ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20160629224348
Original scientific paper

Solving the Software Project Scheduling Problem Using Intelligent Water Drops

Broderick CRAWFORD, Ricardo SOTO, Gino ASTORGA, Carlos CASTRO, Fernando PAREDES, Sanjay MISRA, José-Miguel RUBIO

Abstract: Within the category of project scheduling problems, there is a specific problem within the software industry referred to as the software project scheduling problem. The problem consists in the correct allocation of employees to the different tasks that make up a software project, bearing in mind time and cost restraints. To achieve this goal, the present work first uses metaheuristic intelligent water drops illustrating; this is a recent stochastic swarm-based method increasingly used for solving optimization problems. Finally, the results and comparisons with experiments performed with other techniques are presented, demonstrating the solidity of the approach presented.

Keywords: combinational optimization; computational intelligence; intelligent water drops; metaheuristics; software project scheduling problem

1 INTRODUCTION

Given the current competitive world, it is necessary to optimize certain activities within the software project scheduling problem where, due to the complexity of activities confronted by project managers, it is necessary to initiate, plan, execute and close a project.

The Software Project Scheduling Problem (SPSP) is a specific case of the Project Scheduling Problems (PSP) family; in this problem, employees are allocated to tasks with the objective of minimizing the cost and duration of a project while taking into account the precedence of tasks and resource constraints [1, 2]. It is related to another problem type called the Resource Constraint Project Scheduling Problem (RCPSP), which finds an optimal schedule that meets the precedence requirement and minimizes the project duration and cost.

Scheduling problems can be classified considering various factors such as the amount of stages in the job process, the amount of machines present for each stage, different job processing requirements, setup time/cost requirements and the performance measure to be optimized [3]. The SPSP is an NP-hard problem [4]. The SPSP is used to organize many activities that require various resources; these resources may or may not be renewable and comply with the objectives set out above. This problem has as its central element the human resource, which is distributed to activities based on skills. An important feature is that PSP [5] has various goals, whereas the main objective of SPSP is to ensure that the project cost and duration are as low as possible [6].

Unless the project that we need to schedule is similar to a previous one, this is one of the most difficult tasks that project managers have to face. They need to estimate the time and resources required to complete each of the activities and organize them into a coherent sequence.

The project scheduling activity consists in dividing the total project into a series of activities, which may be performed in parallel, and estimating the necessary time for each of them. It must be ensured that the handwork is used optimally [7], as shown in Fig. 1.

The problem SPSP consists of assigning a set of tasks to a set of employees, thus introducing the difficulty of each employee possessing the skills necessary to complete this task. The allocation must meet the requirement that a project incurs minimal cost and is completed in the least amount of time. The employees have a salary and possess several skills that allow them to work on several tasks during a working day. The main methods used to solve this problem correspond to the set of techniques based on priority rules or classes of metaheuristics.

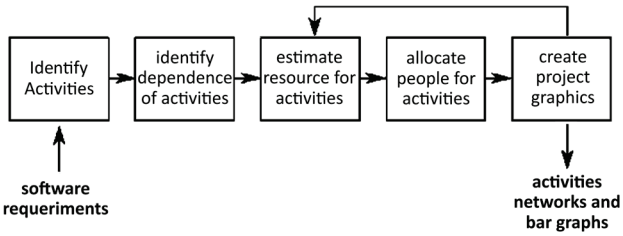

Figure 1 Project scheduling process

The SPSP has been resolved using incomplete techniques; Tab. 1 shows works related to Scheduling Problems:

In [8], genetic algorithms (GAs) are used to solve many different software project scenarios; [9] uses Genetic Algorithms to present a Time-line based model; [4] proposes an ant colony optimization (ACO) approach known as ACS-SPSP algorithm. Moreover, [10] solves SPSP using a max-min ant system with a hyper-cube framework; [11] analyzed the scalability of eight multi-objective algorithms using instances of increasing size for the SPS problem. Finally, Biju proposes a differential evolution (DE) method that is a direct search stochastic optimization technique that is fairly fast and robust.

Table 1 Summary of similar works

Technique	Reference
Ant colony optimization (ACO)	[4]
Genetic algorithms (GA)	[8]
Time-line with genetic algorithms	[9]
Hyper-cube Ant Colony	[10]
Scalability analysis of multi-objective metaheuristics solving SPSP	[11]
Differential evolution (DE) algorithm	[12]

Intelligent Water Droplets (IWD) [13] is a relatively new Metaheuristic based fundamentally on the behavior of water droplets as they move in a natural way from one point to another on a river bank. It corresponds to a probabilistic type approach to solving computational problems, improving the search for good ways through

graphics. IWD can effectively solve various Combinatorial NP-hard problems.

IWD can be classified as an intelligence of swarm. Swarm Intelligence is an attractive area of research in the field of artificial intelligence where simple agents and artificial elements are mutually supportive for solving complex problems.

The proposed algorithm is implemented and different instances are executed to study the convergence and obtain solutions of good quality. Moreover, we performed different tests to get a better parametrization. The instances considered, for example, various amounts of iterations. They also tested different amounts of skill, tasks and employees. The results obtained were compared with previous works that used other techniques: Hyper-Cube Ant Colony (MMAS-HC) and Ant Colony System (ACO).

The contribution of this work is to build a different solution for the SPSP adapting the metaheuristic IWD. This work is organized in the following way. Section 2 presents a detailed explanation of the SPSP. The IWD algorithm is presented in detail in Section 3. Section 4 shows the design of the solution for SPSP including subsections presenting the construction graph, update of the soil rules and the algorithm used. Section 5 presents an example of the instance used. Section 6 presents the results obtained, and Section 7 presents the conclusions.

2 PRESENTATION OF THE PROBLEM TO SOLVE

The SPSP is considered a recurring problem in the management of software companies. Its goal is to find the correct allocation of staff to project tasks. The SPSP should consider the remuneration of employees and their skills in the assignment of the needs of each task [8]. In SPSP, the main resources are: the task, i.e., the activities to be completed during the execution of the project, and the employees, i.e., those who must work on different tasks. This approach should allocate staff with the necessary skills to the appropriate tasks.

Different skills are required to complete the various tasks that employees must have in order to be assigned to those tasks. Skills can be as follows: senior programmer, junior programmer, interfaces expert, database expert, leader.

$S = \{s_1, \ldots, s_{|S|}\}$, where $|S|$ corresponds to number of skills.

The tasks of the project are all the necessary activities to be performed to complete the software project. These activities may include analysis, design of different components, its programming, its documentation and the entire testing process. The precedence of the tasks is an important issue because a software project consists of sequential tasks that must have an established precedence. The Task Precedence Graph (TPG) is used to control task precedence. This is a non-cyclical graph denoted as $G(E, V)$. The group of tasks corresponds to $V = \{t_1,\ldots, t_{|T|}\}$. The relationship between the order of the tasks is shown by a group of edges. The edge $(t_i, t_j) \in E$ indicates that the task t_i is the predecessor task of t_j. Therefore, the set of tasks required for the project is defined as: $T = \{t_1,\ldots, t_{|T|}\}$, where $|T|$ is the number of tasks. Every task presents two characteristics:

- t_j^{sk} is the set of skills for task j. This is a subset of S corresponding to all the necessary skills to complete the task j.
- t_j^{eff} corresponds to the skill set for executing task j. It is a real number, a subset of S, which corresponds to the workload of task j.

Table 2 Description of the elements for SPSP model

Name	Description
$S = \{s_1, \ldots, s_{\|S\|}\}$	Is the set of skills required for the project
$T = \{t_1,\ldots, t_{\|T\|}\}$	Is the group of tasks necessary for the project
$G(E,V)$	Is the group of tasks necessary to complete the project
$V = \{t_1,\ldots, t_{\|T\|}\}$	Group of vertex consisting of the different tasks
$E = \{t_1, t_2,\ldots,t_n, t_{\|T\|}\}$	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 required person per month to realize task j.
$EMP = \{e_1,\ldots,e_{\|E\|}\}$	Set of employees available for the project
e_i^{sk}	Set of skills of e_i. It is a subset of S.
$e_i^{\max d}$	Maximum dedication of e_i, $e_i E[0,1]$
e_i^{rem}	Monthly remuneration of e_i
$M = m_{ij}$	Dedication of employee i to task j
t_j^{init}	The start time of task j
t_j^{term}	Time to end of task j
$t_j^{\cos}$	Cost of task j
t_j^{len}	Time to complete task j
p^{init}	The time required to complete the entire project
$p^{\cos}$	Total cost of the project
$pt_i^{inoverwit}$	Overtime work of employee e_i
p^{overw}	Extra hours work of the entire project

The human resources are the main element in this problem; they must have diverse skills, and usually they are engineers with corresponding skills. The project has a group of personnel who are employed at tasks. The project manager must assign the personnel ideal for each activity. The difficulty is to implement proper programming where employees are assigned to the appropriate task. $EMP = \{e_1,\ldots,e_{|E|}\}$ represents a group of persons where $|E|$ corresponds to the personnel quantity involved in the project. Every worker has three characteristics:

- e_i^{sk} is the set of the employee skills i. $e_i^{sk} \subseteq S$.
- $e_i^{\max d}$ represents the maximum degree of work j.

This is a value of the hours of work assigned to the project in a working day $e_i^{\max d} \in [0,1]$. If the $e_i^{\max d} = 1$, it means that the dedication is the entirety of the project. If $e_i^{\max d}$ is less than one, this is a partial worker. For example: if $e_i^{\max d} = 0.25$, the employee has only 1/4 of dedication to the project.

- e_i^{rem} represents the monthly remuneration of employee i, being a real value.

2.1 Model Description

The principal components used by the SPSP model are: tasks, employees and their corresponding skills. The components of this model are shown in Tab. 2.

An array $|E \times T|$ can be used to represent this problem. The amount of human resources and the project activities determine the size of the array $|E| \times |T|$. The elements of the array $m_{ij} \in [0,1]$ are a real number that represents the dedication of the i employee to the task j. If $m_{ij} = 0$, the human resource is not assigned to task j. If $m_{ij} = 1$, the human resource is assigned to working on task j all day. For example, if $m_{ij} = 0.5$, employee i uses only half a day for the project.

The generated solution is an array that is not always feasible. This happens, for example, when all the elements of column i are 0, indicating that there are no employees dedicated to task j. This solution is not feasible because the j task cannot be finished. Given the above, some constraints are defined for the purpose of obtaining feasible solutions from the array M.

- The tasks are given as a minimum to an employee as shown in Eq. (1).

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

- Eq. (2) shows the employees assigned to tasks j, where the skills needed for task t_j are a subset of the union of the skills of the employee assigned to the task.

$$t_j^{sk} \subseteq \bigcup e_i^{sk} \forall j \in \{1,...,T\} \tag{2}$$

where t_j^{sk} corresponds to the skill required in task j and e_i^{sk} represents the skills of the employee i.

Fig. 2 represents an example of TPG task precedence and the necessary skills $t^{\max}$ accompanied by the effort required t^{eff}. In this case, we have several employees, $EMP = \{e_1, e_2, e_3, e_4\}$, and for each of them, there is a group of skills, a maximum quantity of dedication and one correspondent payment.

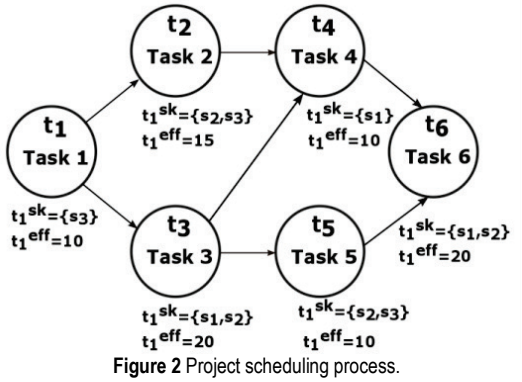

Figure 2 Project scheduling process.

One solution is to complete the Marray so that tasks are dedicated to employees who have the right skills. An example of what is raised is in Tab. 3.

Table 3 Example of solution for array M

$M_{(i,j)}$	t_1	t_2	t_3	t_4	t_5	t_6
e_1	0.00	0.50	0.00	1.00	0.25	0.00
e_2	0.25	0.00	0.75	0.00	0.50	1.00
e_3	1.00	0.00	0.25	0.25	0.00	0.00
e_4	0.00	0.75	0.00	0.25	0.00	0.50

First, it is necessary to validate the viability of the result, for which we use the length of the tasks and cost of the project. The length of each activity is considered as t_j^{len}, $j \in \{1, 2,..., |T|\}$; for this, we occupy the array M and t_j^{eff} :

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

Now, we need to get the start t_j^{init} and the termination t_j^{term} of each task j. To obtain that value, we use the precedence relation $TPG(V, E)$. The calculation of the beginning of each task is given by Eq. (4).

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

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

With this, we have the start time t_j^{init}, the end time t_j^{term} and the duration for j task with $j = \{1,..., |T|\}$. This is used to construct the Gantt chart of the project (Fig. 3). This reflects a program that does not have a stagnation time. For cases where it occurs, it should be extended to the end of tasks t_j^{term}. This requires knowing the length of the final task to determine the full duration of project p^{len}; this is shown in Eq. (6).

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

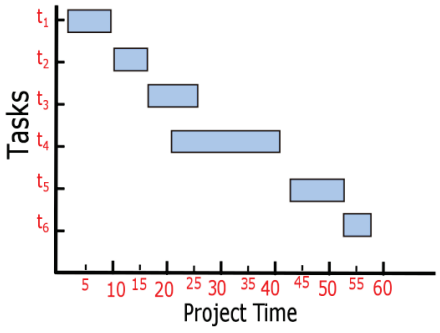

Figure 3 Project scheduling process

To know the total cost, it is necessary to have the cost associated with all the tasks as $t_j^{\cos}$ with $j \in \{1, 2,..., |T|\}$ using Eq. (7).

$$t_j^{\cos} = \sum_{i=1}^{|E|} e_{ij}^{rem} m_{ij} t_j^{len} \quad (7)$$

$$p^{\cos} = \sum_{i=1}^{|T|} t_j^{\cos} \quad (8)$$

The objective is to determine the minimum duration p^{len} and the cost $p^{\cos}$; consequently, we use a fitness function where w^{len} and $w^{\cos}$ are real values. For this, $w^{\cos}$ is assigned with the length $Average^{-1}$. Then, we multiply the parameters by the corresponding $p^{\cos}$ and w^{len}. The function fitness that it is necessary to minimize in this problem is the following:

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

An element that is not considered is overwork. Since it increases the cost and time associated with each task, $p^{\cos}$ and p^{len} are consequently increased in the software project. Overwork is defined as e_i^w. The equation is based on the workload of the human resource, and time t is used to calculate it. This is presented in Eq. (10).

$$e_i^w(t) = \sum_{\{t_j^{init} \le t \le t_j^{term}\}} m_{ij}(t) \quad (10)$$

If employee e_i experiences overwork e_i^w, the work at t moment is greater than the maximum dedication, e.g., $e_i^{\max d}(t) > e_i^{\max d}$. To calculate overwork $rampx(x)$, the following equation is defined.

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

Calculating employee overtime for the totality of the project is achieved with the following equation.

$$e_i^{overw} = \sum_{t=0}^{p^{len}} ramp(e_i^w(t) - e_i^{\max d}) \quad (12)$$

To obtain the extra work of project p^{overw}, the totality of the employees is considered. In this case, we use Eq. (13).

$$p^{overw} = \sum_{i=1}^{|E|} (e_i^{overw}) \quad (13)$$

Once we have all the values, validate if the solution is feasible when this is completed for all tasks, and without extra work, $p^{overw} = 0$.

3 GENERAL DESCRIPTION OF THE METAHEURISTIC

In the year 2007, Hamed Shah-Hosseini [14] introduced a new algorithm called IWD to solve the Traveling Salesman Problem (TSP). These metaheuristics are based on the ideal trajectory of water drops moving from one point to another in a river. It is considered a constructive metaheuristic similar to ACO. The IWD is a recent metaheuristic appropriate for combinatorial optimization problems [15].

This trip generates three important actions:

- The drop increases your velocity.
- Soil joins the drop.
- The soil is decreased where the drop passes.

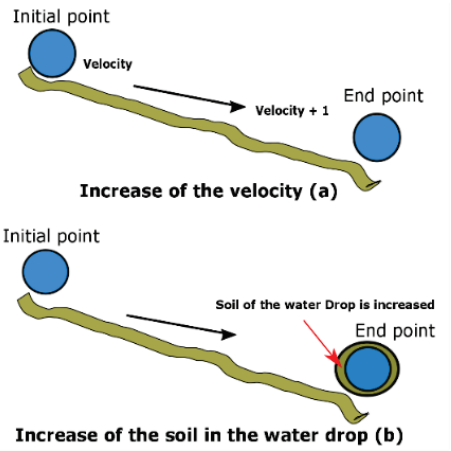

Figure 4 Water drop behavior

When a drop moves from an initial point to a new point, the velocity is affected in an increasing way; this change is shown in Fig. 4(a); moreover, during displacement, the water drop incorporates the soil into the drops structure, as shown in Fig. 4(b). Additionally, the soil incorporated into the drop is extracted from the soil of the river bed; therefore, the displacement of the drop from one point to another is associated with two actions: It reduces soil in the river and incorporates the extracted soil into the water drop, as shown in Fig. 5(a). Consequently, the velocity at which a drop moves it is very important; given two water drops of similar size and route, a higher velocity will cause a greater collection of soil, as shown in Fig. 5(b).

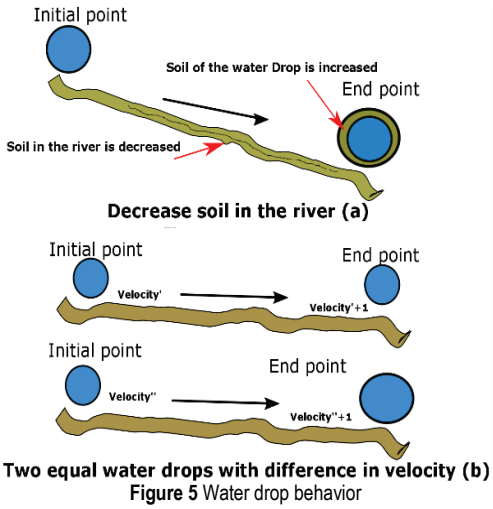

Figure 5 Water drop behavior

For each drop, the initial velocity and the quantity of soil where it should pass are considered. The drops pass by a certain amount of nodes, being a discreet space. The drop to decide which node to pass considers the existing soil between the origin and destination nodes. The

probability of choice of the new node is given by the following equation:

$$p_i^k(j) = \frac{f(soil(i,j))}{\sum_{\forall \notin l^k_{Visited}} f(soil(i,l))} \quad (14)$$

This probability equation is formed by $f(soil(i,j))$, which is obtained with the following equation:

$$f(soil(i,l)) = \frac{1}{\varepsilon + h(soil(i,j))} \quad (15)$$

where ε is a small value greater than zero to avoid division by zero and $h(soil(i,j))$ is used to choose the amount $soil(i,l)$ that connects nodes i with j by an amount greater than zero, obtained using the following equation:

$$h(soil(i,j)) \begin{cases} soil(i,j) \text{ if } \min(soil(i,l)) \geq 0 \\ soil(i,l) - \min(soil(i,l)) \text{ otherwise} \end{cases} \quad (16)$$

The function $\min(soil(i,l)$ returns the minimum value of its argument, and as denoted in the formula above, $soil(i,l)$ corresponds to each of the possible nodes to visit, where lE indicates the consult for the minimum soil value among those that do not belong to the vector of nodes already visited by the drop.

Fig. 6, represents an example of selection probabilities where a water drop found in node A is required to jump to a new node, B, C, D, E or F. In this example, the arc between nodes corresponds to the soil. The probability of election of every node is calculated as follows:

$$sum = (\frac{1}{20} + \frac{1}{30} + \frac{1}{10} + \frac{1}{40} + \frac{1}{80}) \quad (17)$$

$$(0.05 + 0.0333 + 0.1 + 0.025 + 0.0125) = 0.22083 \quad (18)$$

$$P(B) = \frac{\frac{1}{20}}{0.22083} = 0.2264 \quad (19)$$

$$P(C) = \frac{\frac{1}{30}}{0.22083} = 0.1509 \quad (20)$$

$$P(D) = \frac{\frac{1}{10}}{0.22083} = 0.4528 \quad (21)$$

$$P(E) = \frac{\frac{1}{40}}{0.22083} = 0.1132 \quad (22)$$

$$P(F) = \frac{\frac{1}{80}}{0.22083} = 0.0566 \quad (23)$$

Where D has the major probability of being selected due to the lower value of the soil.

Whenever a drop jumps from one node to another, its velocity increases. The new speed is calculated as follows:

$$vel^k(t+1) = vel^k(t)\frac{SA_v}{SB_v + SC_v soil(i,j)} \quad (24)$$

where SA_v, SB_v and SC_v corresponds to amounts greater than zero and the $soil(i,j)$ is the soil between nodes.

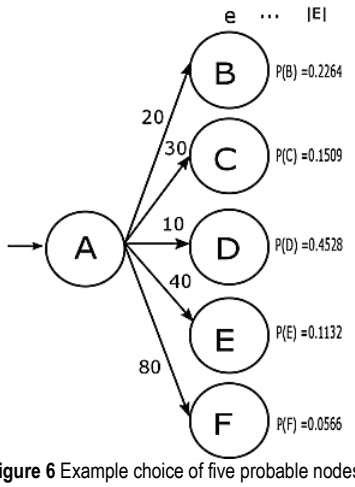

Figure 6 Example choice of five probable nodes

The soil incorporated into the drop between the two nodes is shown with Eq. (25).

$$soil^k = soil^k + \Delta soil(i,j) \quad (25)$$

$$soil(i,j) = (1 - p_0)soil(i,j) - p_n \Delta soil(i,j) \quad (26)$$

where $\Delta soil(i,j)$ is obtained with the following equation:

$$\Delta soil(i,j) = vel^k(t)\frac{SA_s}{SB_s + SC_s time(i,j:vel^k(t+1))} \quad (27)$$

where parameters SA_s, SB_s and SC_s correspond to amounts greater than zero and the time of the drop K to move from one point to another, represented by $time(i,j:vel^k(t+1))$.

The time is obtained with the following equation:

$$time(i,j:vel^k(t)) = \frac{HUD(i,j)}{vel^k(t+1)} \quad (28)$$

where $HUD(i,j)$ represents a heuristics function. It indicates the grade of undesirability of moving from one node to another.

The value of the new soil product from the elimination of soil from the arc between node i and node j is denoted as $soil(i,j)$. It is obtained as follows:

$$soil(i,j) = p_s soil(i,j) - p_s \Delta soil(i,j) \quad (29)$$

where p_S is a value between zero and one.

It is necessary to consider that the update of the soil corresponds to the best drop of each iteration, which is obtained by means of the following equation:

$$soil(i,j) = (1+p_s)soil(i,j) - p_s soil_{IB}^{k} \frac{1}{q(T^{IB})} \quad (30)$$

where p_S is positive amounts that are in a domain between [0,1]. The parameter $soil_{IB}^{k}$ is the accumulated soil of the k drop with the best quality solution of the iteration, and $q(T^{IB})$ is the fitness of the best drop.

$$T^{IB} \begin{cases} T^{IB} & \text{if } q(T^{TB}) > q(T^{IB}) \\ T^{TB}(soil(i,l)) & \text{otherwise} \end{cases} \quad (31)$$

In this way, we have the best solution in the execution of all the iterations.

3.1 Algorithm Description

The algorithm used in this work of intelligent water drops is based on [16], as shown below:

1: input: Problem Instance
2: Assign values to static parameters
4: Assign values to dynamic parameters are initialized
5: Randomly are distributed the IWDs
6: Register the visited node
7: **Repeat**
for IWD = 1 to IWDmax**do**
7.1: IWD travel on the graph choosing nodes
Update Velocity
Compute DeltaSoil
update Edge Soil
update IWD soil
end for
8: Find the iteration best solution
9: Update the soils of the best solution
10: if (iteration best solution > total best solution) then
11: total best solution = iteration best solution
13: else total best solution
14: Increment the iteration
15: **until** (iteration max is complete)
16: The algorithm ends at this point with the best total solution

4 OVERVIEW OF THE INTELLIGENT WATER DROPS FOR SOFTWARE PROJECT SCHEDULING PROBLEM

Here, we present the technique to solve the SPSP for which we must create a Construction Graph.

4.1 Construction Graph

The first thing that must be done is adapt the SPSP so it can be graphically represented. To represent the problem, the heuristic and the value of the soil where the drops moved is used for a direction graph. The human resource has to be incorporated into the project activities, and the allocation to each task is shown in the TPG graph. The purpose of the construction graph is to represent the association of the employee to a task. This is built for each task in TPG. This is divided in a graph with nodes and edges. The graph represents the employees, and their proportion of dedication to the task is called *den*. This is calculated as follows:

$$den = \frac{1}{midn} + 1 \quad (32)$$

where *midn* is the smallest dedication to a project activity. This structure is represented in Fig. 7. The availability of the human resource of the activity must be a multiple of *midn* or 0. Task division consists of the following steps:

- generate a start node and insert in the first column $column_0$.
- generate columns $column_i$. With i = {1, 2,…, $|E|$}. Each column consists in *den*.
- place a final node and put it in the last column $column_{|E|+1}$.
- build edges and incorporate them between columns.

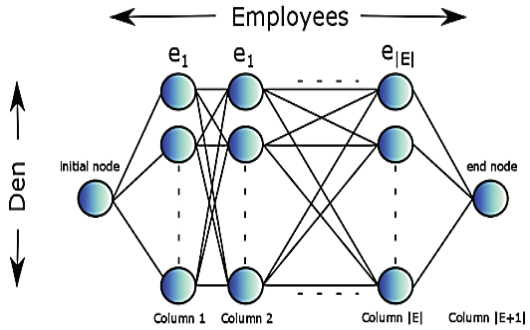

Figure 7 TPG graphic with precedence task

The drops move from the left to the right selecting edges from column 1 to column $|E|$ without return.

The water drops select one node by column.

When a drop completes its travel, the dedication of the employee to a task is completed. To know the dedication of the i employee to a task j, the water drops travel in the construction graph by selecting nodes probabilistically using Eq. (33).

$$p_i^k = \frac{f(soil(i,j))}{\sum_{\forall \notin l_{visited}^{k}} f(soil(i,j))} \quad (33)$$

$$f(soil(i,j)) = \frac{1}{\varepsilon + h(soil(i,j))} \quad (34)$$

where ε is a small value greater than zero that makes it possible to avoid division by zero.

$$h(soil(i,j)) \begin{cases} soil(i,j) & \text{if } \min(soil\ (i,\ l)) \geq 0 \\ soil\ (i,\ l) - \min(soil\ (i,\ l)) & \text{otherwise} \end{cases} \quad (35)$$

where $soil(i,j)$ is the soil between nodes i and j.

6 RESULTS OBTAINED

The experiments were carried out on a computer with Intel Core i7-4510U 2.0 GHz (4M Cache, up to 3.10 GHz) and Windows 7 Professional; construction was based on the Java language.

The experiments focused on comparing the results with another constructive metaheuristic, in this case ACS; however, MMAS-HC was also compared as a reference (Tab. 5).

To verify the operation, each instance was executed on thirty occasions using the parameters of Tab. 4.

Table 4 Values used for the parameters

Name	Description	Value
N_{iwd}	Number of IWD	50
$N_{iwditer}$	Number of iterations	900
ε	To avoid division by zero a small value greater than zero	0,001
sa_v	Static parameter to update speed	1
sb_v	Static parameter to update speed	0,01
sc_v	Static parameter to update speed	1
sa_s	Static parameter to update soil	1
sb_s	Static parameter to update soil	0,01
sc_s	Static parameter to update soil	1
p_n	Positive value	0,9

It is a problem of minimization; therefore, the instance that obtained a better result was the 5 employees being superior to ACS but not better than MMAS-HC.

Table 5 Fitness comparison

Instance with 2-3 skill	MMAS-HC [10]	ACS [4]	IWD
5-10 (employee-task)	3.311	3.558	**3.353**
10-10 (employee-task)	2.617	2.638	2.763
15-10 (employee-task)	1.996	2.083	2.186
20-10 (employee-task)	1.892	1.858	1.906
10-20 (employee-task)	6.211	6.369	6,603

7 CONCLUSION

We have solved the SPSP problem using the IWD metaheuristic, obtaining good results. The results were compared with a few previous works [10, 4].

The contribution of the investigated proposal is to use a new constructive metaheuristic and a slightly studied problem. For the development of the approach, two heuristics to guide the solutions search were considered. The results obtained were not superior to ACS; in general, these were encouraging, but the exception was with instances where five employees were able to obtain better results than ACS. Future research should address the use of an autonomous search with the aim of improving the results; also, characteristics of other metaheuristics can be used that can improve the search for good solutions. So, you should also consider the incorporation of new heuristics with the objective of better searches for solutions, especially for those instances where the number of employees is large and it requires a larger amount of skills. In addition, the behavior of IWD static parameter values other than those reviewed in the literature – which were used in this work – should be studied.

Acknowledgements

Broderick Crawford is supported by Grant CONICYT/FONDECYT/REGULAR/1171243. Ricardo Soto is supported by Grant CONICYT/FONDECYT/ REGULAR/1160455. Gino Astorga is supported by Postgraduate Grant PUCV 2015.

8 REFERENCES

[1] 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. https://doi.org/10.1016/j.eswa.2013.09.018

[2] 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. https://doi.org/10.1016/j.eswa.2010.12.059

[3] Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research, 246*(2), 345-378. https://doi.org/10.1016/j.ejor.2015.04.004

[4] 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. https://doi.org/10.1016/j.cor.2012.05.007

[5] Lujic, R., Saric, T., Simunovic, G. (2008). Application of Genetic Algorithm to the Technological Operations Scheduling Problem. *Metalurgija, 47*(2), 103-107.

[6] Barreto, A., Barros, M. D. O., & Werner, C. M. (2008). Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research*, *35*(10), 3073-3089. https://doi.org/10.1016/j.cor.2007.01.010

[7] Sommerville, I. (2005). *Ingeniería del software*. Séptima edición. Pearson Educación.

[8] Alba, E. & Chicano, J. F. (2007). Software project management with {Gas}. *Information Sciences, 177*(11), 2380-2401. https://doi.org/10.1016/j.ins.2006.12.020

[9] Chang, C. K., 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. https://doi.org/10.1016/j.infsof.2008.03.002

[10] Crawford, B., Soto, R., Johnson, F., Misra, S., Paredes, F., & Olguín, E. (2015). Software project scheduling using the hyper-cube ant colony optimization algorithm. *Tehnicki vjesnik-Technical Gazette, 22*(5), 1171-1178.

[11] Luna, F., González-Álvarez, D. L., Chicano, F., & Vega-Rodríguez, M. A. (2014). The software project scheduling problem: A scalability analysis of multiobjective metaheuristics. *Applied Soft Computing, 15*, 136-148. https://doi.org/10.1016/j.asoc.2013.10.015

[12] Biju, A. C., Victoire, T. A. A., & Mohanasundaram, K., (2015). An improved differential evolution solution for software project scheduling problem. *The Scientific World Journal*, Article ID 232193. https://doi.org/10.1155/2015/232193

[13] Shah-Hosseini, H. An approach to continuous optimization by the intelligent water drops algorithm. // *Procedia-Social and Behavioral Sciences*, *32*, (2012), pp.224-229. https://doi.org/10.1016/j.sbspro.2012.01.033

[14] Hosseini, H. S. (2007). Problem solving by intelligent water drops. *Evolutionary Computation*, 3226-3231.

[15] Alijla, B. O., Wong, L.-P., Lim, C. P., Khader, A. T., & Al-Betar, M. A. (2014). A modified intelligent water drops algorithm and its application to optimization problems. *Expert Systems with Applications, 41*(15), 6555-6569. https://doi.org/10.1016/j.eswa.2014.05.010

[16] Niu, S., Ong, S., & Nee, A. (2013). An improved intelligent water drops algorithm for solving multi-objective job shop scheduling. *Engineering Applications of Artificial Intelligence, 26*(10), 2431-2442. https://doi.org/10.1016/j.engappai.2013.07.011

Contact information:

Broderick CRAWFORD, PhD, Professor
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2950, Valparaíso, Chile
E-mail: broderick.crawford@pucv.cl

Ricardo SOTO, PhD, Professor
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2950, Valparaíso, Chile
E-mail: ricardo.soto@pucv.cl

Gino ASTORGA, PhD student
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2950, Valparaíso, Chile
and
Universidad de Valparaíso
Prat 856, Valparaíso, Chile
E-mail: gino.astorga@uv.cl

Carlos CASTRO, PhD, Professor
Universidad Técnica Federico Santa María
Avenida España 1680, Valparaíso, Chile
E-mail: Carlos.Castro@inf.utfsm.cl

Fernando PAREDES, PhD, Professor
Escuela de Ingeniería Industrial, Universidad Diego Portales,
Manuel Rodríguez Sur 415, Santiago, Chile
E-mail: fernando.paredes@udp.cl

Sanjay MISRA, PhD, Professor
Department of Computer and information Science,
Covenant University, Nigeria
and
Atilim University, 06836 – Incek, Ankara Turkey
E-mail: smisra@atilim.edu.ng

José Miguel RUBIO, PhD student, Professor
Universidad Tecnológica de Chile INACAP
Brown Norte 260, Ñuñoa, Chile
E-mail: jrubiol@inacap.cl