docking Problem (MDP)

Virtual Screening (VS) methods analyze large libraries of small molecules (*ligands*) to search for those structures which are most likely to bind to a protein receptor or enzyme [9]. VS is computationally demanding and requires the application of approximate optimization and high performance computing techniques [10], [11].

Molecular docking is used to predict non-covalent binding of molecules. Normally, the molecules to be bound are a macromolecule (receptor) and a small molecule (ligand). There are several scoring functions that can be used to measure the strength of the bond [12]. The Lennard-Jones potential is used here. The sum of the interactions of each atom of an active site of the protein with each atom of the ligand is computed. So, the computing cost is quadratic in the sizes of the molecules.

The ligand can bind at several spots on the receptor molecule. There are six degrees of freedom for the ligand, three for its translation and three for the rotation. The spot, translation and rotation which give the lowest value of the

Supported by the Spanish MINECO and European Commission FEDER funds, under grants TIN2015-66972-C5-3-R and TIN2016-78799-P (AEI/FEDER, UE).

scoring function are sought. The values of the movements and rotations of the ligand can be approached with metaheuristics, and a search is conducted in the neighborhood of each spot. So, independent metaheuristics are applied at each spot and the reference set for the search is composed of a subset for each spot.

B. Kinetic Constants of a Chemical Reaction (KCP)

When chemical reactions occur in heterogeneous phase the variables that affect the reaction rate are temperature, pressure, composition and mass transfer rate. The kinetics of the dissolution of calcium carbonate is used for the experiments. It is a function of the concentration of the various carbonate species in the solution and, therefore, a function of the partial pressure of carbon dioxide and pH. Depending on the value of the pH, there are different ways in which the dissolution occurs: by reaction with acetic acid, by reaction with carbonic acid, and by the hydrolysis reaction [13]. Therefore, the variation of moles of calcium over time is

$$\frac{1}{V}\frac{dN_{Ca^{2+}}}{dt} = -k_1 a^{n_1}\left[H_3O^+\right]^{n_2} - k_2 a^{n_3}\left[H_2CO_3\right]^{n_4} - k_3 \quad (1)$$

where k_i are the combined reaction rate constants, n_i are the reaction orders, and a is the area of the tablet, which is known for each increase of time. The integration of the equation for a certain value of time, t_j, gives the increase of calcium in the solution as a function of the concentrations of protons and carbonic acid. An individual is represented by a real vector of size seven. Every time the fitness of an individual is computed, the whole chemical system is solved with an iterative process, and the fitness is obtained by comparing the values obtained with the simulation with those obtained experimentally.

III. Parameterized Metaheuristics and Hyperheuristics

The application of a parameterized schema of metaheuristics (Algorithm 1) to the two case studies is considered. Schemas of this type have been applied to other problems, and the fitness landscape analysis techniques here used can be extended to other hard optimization problems.

Algorithm 1 Parameterized metaheuristic schema

```
Initialize(S,ParamIni) //Generate initial set and improve elements
while (not EndCondition(S,ParamEnd)) do
  SS=Select(S,ParamSel) //Select elements for combination
  SS1=Combine(SS,ParamCom) //Combine pairs of elements
  SS2=Improve(S,SS1,ParamImp) //Improve and diversify
  S=Include(S,SS1,SS2,ParamInc) //Update the reference set
end while
```

The schema comprises some basic functions which work on sets (S, SS, $SS1$ and $SS2$) of configurations. Some metaheuristic parameters ($ParamX$) are included in each basic function, which can be implemented in different ways, and the number of parameters and their meanings also change. Landscape analysis is conducted to study the influence of the parameters on the fitness. The sets, the basic functions and the meaning of the metaheuristic parameters in the schema are commented on.

`Initialize`: Sets of candidate solutions are randomly generated. For the MDP, a subset is generated at each spot, but in the experiments only one spot is considered for low execution times. The number of elements is given by the parameter Initial Number of Elements ($INEIni$), which determines if the metaheuristic is closer to a local search ($INEIni = 1$) or to a population based method.

Some of the generated elements are improved, and the intensity of the improvement can be varied to determine the importance of the local search in the metaheuristic. A percentage ($PEIIni$) of the elements is improved by comparing its fitness with that of a neighbor configuration, which substitutes the original element if the fitness is improved. The intensity of the improvement is determined by parameter $IIEIni$, which represents the maximum number of neighborhood analysis steps.

Another intensification parameter is considered for MDP. A ligand has a number of flexibility junctions determined by its crystallographic structure. In the intensification, $NEIFlex$ represents the number of configurations obtained by rotation of the junctions to be evaluated for each candidate element. This parameter depends on the characteristics of the ligand, and so the same value is considered for improvements in the functions of the schema (after initialization and in the improvements inside the loop).

A number of the best elements is selected for the following iteration. The implementations for the two problems differ slightly at this point. In the KCP only one parameter is used, for the Number of Best Elements to include ($NBEIni$). But in the MDP a Number of Best Elements ($NBEIni$) and of Worst Elements ($NWEIni$) is established, in order to not concentrate the search on only the most promising areas. The selection can be made in different ways (random, roulette, etc). Our implementation selects the $NBEIni$ best elements, and $NWEIni$ elements are randomly selected from the half elements with the worst scores.

`EndCondition`: Typically, the end condition of a metaheuristic is a maximum number of iterations ($MNIEnd$) or a maximum number of iterations without improving the best solution ($MIREnd$). Because we are interested in the study of the influence of the parameters on the fitness, the experiments are carried out for fixed times.

`Select`: A number of the best ($NBESel$) and the worst ($NWESel$) elements of the reference set are selected. The selection is made in the same way as the elements to be included in the reference set are selected for the MDP after the improvement in the initialization.

`Combine`: A number of combinations is carried out between best ($NBBCom$), worst ($NWWCom$) and best-worst ($NBWCom$) pairs of elements. The inclusion of worst elements diversifies the search. In the MDP the combination of a pair of elements gives a new element obtained as the mean of the translation and rotation parameters of the two elements. In the KCP, the classical crossing by a middle point is used.

`Improve`: Some of the elements obtained by combination are improved in the same way as in the improvement in the initialization. The percentage of elements to be improved is $PEIImp$, and the intensification of the improvement is determined by $IIEImp$.

A diversification strategy is applied to avoid falling into local minima. The percentage of elements from the reference subsets to be diversified is $PEDImp$, and the elements obtained by diversification are improved with an intensification $IIDImp$. An element is diversified by randomly generating a new value for one of its parameters, which is selected randomly. The elements generated by diversification are improved to avoid early death.

`Include`: The $NBEInc$ best elements from all those in the reference set and those generated by combination, diversification and improvement are included in the reference set for the next iteration. The reference set is completed with $FNEIni - NBEInc$ elements randomly selected from the remaining ones.

There are eighteen metaheuristic parameters in the implementation for the MDP and sixteen for the KCP, and their influence on the fitness was analyzed varying the execution time and without considering the end condition parameters.

Hyperheuristics are methods for the selection of satisfactory metaheuristics [14]. The schema in Algorithm 1 can be used for the development of hyperheuristics. The optimization problem now consists in selecting the values of the metaheuristic parameters which give the best fitness for some training problems. The hyperheuristic is therefore another metaheuristic that can in turn be developed as a parameterized metaheuristic with the parameterized schema. The general ideas of this type of hyperheuristics when applied for the two case studies are:

- The possible solutions for the hyperheuristic are vectors of metaheuristic parameters (metaheuristics). The sizes of the vectors depend on the implementation of the parameterized schema: 18 or 16 parameters for the MDP and the KCP, with only 16 or 14 if the end condition is a fixed execution time. The hyperheuristic searches in a predetermined search range for each parameter. For the experiments with the MDP, $INEIni$ varies between 20 and 200, and a range between 0 and 100 is established for the rest of the parameters; and with the KCP, $INEIni$ and $FNEIni$ are between 20 and 200, the intensification parameters between 0 and 50 and the rest of the parameters between 0 and 100.
- The fitness for each metaheuristic is obtained through its application to some training problems. Only one training problem can give problem-dependent results, and a large number of training problems means large execution times. Fitness landscape analysis can help in determining how a particular problem is representative of the behavior of other problems.
- The execution time of a hyperheuristic of this type is very high due to the repeated application of metaheuristics to the training instances. So, the analysis of the fitness landscape could help to guide the search of the hyperheuristic for low search times.
- In general, to reduce the high execution times, the metaheuristic parameters for the hyperheuristic will be smaller than those of the metaheuristics it searches for. The sets in the search are smaller, as are the number of elements to be improved and the intensification of the improvement.
- The improvements in the initialization and inside each iteration are carried out by analyzing elements in the neighborhood, with the neighbors of an element being those which are obtained from it by changing one element of the metaheuristic parameter vector (increasing or decreasing by one).
- The combination of a pair of metaheuristics is carried out with the typical method of selecting a crossing point and combining the initial part from one ascendant with the second part from the other. If an invalid configuration of the metaheuristic parameters is generated, it is discarded.
- The diversification consists of the substitution of the value of a randomly selected metaheuristic parameter by another value randomly selected from the search range for this parameter.

There is an optimization problem at the bottom (MDP or KCP), and the fitness landscape analysis of metaheuristics for this problem studies the shape of the fitness function for some instances of the problem. At a higher abstraction level, the hyperheuristic is now the metaheuristic to be guided with the results of the fitness landscape analysis.

IV. Fitness Landscape Analysis

The fitness landscape analysis of the problem of finding satisfactory metaheuristics for an optimization problem is a hard task. The fitness of each metaheuristic is measured through its application to various training instances of the problem. The optimization problem can have high computational costs, which, together with the number of training problems, means high execution times. Furthermore, the results obtained with one or several training problems should be representative for other problems. For simplification, for the two case studies experiments were carried out for three instances of the problem and for 100 randomly generated metaheuristics, with the values of the metaheuristic parameters in the ranges previously indicated.

The characteristics of the pairs for the MDP are given in Table I; they correspond to the Angiotensin-converting enzyme (ACE), the Glycogen phosphorylase beta (GPB) and the Poly(ADP-ribose) polymerase (PARP), and the table shows the number of atoms of the receptor and the ligand and the number of junctions of the ligand. The best fitness when applying each metaheuristic to the corresponding receptor-ligand pair was stored at intervals of 30 seconds, starting at 30 seconds and finishing at 600 seconds. Three experiments for which laboratory data are available are used for the KCP. In this case, each metaheuristic is run for 100 seconds, and the fitnesses obtained at intervals of 5 seconds are stored. The results of the experiments are analyzed to study three aspects:

TABLE I
CHARACTERISTICS OF THE RECEPTOR-LIGAND PAIRS USED IN THE EXPERIMENTS.

pair	#atoms receptor	#atoms ligand	#junctions
ACE	9198	59	13
GPB	13261	29	1
PARP	5588	32	3

- The influence of the parameters in the fitness is analyzed, both individually (which parameters influence the fitness most and if the influence is positive or negative) and globally (if the distance between metaheuristics is related to the difference of their fitnesses). The hyperheuristics could concentrate the search in the parameters which prove to be the most influential, and in the ranges where the best fitnesses are obtained.
- The influence of the execution time on the distribution of the fitness and on the influence of the parameters in the fitness. This study can help establish a time limit for the application of the metaheuristics to the training problem, so reducing training time.
- Whether the results obtained for one instance are similar to the others. If they are similar or if some relation between them can be deduced, the number of training problems could be low (in the best case only one), so reducing training time.

The influence of the metaheuristic parameters on the fitness can be analyzed to determine the parameters with most influence and the ranges where the search should be concentrated. The evolution of the correlation coefficient of the metaheuristic parameters with respect to the fitness is shown in Figures 1 (MDP) and 2 (KCP). A negative value indicates that when the value of the parameter increases the fitness improves (decreases). The figures on the left show the coefficient for the parameters with the greatest positive influence on the fitness, and those on the right correspond to the other parameters. The colors identify the basic functions in the schema: blue for initialization, red for selection, black for combination, brown for improvement, cyan for inclusion, and green for *NEIFlex*.

In the MDP, the parameter which most positively influences the fitness is not always the same. For ACE it is $NEIFlex$, which also influences PARP, but with no influence on GPB. It can be logical if we consider the number of flexible junctions in each case, which is 13 for ACE, 3 for PARP and 1 for GPB (Table I). So, it seems the search by flexing the ligand is more advisable for more flexible junctions. Another influential parameter is $NBEInc$, with a correlation coefficient of around -0.2 in ACE and PARP, and a little higher than -0.1 in GPB. It is also normal that considering a large number of promising elements will provide better fitness. The percentage of elements to diversify ($PEDImp$) is the best parameter for GPB and PARP, and the value is slightly higher than -0.2 in ACE. So, it seems that diversification is better than exploring the neighborhood of promising metaheuristics. Surprisingly, the combination of non promising elements ($NWWCom$) seems to be the fourth parameter in importance for good fitness, maybe because combining these elements avoids intensification in their neighborhood.

The relevance of the parameters changes for the KCP. The most relevant parameter is always clearly $IDEImp$, which has an inverse relation with the fitness, so it is advisable not to work too much on the improvement of elements after diversification. The other parameters have little influence. The initialization parameters have in general a positive influence and, in particular, the improvement parameters in the initialization. The improvement of the elements in the successive steps ($PEIImp$ and $IIEImp$) also has a positive influence. So, the local searches lead to better results for this problem.

In any case, independently of which parameters have most influence on the fitness, the analysis can serve to guide the hyperheuristic according to the problem it is being applied to and the particular implementation of the parameterized metaheuristic schema for that problem.

There is a need to reduce the training time of hyperheuristics. Figures 1 and 2 show the relation of the metaheuristic parameters with the fitness at several time-steps. After a time (around 250 and 50 seconds) the influence is stable. Furthermore, as can be seen in Figure 3, the correlation coefficient of the fitnesses obtained at one time step with respect to those obtained at the final time is higher than 0.9 after 250 seconds for the MDP and after 60 seconds for the KCP. With the application of the metaheuristics to a training problem we are not searching for optimum solutions for this particular problem, but for satisfactory metaheuristics for the general problem. So, the training time could be reduced to approximately half the time used in the experiments.

There are other possibilities to reduce the training time. For example, with the 100 metaheuristics experimented with in 20 time-steps, if the five metaheuristics with the worst fitness are discarded at each step, the total training time is halved. The difference in the behavior with the different problems and instances considered can be seen in Figure 4, where the mean fitness of the five metaheuristics discarded at each time step is shown. The similarity in the results for GPB is observed in its flat shape. For the MDP, the slope of the lines seems to be related to the number of flexible junctions (Table I) and the difficulty of approaching the optimum.

V. Conclusions and Future Work

This paper presents a first approach to the analysis of the fitness landscape for the problem of finding satisfactory metaheuristics for hard optimization problems. A molecule-docking problem and a problem of obtaining the values of kinetic constants are used as case studies, but the same methodology can be applied to the search for metaheuristics for other problems. The metaheuristics considered are implemented with a parameterized schema, which is used for the development of a hyperheuristic which searches in the space of metaheuristics. The fitness for each metaheuristic is obtained with its application to some training problems. The main difficulty in the fitness landscape analysis is the

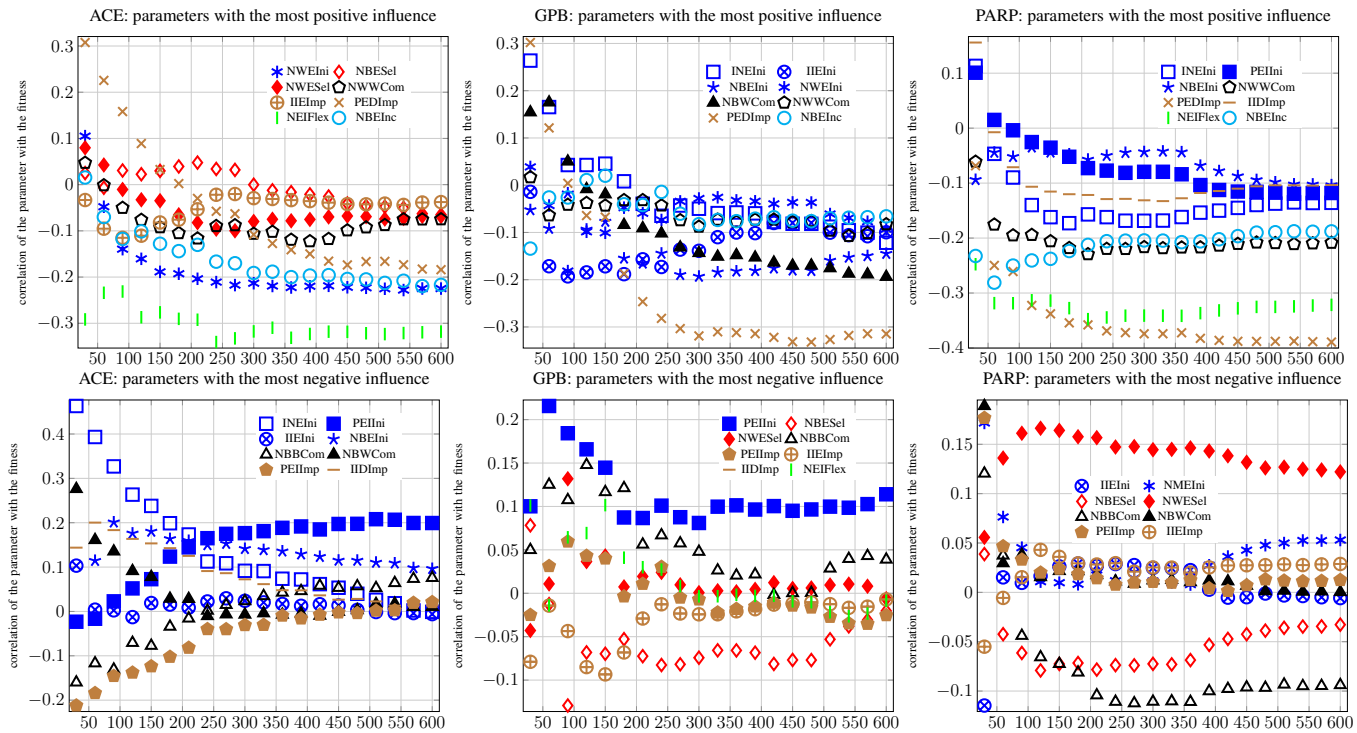


Fig. 1. Evolution of the correlation coefficient of the metaheuristic parameters with the fitness, for the MDP, for several execution times for the receptor-ligands ACE (left), GPB (middle) and PARP (right). Top: the eight parameters with the most direct relation with the fitness after 600 seconds; Bottom: the eight parameters with the most inverse relation.

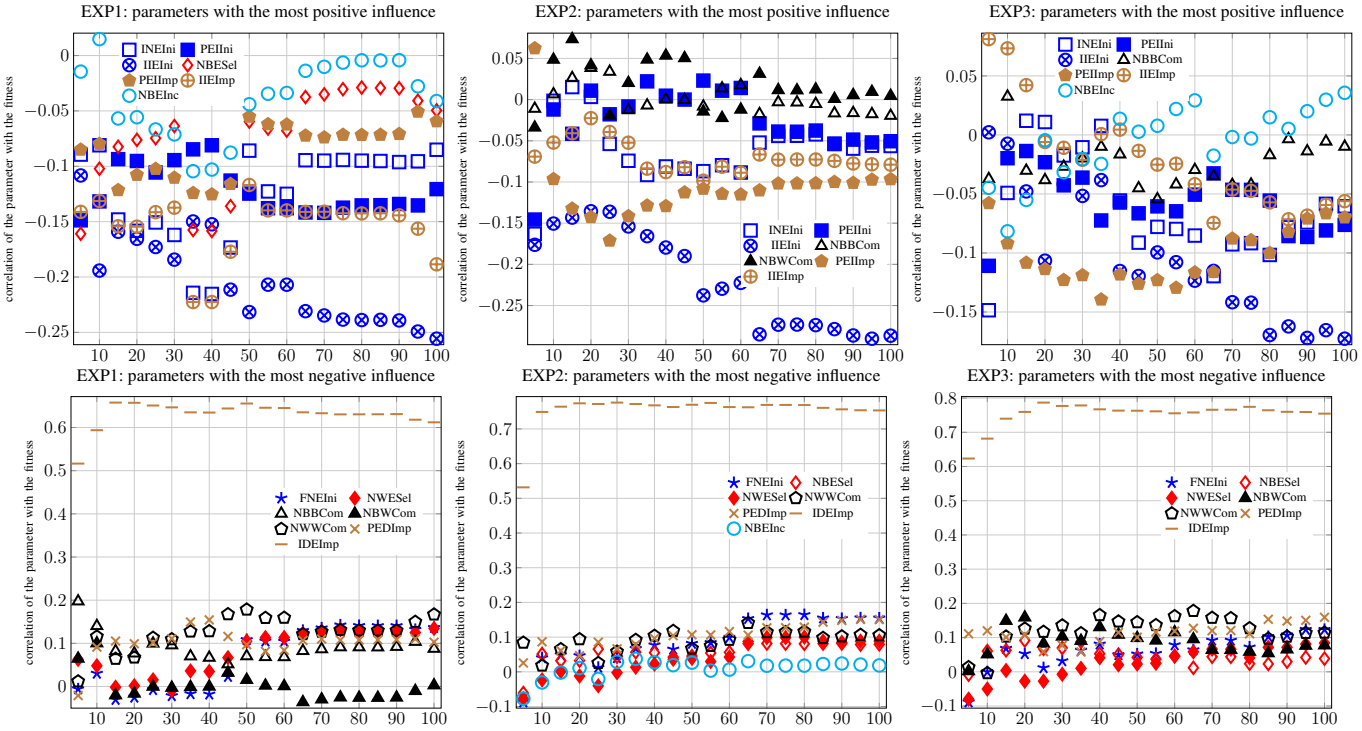


Fig. 2. Evolution of the correlation coefficient of the metaheuristic parameters with the fitness, for the KCP, for several execution times for the experiments EXP1 (left), EXP2 (middle) and EXP3 (right). Top: the seven parameters with the most direct relation with the fitness after 100 seconds; Bottom: the seven parameters with the most inverse relation.

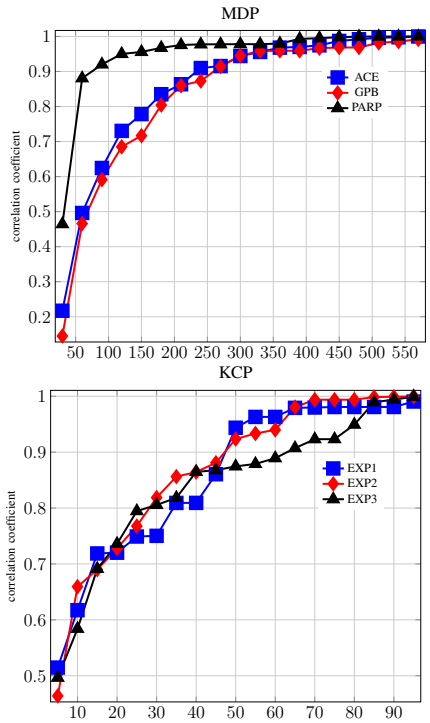


Fig. 3. Evolution of the correlation coefficient of the fitness at different time-steps with respect to the fitness. Top: MDP after 600 seconds; Bottom: KCP after 100 seconds.

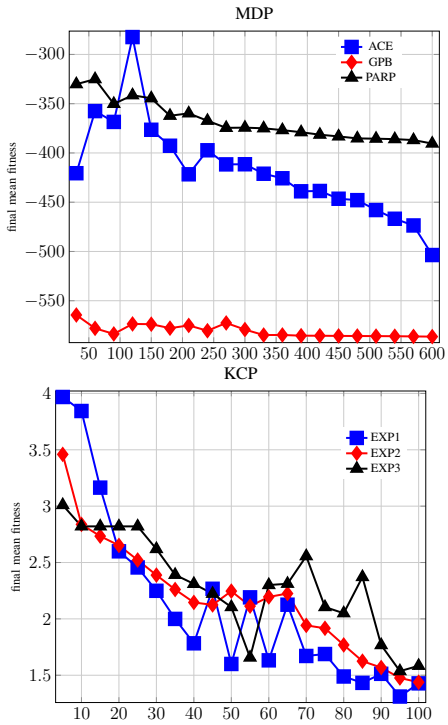


Fig. 4. Mean of the final fitness of the metaheuristics discarded at different times-steps. Top: MDP, plot each 30 seconds; Bottom: KCP, plot each 5 seconds.

randomness of the application of the metaheuristics, which makes the surface of the fitness function dependent on the random execution and on the instances used in the training process. The influence of the parameters and of the training time on the fitness has been experimentally analyzed.

The results of our analysis are being used to guide the implementation of hyperheuristics on top of the parameterized metaheuristic. More experiments with more instances of the case study problems are needed, and similar studies for other optimization problems at the bottom of the metaheuristics hierarchy will be conducted.

References

[1] W. Hordijk, "A measure of landscapes," *Evol. Comput.*, vol. 4, pp. 335–360, Dec. 1996.

[2] K. Malan and A. P. Engelbrecht, "A survey of techniques for characterising fitness landscapes and some possible ways forward," *Inf. Sci.*, vol. 241, pp. 148–163, 2013.

[3] P. F. Stadle, "Towards a theory of landscapes," in *Complex Systems and Binary Networks*, vol. 461 of *Lecture Notes in Physics*, ch. 10, pp. 78–163, 2007.

[4] E. López-Camacho, M. J. García-Godoy, J. García-Nieto, A. J. Nebro, and J. F. A. Montes, "Solving molecular flexible docking problems with metaheuristics: A comparative study," *Appl. Soft Comput.*, vol. 28, pp. 379–393, 2015.

[5] J. Cutillas-Lozano and D. Giménez, "Determination of the kinetic constants of a chemical reaction in heterogeneous phase using parameterized metaheuristics," in *Proceedings of the International Conference on Computational Science*, pp. 787–796, 2013.

[6] F. Almeida, D. Giménez, J.-J. López-Espín, and M. Pérez-Pérez, "Parameterised schemes of metaheuristics: basic ideas and applications with Genetic algorithms, Scatter Search and GRASP," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 43, no. 3, pp. 570–586, 2013.

[7] J. Cutillas-Lozano, D. Giménez, and F. Almeida, "Hyperheuristics based on parametrized metaheuristic schemes," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 361–368, 2015.

[8] J. M. Cecilia, J.-M. Cutillas-Lozano, D. Giménez, and B. Imbernón, "Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem," *The Journal of Supercomputing*, Mar 2017.

[9] U. Rester, "From virtuality to reality-Virtual screening in lead discovery and lead optimization: a medicinal chemistry perspective," *Current Opinion in Drug Discovery & Development*, vol. 11, no. 4, pp. 559–568, 2008.

[10] S. Garcia Sanchez, E. Ramirez Aportela, J. I. Garzón, P. Chacón, A. S. Montemayor, and R. Cabido, "FRODRUG: A virtual screening GPU accelerated approach for drug discovery," in *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 594–600, 2014.

[11] B. Imbernón, J. M. Cecilia, and D. Giménez, "Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems," in *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*, pp. 50–58, 2016.

[12] E. Yuriev, M. Agostino, and P. A. Ramsland, "Challenges and advances in computational docking: 2009 in review," *Journal of Molecular Recognition*, vol. 24, no. 2, pp. 149–164, 2011.

[13] L. N. Plummer, T. M. L. Wigley, and D. L. Parkhurst, "The kinetics of calcite dissolution in CO_2–water systems at 5 to 60°C and 0.0 to 1.0 atm CO_2," *American Journal of Science*, vol. 278, pp. 179–216, 1978.

[14] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, *A Classification of Hyper-heuristic Approaches*, pp. 449–468. Springer, 2010. In Michel Gendreau, Jean-Yves Potvin, editors, Handbook of Meta-heuristics.

Generador de Planificaciones Conjuntas para Cuadrante de Rotación de los Conductores de Autobuses

Eva M. Almansa Aránega
Dept. Ciencia de la Computación e Inteligencia Artificial
Universidad de Granada
Granada, España
Eva.M.Almansa@gmail.com

Samuel Lopéz Liñán
Dept. Ciencia de la Computación e Inteligencia Artificial
Universidad de Granada
Granada, España
samusanlzl@gmail.com

Francisco José Rodríguez Serrano
R&D Coordinator
SHS Consultores S.L.
Sevilla, España
fjrodriguez@shsconsultores.es

José Manuel Benítez Sánchez
Dept. Ciencia de la Computación e Inteligencia Artificial
Universidad de Granada
Granada, España
J.M.Benitez@decsai.ugr.es

***Abstract*—El Problema de Generación de Cuadrante de Rotación de los Conductores de Autobuses (PGCRCA) se refiere a la planificación del horario de trabajo de los conductores a lo largo de un determinado periodo de tiempo, como por ejemplo un año. Es un problema de naturaleza NP-duro donde habitualmente existen un número alto de restricciones a cumplir, dificultando las decisiones en cada paso de la construcción de una solución. Existen distintas propuestas de resolución de versiones simplificadas en la literatura. En este trabajo abordamos un problema real con amplia diversidad de restricciones y un número mucho mayor del recogido en las publicaciones: diferentes planificaciones que comparten recursos. Hemos desarrollado una heurística que resuelve eficazmente todos los casos reales planteados por empresas de distinto tamaño del sector. Eficacia y eficiencia de la heurística han sido evaluadas en un extenso conjunto de casos reales y se ha podido constatar las buenas propiedades computacionales del algoritmo.**

Keywords—cuadrantes de rotación de conductores de autobuses; rotación acíclica; backtracking

I. Introducción

Dentro de la gestión de transporte de viajeros en trasporte público (TP), un problema es la asignación del horario de trabajo para cada uno de sus conductores de autobuses. El horario de trabajo presenta un alto coste a la empresa. No sólo para obtener una correcta asignación de tareas se debe de tener en cuenta el convenio colectivo, sino además la equidad en el horario de cada conductor. De este modo, el reparto de tareas repercute directamente en el coste para la empresa y en la satisfacción de cada conductor.

La rotación se encuentra en distintos contextos laborables y es uno de los problemas que menos se ha tratado en la literatura académica en comparación con el problema *Cuadrante de Personal* (PCP). Su complejidad reside en la asignación de turnos y descansos con condiciones similares dentro de un periodo determinado y su *rotación* en el siguiente periodo, según corresponda, hasta completar el tiempo de planificación.

El tipo de problema a resolver, a su vez presenta una versión distinta a todas las recogidas en la literatura, diferentes planificaciones de forma simultánea. Lo cual requiere de una propuesta de solución específica.

Un *turno* es una secuencia de tareas a lo largo de un día, el cual tiene características que lo distinguen del resto, como la franja horaria y la hora de inicio y fin. La *homogeneidad* (franja horaria) y la *continuidad* (hora de inicio y fin) se utilizan para valorar la similitud entre los turnos dentro de un periodo. Un *periodo* es el tiempo que determina el siguiente cambio de turno o descanso, puede ser una semana o más de una, a lo largo de una planificación. Una *planificación* se encuentra en un rango desde un fecha determinada y puede comprender hasta un año. Pueden existir varias planificaciones, lo que conlleva un reparto de los recursos, el cuadrante de los conductores. Cada planificación es independiente de las demás con respecto a las características que la definen, como por ejemplo, la fecha de inicialización y finalización; el tipo de turnos; o el tipo de conductores que usa. Los *tipos de conductores* difieren entre sí en la disponibilidad y/o en sus condiciones laborables. Lo cual describe una variante *acíclica* del problema de Generación de Cuadrante de Rotación de los Conductores de Autobuses (PGCRCA), resultando más compleja que la variante *cíclica*, donde no existen diferencias entre las condiciones de los conductores.

Por lo tanto, PGCRCA es considerado un problema de optimización *multi-objetivo* sobre las condiciones para la satisfacción del conductor y el coste de la empresa.

El resto del artículo está organizado de la siguiente forma. En la Sección II, se presenta la revisión bibliográfica sobre la

metodología del PGCRCA. Sección III introduce el problema PGCRCA. Sección IV presenta la propuesta, el planificador de rotación. Sección V proporciona distintos ejemplos con diferentes características para ilustrar la aplicación del método propuesto. Finalmente, las conclusiones se encuentran en la Sección VI.

II. Estado del arte

El problema de tener un alto número de restricciones implica un aumento del número de decisiones en la generación de una planificación. Por lo que para comprobar todas las posibilidades sería necesaria una búsqueda recursiva, con el añadido problema del tiempo de complejidad. En [1] define la complejidad del clásico *Problema de Satisfacción de Restricciones* (PSC) utilizando una búsqueda recursiva.

En el PGCRCA, para conseguir un reparto equitativo en la carga de trabajo total hay que tener en cuenta demasiados factores. La normativa de los conductores implica no sólo las reglas de su contrato laboral, sino que también hay que considerar la rotación de los turnos y descansos. Por esta razón, podemos encontrar distintas formulaciones con diferentes modelos matemáticos y con diferentes objetivos. Una de las primeras definiciones que se pueden encontrar en la literatura, se encuentra en [2] que define el problema como un *Multi-level Bottleneck Assigment* (MBA) y demuestra que es NP-duro. Como propuesta, describe un algoritmo que resuelve de forma iterativa un conjunto de sub-problemas para obtener una solución "óptima asintóticamente". De este modo, se puede encontrar en la literatura distintos enfoques para tratar de resolver este tipo de problema. Algunas propuestas, tales como en [3] con dos heurísticas evolutivas; y en [4] con una solución basada en la técnica *colonia de hormigas* comparada con dos técnicas, *búsqueda tabú* y *enfriamiento simulado*. Otros enfoques como, [5] divide el problema en dos etapas: primero resuelve el problema de modo *cíclico*, es decir, todos los conductores son de igual tipo, con iguales condiciones; y segundo aplica una mejora con un *algoritmo genético*.

El [6] contiene las definiciones, anotaciones, y propone una taxonomía del problema de Cuadrante de Rotación (PCR). De este modo, se puede conocer el alcance y diferencias entre todas las variantes posibles dentro del contexto del PCR. Al igual que se logra comprender la dificultad de la generación de una solución factible según qué variante estemos tratando.

Los [7] y [8] recogen revisiones bibliográficas de distintas especificaciones del PCR. Por un lado, el [7] propone resolver de forma conjunta el PCP y el PCR, basándose en otros artículos que integran los dos problemas en uno; y la regulación a la hora de asignar turnos para una mayor satisfacción de los conductores. El modelo consiste en resolver los dos problemas de forma relajada inicialmente y luego en aplicar unas *post-optimizaciones* para obtener soluciones alternativas que mejoren los resultados hasta llegar al óptimo o hasta los criterios de parada que se establezcan. La función objetivo que establecen para buscar la equidad entre los conductores, es la suma del valor absoluto de la distancia de cada turno (media de horas mensuales trabajadas) a un objetivo de número de horas mensuales. Por otro lado, el [8] en su propuesta trata conjuntamente las dos variantes del PCR: rotación *cíclica* y *acíclica*. C*íclica* se refiere cuando los conductores pertenecen a la misma clase y la rotación es exactamente igual en la línea de trabajo, variando únicamente la hora de inicio. A*cíclica* se refiere a que cada conductor tiene distintas condiciones por lo que existen distintos tipos de conductores.

III. Cuadrante rotación de conductores de autobuses

La generación de cuadrantes de rotación es considerada un problema que forma parte de un área clásica (Sistema de planificación de transportes) en el campo de la investigación de operaciones en el ámbito de los sistemas de transportes. Tradicionalmente este problema se ha subdividido en varios sub-problemas debido a la complejidad del mismo. En la Fig. 1, aparecen descritos éstos y el flujo clásico de resolución de los mismos [9]). A continuación, describiremos de forma general cada uno de los problemas:

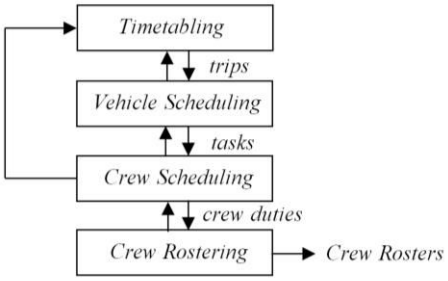


Fig. 1. Sub-problemas del sistema de planificación de transportes.

1) Generación de horarios (Timetabling)[10]: Partiendo de la información sobre el número de líneas y el número de servicios y/o frecuencias requeridas se generan los horarios de cada una de las líneas. Estos horarios incluyen los trayectos, cada uno de los cuales se describe con hora de salida, hora de llegada, punto de partida y punto de llegada.

2) Planificación de vehículos (Vehicle scheduling problem) [11]: Sobre la base de los trayectos generados en la fase anterior, el objetivo en esta etapa es optimizar el uso de los vehículos de forma que se cubran todos los trayectos definidos en la etapa anterior y se minimice el coste de operación. Este coste de operación viene definido por factores como el número de autobuses necesarios, los kilómetros recorridos, y los tiempos de espera de los vehículos entre distintos trayectos.

3) Planificación de conductores (Crew/bus-driver scheduling Problem) [12]: Una vez se dispone de una asignación de vehículos a los distintos trayectos es necesario asignar conductores que cubran de forma óptima las tareas diarias, esto es, la conducción desde el depósito de vehículos al comienzo de la jornada hasta la vuelta al mismo al final de la jornada. Cada una de estas tareas se puede dividir en sub-tareas que vienen definidas por estaciones o puntos en los que es posible realizar un cambio de conductor. Por tanto, el resultado final es la generación de un conjunto de sub-tareas y su asignación a un conductor. Usualmente esta planificación tiene un horizonte temporal corto, aproximadamente 24 horas.

4) Generación de cuadrantes de rotación (Bus crew rostering problem) [12]. En este caso se genera una planificación de los turnos de trabajo con un horizonte

temporal más amplio que para el caso anterior, por ejemplo 1 mes o 1 año, aparte de tener en cuenta ciertas restricciones como puede ser el número de horas de descanso entre dos turnos de trabajo, número de días libres, etc. El resultado final es una secuencia de asignación de tareas diarias para cada conductor que tenga en cuenta las restricciones fijadas y que consiga minimizar los costes de operación.

El PGCRCA se puede formular en los siguientes términos: los servicios a prestar están organizados en un conjunto de turnos, repartidos por días según un calendario establecido. El personal está compuesto por un conjunto de conductores, con características posiblemente distintas en cuanto a disponibilidad (ajena), y tipo de turnos que puede hacer. El objetivo es asignar cada turno a un conductor de modo que se optimicen varios criterios y se cumplan una serie de restricciones, como por ejemplo, contemplar dedicaciones en días laborables y festivos, y respetar días de descanso.

Para generar una solución se tiene una función *multi-objetivo*, con prioridad para cada criterio de optimización. Los criterios de optimización permiten generar diferentes soluciones por ser flexibles en cada aplicación. Sin embargo, las restricciones de los conductores se tienen que cumplir para considerar la solución como factible.

A. *Criterios de optimización*

En orden de mayor a menor preferencia:

1) Minimizar el número de conductores en el periodo.
2) Minimizar el número de conductores en la planificación.
3) Maximizar el número de conductores que cumplen las horas en el periodo.
4) Maximizar el número de conductores que cumplen las horas en las planificaciones.
5) Maximizar el número de descansos continuos.
6) Maximizar el número de asignaciones de turnos en la misma franja (mañana, tarde o noche) a lo largo del periodo, homogeneidad.
7) Maximizar el número de turnos con igual características en cada periodo, continuidad.
8) Maximizar el número de periodos que cumplan la rotación.

B. *Restricciones de los conductores para ser asignables*

Las restricciones que se han de cumplir son:

- Estar disponible (motivos de no estar disponible serían: vacaciones, enfermedad, baja laboral, o bien, ausencia).
- Cumplir los requisitos para ser asignable, compatibilidad con los turnos y tener un estado asignable (según indique la agenda).
- Cumplir los descansos en el periodo.
- Descanso de al menos 12 horas entre que finaliza un turno y comienza el siguiente.
- No sobrepasar las horas de trabajo indicadas en el periodo.
- Rotación de descansos en días laborables.
- Excepciones de línea.
- Mantener continuidad en los tipos de turnos durante la semana.
- Cuadrantes para una o más planificaciones.
- Se deben repartir todos los turnos durante el tiempo que dure cada planificación.

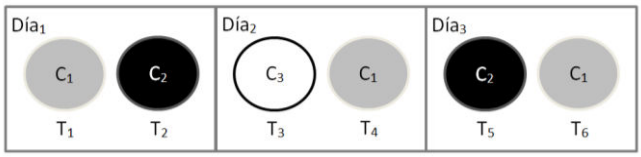
IV. Planificador de rotación

Denominamos *Planificador de Rotación* (PlanRot) a la propuesta para resolver el problema PGCRCA que hemos desarrollado. PlanRot combina la aplicación de la técnica *backtracking* con distintas etapas de refinamiento (mejora) en la generación de una solución factible.

La Fig. 2 representa un ejemplo de una solución donde cada color del círculo indica una franja horaria distinta. Partimos de una lista de turnos distribuidos a lo largo de los días de la planificación. Para cada turno |T| se realiza una búsqueda de selección del conductor |C| que presente mejores condiciones.

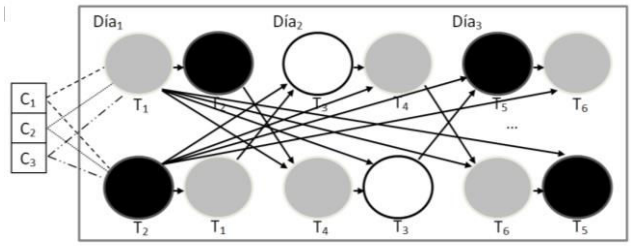
La Fig. 3 es un ejemplo de la estructura del espacio de búsqueda. Ilustra todas las posibles combinaciones que dependen del orden de los turnos y del orden de selección de los conductores. Toda decisión tomada en la asignación de un conductor a un turno repercute en la siguiente asignación. De este modo, el espacio de búsqueda se ve reducido debido a las características de cada turno cubierto, dado que restringe en las siguientes decisiones, al igual que se obtienen valoraciones diferentes en los criterios de optimización.

Fig. 2. Ejemplo de representación de la solución. Aquellos conductores *(C_1, C_2 y C_3)* que mejor cumplan las condiciones son asignados a cada turno *(T_1, ..., T_6)* de la lista que suceden a lo largo de una planificación *($Día_1$, $Día_2$ y $Día_3$)*.



Conductor	$Día_1$	$Día_2$	$Día_3$
C_1	T_1	T_4	T_5
C_2	T_2	-	T_6
C_3	-	T_3	-

Fig. 3. Ejemplo de estructura del espacio de búsqueda. La selección del orden los conductores y el orden en la asignación de turnos representa el espacio de búsqueda.



A. *Heurística. Criterios de optimización*

Empíricamente, se ha llegado a la conclusión de que la ordenación de los turnos es significativa. Por lo que, se comienza asignando los turnos por semanas completas, de lunes a domingo (o en su defecto desde el día de inicio de la planificación hasta el domingo). En cada semana se asignan primero aquellos que se encuentren en los días con mayor número de turnos y en caso de que sea igual, se comienza por los turnos del fin de semana.

Para la selección de un conductor se tienen en cuenta la valoración a partir de los criterios de optimización descritos en la Sección III. El orden de prioridad en los criterios de optimización del *1)* a *5)* es fijo. Los criterios *1)* y *2)* son los más importantes, por lo que la función objetivo se define con la función *minimizar el número de conductores*. Es decir, durante la generación de la planificación se evalúan distintas asignaciones de conductores a un turno, y se selecciona aquella que presente el mínimo número de conductores. En el caso de que existan soluciones con igual número de conductores se evalúan los criterios de siguiente orden. Se comprobarán en orden desde el *3)* al *5)* mientras que no resulte una asignación con mejora en algún criterio evaluado, y que por tanto, ya no será necesario evaluar con los siguientes criterios: por ejemplo, si existe una única asignación que presenta mejora con el criterio *3)* ya no se evaluarán los criterios *4)* y *5)*. Por otro lado, se permite cambiar la prioridad de los criterios de optimización *6)*, *7)* y *8)*. De este modo, se pueden obtener diferentes soluciones y posteriormente seleccionar aquella qué mejor se ajuste a los criterios de optimización en orden del *1)* al *5)*.

B. *Backtracking. Sistema de penalización*

La técnica algorítmica *backtracking* se ha utilizado para la parte principal del problema. Se trata de una técnica exacta que localiza una solución óptima en un espacio de búsqueda cuando se representa en forma de árbol. El mecanismo de exploración que usa es una búsqueda en profundidad (post-orden). Para acelerar el proceso de búsqueda emplea una función de poda que permite descartar subárboles (soluciones parciales y todas las soluciones derivadas de esta) sin tener que explorar todo el subárbol. Así cuando un subárbol es descartado, la exploración vuelve un nivel hacia arriba en el árbol, buscando la siguiente rama a explorar. La exploración de un nodo dado termina cuando se han explorado todos sus hijos. La exploración del árbol completo termina cuando se ha explorado la raíz. Se garantiza así encontrar la solución óptima. Para que el proceso sea computacionalmente factible la definición de la función de poda es crítica. Es fundamental, encontrar una que pode cuanto más mejor. Sin embargo, considerando que el problema es de naturaleza NP-duro siempre existirán casos que obligarán a una exploración muy amplia del espacio de soluciones, con un coste computacional muy elevado.

Para formular el espacio de búsqueda (árbol de búsqueda), las *n*-uplas que forman el espacio de soluciones tienen tantas componentes como turnos a cubrir. Los valores que puede tomar cada componente son los posibles conductores. De esta forma, se incluyen todas las posibles asignaciones entre turnos y conductores disponibles. En cada nivel del árbol se evalúan los posibles conductores candidatos a realizarlo.

Para realizar la poda se utiliza el valor de la mejor solución encontrada que es igual al número de conductores (criterios de optimización *1)* y *2)*) más una pequeña penalización (criterios de optimización del *3)* al *8)*). Esta penalización, siempre un número racional entre 0 y 1, evalúa otros objetivos secundarios como el que la solución sea poco homogénea (una solución homogénea es cuando los conductores realizan turnos del mismo tipo, mañana o tarde durante una semana y el tiempo de trabajo de los conductores es lo más parecido posible). Estas penalizaciones están directamente relacionadas con los criterios de optimización:

- *Mayor desviación típica sobre el total de minutos de trabajo.*
- *No homogeneidad.*
- *No continuidad.*
- *Descansos no continuos.*
- *Incumplimiento de rotación.*

C. *Métodos de refinamiento*

Los métodos de mejora de las soluciones siguen una heurística específica basada en parte en el algoritmo *backtracking*. El recorrido en el método más complejo se basa en asignar un turno específico que no ha sido posible asignar a ningún conductor. Este método intenta asignar el turno al conductor con menos turnos, de forma que si no es posible asignarlo entonces comprueba si quitándole un turno a ese conductor sería posible asignarlo. Si así fuese, entonces se quita un turno y se asigna el turno actual. Luego se repite el mismo procedimiento con el nuevo turno sin asignar que se ha quitado. Además de éste se aplican otros 3 métodos concretos que tratan de mejorar la solución centrándose en cuestiones más específicas.

El algoritmo PlanRot realiza una búsqueda con distintos orden en los criterios de optimización en los criterios *6)*, *7)* y *8)*; esto implica que resulten diversas construcciones. En muchos casos, las soluciones presentan periodos de tiempo con una incompleta continuidad, homogeneidad e incluso días sin asignar. Los *días sin asignar* aparecen en las semanas en las que no se ha completado el número máximo de turnos para asignar los días de descanso. Este fenómeno está provocado principalmente por dos motivos: uno, la rotación de los turnos y descansos; y dos, el tiempo mínimo de descanso entre que finaliza un turno y comienza el siguiente, sobre todo al rotar de domingo al lunes.

La mejora de la calidad de las soluciones se realiza mediante las siguientes etapas:

1) Reasignación de turnos: Los conductores que presentan mayor número días sin asignar se eliminan de la planificación. De este modo todos sus turnos quedan sin asignar. Teniendo en cuenta que, el número máximo de turnos a asignar deberá de ser igual o menor que el número de días sin asignar del resto de conductores. Mientras sea posible, se realiza el intercambio entre los turnos y los días sin asignar más compatibles.

2) Reasignación de turnos intensiva: A partir de la etapa previa, se determina el tipo de incompatibilidad en aquellos turnos que no hayan podidos ser reasignados con el fin de

reasignarlos en algún otro conductor. Estas incompatibilidades se clasifican en los siguientes casos y se realiza el intercambio siempre que exista otro conductor con el día libre y compatible al turno a asignar:

- Un conductor tiene todos los días de trabajo asignados.
- Franja horaria.
- Rotación de descansos.

3) Reasignación de descansos: Recorre cada semana e identifica los conductores con descansos discontinuos. Una vez localizados, realiza un nuevo recorrido para intercambiar entre los conductores identificados los días de descanso. Para realizar el intercambio, al menos tiene que beneficiar a un conductor por conseguir los días de descanso continuos.

4) Reasignación de turnos no homogéneos: Recorre cada día y obtiene los turnos no homogéneos de cada conductor y realiza intercambio entre otro conductor con turno no homogéneo. Se selecciona la mejor combinación de intercambio entre turnos no homogéneos.

V. Resultados experimentales

Se ha aplicado PlanRot sobre casos reales planteados por empresas. Estos casos difieren entre sí en el tamaño, como por ejemplo, número de turnos; número de semanas; tipos de conductores; o número de planificaciones.

La TABLA I. muestra la descripción de cada caso: característica más descriptiva del caso real; el número de planificaciones a resolver; el número de semanas del cuadrante; el número de rotaciones de turnos por periodo; y las rotaciones por semanas en los descansos en días laborables. Los casos de prueba se encuentran ordenados de menor a mayor complejidad.

La TABLA II. contiene los resultados de las soluciones obtenidas comparadas con las soluciones óptimas de cada caso; y el computo de procesamiento en segundos. El número óptimo de conductores se ha obtenido analizando las posibles combinaciones. Asimismo, el algoritmo PlanRot encuentra en todos los casos una solución óptima con el mínimo número de conductores necesario. En general, los casos de prueba que necesitan menos de 10 segundos para obtener una respuesta, son aquellos los que sólo utilizan el método heurístico con los criterios de optimización. El resto presentan mayor tiempo de cómputo por ser necesario aplicar el sistema de refinamiento, con un máximo de menos de 2 minutos.

TABLA I. Información de los casos reales

Caso	Descripción	Plan	Semanas	Rotación turnos	Rotación descansos
1	Asignación	1	2	-	-
2	2 tipos de días: laborable y no	1	1	-	-
3	2 tipos de conductores	1	1	-	-
4	Comienzo a mitad de la semana, 15 días	1	3	-	-
5	Rotación descansos	1	2	-	1
6	Agenda conductores	1	1	-	-
7	1 tipo de día	1	4	2	1
8	2 tipos de días	1	4	2	1
9	Comienzo a mitad de la semana, 15 días	1	3	1	-
10	2 tipos días, 2 tipos conductores	1	4	1	1
11	3 franjas horarias	1	2	1	-
12	2 planificaciones	2	1	-	-
13	Larga duración	1	18	1	2
14	Año completo	1	52	1	2
15	3 tipos de conductor	1	2	1	1
16	3 planificaciones	3	2	1	1
17	4 planificaciones	4	2	1	1
18	30 semanas	1	30	1	2
19	40 semanas	1	30	1	2
20	50 semanas	1	50	1	1
21	Distintos periodos (2, 3 y 4 semanas)	3	30	1	1

TABLA II. Soluciones

Nº	Nº óptimo conductores	PlanRot	Tiempo (seg)
1	10	10	5.914
2	9	9	4.506
3	10	10	9.667
4	10	10	11.874
5	10	10	8.432
6	11	11	7.111
7	10	10	9.469
8	9	9	8.533
9	8	8	9.142
10	8	8	4.285
11	13	13	6.386
12	16	16	5.722
13	10	10	24.787
14	18	18	30.895
15	10	10	5.144

Nº	Nº óptimo conductores	PlanRot	Tiempo (seg)
16	20	20	10.054
17	30	30	7.164
18	10	10	63.492
19	10	10	83.022
21	10	10	101.888

VI. Conclusiones

En este trabajo hemos abordado el problema PGCRCA. Se trata de una variante más compleja de un problema de naturaleza NP-duro. La versión estudiada considera distintos tipos de conductores y varias planificaciones simultáneamente. Para resolverlo de forma eficaz hemos propuesto una heurística basada en *backtracking* que combina distintos métodos de refinamiento. La evaluación experimental permite comprobar la eficacia y eficiencia de la propuesta sobre un nutrido conjunto de casos reales.

Agradecimientos

Este trabajo ha sido parcialmente financiado por los proyectos del Ministerio de Economía y Competitividad TIN2016–81113–R y de Excelencia de la Junta de Andalucía, P12–TIC–2958.

Bibliografía

[1] Marek, Victor W., y Jeffrey B. Remmel. “The complexity of recursive constraint satisfaction problems.” *Annals of Pure and Applied Logic* 161 (2009): 447-457.

[2] Carraresi, P., y G. Gallo. “A multi-level bottleneck assignment approach to the bus drivers rostering problem.” *European Journal of Operational Research* 16 (1984): 163-173.

[3] Xie, Lin, “Metaheuristics approach for solving multi-objective crew rostering problem in public transit,” 2013.

[4] Moz, Margarida, Ana Respcio, y Margarida Vaz Pato, “Bi-objective evolutionary heuristics for bus driver rostering,” *Public Transport* (Springer Nature) 1 (aug 2009): 189-210.

[5] Ma, Jihui, Tao Liu, y Wenyi Zhang, “A Genetic Algorithm Approach to the Balanced Bus Crew Rostering Problem,” *Journal of Traffic and Logistics Engineering Vol* 2 (2014).

[6] Ernst, A. T., H. Jiang, M. Krishnamoorthy, y D. Sier, “Staff scheduling and rostering: A review of applications, methods and models,” *European Journal of Operational Research* (Elsevier BV) 153 (feb 2004): 3-27.

[7] Valdes, Vera, y Victor Andres, “Integrating Crew Scheduling and Rostering Problems,” Ph.D. dissertation, alma, 2010.

[8] Xie, Lin, y Leena Suhl, “Cyclic and non-cyclic crew rostering problems in public bus transit.,” *{OR} Spectrum* (Springer Nature) 37 (apr 2014): 99-136.

[9] Freling, R., “Models and Techniques for Integrating Vehicle and Crew Scheduling”. Ph.D. thesis, Tinbergen Institute, Erasmus University Rotterdam (1997).

[10] Ibarra-Rojas, Omar J., y Yasmin A. Rios-Solis. “Synchronization of bus timetabling.” *Transportation Research Part B: Methodological* 46 (2012): 599-614.

[11] Park, Yang-Byung. “A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines.” *International Journal of Production Economics* 73 (2001): 175-188.

[12] Ma, Ji Hui, Tao Liu, y Wei Guan. “Solving Balanced Bus Crew Rostering Problem with Genetic Algorithm.” *Sustainable Cities Development and Environment Protection.* Trans Tech Publications, 2013. 2070-2074.