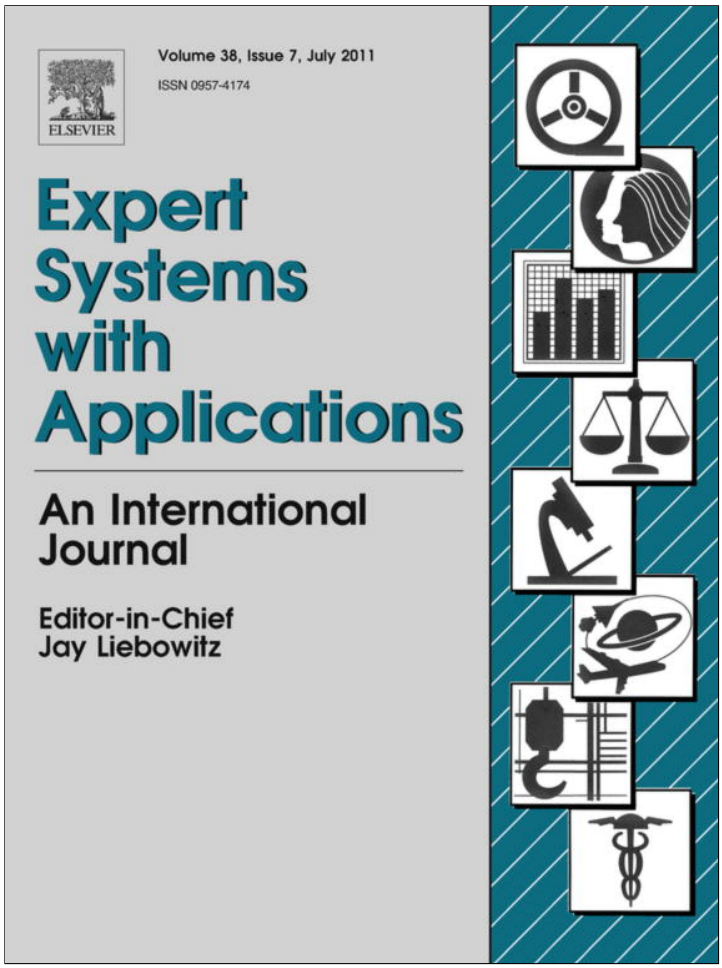

Expert Systems with Applications 38 (2011) 8403–8413

Contents lists available at ScienceDirect

Expert Systems with Applications

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

A knowledge-based evolutionary assistant to software development project scheduling

Virginia Yannibelli *, Analía Amandi

ISISTAN Research Institute, Fac. Cs. Exactas, UNCPBA, Campus Universitario, Paraje Arroyo Seco, Tandil (7000), Buenos Aires, Argentina
CONICET, Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina

ARTICLE INFO

Keywords:
Project scheduling
Software projects
Human resource assignment
Multi-skilled resources
Heterogeneous effectivities
Genetic algorithms

ABSTRACT

The scheduling of software development projects is a central, non-trivial and costly task for software companies. This task is not exempt of erroneous decisions caused by human limitations inherent to project managers. In this paper, we propose a knowledge-based evolutionary approach with the aim of assisting to project managers at the early stage of scheduling software projects. Given a software project to be scheduled, the approach automatically designs feasible schedules for the project, and evaluates each designed schedule according to an optimization objective that is priority for managers at the mentioned stage. Our objective is to assign the most effective set of employees to each project activity. For this reason, the evaluation of designed schedules in our approach is developed based on available knowledge about the competence of the employees involved in each schedule. This knowledge arises from historical information about the participation of the employees in already executed projects. In order to evaluate the performance of our evolutionary approach, we present computational experiments developed over eight different sets of problem instances. The obtained results are promising since this approach has reached an optimal level of effectivity on seven of the eight mentioned sets, and a high level of effectivity on the remaining set.

1. Introduction

A software development project is defined with the aim of creating a software product based on the requirements of a particular client. This kind of projects consists of activities corresponding to the analysis of the client requirements, and to the design, coding, testing, installing, and maintaining of the software application. These activities require human resources with different skills to be developed.

The design of an initial schedule for a software development project is a central, non-trivial and costly task for software companies. This task implies to define feasible start times (i.e., the precedence relations between the activities must not be violated) and feasible human resource assignments (i.e., the resource requirements must be satisfied) for project activities. In addition, to define the mentioned resource assignments, it is necessary to estimate the effectivity of the human resources in relation to different project activities. This is because the development and the results of an activity depend on the effectivity of the resources assigned to it. Then, to estimate the mentioned effectivity, it is indispensable to have knowledge about the effectivity of the available human resources in already executed projects (Heerkens, 2002).

In general, the initial scheduling of a given software project is a task manually developed by project managers. This task requires a considerable amount of time, effort, experience in scheduling, and knowledge about the available human resources that can be considered to project activities. On the other hand, this task is not exempt of erroneous decisions caused by human limitations inherent to the managers (e.g., lack of knowledge about the organization and their resources, decisions based on incorrect assumptions, etc.). Taking into account the requirements and drawbacks of the manually developed scheduling, it is considered valuable assisting to managers at the early stage of designing project schedules. In this sense, the knowledge-based automatic design has the aim of providing initial schedules in efficient way (i.e., the time required by the automatic design is lower than the time required by managers) and in effective way (i.e., erroneous decisions are minimized when considering all the available knowledge arising from information about already executed projects).

Since 30 years ago, many different kinds of algorithms (i.e., exacts, heuristics and metaheuristics) have been proposed in literature to automatically solve project scheduling problems. In this context, different works have considered specificities of human resources (i.e., skills, efficiency, workload capacity, and cost per

* Corresponding author at: ISISTAN Research Institute, Fac. Cs. Exactas, UNCPBA, Campus Universitario, Paraje Arroyo Seco, Tandil (7000), Buenos Aires, Argentina. Tel.: +54 (02293) 439682x28; fax: +54 (02293) 439681x52.

E-mail addresses: vyannibe@exa.unicen.edu.ar (V. Yannibelli), amandi@exa.unicen.edu.ar (A. Amandi).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.01.035

time unit). However, to the best of our knowledge, few works have considered human resources with resource-specific effectivities (Bellenguez & Néron, 2004b; Hanne & Nickel, 2005), a central aspect in software projects, although with a great number of simplifications. In particular, the methods proposed in the said works to estimate the effectivity of the human resources assigned to the activities contain many simplifications. In these sense, the effectivity of a human resource in a particular activity is estimated only considering the effectivity level of the resource in relation to one of the skills required for the activity. Nevertheless, there are others contextual factors that also determine the effectivity of a human resource (i.e., the general characteristics of the activity to which the resource is assigned, the other resources with which the resource in question must work, and the experiences and attributes of the resource in question) (Barrick, Stewart, Neubert, & Mount, 1998; Heerkens, 2002). Therefore, the effectivity of a human resource in a particular activity needs to be estimated in relation to all the mentioned factors.

In this paper, the problem of scheduling a software development project is addressed with the aim of assisting to project managers at the early stage of scheduling software projects. Thus, as part of the problem, we consider an optimization objective priority for managers at the mentioned stage. This objective is to assign the most effective set of human resources to each project activity.

To solve the problem in question, we propose a knowledge-based genetic algorithm. Considering a given software project, this algorithm designs feasible schedules for the project, and evaluates each designed schedule according to the mentioned optimization objective. This evaluation is developed based on available knowledge about the effectivity of the human resources involved in each schedule. The mentioned knowledge arising from historical information about the participation of the resources in already executed projects.

We consider a genetic algorithm because of the following reason. The problem addressed here is a special case of the RCPSP (Resource Constrained Project Scheduling Problem) (Blazewicz, Lenstra, & Rinnooy Kan, 1983) and, therefore, is a *NP-Hard* problem. In this sense, the genetic algorithms have shown to be effective in the resolution of a wide variety of *NP-Hard* problems and, in particular, in the resolution of the RCPSP (Kolisch & Hartmann, 2006).

The remainder of the paper is organized as follows. Section 2 defines the addressed problem. Section 3 describes the genetic approach proposed to solve the problem. In Section 4, the developed computational experiments are presented and their results are analyzed. Section 5 gives a brief review of approaches proposed in literature for project scheduling problems in which the assignment of human resources is considered. Finally, Section 6 points out the conclusions of the present work.

2. Problem description

In this paper, we consider the problem of scheduling a software development project with the aim of assisting to project managers at the early stage of the scheduling projects.

A software development project contains a set A of N activities, $A = \{1, \ldots, N\}$, that has to be scheduled (i.e., the starting time and the human resources of each activity have to be defined). These activities are deduced by the project manager considering the different stages of a software development process (i.e., the analysis of the requirements of the client, the development of the software application, the tests required to be sure that the application answers the client's request, etc.). In addition, the duration, precedence relations and resource requirements of each activity also are defined by the manager.

Thus, the duration of each activity j is known and notated as d_j. Moreover, it is considered that pre-emption of activities is not allowed, that is to say, when an activity starts, it must be developed period by period until it is completed (i.e., the d_j periods of time must be consecutive).

Among some project activities there are precedence relations due to technological requirements. In general, each of the activities consumes products generated by other activities (e.g., a testing activity consumes the product of at least one coding activity). Thus, the precedence relations establish that each activity j cannot start until all its immediate predecessors, given by the set P_j, have completely finished.

The project activities require human resources (employees) skilled in different knowledge areas to be developed. Specifically, to be developed, each activity requires one or several skills, and a given number of employees for each skill. It is considered that a skill is a specialization in a knowledge area (e.g., programmer, analyst of requirements, designer, tester, etc.).

A software company has qualified workforce to develop their projects. This workforce is made up of a number of employees, and each employee masters one or several skills.

Taking into consideration a given project, the set SK represent to the K skills required to develop the project, $SK = \{1, \ldots, K\}$, and the set AR_k represent to the available employees with skill k. Then, the term $r_{j,k}$ represents the number of employees with skill k required for the activity j of the project. The values of the terms $r_{j,k}$ are known for each project activity.

It is considered that an employee cannot take over more than one skill within a given activity. In addition, it is considered that an employee cannot be assigned to more than one activity at the same time.

Based on the previous assumptions, an employee can be assigned to different activities although not at the same time, can take over of different skills required for an activity but not in a simultaneous way, and can integrate different possible sets of employees for each activity. Then, for each available employee, it is possible to define different work contexts.

We consider the work context of an employee r, denoted as $C_{r,j,s,g}$, is composed for four main components. The first component is the activity j to which r is assigned (i.e., the complexity of j, their domain, their general characteristics, etc.). The second component is a skill s required by j and that must be managed for r (i.e., the tasks associated to s). The third component is the set of employees g that has been assigned to j and that includes to r (i.e., r must work in collaboration with the other employees assigned to j). The fourth component refers to the attributes of r (i.e., their skills, their experiences, the labor relations between r and the other employees of g, their knowledge, their studies, etc.).

The four described components are considered as the main factors that determine the effectivity level of an employee (Barrick et al., 1998; Heerkens, 2002; Wysocki, 2003). For this reason, the effectivity of an employee needs to be defined in relation to all components of their work context. Then, for each employee, it is possible to define different effectivity levels in relation to different work contexts.

Given a employee r, in relation to each possible context $C_{r,j,s,g}$ for r, an effectivity level, $e_{rCr,j,s,g}$, is defined describing how well r can handle, within the activity j, the tasks associated to the skill s, considering that r must work in collaboration with the other employees of the set g. The effectivity level is represented as a real value over the range [0, 1].

The problem of scheduling a software development project implies to define feasible start times (i.e., the precedence relations between the activities must not be violated) and feasible human resource assignments (i.e., the human resource requirements must be satisfied) for project activities in such a way that an optimization

objective is reached. In this sense, it is considered an objective priority for software project managers at the early stage of designing project schedules. This objective is to assign the most effective set of employees to each project activity. The mentioned objective is modeled by Eqs. (1) and (2)

$$\max_{\forall s \in S} \left(e(s) = \sum_{i=1}^{N} e_{R(i,s)} \right) \quad (1)$$

$$e_{R(i,s)} = \frac{\sum_{r=1}^{|R(i,s)|} e_{rC_{r,i,s(r,i,s),R(i,s)}}}{|R(i,s)|} \quad (2)$$

Eq. (1) maximizes the effectivity of the sets of employees assigned to the N activities of a given project. In this equation, the set S contains all the feasible schedules for the project in question. Then, $R(i, s)$ is the set of employees assigned to the activity i in the schedule s, and the term $e_{R(i,s)}$ represents the effectivity level corresponding to $R(i, s)$.

In Eq. (2), $e_{R(i,s)}$ is calculated as the mean effectivity level of the employees belonging to $R(i, s)$. It is considered that, in software project activities, the effectivity level of a set of employees depends on the effectivity level of each employee belonging to the set (Barrick et al., 1998; Boon & Sierksma, 2003; Heerkens, 2002; Wysocki, 2003). Moreover, it is considered that the individual levels of effectivity are equally important in determining the effectivity of a set of employees. Based on the mentioned assumptions, the more appropriate option to operationalize $e_{R(i,s)}$ is to calculate the mean value of the individual levels of effectivity corresponding to the employees of the set $R(i, s)$ (Barrick et al., 1998). On the right side of formula (2), the terms(r, i, s) represents to the skill managed for employee r within the activity i in the schedule s.

2.1. Problem example

In this section, a brief example of a software development project scheduling problem is presented. The problem refers to the development of a web site. Then, a feasible and optimal schedule for the project in question is detailed.

The project contains eleven activities deduced from the different stages of a software development process. Table 1 presents the mentioned activities and an estimated duration for each activity. Among some project activities there are precedence relations, these relations are shown by the graph of Fig. 1.

The project activities require employees with different skills to be developed. In this sense, Table 2 details the skills required for each activity, and the number of employees required for each skill.

To develop the complete project, five employees with multiple skills are available. Table 3 details the skills managed for each of the five employees.

Considering the skill requirements of each project activity, and the skills of the available employees, it is possible to define more than one feasible set of employees for the activities 2, 6, 10, 7, 8 and 9. For example, it is possible to define three feasible sets of employees for the activity 6 (i.e., the sets (E2, E4), (E2, E5) and (E4, E5)).

Then, taking into account the optimization objective of the problem, i.e., to maximize the effectivity level of the sets of employees assigned to the activities, it is necessary to define the effectivity level of all the possible sets for each of the previously mentioned activities. Subsequently, to define the effectivity level of the mentioned sets, it is necessary to know the effectivity level of the employees included in the sets. In this sense, Tables 4–6 show the effectivity level of each of employees that can be assigned to the activities 6, 10 and 2, respectively. In each of these tables, for each employee that can be assigned to the activity corresponding to the table, all the possible work contexts inherent to the activity are presented, and an effectivity level in relation to each mentioned context is presented. For example, in Table 4, the first column indicates that the employee E2 can be assigned to the activity 6. Then, the two rows corresponding to E2 present two possible work contexts for E2: E2 can be assigned as programmer to activity 6 working in collaboration with the employee E4 or can be assigned as programmer to activity 6 working in collaboration with the employee E5. Then, the fourth column of the table indicates that, in relation to the first mentioned work context, the effectivity level of E2 is equal to 1 and, in relation to the second mentioned work context, the effectivity level of E2 is equal to 0.9.

Based on the content of Tables 4–6, and considering the way in which the effectivity of a set of employees is calculated (Formula (2)), it is possible to say that the employees E2 and E4 form the most effective set of programmers to develop the activity 6, the employees E2 and E4 also form the most effective set of programmers to develop the activity 10, and the employees E2 and E1 form the most effective set of designers to develop the activity 2.

In addition to the content of Tables 4–6, it is considered that the employee E2 is the most effective programmer to develop the activity 7, the employee E4 is the most effective programmer to develop the activity 8, and the employee E5 is the most effective programmer to develop the activity 9.

Table 1
Activities of the project example.

Number of activity	Activity	Duration (in weeks)
1	Analysis of client's requirements	1
2	Design of the architecture of the web site	2
3	Design of the user interface of the web site	2
4	Design of the functional modules of the web site	2
5	Design of the data base of the web site	2
6	Programming of the architecture	2
7	Programming of the user interface	3
8	Programming of the functional modules	3
9	Programming of the data base	3
10	Integration of architecture/interface/modules/data base	2
11	Testing of the web site	2

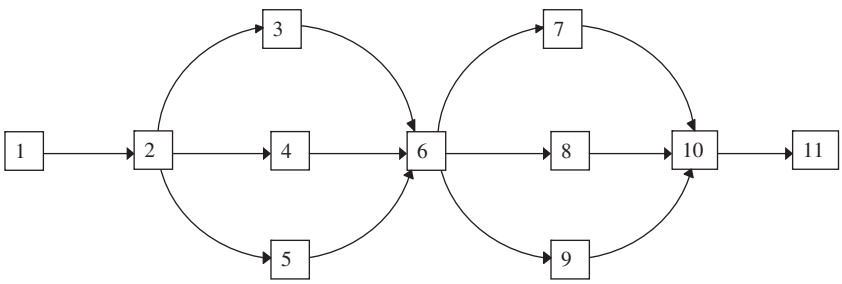

Fig. 1. Precedence graph of the project example.

Table 2
Skills required for each activity of the project example.

Number of activity	Analyst of requirements	Designer of architectures	Designer of user interfaces	Designer of functional modules	Designer of data base	Programmer	Tester
1	1						
2		2					
3			1				
4				1			
5					1		
6						2	
7						1	
8						1	
9						1	
10						2	
11							2

Table 3
Skills managed for each one of the five available employees for the project example.

Employee	Analyst of requirements	Designer of architectures	Designer of user interfaces	Designer of functional modules	Designer of data base	Programmer	Tester
E1		√		√			√
E2		√	√			√	√
E3	√	√			√		
E4						√	
E5						√	

Table 4
Effectivity levels of each employee that can be assigned to activity 6. For each mentioned employee, different effectivity levels are defined in relation to different work contexts.

Employee	Activity	Skill	Set of employees	Effectivity level
E2	6	Programmer	(E2, E4)	1
	6	Programmer	(E2, E5)	0.9
E4	6	Programmer	(E4, E2)	1
	6	Programmer	(E4, E5)	0.7
E5	6	Programmer	(E5, E2)	0.7
	6	Programmer	(E5, E4)	0.5

Table 5
Effectivity levels of each employee that can be assigned to activity 10. For each mentioned employee, different effectivity levels are defined in relation to different work contexts.

Employee	Activity	Skill	Set of employees	Effectivity level
E2	10	Programmer	(E2, E4)	1
	10	Programmer	(E2, E5)	0.9
E4	10	Programmer	(E4, E2)	1
	10	Programmer	(E4, E5)	0.8
E5	10	Programmer	(E5, E2)	0.7
	10	Programmer	(E5, E4)	0.6

Table 6
Effectivity levels of each employee that can be assigned to activity 2. For each mentioned employee, different effectivity levels are defined in relation to different work contexts.

Employee	Activity	Skill	Set of employees	Effectivity level
E1	2	Designer of architectures	(E1, E2)	1
	2	Designer of architectures	(E1, E3)	0.8
E2	2	Designer of architectures	(E2, E1)	1
	2	Designer of architectures	(E2, E3)	0.9
E3	2	Designer of architectures	(E3, E1)	0.8
	2	Designer of architectures	(E3, E2)	0.9

It is considered that the previously presented knowledge about the effectivity of the employees in different contexts arises from available historical information about the mentioned employees.

Fig. 2 shows a feasible and optimal schedule for the previously described project, i.e., this schedule assigns the most effective set of employees to each project activity. The mentioned schedule details the starting time defined for each activity and the employees assigned to each activity.

3. A knowledge-based genetic algorithm

To solve the problem addressed in this paper, we propose a knowledge-based genetic algorithm. Genetic algorithms are adaptive heuristic methods of search and optimization inspired on Darwin's theory of evolution (Darwin, 1859; Eiben & Smith, 2007; Goldberg, 2007). According to these algorithms, an initial population of candidate solutions to a problem evolves towards the optimal solutions based on the natural principles of natural selection, crossover, and mutation.

The general behavior of the algorithm proposed here is described below. This behavior is based on the general structure of a genetic algorithm which was developed by Holland (1975).

Considering a given software project, the algorithm starts the evolution from an initial population of solutions in which each solution codifies a feasible project schedule. Then, each solution of the population is decoded, i.e., the related schedule is built, and evaluated according to the optimization objective of the problem by a fitness function. The mentioned objective is to maximize the effectivity of the sets of employees assigned to project activities. Thus, the said fitness function evaluates each solution based on available knowledge about the effectivity of the employees involved in the solution. Then, a selection process is used to select a number of solutions from the current population according to

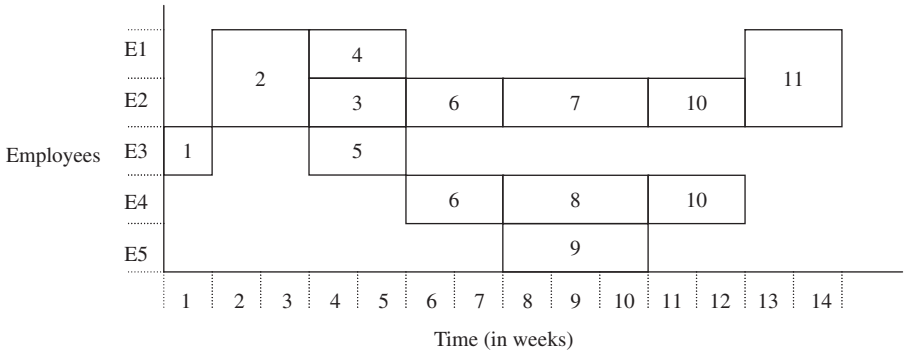

Fig. 2. Optimal schedule for the project example.

some selection strategy. In general, the solutions with the greatest values of fitness will have more chances of being selected. The selected solutions are paired, and a crossover process is applied to each pair of solutions to generate new feasible ones. Then, a mutation process is applied to modify the components of the solutions generated by the crossover. The purpose of using the mutation process is to promote diversity in the current population of solutions. Finally, a deterministic crowding strategy (Goldberg, 2007) is used to determine a new population by selecting between the solutions of the current population and the new solutions generated. The described process is repeated until some stopping criterion is reached.

Details about each of the different components of the proposed algorithm will be presented in the next sections. The main components of the algorithm are the representation of solutions, the generation of the initial population, the fitness function, and the selection, crossover, and mutation processes.

3.1. Representation or encoding of solutions

In the algorithm, each solution in a population represents a particular project schedule. The solutions must be represented or encoded in such a way that the application of different crossover and mutation operators generates new feasible solutions. Therefore, it is necessary to define an appropriate encoding to project schedules.

We propose a representation based on the standard activity list representation (Hartmann, 1998; Kolisch & Hartmann, 1999, 2006). Details about the proposed representation are presented below.

Each solution is represented by two lists having as many positions as activities in the project. Fig. 3 shows the proposed representation for a project with N activities.

The first list is a standard activity list. This list is a feasible precedence list of the activities involved in the project, i.e., each activity j can appear on the list in any position higher than the positions of all its predecessors. Moreover, each activity j in the list contains information about its development, i.e., activity duration and requirement of employees of each skill k.

The second list is an assigned resources list. This list contains information about the employees of every skill k assigned to each activity of the project. Specifically, the position j on this list details the employees of every skill k assigned to the activity j. The detailed information about the employees assigned to each activity is not considered in the standard activity list representation.

3.1.1. Decoding of solutions

In order to build the schedule related to the representation, the serial method proposed by Kelley is considered (Kelley, 1963; Kolisch & Hartmann, 1999). The previously mentioned method schedules the activities, one by one, in the order given by the

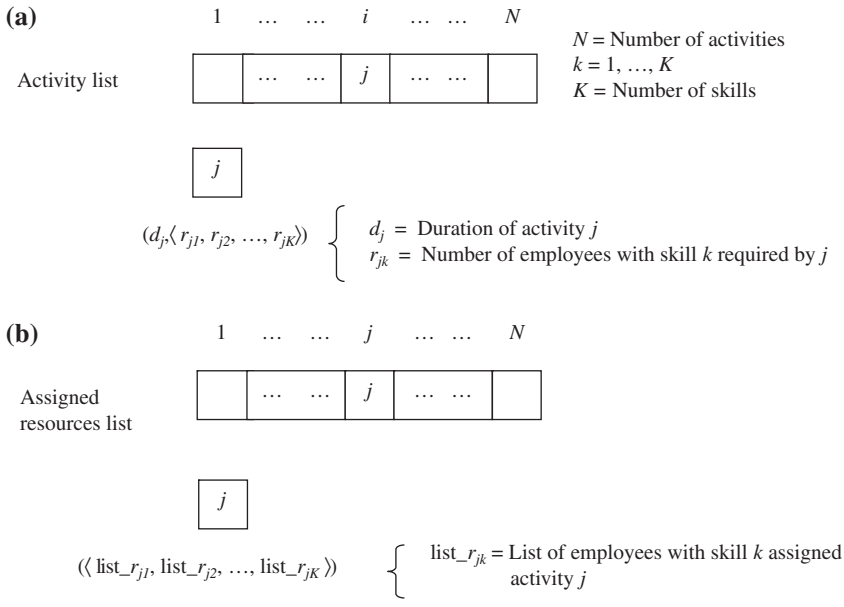

Fig. 3. Representation proposed for project schedules.

activity list (forward scheduling). In Fig. 3a, the activity j represents the ith activity that must be inserted in the schedule. When activity j is chosen to be inserted in the schedule, all its predecessors, which appear in some position $[1, (i-1)]$ in the activity list, will have already been scheduled. Then, the activity j obtains the earliest feasible starting time, i.e., the activity can start its development after all its predecessors have been completed, and when all the employees assigned to the activity are available (in Fig. 3b, the position j details the employees, of every skill k, assigned to the activity j). Thus, the related schedule is always a feasible schedule.

It is important to note that when the serial method is applied, one and only one schedule can be deduced from a given encoded solution, but different encoded solutions could be transformed in the same schedule. In addition, it is worth pointing out that the parallel method cannot be directly applied to an activity list in order to transform the solution into its corresponding schedule (Kolisch & Hartmann, 1999).

Fig. 4 shows one feasible encoded solution to the project example described in Section 2.1, and Fig. 5 presents the schedule built from this solution by the described serial method. The schedule indicates the specific employees assigned to each activity and the starting time defined for each activity.

3.2. Initial population

The initial population contains a specific number of feasible solutions to the project to be scheduled, i.e., each solution of this population represents a feasible schedule to the project in question.

Each solution of the initial population is randomly generated. It has been decided to use a random approach to generate these solutions because a random approach guarantees a good level of genetic diversity in the initial population (Goldberg, 2007). By means of such diversity, it is attempted to prevent the premature convergence of the algorithm.

In order to obtain each solution, we use a process which is divided into two stages. The first stage determines the positions of the project activities on the activity list. The second stage defines the employees that are assigned to each activity based on the human resource requirements of such activities, i.e., the second stage defines the assigned resources list.

The first stage is developed as follows. It begins with an empty activity list. The next activity for the list is randomly taken from the activities not yet inserted on the list while all its predecessors have already been inserted in it. Thus, on the list, each activity appears in a random position following all its predecessors.

The second stage assigns employees to each project activity. Each activity j requires r_{jk} employees with skill k. For each activity j, r_{jk} employees with skill k are randomly selected from the group of available employees with skill k, AR_k, and the selected employees are assigned to the activity j (i.e., are assigned to the position j of the assigned resources list).

The described random process guarantees that the precedence relationships between the activities and the human resource requirements of each of the activities are respected during the construction of each solution.

3.3. Fitness function

The fitness function evaluates the level of fitness of each solution in relation to the predefined optimization objective. In this case, the objective is to maximize the effectivity level of the sets of employees assigned to the project activities.

Given a solution for a project p, the fitness function decodes the schedule s related to the solution by using the method described in Section 3.1.1. Then, the function calculates the value of the term $e(s)$ corresponding to s (Formulas (1) and (2)). Thus, this function gives a real value over $[0, \ldots, N]$ to s. This value represents the effectivity level of the sets of employees assigned to project activities by the schedule s.

To calculate the term $e(s)$, the function needs know the value of the terms $e_{rCr,j,s,g}$ inherent to s (Formula (2)). We consider that each project p has an associated knowledge base, and that this base contains the value corresponding to each of the mentioned terms. Then, the base is queried by the fitness function to obtain the value of the said terms.

Moreover, we consider the value of each term $e_{rCr,j,s,g}$ arises mainly from historical information about the participation of the employee r in projects already executed. The estimation of the effectivity of a set of employees, in relation to a particular activity, based on knowledge about the employees included in the set is a usual process used by project managers (Heerkens, 2002).

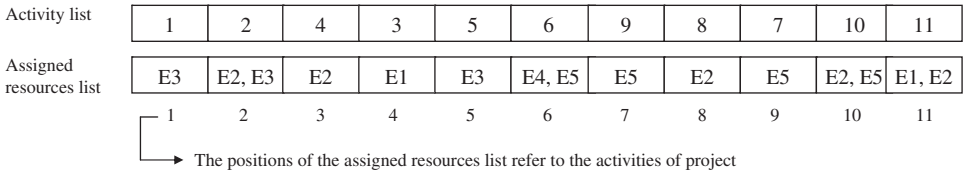

Fig. 4. Feasible solution for the project example.

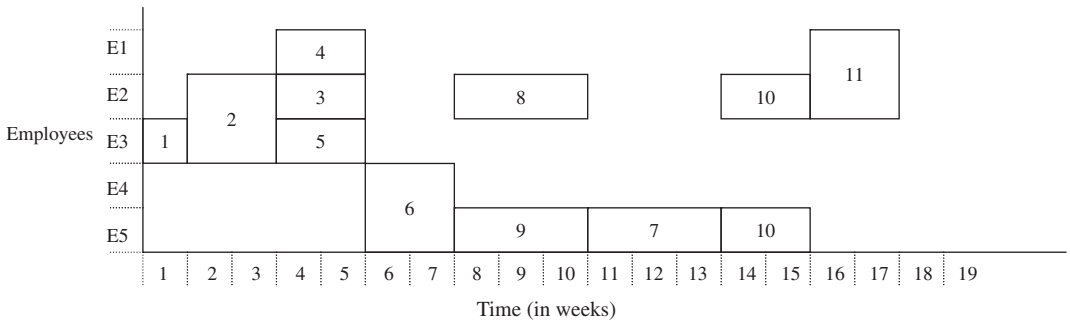

Fig. 5. Schedule decoded from the solution shown in Fig. 4.

3.4. Selection

The selection process selects the best solutions in the current population to conform pairs of solutions (parent solutions) which will be used to generate new solutions (offspring solutions) by the crossover and mutation processes.

This process considers the fitness values of the solutions at moment of selecting a solution and, moreover, is generally biased by a random factor. Thus, the solutions with the highest fitness values will have more chances of being selected.

There are several schemes to develop the selection process. One of the most applied schemes in literature to be used in this paper is the 2-tournament selection (Eiben & Smith, 2007; Goldberg, 2007).

In the 2-tournament selection, two solutions are randomly chosen from the current population and compete for survival. The best of them (i.e., the solution with the best fitness value) is selected and considered as the first member of the pair. The worst of them is returned to the population. This operation is repeated to obtain the second member of the pair.

The process previously described is repeated until a number $M/2$ of pairs is obtained, considering to M as the population size.

3.5. Crossover

The selection process determines the pairs of solutions in the current population that should be recombined, and each pair undergoes the crossover operation with a probability of P_c. The crossover developed in a pair of solutions (parent solutions) generates two new solutions (offspring solutions).

The crossover operator is one of the most important genetic operators because it preserves and combines the best characteristics of the parent solutions so as to define new best solutions (Goldberg, 2007).

The crossover operators are directly applied to pairs of encoded solutions. Thus, the crossover must be designed based on the representation defined for the solutions. In this case, the proposed representation consists of two lists: an activity list and an assigned resources list. Therefore, we propose a crossover operator that contains a feasible crossover operation for each of the mentioned lists.

3.5.1. Crossover operation for activity lists

This operation is applied to the activity lists of two parent solutions that must be recombined. The result of this operation consists of two new activity lists. The first new list is assigned to the first offspring and the second new list is assigned to the second offspring.

The behavior of this operation is described below. Firstly, the operation defines a random crossover-point k, considering k between 1 and N. Then, the first k activities in the list of parent 1 are positioned in the first k positions of the list of offspring 1, in the same order. The remaining activities are positioned in the empty positions of the list of offspring 1 considering the relative order of said activities in the list of parent 2. In this way, the activity list generated for offspring 1 is a precedence feasible list.

The generation of the activity list for offspring 2 is similar to the generation of the list for offspring 1. However, the list of offspring 2 inherits the first k activities directly from the list of parent 2, and the order of the remaining activities from the list of parent 1.

Fig. 6 shows an example of described crossover operation for activity lists. In this example, the operation is applied to the activity lists corresponding to two parent solutions defined for the project example presented in Section 2.1.

It is necessary to mention that the described operation correspond to the one-point crossover developed by Hartmann (1998).

3.5.2. Crossover operation for assigned resources lists

This operation is applied to the assigned resources lists of two parent solutions that must be recombined. The result of this operation consists of two new assigned resources lists. The first new list is assigned to the first offspring and the second new list is assigned to the second offspring.

The behavior of this operation is described below. For each activity j of the project, the operation chooses a random value r over [0, 1], if $r \leqslant 0.5$, the resource assignment for j in the list of offspring 1 (in the list of offspring 2) is inherited from parent 1 (from parent 2). On the other hand, if $r > 0.5$, the resource assignment for j in the list of offspring 1 (in the list of offspring 2) is inherited from parent 2 (from parent 1). This operation always leads to new feasible lists.

Fig. 7 shows an example of the described crossover operation for assigned resources lists. In this example, the operation is applied to the assigned resources lists corresponding to two parent solutions defined for the project example presented in Section 2.1.

3.6. Mutation

This process randomly alters one or more characteristics of some solutions obtained after the crossover process. The mutation operator has the aim of introducing genetic diversity in the population (Goldberg, 2007).

The mutation operators are directly applied to encoded solutions. Thus, the mutation must be designed based on the representation defined for the solutions. Therefore, in this section, we propose a mutation operator that contains a feasible mutation operation for activity lists and a feasible mutation operation for assigned resources lists.

3.6.1. Mutation operation for activity lists

This operation is applied to the activity list of a solution that must be mutated. The result of this operation consists of a new activity list for the said solution.

The behavior of this operation is described below. For each activity on the activity list, a new position is randomly chosen. In particular, the new position must be higher than the position

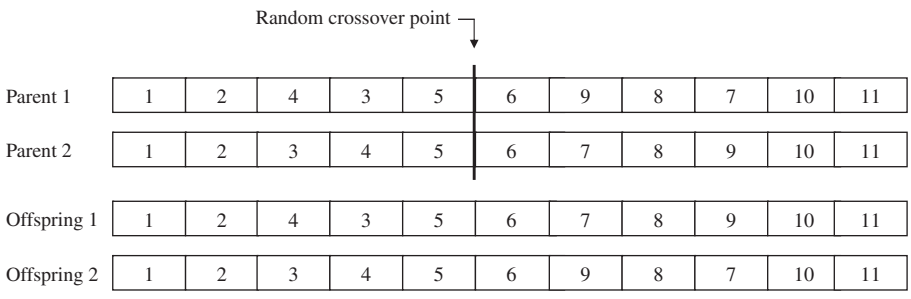

Fig. 6. Example of the crossover operation considered for activity lists.

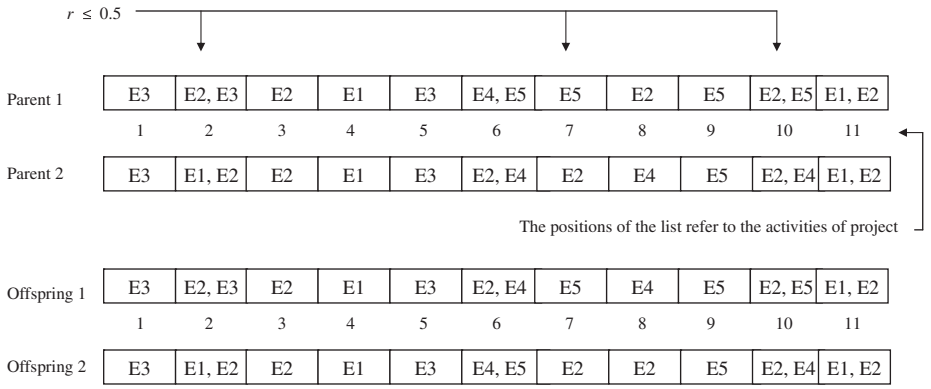

Fig. 7. Example of the crossover operation considered for assigned resources lists.

corresponding to any of the activity's predecessors, and lower than the position corresponding to any of the activity's successors. The activity is inserted in the new position with a probability of P_m. By this procedure, only precedence feasible lists are generated.

Fig. 8 shows an example of this mutation operation. In this example, the operation is applied to the activity list corresponding to a solution defined for the project example presented in Section 2.1.

This mutation operation is an adaptation of the procedure proposed by Boctor (1996) in his simulated annealing algorithm to generate a neighbor.

3.6.2. Mutation operation for assigned resources lists

This operation is applied to the assigned resources list of a given solution. The result of this operation consists of a new assigned resources list for the said solution.

The behavior of this operation is described below. For each activity of the project, the operator defines a new resource assignment with a probability of P_m. If a new resource assignment must be defined for a given activity, the second stage of the mechanism used to create the random solutions of the initial population is considered (Section 3.2). This operation always leads to new feasible lists.

Fig. 9 shows an example of this mutation operation. In this example, the operation is applied to the assigned resources list corresponding to a solution defined for the project example presented in Section 2.1.

4. Computational experiments

In this section, we describe the computational experiments developed to evaluate the performance of the proposed genetic algorithm. Then, the results obtained by the experiments are presented and analyzed.

We have generated different instances of project scheduling problems by ProGen (Kolisch & Sprecher, 1997; Kolisch, Sprecher,

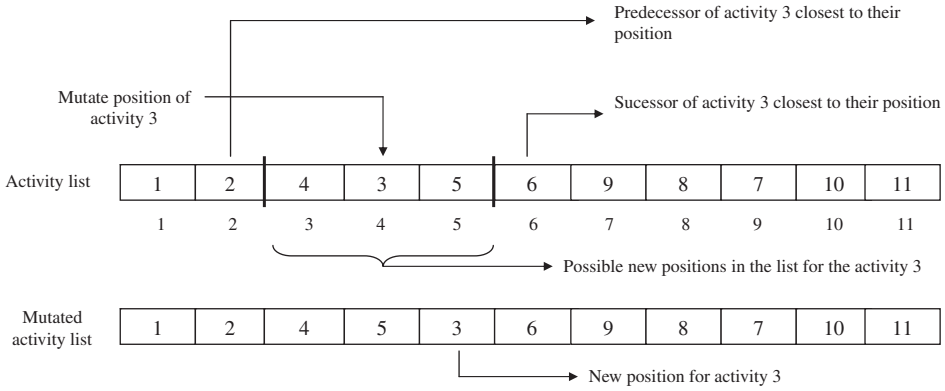

Fig. 8. Example of the mutation operation considered for activity lists.

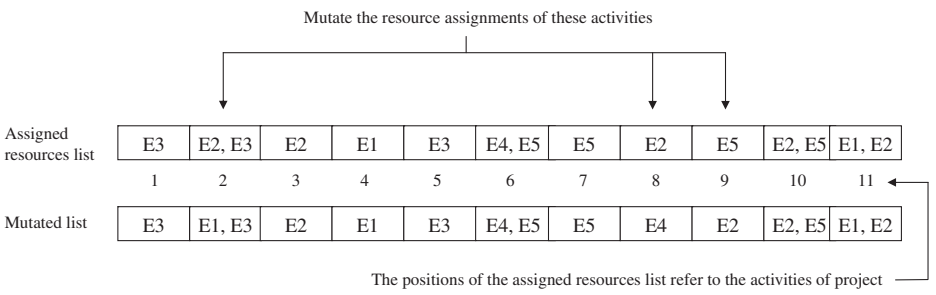

Fig. 9. Example of the mutation operation considered for assigned resources lists.

& Drexl, 1995), and then we have extended the content of the generated instances with the aim of adapting these instances to the characteristics of the proposed algorithm. Table 7 shows the characteristics of each generated set of instances. The first column specifies the name of each set, the second column details the number of activities of the instances of each set, the third column details the number of possible set of employees for activities corresponding to the instances of each set, and the fourth column details the number of instances in each set.

Each instance generated by ProGen specifies information about a precise number of activities. For each activity, the instance details the duration, the precedence relations, and the number of required resources for each possible type of resources. Moreover, each instance specifies information about the available resources. Specifically, each instance defines four types of available resources, and a number of available resources for every type. We have considered that each type of resources is a different skill, and that the number of available resources for each type corresponds to the number of available employees for each skill. Moreover, we have considered that each employee only manages one of the four possible skills.

For each instance generated by ProGen, we have created a knowledge base. As was mentioned in Section 3.3, the knowledge base associated to a given instance is queried by the fitness function of the algorithm to evaluate any solution designed for the instance. Considering the mentioned assumption, for each generated instance, we have created a base that contains all the terms $e_{rCr,j,s,g}$ inherent to each employee r of the instance – these terms are defined considering each of the possible work contexts for r in the instance – and a value over [0, 1] for each of the mentioned terms.

Then, for each extended instance, we have defined an optimal solution with a fitness value equal to N, considering that N is the maximum fitness value for the solutions of an instance with N activities. Specifically, for each extended instance, we have designed a feasible solution s. The solution s was designed by the mechanism used to create the solutions of the initial population of the genetic algorithm (Section 3.2). Then, considering the objective of assigning a fitness value N to s and considering the way in which the fitness value of a given solution is calculated (Section 3.3), we have defined all existing terms $e_{rCr,j,s,g}$ in the solution s, we have assumed a value equal to 1 for each of the mentioned terms, and we have added the mentioned terms with the assumed values to the knowledge base of the instance. Thus, we have defined a feasible solution s with a fitness value equal to N for the instance in question. Considering that all possible solutions for the mentioned instance have a value lower than or equal to N, the solution s can be considered as an optimal solution for the instance.

The proposed genetic algorithm has been tested 20 times on each of the extended instances. We denote *Aval* to the average value of the 20 solutions obtained for each instance. Table 8 gives the parameter values used for these experiments. The parameters have been fixed thanks to preliminary experiments that have shown that these values lead to the best and the most stable results.

Table 8
Values of parameters of the genetic algorithm.

Parameter	Value
Crossover probability P_c	0.8
Mutation probability P_m	0.05
Population size	50
Number of generations	400

We have considered two measures to evaluate the performance of the algorithm in relation to each of the instance sets. The first measure is the average percentage of deviation from the optimal solution (*Av. Dev.* (%)). For each set of instances, we compute the average percentage of deviation of the average value (*Aval*) of the 20 solutions generated from the value of the optimal solution of each instance, taken over the set of instances. The second measure is the number of instances where the value of the optimal solution is achieved (*Optimal* (%)). For each set of instances, we compute the percentage of instances where the value of the optimal solution is achieved at least once among the 20 solutions generated.

Table 9 reports the results obtained by the experiments. The second column of the table reports the *Av. Dev.* (%) for each instance set, and the third column reports the *Optimal* (%) for each instance set.

The *Av. Dev.* (%) obtained by the genetic algorithm for c20_5, c20_10, c30_5, c30_10, c40_5, c40_10 and c50_5 is 0%. These results indicate that the algorithm has found optimal solutions for each instance in each mentioned set.

The *Optimal* (%) obtained by the genetic algorithm for c20_5, c20_10, c30_5, c30_10, c40_5, c40_10 and c50_5 is 100%. Considering that the algorithm was run 20 times on each instance, these results indicate that the algorithm has found an optimal solution in at least once of the 20 runs carried out on each instance pertaining to the mentioned sets. If these results are observer together with the values of *Av. Dev.* (%) obtained for the said sets, it is possible to say that the algorithm has found an optimal solution in each of the 20 runs carried out on each instance pertaining to the mentioned sets.

The *Av. Dev.* (%) obtained by the genetic algorithm for c50_10 is 0.4%. Considering that each instance of c50_10 has a known optimal solution with a fitness value equal to 50, an average deviation equal to 0.4% indicates that the average value of the solutions obtained by the algorithm is 49.8. An average value equal to 49.8 is very close to the value of the known optimal solutions. Therefore, we consider that a deviation of 0.4% is a very low deviation value. Thus, it is possible to say that the algorithm has obtained high quality solutions for the instances of c50_10.

The *Optimal* (%) obtained by the genetic algorithm for c50_10 is 100%. This result indicates that the algorithm has found an optimal solution in at least once of the 20 runs carried out on each instance pertaining to the mentioned set.

Based on the values of *Av. Dev.* (%) and *Optimal* (%) obtained by the algorithm for each of the eight sets, it is possible to establish

Table 7
Characteristics of instance sets.

Instance set	Number of activities to plan	Number of sets of employees for activity	Number of instances
c20_5	20	1–5	30
c20_10	20	1–10	30
c30_5	30	1–5	30
c30_10	30	1–10	30
c40_5	40	1–5	30
c40_10	40	1–10	30
c50_5	50	1–5	30
c50_10	50	1–10	30

Table 9
Results obtained by the computational experiments.

Instance set	Av. Dev. (%)	Optimal (%)
c20_5	0	100
c20_10	0	100
c30_5	0	100
c30_10	0	100
c40_5	0	100
c40_10	0	100
c50_5	0	100
c50_10	0.4	100

that the algorithm has reached an optimal level of effectivity on the sets c20_5, c20_10, c30_5, c30_10, c40_5, c40_10 and c50_5, and that the algorithm has reached a high level of effectivity on the set c50_10.

5. Related works

In literature, different works have considered specificities of human resources (i.e., multiple skills, heterogeneous efficiencies, workload capacity and cost per time unit) in the context of project scheduling problems. However, to the best of our knowledge, few works have considered human resources with resource-specific effectivities (Bellenguez & Néron, 2004b; Hanne & Nickel, 2005), a central aspect in software projects scheduling, although with a great number of simplifications.

Bellenguez (2008), Bellenguez and Néron (2004a, 2007), Néron (2002) and Néron, Bellenguez, and Heurtebise (2006) address the Multi-Skill Project Scheduling Problem (MSPSP). In this problem, to be developed, each project activity requires specific skills and a given number of human resources (employees) for each required skill. Each available employee masters one or several skills, and all the employees that manage a given skill have the same effectivity level in relation to the mentioned skill (homogeneous and static effectivities). Bellenguez and Néron (2004b) consider the MSPSP with hierarchical levels of skills. In this case, given a skill, for each employee that manages the skill, an effectivity level is defined in relation to the mentioned skill (heterogeneous effectivities in relation to the skills). Then, each project activity requires one or several skills, a minimum effectivity level for each skill, and a number of resources for each pair skill-level.

Li and Womer (2009) address the MSPSP with the aim of minimizing the total staffing cost, and consider a specific workload capacity (i.e., weeks in a project) and a specific salary for each employee. In Drezet and Billaut (2008) the objective in the MSPSP is to minimize the maximal lateness of the project, and consider a specific workload capacity (i.e., duration of work per day, etc.) for each employee. In both works, homogeneous effectivities in relation to each skill are considered.

In the previously mentioned works, it is considered that all sets of employees that can be assigned to a particular activity have the same effectivity on the development of the activity. Specifically, with respect to the effectivity, the said sets are merely treated as unary resources with homogeneous effectivities.

Hanne and Nickel (2005) address the MSPSP with three different optimization objectives (i.e., time, quality and cost). In this work, most activities require only one employee with a particular skill, and each employee manages different skills. In this case, heterogeneous effectivities in relation to each skill are considered. Then, the effectivity of an employee in a given activity is defined only considering the effectivity level of the employee in relation to the skill required for the activity.

It is necessary to mention that MSPSP can be seen as a particular case of the MM-RCPCP (Multi-Mode Resource Constrained Project Scheduling Problem). However, in the MSPSP the number of modes for each activity can be exponential with the number of employees having at least one of the required skills, which is too large for a full enumeration (Bellenguez & Néron, 2004b).

Valls, Gomez-Cabrero, Pérez, and Quintanilla (2007), Valls, Pérez, and Quintanilla (2009), Aickelin, Burke, and Li (2009) and Focacci, Laborie, and Nuijten (2000) address the Skilled Workforce Project Scheduling Problem (SWPSP) considering different optimization objectives. In this problem, each project activity requires just one worker with a particular skill to be developed, and each worker has different skills. In Valls et al. (2007, 2009), given a skill, for each resource that manage the skill, an efficiency level is defined (heterogeneous efficiencies in relation to each skill). In Aickelin et al. (2009) and Focacci et al. (2000), homogeneous efficiencies in relation to each skill are considered. In the four mentioned works, homogeneous effectivities in relation to each skill are considered. The SWPSP can be seen as a case of a MM-RCPSP. However, the SWPSP consider other optimization objectives and consider the worker timetables (Valls et al., 2007).

Heimerl and Kolisch (2010) and Gutjahr, Katzensteiner, Reiter, Stummer, and Denk (2008) address the problem of scheduling multiple projects taking into account different optimization objectives. In Heimerl and Kolisch (2010), human resources with homogeneous effectivities and heterogeneous efficiencies in relation to each skill are considered. Moreover, a specific workload capacity and cost per time unit for each resource is considered. Gutjahr et al. (2008) consider human resources with heterogeneous effectivities and efficiencies in relation to each skill.

6. Conclusions

We have addressed the problem of scheduling a software development project with the aim of assisting to project managers at the early stage of designing project schedules. Consequently, as part of the problem, we have considered an optimization objective priority for managers at the mentioned stage. This objective is to assign the most effective set of employees to each project activity. In relation to this objective, we have considered that the effectivity of a set of employees depends on the effectivity of each one of the employees belonging to the set. On the other hand, we have established that the effectivity level of an employee is determined by their work context. Therefore, for each employee, it is possible to define different effectivity levels in relation to different work contexts. To the best of our knowledge, the work context influence on effectivity of the employees has not been considered in previous related works.

To solve the problem in question, we have proposed a knowledge-based genetic algorithm. Considering a given software project, this algorithm designs feasible schedules for the project, and evaluates each designed schedule according to the previously mentioned optimization objective. This evaluation is developed based on available knowledge about the effectivity of employees involved in each schedule. Specifically, for each designed schedule, the algorithm estimates the effectivity level of the sets of employees assigned to the project activities. The effectivity of a set assigned to a particular activity is estimated based on the effectivity level of the employees included in the set. Then, the effectivity level of an employee, in relation to a given work context, is defined based on available knowledge arising from historical information about the participation of the employee in already executed projects.

Different experiments have been carried out to evaluate the performance of the algorithm proposed. The experiments consisted in the solving of instances corresponding to eight different sets: c20_5, c20_10, c30_5, c30_10, c40_5, c40_10, c50_5 and c50_10. The instances belonging to different sets have different complexity, i.e., have a different search space. Thus, the utilization of the mentioned sets has allowed evaluating the genetic algorithm performance over instances that have a different complexity.

In the experiments, we have measured the average percentage deviation from the optimal solution (*Av. Dev.* (%)) for each instance set, and the percentage of problems for which an optimal solution (*Optimal* (%)) was found for each instance set.

Based on the values of *Av. Dev.* (%) and *Optimal* (%) obtained by the algorithm for each of the eight sets, it has been possible to establish that the algorithm has reached an optimal level of

effectivity on the sets c20_5, c20_10, c30_5, c30_10, c40_5, c40_10 and c50_5, and that the algorithm has reached a high level of effectivity on the set c50_10. Therefore, we conclude that the results reached by the proposed algorithm are actually promising.

References

Aickelin, U., Burke, E., & Li, J. (2009). An evolutionary squeaky wheel optimization approach to personnel scheduling. *IEEE Transactions on Evolutionary Computation, 13*(2).

Barrick, M. R., Stewart, G. L., Neubert, M. J., & Mount, M. K. (1998). Relating member ability and personality to work-team processes and team effectiveness. *Journal of Applied Psychology, 83*, 377–391.

Bellenguez, O. (2008). A reactive approach for the multi-skill project scheduling problem. In *PATAT 2008 proceedings of the 7th international conference on the practice and theory of automated timetabling*.

Bellenguez, O., & Néron, E. (2004a). Methods for the multi-skill project scheduling problem. In *9th International workshop on project management and scheduling (PMS'2004), Nancy* (pp. 66–69).

Bellenguez, O., & Néron, E. (2004b). *Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills*. In *PATAT 2004. Lecture notes in computer science* (3616, pp. 229–243). Springer.

Bellenguez, O., & Néron, E. (2007). A branch-and-bound method for solving multi-skill project scheduling problem. *RAIRO – Operations Research, 41*(2), 155–170.

Blazewicz, J., Lenstra, J., & Rinnooy Kan, A. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics, 5*, 11–24.

Boctor, F. F. (1996). An adaptation of the simulated annealing algorithm for solving resource constrained project scheduling problems. *International Journal of Production Research, 34*, 2335–2351.

Boon, B. H., & Sierksma, G. (2003). Team formation: Matching quality supply and quality demand. *European Journal of Operational Research, 48*, 277–292.

Darwin, C. (1859). *The Origin of Species*. London: John Murray.

Drezet, L. E., & Billaut, J. C. (2008). A project scheduling problem with labour constraints and time-dependent activities requirements. *International Journal of Production Economics, 112*, 217–225.

Eiben, A. E., & Smith, J. E. (2007). *Introduction to evolutionary computing* (2nd ed.). 978-3-540-40184-1. Springer.

Focacci, F., Laborie, P., & Nuijten, W. (2000). Solving scheduling problems with setup times and alternative resources. In *Proceedings of the fifth international conference on artificial intelligence planning and scheduling*.

Goldberg, D. E. (2007). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Publishing Company, Inc..

Gutjahr, W. J., Katzensteiner, S., Reiter, P., Stummer, Ch., & Denk, M. (2008). Competence-driven project portfolio selection, scheduling and staff assignment. *Central European Journal of Operations Research, 16*(3), 281–306.

Hanne, T., & Nickel, S. (2005). A multiobjective evolutionary algorithm for scheduling and inspection planning in software development projects. *European Journal of Operational Research, 167*, 663–678.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics, 45*, 733–750.

Heerkens, G. R. (2002). *Project management*. Ed. McGraw-Hill.

Heimerl, C., & Kolisch, R. (2010). *Scheduling and staffing multiple projects with a multi-skilled workforce. OR spectrum* (Vol. 32, pp. 343–368). Springer.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, Michigan: University of Michigan Press.

Kelley, J. E. (1963). The critical-path method: Resources planning and scheduling. In J. F. Muth & G. L. Thompson (Eds.), *Industrial scheduling* (pp. 347–365). Prentice-Hall.

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz (Ed.), *Project scheduling: Recent models, algorithms and applications* (pp. 147–178). Kluwer Academic.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research, 174*, 23–37.

Kolisch, R., & Sprecher, A. (1997). PSPLIB – A project scheduling library. *European Journal of Operational Research, 96*, 205–216.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science, 41*, 1693–1703.

Li, H., & Womer, K. (2009). Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling, 12*, 281–298.

Néron, E. (2002). Lower bounds for the multi-skill project scheduling problem. In *Eighth international workshop on project management and scheduling, Valencia, Spain*.

Néron, E., Bellenguez, O., & Heurtebise, M. (2006). Decomposition method for solving multi-skill project scheduling problem. In *Proceedings of the tenth international workshop on project management and scheduling, Poznan*.

Valls, V., Gomez-Cabrero, D., Pérez, M. A., & Quintanilla, S. (2007). Project scheduling optimization in service centre management. In *Tijdschrift voor Economie en Management* (pp. 341–366). Belgium: Katholieke universiteit te Leuven. 3.

Valls, V., Pérez, A., & Quintanilla, S. (2009). Skilled workforce scheduling in service centers. *European Journal of Operational Research, 193*(3), 791–804.

Wysocki, R. K. (2003). *Effective project management* (3rd ed.). 0-471-43221-0. Wiley Publishing.