# A Random Key Based Genetic Algorithm for the Resource Constrained Project Scheduling Problem*

***Jorge José de Magalhães Mendes***
Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto, Portugal
jjm@isep.ipp.pt

***José Fernando Gonçalves***
Faculdade de Economia da Universidade do Porto
Rua Dr. Roberto Frias
4200-464 Porto, Portugal
jfgoncal@fep.up.pt

***Maurício G.C. Resende***
Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA
mgcr@research.att.com

## 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 computation results validate the effectiveness of the proposed algorithm.

**Keywords**: Project Management, Scheduling, Genetic Algorithms, Random Keys, RCPSP.

* AT&T Labs Research Technical Report TD-6DUK2C, June 30, 2005.

## 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 $J = \{0, 1, \ldots, n, n+1\}$ denote the set of activities to be scheduled and $K = \{1, ..., k\}$ the set of resources. The activities *0* and *n+1* are dummy, have no duration, and represent the initial and final activities. The activities are interrelated by two kinds of constraints:

- First, the precedence constraints, which force each activity $j$ to be scheduled after all predecessor activities, $P_j$, are completed.

- Second, performing the activities requires resources with limited capacities.

While being processed, activity $j$ requires $\boldsymbol{r_{j,k}}$ units of resource type $k \in K$ during every time instant of its non-preemptable duration $\boldsymbol{d_j}$. Resource type $\boldsymbol{k}$ has a limited capacity of $\boldsymbol{R_k}$ at any point in time. The parameters $\boldsymbol{d_j}$, $\boldsymbol{r_{j,k}}$ and $\boldsymbol{R_k}$ are assumed to be non-negative and deterministic; for the project start and end activities we have $\boldsymbol{d_0}$= $\boldsymbol{d_{n+1}}$=0 and $\boldsymbol{r_{0,k}} = \boldsymbol{r_{n+1,k}}$ =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 ($C_{max}$), i.e. the to complete all jobs.

Let $\boldsymbol{F_j}$ represent the finish time of activity $j$. A schedule can be represented by a vector of finish times ($F_{1,}, F_m, \ldots, F_{n+1}$). Let $\boldsymbol{A}(t)$ be the set of activities being processed at time instant $t$.

Figure 1 gives an example of a project comprising $\boldsymbol{n}$ = 6 activities which have to be scheduled subject to 2 renewable resource types with a capacity of 4 and 2 units respectively. A feasible schedule with an optimal makespan of 15 time-periods is represented in Figure 2.

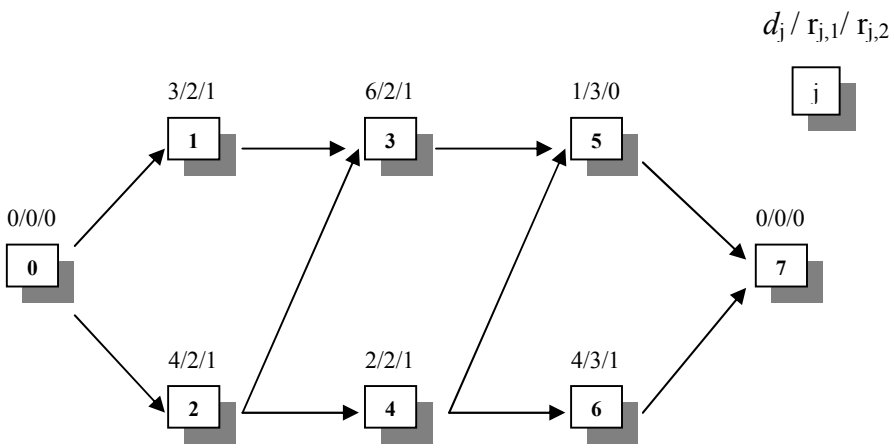


**Figure 1** – Project network example.

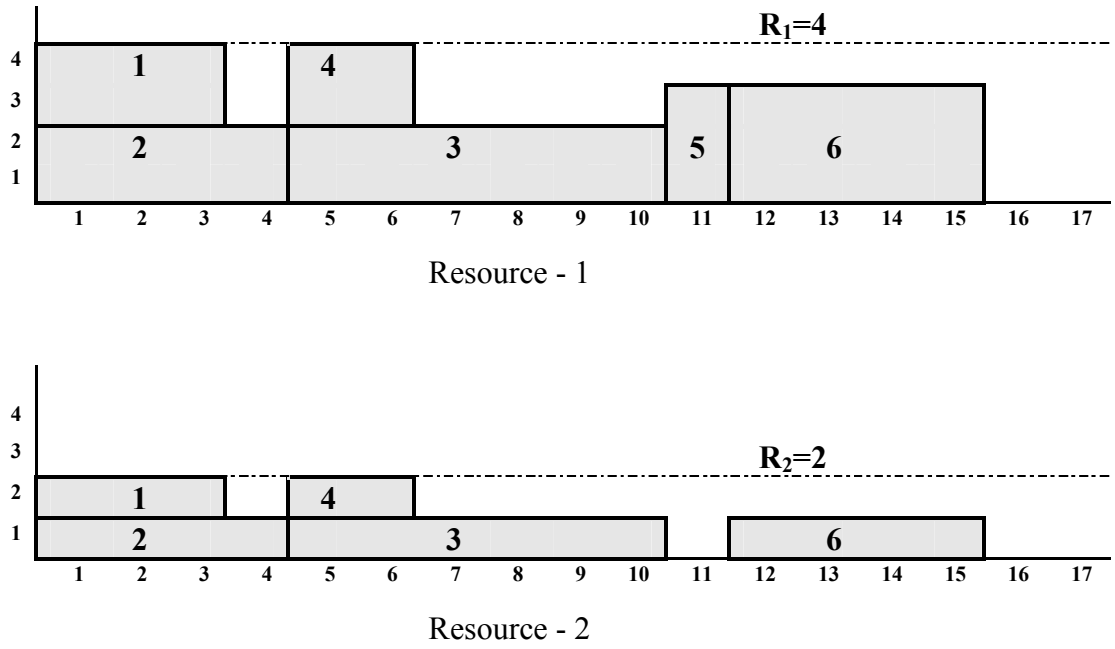


**Figure 2** – Resource utilization example.

The conceptual model of the *RCPSP* was described by Christofides et al. (1987) the following way:

$$\text{Min } F_{n+1} \tag{1}$$

Subject to:

$$F_l \le F_j - d_j \qquad j = 1,...,n+1 \;;\; l \in P_j \tag{2}$$

$$\sum_{j \in A(t)} r_{j,k} \le R_k \qquad k \in K \;;\; t \ge 0 \tag{3}$$

$$F_j \ge 0 \qquad j = 1,...,n+1 \tag{4}$$

The objective function (1) minimizes the finish time of activity $n+1$, and therefore minimizes the *makespan*. Constraints (2) impose the precedence relations between activities and constraints (3) limit the resource demand imposed by the activities being processed at time $t$ to the capacity available. Finally (4) forces the finish times to be non-negative.

A great number of exact methods to solve the *RCPSP* are proposed in the literature. Currently, the most competitive exact algorithms appear to be the ones of Demeulemeester and Herroelen (1997), Sprecher (1997), Brucker et al. (1998), Klein and Scholl (1998 a,b), and Mingozzi et al. (1998).

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, the use of heuristic solution procedures when solving large problem instances is justified.

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 metaheuristic 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 type have been proposed in the literature, e.g., Alvarez-Valdes and Tamarit (1989), Boctor (1990), Cooper (1976, 1977), Davis and Patterson (1975), Lawrence (1985), Kolisch (1996), (Kolisch and Hartmann, 1999), and Tormos and Lova (2001, 2003).

The second class is applied to improve on 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); and Valls et al. (2003, 2004)) , simulated annealing, ( Slowinski et al. (1994); Boctor (1996); Bouleimen and Lecocq (2003); and Valls et al. (2004)), tabu search (Pinson et al. (1996); Baar et al. (1998); Thomas and Salhi (1998); and Nonobe and Ibaraki (2002)) and local search-oriented approaches (Fleszar and Hindi (2000) and Palpant et al. (2004)).

Some surveys are provided by Icmeli et al. (1993), Herroelen et al. (1998), Brucker et al. (1999), Klein (1999), Hartmann and Kolisch (1999,2000), Kolisch and Padman (2001), and Demeulemeester and Herroelen (2002).

In this paper, we present a new genetic algorithm for solving the resource constrained project scheduling problem. The remainder of the paper is organized as follows. Section 2 presents the different classes of schedules. Section 3 presents our approach to solve the resource constrained project scheduling problem: the genetic algorithm and schedule generation procedure. Section 4 reports the computational results and Section 5 presents the conclusions.

## 2. Types of Schedules

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

- **Semi-active schedule**: 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.

- **Active schedule:** 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.

- **Non-delay schedule**: 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.

In what follows we will use parameterized active schedules (Gonçalves and Beirão (1999) and Gonçalves et al. (2005)), in which no resource is kept idle for more than a predefined value if it could start processing some activity. If the predefined value is set to zero, then we obtain a non-delay schedule. The basic concepts of this type of schedule is 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. In order to reduce the solution space and to control the delay times, we used the concept of parameterized active schedules.

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 solution space. A maximum delay time equal to zero is equivalent to restricting the solution space to non-delay schedules.

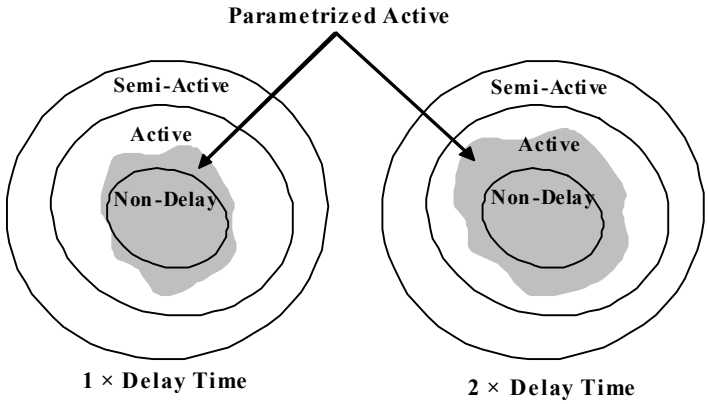


**Figure 3** – Parameterized active schedules.

Section 3.2 presents the pseudo-code to generate parameterized active schedules.

## 3. The New Approach

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

- *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.

- *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.

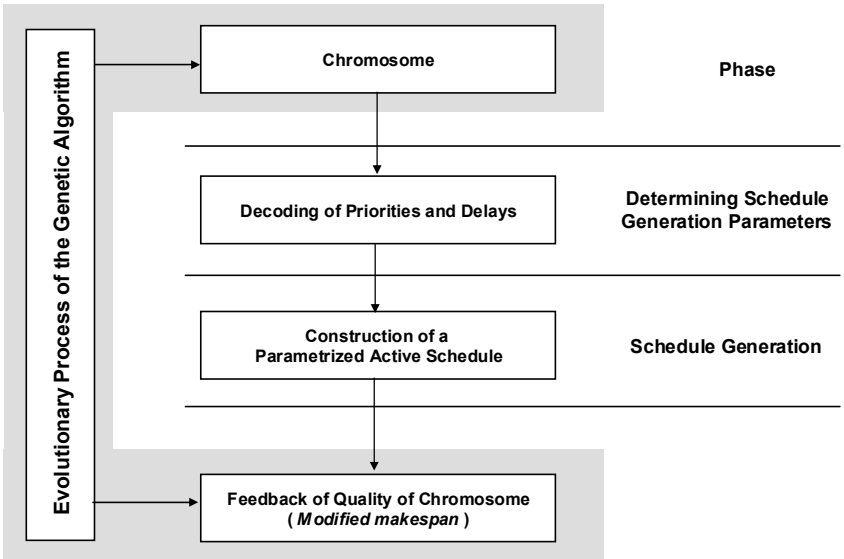


**Figure 4** – Architecture of the new approach.

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). They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection, i.e. *survival of the fittest*, first clearly stated by Charles Darwin in *The Origin of Species*. By mimicking this process, genetic algorithms are able to *evolve* solutions to real world problems, if they have been suitably encoded.

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 run, parents must be *selected* for reproduction, and *recombined* to generate offspring (see Figure 5).

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*). In genetic terminology, 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.

```
Genetic Algorithm
{
        Generate initial population P_t
        Evaluate population P_t
        While stopping criteria not satisfied Repeat
        {
                Select some elements from P_t to copy into P_t+1
                Crossover of some elements of P_t and put into P_t+1
                Mutation of some elements of P_t and put into P_t+1
                Evaluate new population P_t+1
                P_t = P_t+1
        }
}
```

**Figure 5** - Standard genetic algorithm.

### 3.1.1 Chromosome Representation

The genetic algorithm described in this paper uses a random key alphabet **U**(0,1) and an evolutionary strategy identical to the one proposed by Bean (1994). 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 (random numbers). Each solution chromosome is made of ***2n*** genes where ***n*** is the number of activities.

$$\boldsymbol{Chromosome = (gene_1, gene_2, ..., gene_n, gene_{n+1}, ..., gene_{2n})}$$

The first ***n*** genes are used as activities priorities and the genes between ***n+1*** and ***2n*** are used to determine the delay times.

### 3.1.2 Decoding the Priorities of the Activities

In a previous version of the algorithm, the priorities of the activities were given directly by the genetic algorithm, i.e.,

$$\boldsymbol{PRIORITY_j = gene_j} \quad \boldsymbol{j} = 1,...,\boldsymbol{n}.$$

This approach worked quite well. However, we thought that it could be improved mainly if we could somehow provide the genetic algorithm with some information about the structure of the problem.

The priority decoding expression used in this 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 have used the following expression

$$\frac{\boldsymbol{LLP_j}}{\boldsymbol{LCP}},$$

where $\boldsymbol{LLP}_j$ 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 (notice that $0 \le \frac{\boldsymbol{LLP_j}}{\boldsymbol{LCP}} \le 1$ ).

The factor that adjusts the priority to account for capacity is given by

$$\frac{1+\boldsymbol{gene_j}}{2}.$$

The final decoding expression is given by

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

### 3.1.3 Decoding the Delay Times

The genes between ***n+1*** and ***2n*** are used to determine the delay times used when scheduling an activity. The delay time used by each scheduling iteration ***g***, $Delay_g$ , is given by the following expression:

$$Delay_g = gene_{n+g} \times 1.5 \times MaxDur ,$$

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 for schedules having the same makespan, we developed a new measure of fitness that we will call ***Modified makespan***.

The ***Modified makespan*** combines the makespan of the schedule with a measure of the potential for improvement of schedule. This measure of the potential for improvement of schedule will have values in the interval **] 0, 1 [** . The rationale for this new measure is based on the idea that if we have two schedules with the same makespan value, 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 of activity ***j*** to activity ***n+1***, ***Dist(j)*** , is equal to the smallest number of activities in a path connecting them, including activity ***j*** and excluding activity ***n+1***.

The ***Modified makespan*** of distance ***L*** is given by the following expression:

$$F_{n+1} + \frac{\sum_{d=1}^{L} \sum_{i \mid Dist(i)=d} F_i}{\sum_{d=1}^{L} \sum_{i \mid Dist(i)=d} F_{n+1}} . \qquad (5)$$

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 distance of the activities 1-6 is:

| ***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:

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

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. There are many variations of genetic algorithms obtained by altering 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. Mutation allows for random alteration of genetic material. Reproduction and crossover operators tend to increase the quality of the populations and force convergence. Mutation opposes convergence and replaces genetic material lost during reproduction and crossover.

Reproduction is accomplished by first copying some of the best individuals from one generation to the next, in what is called an elitist strategy (Goldberg, 1989). The advantage of an elitist strategy over traditional probabilistic reproduction is that the best solution is monotonically improving from one generation to the next. The potential downside is population convergence to a local minimum. This can, however, be overcome by high mutation rates as described below.

Parameterized uniform crossovers (Spears and DeJong, 1991) are employed in place of the traditional one-point or two-point crossover. After two parents are chosen randomly from the full old population (including chromosomes copied to the next generation in the elitist pass), at each gene we toss a biased coin to select which parent will contribute the allele. Figure 6 presents an example of the crossover operator. It assumes that a coin toss of heads selects the gene from the first parent, a tails chooses the gene from the second parent, and that the probability of tossing a heads is, for example, 0.7.

| | | | | |
|---|---|---|---|---|
| **Chromosome 1** | **0.32** | **0.77** | **0.53** | **0.85** |
| **Chromosome 2** | **0.26** | **0.15** | **0.91** | **0.44** |
| **Rnd Number** | **0.58** | **0.89** | **0.68** | **0.25** |
| **Prob. Cross = 0.7** | **< 0.7** | **> 0.7** | **< 0.7** | **< 0.7** |
| **Offspring Chromosome** | **0.32** | **0.15** | **0.53** | **0.85** |

**Figure 6** - Example of Parameterized Uniform crossover.

Rather than the traditional gene-by-gene mutation with very small probability at each generation, one or more new members of the population are randomly generated from the same distribution as the original population. This process prevents premature convergence of the population, like in a mutation operator, and leads to a simple statement of convergence.

Figure 7 depicts the transitional process between two consecutive generations.

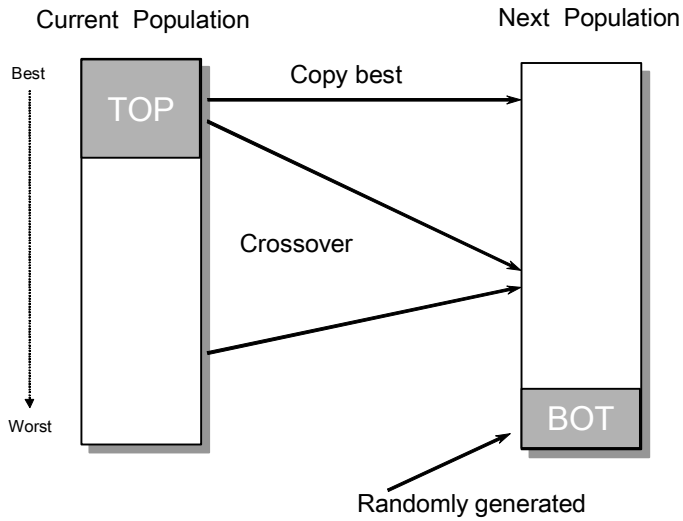


**Figure 7** - Transitional process between consecutive generations.

### 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 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 $FS_g$ 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.

$$E_g = \{ j \in J \setminus S_{g-1} \mid F_i \leq t_g + Delay_g \ \ (i \in P_j) \}.$$

The algorithmic description of the scheduling generation scheme used to generate parameterized active schedules is given by pseudo-code shown in Figure 8.

The makespan of the solution is given by the maximum of all predecessors activities of activity $n+1$, i.e. $F_{n+1} = Max_{l \in P_{n+1}} \{F_l\}$.

---

**Initialization**: $\boldsymbol{g} = 1,\ \boldsymbol{t}_1 = 0,\ \boldsymbol{A}_0 = \{0\},\ \boldsymbol{FS}_0 = \{0\},\ \boldsymbol{S}_0 = \{0\},\ \boldsymbol{RD}_k(0) = \boldsymbol{R}_k\ \ (\boldsymbol{k} \in \boldsymbol{K})$

**while** $|\boldsymbol{S}_g| < \boldsymbol{n} + 2$ **repeat**

{

Update $\boldsymbol{E}_g$

**while** $E_g \neq \{\}$ **repeat**

{

Select activity with highest priority

$$\boldsymbol{j}^* = \underset{\boldsymbol{j} \in \boldsymbol{E}_g}{\operatorname{argmax}} \{\ \boldsymbol{PRIORITY}_j\ \}$$

Calculate earliest finish time (in terms of precedence only)

$$\boldsymbol{EF}_{j^*} = \max_{\boldsymbol{i} \in \boldsymbol{P}_j} \{\boldsymbol{F}_i\} + \boldsymbol{d}_{j^*}$$

Calculate the earliest finish time (in terms of precedence and capacity)

$$\boldsymbol{F}_{j^*} = \min \ \{\ \boldsymbol{t} \in \left[\ \boldsymbol{FMC}_{j^*} - \boldsymbol{d}_{j^*}\ , \infty\ \right] \cap \boldsymbol{FS}_g \ \mid\ \boldsymbol{r}_{j^*,k} \leq \boldsymbol{RD}_k\ (\tau)\ ,\ \boldsymbol{k} \in \boldsymbol{K} \mid \boldsymbol{r}_{j^*,k} > 0\ ,\ \tau \in \left[\boldsymbol{t},\ \boldsymbol{t} + \boldsymbol{d}_{j^*}\right]\ \} + \boldsymbol{d}_{j^*}$$

Update $\boldsymbol{S}_g = \boldsymbol{S}_{g-1} \cup \{\ \boldsymbol{j}^*\ \}\ ,\ \boldsymbol{FS}_g = \boldsymbol{FS}_{g-1} \cup \{\ \boldsymbol{F}_{j^*}\ \}$

Iteration increment: $\boldsymbol{g} = \boldsymbol{g}+1$

Update $\boldsymbol{A}_g,\ \boldsymbol{E}_g,\ \boldsymbol{RD}_k(\boldsymbol{t}) \mid \boldsymbol{t} \in \left[\ \boldsymbol{F}_{j^*} - \boldsymbol{d}_{j^*}\ ,\ \boldsymbol{F}_{j^*}\ \right],\ \boldsymbol{k} \in \boldsymbol{K} \mid \boldsymbol{r}_{j^*,k} > 0$

}

Determine the time associated with activity $\boldsymbol{g}$

$$\boldsymbol{t}_g = \min\ \{\ \boldsymbol{t} \in \boldsymbol{FS}_{g-1} \mid \boldsymbol{t} > \boldsymbol{t}_{g-1}\}$$

}

---

**Figure 8** - Pseudo-code used to construct parameterized active schedules.

For ***nGen*** generations, this procedure generates the following number of parameterized active schedules:

$$\boldsymbol{nCrom} + \ \ (\boldsymbol{nCrom}\ (1 - \boldsymbol{Top})) \times (\boldsymbol{nGen} - 1)$$

where ***nCrom*** is the number of chromosomes and ***Top*** is the percentage of previous population copied for the next generation.

## 4. Computational Experiments

This section presents the computational experiments obtained by the algorithm ***GAPS*** (***G***enetic ***A***lgorithm for ***P***roject ***S***cheduling).

To illustrate the effectiveness of the algorithm described in this paper, 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. Instances 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 algorithm was implemented in Visual Basic 6.0 and the tests were run on a computer with a 1.333 GHz AMD Thunderbird CPU on the MS Windows Me operating system.

The proposed algorithm is compared with the following algorithms:

**Local search-oriented approach**

- Palpant et al. (2004)

**Problem and Heuristic Space**

- Leon, Ramamoorthy (1995)

**Priority Rule Based Sampling Methods**

- Tormos, Lova (2003) – sampling LFT, forward-bakward improvement (FBI)
- Schirmer (1998)
- Kolisch, Drexl (1996)
- Kolisch (1996) – single pass LFT (serial)
- Kolisch (1996) – single pass LFT (parallel)
- Kolisch (1996) – single pass WCS
- Kolisch (1995) – random (serial)
- Kolisch (1995) – random (paralell)

**Genetic Algorithms**

- Valls et al. (2004) – GA - forward-bakward improvement (FBI)
- Valls et al. (2003) – GA – hybrid, forward-bakward improvement (FBI)
- Kochetov, Stolyar (2003) – GA, TS path relinking
- Hartmann (2002) – GA self adapting
- Hartmann (1998) – GA activity list
- Hartmann (1998) – GA random key
- Hartmann (1998) – GA priority rule

**Simulate Annealing**

- Bouleimen, Lecocq (2003)

**Tabu Search**

- Nonobe, Ibaraki (2002)
- Baar (1998)

The experiments were performed using the following configuration for all problems:

| | |
|---|---|
| **Population Size:** | **5 × number of activities in the problem** |
| **Crossover Probability:** | **0.7** |
| **TOP :** | **The top 15% from the previous population chromosomes are copied to the next generation.** |
| **BOT :** | **The bottom 20% of the population chromosomes are replaced with randomly generated chromosomes.** |
| **Fitness:** | **Modified makespan (*L=2*) (to minimize)** |
| **Number of seeds:** | **20** |
| **Stopping Criterion:** | **250 Generations** |

Table 1 summarizes the average deviation percentage from the optimal makespan ($\boldsymbol{D_{OPT}}$), for the instance set J30. The ***GAPS*** algorithm obtained $\boldsymbol{D_{OPT}}$ = 0.01. The number of instances for which our algorithm obtains the optimal solution is 477, i.e., 99,74%. The number of generated schedules was 31898 (5×30 + 249×0.85×5×30).

**Table 1 -** Experimental results for J30.

| ***Algorithm*** | ***Reference*** | $\boldsymbol{D_{OPT}}$ |
|---|---|---|
| GA, TS – path relinking | Kochetov, Stolyar (2003) | 0.00 |
| Decomp. & local opt. | Palpant et al. (2004) | 0.00 |
| ***GAPS*** | ***Mendes, Gonçalves and Resende*** | **0.01** |
| GA – hybrid, FBI | Valls et al. (2003) | 0.02 |
| GA - FBI | Valls et al. (2004a) | 0.02 |
| Population-based | Valls et al. (2004b) | 0.10 |
| Sampling – LFT, FBI | Tormos, Lova (2003) | 0.05 |
| TS – activity list | Nonobe, Ibaraki (2002) | 0.05 |
| GA – self adapting | Hartmann (2002) | 0.22 |
| GA - activity list | Hartmann (1998) | 0.25 |
| SA - activity list | Bouleimen, Lecocq (2003) | 0.23 |
| Sampling - adaptative | Schirmer (1998) | 0.44 |
| TS - schedule scheme | Baar (1998) | 0.44 |
| Sampling – adaptative | Kolisch, Drexl (1996) | 0.52 |
| Single pass/sampling - LFT | Kolisch (1996) | 0.53 |
| GA – random key | Hartmann (1998) | 0.56 |
| Sampling - random | Kolisch (1995) | 1.00 |
| GA – priority rule | Hartmann (1998) | 1.12 |
| Single pass/sampling – WCS | Kolisch (1996) | 1.28 |
| Single pass/ sampling – LFT | Kolisch (1996) | 1.29 |
| Sampling - random | Kolisch (1995) | 1.48 |
| GA – problem space | Leon, Ramamoorthy (1995) | 1.59 |

Tables 2 summarizes the average deviation percentage from the well-known critical path-based lower bound ($\boldsymbol{D_{LB}}$ ) for the instance set J60. This lower bound values are reported by Stinson et al. (1978)). For the J60 instances the ***GAPS*** algorithm obtained $\boldsymbol{D_{LB}}$ = 10.82. The number of generated schedules was 63795 (5×60 + 249×0.85×5×60).

**Table 2 -** Experimental results for J60.

| ***Algorithm*** | ***Reference*** | $D_{LB}$ |
|---|---|---|
| GA – hybrid, FBI | Valls et al. (2003) | 10.73 |
| GA, TS – path relinking | Kochetov, Stolyar (2003) | 10.74 |
| GA - FBI | Valls et al. (2004a) | 10.74 |
| Decomp. & local opt. | Palpant et al. (2004) | 10.81 |
| ***GAPS*** | ***Mendes, Gonçalves and Resende*** | **10.82** |
| Population-based | Valls et al. (2004b) | 10.89 |
| Sampling – LFT, FBI | Tormos, Lova (2003) | 11.36 |
| TS – activity list | Nonobe, Ibaraki (2002) | 11.58 |
| GA – self adapting | Hartmann (2002) | 11.70 |
| GA - activity list | Hartmann (1998) | 11.89 |
| SA - activity list | Bouleimen, Lecocq (2003) | 11.90 |
| Sampling - adaptative | Schirmer (1998) | 12.59 |
| TS - schedule scheme | Baar (1998) | 13.48 |
| Sampling – adaptative | Kolisch, Drexl (1996) | 13.06 |
| Single pass/sampling - LFT | Kolisch (1996) | 13.53 |
| GA – random key | Hartmann (1998) | 13.32 |
| Sampling - random | Kolisch (1995) | 15.17 |
| GA – priority rule | Hartmann (1998) | 12.74 |
| Single pass/sampling – WCS | Kolisch (1996) | 13.21 |
| Single pass/ sampling – LFT | Kolisch (1996) | 13.23 |
| Sampling - random | Kolisch (1995) | 14.30 |
| GA – problem space | Leon, Ramamoorthy (1995) | 13.49 |

Tables 3 summarizes the average deviation percentage from the well-known critical path-based lower bound ($\boldsymbol{D_{LB}}$ ) for the instance set J120. This lower bound values are reported by Stinson et al. (1978)). For the J120 instances the ***GAPS*** algorithm obtained $\boldsymbol{D_{LB}}$ = 31.91. The number of generated schedules was 127590 (5×120 + 249×0.85×5×120).

Table 4 shows the CPU time (in seconds) spent for 250 generations of the genetic algorithm:

## 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.

**Table 3 -** Experimental results for J120.

| ***Algorithm*** | ***Reference*** | ***J120*** $D_{LB}$ |
|---|---|---|
| GA – hybrid, FBI | Valls et al. (2003) | 31.24 |
| Decomp. & local opt. | Palpant et al. (2004) | 31.58 |
| Population-based | Valls et al. (2004b) | 31.58 |
| ***GAPS*** | ***Mendes, Gonçalves and Resende*** | **31.91** |
| GA, TS – path relinking | Kochetov, Stolyar (2003) | 32.06 |
| GA - FBI | Valls et al. (2004a) | 32.81 |
| Sampling – LFT, FBI | Tormos, Lova (2003) | 33.71 |
| TS – activity list | Nonobe, Ibaraki (2002) | 35.85 |
| GA – self adapting | Hartmann (2002) | 35.39 |
| GA - activity list | Hartmann (1998) | 36.74 |
| SA - activity list | Bouleimen, Lecocq (2003) | 37.68 |
| Sampling - adaptative | Schirmer (1998) | 38.70 |
| Sampling – adaptative | Kolisch, Drexl (1996) | 40.45 |
| Single pass/sampling - LFT | Kolisch (1996) | 38.75 |
| GA – random key | Hartmann (1998) | 42.25 |
| Sampling - random | Kolisch (1995) | 47.61 |
| GA – priority rule | Hartmann (1998) | 38.49 |
| Single pass/sampling – WCS | Kolisch (1996) | 38.77 |
| Single pass/ sampling – LFT | Kolisch (1996) | 41.84 |
| Sampling - random | Kolisch (1995) | 43.05 |
| GA – problem space | Leon, Ramamoorthy (1995) | 40.69 |

**Table 4** – CPU time for 250 generations.

| **Classes of instances** | **J30** | **J60** | **J120** |
|---|---|---|---|
| **Average CPU time for 250 generations** | **5.02** s | **20.11** s | **112.46** s |

## Acknowledgements

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

## References

Alvarez-Valdez, R. and Tamarit, J.M. (1989). Heuristic algorithms for resource-constrained project scheduling: A review and empirical analysis. In: Slowinski, R., Weglarz, J. (Eds.), *Advances in Project Scheduling*. Elsevier, Amsterdam, pp. 113-134.

Baar, T., Brucker, P., and Knust, S. (1998). Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In: S.Voss, S. Martello, I. Osman, and C. Roucairol (Eds.), *Meta-heurisitics: Advances and Trends in Local Search Paradigms for Optimization* , Kluwer, Boston, pp. 1-8.

Bean, J.C. (1994). Genetics and random keys for sequencing and optimization, *ORSA Journal on Computing* 6, pp. 154-160.

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A. H.G. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5, pp. 11-24.

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

Bouleimen, K. and Lecocq, H. (1998). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem. Techinical Report, Service de Robotique et Automatisation, Université de Liège.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling : Notation, classification, models, and methods. European Journal of Operational Research 112, pp. 3-412.

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

Christofides, N., Alvarez-Valdés, R., and Tamarit, J.M. (1987). Problem scheduling with resource constraints : A branch and bound approach. *European Journal of Operational Research*, 29, pp. 262-273.

Cooper, D.F. (1976). Heuristics for scheduling resource-constrained projects: An experimental investigation. *Management Science*, 22(11), pp. 1186-1194.

Cooper, D.F. (1977). A note on Serial and Parallel heuristics for resource-constrained project scheduling. *Foundations of Control Engineering*, 2(4), pp. 131-133.

Davis, E.W. and Patterson, J.H. (1975). A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21 (11), pp. 944-955.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science* 43, pp. 1485-1492.

Demeulemeester, E. and Herroelen, W. (2002). *Project scheduling – A research handbook*. Kluwer Academic Publishers, Boston.

Fleszar, K. and Hindi, K. (2000). Solving the resource–constrained project scheduling problem by a variable neighbourhood search. Technical report, Brunel University, Department of Systems Engineering.

Gonçalves, J.F. and Beirão, N. (1999). Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações. Revista Associação Portuguesa de Investigação Operacional, 19, pp. 123-137. (In Portuguese)

Gonçalves, J.F., Mendes, J.M. and Resende, M.G.C., (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*. Vol. 167, pp. 77-95.

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

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

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

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

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

Icmeli, O , Erenguc, S.S., and Zappe, C.J. (1993). Project scheduling problems: A survey, International Journal of Operations & Production management 13 (11), pp. 80-91.

Klein, R. (2000). Bidirectional planning : Improving priority rule-based heuristics for scheduling resource-constrained projects. European Journal of Operational Research 127, pp. 619-638.

Klein, R. (1999). Scheduling of resource-constrained projects. Kluwer, Dordrecht.

Klein, R. and Scholl, A. (1998a) . Progress: Optimally solving the generalized resource-constrained project scheduling problem. Working paper, University of Technology, Darmstadt.

Klein, R. and Scholl, A. (1998b) . Scattered branch and bound: An adaptative search strategy applied to resource-constrained project scheduling problem. Working paper, University of Technology, Darmstadt.

Kolisch., R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. The International Journal of Management Science, Omega 29, pp. 249 –272.

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

Kolisch, R. (1996). Serial and Parallel resource-constrained project scheduling methods revisite : Theory and computation. European Journal of Operational Research 90, pp. 320-333.

Kolisch. R. (1996). Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14(3), pp. 179-192.

Kolisch, R. and Drexl, A. (1996). Adaptative search for solving hard project scheduling problems. Naval Research Logistics 43, pp. 43-23.

Kochetov, Y. and Stolyar, A. (2003). 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*, Russia.

Kolisch, R. and 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, Dordrecht, pp. 147-178.

Lawrence, S.R. (1985). Resource constrained project scheduling – A computational comparison of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.

Mendes, J.M., (2003). Sistema de apoio à decisão para planeamento de sistemas de produção do tipo projecto. Departamento de Engenharia Mecânica e Gestão Industrial. Faculdade de Engenharia da Universidade do Porto. Ph. D. Thesis (In Portuguese).

Mingozzi, A ., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact Algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science* 44, pp. 714-729.

Nonobe, K. and Ibaraki, T. (2002). 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.

Palpant, M., Artigues, C. and Michelon, P. (2004). LSSPER: Solving the resource–constrained project scheduling problem with large neighbourhood search. Annals of Operations Research, 131, pp. 237-257.

Spears, W.M., and Dejong, K. A. (1991). On the virtues of parameterized uniform crossover, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 230-236.

Schirmer, A. and Riesenberg, S. (1998). Case-based reasoning and parameterized random sampling for project scheduling. Technical report, University of Kiel, Germany.

Sprecher, A. (1997). Solving the RCPSP efficiently at modest memory requirements. Working paper, University of Kiel, Germany.

Stinson, J. P., Davis, E. W., and Khumawala, B. M. (1978). Multiple resource-constrained scheduling using branch and bound, *AIIE Transactions* 10, pp. 252-259.

Tormos, P. and Lova, A. (2003). Integrating heuristics for resource constrained project scheduling: One step forward. Technical report, Department of Statistics and Operations Research, Universidad Politecnica de Valencia.

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

Valls, V., Ballestin, F. and Quintanilla, M.S. (2004a). Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, To appear.

Valls, V., Ballestin, F. and Quintanilla, M.S. (2004b). A population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research,* 131, pp. 305-324.