Available online at www.sciencedirect.com
ScienceDirect
International Journal of Project Management 28 (2010) 619–628
International Journal of Project Management
www.elsevier.com/locate/ijproman

Project scheduling with limited resources using a genetic algorithm

Jairo R. Montoya-Torres [a,*], Edgar Gutierrez-Franco [b,c], Carolina Pirachicán-Mayorga [a]

[a] Escuela Internacional de Ciencias Económicas y Administrativas, Universidad de La Sabana, Km 7 autopista norte de Bogotá D.C., Chía, Cundinamarca, Colombia
[b] Facultad de Ingeniería, Universidad de La Sabana, Km 7 autopista norte de Bogotá D.C., Chía, Cundinamarca, Colombia
[c] Center for Transportation and Logistics, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307, USA

Received 10 February 2009; received in revised form 8 October 2009; accepted 15 October 2009

Abstract

This paper presents a genetic algorithm for the Resource-Constrained Project Scheduling Problem (RCPSP). In comparison with previous genetic algorithms proposed in literature for this problem, this paper proposes an alternative representation of the chromosomes using a multi-array object-oriented model in order to take advantage of programming features in most common languages for the design of decision support systems. The approach was tested on sets of standard problems taken from the literature and freely available on the Internet (PSPLIB). Computational results validate the effectiveness of the proposed algorithm and show that our procedure equals most of previous results with less computational time.

Keywords: Project scheduling; Genetic algorithm; Object-oriented model; Experimental analysis

1. Introduction

In the late 1950s the development of PERT (Project Evaluation and Review Technique) and CPM (Critical Path Method) techniques allowed projects to be portrayed by network diagrams where jobs or activities are represented by arcs, events are represented by nodes, and the inter-relations between the jobs or activities are defined by the network structure. However, project scheduling with these techniques deals only with the time aspect without consideration of resource restrictions and/or cash flows. Yet, in many real-life situations, delays in the execution time of certain activities occur when resources required by these activities are not available in sufficient quantities during the time interval when they are scheduled to take place. This particular problem is known as the "Resource-Constrained Project Scheduling Problem" (RCPSP) in the literature.

Formally, the RCPSP is described as follows. A project consist of a set $V = \{1, \ldots, n\}$ of n activities. Activities 1 and n are dummy activities respectively representing the start and end of the project. The activities are interrelated by two kinds of constraints:

(a) Precedence constraints force each activity j to be scheduled after all predecessor activities $Pred_j$ are completed.
(b) Activities require resources with limited capacities. Concerning the second constraint, while being processed, activity j requires r_{jk} units of resource type $k \in K$ during every time instant of its non-preemptable duration d_j. Resource type k has a limited capacity of R_k at any point in time.

For the project scheduling problem considered in this paper, without lost of generality, the parameters d_j, r_{jk} and R_k are assumed to be integer, non-negative, deterministic and known at the initial time of scheduling. For the project's start and end activities, we have $d_1 = d_n = 0$ and $r_{1k} = r_{nk} = 0$, for all $k \in K$. An instance of the RCPSP is

* Corresponding author.
E-mail addresses: jairo.montoya@unisabana.edu.co, jrmontoy@yahoo.com (J.R. Montoya-Torres).

0263-7863/$36.00 © 2009 Elsevier Ltd and IPMA. All rights reserved.
doi:10.1016/j.ijproman.2009.10.003

solved by finding starting (or ending) times of each activity satisfying both precedence and resource constraints, such that the total duration of the project (i.e., the makespan), noted as C_{max}, is minimized.

The literature about resolution approaches for RCPSP is very extensive (see Section 2), including exact methods, heuristic and meta-heuristic resolution procedures. It has been shown by Blazewicz et al. (1983) that the RCPSP, as a generalization of the classical job shop scheduling problem, belongs to the class of NP-hard optimization problems, therefore justifying the use of heuristics when solving large problem instances.

The aim of this paper is to propose a genetic algorithm based on an object-oriented model representation to solve the project scheduling problem with limited resources (RCPSP). Computational experiments are performed using standard instances from the literature (those available at PSPLIB). The performance of our algorithm is compared with optimum values, lower bounds and best heuristic values reported on PSPLIB website.

This paper is organized as follows. Section 2 presents an overview of recent literature to solve the RCPSP, especially focusing on the implementation of previous genetic algorithms. Section 3 describes in detail the proposed genetic algorithm based on object-oriented programming model. Section 4 is devoted to computational experiments and analysis of results. The paper ends in Section 5 by presenting some concluding remarks.

2. Related literature

2.1. Overview of approaches to solve the RCPSP

As stated before, the RCPSP is known to be a NP-hard optimization problem (Blazewicz et al., 1983). That means that is not possible to find an efficient algorithm to optimally solve large-size instances in reasonable computational time. Some surveys about the solution techniques proposed for the RCPSP are provided by Icmeli et al. (1993), Herroelen et al. (1998), Brucker et al. (1999), Klein (1999), Hartmann and Kolisch (2000), Kolisch and Padman (2001) and Montoya-Torres (2009). Several exact methods to solve the RCPSP are proposed in the literature. Currently, the most competitive exact algorithms seem to be the ones of Demeulemeester and Herroelen (1997), Brucker et al. (1998), Klein and Scholl (1998a,b), Mingozzi et al. (1998), and Sprecher (2000). Stork and Uetz (2005) present several complexity results related to generation and counting of all circuits of an independence system, and study their relevance in the solution of RCPSP.

Several authors propose procedures for computing lower bounds on the makespan. Brucker and Knust (2003) present a destructive lower bound for the multi-mode Resource-Constrained Project Scheduling Problem with minimal and maximal time-lags. Brucker and Knust (2003) developed a destructive lower bound for the RCPSP, where the lower bound calculations are based on two methods for proving infeasibility of a given threshold value for the makespan. The first uses constraint propagation techniques, while the second is based on a linear programming formulation. Demassey et al. (2005) propose a cooperation method between constraint programming and integer programming.

Most of the heuristics methods used for solving Resource-Constrained Project Scheduling Problems either belong to the class of priority rule based methods or to the class of meta-heuristic based approaches (Kolisch and Hartmann, 1999). The first class of methods starts with none of the jobs being scheduled. Subsequently, a single schedule is constructed by selecting a subset of jobs in each step and assigning starting times to these jobs until all jobs have been considered. This process is controlled by the scheduling scheme as well as priority rules with the latter being used for ranking the jobs. Several approaches of this class have been proposed in the literature (Davis and Patterson, 1975; Cooper, 1976; Boctor, 1990; Kolisch, 1996a,b; Tormos and Lova, 2001). The second class of methods is based on the design of meta-heuristics which improve upon an initial solution. This is done by successively executing operations which transform one or several solutions into others. Several meta-heuristics approaches have been proposed in the literature: simulated annealing (Slowinski et al., 1994; Boctor, 1996; Bouleimen and Lecocq, 2003), tabu search (Pinson et al., 1994; Thomas and Salhi, 1998), local search-oriented approaches (Fleszar and Hindi, 2004; Palpant et al., 2004; Leon and Ramamoorthy, 1995), and genetic algorithms (Leon and Ramamoorthy, 1995; Lee and Kim, 1996; Hartmann, 1998, 2002; Kohlmorgen et al., 1999; Kim et al., 2003; Valls et al., 2008; Mendes et al., 2009). Since this paper proposes the use of a genetic algorithm, next section will analyze more in detail the works of these lasts authors in order to state the difference between those approaches and ours.

2.2. Previous genetic algorithms for the RCPSP

A genetic algorithm (GA) is a problem solving technique that uses the concepts of evolution and hereditary to produce good solutions to complex problems that typically have enormous search spaces and are therefore difficult to solve. A schematic representation of the behavior of a GA is presented in Fig. 1. The biggest difference with other meta-heuristics (like tabu search (TS) or simulated annealing (SA)) is that GA maintains a population of solutions rather than a unique current solution. Solutions are coded as finite-length strings called chromosomes and a measure of their adaptation (*the fitness*) is computed by an engine. Starting from an existing population, each iteration generates new chromosomes by applying operators (like crossover and mutation) to two chosen parents. The main advantage of GA is its intrinsic parallelism, which allows the exploration of a larger solution space (Sevaux and Dauzère-Pérès, 2003). A well-designed GA allows for the efficient and effective exploration and exploitation of the

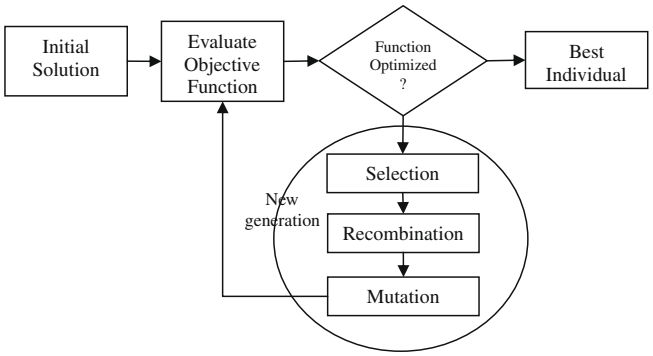


Fig. 1. General description of a GA.

problem's search space of feasible solutions in an effort to identify the global optima, or near-optimal, solution to difficult problems. Early applications of GA's are found in the literature to solve complex combinatorial optimization problems (see Back, 1995; Fogel, 1995).

In project scheduling, several authors have implemented this meta-heuristic. Lee and Kim (1996) compared the use of genetic algorithms with tabu search and simulated annealing for the classical RCPSP (with makespan minimization). Their encoded the solution as a string of number representing priorities of activities. The initial population is obtained using a random method, and three operators are performed to produce a next generation with better chromosomes. Reproduction is done by randomly selecting and duplicating a chromosome with a probability which is proportional to the fitness value. Crossover is performed by randomly selecting two chromosomes and swapping substrings after crossover point (the process is repeated until all chromosomes in the current population are mated). Mutation is also performed by randomly selecting two chromosomes and their values are changed with each other with a probability. The fitness value of solution i is computed as $\exp(-hv_i)$, where v_i is the makespan value of the solution i and $h = 0.004$ is a parameter to be chosen to make the fitness value within the range [0, 1]. Finally, these authors defined to terminate the execution once a certain number of generations do not improve the current solution.

Kohlmorgen et al. (1999) studied the impact of parallel execution strategies for genetic algorithms in various combinatorial optimization problems, including RCPSP. They particularly tested the "island model" and the "neighborhood model". In the "island model", genetic algorithms are executed concurrently on several independent sub-populations with the added possibility of regularly exchanging good individuals between neighborhood islands (Tanese, 1989). In the "neighborhood model", the population is distributed over the processors of a large mesh connected array. This spatial arrangement of the population allows the natural use of local neighborhoods in the selection of parents for producing new individuals for the next generation (Manderick and Spiessens, 1989). This study was especially focused on how the number and size of sub-populations and the migration rate, interval and strategy (for the first model) and the selection strategy and neighborhoods (for the second model) influenced the course of evolution and the quality of the generated solutions.

Kim et al. (2003) proposed a hybrid genetic algorithm in which the design of the genetic operator is based on a fuzzy logic controller and the initialization is done by a serial method. Their computational test also considered experimenting with various genetic operators such as compounded partially mapped crossover, position-based crossover, swap mutation, and local search-based mutation. Tseng and Chen (2006) also proposed a hybrid meta-heuristic procedure, named ANGEL, combining ant colony optimization (ACO), genetic algorithm (GA) and local search strategy. Their algorithm starts with the implementation of ACO to search the solution space and generate activity lists to provide the initial population for the genetic algorithm. Next, GA is executed and the pheromone set in ACO is updated when GA obtains a better solution. When GA terminates, ACO searches again by using a new pheromone set. ACO and GA search alternately and cooperatively in the solution space. A local search procedure is also included in order to improve the solution obtained by ACO or GA. A final search procedure is applied upon the termination of ACO and GA.

The hybrid genetic algorithm proposed by Valls et al. (2008) introduces several changes in the GA paradigm. These changes are: a crossover operator specific for the RCPSP, a local improvement operator that is applied to all generated schedules, a new way to select the parents to be combined, and a two-phase strategy by which the second phase re-starts the evolution from a neighbor's population of the best schedule found in the first phase.

Mendes et al. (2009) combine a genetic algorithm and a schedule generator procedure that generates parameterized active schedules. They also introduced a measure to compute a modified makespan which is used fitness function in their GA. Chromosomes represent the priorities of the activities and delay times. For each chromosome, two phases are applied: decoding of priorities, delay times, and schedule generation. The first phase is responsible for transforming the chromosomes supplied by the genetic algorithm into the priorities of the activities and delay times. The second phase makes use of priorities and the delay times defined in the first phase and constructs parameterized active schedules.

3. Proposed genetic algorithm

This section presents in detail the different components of the genetic algorithm implemented in this paper to solve the RCPSP. A main difference between our approaches and those previously existing in literature is that our proposed implementation utilizes object-oriented programming (OOP) paradigm to improve computational efficiency while maintaining effectiveness. We first explain the object-ori-

ented model we implement and then explain in detail the configuration of the genetic algorithm.

3.1. Object-oriented model

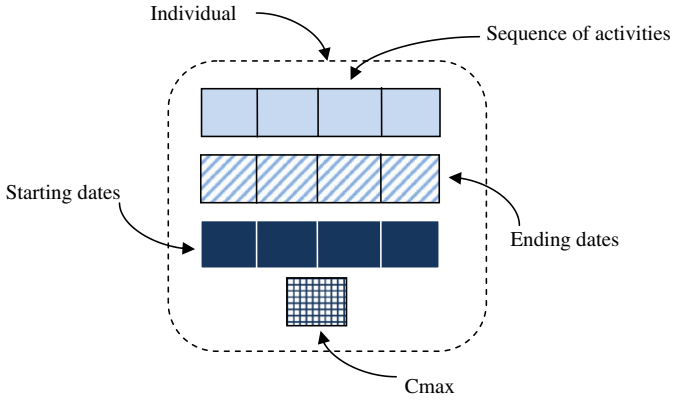
For the implementation of genetic algorithm in the case of the RCPSP, we have first developed an object-oriented conceptual model. This model allows us to identify the structure of the individuals in the algorithm. We proposed the use of object-oriented programming (OOP) model since it makes the syntax for programming powerful, but at the same time avoiding the overly complex features that have bogged down other programming languages (Flanagan, 2002). By keeping the language simple, OOP is also easier for programmers to write robust code. That is, since GA are composed by different parts and procedures that the technique needs in order to find a solution (crossover, selection, mutation, scaling scheme, and population), its behavior to find a good solution can be easily represented by an object-oriented programming approach. The OOP help necessary architecture to support all the necessary relationships for find a better solution. The process makes the GA design more effective, useful and applicable for the programmer. The class library in OOP represents chromosomes as an object with its own features. Therefore, the whole population behavior is more controllable for the programmer when using this framework through the whole algorithm. This structure helps the analyzer to put on the machine the data for encapsulate the concept and procedure of GA technique, which means that there is an attachment between data items and procedures representing a good advantage, in comparison with structured programming. The major advantage that we find is the simplicity for coding: software objects model real world objects. The complexity is hence reduced and the program structure is clearer in terms of:

- the possibility to modify the code: it is easier to make minor changes in data representation or in procedures within chromosomes,
- the easiness to make extensions, to add new features or to change parameters, and
- the maintainability of the features of chromosomes that are separated from each other.
- All this may make locating and fixing problems within the code easier than when using structured programming.

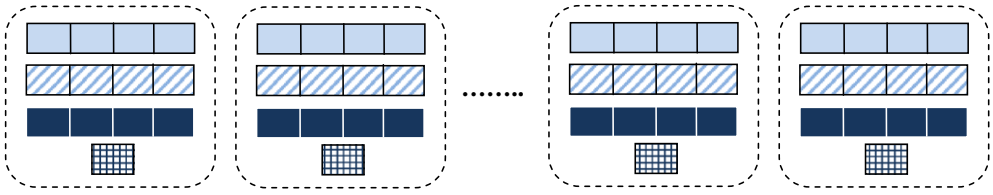
As we will explain in Section 4 (computation experiments) we used JAVA language which includes a number of new features, most notably generic types, which increase both the complexity and the power of the language. This programming language is integrated into practically all major operating systems, is simple and elegant with a well-designed, intuitive set of applications, and permits write better code with fewer bugs than for other platforms, thus reducing development time.

The RCPSP was modeled by a class called "individual" (see Fig. 2). This class contains four attributes:

(1) Activities Sequence: array that contains sequence of the activities.
(2) Starting Dates: vector that contains starting dates of each activity respecting precedence constraints.



(a) Object-oriented model of an individual (chromosome)



(b) Group of individuals (population)

Fig. 2. Object-oriented model for the genetic algorithm of this paper.

(3) Ending Dates: vector that contains the completion times of each activity.
(4) C_{max} (makespan): the value of the objective function.

Fig. 2 shows the representation based on the object-oriented model of a group of individuals. This structure is found within every generation at the *genetic algorithm*. As explained before in this section, the use of an object-oriented individual allows us to draw the structure object-oriented programming languages.

3.2. The algorithm

The pseudo-code of the algorithm is presented in Fig. 3. Each component of the code is explained next.

The main elements of a GA are solution codification, crossover operators, and evaluation function. Other important elements of the structure of a GA are the initial population, mutation operators, parents' selection rules and elimination of one or more individuals and the algorithm's stopping criterion. It is to note that the initial population is generated using a Schedule Generation Scheme (SGS): step 2 in Fig. 3.

3.3. Initial population: Schedule Generation Scheme (SGS)

Schedule Generation Schemes (SGS) are a very important part of many procedures in RCPSP. The SGS build a partial sequence, which initially sets the activity 1 (project starting activity) to time 0. A partial sequence is a sequence where only a subset of activities has been scheduled. There are two different SGS: series and parallel. The series scheme uses the increase of number of activities to make the steps, while parallel scheme uses the time increase. In this paper we used serial SGS as it generates active programs (i.e. an operation cannot be moved backwards in time without moving one or more forward) since it have showed in literature that parallel method does not generally outperform the serial method (Brucker et al., 1999; Kolisch, 1996b).

Serial SGS consists on $g = 1, \ldots, n-2$ stages, with n being the number of activities in the project. In every stage, an activity is selected and scheduled as early possible, respecting precedence relationship and resource constraints. There are two associated sets to every g stage. The set Sec_g has activities already scheduled and the set D_g contains selectable activities. Let F_g be a set of completion times for all elements of set D_g. An activity is selectable if it has not been scheduled and if all its predecessors have already been scheduled. Let $\overline{R}_k(t)$ be the remaining availability of the resource of type k at time t. The value of $\overline{R}_k(t)$ is computed as:

$$\overline{R}_k(t) = R_k - \sum_{\substack{j|S_j \leqslant t \\ f_j > t}} r_{jk} \quad \forall k \in K. \tag{1}$$

And $F_g = \{f_g | j \in Sec_g\}$ corresponds to completion times of activities. The pseudo-code of the Series algorithm is presented in Fig. 4.

At the first step, starting and completion time of dummy activity 1 are set to be 0, as this is the beginning of the project. This activity is hence included in the partial sequence. When starting each step, the algorithm computes the set D_g

Initialization:

1. Set $t = 0$.

2. Generate initial population Pt using SGS

Algorithm:

3. Evaluate the population Pt

4. WHILE stopping criterion is not met DO

5. {Select some elements from Pt to copy in $Pt+1$

6. Perform crossover on selected elements of Pt and move them to $Pt+1$, according to crossover probability

7. Perform mutation on selected elements from Pt and take them to $Pt+1$, according to mutation probability

8. Evaluate the new population $Pt+1$

9. $Pt = Pt+1$ }

End

Fig. 3. Pseudo-code of the genetic algorithm.

Step 1. Initialization

$s_1 = f_1 = 0$,

$Sec_1 = \{1\}$

Step 2. FOR $g = 2$ TO $n-1$ DO

2.1. Compute Dg, Fg, $\overline{R}_k(t)$ for $k \in K$ and $t \in Fg$

2.2. Select $j \in Dg$

2.3. $ES_j = \max_{h \in Pred_n} \{f_h\}$

2.4. $s_j = \min\{t \mid t \geq ES_j, t \in Fg, r_{jk} \leq R_k(\tau),$ for all $k \in K, \tau \in [t, t+d_j] \cap Fg\}$

2.5. $f_j = s_j + d_j$

2.6. $Sec_g = Sec_{g-1} \cup \{j\}$

Step 3. $f_n = \max_{h \in Pred_n} \{f_h\}$

Fig. 4. Pseudo-code of Series algorithm.

of selectable activities, the set F_g of completion times and the remaining capacity of each resource $\overline{R}_k(t)$. When an activity j is selected from the set of selectable activities, its earliest starting time ES_j is computed, respecting the precedence constraints.

A feasible sequence $S = (s_1, s_2, \ldots, s_n)$ is said to be left-active or left-justified if there is no activity that may be started before the sequence starts without delaying any other activity. Similarly, a feasible sequence $S = (s_1, s_2, \ldots, s_n)$ is said to be right-active or right-justified if there is no activity that can be started after the sequence starts without delaying any other activity or delaying the sequence duration. Kolisch (1996a,b) proved that Series SGS generates active sequences. Series SGS is able to transform any list of activities in a feasible sequence. To do so, it is necessary to change the step 2.2 in algorithm of Fig. 5 for "$j = j_g$" (Hartmann, 1998). Parallel SGS may also apply, computing a priority vector according to the activities list by using $p_i = order(i, \lambda)$, which is the order or position of activity i in sequence λ, and then selecting activity in set D_g with the lowest priority level. This modification leads to selecting activities to schedule in an order not designated by the activities list. Hence, usually Serial SGS is used as the scheduling scheme when implementing this representation.

With Serial SGS codification all active sequences can be obtained. In general, an active sequence admits more than one representation as activity list and, *a priori*, redundancy is not controllable.

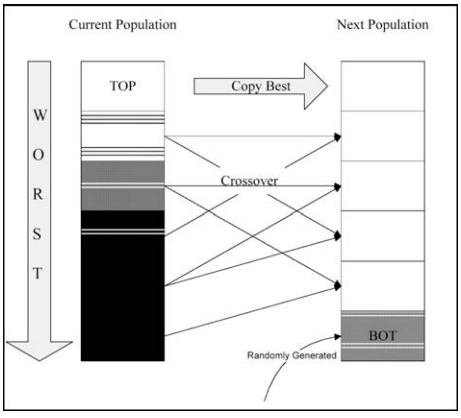


Fig. 5. Schematic representation of the evolution strategy (TOP is the top of the list, and BOT corresponds to the bottom part of the list).

3.4. Chromosome representation

The chromosome representation for the RCPSP presented here is based on the concept of activities list. According to Hartmann (2002), compared with other representations, the activities list has shown better performances. In an activities list $\lambda = (j_1, j_2, \ldots, j_J)$, each non-dummy activity appears exactly once. Only activities lists that meet precedence constraints are considered. Obviously, it is possible to create an activities list if predecessor activities are executed always before any activity on the list. This can be formally expressed as: $P_j \subset \{0, j_1, \ldots, j_{i-1}\}$ for $i = 1, \ldots, J$. An individual is represented by $I = (j_1^I, \ldots, j_J^I)$. In the activities sequence, precedence constraints are supposed to be met. This can be formally expressed as: $\{j_1^I, \ldots, j_J^I\} = \{1, \ldots, J\}$ and $P_{j_1^I} \subseteq \{0, j_1^I, \ldots, j_{i-1}^I\}$ for $i = 1, \ldots, J$. Each chromosome is related to a schedule that is computed by using Serial SCS. The initial node that corresponds to a dummy activity starts at time 0.

Following activities are scheduled in the order defined by list $(j_1, j_2, \ldots, j_J)$.

3.5. Crossover operator

Two types of crossover operators were considered: one-point and two-point crossover.

3.5.1. One-point crossover

Two individuals, denoted as Parent 1 (λ^1) and Parent 2 (λ^2) are selected, with $\lambda^1 = \{j_1^1, \ldots, j_n^1\}$ and $\lambda^2 = \{j_1^2, \ldots, j_n^2\}$. Then an integer number q is generated randomly between 1 and J to obtain two new individuals: Son 1 (h^1) and Son 2 (h^2). Activities in positions $i = 1, \ldots, q$ in Son 1 (h^1) are taken form Parent 1 (λ^1) as $h^1(j_i) = \lambda^1(j_i)$. Activities in positions $i = q+1, \ldots, J$ in Son 1 (h^1) are taken from Parent 2 (λ^2) without including activities already taken from Parent 1: $h^1(j_i) = \lambda^2(j_k)$, where k is the smallest index $\lambda^2(j_k) \notin \{h^1(j_1, \ldots, j_{i-1})\}$. As a result, relative positions on Parents' list are preserved. As an example, let us consider Parent 1 $\lambda^1 = \{1, 3, 2, 5, 4, 6\}$ and Parent 2 $\lambda^2 = \{2, 4, 6, 1, 3, 5\}$. With $q = 3$, Son 1 is $h^1(1, 3, 2, 6, 4, 5)$.

3.5.2. Two-point crossover

Two-point crossover is an extension of one-point crossover. Two integer numbers q_1 and q_2 are randomly generated with $1 \leqslant q_1 \leqslant q_2 \leqslant J$. Now, Son 1 h^1 is generated with activities list on positions $i = 1, \ldots, q_1$ taken from Parent 1 λ^1, that is $h^1(j_i) = \lambda^1(j_i)$; activities in positions $i = q_1 + 1, \ldots, q_2$ are taken from Parent 2 λ^2, that is $h^1(j_i) = \lambda^2(j_k)$ where k is the smallest index as $\lambda^2(j_k) \notin \{h^1(j_1, \ldots, j_{i-1})\}$; and finally positions $i = q_2 + 1, \ldots, J$ are again taken from Parent 1 λ^1, that is $h^1(j_i) = \lambda^1(j_k)$, where k is the smallest index as $\lambda^1(j_k) \notin \{h^1(j_1, \ldots, j_{i-1})\}$. Taken the same example of the previous subsection, let us consider Parent 1 $\lambda^1 = \{1, 3, 2, 5, 4, 6\}$ and Parent 2 $\lambda^2 = \{2, 4, 6, 1, 3, 5\}$. With $q_1 = 1$ and $q_2 = 2$, Son 1 is $h^1(1, 2, 4, 3, 5, 6)$.

3.6. Mutation

Given the chromosome based on the activities list λ, the mutation operator modifies the structure of the chromosome as follows. For every position $i = 1, \ldots, j-1$, activities j_i and j_{i+1} switch their positions with probability p_{mut} if the resulting activities list continues to meet the precedence constraints. Mutation operator may create activities list that have not yet been created by crossover operators.

3.7. Evolution strategy and fitness function

The evolution strategy implemented in our algorithm is based on the elitist list. It consists on selecting the best individual (or set of individuals) and then using it to generate an important number of individuals in the next generation, while the rest of the population is generated from all the individuals. Fig. 5 shows a schematic representation of this strategy.

Finally, the fitness function of our genetic algorithm is defined as makespan-dependent. Since our objective is to minimize the makespan of the project, the fitness function is computed as:

$$Fitness = \frac{1}{C_{\max}}. \tag{2}$$

4. Computational experiments

4.1. Data sets and experimental conditions

The genetic algorithm was programmed using JAVA and run using Eclipse interface on a PC with AMD Athlon™ 64X2 Dual Core processor 4800+ with 2.51 GHz and 1.87 GB of RAM. For the experimental design, instances were taken from the library PSPLIB available at: http://129.187.106.231/psplib/. The whole set of projects with 30, 60 and 90 activities were considered (data sets J30, J60, and J90). On this website, optimum, lower bounds and best heuristic values for the makespan are given for those data sets.

In order to run the algorithm, the following conditions have been defined. The population size was defined to be 100 or 150 individuals. Two types of crossover were considered in initial set of runs: one-point and two-point crossover. After preliminary experiments, two-point crossover showed to outperform one-point crossover (Gutierrez-Franco et al., 2007). Hence, for the experiments presented in this paper, two-point crossover was considered. According to the set of instances published on the above-mentioned website and the combination factors in this experimental design, a total of 2400 different instances were used in the experiment. A total of 100 iterations were run for each instance and each combination of parameters. Crossover probability was defined to be 0.7, while mutation probability was defined as 0.1, since a low value gives good results in empirical studies (Goldberg, 1989; Lee and Kim, 1996).

4.2. Experimental results

Tables 1 and 2 present the summary of average results over all replications for experiments with different number of jobs, respectively with populations of 100 and 150 individuals. Columns corresponding to the optimum makespan (OPT), the lower bound value (LB), and the best approximation value obtained using a heuristic algorithm (H) was taken from the PSPLIB website. The column in the table corresponding to the average of the makespan value obtained using the proposed genetic algorithm is denoted as OOP-GA. The deviation, in percentage, (*%dev*) of the proposed algorithm from the optimum (*%dev(opt)*), from the lower bound (*%dev(lb)*) and from the best heuristic value (*%dev(bh)*) are presented. These values are computed

Table 1
Results with population of 100 individuals.

Number of activities	OPT	BH	LB	OOP-GA	Run (s)	*%dev(opt)* (%)	*%dev(bh)* (%)	*%dev(lb)* (%)
30	54.84		54.84	56.24	1.34	3		3
60	–	74.29	74.24	77.84	4.32		5	5
90	–	85.75	87.38	94.07	8.06		10	8
Average deviation						**3**	**7**	**5**

Table 2
Results with population of 150 individuals.

Number of activities	OPT	BH	LB	OOP-GA	Run (s)	*%dev(opt)* (%)	*%dev(bh)* (%)	*%dev(lb)* (%)
30	54.84		54.84	56.11	1.99	2		2
60	–	74.29	74.24	77.68	5.85		5	5
90	–	85.75	87.38	93.90	11.19		9	7
Average deviation						**2**	**7**	**5**

using Eq. (3), where Z represents, respectively, the optimum (OPT), the lower bound (LB) or the best heuristic (BH) value of the makespan taken as an average for each number of activities

$$\%dev = \frac{C_{\max}(\text{OOP-GA}) - C_{\max}(Z)}{C_{\max}(Z)} \times 100\%. \quad (3)$$

We can observe in Table 1 that the solution value given by our procedure is never greater that 10% of either the optimum, the best currently known makespan value or the lower bound. It is to note although that with the increase in the population size from 100 to 150 individuals, the gap between the OOP-GA solution and previous values decreases in average, for each respective instance set. Table 3 presents the percentage of times, over the total number of instances, that the proposed heuristic obtained the same makespan value than previous results. We can observe that for instance, for the case of 30 activities, the proposed OOP-GA obtained the optimal solution in 58% of the cases when the population size was 100 individuals, and in 64% of the cases with a population of 150 individuals. For larger instances, that is with 60 and 90 activities in the project, our heuristic obtained the same value of the best heuristic or even the lower bound in always more than 53% of the instances.

Table 3
Percentage of times that the proposed OOP-GA equals previous results.

Number of activities	Population size 100			Population size 150		
	OPT (%)	LB (%)	BH (%)	OPT (%)	LB (%)	BH (%)
30	58	–	58	64	–	64
60	–	55	55	–	53	53
90	–	59	59	–	58	58

4.3. Comparison with state-of-the-art algorithms

In this subsection we compare the performance of our genetic algorithm with other genetic algorithms previously published in scientific literature. As far as the paper is concerned we have looked at the algorithms referred to in Kolisch and Hartmann (2006). We have also included results not presented in that reference since they have published latter: those proposed by Valls et al. (2008) and Mendes et al. (2009). Among all those algorithms, we have to include only those which are based on genetic algorithms.

Table 4
Performance deviation from optimum (J30) or lower bound (J60) of OOP-GA with some state-of-the-art algorithms.

Algorithm	SGS	Reference	Percent deviation	
			J30 (%)	J60 (%)
OOP-GA	Serial	This paper	2	5
GA-DBH	Serial	Debels and Vanhoucke (2005)	0.02	10.68
GA, TS-path relinking	Serial	Kochetov and Stolyar (2003)	0.00	10.74
GAPS	Param. active	Mendes et al. (2009)	0.01	10.67
Hybrid GA	Serial	Valls et al. (2008)	0.02	10.73
GA-self adapting	Serial	Hartmann (2002)	0.08	11.21
GA-activity list	Serial	Hartmann (1998)	0.08	11.23
Population-based	Serial	Valls et al. (2004)	0.10	10.89
GA-random key	Serial	Hartmann (1998)	0.23	12.25
GA-priority rule	Serial	Hartmann (1998)	0.88	12.26
GA-problem space	Mod. Par.	Leon and Ramamoorthy (1995)	1.59	13.49

This comparison of computational results is presented in Table 4 for instance sets with 30 and 60 activities, respectively (instances with 90 activities are not reported in such references). The table is divided into five columns. First column reports some keywords describing the algorithm. Second and third columns correspond, respectively, to the Schedule Generation Scheme (SGS) employed by the algorithm and its authors (reference). Fourth and fifth columns finally present the percentage deviation from either the optimum (projects with 30 activities) or the lower bound (projects with 60 activities) value of the makespan as reported in the above-mentioned references. We can observe that for projects with 30 activities, our algorithm did not outperformed previous results from literature. However, our OOP-GA algorithm clearly outperformed previous results and ranks first for the case of projects with 60 activities (data sets J60).

5. Concluding remarks

This paper studied the problem of project scheduling with activities sharing limited resources. In the literature this problem is commonly known as the Resource-Constrained Project Scheduling Problem (RCPSP) and is difficult to solve since it is NP-complete. This means that is not possible to find optimal solutions for large-size instances in reasonable computational time. This paper proposed a genetic algorithm to solve the problem in which the chromosome representation is based on the features of object-oriented programming. This is intended to allow the design of efficient decision support system for project management. An experimental study was performed with instances taken from the literature, available at the PSPLIB website. Results showed that our solution procedure is enough efficient in terms of computational time and effective in solution quality, by given the same solution value of the optimum makespan or previous best heuristic for more than 50% of the studied instances. Since these results are encouraging, we are now working on the design of a decision support tool based on this genetic algorithm. Researches will now address the problem of including non-renewable resources and other objective functions to include activity costs.

Acknowledgements

This work is part of a project supported by Research Funds from Universidad de La Sabana, Colombia, under Grant CEA-24-2008. This paper is also part of the research work under the HAROSA KC. Authors wish to acknowledge the anonymous referees for their comments to improve the presentation of this paper.

References

Back, T., 1995. Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York.

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5, 11–24.

Boctor, F.F., 1990. Some efficient multi-heuristic procedures for resource-constrained project scheduling. European Journal of Operational Research 49, 3–13.

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.

Bouleimen, K., Lecocq, H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149, 268–281.

Brucker, P., Knust, S., 2003. Lower bounds for resource-constrained project scheduling problems. European Journal of Operational Research 149, 302–313.

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107, 272–288.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: notation, classification, models, and methods. European Journal of Operational Research 112, 3–41.

Cooper, D.F., 1976. Heuristics for scheduling resource-constrained projects: an experimental investigation. Management Science 22, 1186–1194.

Davis, E.W., Patterson, J.H., 1975. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Management Science 21, 944–955.

Debels, D., Vanhoucke, M., 2005. A decomposition-based heuristic for the resource-constrained project scheduling problem. Technical Report 2005/293, Faculty of Economics and Business Administration, University of Ghent, Ghent, Belgium.

Demassey, S., Artigues, C., Michelon, P., 2005. Constraint-propagation-base cutting planes: an application to the resource-constrained project scheduling problem. INFORMS Journal of Computing 17, 52–65.

Demeulemeester, E., Herroelen, W., 1997. New benchmark results for the resource-constrained project scheduling problem. Management Science 43, 1485–1492.

Flanagan, D., 2002. Java in a Nutshell, fifth ed. O'Reilly.

Fleszar, K., Hindi, K.S., 2004. Solving the resource-constrained project scheduling problem by a variable neighborhood search. European Journal of Operational Research 155, 402–413.

Fogel, D.B., 1995. Evolutionary Computation, Towards a New Philosophy of Machine Intelligence. IEEE Press.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wiley, Reading, USA.

Gutierrez-Franco, E., La Torre-Zurita, F., Mejía-Delgadillo, G., 2007. A genetic algorithm for the resource constrained project scheduling problem (RCPSP). In: Proceedings of the 19th International Conference on Production Research (ICPR-19). Valparaiso, Chile, July 29–August 2, 2007. CD-ROM.

Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45, 279–302.

Hartmann, S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics 49, 433–448.

Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127, 394–407.

Herroelen, W., De Reyck, B., Demeulemeeste, E., 1998. Resource-constrained project scheduling: a survey of recent developments. Computers & Operations Research 25, 279–302.

Icmeli, O., Erenguc, S.S., Zappe, C.J., 1993. Project scheduling problems: a survey. International Journal of Operations & Production Management 13, 80–91.

Kim, K.W., Mitsuo, G., Yamazaki, G., 2003. Hybrid genetic algorithm with fuzzy logic for resource-constrained project scheduling. Applied Soft Computing 2, 174–188.

Klein, R., 1999. Scheduling of resource-constrained projects. Kluwer Academic Publishers, Dordrecht.

Klein, R., Scholl, A., 1998a. Progress: optimally solving the generalized resource-constrained project scheduling problem. Technical Report, University of Technology, Darmstadt.

Klein, R., Scholl, A., 1998b. Scattered branch and bound: an adaptive search strategy applied to resource-constrained project scheduling problem. Technical Report, University of Technology, Darmstadt.

Kochetov, Y., Stolyar, A., 2003. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the 3rd International Workshop on Computer Science and Information Technologies.

Kohlmorgen, U., Schmeck, H., Haase, K., 1999. Experiences with fine-grained parallel genetic algorithms. Annals of Operations Research 90, 203–219.

Kolisch, R., 1996a. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14, 179–192.

Kolisch, R., 1996b. Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. European Journal of Operational Research 90, 320–333.

Kolisch, R., Hartmann, S., 1999. Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Weglarz, J. (Ed.), Handbook on Recent Advances in Project Scheduling. Kluwer Academic Publishers, Dordrecht, pp. 147–178.

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., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega 29, 249–272.

Lee, J.K., Kim, Y.D., 1996. Search heuristics for resource constrained project scheduling. Journal of the Operational Research Society 47, 678–689.

Leon, V.J., Ramamoorthy, B., 1995. Strength and adaptability of problem-space based neighborhoods for resource constrained scheduling. OR Spectrum 17, 173–182.

Manderick, B., Spiessens, P., 1989. Fine-grained parallel genetic algorithms. In: Schaffer, J.D., Kaufmann, M. (Eds.), Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 428–433.

Mendes, J.J., Gonçalves, J.F., Resende, M.G.C., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers & Operations Research 36, 92–109.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1998. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. Management Science 44, 714–729.

Montoya-Torres, J.R., 2009. Literature review of meta-heuristic procedures for resource-constrained project scheduling problems. Research Report EICEA-200906-02, School of Economics and Management Sciences, Universidad de La Sabana, Colombia.

Palpant, M., Artigues, C., Michelon, P., 2004. LSSPER: solving the resource-constrained project scheduling problem with large neighborhood search. Annals of Operations Research 131, 237–257.

Pinson, E., Prins, C., Rullier, F., 1994. Using tabu search for solving the resource constrained project scheduling problem. Technical Report, Universite Catholique de l'Ouest, Angers.

Sevaux, M., Dauzère-Pérès, S., 2003. Genetic algorithms to minimize the weighted number of late jobs on a single machine. European Journal of Operational Research 151, 296–306.

Slowinski, R., Soniewicki, B., Weglarz, J., 1994. DSS for multiobjective project scheduling. European Journal of Operational Research 79, 220–229.

Sprecher, A., 2000. Scheduling resource-constrained projects competitively at modest memory requirements. Management Science 46, 710–723.

Stork, F., Uetz, M., 2005. On the generation of circuits and minimal forbidden sets. Mathematical Programming 102, 185–203.

Tanese, R., 1989. Distributed genetic algorithms. In: Schaffer, J.D., Kaufmann, M. (Eds.), Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 428–433.

Thomas, P.R., Salhi, S., 1998. A tabu search approach for the resource constrained project scheduling problem. Journal of Heuristics 4, 123–139.

Tormos, P., Lova, A., 2001. A competitive heuristic solution technique for resource-constrained project scheduling. Annals of Operations Research 102, 65–81.

Tseng, L.Y., Chen, S.C., 2006. A hybrid metaheuristic for the resource-constrained project scheduling problem. European Journal of Operational Research 175, 707–721.

Valls, V., Ballestín, F., Quintanilla, S., 2004. A population-based approach for the resource-constrained project scheduling problem. Annals of Operations Research 131, 305–324.

Valls, V., Ballestín, F., Quintanilla, S., 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 185, 495–508.