Expert Systems with Applications 41 (2014) 6634–6645

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A Max–Min Ant System algorithm to solve the Software Project Scheduling Problem

Broderick Crawford [a,b], Ricardo Soto [a,d], Franklin Johnson [a,c,*], Eric Monfroy [e], Fernando Paredes [f]

[a] Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2950, Valparaíso, Chile
[b] Universidad Finis Terrae, Av. Pedro de Valdivia 1509, Santiago, Chile
[c] Universidad de Playa Ancha, Av. Leopoldo Carvallo 270, Valparaíso, Chile
[d] Universidad Autónoma de Chile, Pedro de Valdivia 641, Santiago, Chile
[e] CNRS, LINA, University of Nantes, 2 rue de la Houssinière, Nantes, France
[f] Escuela de Ingeniería Industrial, Universidad Diego Portales, Manuel Rodríguez Sur 415, Santiago, Chile

ARTICLE INFO

Article history:
Available online 14 May 2014

Keywords:
Ant Colony Optimization
Project management
Software engineering
Software Project Scheduling Problem

ABSTRACT

The Software Project Scheduling Problem is a specific Project Scheduling Problem present in many industrial and academic areas. This problem consists in making the appropriate worker-task assignment in a software project so the cost and duration of the project are minimized. We present the design of a Max–Min Ant System algorithm using the Hyper-Cube framework to solve it. This framework improves the performance of the algorithm. We illustrate experimental results and compare with other techniques demonstrating the feasibility and robustness of the approach, while reaching competitive solutions.

1. Introduction

In project management, the manager has to initiate, schedule, execute, control, and close the project, this is a complex job for the project manager. This is the reason why we need to optimize these activities. The Software Project Scheduling Problem (SPSP) is an specific Project Scheduling Problem (PSP) (Chen, 2011; Laalaoui & Bouguila, 2014) which consists in making the appropriate worker-task assignment that minimizes cost and duration for the whole project, so the task precedence and resource constraints are satisfied (Alba & Chicano, 2007; Chang, yi Jiang, Di, Zhu, & Ge, 2008; Nan & Harter, 2009). The PSP consists in organizing multiple activities which can be assigned multiple resources, which can be renewable or non-renewable, so that meet the defined objectives. While in SPSP the main resource are the employees, who must to be assigned to project tasks according to their skills. Another difference is that PSP may have different single objectives, while in SPSP the main objective is to minimize the duration and cost of the project.

The SPSP is a variation of another NP-hard problem called Resource-Constrained Project Scheduling (RCPSP) is a very popular problem in the literature (Brucker, Drexl, Mhring, Neumann, & Pesch, 1999). RCPSP has many variations such as single-mode RCPSP, multi-mode RCPSP, RCPSP with non-regular objective function, stochastic RCPSP among others (Fahmy, Hassan, & Bassioni, 2014; Hartmann & Briskorn, 2010; Tavana, Abtahi, & Khalili-Damghani, 2014; Wu, Wan, Shukla, & Li, 2011). This has given rise to multiple jobs, which derives the SPSP.

Alba proposes the original modeling of the SPSP and its resolution by Genetic Algorithms (GA) (Alba & Chicano, 2007). Chang proposes a Time-line based model for SPSP using Genetic Algorithms (Chang et al., 2008). Later Xiao proposes a genetic algorithm and Ant Colony System (ACS) to solve SPSP (Xiao, Ao, & Tang, 2013), and Crawford proposes a new resolution by using Max–Min Ant Systems (MMAS) in Crawford, Soto, Johnson, and Monfroy (2013a). Luna makes a scalability analysis of multi-objective metaheuristics solving SPSP (Luna, González-Álvarez, Chicano, & Vega-Rodríguez, 2014). Other articles have been written about SPSP only as surveys.

SPSP is a problem where a set of employees must be assigned to a set of tasks, so all tasks are completed by employees who have the necessary skills to accomplish tasks. This assignment should minimize the cost and duration of the whole project. The employees have remuneration and several skills and they can work on multiple tasks during the workday. Most of the methods used for solving the problem belong to the class of priority-rules based methods or the class of metaheuristics.

ACO is a promising metaheuristic (Dorigo, Maniezzo, & Colorni, 1996), inspired from the behavior of real ant colonies. It is a probabilistic approach for solving computational problems which can

* Corresponding author at: Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2950, Valparaíso, Chile. Tel.: +56 322273719.
E-mail address: franklin.johnson@upla.cl (F. Johnson).

http://dx.doi.org/10.1016/j.eswa.2014.05.003
0957-4174/

be reduced to finding good paths through graphs. This metaheuristic can solve several NP-hard combinatorial problems effectively (Christina & Miriam, 2012; Liao, Egbelu, Sarker, & Leu, 2011).

In this paper we present the model of SPSP, and the design of a Max–Min Ant System algorithm (Dorigo & Stützle, 2004; Stützle & Hoos, 2000) using the Hyper-Cube framework (HCF) (Blum & Dorigo, 2004) to solve the Software Project Scheduling Problem. MMAS is a specific ACO algorithm in which only the best ant is used to update the pheromone information.

We introduce the ACO Hyper-Cube (ACO-HC) algorithm, to solve the SPSP. This algorithm uses the HCF. HCF was originally proposed by Blum and Dorigo (Blum & Dorigo, 2004). This framework automatically handles the limits of pheromone values by modification in the update pheromone rule. This modification allows the algorithms to be more robust and easier to implement. The HCF can be applied to different ACO algorithms (Johnson, Crawford, & Palma, 2006).

We implement our proposed algorithm, and we conducted a series of tests to analyze the convergence to obtain better solutions. In addition we work in different tests to get the best parameterization. The tests were performed using different numbers of tasks, employees, and skills. The results were compared with other techniques such as Ant Colony System and Genetic Algorithms.

In this contribution we have built a solution for Software Project Scheduling Problem adapting the Max–Min Ant System metaheuristic and integrating it with the Hyper-Cube framework. That is, we have integrated two techniques for building competitive and robust solutions improving the solutions of other proposals.

This paper is organized as follows. In Section 2 we present a detailed definition of SPSP, in Section 3 is presented a description of ACO-HC. Section 4 presents the design of an ACO-HC for SPSP. In a subsection we presents the construction graph, pheromone update rules, the heuristic information, and the algorithm. Section 5 presents the experimental results. The conclusions are outlined in Section 6.

2. Description of the Software Project Scheduling Problem

The Software Project Scheduling Problem is one of the most common problems in managing software engineering projects (OZDAMAR & ULUSOY, 1995). It consists in finding a worker-task schedule for a software project. SPSP should consider remunerations and employee skills which must be assigned to project tasks according to the requirements of these tasks (Alba & Chicano, 2007; Barreto, Barros, & Werner, 2008; Xiao et al., 2013). The most important resources involved in SPSP are: the tasks, which are the job needed to complete the project, the employees who work in the task, and finally the skills. The employees have multiples skills, and the tasks required a set of skills. It should make a careful allocation according to the skills needed for the tasks and the skills of employees.

2.1. Description of skills

As mentioned above, the skills are the abilities required for completing the tasks, and the employees have all or some of these abilities. These skills can be for example, design expertise, programing expert, leadership, GUI expert. The set of all skills associated with software project is defined as $S = \{s_1, \ldots, s_{|S|}\}$, where $|S|$ is the number of skills.

2.2. Description of tasks

The tasks are all necessary activities for accomplishing the software project. These activities are for example, analysis, component design, programing, documentation, testing. The software project is a sequence of tasks with different precedence among them. Generally, we can use a graph called task-precedence-graph (TPG) to represent the precedence of these tasks. This is a non-cyclic directed graph denoted as $G(V, E)$. The set of tasks is represented by $V = \{t_1, t_2, \ldots, t_{|T|}\}$. The precedence relation of tasks is represented by a set of edges E. An edge $(t_i, t_j) \in E$, means t_i is a direct predecessor task t_j. Consequently, the set of tasks necessary for the project is defined as $T = \{t_1, \ldots, t_{|T|}\}$, where $|T|$ is the maximum number of tasks. Each task have two attributes:

- t_j^{sk} is a set of skills for the task j. It is a subset of S and corresponds to all necessary skills to complete a task j.
- t_j^{eff} is a real number and represents the workload of the task j.

2.3. Description of employees

The most relevant resource in this problem is the employees. The employees have multiple skills, commonly the employees are software engineers, and their skills are software engineering skills. The project has a set of employees and they work on the tasks. The project manager needs to assign the employees to the appropriate tasks. The problem is to create a worker-task schedule where employees are assigned to suitable tasks. The set of employees is defined as $EMP = \{e_1, \ldots, e_{|E|}\}$, where $|E|$ is the number of employees working on the project. Each employee has tree attributes:

- e_i^{sk} is a set of skills of employee i. $e_i^{sk} \subseteq S$.
- e_i^{maxd} is the maximum degree of work. it is the ratio between hours for the project and the workday. $e_i^{maxd} \in [0, 1]$, if $e_i^{maxd} = 1$ the employee has total dedication to the project, if the employee has e_i^{max} less than one, in this case is a part-time job.
- e_i^{rem} is a real number. It is the monthly remuneration of employee i.

2.4. Model description

The model for SPSP use tasks, employees, and skills previously described. We describe the elements of the model in Table 1.

The SPSP solution can be represented as a matrix $M = [E \times T]$. The size $|E| \times |T|$ is the dimension of matrix determined by the number of employees and the number of tasks. The elements of the matrix $m_{ij} \in [0, 1]$, corresponds to real numbers, which represent the degree of dedication of employee i to task j. If $m_{ij} = 0$, the employee i is not assigned to task j, that means the employee does not dedicate time for this task. If $m_{ij} = 1$, the employee i works all day in the task j. For example, if $m_{ij} = 0.5$, the employee i uses 50 percent of his time on the project.

The solutions generated in this matrix not always are feasible. That happens when all elements of a column i are 0, that means the employees are not assigned to the task i. This solution is not feasible because the task i is not finished. For this reason we define some constraints to obtain a feasible solution from the matrix M;

- First, all tasks are assigned at least one employee as is presented in Eq. (1).

$$\sum_{i=1}^{|E|} m_{ij} > 0 \quad \forall j \in \{1, \ldots, T\} \tag{1}$$

- Second, the employees assigned to the task j have all the necessary skills to carry out the task. This is presented in Eq. (2) follows by the skills needed for the task t_j are a subset of the union of the skills the employees assigned to the task.

Table 1
SPSP Model.

Item	Description
$S = \{s_1, \ldots, s_{|S|}\}$	Set of skills associated to software projects
$T = \{t_1, \ldots, t_{|T|}\}$	set of tasks necessary for the project
$G(V, E)$	The graph TGP represent the task precedence
$V = \{t_1, t_2, \ldots, t_{|T|}\}$	Set of vertex consisted of all tasks
$E = \{(t_i, t_j), \ldots, (t_n, t_{|T|})\}$	Edge set, the task t_i must be done before t_j
t_j^{sk}	Set of skills for the task j. It is a subset of S
t_j^{eff}	Effort person-months to complete the task j
$EMP = \{e_1, \ldots, e_{|E|}\}$	Set of employees
e_i^{sk}	Set of skills of e_i. It is a subset of S
e_i^{maxd}	Maximum degree of dedication of e_i, $e_i \in [0, 1]$
e_i^{rem}	Monthly remuneration of e_i
$M = (m_{ij})$	The employee i is assigned to task j, with m_{ij} dedication
t_j^{init}	Initialization time for task j
t_j^{term}	Termination time for task j
t_j^{cos}	Cost associated to task j
t_j^{len}	Required time to complete the task j
p^{len}	Required time to complete the whole project
p^{cos}	Total cost associated to project
e_i^{overw}	Overtime work of employee e_i
p^{overw}	Overtime work of the whole project

$$t_j^{sk} \subseteq \bigcup_{i|m_{ij}>0} e_i^{sk} \quad \forall j \in \{1, \ldots, T\} \tag{2}$$

We represent in Fig. 1 an example for the precedence tasks TPG and their necessary skills t^{sk} and effort t^{eff}. For the example presented in Fig. 1, we have a set of employees $EMP = \{e_1, e_2, e_3\}$, and each one of these have a set of skills, maximum degree of dedication, and remuneration, ilustrated in Fig. 2.

A solution consists in completing the matrix M, previously described, such tasks are assigned to employees who have the necessary skills. A solution for the previous problem is described in Fig. 3.

First, it should be evaluated the feasibility of the solution, then using the duration of all tasks and cost of the project, we appraise the quality of the solution.

We compute the length time for each task as $t_j^{len}, j \in \{1, \ldots, |T|\}$, for this we use matrix M and t_j^{eff} according to the Eq. (3).

$$t_j^{len} = \frac{t_j^{eff}}{\sum_{i=1}^{|E|} m_{ij}} \tag{3}$$

$$e_1^{sk} = \{s_1, s_2, s_3\} \quad e_2^{sk} = \{s_1, s_2, s_3\} \quad e_3^{sk} = \{s_1, s_2, s_3\}$$
$$e_1^{maxd} = 1.0 \quad e_2^{maxd} = 0.5 \quad e_3^{maxd} = 1$$
$$e_1^{rem} = \$X_1 \quad e_2^{rem} = \$X_2 \quad e_3^{rem} = \$X_3$$

Fig. 2. Information of employees for the example in Fig. 1.

Now we can obtain the initialization time t_j^{init} and the termination time t_j^{term} for task j. To calculate these values, we use the precedence relationships, which is described as TPG $G(V, E)$. We must consider tasks without precedence, in this case the initialization time $t_j^{init} = 0$. To calculate the initialization time of tasks with precedence firstly we must calculate the termination time for all previous tasks. In this case t_j^{init} is defined in Eq. (4).

$$t_j^{init} = \begin{cases} 0 & \text{if } \forall l \neq j, (t_l, t_j) \notin E \\ max\{t_l^{term} \mid (t_l, t_j) \in E\} & else \end{cases} \tag{4}$$

$$t_j^{term} = t_j^{init} + t_j^{len} \tag{5}$$

Now we have the initialization time t_j^{init}, the termination time t_j^{term} and the duration t_j^{len} for task j with $j = \{1, \ldots, |T|\}$, that means we have all elements to generate a Gantt chart. This equation represents a forward programing where there is no idle time. If there should be included in the duration of the task to extend t_j^{term}. To calculate the total length of the project p^{len}, we only need termination time of last task, this is described in Eq. (6).

$$p^{len} = max\{t_l^{term} \| \forall l \neq j(t_j, t_l)\} \tag{6}$$

To calculate the cost of the whole project, we need firstly to compute each cost associate to task as t_j^{cos} with $j \in \{1, \ldots, |T|\}$ using Eq. (7), and then the total cost p^{cos} is the sum of costs according to the Eq. (8).

$$t_j^{cos} = \sum_{i=1}^{|E|} e_i^{rem} m_{ij} t_j^{len} \tag{7}$$

$$p^{cos} = \sum_{j=1}^{|T|} t_j^{cos} \tag{8}$$

The target is minimize the total duration p^{len} and the total cost p^{cos}. Therefore a fitness function is used, where w^{cos} and w^{len} represent the importance of p^{cos} and p^{len}. Then w^{cos} and w^{len} are real

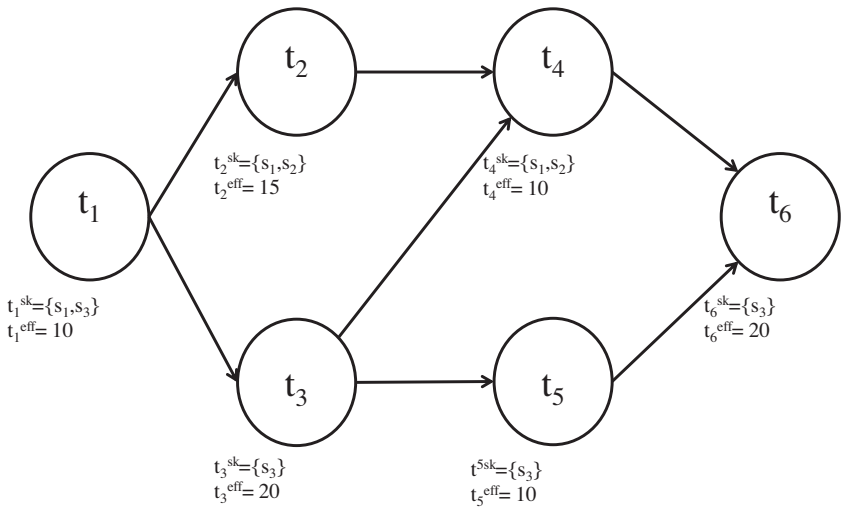

Fig. 1. Task precedence graph TGP with $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}, d = \{s_1, s_2, s_3\}$.

M(ij)	t_1	t_2	t_3	t_4	t_5	t_6
e_1	1.00	0.50	0.00	0.25	0.00	0.00
e_2	0.25	0.50	1.00	0.50	0.00	0.25
e_3	0.50	1.00	0.00	0.50	1.00	0.25

Fig. 3. A possible solution for matrix *M*.

numbers, this parameters allow us to meet the different magnitudes. For this, w^{cos} is initialized in (average cost)$^{-1}$, and w^{len} is initialized in (average length)$^{-1}$. Then multiplying the parameters by the corresponding p^{cos} and p^{len} the units are canceled and the values are in the same magnitude. Then, the fitness function to minimize is given by Eq. (9).

$$f(x) = \left(w^{cos}p^{cos} + w^{len}p^{len}\right) \tag{9}$$

An element not considered is the overtime work that may increase the cost and duration associated to a task, consequently increasing p^{cos} and p^{len} to the software project. We define the overtime work as e_i^w. To calculate it, we use a function based on the work load of employee at time t as is presented in Eq. (10).

$$e_i^w(t) = \sum_{\left\{t_j^{init} \leqslant t \leqslant t_j^{term}\right\}} m_{ij}(t) \tag{10}$$

If the employee e_i has overtime work e_i^w, which means the work in instant t is larger than maximum degree of work, i.e., $e_i^w(t) > e_i^{maxd}$. To calculate the overwork, we define the Eq. (11).

$$ramp(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{11}$$

Now we can calculate the overwork of an employee in the whole project using the Eq. (12)

$$e_i^{overw} = \sum_{t=0}^{p^{len}} ramp(e_i^w(t) - e_i^{maxd}) \tag{12}$$

To obtain the project overwork p^{overw}, we must consider all employees. To do this, we can use the Eq. (13).

$$p^{overw} = \sum_{i=1}^{|E|} e_i^{overw} \tag{13}$$

With all variables required, we can determine if the solution is feasible. In this case, is feasible when the solution can complete all tasks, and there is no overwork, that means the $p^{overw} = 0$.

3. The Hyper-Cube framework for ACO

The ACO algorithm exploits an optimization mechanism for solving discrete optimization problems in various engineering domains. ACO is a metaheuristic (Dorigo & Gambardella, 1997), inspired from the behavior of real ant colonies. Furthermore it is a probabilistic approach for solving computational problems which can be reduced to finding good paths through graphs. The ants use two resources to build solutions; the memoristic information determined by the pheromone trails deposited by ants and the heuristic information determined by an specific problem.

The Hyper-Cube was proposed by Blum and Dorigo (2004) and Crawford et al. (2013a). This framework implements ACO algorithms, that explicitly defines the multidimensional space for the pheromone values as the convex hull of the set of 0–1 coded feasible solutions of the combinatorial optimization problem under consideration.

The Hyper-Cube framework for ACO makes a modification in the pheromone update rule, which is obtained via a normalization of the original pheromone update equation. This allows a more robust and autonomous handling of pheromone values to improve the exploration of the solution space.

An advantage to implement this framework to Max–Min Ant System (Stützle & Hoos, 2000), that is no need to reset τ_{min} and τ_{max} values. These values are known (0 and 1, respectively). Another advantage is when working with normalized values becomes more explicit the relationship between the amounts of pheromone deposited on a component and its use for the construction of solutions.

4. Max–Min Ant System for Schedule Software Project

The Max–Min Ant System is an Ant Colony Optimization algorithm (Stützle & Hoos, 2000), which establishes a minimum and maximum value for the pheromone, and provides that only the best ant can update the pheromone trail. To adapt a Max–Min Ant System to SPSP using a Hyper-Cube framework (Johnson et al., 2006; Rubio, Crawford, & Johnson, 2008) we must establish an appropriate construction graph and define the use of pheromone as well as heuristic information associated with the specified problem (Abdallah, Emara, Dorrah, & Bahgat, 2009; Chen & Zhang, 2013).

The construction graph structure and pheromone matrix, with the respective ACO-HC algorithm is presented in the following subsections. This algorithm makes the association of employees to tasks according to the needs of the tasks, evaluating the quality of the solution. As presented in Fig. 4.

4.1. Construction graph

For constructing a solution the ants travel through the construction graph. The ants start from an initial node and then select the nodes according to a probability function. This function is given by the pheromone and heuristic information of the problem, their relative influence is given by α and β, respectively (Berrichi, Yalaoui, Amodeo, & Mezghiche, 2010; Dorigo & Caro, 1999; Dorigo et al., 1996).

The first step is adapting the SPSP to graph representation. To adapt the problem we use the heuristic information and pheromone to construct a directed graph. The employees must be assigned to a project task, and their dedication to each task is represented in the graph TGP. We can use this representation to define the construction graph. The proposed construction graph represents the association of employee and their dedication to a task. This representation is constructed for each task in the TGP; it is divided into a graph with node and edge. The construction graph consists of each employee and their ratio of dedication contributions for the task, it is defined as *den*. This variable is density of nodes and it is defined as:

$$den = \frac{1}{mind} + 1, \tag{14}$$

where *mind* is the lowest degree of dedication to a task. This structure is presented in Fig. 5. The employees dedication to a task can be

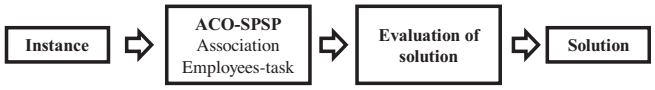

Fig. 4. Resolution structure for Scheduling Software Project.

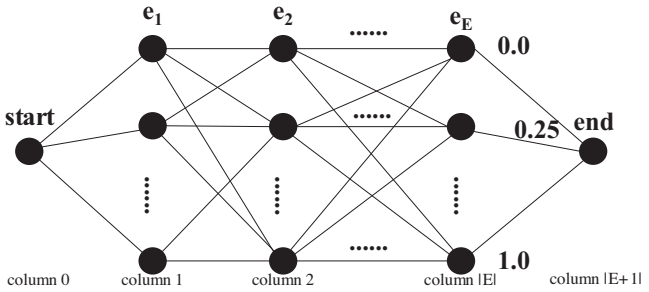

Fig. 5. Construction Graph is a matrix $CG = [den \times E]$ with $mind = 0.25$ for a task.

0 or integer multiple of *mind*. We can describe the split operation of a task as: first, generate a start node and insert into first column $column_0$. Second, generate the columns $column_i$, with $i = \{1, \ldots, |E|\}$. Each column consists in *den* nodes. Third, construct an end node and put into the last column $column_{|E|+1}$. Fourth, constructs edges, add edges between columns.

The ants travel to the start node to the end node choosing edges from the column 1 to column $|E|$ without returning, this because it is a directed graph that goes from left to right. The ants choose only one node per each column. When the ant completes a tour, the dedication of employees to a task is complete. To calculate the dedication of the employee i to the task, we just need the node j, with column i and the calculation is $j * mind$.

These activities of ants must be done for each task in SPSP model. The ants travel in random order selecting nodes. A tour corresponds to an assignment of employees to task and also determines the dedication to the task. Then we have to evaluate the quality and feasibility of the solution generated.

The ants travel through the construction graph selecting ways of probabilistically way, using the Eq. (15).

$$p_{ij}^{t} = \frac{[\tau_{ij}]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l=0}^{den}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \quad j \in \{1, \ldots, den\} \tag{15}$$

where τ_{ij} is the pheromone and η_{ij} is the heuristic information of the problem on the path between node i to j in the graph CG for t task. α and β are two fixed parameters, which are used to determine the pheromone and heuristics influences. To determine the values of α and β we can use empirical tuning of the variables. The pheromone τ_{ij} and heuristic information η_{ij} are positive real number < 1. The heuristic information can be defined according to different strategies. That is explained in the following subsections.

4.2. Pheromone update

In the Hyper-Cube framework the pheromone trails are forced to stay in the interval $[0, 1]$ and Min–Max Ant System the pheromone stay in the interval $[\tau_{min}, \tau_{max}]$. To adapt the Hyper-Cube framework to MMAS we define a $\tau_{min} = 0$ and, $\tau_{max} = 1$, and represent the pheromone update rule. The MMAS behavior is not altered, only a new pheromone update rule is created in order to be consistent with HCF. The HCF can be adapted to any ACO algorithm. Then two approaches are used to create a new proposal. In MMAS only one single ant is used to update the pheromone trails after each iteration, that ant can be the *iteration-best* ant or *global-best* ant. The use of only one solution, for the pheromone update allows improving the exploitation of the search space. For this reason, the best solutions are used to reinforce the pheromone trails. A good selection between *iteration-best* and *global-best* ant for updating the pheromone trails can control the search in the solution space. When using only *global-best*, the solution can converge quickly and lost exploration of solution space, with the consequent danger of getting trapped in poor quality solutions. This problem is minimized when *iteration-best* is selected for the pheromone trail update since the *iteration-best* solutions may differ considerably from iteration to iteration and a larger number of solution components may receive occasional reinforcement. The best approach is using mixed strategies like choosing *iteration-best* as a default for updating the pheromones and using *global-best* only every fixed number of iterations. Computationally we represent the evaporation of pheromone and in addition the amount of pheromone in the ant path through the graph once a tour is completed using the following formula:

$$\tau_{ij} = \rho\tau_{ij} + (1 - \rho)\Delta\tau^{upd} \tag{16}$$

where ρ is a rate of evaporation $\rho \in]0, 1]$. If ρ is high, the new pheromone value is less influenced by $\Delta\tau^{upd}$, but much influenced by the previous pheromone value, vice versa. And $\Delta\tau^{upd}$ it is associated with quality of the current solution of *upd* ant. *upd* ant is *iteration-best* ant. We can use an updating pheromone strategy considering duration, cost, and overwork of the project as follows (Johnson et al., 2006):

$$\Delta\tau^{upd} = \left(w^{cos}p^{cos} + w^{len}p^{len} + w^{overw}p^{overw}\right)^{-1} \tag{17}$$

where p^{len} is the total duration, p^{cos} is the total cost of software project, and p^{overw} is the project overwork. If the solution is feasible $p^{overw} = 0, p^{overw} > 0$ otherwise. w^{cos}, w^{len}, and w^{overw} are values that weight the importance of p^{cos}, p^{len}, and p^{overw} of the software project. These parameters are used to adjust the importance between p^{cos}, p^{len}, and p^{overw} using the same measurement. The Δupd is the amount of pheromone added based on the quality of solution generated by *upd* ant.

4.3. Heuristic information

We need to represent the heuristic information, that information is used to enhance the search ability of ants. The ants need to find the proper nodes using the problem information. The ants travel for a matrix m_{ij} as the node at column i and row j. To obtain the dedication of an employee e_i to a selected task, we must calculate $j * mind$. We use two heuristic information; H1 and H2. H1 uses the dedication of employee e_i to other task. If an employee works more in the previous tasks, that employee will has less dedication available for subsequent tasks. Consequently, the employee have less probability to be assigned to the current task. The heuristic information H1 $h[i]$ to select node i for task t_k can be calculated using Eq. (18).

$$h[i] = \begin{cases} \frac{tmp[den-i-1]}{sum}, & \text{if } allocD[k] > 0.5, \\ \frac{tmp[i]}{sum}, & \text{else} \end{cases} \tag{18}$$

where

$$tmp[i] = \begin{cases} ded[i] + allocD[k] - 0.5, & \text{if } allocD[k] > 0.5, \\ ded[i] + allocD[k], & \text{else} \end{cases} \tag{19}$$

The *SUM* is the summation of all values of the array *tmp*, and it can be calculated as follows:

$$sum = \sum_{i=0}^{den} tmp[i] \tag{20}$$

The components used in the previous formula are presented in Table 2.

H2 uses a constant strategy as the remuneration of employees. If an employee has a high remuneration is less likely to be selected. If an employee has a low remuneration will have a high probability

Table 2
Components of heuristic information.

Item	Description
$tmp = \{1, \ldots, den\}$	An array of temporal values
$h = \{1, \ldots, den\}$	An array of heuristic information for nodes
$ded = \{1, \ldots, den\}$	An array of dedication for candidate nodes
$allocD = \{1, \ldots, \lvert E\rvert\}$	An array with allocated dedications for employees
sum	Is the sum of all array temp

of being assigned to a task. The heuristic information H2 is an array $h[i]$ for employee i can be calculated with Eq. (21).

$$h[i] = (e_i^{rem})^{-1} \tag{21}$$

4.4. Algorithm description

The general structure of the algorithm is presented in Fig. 4, in which some modifications added to the ones presented in Stützle and Hoos (2000). A new assignation values to τ_{max}, τ_{min} and a new pheromone update rules. ACO-HC algorithm to solve the Software Project Scheduling problem can be described as follow:

Algorithm 1 – ACO-HC for SPSP Algorithm

```
1  Input: Problem instance I
2  Initialize the pheromone values
3  Repeat
4    For g = 1 to G do
5      initialize process of ant
6      For a = 1 to M do
7        For t = 1 to T do
8          ant a travel on the matrix CG = [den × E] choosing
           employee and their dedication to a task t
9          storing partial route for a ant
10       End for
11       compute the feasibility of a ant according as
         established in Section 2.4
12       compute the fitness of the solution using Eq. (9).
13     End for
14     select the best solution
15     update pheromone values
16   End for
17 select the best global solution (optional)
18 apply pheromone update for global-best
19 Until (iterations or time) is complete
20 Output: An optimized solution for I
```

The algorithm firstly reads the problem instance. That instance provides all the necessary data to generate the SPSP, such as number of tasks, and their required skills, number of employees, task precedence information for generating the TGP, the set of skills of every employee, and their remunerations. Then, we have to split operation to the task and then using the ACO-HC to generate solutions. To determine the quality of solutions, we use Eq. (9) that is the fitness function. That function minimizes the cost and duration for the whole project. Now we describe ACO-HC for SPSP as follows:

- To initialize the parameters τ_{max} and τ_{min}. α and β, number of ants M, and number of iterations G.
- To initialize the pheromone values; to put the pheromone on the edge of the construction graph CG, then to initialize the pheromone to τ_{max}.
- Ants constructs solutions. Each ant travels through the construction graph selecting nodes. It is to fix the dedication degree of the employees in the task. When an ant completes a tour, repeats his process for the next task. Finally when the tours for all tasks are finished a solution is constructed.
- Now we analyze the solution and evaluate the quality using the fitness function.
- To update the pheromone values; select the best ant feasible or infeasible and deposit the pheromone on path.
- To start the next generation. A new ant generation starts the route, when the ant ends update the pheromone values. Repeat this until the termination condition is satisfied.
- To obtain the optimal solution for SPSP.

5. Experimental results

In this section we present the experimental results. First, we describe the programing environment. We implement the algorithm in Java using a Intel core i7, 2,0 Ghz, 4 GB of RAM PC running Windows 7 Professional.

Second, we describe the characteristics of the instance set. We use instances generated by the same generator using in Xiao et al. (2013) and Alba and Chicano (2007). The generator [1] creates random instances, but we use the set of available instances with the same parameters used by Xiao and Alba. The instances have different number of tasks, employees, skills, and the task precedence graph (TGP). The instances are labeled as <employees>e_ <tasks>t_ <skills>s. The instance set used has $\{5, 10, 15\}$ employees, $\{10, 20, 30\}$ tasks, and $\{5, 10\}$ skills.

The algorithm was ran 10 trials for each instance and we report the average value from than 10 trials. To compare the different results we use the *hitrate*: feasible solutions in 10 runs, *cost*: average cost of feasible solutions, *duration*: average duration of feasible solutions, and *fitness*: average fitness of feasible solutions. The influence analyzes were conducted using the heuristic H1 for ACO-HC.

5.1. Parameter tuning

It is known that the ACO algorithms are sensitive to changes in parameters, that is why we conducted a series of experiments to find the best parameter values. We define two sets of parameters; fixed parameters such as number of iterations only determine the limit of iterations, α and β are important parameters that have been previously defined according to the literature and previous experiments. The variable parameters which we must define its value are m and ρ. These parameters can strongly affect the performance of the algorithm, so we tested different values.

We used 10e_10t_10s instance with *mind* = 0.25 to tune the variable parameters. *mind* is the lowest degree of dedication to a task and is defined by the problem, the fixed parameters used are $\alpha = 1$, $\beta = 2$, and number of iterations $N_{it} = 1000$. To guarantee the independence between m and ρ, we performed tests by fixing a variable and varying other, and vice versa. The results presented in the Fig. 6 indicate that the best results are obtained with the combination of $m = 200$ and $\rho = 0.02$. Then we conducted a series of experiments to display the convergence according iterations. In Fig. 7(a), we set ρ =0.02 and vary m. In Fig. 7(b) we set $m = 200$ and vary ρ.

We can observe in Fig. 7(a) that the fitness obtained with different number of ants. We demonstrate that the best fitness (low fitness) is obtained with $m = 200$ and $m = 300$ and the worst fitness is obtained with $m = 100$ and $m = 10$. To obtain the best results in

[1] http://tracer.lcc.uma.es/problems/psp/generator.html.

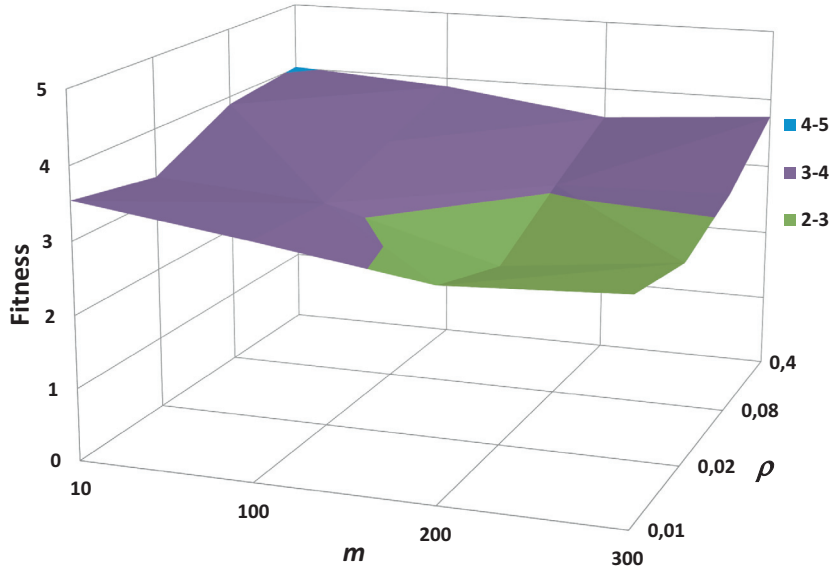

Fig. 6. Surface plot, with interaction, m and ρ.

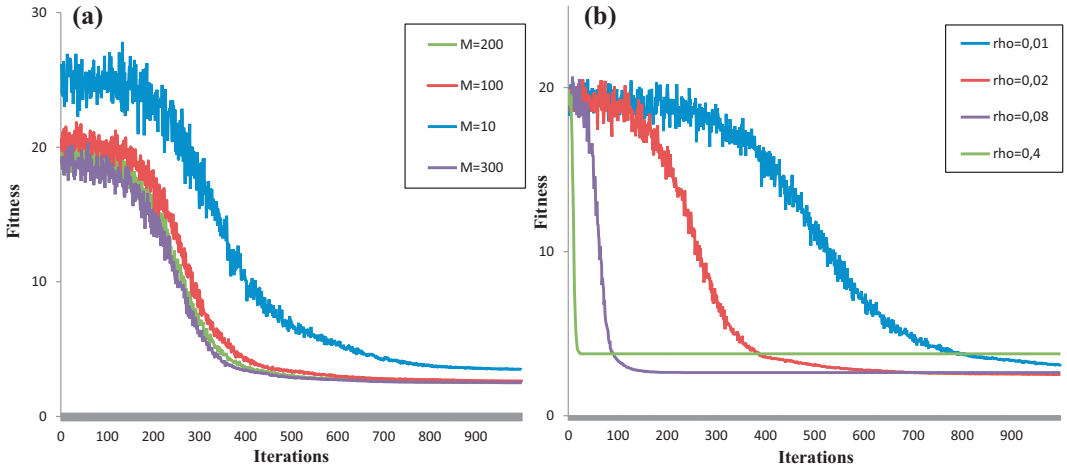

Fig. 7. *AvgFitness* using different values for m and ρ.

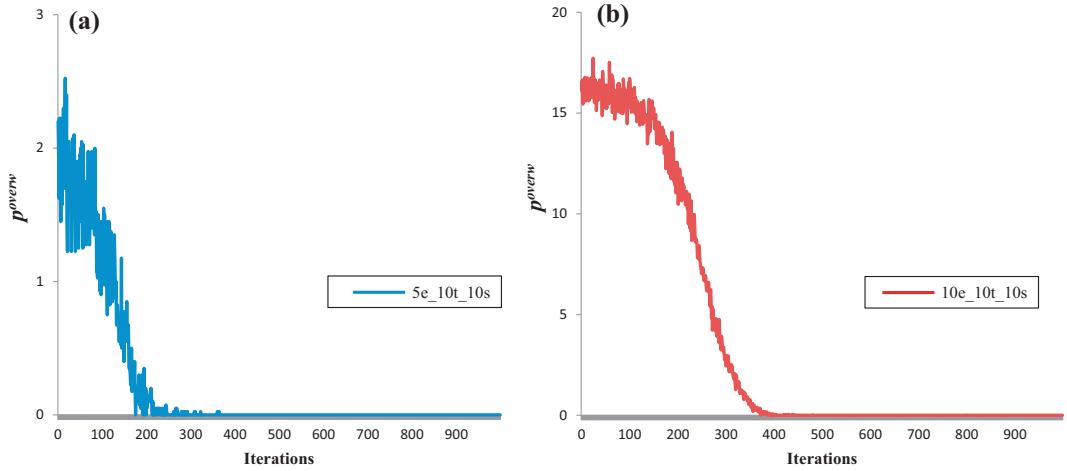

Fig. 8. Convergence to feasible solutions, Avg p^{overw} for 5e_10t_10s and 10e_10t_10s instances.

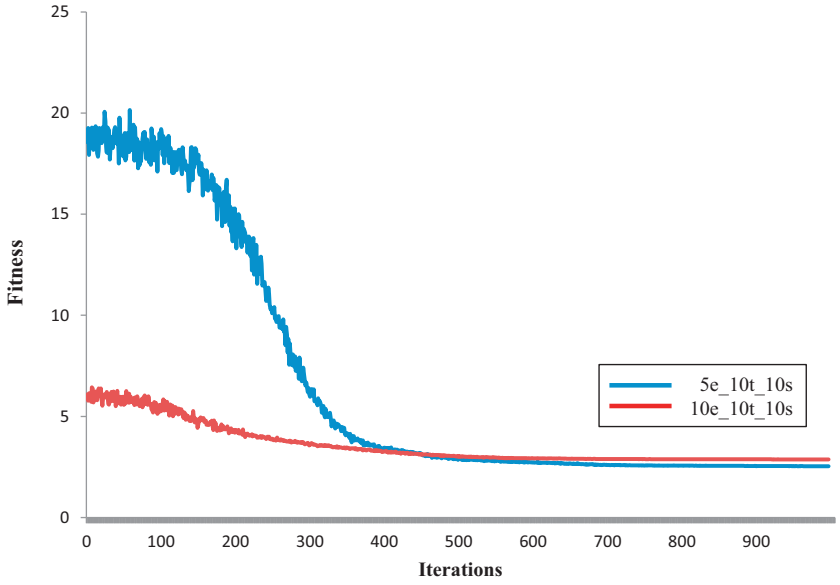

Fig. 9. Convergence to best solutions, Avg *fitness* for 5e_10t_10s and 10e_10t_10s instances.

shortest time, we used $m = 200$. With $m = 300$ the solutions have the same fitness to $m = 200$, but there is a higher computational cost of executing 300 instead of 200 ants.

In Fig. 7(b) we observe the different fitness obtained with $\rho = 0.01$, $\rho = 0.02$, $\rho = 0.08$, and $\rho = 0.4$, for this parameter the best fitness is obtained with $\rho = 0.02$. When $\rho = 0.01$ the fitness converge too slowly, if the ρ is very large (0.08 or 0.4) the fitness converges very fast to a suboptimal value, with $\rho = 0.02$ converges smoothly to fitness better value.

According to the data collected and analyzed, we can summarize the best parameters to use for our algorithm. For variable parameters have $m = 200$, $\rho = 0.02$, and the fixed parameters $\alpha = 1$, $\beta = 2$, $N_{it} = 1000$. these parameters will be used in the next experiments.

Table 3
Results with number of task = 10, 20, 30 using heuristic H1.

Instance	Task	Hit rate	Fitness	p^{len}	p^{cos}
5e_10t_5s	10	100	3.136531	23	836531
5e_20t_5s	20	30	8.566778	64	2166660
5e_30t_5s	30	–	–	–	–
10e_10t_5s	10	100	2.134546	13	834546
10e_20t_5s	20	30	6.741111	45	2241111
10e_30t_5s	30	–	–	–	–
5e_10t_10s	10	100	2.875215	20,7	805790
5e_20t_10s	20	20	10.720411	83	2419718
5e_30t_10s	30	–	–	–	–
10e_10t_10s	10	100	2.612948	17	912948
10e_20t_10s	20	50	6.249782	42	2049782
10e_30t_10s	30	–	–	–	–

Table 4
Results with number of employees = 5,10,15.

Instance	Employees	Hit rate	Fitness	p^{len}	p^{cos}
5e_10t_10s	5	100	2.875215	20.7	805764
10e_10t_10s	10	100	2.612948	16.2	916730
15e_10t_10s	15	100	1.906068	12	706077

5.2. Convergence analysis

In order to analyze the convergence of the algorithm, we use the best settings obtained according to the results presented in the previous section with two different instances. To demonstrate de convergence of ACO-HC algorithm to feasible solutions, we present the average of project overwork p^{overw} in 10 runs and 1000 iterations for 10e_10t_10s and 5e_10t_10s instances. We consider a solution feasible if can make the project complete and project overwork is 0 (i.e., $p^{overw} = 0$).

In Fig. 8 we present the convergence of two instances using ACO-HC. In the instance 5e_10t_10s (Fig. 8(a)) the initial solutions are not feasible, but around 200 iteration p^{overw} converges to 0. In the instance 10e_10t_10s (Fig. 8(b)) the convergence to feasible solution is at 400 iterations, we observe a difficult to converge to feasible solutions. This does not imply better solutions, only indicates when it is possible to obtain feasible solutions.

We can see in Fig. 9 the convergence to better solution per instance using ACO-HC. For instance 10e_10t_10s (red line) we can obtain better solutions from iteration 400 and converges slowly to a best solution, in instance 5e_10t_10s (blue line) we can obtain better solutions from iteration 200, but converges to a solution faster. In addition, we compare the best solutions, because they are different instances.

5.3. Influence of task number

We used 12 instances to analyze the influence of the number of tasks on the solutions. The instances have the same values per each of 4 groups. We compare the number of task with $\{10, 20, 30\}$, the number of employees and skills remain constant for each group. The results are show in Table 3.

First we observe that the hit rate decreases when the number of tasks increases and is more difficult to find feasible solutions. When $t = 30$ was not possible to find feasible solutions in the four groups using the heuristic H1. We can infer that increasing tasks employees have more work and is more difficult to meet $p^{overw} = 0$. Furthermore we can see that increasing tasks directly increases the cost and duration of the whole project.

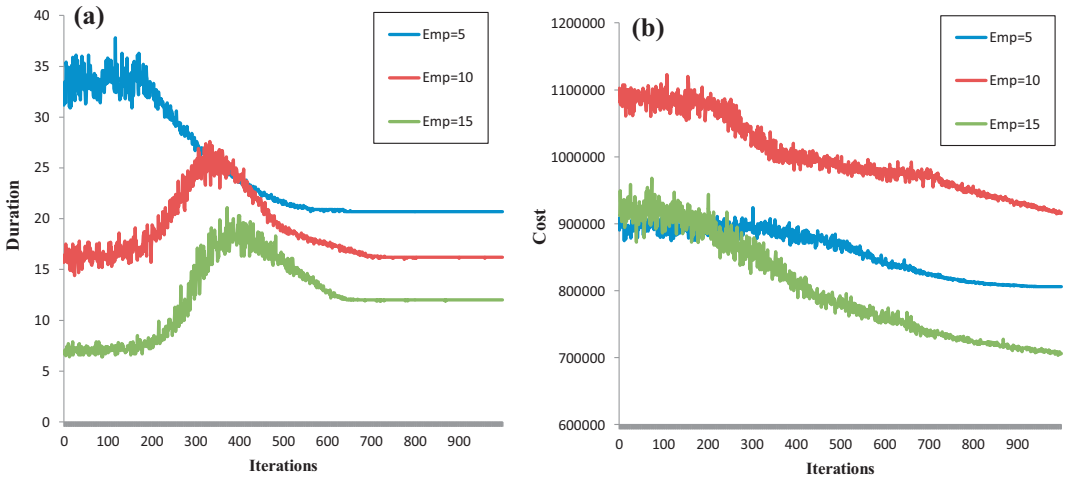

Fig. 10. Duration (a) and cost (b) with different number of employees.

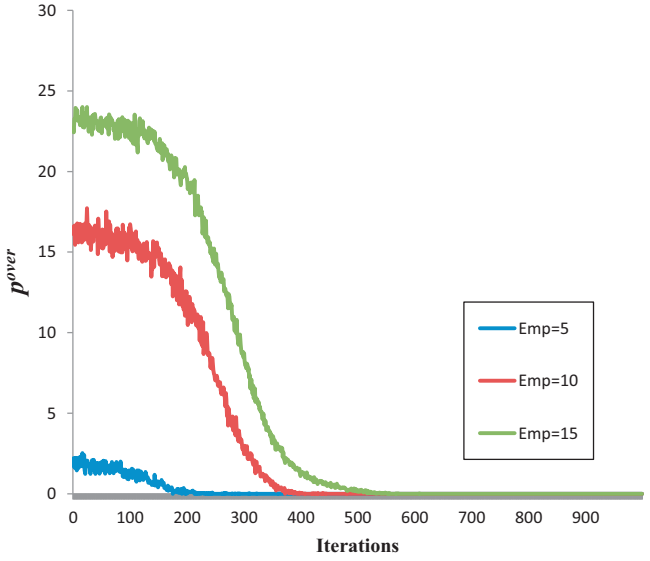

Fig. 11. Convergence of p^{overw} with different number of employees.

Table 5
Results with number of skills = 5,10.

Instance	Skill	Hit rate	Fitness	p^{len}	p^{cos}
5e_10t_5s	5	100	3.136531	23	828603
5e_10t_10s	10	100	2.875215	20	812555
5e_20t_5s	5	30	8.566778	64	2166660
5e_20t_10s	10	20	10.720411	83	2422418
10e_10t_5s	5	100	2.134546	13	819925
10e_10t_10s	10	100	2.612948	15	905118
10e_20t_5s	5	30	6.741111	45	2241111
10e_20t_10s	10	50	6.249782	50	1937460

5.4. Influence of employees number

We used 3 instances to analyze the influence of the number of employees on the solution. We only use three instances that represent the range of possible values for the number of employees in the available instances. These instances represent the minimum value, a mean value and a high value of employees. We compare the number of employees with $\{5, 10, 15\}$, the number of task and skill remain constant for each group, the maximum dedication of employees and salaries are the same. The results are shown in Table 4.

Table 6
Comparison with other techniques.

Instance	Algorithms	Hit rate	S	Fitness
5e_10t_5s	ACO-HC (H1)	100	0.0329	3.1365
	ACO-HC (H2)	100	0.0515	**2.7750**
	ACS	100		3.5149
	GA	95		3.5874
5e_10t_10s	ACO-HC (H1)	100	0.0447	**2.8752**
	ACO-HC (H2)	100	0.0749	3.3449
	ACS	100		3.4049
	GA	90		3.5287
10e_10t_5s	ACO-HC (H1)	100	0.0075	2.1345
	ACO-HC (H2)	100	0.0423	**2.0967**
	ACS	100		2.5773
	GA	97		2.8413
10e_10t_10s	ACO-HC (H1)	100	0.0927	2.6129
	ACO-HC (H2)	100	0.0405	**2.2660**
	ACS	100		2.6440
	GA	100		2.5371
10e_20t_5s	ACO-HC (H1)	30	0.2462	6.7411
	ACO-HC (H2)	90	0.3338	6.8769
	ACS	67		6.3856
	GA	19		**6.2766**
10e_20t_10s	ACO-HC (H1)	50	0.5054	6.2497
	ACO-HC (H2)	100	0.1321	**5.5963**
	ACS	65		6.2984
	GA	71		6.1869
10e_30t_5s	ACO-HC (H1)	-	-	-
	ACO-HC (H2)	70	1.8055	**10.6546**
	ACS	-	-	-
	GA	-	-	-
10e_30t_10s	ACO-HC (H1)	-	-	-
	ACO-HC (H2)	90	0.4124	**10.1649**
	ACS	-	-	-
	GA	-	-	-

The best fitness per instance is bold.

First we note that the only resource for this problem is human resources (employees) and the duration of the tasks is directly related to the use of this resource. Given that the duration of a task vary according to workers assigned to that task, it is important to show that this interaction can vary the duration of the project. We only interpret the data presented in the graphs, which are only

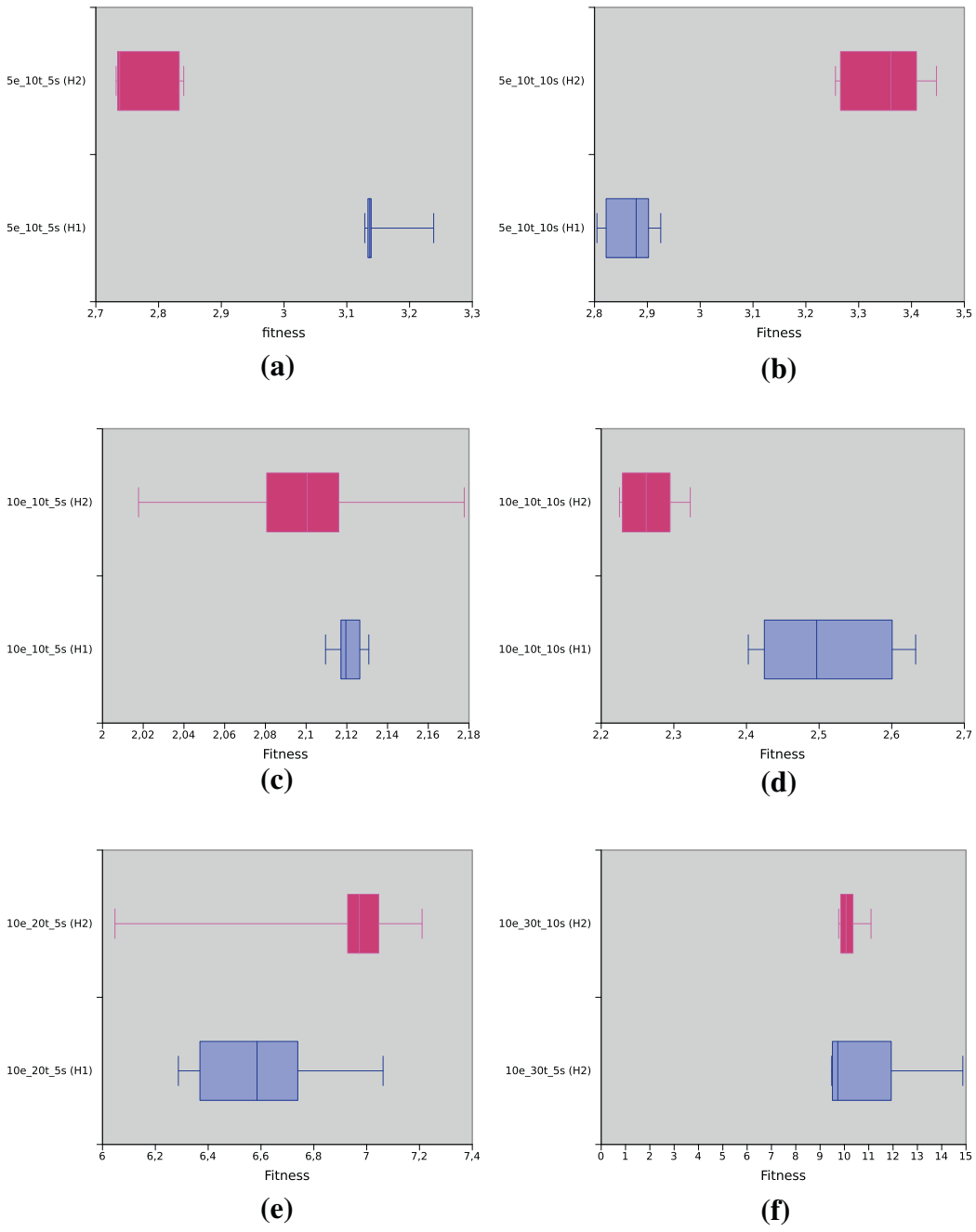

Fig. 12. A comparative boxplot with heuristics (H1) and (H2) with different instances.

a particular case study based on the tested instances. These results are not generalizable to real cases; they lack complex situations that can occur in actual software projects.

We compared the results and found that by increasing the number of employees decreases the project duration. The hit-rate for these instances is always 100%, but we cannot find a direct relation with the project cost. When the number of employees is 5 the project cost is greater than when using 15. We can see this in the Table 4 and Fig. 10.

As it is show in Table 4 the fitness of the solutions has a direct relationship to the number of employees, when the number of employees increases also increases the fitness. If we observe convergence to feasible solutions on the Fig. 11, with increasing the number of employees is more difficult to find feasible solutions. The algorithm converges more slowly when the number of employees is greater.

5.5. Influence of skill number

In order to analyze the influence of the required skills for task on the solutions, we used 8 instances. The instances have the same values per each of 4 groups. We compare the number of skills with $\{5, 10\}$, the number of employees and tasks remain constant for each group. The results are show in Table 5.

According to the table, the instances with $t = 10$ all have a hit rate of 100%, while instances with $t = 20$ change depending on the number of skills. We can see that for instances with $t = 20$, increasing the number of skills required for a task, it is more diffi-

cult to find feasible solutions. The results do not allow us to determine a direct relationship, between the increase in the skills required by a task and an increase in the duration or cost of the whole project.

5.6. Comparative results with other techniques

To evaluate our proposal, we decided to compare it with other techniques. This presents multiple problems, because the instances are generated according to the parameters entered by each researcher. Another problem is that we have not found published results that clearly indicate all the parameters of their instances.

Some results are presented in Xiao et al. (2013) by Xiao, using the similar parameter to our instances. Xiao presents results using Ant Colony System (ACS) and Genetic Algorithms (GA). To understand clearly we transform the fitness presented by the author as fitness^{-1} to obtain the same fitness used by us. The comparative results are presented in Table 6.

From Table 6 we can compare the hit rate and the fitness of the solutions. In this case for the instances with task = 10 always have a hit rate of 100% for all numbers of employees or skills. But in the instances with task = 20, ACS has better hit rate in one instance.

Regarding the fitness we can see that ACO-HC (H1 and H2) has better results for all instances with task = 10. On the other hand, for instances with 10e_20t_5s, the best results are using GA, but in 10e_20t_10s the best result is obtained with ACO-HC (H2). If we analyze the results with task = 30, the best results is obtained with ACO-HC (H2), the other algorithms could not obtain results. For all instances with task = 30 we used $N_{it} = 2500$.

Then we can compare the two heuristics H1 and H2. We can observe the standard deviation presented in the Table 6. We present comparative boxplot between H1 and H2 in Fig. 12. The results show that H2 performs better than H1, but H1 has a lower standard deviation in small instances, so we could indicate that the values are more concentrated on average.

6. Conclusion

We have presented a new approach to solve the SPSP using a Max–Min Ant System algorithm. As a result, we propose implement the Hyper-Cube framework for ACO to improve the algorithm.

The design we proposed is a representation of the problem in order to ACO algorithm can solve it, proposing a construction graph and two pertinent heuristics. Furthermore, we defined a proper fitness function able to allow optimization of the generated solutions.

We present the experimental results for our proposed algorithm, and we conducted a series of tests to get the best parameterization and evaluate the convergence to better solutions. The tests were performed using different numbers of tasks, employees, and skills.

The results were compared with Ant Colony System and Genetic Algorithms. We demonstrate that our proposal gives the best results for smaller instances. For more complex instances was more difficult to find solutions, but our solutions always obtained a low cost of the project, in spite of increasing the duration of the whole project. For the instances with 30 tasks only solutions are obtained with heuristic H2.

Our contribution in this research was to take a problem which has been little exploited and have proposed a new algorithm to solve it. For this, we have taken two approaches; the solution through the Max–Min Ant System and Hyper-Cube framework for ACO, which have been tailored to meet SPSP. We have shown empirically that the new algorithm is able to deliver better solutions with better hit rate for multiple instances tested. Therefore it is a feasible and robust proposal. SPSP is a problem that brings us closer to the estimation of software projects, but still has simplified features that should be changed to real cases.

As future work, we hope to integrate the search autonomously, to improve the algorithm and the search for solutions. There are studies that have reported good results with these techniques (Crawford, Soto, Castro, & Monfroy, 2011; Crawford et al., 2013b; Monfroy et al., 2013).

Acknowledgments

Broderick Crawford is supported by Grant CONICYT/FONDECYT/REGULAR/ 1140897. Ricardo Soto is supported by Grant CONICYT/FONDECYT/INICIACION/ 11130459. Fernando Paredes is supported by Grant CONICYT/FONDECYT/REGULAR/ 1130455.

References

Abdallah, H., Emara, H. M., Dorrah, H. T., & Bahgat, A. (2009). Using ant colony optimization algorithm for solving project management problems. *Expert Systems with Applications, 36*(6), 10004–10015.

Alba, E., & Chicano, F. (2007). Software project management with gas. *Information Sciences, 177*(1), 2380–2401.

Barreto, A., Barros, M. d. O., & Werner, C. M. L. (2008). Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers and Operations Research, 35*(10), 3073–3089.

Berrichi, A., Yalaoui, F., Amodeo, L., & Mezghiche, M. (2010). Bi-objective ant colony optimization approach to optimize production and maintenance scheduling. *Computers and Operations Research, 37*(9), 1584–1596.

Blum, C., & Dorigo, M. (2004). The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 34*(2), 1161–1172.

Brucker, P., Drexl, A., Mhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research, 112*(1), 3–41.

Chang, C. K., yi Jiang, H. Y., Di, Y., Zhu, D., & Ge, Y. (2008). Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology, 50*(11), 1142–1154.

Chen, R.-M. (2011). Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem. *Expert Systems with Applications, 38*(6), 7102–7111.

Chen, W., & Zhang, J. (2013). Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering, 39*(1), 1–17.

Christina, P., Miriam, D. (2012). Adaptive task scheduling based on multi criterion ant colony optimization in computational grids. In *2012 international conference on recent trends in information technology (ICRTIT)* (pp. 185–190).

Crawford, B., Soto, R., Castro, C., & Monfroy, E. (2011). Extensible cp-based autonomous search. In *Proceedings of HCI international. CCIS* (Vol. 173, pp. 561–565). Springer.

Crawford, B., Soto, R., Johnson, F., & Monfroy, E. (2013a). Ants can schedule software projects. In C. Stephanidis (Ed.). *HCI international 2013 – posters' extended abstracts, communications in computer and information science* (Vol. 373, pp. 635–639). Berlin Heidelberg: Springer.

Crawford, B., Soto, R., Monfroy, E., Palma, W., Castro, C., & Paredes, F. (2013b). Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization. *Expert Systems with Applications, 40*(5), 1690–1695.

Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation* (pp. 1470–1477). IEEE Press.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 26*(1), 29–41.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. USA: MIT Press.

Fahmy, A., Hassan, T. M., & Bassioni, H. (2014). Improving {RCPSP} solutions quality with stacking justification – application with particle swarm optimization. *Expert Systems with Applications* (0).

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research, 207*(1), 1–14.

Johnson, F., Crawford, B., Palma, W. (2006). Hypercube framework for ACO applied to timetabling. In *IFIP AI* (pp. 237–246).

Laalaoui, Y., & Bouguila, N. (2014). Pre-run-time scheduling in real-time systems: Current researches and artificial intelligence perspectives. *Expert Systems with Applications, 41*(5), 2196–2210.

Liao, T. W., Egbelu, P., Sarker, B., & Leu, S. (2011). Metaheuristics for project and construction management – a state-of-the-art review. *Automation in Construction, 20*(5), 491–505.

Luna, F., González-Álvarez, D. L., Chicano, F., & Vega-Rodríguez, M. A. (2014). The software project scheduling problem: A scalability analysis of multi-objective metaheuristics. *Applied Soft Computing, 15*(0), 136–148.

Monfroy, E., Castro, C., Crawford, B., Soto, R., Paredes, F., & Figueroa, C. (2013). A reactive and hybrid constraint solver. *Journal of Experimental and Theoretical Artificial Intelligence, 25*(1), 1–22.

Nan, N., & Harter, D. E. (2009). Impact of budget and schedule pressure on software development cycle time and effort. *IEEE Transactions on Software Engineering, 35*(5), 624–637.

Ozdamar, L., & Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. *IIE Transactions, 27*(5), 574–586.

Rubio, J. M., Crawford, B., Johnson, F. (2008). Solving the university course timetabling problem by hypercube framework for ACO. In *ICEIS* (2) (pp. 531–534).

Stützle, T., & Hoos, H. H. (2000). Max-min ant system. *Future Generation Computer Systems, 16*(8), 889–914.

Tavana, M., Abtahi, A.-R., & Khalili-Damghani, K. (2014). A new multi-objective multi-mode model for solving preemptive time-cost-quality trade-off project scheduling problems. *Expert Systems with Applications, 41*(4, Part 2), 1830–1846.

Wu, S., Wan, H.-D., Shukla, S. K., & Li, B. (2011). Chaos-based improved immune algorithm (CBIIA) for resource-constrained project scheduling problems. *Expert Systems with Applications, 38*(4), 3387–3395.

Xiao, J., Ao, X. T., & Tang, Y. (2013). Solving software project scheduling problems with ant colony optimization. *Computers and Operations Research, 40*(1), 33–46.