Available online at www.sciencedirect.com

ScienceDirect

Applied Soft Computing 00 (2017) 1–25

Procedia Computer Science

Comparative Study of Pheromone Control Heuristics in ACO Algorithms for Solving RCPSP Problems

Antonio Gonzalez-Pardo[a,*], Javier Del Ser[b,c,d], David Camacho[a]

[a] *Escuela Politécnica Superior, Universidad Autonoma de Madrid, Madrid, Spain.*
[b] *TECNALIA, 48160, Derio, Spain.*
[c] *University of the Basque Country (EHU/UPV), 48013 Bilbao, Spain.*
[d] *Basque Center for Applied Mathematics (BCAM), 48009 Bilbao, Spain.*

Abstract

Constraint Satisfaction Problems (CSP) belong to a kind of traditional *NP-hard* problems with a high impact on both research and industrial domains. The goal of these problems is to find a feasible assignment for a group of variables where a set of imposed restrictions is satisfied. This family of NP-hard problems demands a huge amount of computational resources even for their simplest cases. For this reason, different heuristic methods have been studied so far in order to discover feasible solutions at an affordable complexity level. This paper elaborates on the application of Ant Colony Optimization (ACO) algorithms with a novel CSP-graph based model to solve Resource-Constrained Project Scheduling Problems (RCPSP). The main drawback of this ACO-based model is related to the high number of pheromones created in the system. To overcome this issue we propose two adaptive *Oblivion Rate* heuristics to control the number of pheromones: the first one, called *Dynamic Oblivion Rate*, takes into account the overall number of pheromones produced in the system, whereas the second one inspires from the recently contributed Coral Reef Optimization (CRO) solver. A thorough experimental analysis has been carried out using the public PSPLIB library, and the obtained results have been compared to those of the most relevant contributions from the related literature. The performed experiments reveal that the Oblivion Rate heuristic removes at least 79% of the pheromones in the system, whereas the ACO algorithm renders statistically better results than other algorithmic counterparts from the literature.

Keywords: Oblivion Rate, Pheromone Control, Ant Colony Optimization, Project Scheduling Problems, Constraint Satisfaction Problems, Coral Reef Optimization

1. Introduction

From a mathematical perspective there is a wide number of complex industrial problems that can be modeled as Constraint Satisfaction Problems (CSP) [1, 2]. The main techniques, algorithms and methods stemming from this research area have been utilized over the last decades in real application scenarios with an increasing level of complexity, such as scheduling and planning [3, 4], travel and car routing [5, 6], among others. Within all these applications, one of the critical issues when tackling CSP relates to the exponential growth of the computational resources needed to solve problem instances with a realistic dimensionality.

Any CSP builds upon a set of objects or variables that must be assigned a particular value while satisfying, at the same time, a number of posed constraints. Therefore, a CSP instance is composed of a set of variables (X) that can

*Corresponding author
Email addresses: antonio.gonzalez@uam.es (Antonio Gonzalez-Pardo), javier.delser@tecnalia.com (Javier Del Ser), david.camacho@uam.es (David Camacho)

take on binary, integer or real values from their domain (D). The values assigned to these objects must fulfill several constraints (C) that represent restrictions to the values that this variable could be given. Therefore, any CSP can be represented as a triple $< X, D, C >$ where X is the set of objects of the problem, D denotes the set of domains that establish the different values or alphabets for the objects described in X, and C represents the set of constraints that restrict the values that the objects can take from their domains [1, 2].

Seminal algorithmic approaches for solving this family of problems are based on the application of linear programming [7], which are effective only when dealing with small problem instances. In order to handle and alleviate part of the combinatorial complexity of the searching process, heuristic (informed) search has been extensively studied to date in the related literature, such as branch-and-bound [8], Tabu Search [9], Simulated Annealing [10], Genetic Algorithms [11] and Swarm Algorithms, among others [12]. Some of these algorithms have been specifically designed to tackle CSP paradigms, such as chronological backtracking (CBT), back-jumping, conflict-directed backtracking, learning or look-ahead techniques, e.g. Forward Checking. These approaches are usually combined with other techniques like constraint propagation techniques (i.e. node, arc and path consistency [1]) so as to modify the constraint satisfaction problem ensuring its local consistency conditions (those related to the consistency of subsets of variables and constraints). For example, a popular approach utilized to maintain consistency in a CSP is the AC-3 algorithm [13]. Previous search and consistency ensuring algorithms are usually combined with a set of heuristics to improve the system performance. During the search procedure, these heuristics can be applied to select the next variable to be assigned (minimum width, maximum degree or maximum cardinality, amongst others), or to select a value for a given variable (e.g. min-conflicts, promise, max-domain-size, weighted-max-domain-size and point-domain-size [14]).

In this application context, there are other kinds of bio-inspired algorithms that can be exploited for search and optimization purposes. Examples abound: from Evolutionary Algorithms (EA) [15] to Swarm Intelligence (SI) [16, 17], among others. These families belong to the so-called Computational Intelligence (CI) research field. As such, techniques and solvers in the SI portfolio allow for the generation of collective behavior in self-organizing systems [18], where the interactions among individuals produce collective knowledge of social colonies. Some examples of these algorithms are Ant Colony Optimization (ACO, [19, 20]), Particle Swarm Optimization (PSO, [21, 22]) or Bee Colony Optimization (BCO, [23, 24]). These search techniques are composed by a population that traverses the solution space by means of different – usually stochastically controlled – operators. When applying any of the previous EA or SI techniques to CSP problems, a goal of utmost importance is to derive a suitable representation and modeling of the CSP problem for the selected algorithm, which impacts directly on its search efficiency.

This being said, this paper gravitates on how to optimally design the constituent solution encoding strategy of the popular ACO algorithm so as to efficiently tackle Resource-Constrained Project Scheduling Problems (RCPSP), a particular subclass of CSP problems towards which an upsurge of research activity has been devoted in the last decade. Besides the design of a graph model suited to the particularities of this problem formulation, this work delves into the incorporation of side heuristic procedures – hereafter referred to as *Oblivion Rate* heuristics – to the naïve ACO scheme so as to control the exponential growth of the number of pheromones created in the system. In particular two adaptive Oblivion Rate approaches will be under analysis and subsequent discussion: a dynamic rate and a bio-inspired approach based on the Coral Reefs Optimization (CRO) solver, correspondingly coined as *Dynamic* and *CRO-based* Oblivion Rate heuristics.

The research work presented in this manuscript builds upon preliminary results and findings presented in [25], where a CSP-graph based representation for ACO algorithms was first described to solve CSP models. In [26] an initial description of this model and its application to the classical N-Queens problem were discussed; different, simple mathematical functions were used to analyze the performance of a static Oblivion Rate heuristic. In [26] the model was applied to a reduced subset of RCPSP problems extracted from the public PSPLIB repository. Only the performance of the ACO model without any Oblivion Rate was evaluated against the best-known solution. The main contributions of this work can be summarized as follows:

- A thorough analysis and description of both our model and the ACO algorithm.
- The *Dynamic Oblivion Rate*: a new heuristic that takes into account the number of pheromones created in the system to determine the percentage of pheromones to be removed.
- A new Oblivion Rate heuristic based on the bio-inspired Coral Reef Optimization algorithm.

- A complete analysis of both heuristics, i.e. *CRO-based Oblivion Rate* and *Dynamic Oblivion Rate*, using the Single-Mode datasets extracted from PSPLIB dataset.
- A detailed analysis of the ACO model when these dynamics heuristics are applied to CSP and Scheduling problems.

The remainder of the paper is structured as follows. First a brief introduction to the principles of the naive ACO algorithm can be found in Section 2, followed by a description of classical ACO models for solving CSP problems in Section 3. In Section 4 a brief introduction to Resource-Constrained Project Scheduling Problems (RCPSP) is provided as an example of CSP problems, whereas the adaptation of the proposed model to solve RCPSP problems is detailed in Section 5. The Oblivion Rate heuristic proposed in this work is described in Section 6, whose performance is discussed and compared to other algorithms in the literature in Section 7. Finally, several concluding remarks are drawn in Section 8.

2. Introduction to ACO Algorithms

As anticipated in the previous section, in this work the classical Ant Colony Optimization (ACO) algorithm [27] is applied to solve CSP problems. From a general perspective, ACO is used to find the shortest path between two nodes in a graph $G = (V, E)$, where V is the set of nodes that compose the graph, and E represents the connections between nodes. The goal of the deployed ants is to travel from a source node (the nest) to a destination node (the food) using the shortest path through the graph. Communication among ants is based on the *stigmergy* mechanism [28], which enables an indirect communication through the environment by allowing ants to deposit pheromones that will guide other ants in their corresponding search for food. When a given ant has finished its path, it returns to the nest node depositing pheromones on the followed path. The quantity of pheromones deposited in the graph will depend on the quality of the solution found by the ant at hand. This means that better solutions (better paths) will be represented by pheromones of higher value.

Pheromones are used to guide the ants during their search. This means that the pheromone value will bias the routing decision of the ants. Initially, when there are no pheromones in the graph ants select uniformly at random the next destination of its path along the graph. When the system contains pheromones, those paths with higher pheromone values will have more chances of being selected. This characteristic and the representation of the solution as pheromones give rise to a self-organizing behavior [27]. Mathematically speaking, the probability of moving from node i to node j (provided that both nodes are connected, otherwise the probability is 0) will be given by:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{u \in \mathcal{N}_i^k} \tau_{iu}^{\alpha} \cdot \eta_{iu}^{\beta}}, \tag{1}$$

where $\mathcal{N}_i^k$ is the set of feasible nodes connected to node i for ant k; τ_{ij} represents the pheromone value deposited on the edge from node i to node j; and η_{ij} represents the heuristic value of moving from node i to node j. In this expression, α and β are two parameters in charge of controlling the influence that the pheromones and the heuristic function exert over the behavior of ants. On the one hand, if $\beta \gg \alpha$, ants will be guided mainly by the heuristic function. On the other hand, if $\beta \ll \alpha$, the ants will follow the first discovered path and thus the algorithm will show a fast convergence to suboptimal routes.

To allow for the exploration of new solutions pheromones are assumed to evaporate according to a specific evaporation rate. This procedure can be understood as a decreasing rate in the pheromone values, and is computed as:

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t - 1), \tag{2}$$

where the parameter ρ defines the evaporation rate, and $\tau_{ij}(t)$ represents the pheromone value between nodes i and j in a given moment t. The evaporation rate defined as $\rho \in [0, 1]$ avoids the selection of the worst solutions found by the ACO algorithm due to their low pheromone value.

3. Ant Colony Optimization Algorithms applied to Constraint Satisfaction Problems

The application of ACO algorithms to solve CSPs has been studied in the research community over the last years [26, 29, 30]. When undertaking this task, the first step is to model the CSP as a graph over which the ACO is executed. The CSP, defined by the triple $\langle X, D, C\rangle$, is usually modeled using a graph $G = (V, E)$ such that

$$\begin{aligned} V &= \{\langle X_i, v\rangle | X_i \in X \text{ and } v \in D(X_i)\} \\ E &= \left\{\left(\langle X_i, v\rangle, \left\langle X_j, w\right\rangle\right) \in V^2 | X_i \neq X_j\right\} \end{aligned} \tag{3}$$

where nodes V represent the variable/value pairs $< variable, value >$, and E represents edges connecting those nodes whose associated variables X are different from each other.

Although several approaches have been applied to the CSP obtaining promising results [29, 30], it has not been until recently when several pitfalls were identified in the way the CSP is modeled as a graph [26]. These problems spring from the size of the resulting graph and the type of CSPs that can be represented. Regarding the size of the resulting graph, if the problem has N variables and each of them can take M different values, the resulting graph will contain $N \cdot M$ nodes. As the graph is almost fully connected, the number of edges is $(N \cdot M) \cdot (M \cdot (N-1)) = N^2 \cdot M^2 - N \cdot M^2 \cong N^2 \cdot M^2$. This observation implies that problems composed by many variables or by variables that could take on a high number of different values would become really difficult to model and almost computationally prohibitive to handle due to the size of their underlying graph.

Another limitation related to this approach is that continuous problems cannot be represented, i.e. only those problems with a finite set of values for the variables can be modelled. Examples of discrete problems are Binary Constraint Satisfaction Problems, i.e. SAT problems, or integer CSP problems with finite domains such as the N-Queens Problem. This representation has been used by Solnon in [31]. In this case each queen is defined by the row and the column of the square where it is placed and the values for both variables goes from 1 to N. Therefore, the number of nodes in each graph is $N \cdot (2 \cdot N) = 2 \cdot N^2$ and the number of edges is $(2 \cdot N^2) \cdot ((2 \cdot N^2) - N) \cong 4 \cdot N^4$. Another approach again for the N-Queens Problem is the one proposed in [30]. In this case, the graph consists of N layers and each layer contains N^2 nodes. By using this representation the graph contains a layer for each queen, and each layer is composed by all the squares of the chess board. Then each node of a given layer i is connected to all nodes of layer $i+1$ with a directional edge going from layer i to layer $i+1$, except for the last layer whose nodes are not connected to any other. Therefore, the resulting graph for N queens contains $N \cdot N^2$ nodes and $(N^2 \cdot N^2) \cdot (N-1) \cong N^5$ edges.

The CSP graph representation selected in this paper was initially proposed in [26]. This representation focuses on the reduction of the graph size resulting from the modeling of the CSP as a graph. In this approach, the size of the resulting graph is drastically reduced because each variable in the problem is represented only by one node, independently of the number of values that can be assigned to this variable (as it is traditionally represented in CSP solvers). In this manner, given any problem composed by N variables whose value can be drawn from a set of M different values, the resulting graph will have only N nodes, instead of $N \cdot M$ nodes created in classical graph models. This representation was applied to the N-Queens Problem, though it can be used in other CSP instances arising in e.g. video games [32].

The restrictions of the problem are represented in the edges of the graph. Two nodes will be connected if there is at least one restriction that involves the variables represented by the nodes. For example, given the nodes $\mathcal{N}_1$ (that represent the variable x_1) and $\mathcal{N}_2$ (correspondingly, variable x_2) there will be an edge connecting both nodes if there is at least one constraint involving the values of x_1 with the values of x_2. Under this representation approach the number of edges is drastically reduced due to the decrease of the number of nodes.

This simplification in the graph size entails a change in the behavior of the ants. In classical ACO approaches ants select the next node to visit using Expression 1. As the node itself contains the assignment, ants only deposit a small quantity of pheromone on the graph and repeat the process until they finish their execution. When adopting the new representation the ant behavior becomes more complex because ants are in charge of selecting a specific value for the variable encoded in the node (see Algorithm 1).

In Algorithm 1 ants evaluate different values that can be assigned to the variable encoded in the corresponding node (Line 1). This evaluation is performed by using the heuristic function defined for the specific problem. Then the pheromone information deposited in the graph is used in Line 2. Once the pheromone and the heuristic values are obtained, ants select one value for the variable encoded in the node (Line 3). Every ant updates its personal assignments, i.e. its local solution, and retrieves the possible nodes to visit. If there is at least one possible destination,

Algorithm 1: Ants' behavior needed in the proposed graph representation.

```
1   EvalValList ← getEvaluatedValues(currentNode)
2   PherList ← getPheromoneInformation()
3   selectedVal ← selectValue(EvalValList,PherList)
4   updatePersonalAssignment(selectedVal)
5   D ← getPossibleNodes(currentNode)
6   if (D ≠ null) then
7           node ← selectNextNode(D)
8           currentNode ← node
9   else
10          resetAnt()
11  end
```

the ant selects one of them to visit in the next time step. Otherwise, the ant finishes its execution and goes back to the nest updating, at the same time, the pheromone information that has deposited through the graph (Line 10).

Another consequence of the graph reduction is the increase in the number of pheromones deposited in the graph. Pheromones are placed on the edges of the graph because the validity of a specific value in a node depends on the given values to the rest of variables in the other nodes. Thereby, the edge connecting nodes i and j stores all pheromones related to the variables and values for these nodes. Depending on the complexity of the problem being tackled, the number of pheromones stored in the graph might saturate the system. The total number of different pheromones in an edge is proportional to the size of the domains of the variables involved in the constraint represented by the edge. That is, if $|D(var_s)|$ denotes the different values that the source variable can take, and $|D(var_d)|$ represents different values for the destination variable, the edge connecting source and destination node could store, a maximum of $|D(var_s)| \cdot |D(var_d)|$ different pheromones.

In order to reduce the number of pheromones stored in the graph an *Oblivion Rate* heuristic is incorporated to the system. This heuristic removes a subset of pheromones from the network. It is important to note that this heuristic must be carefully designed, because it impacts directly on the system performance. Consequently, the design of this heuristic depends on the problem being addressed. In this work, two new adaptive Oblivion Rate heuristics have been analyzed. Both of them, the CRO-based Oblivion Rate and the Dynamic Oblivion Rate, are described in detail in Section 6.

4. Resource-Constrained Project Scheduling Problem

This work gravitates on the use of ACO algorithms to the Resource-Constrained Project Scheduling Problem (RCPSP) [33]). The goal of this class of problems is to find an optimal schedule of the activities that compose a project subject to the availability and demand of different resources required to undertake these tasks. In mathematical terms, a project is composed by a set of activities $\mathcal{J} = \{0, \ldots, n+1\}$, a set of resource types $\mathcal{Q} = \{1, \ldots, q\}$ and a specific number of resources for each resource type $r_q \forall q \in \mathcal{Q}$. A project composed by n activities has always $n+2$ activities in the set $\mathcal{J}$ because activity 0 and $n+1$ are dummy activities explicitly included to represent the start and end of the project and do not imply any duration nor need for resources.

Each activity can be executed in one or more different modes. If activities can be executed only in one mode, the problem is labeled as *Single-Mode*. Likewise, if activities can be executed in more than one mode, the problem is called *Multi-Mode*. The modes of a given activity represent different ways to execute this activity. For the same activity, modes differ in both the duration needed to complete the activity and the set of resources required for its accomplishment. Formally, the set of different modes of the activity j is denoted as $Mode_j$, the duration of activity j executed with mode m is denoted as d_{jm} and it requires r_{jmq} units of the resource $q \in \mathcal{Q}$. Moreover, s_j denotes the time when activity j started the execution, and f_j denotes the time when such an activity has finished. Note that $f_j = s_j + d_{jm}$ because the execution of any activity cannot be interrupted.

Each project may also contain *precedence constraints* that establish relations of time interdependence between the different activities that compose the project. If a given activity j has a precedence constraint with activity i, activity

i cannot be executed until activity j has finished (i.e. $s_i \geq f_j$). By considering these constraints each activity can be assigned two lists, namely, $\mathcal{P}_j$ and $\mathcal{S}_j$, which contain its direct predecessors and successors. It is relevant to note that activity 0 is the only start activity and hence has no predecessors. Likewise, activity $n+1$ is the only end activity and consequently, has no successors.

A solution for a RCPSP is schedule for the different activities that compose the project. This schedule is composed by the start time for all the activities that compose the project, $\mathbb{S} = \{s_x \mid \forall x \in \mathcal{J}\}$ and the different execution modes for the activities. For a given schedule, the start time is the initial time for activity 0 (s_0) and the finish time is the time for activity $n+1$ (f_{n+1}). The best solution is those with a minimum makespan [34], i.e. the difference between its finishing and starting times ($f_{n+1} - s_0$). A schedule will be declared *feasible* if it satisfies the following constraints:

- Any activity must not be started before all its predecessors have finished. $s_i \geq f_j \mid \forall j \in \mathcal{P}_i, i \in \mathcal{J}$.
- At any time t, the sum of resources required for the activities in execution must not exceed the resource capacities of the project.

The mathematical formulation of the RCPSP problem also requires the definition of two further binary variables: x_{jt}^m and y_j^m. The first defines whether activity j is executed at time step t using the execution mode m, whereas the second variable indicates if activity j is executed in mode m. By considering these defined variables and the previous notation, the RCPSP problem is formulated as follows:

$$\text{Minimize} \quad f_{n+1} - s_0, \tag{4}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{jt}^m = n \quad \forall m \in Mode_j, \ \forall t, \tag{5}$$

$$\sum_{m \in Mode_j} x_{jt}^m = 1 \quad \forall j \in \mathcal{J}, \ \forall t, \tag{6}$$

$$\sum_{m \in Mode_j} y_j^m = 1 \quad \forall j \in \mathcal{J}, \tag{7}$$

$$\sum_{j \in \mathcal{J}} \sum_{m \in Mode_j} x_{jt}^m \cdot r_{jmq} \leq r_q \quad \forall t, \tag{8}$$

$$s_j \leq f_i \quad \forall i \in \mathcal{P}_j. \tag{9}$$

As stated in Expression (4), the goal of any RCPSP is to minimize the makespan of the project, i.e. to minimize the ending time for the last activity that composes the project. The constraints imposed in the problem are represented by Expressions (5) through (9). These constraints establish that all the activities must be executed – Expression (5) – and that each activity must be executed once (Expression (6)). Likewise, Expression (7) represents the requirement that each activity have to be executed in only one mode. Furthermore, as indicated by the inequality in (8), at any t the overall amount of resources demanded by the activities in execution must not exceed the total quantity of resources available for the project. Finally, the last constraint in Expression (9) takes into account precedence constraints within activities.

5. Modelling a RCPSP using a CSP Graph-Based Representation

The solution to any RCPSP consists of a schedule of the timesteps when the different activities should start. Moreover, if the activities have different execution modes the solution of the problem must include the selected execution mode for each activity. To model any RCPSP as a graph using the classical approaches, two set of nodes must be created for each activity: the first one maps the variable *startTime* to its corresponding value, whereas the second one indicates the execution *mode* for each activity. The resulting graph is extremely large due to the high number of possible *variable-value* combinations, as well as to the number of involved activities. For example, given a problem composed by X activities (each with M modes) the graph will have $(X \cdot M) + 2$ nodes related to the execution modes. Note that the "+2" corresponds to the start and end activities of the problem. These special nodes have

only one execution mode, its duration is always 0 and they do not require any resources to be completed. Regarding the nodes related to the variable *startTime* (s), an issue arises because this variable is defined as $s \in [0, \infty)$. As a workaround this variable can be upper-bounded by the due date of the project, i.e. the maximum allowed makespan. Assuming that the due date of any RCPSP is d_d, the *startTime* for any activity is redefined as $s \in [0, d_d]$. Following this rationale the variable *startTime* of any activity can take $d_d + 1$ different values, and the total number of nodes in the graph is given by

$$|V| = (X \cdot M) + 2 + X \cdot (d_d + 1), \tag{10}$$

from which, since the graph is fully connected, i.e. each node is connected to the rest of nodes, the total number of edges in the graph results in

$$|E| = |V| \cdot (|V| - 1) \tag{11}$$

By using the approach described in [26], the number of nodes in the graph is equal to the number of activities that compose the problem. The task of scheduling those activities corresponds to the assignment of the *startTime* and the execution mode for all of them.

In order to create the edges of the graph, the *Activity-On-Node* (AoN) network is used. An example for the AoN for a simple RCPSP is depicted in Figure 1(a). It can be seen in this figure that nodes represent activities and edges represent the precedence constraint that relates different activities. For example, the edge connecting activity 1 to activity 3 represents the precedence constraint that establishes the execution of activity 3 once activity 1 has been completed. Right over the node representing every activity there are two numbers, indicated as d/r, that represent the duration of the activity (d) and the number of resources needed (r) to complete this activity. Finally, the optimal schedule for the project represented in Figure 1(b) can be found in Figure 1(b). The x-axis represents the duration of the project, whereas the utilization of the resources available are represented by the y-axis. The different boxes of this figure represent the different activities, where the width of the box defines the duration, starting and ending time of the activity; and the height of the box evinces the number of resources needed to complete this activity. The number inside every box identifies the activity that is scheduled.

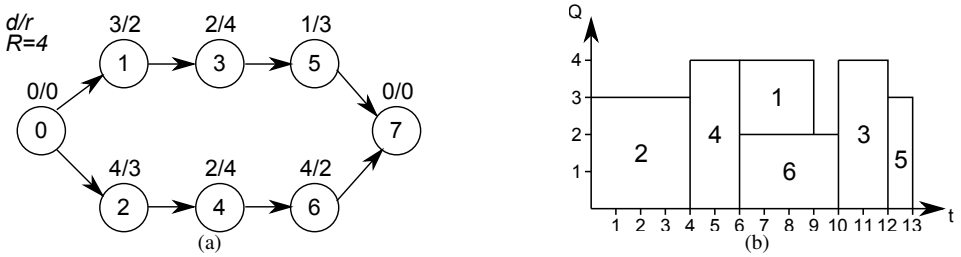

Figure 1. (left) Activity-On-Node (AoN) graph used to represent the precedence constraints in a simple RCPSP problem composed by (5 + 2) activities and 4 resources; (right) Optimal solution.

The application of a naïve ACO algorithm to this AoN is not straightforward because:

1. Parallelism is not allowed. This means that the AoN (shown in Figure 1(a)) only permits the execution of one of the branches of the graph, namely, either the execution of activities $1-3$ or $2-4-5$. Ants could not execute, at the same time, activities 1 and 2 as dictated by the optimal solution in Figure 1(b).

2. There are some activity orderings that are not represented in the AoN. For example, in Figure 1(b) once activity 4 has finished, activity 3 starts its execution, but these activities are not connected in the AoN because they are not predecessors.

For the above reasons the final graph is composed by the AoN and a set of new edges that overcome these noted issues. In order to create the final graph, the list of indirect predecessors ($\mathcal{P}^*$) and indirect successors ($\mathcal{S}^*$) for each activity must be computed. Regarding the example in Figure 1(a) the single direct successor of activity 2 is 4, i.e. $\mathcal{S}_2 = \{4\}$. The indirect successors set is composed by the direct successors of activity 4, and the direct successors of

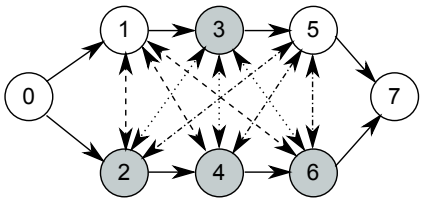

Figure 2. Resulting graph for the simple RCPSP problem showed in Figure 1(a). The solid arrows represents the precedence constraints of the project, whereas the dotted arrows are the special edges included to allow the parallelization of different activities.

all direct successors of activity 4, i.e. $\mathcal{S}_2^* = \{5, 6\}$, 5 because it is the direct successor of activity 4 (and 4 is the direct successor of activity 2) and 6 because it is the direct successor of 5.

Once the lists of indirect successors and indirect predecessors have been computed for all activities in the project, the final graph can be arranged (see Figure 2) as follows:

1. The graph contains as many nodes as activities contained in the project.
2. An edge from activity i to activity j is added if both nodes are connected in the corresponding AoN, i.e. if activity j is a direct successor of activity i ($j \in \mathcal{S}_i$). This type of edges represents the precedence constraints of the problem.
3. Another type of edges is required to allow for the parallel execution of activities (i.e. at the same time). This new type of edge does not represent constraints in the project and they connect those activities that are independent from each other. In other words, the execution of one activity does not require that the other activity has been finished. More formally, an edge from activity i to activity j is added if activity j is neither a direct successor nor an indirect successor of activity i. This means, the edge from activity i to activity j will exist if $j \notin \mathcal{S}_i \wedge j \notin \mathcal{S}_i^*$.

Finally, the new decision graph created for the RCPSP showed in Figure 1(a) is depicted in Figure 2. In this graph any ant located in activity i can execute any activity j if the following conditions are met:

1. There exists an edge from activity i to activity j.
2. All direct and indirect predecessors of activity j have finished, i.e. $currentTime \geq f_x \; \forall x \in \mathcal{P}_j, \mathcal{P}_j^*$.
3. There are enough resources to execute activity j in any of its execution modes.

Once the graph has been created, the next step is to define the ants' behavior for solving the problem. The goal is to minimize the makespan of the project, i.e. to minimize the finish time $f_(n+1)$ of the last activity assuming, without loss of generality, that $s_0 = 0$. The ant behavioral procedure is described in Algorithm 2.

Algorithm 2: One-step ant behavior in the ACO algorithm applied to the RCPSP.

```
1   A ← getPossibleActivities(currentAct)
2   while A ≠ NULL do
3       act ← selectRandomActivity(A)
4       M ← getModes(act)
5       values ← evaluateModes(M)
6       PherList ← pheromoneInformation(act)
7       selectedMode ← selectMode(values,PherList)
8       updateLocalSolution(act,selectedMode)
9       currentAct ← act
10      A ← getPossibleActivities(currentAct)
11  end
```

In the previous algorithm ants compute the possible activities that can be executed (Line 1), and check whether there is any pair (activity, mode) that can be executed taking into account:

- The number of resources that are used in the current step given the local scheduled created by the ant.
- The assigned activities that have finished their execution.

If one or more activities can be executed ants select randomly one of them. Then ants assign an execution mode for the selected activity (Line 7) taking into account the heuristic of the problem (Line 5) and the pheromones (Line 6). Given execution mode m of a given activity i, the heuristic function is defined as

$$\tau_{im} = \gamma \cdot d_{im} + \mu \cdot \left(\sum_{q \in Q} r_{iq} \right), \tag{12}$$

where it can be noted that the heuristic value is composed by the time required to finish the activity in this mode ($d_i m$) and the number of resources needed in the execution ($\sum_{q \in Q} r_{iq}$). Parameters γ and μ are introduced to tune the influence that the duration and the resources have on the heuristic function. The goal of the problem is to minimize the makespan of the schedule, therefore execution modes with lower duration should be preferential for execution with respect to other modes that require more time to be completed. Once the ant at hand has selected the mode for the current activity, it updates its local solution (Line 8). The update consists of adding the current activity with the selected mode to its local schedule.

Finally, every ant must assign the resources demanded by the activity. This is a key concept in the ant behavior, because the ant will execute a new activity only if it has enough resources to do so. This check is performed in Line 10: once the ant has updated its position in the graph it retrieves the set of possible activities to be executed. If there is at least one activity, the ant repeats the process again. This approach allows ants to execute activities concurrently, i.e. two or more activities can be executed at the same time if the sum of needed resources does not exceed the amount of unassigned resources of the project.

6. The Oblivion Rate Heuristic

The reduction in the size of the graph affects the number of pheromones created in the ACO model. In the selected model, pheromones are placed on the edges of the graph because the validity of a specific value in a node depends on those given to the rest of nodes. Hence, the edge connecting node i to node j stores all the pheromones related to the variables and values for these nodes.

Depending on the problem it could be the case that the number of pheromones in the graph collapse the system. The total number of different pheromones in an edge is proportional to the size of the domains of the variables involved in the constraints represented by the edge. That is, if $|D(var_s)|$ denotes the different values that the source variable can take, and $|D(var_d)|$ represents different values for the destination variable, the edge connecting source and destination node could store at maximum $|D(var_s)| \cdot |D(var_d)|$ different pheromones.

In every step of the algorithm (see Algorithm 3) a heuristic approach will be used to remove a variable number of pheromones. This heuristic, named *Oblivion Rate*, is designed to avoid that the number of pheromones stored in the graph could generate performance issues during the execution of the algorithm due to an overload of memory consumption in the system. We will hereafter refer to these performance problems as *saturation*. The definition of Oblivion Rate is a critical for the model because it will determine the behavior of the ACO algorithm. This heuristic must satisfy several requirements:

1. The oblivion function must reduce the number of pheromones contained in the graph in order to avoid the saturation of the system.
2. The number of pheromones should decrease gradually for a well-behaved, smooth convergence of the search procedure.
3. There must be always pheromones in the graph because if in any step the system removes all the pheromones, the "history" of the execution would be lost, hence it would be similar to launching the execution from scratch.
4. In order to converge to potentially good solutions, the system should remember the best set of pheromones (those with the highest pheromone values).

The Oblivion Rate is executed at every step of the algorithm following Algorithm 3. The Oblivion Rate compiles all pheromones in the graph and sorts them by the pheromone value. Then it computes the number of pheromones that must be forgotten (n) using an Oblivion percentage (p). This parameter is a key concept for the proposed heuristic, as it establishes the percentage of pheromones that must be removed from the system. Finally, the n pheromones with worst values are removed from the system.

Algorithm 3: Description of the *Oblivion Rate* heuristic.

```
1 P ← getAllPheromones()
2 sortPheromones(P)
3 p ← getOblivionPercentage()
4 n ← P · p
5 i ← 1
6 while i ≤ n do
7 |   removeWorstPheromone(P)
8 |   i ← i + 1
9 end
```

In this work two different Oblivion Rate heuristics have been analyzed and applied to RCPSP problems: the *Dynamic Oblivion Rate*, and the *CRO-based Oblivion Rate*.

6.1. Dynamic Oblivion Rate

This Oblivion Rate function takes into account the number of pheromones created in the system to determine the percentage of pheromones that are to be removed. This heuristic is driven by a negative exponential function that uses an adaptive exponent value that depends on the *saturation* of the system. This saturation is defined taking into account the total number of possible pheromones in the system to compute the percentage of pheromones that will be removed. This heuristic applied at step t is defined as:

$$R(t) = 1 - \frac{1}{t^{S(t)}}, \tag{13}$$

where $S(t)$ represents the saturation of the system at step t. This saturation takes into account the number of pheromones created in the graph ($P(t)$) and the total number of pheromones that can be created (*MaxPher*), yielding

$$S(t) = \frac{P(t)}{MaxPher}. \tag{14}$$

In order to compute the maximum number of pheromones, Expression 15 provides an upper bound value using the classical graph-based representation described in Section 3. This upper bound is computed by estimating the number of nodes and edges that the graph would contain by using the classical representation, i.e.

$$MaxPher(j, m) = j \cdot m \cdot (j-1) \cdot m = j \cdot m^2 \cdot (j-1). \tag{15}$$

Using this classical representation, each activity can be executed in m different ways, hence the classical graph would have $j \cdot m$ nodes. The resulting graph is highly connected and only the nodes representing the same variable are not connected. This means that each node is connected to $m \cdot (j-1)$ nodes and the total number of edges is $j \cdot m \cdot (j-1) \cdot m = j \cdot m^2 \cdot (j-1)$. This overall number of edges will be adopted as the upper bound for the maximum number of pheromones in our model.

6.2. CRO-based Oblivion Rate

The Coral Reef Optimization (CRO) is a novel heuristic approach based on the formation and reproduction of the coral reefs [35]. The main problem of corals is to find a place on the reef where they can settle and grow. The space in the reef is a limited resource [36] when compared against the high reproduction rates typically featured by corals. For

this reason, corals must fight to obtain a place on the reef [37, 38]. The result of this fight is that some individuals die because they cannot defend the place where they are located, or because they do not find any place on the reef where to settle.

The CRO algorithm emulates the life of corals in the reef during a maximum of G_{max} generations. For each generation, corals reproduce and fight for the space as shown in Algorithm 4. The algorithm includes 1) a replica control procedure that avoids the best coral to populate the whole reef; 2) a reproduction phase where new corals (solutions) are created; 3) the *larvae* setting where newly produced solutions try to settle in the coral; and finally 4) a predation procedure by which worse corals can die and disappear from the reef.

Algorithm 4: General Coral Reef Optimization algorithm

```
1   coralInitialization
2   g ← 0
3   while g < G_max do
4           Reproduction phase
5           Evaluation of the new corals
6           Larvae setting
7           Predation process
8           Replica Control Procedure
9           g ← g + 1
10  end
11  Return the best coral in the reef
```

The reef (Λ) is defined by a set of corals ($\Lambda = \{C_1, ..., C_n\}$), where each coral represents a possible solution (I) for the problem addressed. An evaluation function measures the *health* or goodness of each coral $f(C_i) : I \Rightarrow \mathbb{R}$. This evaluation permits to compare different corals, and is used in several constituent processes of the solver such as the selection or reproduction procedures. Corals are located in an empty $\mathcal{N}x\mathcal{M}$ square grid that represents the reef and they are randomly initialized. The number of corals created in the initialization of the coral (Line 1) is defined by the size of the reef and a parameter that determines the initial population density (ρ). Thus, the initial number of corals in the reef is computed as $\mathcal{N} \cdot \mathcal{M} \cdot \rho$.

In this research work the *larvae* setting process is utilized to control the number of pheromones in the system. This process allows new corals to find a place into the reef. New corals are not placed on the reef immediately upon their creation. When a new coral larva is created, it is kept in a pool where the rest of the larvae will be temporally stored. Once all the reproduction processes have finished and all larvae have been added to the aforementioned pool, larvae competitively attempt at settling in a randomly selected square space of the reef. If the square is empty the coral is settled; on the contrary, if the position is not empty the new coral will be settled only if its health is better than the health of the coral already placed on the selected location. In such a case, the old coral dies and is removed from the reef. In the case that the new coral has worst value than the one placed on the reef the new coral starts the process again looking for a new place to be settled. This process is repeated during N_t attempts, where N_t is a parameter defined before the execution of the algorithm. If the new coral is not able to find a location after N_t attempts, the new coral dies. This process is described in Algorithm 5, where N_c is the set of new larvae that is going to search a position

in the reef.

Algorithm 5: Larvae setting procedure

```
   Data: N_c pool containing the new larvae
1      foreach l ∈ N_c do
2          settled ← false;
3          t ← 0;
4          while settled = false ∧ t < N_t do
5              pos ← selectRandomPosition;
6              C ← coralInPosition(pos);
7              if C = NULL then
8                  SettleLarvae(l, pos);
9                  settled ← true;
10             else
11                 if f(l) IsBetterThan f(C) then
12                     Remove(C);
13                     SettleLarvae(l, pos);
14                     settled ← true;
15                 else
16                     t ← t + 1;
17                 end
18             end
19         end
20         if t = N_t then
21             Remove(l);
22         end
23     end
```

In Line 11 the health comparison between the coral settled in the reef and the new coral larvae depends on the goal of the problem. If the goal is to maximize the health of the corals, the comparison would be $f(l) > f(C)$ rather than the generic *IsBetterThan*. Likewise, if the problem is to minimize the health function, the comparison would be $f(l) < f(C)$, where l is the new coral larvae and C is the coral placed on the selected position of the reef.

The adaptation of CRO to the Oblivion Rate heuristic simplifies the algorithm because part of the procedures contained in the nominal CRO are not required to control the growth of pheromones in the ACO model. Under this adaptation each coral represents a pheromone created by the ants, whereas the health of a coral corresponds to the value of the pheromone. This work uses a simplified version of the CRO algorithm because the corals (or pheromones) are produced by ants and there is no reproduction procedure. Therefore, there is no need for including replica control procedures because if any ant adds a coral (or pheromone) that already exists, the health of the existing coral is updated.

Bearing the above remark in mind, the ACO graph features a reef where the different corals – namely, pheromones – are stored. Initially, the reef is empty because there is no pheromone in the graph, and no initialization procedure is hence needed. Ants start their execution and place some pheromone in the reef. At this point, pheromones start with the *larvae setting* procedure: they try to find a place on the reef within a given number of trials determined by parameter. If the pheromone finds an empty place on the coral, the pheromone is settled. But if the position contains an old pheromone both pheromones fight for space. This fight consists of a comparison of the health of the pheromones competing for space. If the old pheromone has better health than the new one, the latter keeps looking for a new position. In the case where the new pheromone is better than the one found in the reef, the new pheromone is settled in the position and the old one is removed. Finally, the depredation process generates free spaces in the reef by removing those corals with the lowest health.

Problem	#Instances	#Activities	#Modes
j30.sm	480	30	1
j60.sm	480	60	
j90.sm	480	90	
j120.sm	600	120	
j10.mm	536	10	3
j12.mm	547	12	
j14.mm	551	14	
j16.mm	550	16	
j18.mm	552	18	
j20.mm	554	20	
j30.mm	640	30	
m2.mm	481	16	2
m4.mm	555		4
m5.mm	558		5

Table 1. Description of the different problems available in the RCPSP dataset.

7. Experimental Results

The main goal of the experimental results discussed in this section is to analyze the performance of both Oblivion Rate heuristics described previously when tackling RCPSP problems. The performance will be measured as the quality of the solutions found by the ACO model, as well as the number of pheromones stored in the system. The dataset used in this work has been extracted from the PSPLIB library [39] by selecting different types of problems, i.e. problems that differ in the number of activities and complexity. An overview of the different RCPSP problems that compose the PSPLib dataset can be found in Table 1. For each problem, the number of instances (different realizations of the same problem), the number of activities that compose the project and the number of execution modes for each activity are given. All experiments have been carried out on a Xeon Haswell-EP E5-2650 V3 2.3 GHz with 10 cores and 256 GB of memory DDR4/2133 MHz.

Despite the diversity of problem instances within the PSPLib dataset, we will focus exclusively on the most difficult problems (which are those problems composed by Single-Mode instances). The selected problems (i.e. `j30.sm`, `j60.sm`, `j90.sm` and `j120.sm`) have been solved using the proposed model without Oblivion Rate (*called normal ACO*), with the *Dynamic Oblivion Rate* and with the *CRO-based Oblivion Rate*. There are several parameters of the ACO algorithm that have been fixed for all the experiments carried out in this work. One of these parameters is the evaporation rate that has been fixed to 5% ($\rho = 0.05$). As described in Section 5, Expression 12 that defines the ants' behavior depends on two parameters, γ and μ. In order to fix the values for both parameters the ACO model has been executed to solve 10% of the different PSPLIB datasets – selected uniformly at random – using a value grid for γ and μ. These values fall within the range from 0 to 1 in such a way that $\gamma + \mu = 1$ because both parameters establish the influence of the duration of the activity and the usage of the resources in the decisions made by ants. In all the cases the configuration of the ant colony is the same: the stopping criteria is 1000 solutions, the colony is composed by 10 ants, α is 1, and the value for β is 2. For each configuration, each problems is solved 10 different times, we measure the average deviations (%) from the critical path lower bound and the best parameter configuration is the one that provides lower deviation. The results are showed in Table 2, but before analyzing these results, a brief description of the measure used is provided.

The average deviations (%) from the critical path lower bound is the most common metric used by the research community for comparing the different algorithms. The critical path of a project is defined as the longest sequence of activities to be completed in time to avoid delays during the project execution. This means that if any of the activities that belongs to the critical path is delayed, the whole project will suffer this delay. An example about this concept is shown in Figure 3. This figure contains an Activity-On-Node network where the nodes represent the activities of the project, the edges represent the precedence constraints, and the number above the nodes denotes the duration of the corresponding activity.

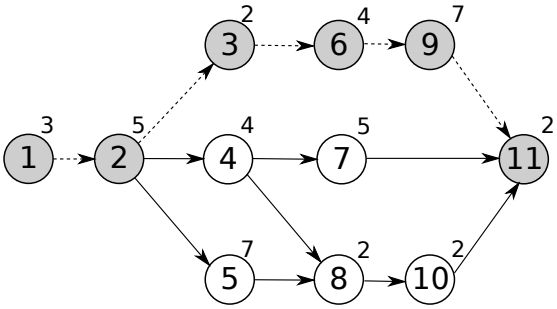

Figure 3. AoN graph of a project composed by 11 activities where the numbers above the nodes represent the time needed to complete the activity. The colored nodes and the dashed edges compose the critical path of this project.

γ	μ	**Average deviation** (%)
0	1	3.0439 ± 0.6
0.1	0.9	2.9262 ± 0.81
0.2	0.8	2.9716 ± 0.73
0.3	0.7	3.0273 ± 0.41
0.4	0.6	**2.7171 ± 0.11**
0.5	0.5	3.0475 ± 0.67
0.6	0.4	2.9574 ± 0.38
0.7	0.3	3.1471 ± 0.57
0.8	0.2	2.8777 ± 0.84
0.9	0.1	2.9088 ± 0.53
1	0	2.9893 ± 0.66

Table 2. Average deviation (%) from the critical path lower bound for 48 instances of `j60.sm` using different values for γ and μ. This table also contains the standard deviation of the Average deviation. The bold value corresponds to the best configuration that is the one that provides the lower Average deviation from the critical path lower bound and also, the lowest standard deviation.

As can be observed in this figure, there are three different branches in the project: the first branch is composed by activities 1, 2, 3, 6 and 9, the second branch contains the activities 1, 2, 4 and 7, and finally, the last branch involves activities 1, 2, 5, 8, and 10. If we analyze the time needed to complete the different branches, the first branch needs 21 time units, the second branch requires 17, and the third branch needs 19 time units. This means that activity 11 cannot start until 22 time units are reached, because any delay in the activities that compose the first branch will result in a delay in the whole project. Nevertheless, small delays on the other two branches will not alter significantly the completion of the project. Based on this analysis, the first branch will be the "critical path" of the project.

In this work, the critical path lower bound (LB) for each problem is known a priori and given in the PSPLIB webpage. The deviation (%) from this critical path of a specific problem (P_i) given a solution S (denoted as $desv(S, P_i)$) can be computed as follows:

$$desv(S, P_i) = \frac{S_m - LB_i}{LB_i} \cdot 100 \tag{16}$$

where P_i represents the problem to be solved, LB_i is the critical path lower bound for problem i, and S_m represents the makespan of the corresponding solution.

In the four datasets under analysis the minimum deviation was obtained when the value of γ and μ was fixed to 0.4 and 0.6, respectively. This evaluation is shown in Table 2, where the average deviations from the critical path obtained for the `j60.sm` dataset are listed with their corresponding standard deviation.

Problem	Normal ACO	Dynamic Oblivion	Reduction pct.
j30.sm	1352	225	83.35%
j60.sm	3789	767	*79.75 %*
j90.sm	11262	1791	**84.09 %**
j120.sm	24781	4813	80.57%

Table 3. Comparison of the number of pheromones created in the system with and without using the *Dynamic Oblivion Rate*. The value in italics corresponds to the lowest reduction percentage, whereas the higher reduction is highlighted in bold.

Parameter	Value
Number of Ants	10
Steps	1000
α	1
β	2
ρ	0.05
Repetitions	10

Table 4. Configuration of the ACO model used to evaluate the computational resources needed by the pheromones.

7.1. Experiment 1: Memory Consumption Analysis

Once the parameters of the ACO model have been optimized, a first set of experiments focus on the reduction in the number of pheromones handled by the system when the Dynamic Oblivion Rate is used. On this purpose, the number of pheromones created in the system without Oblivion Rate is compared to that created with the Dynamic Oblivion Rate. These results are shown in Table 3.

As shown in the above table, the reduction obtained in the system when the Dynamic Oblivion Rate is used is around 80%. This yields that the computational resources (i.e. memory) needed to solve the problems decreases dramatically when the Dynamic Oblivion Rate is used. In order to measure the computational resources required by the ACO algorithms in the benchmark, we have compared the memory used to store the pheromone table by the Normal ACO (i.e. the ACO algorithm without Oblivion Rate) to the memory required by the model using the Dynamic Oblivion Rate. For this experiment we have analyzed the size of the pheromone table required by both algorithms for each execution step. The configuration for the ACO models can be observed in Table 4.

The size of the pheromone table registered for the different datasets can be seen in Figure 4. In this Figure the blue lines represent the memory required by the Normal ACO algorithm, whereas the memory consumption performed by the ACO model with the Dynamic Oblivion Rate is described by the red lines. As it can be observed in the four datasets, there is a significant improvement in terms of the size of the pheromone table required for solving the problem when the ACO model with the Dynamic Oblivion Rate is utilized. In the early steps of the algorithm the size of this table is similar for both algorithms. However, as the algorithm iterates further remarkable differences in terms of size of the pheromone table appear between solvers. It can be also observed that this difference depends on two different factors: the problem size (i.e. the complexity of the problem) and the execution steps of the algorithm.

This important reduction in the number of pheromones generated by the system could affect to the diversity of the solution. For this reason, it is important to analyze the diversity of the solutions in order to check whether the inclusion of the Oblivion Rate reduce the diversity and thus the solutions found by the model are guided by the Oblivion Rate.

Initially, the diversity of the solutions should not be affected by the use of the *Oblivion Rate*. This heuristic is in charge of removing some pheromones from the graph to alleviate the cost of storing all these structures in memory. Also, the Oblivion Rate heuristic removes from the graph those pheromones with the lowest value, i.e. only pheromones representing the worst solutions are the ones selected. This characteristic ensures the convergence of the algorithm. Nevertheless, solutions not represented by the pheromones, or solutions whose pheromones have been already removed, can also be discovered again due to the ACO algorithm contains a stochastic component.

In order to tackle this issue we have measured the mean number of different solutions found by the different

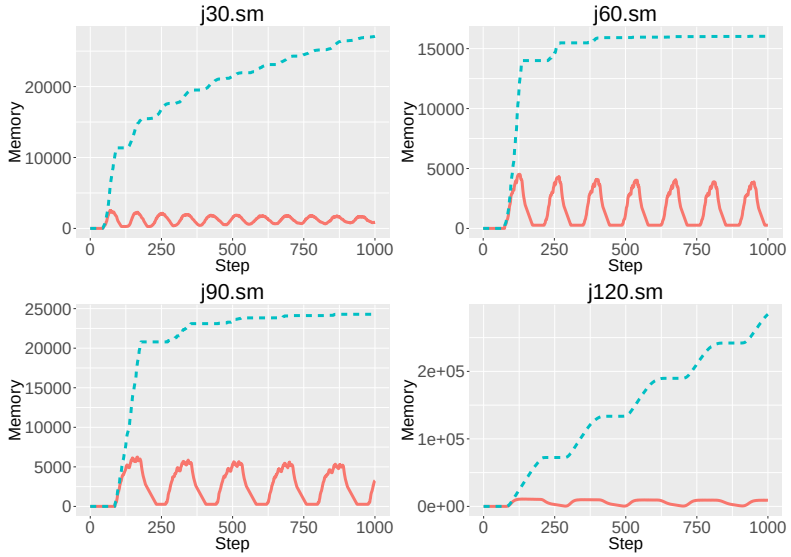

Figure 4. Size of the pheromone table required for the datasets `j30.sm`,`j60.sm`,`j90.sm` and `j120.sm`. For each dataset, we have registered the size of the pheromone table required by the Normal ACO (dashed lines) and the ACO model with the Dynamic Oblivion Rate (solid lines). The configuration for both ACO models is provided in Table 4.

models for the datasets used in this work. As it can be observed in Table 5, there are not any significant difference in terms of "diversity of solutions" among the different models used in this work. For this reason, we can conclude that the the Oblivion Rate does not affect to the diversity of the solutions found by the ACO algorithm.

Problem	**Normal ACO**	**Dynamic Oblivion**	**CRO-based Oblivion**
`j60.sm`	18.66	18.70	18.73
`j90.sm`	20.79	21.80	21.88
`j120.sm`	45	43.78	43.85

Table 5. Mean number of different solutions found by the different ACO models.

7.2. Experiment 2: Analysis of the Quality of the Solutions

This second experiment analyzes the correspondence between the reduction in the number of pheromones and the quality of the solutions found by the solver (given by the makespan). In this new experiment the best makespan published in the related literature is compared to the average makespan obtained by the *Normal ACO model*, the model using the Dynamic Oblivion Rate and the model using a CRO-based Oblivion Rate.

In this experiment three different ACO algorithms are considered. The first one is the classical ACO algorithm without any Oblivion Rate, the second one is an ACO algorithm with a Dynamic Oblivion Rate and the last one is the ACO algorithm using the CRO-based Oblivion Rate. These three ACO algorithms share the same parameter configuration for the colony, which is given in Table 6.

With this configuration, the different ACO algorithms are executed until the first 1000 solutions are found for each instance of the dataset. As such, for the dataset `j30.sm` composed by 480 instances, each ACO algorithm will stop once 480000 solutions have been found.

In order to fairly compare the results of the Dynamic and the CRO-based Oblivion Rates, the size of the coral reef has been defined as the square root of the maximum number of pheromones created by the Dynamic Oblivion Rate for the same problem. When using the Dynamic Oblivion Rate, the simulation framework records the maximum number of pheromones created for each instance of the dataset. In this way, when the ACO algorithm uses the CRO-based

Parameter	Value
Number of Ants	10
α	1
β	2
ρ	0.05
γ	0.4
μ	0.6

Table 6. This table contains the configuration of the different ACO algorithms executed in the second experiment.

Dataset	Best Makespan [40]	Normal ACO	Dynamic Oblivion	CRO-based Oblivion
j30.sm	58.99 ± 14.07	**59.98±14.66**	59.88 ± 16.2	59.88 ± 14.61
j60.sm	79.8 ± 17.42	82.09 ± 20.93	**82.44±20.26**	82.48 ± 20.30
j90.sm	94.94 ± 20.57	99.45 ± 25.13	98.85 ± 25.27	**98.82±25.31**
j120.sm	122.19 ± 39.75	138.44 ± 48.62	**134.74±48.38**	136.59 ± 49.19

Table 7. Results for the PSPLIB dataset, composed by the average minimum makespan published by the research community [40], the average minimum makespan obtained by the proposed model without using the *Oblivion Rate* (Normal ACO), using the *Dynamic Oblivion Rate*, and using the *CRO-based Oblivion Rate*. The best value obtained for the different datasets are highlighted in bold.

Oblivion Rate, the simulation environment retrieves the maximum number of pheromones produced for the problem instance to be solved and computes the size of the reef. Then, the number of trials that will be set to a new coral to find a place on the reef is defined as:

$$Trials = (ReefSize \cdot 0.1) + 1 \tag{17}$$

where *ReefSize* is the size of the reef defined as the square root of the maximum number of pheromones created by the Dynamic Oblivion Rate.

In order to analyze the performance of the model with the different Oblivion Rates, the makespan rendered by the different approaches is compared to the minimum makespan reported by the research community for each problem in the benchmark [40]. The different problems are composed by a large number of instances (see Table 1), for this reason Table 7 contains the average minimum makespan published in the literature (i.e. the mean makespan published for all the instances that compose each problem), the average minimum makespan obtained by the system without the Oblivion Rate and the average minimum makespan obtained with the Oblivion Rate schemes described in this paper. The minimum makespan obtained by the different ACO algorithms is quite similar in all cases, and the results are very close to the minimum makespan published by the research community. These results are even more interesting in light of the high pheromone reduction percentage obtained when the Oblivion Rate is used (see Table 3).

When analyzing the standard deviation of the average makespan it can be observed that when the size of the problem increases – and so does its complexity – the standard deviation increases as well. However, the values of the standard deviations obtained by the model using the different Oblivion Rates are quite similar to those obtained without using this heuristic.

For the sake of statistical consistency of the conclusions drawn above, statistical tests must be performed in order to assess whether the obtained differences between Normal ACO, Dynamic and CRO-based Oblivion in terms of average and standard deviation of the estimated makespans must be deemed statistically significant. In particular this significance has been analyzed by means of a *Wilcoxon rank sum test*, which is a non-parametric hypothesis test that allows checking the null hypothesis that two given samples correspond to continuous distributions with equal medians. It is similar to the widely utilized t-test, but does not require any assumption made on the underlying distributions of the compared data. By using this test we have found that the comparison in terms of mean and standard deviation of the normal ACO, Dynamic Oblivion and CRO-based Oblivion cannot be claimed as conclusive. In other words, the hypothesis that all result samples come from distributions with equal medians cannot be rejected. This interesting

Problem	Num. Instances	Normal ACO	Dynamic Oblv.	CRO-Obv
		Total (%)	Total (%)	Total (%)
j30.sm	480	**280** (**58.33%**)	**280** (**58.33%**)	279 (58.13%)
j60.sm	480	243 (50.63%)	257 (53.54%)	**262** (**54.58%**)
j90.sm	480	257 (53.54%)	256 (53.33%)	**258** (**53.75%**)
j120.sm	600	56 (9.33%)	53 (8.83%)	**55** (**9.17%**)

Table 8. This table contains the number of instances that compose the j30.sm, j60.sm, j90.sm and j120.sm problems from PSPLIB, and for each ACO model analyzed in this paper this table shows in how many instances the corresponding ACO model obtains the best makespan published and its corresponding percentage. The best value obtained for the different datasets are highlighted in bold.

result has a twofold interpretation: on the negative side, there is no statistically relevant difference in makespan performance between the Dynamic and the CRO-based Oblivion Rate; on the positive side, the dramatic pheromone reduction resulting from the application of the Oblivion Rate does not have any statistically valuable impact on the computed makespan scores.

Analyzing the results showed in Table 7, it seems that the ACO algorithm never finds the optimal solution for the different instances of the PSPLib dataset. For this reason, we have carefully delved into these results. First, we have quantified over how many problem instances the different ACO models obtain the *best makespan* reported by the research community. Table 8 shows for each problem, the number of instances that compose each problem and for each ACO model, the number of instances where the corresponding model obtains the best makespan, as well as its relative percentage.

In this table it can be noted that the Normal ACO, the ACO model using Dynamic Oblivion Rate and the ACO model with the CRO-based Oblivion Rate obtain the best makespan in more than 50% of the instances within j30.sm, j60.sm and j90.sm by exploring only the first 1000 obtained solutions for each problem instance. For the most complex problem (j120.sm), the number of instances where the algorithms in the benchmark obtain the best makespan decreases significantly. Nevertheless, the number of instances where the best makespan is reached for any problem remains almost constant independently of the ACO model used. This fact suggests that the usage of the Oblivion Rate does not have any impact on the quality of the solution found by the model.

We have also analyzed the deviation from the best makespan and the best solution found by the algorithms for the problems belonging to the Single-Mode datasets (i.e. j30.sm, j60.sm, j90.sm and j120.sm). In this case, only the performance of the ACO model using the Dynamic Oblivion Rate and the CRO-based Oblivion Rate is analyzed. Once these algorithms have found the first 1000 solutions, we have computed the deviation between the best solution found and the optimal makespan known for each problem instance. Then, we have analyzed the histograms describing the distribution of the aforementioned deviation for each problem.

Such histograms are shown in Figure 5. When assessing these figures, it can be noticed that for j30.sm, j60.sm and j90.sm the distribution graphs using both algorithms are centered around *Deviation* = 0% which means that the best solutions found by these algorithms for the corresponding problems are close to the best makespan. Besides, these histograms support the main conclusions inferred from the information shown in Table 8: both algorithms find the best solution in the majority of problems belonging to j30.sm, j60.sm and j90.sm. In addition, the complexity of the problems degrades the quality of the solutions; as the complexity of the problem increases, the histogram has a bigger right tail that penalizes the optimality of the solutions discussed in Table 7, and also the frequency decreases.

Finally, we have also registered the time needed by each ACO algorithm to find the first 1000 solutions for each problem instance of the different datasets. The goal is to analyze whether the usage of any Oblivion Rate provides any advantage in terms of computational time needed to solve the different problems.

Table 9 contains the mean time (seconds) needed by each algorithm to find the first 1000 solutions in each instance for the datasets. The time needed to find the first 1000 solutions is higher when the ACO model does not use any Oblivion Rate. This degradation is due to the high number of pheromones deployed in the solution graph when the Oblivion Rate is not used. These pheromones saturate the system and penalize the execution of the algorithm in terms of computation time.

Besides the time needed to find the solutions, we have observed that the execution of the Oblivion Rate does not affect to the quality of the solutions found. This means that the quality of the solutions is the almost the same

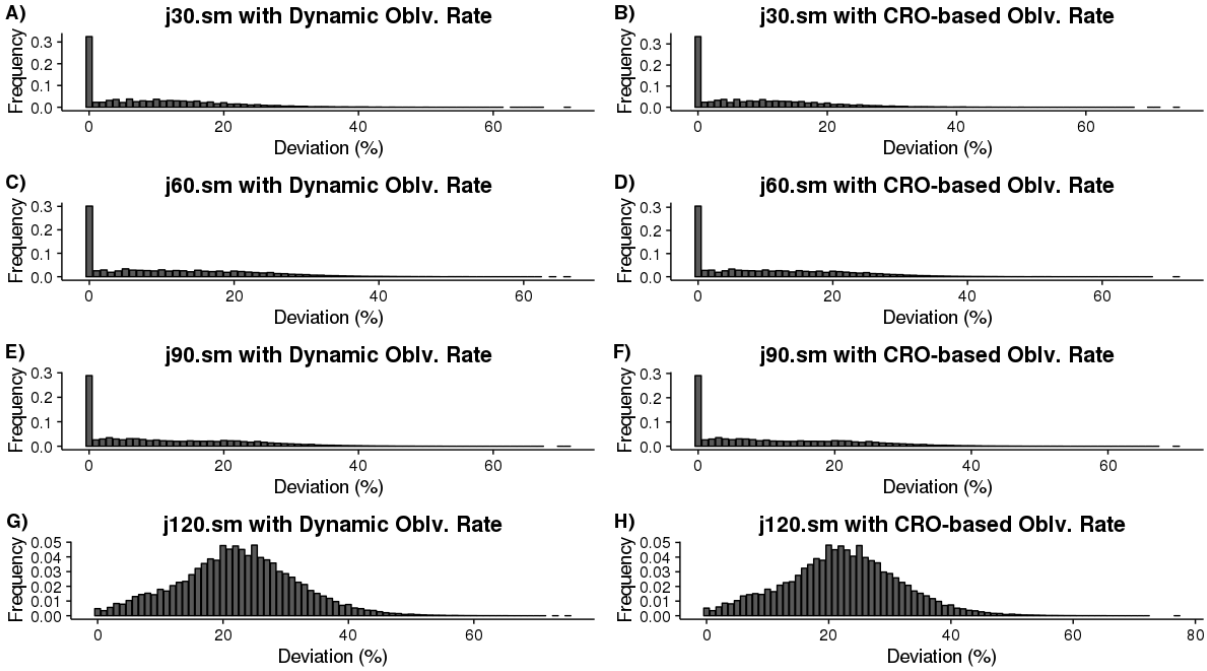

Figure 5. Distribution of the best solutions found by the ACO model when the Dynamic Oblivion Rate (Figures a, c, e and g) and the CRO-based Oblivion Rate (Figures b, d, f and h) are used for solving `j30.sm`, `j60.sm`, `j90.sm` and `j120.sm`. The x-axis represents the deviation (%) from the best makespan.

	`j30.sm`	`j60.sm`	`j90.sm`	`j120.sm`
Normal ACO	2.78	85.96	293.92	2609.08
Dynamic Oblv. Rate	0.68	2.41	5.23	14.05
CRO-based Oblv.	0.89	3.26	7.69	18.51

Table 9. This table shows the average time required (Seconds) by the different ACO algorithms to find the first 1000 solutions in the different instances of the datasets.

independently of the Oblivion Rate used (if any). For all of this, the next experiment performs a comparison against some of the well-known algorithm extracted from the State-of-the-Art, but only the ACO model using the Dynamic Oblivion Rate and the CRO-based Oblivion Rate have been taken into account.

7.3. Experiment 3: Comparison to the State-of-the-Art

Once the performance of the proposed algorithm has been analyzed in terms of both, the reduction in the number of pheromones and makespan of the solutions, the third experiment compares the performance of the proposed algorithm versus some of the algorithms existing in the related literature. In this context, we have discarded those instances belonging to `j30.sm` because results reported in the literature for this family of RCPSP problems do not allow for any margin of improvement due to the relatively low complexity of its compounding problems. For this reason, we will focus on the results obtained from `j60.sm`, `j90.sm` and `j120.sm` datasets. To the knowledge of the authors, and according to [41], only `j60.sm` and `j120.sm` have been under discussion within the research community. Nevertheless, for the sake of completeness results for `j90.sm` obtained with the different ACO models have been included in Table 11.

We have compared our algorithm to the best heuristic algorithms found in the literature. This set of heuristics include works that use Genetic Algorithms (GA) [42, 43, 44], ACO algorithms for solving the RCPSP problem [45], and the majority of these works include other methods such as Forward-Backward improvement (FBI), or the Latest

Algorithm		Max. #Schedules		
		1000	5000	50000
ACO - Dyn. Oblv. (min)		3.51	3.18	2.82
ACO - CRO-based Oblv. (min)		3.55	3.17	2.80
ACO - Dyn. Oblv. (mean)		11.51	11.51	11.50
ACO - CRO-based Oblv. (mean)		11.41	11.40	11.40
HGA	[42]	–	11.14	**10.63**
GA - bi-population	[43]	–	10.95	10.68
GA-MBX	[44]	**11.33**	**10.94**	10.65
ACOSS	[45]	11.75	10.98	10.67
GAPS	[48]	11.72	11.04	10.67
enhanced scatter search	[46]	–	11.10	10.71
scatter Search - FBI	[47]	11.73	11.10	10.71
Hybrid GA	[49]	11.56	11.10	10.73
AHS	[42]	–	11.33	10.85

Table 10. Average deviation (%) from the critical path lower bound for `j60.sm`. Each column highlights in bold the best result that corresponds to the lower deviation obtained by the different algorithms.

Finish Time (LFT) [46, 47] priority rule, among other. In this new set of experiments we have used the ACO model using the Dynamic Oblivion Rate and the proposed CRO-based Oblivion Rate. The reason being that as it has been stated in the previous set of experiments, the use of the Oblivion Rate reduces significantly the number of pheromones in the system and the quality of the solutions is not affected.

The stopping criteria for the algorithm is the number of schedules (solutions) found. As stated in [41], this stopping criteria is based on the assumption that the computational effort for constructing one schedule is similar among different heuristics. Another advantage of this stopping criteria is that it is independent of the programming language used, or the hardware characteristic of the computer where the algorithm is executed. As for the parameters of the Oblivion Rate, the CRO colony is composed by 100 ants that are executed until the specific number of schedules are found. Regarding the values for α and β we have performed a parametrical study combining all the possible values for both parameters ranging from 1 to 3, i.e. for each instance of any dataset the system has found 50000 for each of the 9 different configurations for α and β.

It is important to note that the ACO using the Dynamic Oblivion Rate has been executed first in order to fix the size of the reef to the same number of pheromones created by the Dynamic Oblivion Rate. Regarding the configuration of the best heuristic algorithms found in the literature, we use the configuration than can be found in the corresponding paper for each specific algorithm.

The numerical comparison against the proposed model and the different heuristic algorithms for the `j60.sm`, `j90.sm` and `j120.sm` are shown in Table 10, Table 11 and Table 12, respectively. These tables include a short description of the algorithm, the reference where it was first reported, and the average deviations (in %) from the critical path lower bound for 1000, 5000 and 50000 schedules using all the instances of the corresponding dataset. All these algorithms are sorted by increasing deviation for 50000 schedules.

As it can be observed in these tables we provide two different evaluations for the ACO models. Once the algorithms obtain 1000, 5000 or 50000 solutions for each instance of the problem, we compute the average deviation using all the solutions found in all the instances of the problem (these results are identified by the word “mean” in the Algorithm column), and also we compute the average deviation using the best solution found in each instance (in this case, the word “min” identify these results). This distinction is performed due to the lack of information about what solutions contribute to the computation of the deviation from the critical path lower bound in the published literature.

Analyzing the results contained in these tables, it can be observed that ACO model using the proposed Oblivion Rate (i.e. the CRO-based Oblivion Rate) provides better results that the ACO model using Dynamic Oblivion Rate. The reason for this result can be found in the fight process that pheromones must perform in order to stay in the reef. This procedure makes that not only pheromones are removed from the graph but also only good pheromones remains

Algorithm	Max. #Schedules		
	1000	5000	50000
ACO - Dyn. Oblv. (min)	4.50	4.17	3.74
ACO - CRO-based Oblv. (min)	4.46	4.12	3.74
ACO - Dyn. Oblv. (mean)	12.28	12.279	12.27
ACO - CRO-based Oblv. (mean)	**12.216**	**12.214**	**12.21**

Table 11. Average deviation (%) from the critical path lower bound for `j90.sm`. Each column highlights in bold the best result that corresponds to the lower deviation obtained by the different algorithms.

Algorithm		Max. #Schedules		
		1000	5000	50000
ACO - Dyn. Oblv. (min)		12.757	11.97	11.02
ACO - CRO-based Oblv. (min)		14.28	13.18	11.94
ACO - Dyn. Oblv. (mean)		26.58	26.57	26.55
ACO - CRO-based Oblv. (mean)		**26.532**	**26.53**	**26.51**
ACOSS	[45]	35.19	32.48	30.56
HGA	[42]	–	32.75	30.66
GA - bi-population	[43]	–	32.34	30.82
Hybrid GA	[49]	34.07	32.54	31.24
GA-MBX	[44]	34.02	32.89	31.30
enhanced scatter search	[46]	–	32.61	31.37
GAPS	[48]	35.87	33.03	31.44
scatter Search - FBI	[47]	35.22	33.10	31.57
AHS	[42]	–	33.54	31.97

Table 12. Average deviation (%) from the critical path lower bound for the `j120.sm` dataset. Each column highlights in bold the best result that corresponds to the lower deviation obtained by the different algorithms.

in the reef.

If we analyze the results for the `j60.sm` dataset, the best results for 1000 and 5000 solutions are obtained by the algorithm called GA-MBX proposed by Zamani et. al. [44]. Using the 50000 solutions, the best result is achieve by HGA [42]. Taking into account the mean deviation of all the solutions, the ACO models is not the best solution. Nevertheless, analyzing only 1000 solutions the ACO model using the CRO-based Oblivion is the second best algorithm whereas the ACO model using the Dynamic Oblivion Rate is the third one. This indicates that both models are good approaches if a small number of solutions are taken into account.

As it was previously said, for `j90.sm` the ACO model using the CRO-based Oblivion Rate provides better results than the ACO with the Dynamic Oblivion Rate. This is produced by the fight process included in the CRO algorithm that makes pheromones fight to remain in the system.

For the `j120.sm` the ACO model using the CRO-based Oblivion Rate is the algorithm that outperforms the results provided by the research community. The second algorithm is the ACO model using the Dynamic Oblivion Rate. Both heuristics works in a similar way, although CRO-based Oblivion Rate obtains the best results. It is quite interesting to see how the improvement made by the Oblivion Rate heuristic under these two new configurations allows to achieve excellent results (26.51 for CRO-based Oblivion Rate and 26.55 for Dynamic Oblivion Rate) in the most complex dataset. All these results can be produced due an improved guidance of the ants through the solution space as a result of a more efficient usage and better quality of the deployed pheromone trails.

Finally, analyzing the results showed in Tables 10, 11 and 12 it seems that the ACO algorithm using any Oblivion Rate reaches suboptimal solutions and, thus, there are not any significant improvement in the critical path deviation when 1000, 5000 or 50000 solutions are built. This behaviour is observed only when all the solutions are taken into account to compute the critical path deviation. On the contrary, if the best solutions found for each problems are used, the algorithm converges (as it can be observed in the "min" rows of Tables 10, 11 and 12).

8. Concluding remarks

Constraint Satisfaction Problems provide an interesting framework to test and prove new heuristic approaches such as those belonging to the Swarm Intelligence (SI) field. One of the best known SI algorithms is Ant Colony Optimization (ACO), proposed in [27] and ever since applied to a wide number of application scenarios. In order to apply ACO to Constraint Satisfaction Problems, the problem must be modeled as a graph over which the ACO algorithm is executed. The current procedure to do so is by creating a fully connected graph containing as many nodes as pairs $< variable, value >$ in the problem formulation. This approach has several disadvantages:

1. Depending on the dimensionality of the problem, i.e. its number of variables and domain cardinalities, it could yield a high number of nodes.
2. The number of edges in the graph may grow very fast.
3. This representation is not suitable to deal with problems defined by continuous variables.

In order to solve this problem a new CSP graph-based representation was proposed in [26]. This approach is based on the reduction of the resulting graph complexity by reducing the number of nodes and connections. The idea is to create as many nodes as different variables compose the problem and create edges between those nodes involved in at least, one restriction. However, its main drawback is the high number of created pheromones. In order to solve this problem, a new heuristic called *Oblivion Rate* has been designed. The goal of this heuristic is to remove from the graph a specific number of pheromones in such a way only the most relevant ones remain in the graph.

The performance of two adaptive Oblivion Rate heuristics has been analyzed. On the one hand, the Dynamic Oblivion Rate heuristic corresponds to a negative exponential function that depends on the number of pheromones created in the system in order to determine the percentage of pheromones that will be removed. On the other hand the CRO-based Oblivion Rate heuristic inspires from the Coral Reef Optimization algorithm, which establishes a fight between pheromones to remain in the system. This fight is performed in terms of the quality of the pheromone.

Both heuristics have been applied to the well-known family of *Resource-Constrained Project Scheduling Problems* (RCPSP). In this type of problems a project is defined by a set of activities that require a certain type of resources during their execution. The goal is to find a schedule with the minimum makespan in such a way the precedence

relationships between the activities are satisfied and the number of assigned resources to all the activities in a given time step is less than the total amount of resources in the project.

The evaluation of both heuristics, and also the evaluation of the model without using any *Oblivion Rate* (i.e. the standard ACO algorithm), has taken into account two different factors: the reduction percentage in the number of pheromones in the system and the quality, i.e. the makespan, obtained by each algorithm. Such quality scores have been compared against the best makespan published by the research community and results have been validated by using statistical tests. The experimental results reveal that the use of the Oblivion Rate heuristic is relevant because it provides a significant reduction in the number of pheromones at a statistically negligible quality degradation. The Oblivion Rate heuristic has been found to remove at least 79% of the pheromones in the system without affecting significantly the quality of the produced solutions. This reduction in the number of pheromones is important because as it has been experimentally shown, it directly affects the computational resources (i.e. memory and time) needed by the program to solve the different problems. We have also analyzed the deviation between the best solutions found by the ACO models and the best makespan published in the literature for each problem. The ACO model (using the Dynamic Oblivion Rate and the CRO-based Oblivion Rate) is able to find the optimal solution in more than 50% of the instances that compose `j30.sm`, `j60.sm` and `j90.sm` problems, whereas the rest of solutions found for the same problems are close to its corresponding optimum.

Finally the performance of the model using the different Oblivion Rate heuristics is compared to some of the algorithms published in the related literature. This last set of experiments reveals that the proposed model outperforms the results from the literature with the most complex Single-Mode dataset of the PSPLIB repository (`j90.sm` and `j120.sm`). The reason behind this remarkable performance gain is the enhanced capability of the system to converge towards better solutions due to the removal of pheromones that represent the worst solutions by the Oblivion Rate heuristic.

Acknowledgements

This work has been supported by the next research projects: EphemeCH (TIN2014-56494-C4-4-P) Spanish Ministry of Economy and Competitivity, CIBERDINE S2013/ICE-3095, both under the European Regional Development Fund FEDER, Airbus Defence & Space (FUAM-076914 and FUAM-076915), BID3ABI (Basque Government), and RiskTrack (JUST-2015-JCOO-AG-723180).

References

[1] E. Tsang, Foundations of constraint satisfaction: the classic text, BoD–Books on Demand, 2014.
[2] B. Schröder, Constraint satisfaction problems, in: Ordered Sets, Springer, 2016, pp. 113–153.
[3] G.-F. Deng, W.-T. Lin, Ant colony optimization-based algorithm for airline crew scheduling problem, Expert Systems with Applications 38 (5) (2011) 5787–5793.
[4] H. C. Chang, Y. P. Chen, T. K. Liu, J. H. Chou, Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid taguchi-genetic algorithm, IEEE Access 3 (2015) 1740–1754. doi:10.1109/ACCESS.2015.2481463.
[5] M. R. Jabbarpour, H. Malakooti, R. M. Noor, N. B. Anuar, N. Khamis, Ant colony optimisation for vehicle traffic systems: applications and challenges, International Journal of Bio-Inspired Computation 6 (1) (2014) 32 – 56.
[6] M. Wen, E. Linde, S. Ropke, P. Mirchandani, A. Larsen, An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem, Computers & Operations Research (2016) –doi:http://dx.doi.org/10.1016/j.cor.2016.06.013.
URL http://www.sciencedirect.com/science/article/pii/S0305054816301460
[7] O. Koné, C. Artigues, P. Lopez, M. Mongeau, Event-based milp models for resource-constrained project scheduling problems, Computers & Operations Research 38 (1) (2011) 3–13.
[8] Araya, I., Reyes, V.: Interval branch-and-bound algorithms for optimization and constraint satisfaction: a survey and prospects. Journal of Global Optimization **65**(4) (2016) 837–866
[9] Belhaiza, S., Hansen, P., Laporte, G.: A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. Computers & Operations Research **52** (2014) 269–281
[10] Cao, P., Fan, Z., Gao, R., Tang, J.: A multi-objective simulated annealing approach towards 3d packing problems with strong constraints: Cmosa. In: ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers (2015)
[11] G. B. Orgaz, H. D. Menéndez, D. Camacho, Adaptive k-means algorithm for overlapped graph clustering, Int. J. Neural Syst. 22 (5). doi:10.1142/S0129065712500189.
URL http://dx.doi.org/10.1142/S0129065712500189
[12] I. Boussaïd, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences 237 (2013) 82 – 117.

[13] Y. Zhang, , Y. Zhang, H. C. Yap, Making ac-3 an optimal algorithm, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence, Vol. 1, 2001, pp. 316–321.
[14] A. M. C. A. Koster, S. P. M. van Hoesel, A. W. J. Kolen, Solving partial constraint satisfaction problems with tree decomposition, Networks 40 (3) (2002) 170–180.
[15] G. Pappa, G. Ochoa, M. Hyde, A. Freitas, J. Woodward, J. Swan, Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms, Genetic Programming and Evolvable Machines 15 (1) (2014) 3–35.
[16] C. Blum, D. Merkle, Swarm Intelligence: Introduction and Applications, 1st Edition, Springer Publishing Company, Incorporated, 2008.
[17] A. P. Engelbrecht, Computational Intelligence: An Introduction, 2nd Edition, Wiley Publishing, 2007.
[18] X.-S. Yang, Engineering optimizations via nature-inspired virtual bee algorithms, in: J. Mira, J. R. Álvarez (Eds.), Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Vol. 3562 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 317–323.
[19] R. F. Tavares Neto, M. G. Filho, Literature review regarding ant colony optimization applied to scheduling problems: Guidelines for implementation and directions for future research, Engineering Applications of Artificial Intelligence 26 (1) (2013) 150 – 161.
[20] B. C. Mohan, R. Baskaran, A survey: Ant colony optimization based recent research and implementation on several engineering domain, Expert Systems with Applications 39 (4) (2012) 4618 – 4627.
[21] M. A. M. de Oca, D. Aydın, T. Stützle, An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re) design of optimization algorithms, Soft Computing 15 (11) (2011) 2233–2255.
[22] Q. Jia, Y. Seo, An improved particle swarm optimization for the resource-constrained project scheduling problem, The International Journal of Advanced Manufacturing Technology 67 (9-12) (2013) 2627–2638.
[23] R. Akbari, V. Zeighami, K. Ziarati, Artificial bee colony for resource constrained project scheduling problem, International Journal of Industrial Engineering Computations 2 (1) (2011) 45–60.
[24] K. Ziarati, R. Akbari, V. Zeighami, On the performance of bee algorithms for resource-constrained project scheduling problem, Applied Soft Computing 11 (4) (2011) 3720–3733.
[25] A. Gonzalez-Pardo, D. Camacho, A new csp graph-based representation to resource-constrained project scheduling problem, in: 2014 IEEE Conference on Evolutionary Computation, IEEE Xplore 2014, 2014, pp. 344–351.
[26] A. Gonzalez-Pardo, D. Camacho, A new csp graph-based representation for ant colony optimization, in: 2013 IEEE Conference on Evolutionary Computation, Vol. 1, 2013, pp. 689–696.
[27] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, New Ideas in Optimization (1999) 11–32.
[28] P.-P. Grassé, La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs, Insectes Sociaux 6 (1) (1959) 41–80.
[29] S. Hoseini Semnani, K. Zamanifar, The power of ants in solving distributed constraint satisfaction problems, Applied Soft Computing 12 (2) (2012) 640–651.
[30] S. Khan, M. Bilal, M. Sharif, M. Sajid, R. Baig, Solution of n-queen problem using aco, in: IEEE 13th International Multitopic Conference, INMIC. 2009, 2009, pp. 1 – 5.
[31] C. Solnon, Ants can solve constraint satisfaction problems, IEEE Transactions on Evolutionary Computation 6 (2002) 347–357.
[32] A. Gonzalez-Pardo, D. Camacho, Environmental influence in bio-inspired game level solver algorithms, in: Proceedings of the 7th International Symposium on Intelligent Distributed Computing - IDC 2013, Vol. 511 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2013, pp. 157–162.
[33] K. Gao, P. Suganthan, Q. Pan, T. Chua, T. Cai, C. Chong, Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling, Information Sciences 289 (2014) 76 – 90.
[34] A. Schirmer, Case-based reasoning and improved adaptive search for project scheduling, Naval Research Logistics (NRL) 47 (3) (2000) 201–222.
[35] S. Salcedo-Sanz, A. Pastor-Sánchez, D. Gallo-Marazuela, A. Portilla-Figueras, A novel coral reefs optimization algorithm for multi-objective problems, in: Intelligent Data Engineering and Automated Learning (IDEAL 2013), Vol. 8206 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 326–333.
[36] A. Genin, L. Karp, A. Miroz, Effects of flow on competitive superiority in scleractinian corals, Limnology and Oceanography 39 (4) (1994) 913–924.
[37] R. Ates, Aggresive behaviour in corals, Freshwater and Marine Aquarium 12 (8) (1989) 104–112.
[38] N. E. Chadwick, Interspecific aggressive behavior of the corallimorpharian corynactis californica (cnidaria: Anthozoa): effects on sympatric corals and sea anemones, The Biological Bulletin 173 (1) (1987) 110–125.
[39] R. Kolisch, A. Sprecher, Psplib – a project scheduling problem library, European Journal of Operational Research 96 (1996) 205–216.
[40] E. L. Demeulemeester, W. S. Herroelen, New benchmark results for the resource-constrained project scheduling problem, Management Science 43 (1997) 1485–1492.
[41] R. Kolisch, S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: An update, European Journal of Operational Research 174 (1) (2006) 23–37.
[42] A. Lim, H. Ma, B. Rodrigues, S. T. Tan, F. Xiao, New meta-heuristics for the resource-constrained project scheduling problem, Flexible Services and Manufacturing Journal 25 (1) (2013) 48–73. doi:10.1007/s10696-011-9133-0.
URL http://dx.doi.org/10.1007/s10696-011-9133-0
[43] D. Debels, M. Vanhoucke, A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem, Oper. Res. 55 (3) (2007) 457–469. doi:10.1287/opre.1060.0358.
URL http://dx.doi.org/10.1287/opre.1060.0358
[44] R. Zamani, A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem, European Journal of Operational Research 229 (2) (2013) 552 – 559.
[45] W. Chen, Y. Shi, H. Teng, X. Lan, L. Hu, An efficient hybrid algorithm for resource-constrained project scheduling, Information Sciences 180 (6) (2010) 1031 – 1039, special Issue on Modelling Uncertainty. doi:http://dx.doi.org/10.1016/j.ins.2009.11.044.

URL http://www.sciencedirect.com/science/article/pii/S0020025509005222

[46] M. D. Mahdi Mobini, M. Rabbani, M. S. Amalnik, J. Razmi, A. R. Rahimi-Vahed, Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem, Soft Computing 13 (6) (2009) 597–610. doi:10.1007/s00500-008-0337-5.
URL http://dx.doi.org/10.1007/s00500-008-0337-5

[47] D. Debels, B. D. Reyck, R. Leus, M. Vanhoucke, A hybrid scatter search/electromagnetism meta-heuristic for project scheduling, European Journal of Operational Research 169 (2) (2006) 638 – 653, feature Cluster on Scatter Search Methods for Optimization.

[48] J. Mendes, J. Gonçalves, M. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem, Computers & Operations Research 36 (1) (2009) 92 – 109. doi:http://dx.doi.org/10.1016/j.cor.2007.07.001.
URL http://www.sciencedirect.com/science/article/pii/S0305054807001359

[49] V. Valls, F. Ballestín, S. Quintanilla, A hybrid genetic algorithm for the resource-constrained project scheduling problem, European Journal of Operational Research 185 (2) (2008) 495 – 508. doi:http://dx.doi.org/10.1016/j.ejor.2006.12.033.
URL http://www.sciencedirect.com/science/article/pii/S0377221707000616