

A RANDOM KEY BASED GENETIC ALGORITHM FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

J.J.M. MENDES, J.F. GONÇALVES, AND M. G. C. RESENDE

ABSTRACT. This paper presents a genetic algorithm for the Resource Constrained Project Scheduling Problem (RCPSP). The chromosome representation of the problem is based on random keys. The schedule is constructed using a heuristic priority rule in which the priorities of the activities are defined by the genetic algorithm. The heuristic generates parameterized active schedules. The approach was tested on a set of standard problems taken from the literature and compared with other approaches. The computational results validate the effectiveness of the proposed algorithm.

1. INTRODUCTION

The *Resource Constrained Project Scheduling Problem* (RCPSP) can be stated as follows. A project consists of $n + 2$ activities where each activity has to be processed in order to complete the project. Let $J = \{0, 1, \dots, n, n + 1\}$ denote the set of activities to be scheduled and $K = \{1, \dots, k\}$ the set of resources. Activities 0 and $n + 1$ are dummies, have no duration, and represent the initial and final activities. The activities are interrelated by two kinds of constraints:

- (1) *Precedence constraints* force each activity j to be scheduled after all predecessor activities P_j are completed;
- (2) Activities require resources with *limited capacities*.

While being processed, activity j requires $r_{j,k}$ 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. The parameters d_j , $r_{j,k}$, and R_k are assumed to be integer, non-negative, and deterministic. For the project start and end activities, we have $d_0 = d_{n+1} = 0$ and $r_{0,k} = r_{n+1,k} = 0$, for all $k \in K$. The problem consists in finding a schedule of the activities, taking into account the resources and the precedence constraints, that minimizes the makespan C_{\max} .

Let F_j represent the finish time of activity j . A schedule can be represented by a vector of finish times $(F_1, \dots, F_m, \dots, F_{n+1})$. Figure 1 shows an example of a project comprising $n = 6$ activities which have to be scheduled, subject to two renewable resource types with a capacity of four and two units, respectively. A feasible schedule with an optimal makespan of 15 time-periods is represented in Figure 2.

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

Date: June 30, 2005. Revised November 8, 2006.

Key words and phrases. Project management, scheduling, genetic algorithms, random keys, RCPSP. AT& Labs Research Technical Report TD-6DUK2C.

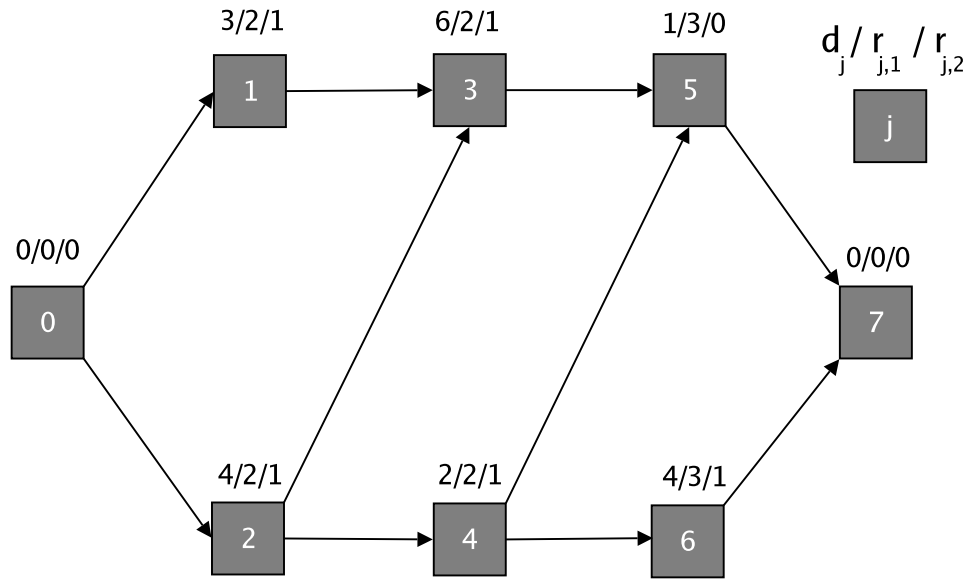


FIGURE 1. Project network example. Activities are represented as boxes and precedences by directed arcs. Parameters $d_j/r_{j,1}/r_{j,2}$ are given for each activity j .

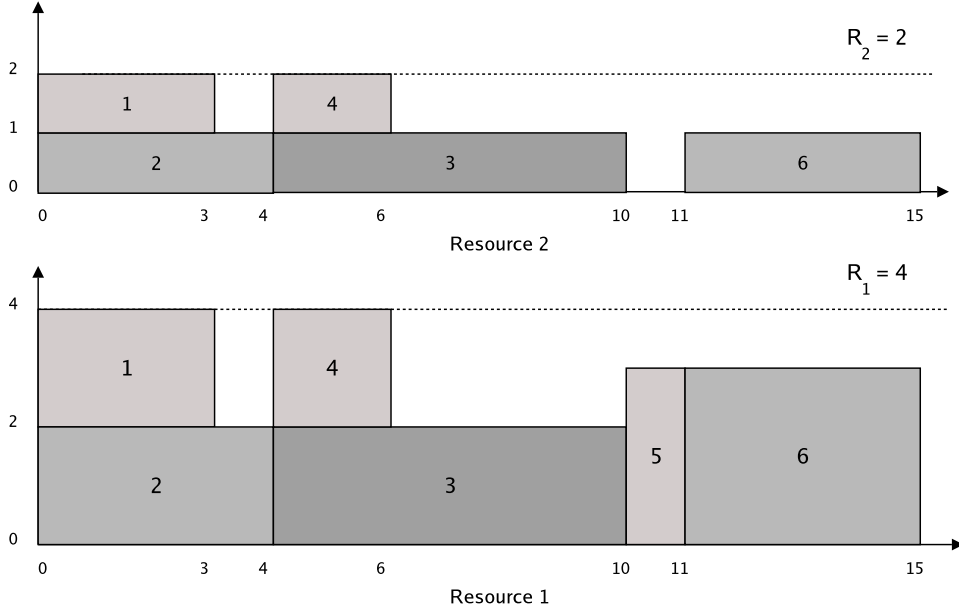


FIGURE 2. Feasible schedule with an optimal makespan of 15 for the project network example of Figure 1.

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.

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.

Several authors propose procedures for computing lower bounds on the makespan. Demassez et al. (2005) propose a cooperation method between constraint programming and integer programming. 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 (2000) 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.

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, e.g. Alvarez-Valdez and Tamarit (1989), Boctor (1990), Cooper (1976, 1977), Davis and Patterson (1975), Lawrence (1985), Kolisch (1996a,b), Kolisch and Hartmann (1999), and Tormos and Lova (2001, 2003). The second class of methods improves upon an initial solution. This is done by successively executing operations which transform one or several solutions into others. Several approaches of this class have been proposed in the literature, e.g. genetic algorithms (Leon and Ramamoorthy (1995), Lee and Kim (1996), Hartmann (1998), Kohlmorgen et al. (1999), Hartmann (2002), Kochetov and Stolyar (2003), Mendes (2003), Valls et al. (2003, 2005), and Debels and Vanhoucke (2005)), simulated annealing (Slowinski et al. (1994), Boctor (1996), and Bouleimen and Lecocq (2003)), tabu search (Pinson et al. (1994), Baar et al. (1998), Thomas and Salhi (1998), Nonobe and Ibaraki (2002), and Gagnon et al. (2004)), local search-oriented approaches (Fleszar and Hindi (2004) and Palpant et al. (2004)), and population-based approaches (Debels et al. (2006) and Valls et al. (2003)).

Some surveys are provided by Icmeli et al. (1993), Herroelen et al. (1998), Brucker et al. (1999), Klein (1999), Kolisch and Hartmann (1999), Hartmann and Kolisch (2000), Kolisch and Padman (2001), and Demeulemeester and Herroelen (2002). Kolisch and Hartmann (2005) and Brucker and Knust (2006) present models and algorithms for complex scheduling problems and discuss the RCPSP.

In this paper, we present a new genetic algorithm for finding cost-effective solutions for the resource constrained project scheduling problem. The remainder of the paper is organized as follows. Section 2 presents the different classes of schedules. In Section 3, we present our hybrid approach to solve the resource constrained project scheduling problem: genetic algorithm and schedule generation procedure. Section 4 reports computational results and concluding remarks are made in Section 5.

2. TYPES OF SCHEDULES

Schedules can be classified into one of following three types of schedules:

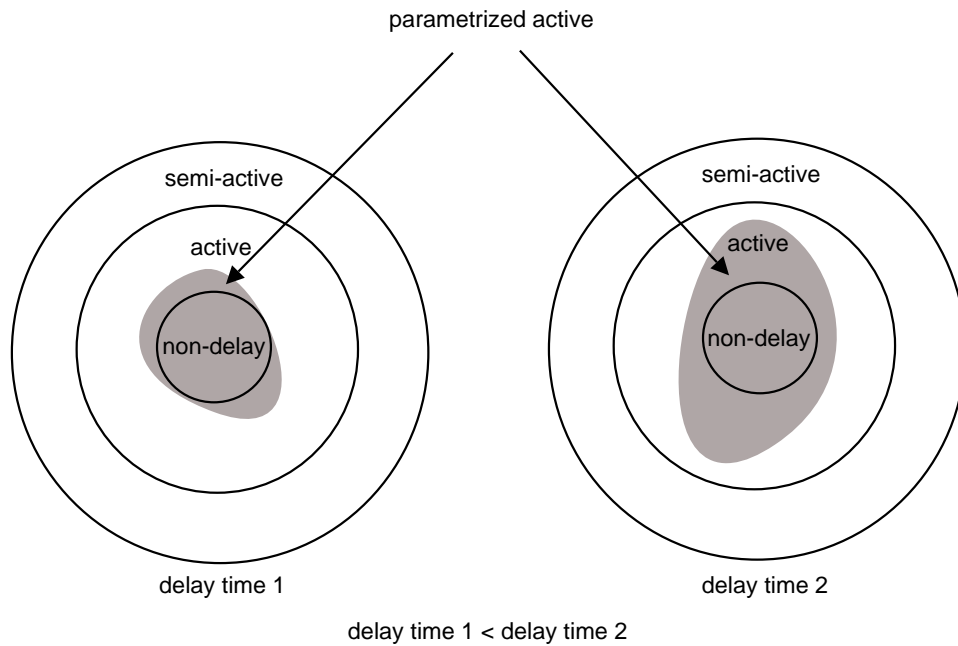


FIGURE 3. Parameterized active schedules for different values of the delay time.

- (1) *Semi-active schedules.* These are feasible schedules obtained by sequencing activities as early as possible. In a semi-active schedule, no activity can be started earlier without altering the processing sequences.
- (2) *Active schedules.* These are feasible schedules in which no activity could be started earlier without delaying some other activity or breaking a precedence constraint. Active schedules are also semi-active schedules. An optimal schedule is always active, so the search space can be safely limited to the set of all active schedules.
- (3) *Non-delay schedules.* These are feasible schedules in which no resource is kept idle when it could start processing some activity. Non-delay schedules are active and hence are also semi-active.

Later in this paper we extend the use of parameterized active schedules as proposed in Gonçalves and Beirão (1999) and Gonçalves et al. (2005). This type of schedule consists of schedules in which no resource is kept idle for more than a predefined period if it could start processing some activity. If the predefined period is set to zero, then we obtain a non-delay schedule. The basic concepts of this type of schedule are presented in the next section.

2.1. Parameterized active schedules. As mentioned above, the optimal schedule is in the set of all active schedules. However, the set of active schedules is usually very large and contains many schedules with relatively large delay times, hence with poor quality in terms of makespan. To reduce the solution space and to control the delay times, we use the concept of parameterized active schedules.

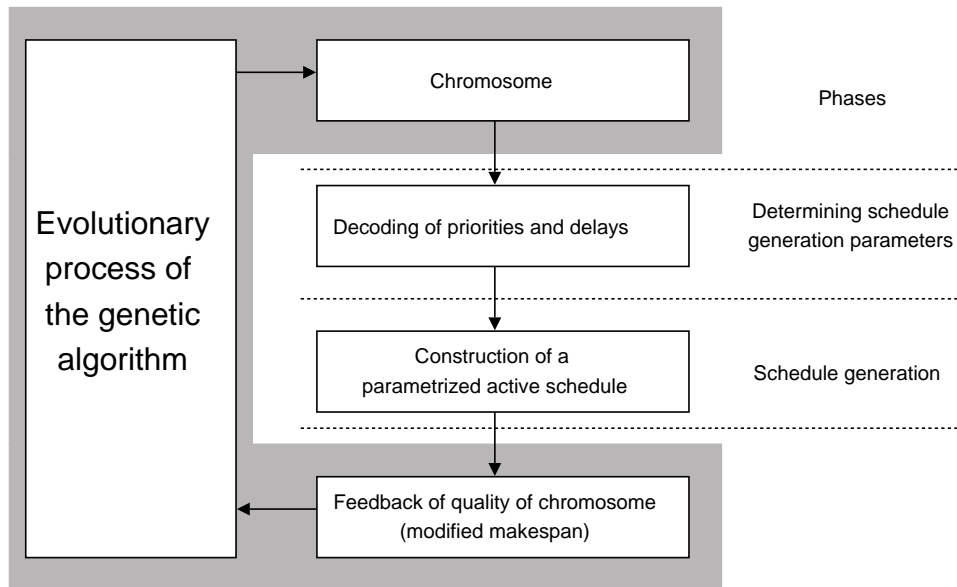


FIGURE 4. Architecture of the new approach.

Figure 3 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay schedules. By controlling the maximum delay time allowed, one can reduce or increase the size of the solution space. A maximum delay time equal to zero is equivalent to restricting the solution space to non-delay schedules. Section 3.2 presents the pseudo-code to generate parameterized active schedules.

3. NEW APPROACH

The new approach combines a genetic algorithm and a schedule generator procedure that generates parameterized active schedules and a novel measure of merit that computes a *modified makespan* value which is used as fitness measure (quality measure) to feedback to the genetic algorithm. The genetic algorithm evolves the chromosomes which represent the priorities of the activities and delay times. For each chromosome the following two phases are applied:

- (1) *Decoding of priorities, delay times.* This phase is responsible for transforming the chromosome supplied by the genetic algorithm into the priorities of the activities and delay times.
- (2) *Schedule generation.* This phase makes use of the priorities and the delay times defined in the first phase and constructs parameterized active schedules.

After a schedule is obtained, the corresponding measure of quality (*modified makespan*) is feedback to the genetic algorithm. Figure 4 illustrates the sequence of steps applied to each chromosome generated by the genetic algorithm. Details about each of these phases will be presented in the next sections.

3.1. Genetic algorithm. Genetic algorithms are adaptive methods, which may be used to solve search and optimization problems (Beasley et al., 1993). Before a genetic algorithm can be run, a suitable *encoding* (or representation) for the problem must be devised.

A *fitness function* is also required, which assigns a figure of merit to each encoded solution. During the execution of the algorithm, parents must be *selected* for reproduction, and *recombined* to generate offspring.

It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as *genes*) are joined together to form a string of values (*chromosome*). The set of parameters represented by a particular chromosome is referred to as an *individual*. The fitness of an individual depends on its chromosome and is evaluated by the fitness function.

The individuals, during the reproductive phase, are selected from the population and recombined, producing offspring, which comprise the next generation. Parents are randomly selected from the population using a scheme, which favors fitter individuals. Having selected two parents, their chromosomes are recombined, typically using mechanisms of *crossover* and *mutation*. Mutation is usually applied to some individuals to guarantee population diversity.

3.1.1. *Chromosome representation.* The genetic algorithm described in this paper uses a random-key alphabet which is comprised of real random numbers between 0 and 1. The evolutionary strategy used is similar to the one proposed by Bean (1994), the main difference occurring in the crossover operator. The important feature of random keys is that all offspring formed by crossover are feasible solutions. This is accomplished by moving much of the feasibility issue into the objective function evaluation. If any random-key vector can be interpreted as a feasible solution, then any crossover vector is also feasible. Through the dynamics of the genetic algorithm, the system *learns* the relationship between random-key vectors and solutions with good objective function values.

A chromosome represents a solution to the problem and is encoded as a vector of random keys. In a direct representation, a chromosome represents a solution of the original problem, and is called *genotype*, while in an indirect representation it does not and special procedures are needed to derive a solution from it called *phenotype*.

In the present context, the direct use of schedules as chromosomes is too complicated to represent and manipulate. In particular, it is difficult to develop corresponding crossover and mutation operations. Instead, solutions are represented indirectly by parameters that are later used by a schedule generator to obtain a solution. To obtain the solution, we use the parameterized active schedule generator described in Section 3.2.

Each solution chromosome is made of $2n$ genes, where n is the number of activities.

$$chromosome = \underbrace{(gene_1, \dots, gene_n)}_{\text{priorities}} \underbrace{(gene_{n+1}, \dots, gene_{2n})}_{\text{delay times}}$$

The first n genes are used to determine the priority of each of the n activities. The genes between $n + 1$ and $2n$ are used to determine the delay times used at each of the n iterations of the scheduling procedure.

3.1.2. *Decoding the priorities of the activities.* In an earlier version of the algorithm, the priorities of the activities were given directly by the genetic algorithm, i.e.

$$PRIORITY_j = gene_j, \text{ for all } j = 1, \dots, n.$$

Though this approach worked quite well, we conjectured that it could be improved if we could somehow provide the genetic algorithm with information about the structure of the problem. The priority decoding expression used in the current version of the algorithm combines what we consider the *ideal* priority under infinite capacity and a factor that corrects this value to account for the real resource load and capacity availability. For the ideal

priority under infinite capacity we use the term LLP_j/LCP , where LLP_j is the longest-length path from the beginning of activity j to the end of the project and LCP is length along the critical path of the project. It is clear that $0 \leq LLP_j/LCP \leq 1$.

The factor that adjusts the priority to account for capacity is given by $(1 + gene_j)/2$. Therefore, the final decoding expression is given by

$$PRIORITY_j = \frac{LLP_j}{LCP} \times \left[\frac{1 + gene_j}{2} \right], \quad j = 1, \dots, n.$$

3.1.3. *Decoding the delay times.* The genes between positions $n + 1$ and $2n$ are used to determine the delay times used when scheduling an activity. The delay time $delay_g$ used by each scheduling iteration g is given by

$$delay_g = gene_{n+g} \times 1.5 \times maxDur,$$

where $maxDur$ is the maximum duration of all activities. The factor 1.5 was obtained after some experimental tuning.

3.1.4. *Fitness function.* Different schedules can have the same value of the makespan. However, the potential for improvement of schedules with the same makespan value might not be the same. To differentiate the potential for improvement between schedules having the same makespan we developed a new measure of fitness, that we call *modified makespan*.

The modified makespan combines the makespan of the schedule with a measure of the potential for improvement of the schedule. This measure of potential for improvement of the schedule will have values in the interval $]0, 1[$. The rationale for this new measure is that if we have two schedules with the same makespan value, then the one with a smaller number of activities ending close to the makespan will have more potential for improvement.

To define the modified makespan, we introduce the concept of the distance of one activity to the final activity (activity $n + 1$). The distance $dist(j)$ of activity j to activity $n + 1$ is equal to the number of activities in the path connecting them that has the smallest number of activities, including activity j and excluding activity $n + 1$.

The *modified makespan* of distance L is given by

$$(1) \quad F_{n+1} + \frac{\sum_{d=1}^L \sum_{i | dist(i)=d} F_i}{\sum_{d=1}^L \sum_{i | dist(i)=d} F_{n+1}}.$$

Note that when $L = 0$, we get a modified makespan equal to the usual makespan. We will use the project network example given in Figure 1 and the schedule presented in Figure 2 to illustrate the calculation of the modified makespan. The distances of activities 1 to 6 are:

activity i	1	2	3	4	5	6
$dist(i)$	3	3	2	2	1	1

The makespan of the project is 15 (see Figure 2). The modified makespan of distance $L = 1$ is

$$15 + \frac{(15 + 11)}{(15 + 15)} = 15.867.$$

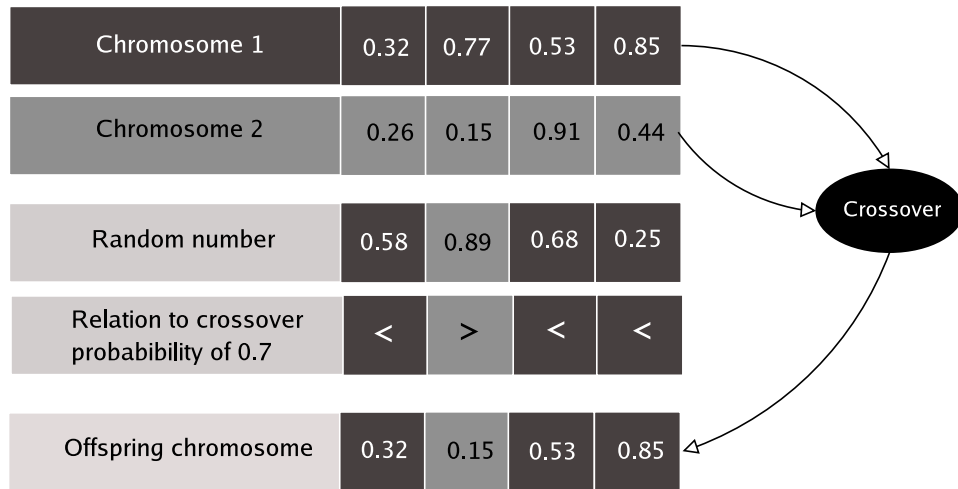


FIGURE 5. Example of parameterized uniform crossover with crossover probability equal to 0.7. For each gene, a random number in the interval $[0, 1]$ is generated. With probability 0.7 the offspring inherits the gene of Chromosome 1 and with probability 0.3, it inherits the one of Chromosome 2.

The modified makespan of distance $L = 2$ is

$$15 + \frac{(15 + 11) + (6 + 10)}{(15 + 15) + (15 + 15)} = 15.7.$$

3.1.5. *Evolutionary strategy.* To breed good solutions, the random-key vector population is operated upon by a genetic algorithm. Many variations of genetic algorithms can be obtained by varying the reproduction, crossover, and mutation operators. The reproduction and crossover operators determine which parents will have offspring, and how genetic material is exchanged between the parents to create those offspring. Reproduction and crossover operators tend to increase the quality of the populations and force convergence. Mutation opposes convergence since it allows for random alteration of genetic material. Given a current population, we perform the following three steps to obtain the next generation:

- (1) *Reproduction.* Some of the best individuals are copied from the current generation into the next (see *TOP* in Figure 6). This strategy is called *elitist* (Goldberg, 1989) and its main advantage is that the best solution is monotonically improving from one generation to the next. However, it can lead to a rapid population convergence to a local minimum. Nevertheless, this can be overcome by using high mutation rates as described below.
- (2) *Crossover.* Regarding the crossover operator, parameterized uniform crossovers (Spears and Dejong, 1991) are used as opposed to the traditional one-point or two-point crossover. Two individuals are randomly chosen to act as parents. One of the parents is chosen amongst the best individuals in the population (*TOP* in Figure 6), while the other is randomly chosen from the whole current population (including *TOP*). For each gene, a real random number in the interval $[0, 1]$ is

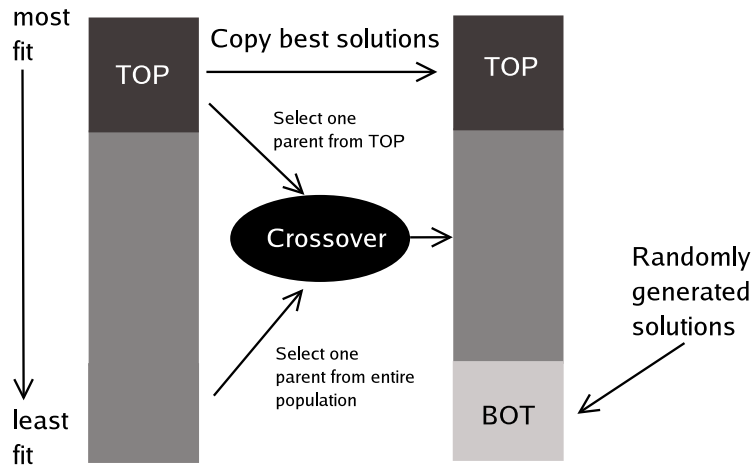


FIGURE 6. Transitional process between consecutive generations. Current population is sorted from best to worst fitness. Top (*TOP*) individuals from current population are copied unchanged to next population. Bottom (*BOT*) individuals in next population are randomly generated. The remaining individuals are generated by applying crossover operator to randomly selected individual from *TOP* individuals of current population and randomly selected individual from the entire current population.

generated. If the random number obtained is smaller than a threshold value, called *crossover probability* ($CProb$), for example 0.7, then the allele of the first parent is used. Otherwise, the allele used is that of the second parent. An example of a crossover outcome is given in Figure 5.

- (3) *Mutation*. In this scheme, mutation is used in a broader sense than usual. The operator we define acts like a mutation operator and its purpose is to prevent premature convergence of the population. Instead of performing gene-by-gene mutation, with very small probability at each generation, we introduce some new individuals into the next generation (see *BOT* in Figure 6). These new individuals (mutants) are randomly generated from the same distribution as the original population and thus, no genetic material of the current population is brought in. This process prevents premature convergence of the population, like in a mutation operator, and leads to a simple statement of convergence, i.e. if a sufficiently large number of generations is carried out, then the entire solution space will be sampled.

Figure 6 depicts the transitional process between two consecutive generations.

The initial population is randomly generated. However, to ensure that some non-delay solutions are included in the initial population, we changed the corresponding delay genes of some chromosomes to zero to make the delay equal to zero, i.e. non-delay. The percentage of non-delay chromosomes in the initial population was set to 25% after some experimentation on a small set of problems.

3.2. Schedule generation procedure. The procedure used to construct parameterized active schedules is based on a scheduling generation scheme that does time incrementing. For each iteration g , there is a scheduling time t_g . The active set comprises all activities

```

procedure CONSTRUCT-SCHEDULES( $E_1$ )
1  Initialize  $g \leftarrow 1$ ;  $t_1 \leftarrow 0$ ;  $A_0 \leftarrow \{0\}$ ;  $\Gamma_0 \leftarrow \{0\}$ ;  $S_0 \leftarrow \{0\}$ ;
2  for  $k \in K$  do initialize  $RD_k(0) \leftarrow R_k$ ;
3  while  $|S_g| < n + 2$  do
4    while  $E_g \neq \emptyset$  do // update  $E_g$ 
5      // select activity with highest priority
6       $j^* \leftarrow \operatorname{argmax}\{PRIORITY_j \mid j \in E_g\}$ ;
7      // compute earliest finish time (in terms of precedence only)
8       $EF_{j^*} \leftarrow \max\{F_i \mid i \in P_{j^*}\} + d_{j^*}$ ;
9      // compute earliest finish time (in terms of precedence and capacity)
10      $F_{j^*} \leftarrow \min\{t \in [EF_{j^*} - d_{j^*}, \infty) \cap \Gamma_g \mid r_{j^*,k} \leq RD_k(\tau),$ 
11        $k \in K \mid r_{j^*,k} > 0, \tau \in [t, t + d_{j^*}]\} + d_{j^*}$ ;
12     Update  $S_g \leftarrow S_{g-1} \cup \{j^*\}$ ;  $\Gamma_g \leftarrow \Gamma_{g-1} \cup \{F_{j^*}\}$ ;
13      $g \leftarrow g + 1$ ; // increment iteration counter
14     Update  $A_g$  and  $E_g$ ;
15     Update  $RD_k(t) \mid t \in [F_{j^*} - d_{j^*}, F_{j^*}]$ ,  $k \in K \mid r_{j^*,k} > 0$ ;
16   end while
17   // determine time associated with activity  $g$ 
18    $t_g \leftarrow \min\{t \in \Gamma_{g-1} \mid t > t_{g-1}\}$ ;
19 end while
end CONSTRUCT-SCHEDULES;

```

FIGURE 7. Pseudo-code of parametrized active schedule construction procedure.

which are active at t_g , i.e

$$A_g = \{j \in J \mid F_j - d_j \leq t_g < F_j\}.$$

The remaining resource capacity of resource k at instant time t_g is given by

$$RD_k(t_g) = R_k(t_g) - \sum_{j \in A_g} r_{j,k}.$$

S_g comprises all activities which have been scheduled up to iteration g , and F_g comprises the finish times of the activities in S_g . Let $delay_g$ be the delay time associated with iteration g , and let E_g comprise all activities which are precedence feasible in the interval $[t_g, t_g + delay_g]$, i.e.

$$E_g = \{j \in J \setminus S_{g-1} \mid F_i \leq t_g + delay_g, \text{ for } i \in P_j\}.$$

The algorithmic description of the scheduling generation scheme used to generate parametrized active schedules is shown in the pseudo-code in Figure 7. The makespan of the solution is given by the maximum of all predecessor activities of activity $n + 1$, i.e. $F_{n+1} = \max\{F_l \mid l \in P_{n+1}\}$.

The basic idea of parameterized active schedules is incorporated in the selection step of the procedure,

$$j^* = \operatorname{argmax}_{j \in E_g} \{PRIORITY_j\}.$$

The set E_g is responsible for forcing the selection to be made only amongst activities which will have a delay smaller or equal to the maximum allowed delay. The parameters $PRIORITY_j$ and $delay_g$ (priority of activity j and delays used at each g) are supplied by the genetic algorithm.

For $nGen$ generations this procedure generates

$$nCrom + [nCrom \times (1 - TOP)] \times (nGen - 1)$$

parameterized active schedules, where $nCrom$ is the number of chromosomes and TOP is the percentage of previous population copied to the next generation.

4. COMPUTATIONAL EXPERIMENTS

This section presents results of the computational experiments done with the algorithm proposed in this paper. We call this algorithm *GAPS* (Genetic Algorithm for Project Scheduling). The algorithm was implemented in Visual Basic 6.0 and the tests were carried out on a computer with a 1.333 GHz AMD Thunderbird CPU on the MS Windows Me operating system.

To illustrate its effectiveness, we consider a total of 1560 instances from three classes of standard RCPSP test problems: J30 (480 instances each with 30 activities), J60 (480 instances each with 60 activities), and J120 (600 instances each with 120 activities). All problem instances require four resource types. Instance details are described by Kolisch et al. (1995). The modified makespan with distance $L = 2$ was used because it gave the best results in a small pilot test.

The proposed algorithm is compared with the following algorithms:

- (1) Local search-oriented approaches:
 - Fleszar and Hindi (2004)
 - Palpant et al. (2004)
- (2) Population-based approaches:
 - Debels et al. (2006)
 - Valls et al. (2004)
- (3) Problem and heuristic space:
 - Leon and Ramamoorthy (1995)
- (4) Priority-rule based sampling methods:
 - Tormos and Lova (2003) – sampling LFT, forward-backward improvement (FBI)
 - Schirmer and Riesenber (1998)
 - Kolisch and Drexl (1996)
 - Kolisch (1996b) – single pass LFT (serial)
 - Kolisch (1996b) – single pass LFT (parallel)
 - Kolisch (1996a,b) – single pass WCS
 - Kolisch (1995) – random (serial)
 - Kolisch (1995) – random (parallel)
- (5) Genetic algorithms:
 - Valls et al. (2005) – GA – forward-backward improvement (FBI)
 - Debels and Vanhoucke (2005) – GA – DBH
 - Valls et al. (2003) – GA – hybrid, forward-backward improvement (FBI)
 - Kochetov and Stolyar (2003) – GA, tabu search, path-relinking
 - Hartmann (2002) – GA self adapting
 - Hartmann (1998) – GA activity list
 - Hartmann (1998) – GA random key
 - Hartmann (1998) – GA priority rule

- (6) Simulated annealing:
 - Bouleimen and Lecocq (2003)
- (7) Tabu search:
 - Nonobe and Ibaraki (2002)
 - Baar et al. (1998)
- (8) Other type heuristics:
 - Möhring et al. (2003) – Lagrangian Relaxation Heuristic

4.1. GA configuration. The present state-of-the-art theory on genetic algorithms provides little insight on how to configure them. In our past experience with genetic algorithms based on the same evolutionary strategy (see e.g. Gonçalves and Almeida (2002), Gonçalves et al. (2005), Gonçalves and Resende (2004), Buriol et al. (2005), and Gonçalves (2006)), we obtained good results with values of *TOP*, *BOT*, and crossover probability (*CProb*) in the following intervals:

Parameter	Interval
<i>TOP</i>	0.10 – 0.20
<i>BOT</i>	0.15 – 0.30
<i>CProb</i>	0.70 – 0.80

For the population size, we obtained good results by indexing it to the size of the problem, i.e. use small size populations for small problems and larger populations for larger problems. Having this past experience in mind and to obtain a reasonable configuration, we conducted a small pilot experiment with combinations of the following values $TOP \in \{0.10, 0.15, 0.20\}$, $BOT \in \{0.15, 0.20, 0.25, 0.30\}$, and $CProb \in \{0.70, 0.75, 0.80\}$. We tried population sizes with 1, 2, 5, and 10 times the number of activities in the project.

The following configuration was held constant for all experiments and all problem instances:

Population Size	$5 \times$ number of activities in the problem
<i>CProb</i>	0.7
<i>TOP</i>	The top 15% from the previous population chromosomes are copied to the next generation.
<i>BOT</i>	The bottom 20% of the population chromosomes are replaced with randomly generated chromosomes.
Fitness	Modified makespan ($L = 2$) (to minimize)
Stopping Criterion	250 generations

The experimental results demonstrate that this configuration provides high-quality solutions and that it is robust.

4.2. Experimental results. Table 1 shows the CPU time (in seconds) spent for 250 generations of the genetic algorithm. Tables 2 and 3 (for algorithms reporting the number of

TABLE 1. CPU time for 250 generations.

Instance class	J30	J60	J120
Average CPU time	5.02s	20.11s	112.46s

schedules generated) and Table 4 (for algorithms not reporting the number of schedules generated) summarize the average percentage deviation from the optimal makespan D_{OPT} for instance set J30. GAPS obtained $D_{OPT} = 0.01$. The number of instances for which our algorithm obtains the optimal solution is 477, i.e. for 99.74% of the instances. The number of generated schedules was 31,773 ($= 5 \times 30 + 249 \times INT(0.85 \times 5 \times 30)$). GAPS ranks first for 1,000 and 5,000 schedules and ranks second for 50,000 schedules. We use only 31,773 schedules which is the number that corresponds to 250 generations.

Tables 5 and 6 (for algorithms reporting the number of schedules generated) and Table 7 (for algorithms not reporting the number of schedules generated) summarize the average percentage deviation from the well-known critical path-based lower bound (D_{LB}) for instance set J60. These lower bound values are reported by Stinson et al. (1978). For the J60 instances, GAPS obtained $D_{LB} = 10.67$. The number of generated schedules was 63,546 ($= 5 \times 60 + 249 \times INT(0.85 \times 5 \times 60)$). GAPS outperformed all other heuristics for 50,000 schedules, ranks second for 5,000, and ranks fourth for 1,000 schedules.

Tables 8 and 9 (for algorithms reporting the number of schedules generated) and Table 10 (for algorithms not reporting the number of schedules generated) summarize the average percentage deviation from the well-known critical path-based lower bound (DLB) for instance set J120. This lower bound values are reported by Stinson et al. (1978). For the J120 instances, GAPS obtained $DLB = 31.20$. The number of generated schedules was 127,341 ($= 5 \times 120 + 249 \times INT(0.85 \times 5 \times 60)$). GAPS ranks seventh for 1,000 schedules and ranks third for 5,000 and 50,000 schedules (we use only 49,464 which is the number that corresponds to 250 generations).

From the above results it is clear that no algorithm dominates GAPS. The approach of Debels et al. (2006) is the one that seems to have similar performance. When 50,000 or more schedules are allowed, GAPS seems to have the best performance. Given that GAPS uses an evolutionary strategy that depends on the number of generations, it is not surprising that, for problems with large number of activities, it does not perform so well when only a small number of schedules generated is allowed.

To demonstrate the contribution of the parameterized active generation scheme to the overall performance of GAPS we also run GAPS using the parallel and serial schedule generation schemes (SGSs). Table 11 presents the results. It is clear that in all cases the parameterized active generation scheme performs considerably better.

Additionally, to demonstrate the contribution of the parameter L in the overall performance of GAPS, we ran GAPS using $L = 0, 1, 2, 3$. Table 12 presents the results. It is clear that the results obtained by using the modified makespan outperform, in all cases, the

TABLE 2. Average percent deviations from optimal makespan – ProGen set $J = 30$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, and the average percent deviations for 1000, 5000, and 50,000 schedules. The table also lists the number of GAPS generations as well as the number of schedules generated by GAPS. Continues on Table 3.

Algorithm	SGS	Reference	Maximum number of schedules Number of generations of GAPS (Number of schedules of GAPS)		
			1000 7 (915)	5000 39 (4976)	50000 250 (31773)
GA, TS – path relinking	Serial	Kochetov and Stolyar (2003)	0.10	0.04	0.00
<i>GAPS</i>	Param. Active	This paper	0.06	0.02	0.01
Scatter Search – FBI	Serial	Debels et al. (2006)	0.27	0.11	0.01
GA – DBH	Serial	Debels and Vanhoucke (2005)	0.15	0.04	0.02
GA – hybrid, FBI	Serial	Valls et al. (2003)	0.27	0.06	0.02
GA – FBI	Serial	Valls et al. (2005)	0.34	0.20	0.02
Sampling – LFT, FBI	Both	Tormos and Lova (2003)	0.25	0.13	0.05
TS – activity list	Serial	Nonobe and Ibaraki (2002)	0.46	0.16	0.05
GA – self adapting	Serial	Hartmann (2002)	0.38	0.22	0.08
GA – activity list	Serial	Hartmann (1998)	0.54	0.25	0.08
SA – activity list	Serial	Bouleimen and Lecocq (2003)	0.38	0.23	-

TABLE 3. Average percent deviations from optimal makespan – ProGen set $J = 30$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, and the average percent deviations for 1000, 5000, and 50,000 schedules. The table also lists the number of GAPS generations as well as the number of schedules generated by GAPS. Continued from Table 2.

Algorithm	SGS	Reference	Maximum number of schedules Number of generations of GAPS (Number of schedules of GAPS)		
			1000 7 (915)	5000 39 (4976)	50000 250 (31773)
Sampling – adaptative	Both	Schirmer and Riesenber (1998)	0.65	0.44	-
TS – schedule scheme	Related	Baar et al. (1998)	0.86	0.44	-
Sampling – adaptative	Both	Kolisch and Drexel (1996)	0.74	0.53	-
Sampling – LFT	Serial	Kolisch (1996b)	0.83	0.53	0.27
GA – random key	Serial	Hartmann (1998)	1.03	0.56	0.23
Sampling – random	Serial	Kolisch (1995)	1.44	1.00	0.51
GA – priority rule	Serial	Hartmann (1998)	1.38	1.12	0.88
Sampling – WCS	Parallel	Kolisch (1996a,b)	1.40	1.28	-
Sampling – LFT	Parallel	Kolisch (1996b)	1.40	1.29	1.13
Sampling – random	Parallel	Kolisch (1995)	1.77	1.48	1.22
GA – problem space	Mod. Par.	Leon and Ramamoorthy (1995)	2.08	1.59	-

TABLE 4. Average percent deviations from optimal makespan – ProGen set $J = 30$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, the average percent deviation, average and maximum CPU times, and the frequency of the CPU used in the experiments.

Algorithm	SGS	Reference	Avg. % deviation	CPU time		CPU freq.
				Avg.	Max.	
Decomp. & local opt.	Serial	Palpant et al. (2004)	0.00	10.26s	123.0s	2.3 Ghz
VNS-activity list	Serial	Fleszar and Hindi (2004)	0.01	0.64s	5.9s	1.0 Ghz
Local search – critical	Serial	Valls et al. (2003)	0.06	1.61s	6.2s	400 MHz
Population-based	Serial	Valls et al. (2004)	0.10	1.16s	5.5s	400 MHz

TABLE 5. Average percent deviations from critical path lower bound – ProGen set $J = 60$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, and the average percent deviations for 1000, 5000, 50,000, and 63,546 schedules. The table also lists the number of GAPS generations as well as the number of schedules generated by GAPS. Continues on Table 6.

Algorithm	SGS	Reference	Maximum number of schedules Number of generations of GAPS (Number of schedules of GAPS)			
			1000 3 (808)	5000 19 (4872)	50000 196 (49830)	63546 250 (63546)
<i>GAPS</i>	Param. Active	This paper	11.72	11.04	10.67	10.67
GA - DBH	Serial	Debels and Vanhoucke (2005)	11.45	10.95	10.68	-
Scatter Search - FBI	Serial	Debels et al. (2006)	11.73	11.10	10.71	-
GA – hybrid, FBI	Serial	Valls et al. (2003)	11.56	11.10	10.73	-
GA, TS – path relinking	Both	Kochetov and Stolyar (2003)	11.71	11.17	10.74	-
GA - FBI	Serial	Valls et al. (2005)	12.21	11.27	10.74	-
GA – self adapting	Both	Hartmann (2002)	12.21	11.70	11.21	-
GA - activity list	Serial	Hartmann (1998)	12.68	11.89	11.23	-
Sampling – LFT, FBI	Both	Tormos and Lova (2003)	11.88	11.62	11.36	-
SA - activity list	Serial	Bouleimen and Lecocq (2003)	12.75	11.90	-	-
TS – activity list	Serial	Nonobe and Ibaraki (2002)	12.97	12.18	11.58	-

TABLE 6. Average percent deviations from critical path lower bound – ProGen set $J = 60$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, and the average percent deviations for 1000, 5000, 50,000, and 63,546 schedules. The table also lists the number of GAPS generations as well as the number of schedules generated by GAPS. Continued from Table 5.

Algorithm	SGS	Reference	Maximum number of schedules Number of generations of GAPS (Number of schedules of GAPS)			
			1000 3 (808)	5000 19 (4872)	50000 196 (49830)	63546 250 (63546)
Sampling - adaptative	Both	Schirmer and Riesenber (1998)	12.94	12.58	-	-
Sampling – adaptative	Both	Kolisch and Drexl (1996)	13.51	13.06	-	-
TS - schedule scheme	Related	Baar et al. (1998)	13.80	13.48	-	-
GA – random key	Serial	Hartmann (1998)	14.68	13.32	12.25	-
GA – priority rule	Serial	Hartmann (1998)	13.30	12.74	12.26	-
Sampling - LFT	Parallel	Kolisch (1996b)	13.59	13.23	12.85	-
Sampling – LFT	Serial	Kolisch (1996b)	13.96	13.53	12.97	-
Sampling - random	Parallel	Kolisch (1995)	14.89	14.30	13.66	-
Sampling – WCS	Parallel	Kolisch (1996a,b)	13.66	13.21	-	-
Sampling - random	Serial	Kolisch (1995)	15.94	15.17	14.22	-
GA – problem space	Mod. Par.	Leon and Ramamoorthy (1995)	14.33	13.49	-	-

TABLE 7. Average percent deviations from critical path lower bound – ProGen set $J = 60$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, the average percent deviation, average and maximum CPU times, and the frequency of the CPU used in the experiments.

Algorithm	SGS	Reference	Avg. % deviation	CPU time		CPU freq.
				Avg.	Max.	
Decomp. & local opt.	Serial	Palpant et al. (2004)	10.81	38.8s	223.0s	2.3 Ghz
Population-based	Serial	Valls et al. (2004)	10.89	3.7s	22.6s	400 MHz
Local search – critical	Serial	Valls et al. (2003)	11.45	2.8s	14.6s	400 MHz
Lagrangian Relax. heuristic	Both, Mod. parallel	Möhring et al. (2003)	15.60	6.9s	57s	200 MHz

TABLE 8. Average percent deviations from critical path lower bound – ProGen set $J = 120$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, and the average percent deviations for 1000, 5000, 50,000, 100,000, and 127,341 schedules. The table also lists the number of GAPS generations as well as the number of schedules generated by GAPS. Continues on Table 9.

Algorithm	SGS	Reference	Maximum number of schedules Number of generations of GAPS (Number of schedules of GAPS)				
			1000 2 (1109)	5000 9 (4672)	50000 97 (49464)	100000 196 (99855)	127341 250 (127341)
GA – DBH	Serial	Debels and Vanhoucke (2005)	34.19	32.34	30.82	-	-
GA – hybrid, FBI	Serial	Valls et al. (2003)	34.07	32.54	31.24	-	-
<i>GAPS</i>	Param. Active	This paper	35.87	33.03	31.44	31.32	31.20
Scatter Search – FBI	Serial	Debels et al. (2006)	35.22	33.10	31.57	-	-
GA – FBI	Serial	Valls et al. (2005)	35.39	33.24	31.58	-	-
GA, TS – path relinking	Both	Kochetov and Stolyar (2003)	34.74	33.36	32.06	-	-
GA – self adapting	Both	Hartmann (2002)	37.19	35.39	33.21	-	-
Sampling – LFT, FBI	Both	Tormos and Lova (2003)	35.01	34.41	33.71	-	-
GA – activity list	Serial	Hartmann (1998)	39.37	36.74	34.03	-	-
SA – activity list	Serial	Bouleimen and Lecocq (2003)	42.81	37.68	-	-	-
TS – activity list	Serial	Nonobe and Ibaraki (2002)	40.86	37.88	35.85	-	-

TABLE 9. Average percent deviations from critical path lower bound – ProGen set $J = 120$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, and the average percent deviations for 1000, 5000, 50,000, 100,000, and 127,341 schedules. The table also lists the number of GAPS generations as well as the number of schedules generated by GAPS. Continued from Table 8.

Algorithm	SGS	Reference	Maximum number of schedules Number of generations of GAPS (Number of schedules of GAPS)				
			1000 2 (1109)	5000 9 (4672)	50000 97 (49464)	100000 196 (99855)	127341 250 (127341)
GA – priority rule	Serial	Hartmann (1998)	39.93	38.49	36.51	-	-
Sampling – adaptative	Both	Schirmer and Riesenber (1998)	39.85	38.70	-	-	-
Sampling – LFT	Parallel	Kolisch (1996b)	39.60	38.75	37.74	-	-
Sampling – WCS	Parallel	Kolisch (1996a,b)	39.65	38.77	-	-	-
Sampling – adaptative	Both	Kolisch and Drexl (1996)	41.37	40.45	-	-	-
GA – problem space	Mod. Par.	Leon and Ramamoorthy (1995)	42.91	40.69	-	-	-
GA – random key	Serial	Hartmann (1998)	45.82	42.25	38.83	-	-
Sampling – LFT	Serial	Kolisch (1996b)	42.84	41.84	40.63	-	-
Sampling – random	Parallel	Kolisch (1995)	44.46	43.05	41.44	-	-
Sampling – random	Serial	Kolisch (1995)	49.25	47.61	45.60	-	-

TABLE 10. Average percent deviations from critical path lower bound – ProGen set $J = 120$. For each algorithm, the table lists its schedule generation scheme (SGS), the reference in which the results are published, the average percent deviation, average and maximum CPU times, and the frequency of the CPU used in the experiments.

Algorithm	SGS	Reference	Avg. % deviation	CPU time		CPU freq.
				Avg.	Max.	
Population-based	Serial	Valls et al. (2004)	31.58	59.4s	264.0s	400 MHz
Decomp. & local opt.	Serial	Palpant et al. (2004)	32.41	207.9s	501.0s	2.3 Ghz
Local search - critical	Serial	Valls et al. (2003)	34.53	17.0s	43.9s	400 MHz
Lagrangian Relax. . heuristic	Both, Mod. parallel	Möhring et al. (2003)	36.00	72.9s	654s	200 MHz

TABLE 11. Average percent deviations using the parallel, serial, and parameterized active schedule generation schemes (SGS). For problem set J30 the deviation is from the optimal makespan while for sets J60 and J120 it is from the critical path lower bound. The fitness function used is the modified makespan with $L = 2$.

Prob. set	SGS	Maximum number of schedules						
		1000	5000	31773	50000	63546	100000	127341
J30	Parallel	1.23	1.13	1.11	-	-	-	-
	Serial	0.58	0.20	0.14	-	-	-	-
	Param. Active	0.06	0.02	0.01	-	-	-	-
J60	Parallel	13.71	12.58	-	12.15	12.12	-	-
	Serial	13.68	12.05	-	11.38	11.35	-	-
	Param. Active	11.72	11.04	-	10.67	10.67	-	-
J120	Parallel	40.20	37.93	-	34.09	-	33.57	33.45
	Serial	43.22	39.93	-	34.31	-	33.99	33.92
	Param. Active	35.87	33.03	-	31.44	-	31.32	31.20

TABLE 12. Average percent deviation of GAPS using the modified makespan fitness function with different values of L for increasing number of schedules generated. For problem set J30 the deviation is from the optimal makespan while for sets J60 and J120 it is from the critical path lower bound.

Prob. set	L	Maximum number of schedules						
		1000	5000	31773	50000	63546	100000	127341
J30	0	0.82	0.38	0.29	-	-	-	-
	1	0.76	0.34	0.25	-	-	-	-
	2	0.06	0.02	0.01	-	-	-	-
	3	0.21	0.15	0.07	-	-	-	-
J60	0	13.76	12.63	-	12.45	12.41	-	-
	1	12.89	12.10	-	11.92	11.44	-	-
	2	11.72	11.04	-	10.67	10.67	-	-
	3	11.87	11.25	-	10.89	10.88	-	-
J120	0	41.94	40.45	-	38.81	-	38.29	38.11
	1	39.56	37.45	-	35.41	-	34.95	34.67
	2	35.87	34.53	-	31.44	-	31.32	31.20
	3	37.71	35.99	-	33.71	-	33.22	32.96

ones obtained by using the makespan ($L = 0$). Additionally, the results confirm that GAPS obtains the best results using the modified makespan when $L = 2$.

5. CONCLUSIONS

This paper presents a genetic algorithm for the resource constrained project scheduling problem. The chromosome representation of the problem is based on random keys. The schedules are constructed using a priority rule in which the priorities are defined by the genetic algorithm. Schedules are constructed using a procedure that generates parameterized active schedules.

The approach was tested on a set of 1560 standard instances taken from the literature and compared with 14 other approaches. The algorithm produced good results when compared with other approaches therefore validating the effectiveness of the proposed algorithm.

ACKNOWLEDGEMENTS

We would to thank Prof. Rainer Kolisch of the Technical University of Munich, Germany, for supplying the problem sets data.

REFERENCES

- R. Alvarez-Valdez and J.M. Tamarit. Heuristic algorithms for resource-constrained project scheduling: A review and empirical analysis. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*, pages 113–134. Elsevier, 1989.
- T. Baar, P. Brucker, and S. Knust. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: Advances and trends in local search paradigms for optimization*, pages 1–8. Kluwer, 1998.
- J.C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.
- D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993. Department of Computing Mathematics, University of Cardiff, UK.
- J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.
- F.F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 49:3–13, 1990.
- F.F. Boctor. An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems. *International Journal of Production Research*, 34:2335–2351, 1996.
- K. Bouleimen and H. Lecocq. 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, 2003.
- P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling : Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.
- P. Brucker and S. Knust. A linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research*, 127:355–362, 2000.
- P. Brucker and S. Knust. Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149:302–313, 2003.

- P. Brucker and S. Knust. *Complex scheduling*. Springer, 2006.
- P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288, 1998.
- L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46(1):36–56, 2005.
- D.F. Cooper. Heuristics for scheduling resource-constrained projects: An experimental investigation. *Management Science*, 22:1186–1194, 1976.
- D.F. Cooper. A note on serial and parallel heuristics for resource-constrained project scheduling. *Foundations of Control Engineering*, 2:131–133, 1977.
- E.W. Davis and J.H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21:944–955, 1975.
- D. Debels, B. De Reyck, R. Leus, and M. Vanhoucke. A hybrid scatter search / electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169:638–653, 2006.
- D. Debels and M. Vanhoucke. 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, 2005.
- S. Demasse, C. Artigues, and P. Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal of Computing*, 17:52–65, 2005.
- E. Demeulemeester and W. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43:1485–1492, 1997.
- E. Demeulemeester and W. Herroelen. *Project scheduling – A research handbook*. Kluwer Academic Publishers, 2002.
- K. Fleszar and K.S. Hindi. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155:402–413, 2004.
- M. Gagnon, F.F. Boctor, and G. d’Avignon. A tabu search algorithm for the resource-constrained project scheduling problem. In *Proceedings of Administrative Sciences Association of Canada Annual Conference (ASAC 2004)*, 2004.
- D.E. Goldberg. *Genetic algorithms in search optimization and machine learning*. Addison-Wesley, 1989.
- J.F. Gonçalves and J.R. Almeida. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8:629–642, 2002.
- J.F. Gonçalves, J.J.M. Mendes, and M.G.C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95, 2005.
- J.F. Gonçalves. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 2006. To appear.
- J.F. Gonçalves and N.C. Beirão. Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações. *Revista Associação Portuguesa Investigação Operacional*, 19:123–137, 1999. In Portuguese.
- J.F. Gonçalves and M.G.C. Resende. An evolutionary algorithm for manufacturing cell formation. *Computers & Industrial Engineering*, 47:247–273, 2004.
- S. Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45:279–302, 1998.

- S. Hartmann. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, 49:433–448, 2002.
- S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127:394–407, 2000.
- W. Herroelen, B. De Reyck, and E. Demeulemeeste. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25:279–302, 1998.
- O. Icmeli, S.S. Erenguc, and C.J. Zappe. Project scheduling problems: A survey. *International Journal of Operations & Production Management*, 13:80–91, 1993.
- R. Klein. *Scheduling of resource-constrained projects*. Kluwer, 1999.
- R. Klein and A. Scholl. Progress: Optimally solving the generalized resource-constrained project scheduling problem. Technical report, University of Technology, Darmstadt, 1998a.
- R. Klein and A. Scholl. Scattered branch and bound: An adaptative search strategy applied to resource-constrained project scheduling problem. Technical report, University of Technology, Darmstadt, 1998b.
- Y. Kochetov and A. Stolyar. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*, 2003.
- U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:203–219, 1999.
- R. Kolisch. *Project scheduling under resource constraints: Efficient heuristics for several problem classes*. Physica-Verlag, 1995.
- R. Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14:179–192, 1996a.
- R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisite : Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996b.
- R. Kolisch and A. Drexl. Adaptative search for solving hard project scheduling problems. *Naval Research Logistics*, 43:43–23, 1996.
- R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, *Handbook on recent advances in project scheduling*, pages 147–178. Kluwer, 1999.
- R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 2005. To appear.
- R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *The International Journal of Management Science, Omega*, 29:249–272, 2001.
- R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703, 1995.
- S.R. Lawrence. Resource constrained project scheduling – A computational comparison of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1985.
- J.-K. Lee and Y.-D. Kim. Search heuristics for resource constrained project scheduling. *Journal of the Operational Research Society*, 47:678–689, 1996.

- V.J. Leon and B. Ramamoorthy. Strength and adaptability of problem-space based neighborhoods for resource constrained scheduling. *Operations Research Spektrum*, 17:173–182, 1995.
- J.J.M. Mendes. *Sistema de apoio à decisão para planeamento de sistemas de produção do tipo projecto*. PhD thesis, Departamento de Engenharia Mecânica e Gestão Industrial, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2003. In Portuguese.
- A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.
- R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49:330–350, 2003.
- K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 557–588. Kluwer Academic Publishers, 2002.
- M. Palpant, C. Artigues, and P. Michelon. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131:237–257, 2004.
- E. Pinson, C. Prins, and F. Rullier. Using tabu search for solving the resource constrained project scheduling problem. Technical report, Universite Catholique de l’Ouest, Angers, 1994.
- A. Schirmer and S. Riesenberger. Case-based reasoning and parameterized random sampling for project scheduling. Technical report, University of Kiel, Germany, 1998.
- R. Slowinski, B. Soniewicki, and J. Weglarz. DSS for multiobjective project scheduling. *European Journal of Operational Research*, 79:220–229, 1994.
- W.M. Spears and K.A. Dejong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.
- A. Sprecher. Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46:710–723, 2000.
- J.P. Stinson, E.W. Davis, and B.M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10:252–259, 1978.
- F. Stork and M. Uetz. On the generation of circuits and minimal forbidden sets. *Mathematical Programming*, 102:185–203, 2005.
- P.R. Thomas and S. Salhi. A tabu search approach for the resource constrained project scheduling problem. *Journal of Heuristics*, 4:123–139, 1998.
- P. Tormos and A. Lova. A competitive heuristic solution technique for Resource-Constrained Project Scheduling. *Annals of Operations Research*, 102:65–81, 2001.
- P. Tormos and A. Lova. Integrating heuristics for resource constrained project scheduling: One step forward. Technical report, Department of Statistics and Operations Research, Universidad Politecnica de Valencia, 2003.
- V. Valls, F. Ballestin, and M.S. Quintanilla. A population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 131:305–324, 2004.
- V. Valls, F. Ballestin, and M.S. Quintanilla. Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, 165:375–386, 2005.

V. Valls, J. Ballestin, and M.S. Quintanilla. A hybrid genetic algorithm for the RCPSP. Technical report, Department of Statistics and Operations Research, University of Valencia, 2003.

(Jorge José de Magalhães Mendes) INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO, INSTITUTO POLITÉCNICO DO PORTO, RUA DR. ANTÓNIO BERNARDINO DE ALMEIDA, 431, 4200-072 PORTO, PORTUGAL.

E-mail address: jjm@isep.ipp.pt

(José Fernando Gonçalves) FACULDADE DE ECONOMIA DA UNIVERSIDADE DO PORTO, RUA DR. ROBERTO FRIAS, 4200-464 PORTO, PORTUGAL.

E-mail address: jfgoncal@fep.up.pt

(Mauricio G. C. Resende) ALGORITHMS AND OPTIMIZATION RESEARCH DEPARTMENT, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.

E-mail address: mgcr@research.att.com