A Genetic Algorithm for the Resource Constrained Multi-Project Scheduling Problem

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

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

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

Abstract

This paper presents a genetic algorithm for the Resource Constrained Multi-Project Scheduling Problem (***RCMPSP***). The chromosome representation of the problem is based on random keys. The schedules are constructed using a heuristic that builds parameterized active schedules based on priorities, delay times, and release dates defined by the genetic algorithm. The approach is tested on a set of randomly generated problems. The computational results validate the effectiveness of the proposed algorithm.

Keywords: Project management, meta-heuristics, genetic algorithm, scheduling.

AT&T Labs Technical Report TD-668LM4, October 29, 2004.

1. Introduction

Project management is a complex decision making process involving the unrelenting pressures of time and cost. A project management problem typically consists of planning and scheduling decisions.

The planning decision is essentially a strategic process wherein planning for requirements of several resource types in every time period of the planning horizon is carried out. Usually, a Gantt chart of projects is developed to generate resource profiles and perform the required leveling of resources by hiring, firing, subcontracting, and allocating overtime resources.

Scheduling involves the allocation of the given resources to projects to determine the start and completion times of the detailed activities. There may be multiple projects contending for limited resources, which makes the solution process more complex. The allocation of scarce resources then becomes a major objective of the problem and several compromises have to be made to solve the problem to the desired level of optimality.

Tools to aid in project scheduling, once activity durations, precedence relationships, and the levels of each resource are known, have existed for some time. Such tools include Gantt charts and the networking tools, such as Critical Path Method (CPM) and the Program Evaluation and Review Technique (PERT). These tools are so well understood that they are incorporated in most, if not all, popular project scheduling software packages.

As valuable as these tools are, they have serious limitations for project activity scheduling in practice. Their use assumes unlimited resources for assignment to project activities exactly when required. Furthermore, they are applied to only one project at a time. In many practical environments where project scheduling is an important activity, resources are constrained in number and more than one project is active at any one time.

2. Problem Description and Conceptual Model

The problem and the conceptual model will be described using Figure 1.The problem consists of a set of $\boldsymbol{I}$ projects, where each project $\boldsymbol{i}$ is composed of activities $\boldsymbol{j} = \{\boldsymbol{N}_{i-1} +1, ..., \boldsymbol{N}_i\}$, where activities $\boldsymbol{N}_{i-1}+1$ and $\boldsymbol{N}_i$ are dummy and represent the initial and final activities of the project $\boldsymbol{i}$. There exists a set of renewable resources types $\boldsymbol{K} = \{\boldsymbol{1}, ..., \boldsymbol{k}\}$. The activities are interrelated by two kinds of constraints. First, the precedence constraints, which force each activity $\boldsymbol{j} \in \boldsymbol{J}$ to be scheduled after all predecessor activities, $\boldsymbol{P_j}$, are completed. Second, processing of the activities is subject to the availability of resources with limited capacities. While being processed, activity $\boldsymbol{j} \in \boldsymbol{J}$ requires $\boldsymbol{r}_{j,k}$ units of resource type $\boldsymbol{k} \in \boldsymbol{K}$ 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 $\boldsymbol{d}_{(N_{i-1}+1)} = \boldsymbol{d}_{N_i} = 0$ and

$r_{N_{i-1}+1,k} = r_{N_i,k} = 0 \ (k \in K)$. Activities ***0*** and ***N+1*** are dummy activities, have no duration and correspond to the start and end of all projects (see Figure 1).

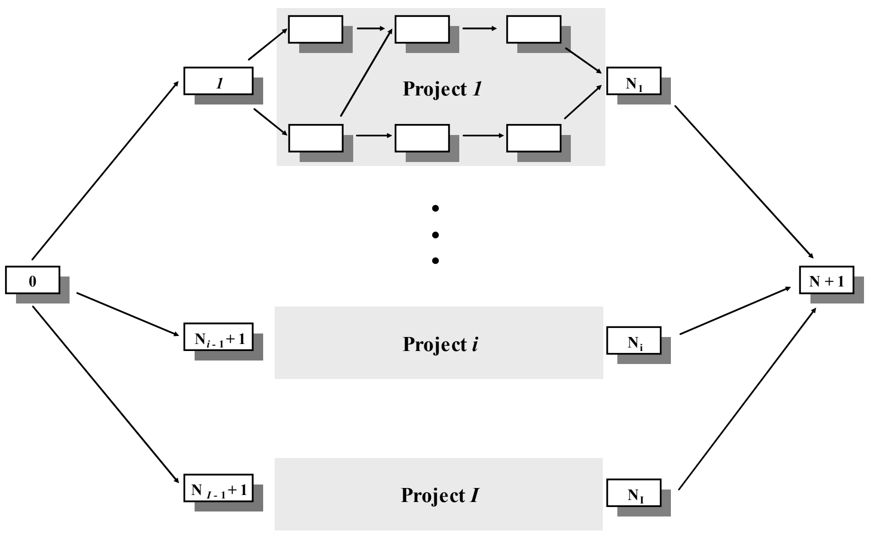


Figure 1 – Project network example.

The Resource Constrained Multi-Project Scheduling Problem (***RCMPSP***) consists in finding a schedule of the activities (i.e. to determine the start and completion times of the detailed activities) taking into account resource availabilities, precedence constraints, while minimizing the performance measure. Let $\boldsymbol{F_j}$ represent the finish time of activity *j*. A schedule can be represented by a vector of finish times ($F_{1,\ldots}F_{\mathrm{m}}, \ldots, F_{\mathrm{n+1}}$). Let $\boldsymbol{A}(t)$ be the set of activities being processed at time instant *t*.

The conceptual model of the ***RCMPSP*** can be described as follows:

$$\textbf{Minimize} \quad \textbf{\textit{Performance Measure}} \ (F_j, \ j = 1, \ldots, N) \qquad (1)$$

Subject to:

$$F_l \le F_j - d_j \qquad j = 1, \ldots, N+1 \ ; \ l \in P_j \qquad (2)$$

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

$$F_j \ge 0 \qquad j = 1, \ldots, N+1 \qquad (4)$$

The objective function (1) seeks to minimize the performance measure. 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, constraints (4) force the finish times to be non-negative.

3. Literature Review

The ***RCMPSP*** is a generalization of the resource constrained project scheduling problem, ***RCPSP***. The ***RCPSP*** has been treated by multiple approaches. In contrast, for the ***RCMPSP*** problem there are only few studies involving the scheduling of several projects.

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. The ***RCMPSP*** problem, as a generalization of the ***RCPSP***, is therefore also NP-hard.

Exact methods to solve the ***RCMPSP*** are proposed in the literature. Currently, the most competitive exact algorithms seem to be ones of Pritsker et al. (1969), Mohanty and Siddiq (1989), Drexl (1991), Deckro et al. (1991) and Vercellis (1994).

Most of the heuristics methods used for solving resource constrained multi-project scheduling problems belong to the class of priority rule based methods. Several approaches of this class have been proposed in the literature, e.g., Fendley (1968), Kurtulus and Davis (1982), Kurtulus and Narula (1985), Dumond and Mabert (1988), Tsubakitani and Deckro (1990), Lawrence and Morton (1993), Wiley et al. (1998), Ash (1999), Lova et al. (2000), Shankar and Nagi (1996), and Mendes (2003).

For the ***RCPSP***, the dominant objective is to minimize the total duration of the project, i.e. the *makespan*, while for the ***RCMPSP***, papers minimizing the average or sum of delays are dominant.

In the next sections, we present our approach to solve the resource constrained multi-project scheduling problem.

4. New Approach

The new approach combines a new measure of performance, a genetic algorithm based on random keys, and a new schedule generation procedure that creates parameterized active schedules (Gonçalves and Beirão (1999) and Gonçalves et al. (2004)). In general terms, the approach innovates in the following two fundamentals areas:

- ***The Model:*** a new measure of performance is developed. It tries to capture reality by integrating due dates, work in process, and stocks. Constraints enforcing the release date concept are also introduced.

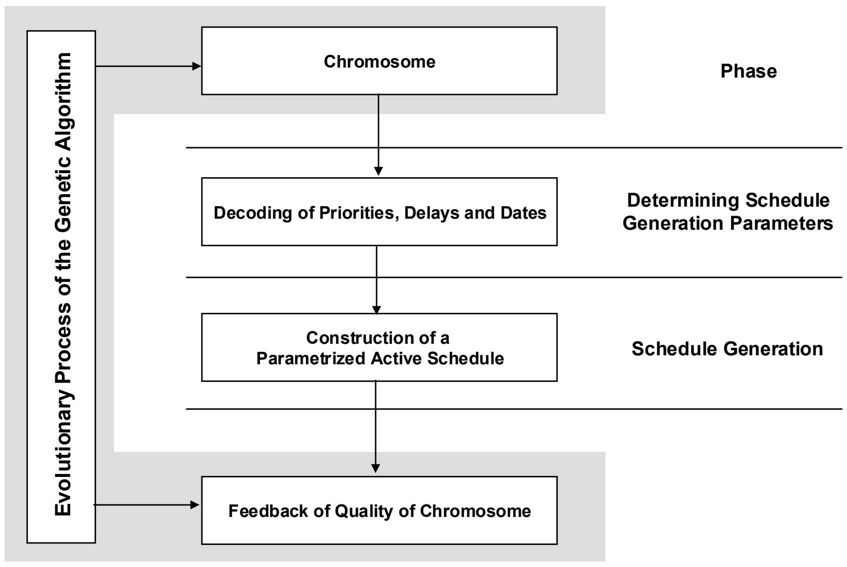
- ***Solution Method:*** considering the difficult to solve real-world problems by exact methods, a new solution approach is developed that combines a genetic algorithm with a schedule generation procedure that creates parameterized active schedules.

The genetic algorithm is responsible for evolving the chromosomes which represent priorities of the activities, delay times, and release dates. For each chromosome, the following two phases are applied:

- *Decoding of priorities, delay times, and release dates*. This phase is responsible for transforming the chromosome supplied by the genetic algorithm into the priorities of the activities, delay times, and release dates.

- *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 quality (performance measure) is fed back to the genetic algorithm. Figure 3 illustrates the sequence of steps applied to each chromosome generated by the genetic algorithm.



Evolutionary Process of the Genetic Algorithm
Chromosome
Phase
Decoding of Priorities, Delays and Dates
Determining Schedule Generation Parameters
Construction of a Parametrized Active Schedule
Schedule Generation
Feedback of Quality of Chromosome

Figure 3 – Architecture of the new approach.

The next sections present the details of the approach.

5. The Model

The conceptual model presented in Section 2 is refined in two ways. A new measure of performance and constraints enforcing the release date concept are introduced. The following sub-sections describe the details of the refinement of the model.

5.1. Measure of Performance

Project management is actually a complex decision making process involving the pressures of due dates (tardiness), stocks (earliness), and work in process (flow time).

The new performance measure incorporates simultaneously three criteria: tardiness, earliness and flow time. The following notation will be used:

$\boldsymbol{D}_\mathrm{i}$	-	Ideal duration for project ***i***.
$\boldsymbol{DD}_\mathrm{i}$	-	Due date for project ***i***.
$\boldsymbol{CD}_\mathrm{i}$	-	Conclusion date for project ***i*** in generated schedule.
$\boldsymbol{BD}_\mathrm{i}$	-	Beginning date for project ***i*** in generated schedule.
$\boldsymbol{T}_\mathrm{i}$	-	Tardiness of project ***i*** = $\boldsymbol{max}\ (\boldsymbol{CD}_i - \boldsymbol{DD}_i, 0)$.
$\boldsymbol{E}_\mathrm{i}$	-	Earliness of project ***i*** = $\boldsymbol{max}\ (\boldsymbol{DD}_i - \boldsymbol{CD}_i, 0)$.
$\boldsymbol{FD}_\mathrm{i}$	-	Flow time deviation for project ***i*** = $\boldsymbol{max}\ (\boldsymbol{CD}_i - \boldsymbol{BD}_i - \boldsymbol{D}_i, 0)$.
$\boldsymbol{CPD}_\mathrm{i}$	-	Critical Path Duration of project ***i***

The new performance measure is defined as follows:

$$a\sum_i T_i^3 + b\sum_i E_i^2 + c\sum_i FD_i^2 \tag{5}$$

where ***a***, ***b,*** and ***c*** are parameters defined by the decision maker.

In a real-world situation, the ideal duration for a project it is not known. To overcome this, we replace

$$c\sum_i FD_i^2$$

by

$$c\sum_i \frac{(CD_i - BD_i)^2}{CPD_i}.$$

5.2. Release Dates

In the conceptual model presented in Section 2, the constraints for the resources are expressed by condition (3). However, there are others types of constraints related with the start of a project which cannot be modeled by condition (3). To be able to model this kind of constraint, we add the following type of constraints to the model

$$F_{N_{i-1}+1} \geq MDL_i \qquad i = 1, \ldots, I$$

where $\boldsymbol{MDL_i}$ represents earliest release date for project ***i***. This constraint is enforced in the model implicitly by assigning a duration $\boldsymbol{DL_i}$ to the initial activity of each project which is larger than $\boldsymbol{MDL_i}$, i.e.,

$$d_{N_{i-1}+1} = DL_i \geq MDL_i \qquad i = 1, \ldots, I$$

6. Schedule Generation Procedure

The schedule generation procedure constructs active schedules. However, the set of active schedules is usually very large and contains many schedules with relatively large delay times, having therefore poor quality in terms of the performance measure. To reduce the solution space, we used the concept of parameterized active schedules introduced by Gonçalves and Beirão (1999) and Gonçalves et al. (2004).

The basic idea of parameterized active schedules consists in controlling the delay times that each activity is allowed. 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 and a maximum delay time equal to infinity is equivalent to allowing active schedules.

Figure 2 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay schedules.

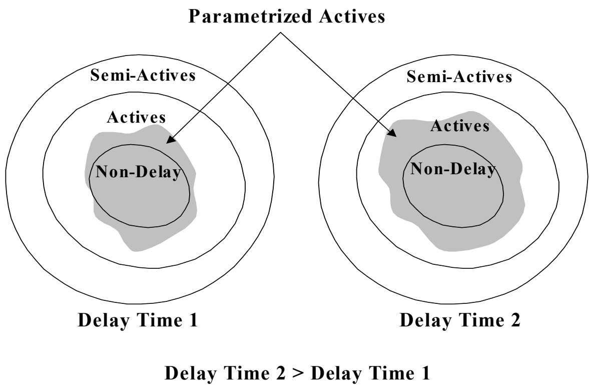


Figure 2 – Parameterized active schedules.

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 $\boldsymbol{t_g}$. The active set comprises all activities which are active at $\boldsymbol{t_g}$, i.e. $\boldsymbol{A_g} = \{ \boldsymbol{j} \in \boldsymbol{J} \mid \boldsymbol{F_j} - \boldsymbol{d_j} \le \boldsymbol{t_g} < \boldsymbol{F_j} \}$. The remaining resource capacity of resource ***k*** at instant time $\boldsymbol{t_g}$ is given by $\boldsymbol{RD_k}(\boldsymbol{t_g}) = \boldsymbol{R_k}(\boldsymbol{t_g}) - \sum_{j \in A_g} r_{j,k}$. $\boldsymbol{S_g}$ comprises all activities which have been scheduled up to iteration ***g***, and $\boldsymbol{F_g}$ comprises the finish times of the activities in $\boldsymbol{S_g}$. Let $\boldsymbol{Delay_g}$ be the delay time associated with iteration ***g***, and let $\boldsymbol{E_g}$ comprise all activities which are precedence feasible in the interval $[\boldsymbol{t_g}, \boldsymbol{t_g} + \boldsymbol{Delay}_g]$, i.e.

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

The algorithmic description of the scheduling generation scheme used to create parameterized active schedules is given by the pseudo-code shown in Figure 4.

```
Initialization: $g = 1,\ t_1 = 0,\ A_0 = \{0\},\ \Gamma_o = \{0\},\ S_0 = \{0\},\ RD_k(0) = R_k\ \ (k \in K)$

while $|S_g| < n+2$ repeat
{
   Update $E_g$
   while $E_g \neq \{\}$ repeat
   {
         Select activity with highest priority
         $j^* = \underset{j \in E_g}{\operatorname{argmax}} \{ PRIORITY_j \}$
         Calculate earliest finish time (in terms of precedence only)
         $EF_{j^*} = \max_{i \in P_j} \{F_i\} + d_{j^*}$
         Calculate the earliest finish time (in terms of precedence and capacity)
         $F_{j^*} = \min \{ t \in [FMC_{j^*} - d_{j^*}, \infty] \cap \Gamma_g \mid r_{j^*,k} \leq RD_k(\tau),$
                 $k \in K \mid r_{j^*,k} > 0,\ \tau \in [t,\ t + d_{j^*}] \} + d_{j^*}$
         Update $S_g = S_{g-1} \cup \{ j^* \}$ , $\Gamma_g = \Gamma_{g-1} \cup \{ F_{j^*} \}$
         Iteration increment: $g = g+1$
         Update $A_g,\ E_g,\ RD_k(t) \mid t \in [F_{j^*} - d_{j^*}, F_{j^*}],\ k \in K \mid r_{j^*,k} > 0$
   }
   Determine the time associated with activity $g$
   $t_g = \min \{ t \in \Gamma_{g-1} \mid t > t_{g-1} \}$
}
```

Figure 4 - Pseudo-code to construct parameterized active schedules.

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

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

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. Figure 5 illustrates the selection step.

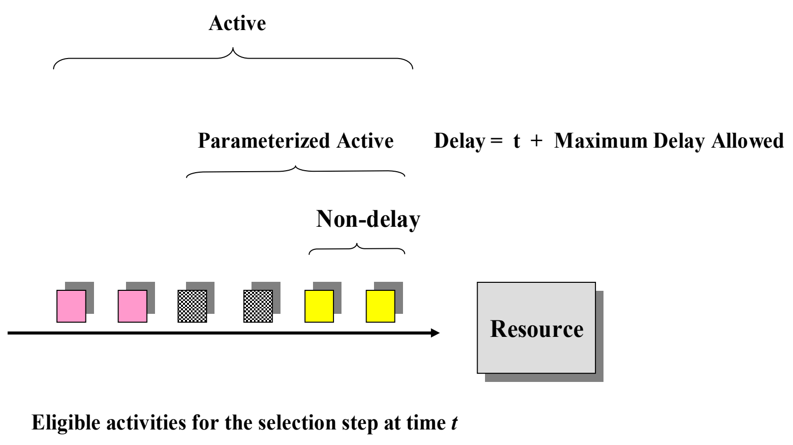

Active
Parameterized Active
Delay = t + Maximum Delay Allowed
Non-delay
Resource
Eligible activities for the selection step at time *t*

Figure 5 - Eligible activities for different types of schedules.

The parameters ***PRIORITY$_j$*** and ***Delay$_g$*** (priority of activity ***j*** and delays used at each ***g***) are supplied by the genetic algorithm. The next section describes the genetic algorithm and shows how it generates the above parameters.

7. 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 are *selected* for reproduction, and *recombined* to generate offspring (see Figure 6).

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.
During the reproductive phase, the individuals 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 some elements of $P_t$ and put into $P_{t+1}$
        Mutate some elements of $P_t$ and put into $P_{t+1}$
        Evaluate new population $P_{t+1}$
        $P_t = P_{t+1}$
    }
}
```

Figure 6 - A standard genetic algorithm.

7.1. Chromosome Representation

The genetic algorithm described in this paper uses a random key alphabet **U**(0,1) similar 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+m*** genes where ***n*** are the number of activities and ***m*** the number of projects.

$$\textit{Chromosome} = (\underbrace{\text{gene}_1, \ldots, \text{gene}_n}_{\text{Priorities}}, \underbrace{\text{gene}_{n+1}, \ldots, \text{gene}_{2n}}_{\text{Delays Times}}, \underbrace{\text{gene}_{2n+1}, \ldots, \text{gene}_{2n+m}}_{\text{Release Dates}})$$

The first ***n*** genes are used to determine the priorities of each activity. The genes between ***n+1*** and ***2n*** are used to determine the delay time used at each of the ***n*** iterations of scheduling procedure. The last ***m*** genes are used to determine the release dates of each of the ***m*** projects.

7.2. Decoding

In this section, we describe how the chromosomes supplied by the genetic algorithm are decoded (transformed) into activity priorities, delays, and release dates. In our approach, we consider the following three solution alternatives:

GaBasic	-	a basic decoding procedure;
GaSlackNd	-	a decoding procedure where the priorities of the activities are static (i.e., the activities priorities are not evolved by the genetic algorithm) and the schedules are non-delay;
GaSlackMod	-	a more sophisticated decoding procedure in which problem specific information is included.

The next sub-section presents the decoding procedures for the activity priorities, delays, and release dates for each of the above solution alternatives.

7.2.1. Decoding of the Activity Priorities

As mentioned in Section 7.1, the first $\boldsymbol{n}$ genes are used to obtain activity priorities. Activity priorities are values between 0 and 1. The higher the value, the higher the priority will be. Below, we present the decoding procedures for the activity priorities according to each of the above proposed solution alternatives.

GaBasic

For this solution alternative, the priority of each activity $\boldsymbol{j}$ is given by the gene value, i.e.

$$\boldsymbol{Priority_j = Gene_j}$$

GaSlackNd

For this solution alternative, the priority of each activity $\boldsymbol{j}$ is given by the normalized slack calculated by the following expression:

$$\boldsymbol{Priority_j = \frac{Slack_j}{MaxSlack}}$$

where

$\boldsymbol{DD_{i \mid j \in i}}$	=	Due date of the project $\boldsymbol{i}$ to which the activity $\boldsymbol{j}$ belongs.
$\boldsymbol{LLP_j}$	=	Longest length path from the beginning of the activity $\boldsymbol{j}$ to the end of the project $\boldsymbol{i}$ to which activity $\boldsymbol{j}$ belongs.
$\boldsymbol{Slack_j}$	=	$\boldsymbol{DD_{i \mid j \in i} - LLP_j}$
$\boldsymbol{MaxSlack}$	=	Maximum Slack for all activities amongst all projects.

GaSlackMod

For this solution alternative, the priority of each activity ***j*** is given by an expression which modifies the normalized slack to produce priority values that are between 70% and 100% of the normalized slack. The priority values are obtained by the following expression:

$$\boldsymbol{Priority}_j = \frac{\boldsymbol{Slack}_j}{\boldsymbol{MaxSlack}} \times \left(0.7 + 0.3 \times \boldsymbol{Gene}_j\right)$$

7.2.2. Decoding of the Delays

The genes between ***n+1*** and ***2n*** are used to determine the delay times, $\boldsymbol{Delay_g}$, used by each scheduling iteration ***g***. Below we present the decoding procedures for the delay times according to each of the above proposed solution alternatives.

GaBasic and **GaSlackMod**
For these solutions alternatives, the delay schedules generated are given by the following decoding expression:

$$\boldsymbol{Delay_g = gene_g \times 1.5 \times MaxDur}$$

where ***MaxDur*** is the maximum duration amongst all activity durations. The factor 1.5 was obtained after experimenting with values between 1.0 and 2.0 in increments of 0.1.

GaSlackNd

For this solution alternative, the delay schedules generated are non-delay therefore the all delays are zero, i.e.

$$\boldsymbol{Delay_g = 0}$$

7.2.3. Decoding of the Release Dates

The last ***m*** genes of each the chromosome, **2*n*+*1*** to ***2n+m* ,** are used to determine the release dates of each project ***i***. All the above solution alternatives (**GaBasic**, **GaSlackNd**, and **GaSlackMod)** use the following decoding expression to obtain the release date of each project ***i***:

$$\boldsymbol{DL_i = MDL_i + Gene_{2n+i} \times \left(DE_i - MDL_i\right) \quad i = 1, \ldots, m}$$

7.3. 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 7 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 (this value is determined empirically). Figure 7 shows one potential crossover outcome:

Coin toss	***H***	***H***	***T***	***H***	***T***
Parent 1	***0.57***	***0.93***	***0.36***	***0.12***	***0.78***
Parent 2	***0.46***	***0.35***	***0.59***	***0.89***	***0.23***
Offspring	***0.57***	***0.93***	***0.59***	***0.12***	***0.23***

Figure 7 - 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 8 depicts the transitional process between two consecutive generations.

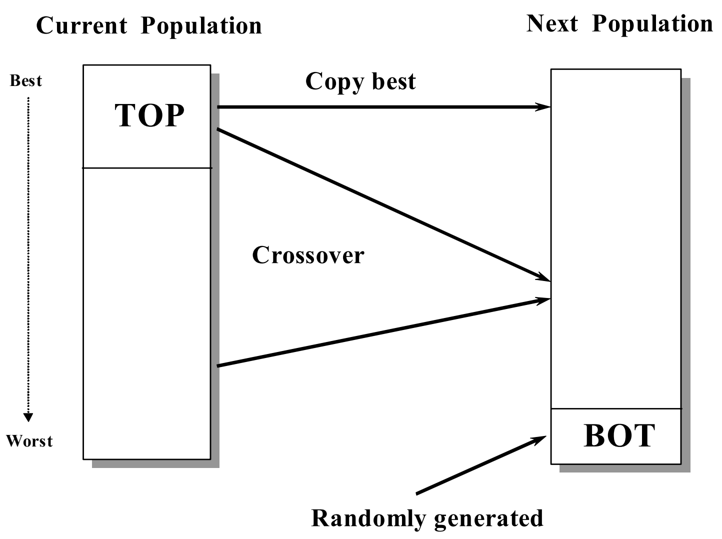


Figure 8- Transitional process between consecutive generations.

In the next section, we describe the problem generator used to produce test problems for the computational experiments.

8. Problem Instance Generator

In the literature we could not find any standard problem instances for the ***RCMPSP***. To overcome this we used the problem generator developed in Mendes (2003). The remainder of this section describes the problem generator.

The problem generator creates problem instances which have as optimal value for the measure of performance described in Section 5.1 the value zero (i.e. tardiness=0, earliness=0, and flow deviation =0).

The problem generator has the following inputs parameters:

- Number of problems to generate;
- Number of projects to include in each problem;
- Average projects number to be simultaneously in execution.

Each multi-project problem instance is generated using the following rules:

1. Each single-project instance to be included in the multi-project instance problem is chosen at random from the 600 single-project instances of type J120 given in Kolisch et al. (1998);

2. The **ideal duration** of each single-project instance is equal to the best known makespan value obtained from the PSPLIB library;

3. The average number of projects to be simultaneously in execution is imposed indirectly by forcing all the single-project instances included in the multi-

project instance to have a **due date** randomly chosen in the interval given by its **critical path duration** and the value given by the **due date upper bound** obtained by the following expression

$$\frac{\text{Sum of } \textbf{ideal duration} \text{ of each single-project instance}}{\text{Number of single-projects to be simultaneously in execution}}$$

4. The **beginning date** of each project is randomly generated in the interval [0 , **problem due date – ideal duration**].

5. The resource capacity in the interval [0 , **due date upper bound**] is calculated by the adding the resource capacities of each single-project instance from its beginning date up until its due date computed in Rule 3. Note that this procedure assigns resource capacities that make it possible to complete each single-project with tardiness=0, earliness=0, and flowtime deviation=0, i.e. it guaranties that the optimal value of measure performance defined in Section 5.1 is zero.

9. Computational experiments

To illustrate the effectiveness of the algorithms described in this paper, we used multi-project instances generated by the problem instance generator described in the previous section.

Four types of multi-project instances where generated, respectively, with 10, 20, 30, and 50 single-project instances. For each problem type, we generated 20 instances. Since each single-project instance has 120 activities, we have that each multi-project instance has 1200, 2400, 3600, and 6000 activities, respectively. Each activity can use up to 4 resources. The average number of projects simultaneously in execution is 3, 6, 9, and 15, respectively, for the problems with 10, 20, 30, and 50 single-projects.

9.1. GA Configuration

The present state-of-the-art theory and practice of genetic algorithms does not provide information on how to configure them. In our past experience with genetic algorithms based on the same evolutionary strategy, see Gonçalves and Almeida (2002), Gonçalves et al. (2004a), and Gonçalves and Resende (2004b), we obtained good results with values of TOP, BOT, and Crossover Probability (CProb) in the following intervals:

Parameter	Interval
TOP	**0.10 – 0.20**
BOT	**0.15 – 0.30**
Crossover Probability (CProb)	**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 in order to obtain a reasonable configuration, we conducted a small pilot study by experimenting the combinations of the following values **TOP**=(0.10, 0.15, 0.20) , **BOT**=(0.15, 0.20, 0.25, 0.30), and **CProb**=(0.70, 0.75, 0.80). We tried population sizes with 0.1 to 2.0 (in intervals of 0.1) times the number of activities in the multi-project problem instance.

For the pilot study, the best results were obtained with the following configuration:

Population Size:	**Min (0.2 $\times$ Number of Activities in the Multi-Project, 250)**
Crossover Probability:	**0.7**
Selection:	**The top 10% from the previous population chromosomes are copied to the next generation.**
Mutation:	**The bottom 20% of the population chromosomes are replaced with randomly generated chromosomes.**
Fitness:	**See Equation (5)**
Stopping Criterion:	**50 Generations**

The above configuration was held constant for all experiments and all problem instances. The following experimental results demonstrate that this configuration provides excellent results in terms of solution quality and that it is very robust.

Table 1 summarizes the experimental results. It lists the fitness, earliness, tardiness, and flow deviation. The columns **Avg**^{1} and **SD**^{1}, **Avg**^{2} and **SD**^{2}, **Avg**^{3} and **SD**^{3}, and **Avg**^{4} and **SD**^{4} represent the average and standard deviation obtained for each of the 20 instances, respectively, for following expressions:

$$\frac{a\sum_{i=1}^{N} T_i^3 + b\sum_{i=1}^{N} E_i^2 + c\sum_{i=1}^{N} FD_i^2}{N}, \quad \frac{\sum_{i=1}^{N} E_i}{N}, \quad \frac{\sum_{i=1}^{N} T_i}{N}, \text{ and } \frac{\sum_{i=1}^{N} FD_i}{N},$$

where N represents the number of projects in each problem.

Algorithm **GaSlackMod** is the best in all aspects relative to the other two. In absolute terms, algorithm **GaSlackMod** obtained, for all instances, earliness, tardiness, and flow deviation close to the optimum value (i.e. zero).

Table 1 – Experimental results.

		Fitness		Earliness		Tardiness		Flow Deviation	
Nº Projects	Algorithm	Avg[1]	SD[1]	Avg[2]	SD[2]	Avg[3]	SD[3]	Avg[4]	SD[4]
10	**GaSlackMod**	**15.55**	**29.04**	**0.00**	**0.00**	**1.17**	**1.44**	**0.22**	**0.43**
	GaSlackND	20.74	21.08	0.00	0.00	2.25	1.35	0.29	0.35
	GaBasic	996.57	1332.56	0.14	0.30	10.15	6.27	5.05	3.22
20	**GaSlackMod**	**4.26**	**10.99**	**0.00**	**0.00**	**0.59**	**0.35**	**0.11**	**0.30**
	GaSlackND	42.94	124.61	0.00	0.01	1.47	1.67	0.33	0.74
	GaBasic	707.30	653.73	0.09	0.11	7.46	5.21	0.49	0.46
30	**GaSlackMod**	**6.59**	**26.42**	**0.00**	**0.00**	**0.64**	**0.75**	**0.04**	**0.08**
	GaSlackSA	11.32	36.23	0.00	0.01	0.87	0.93	0.08	0.15
	GaGen	265.18	253.86	0.01	0.02	4.50	3.65	0.28	0.35
50	**GaSlackMod**	**0.56**	**0.23**	**0.00**	**0.00**	**0.49**	**0.05**	**0.01**	**0.04**
	GaSlackSA	1.36	2.24	0.00	0.00	0.54	0.12	0.03	0.02
	GaGen	268.25	354.95	0.01	0.03	2.91	2.09	0.08	0.10

Table 2 presents the results obtained by algorithm **GaSlackMod** with the performance measure (5), where $\sum_i FD_i^2$ is replaced by $c \sum_i \frac{(CD_i - BD_i)^2}{CPD_i}$.

Table 2 – Experimental results for the algorithm **GaSlackMod.**

Nº Projects	Fitness		Tardiness		Earliness		Flow Time	
	1)	2)	1)	2)	1)	2)	1)	2)
10	**15.55**	**53.45**	**0.00**	**0.00**	**1.17**	**1.11**	**100.15**	**99.12**
20	**4.26**	**84.11**	**0.00**	**0.00**	**0.59**	**0.53**	**95.37**	**95.28**
30	**6.59**	**42.07**	**0.00**	**0.00**	**0.64**	**0.56**	**94.90**	**94.98**
50	**0.56**	**38.94**	**0.00**	**0.00**	**0.49**	**0.49**	**94.79**	**94.79**

1) Average obtained using $\sum_i FD_i^2$, 2)) Average obtained using $c \sum_i \frac{(CD_i - BD_i)^2}{CPD_i}$

The algorithm was implemented in Visual Basic 6.0 and the tests were run on a PC with a 1.33 GHz AMD Thunderbird CPU on the MS Windows Me operating system. The average computational times, in seconds, for each problem instance and for 50 generations are presented in Table 3.

Table 3 – CPU time for 50 generations.

Classes of instances	**10**	**20**	**30**	**50**
Average CPU time for 50 generations	**178 s**	**449 s**	**840 s**	**1860 s**

10. Conclusions

This paper presents a genetic algorithm for the resource constrained multi-project scheduling problem ***RCMPSP***. The chromosome representation of the problem is based on random keys. The schedules are constructed using a heuristic that generates parameterized active schedules based on priorities, delay times, and release dates defined by the genetic algorithm.

The approach was tested on a set problem with 10, 20, 30, and 50 projects (1200, 2400, 3600, and 6000 activities, respectively). The algorithm **GaSlackMod** has better results than any of the other two approaches and obtained values very close to the optimum value (zero) therefore validating the effectiveness of the proposed approach.

References

Ash, R., (1999). Activity Scheduling in the Dynamic, Multi-Project setting: Choosing Heuristics Through Deterministic Simulation. Proceedings of the 1999 Winter Simulation Conference, pp. 937-941, Pheoenix, USA.

Bean, J.C. (1994). Genetics and Random Keys for Sequencing and Optimization, ORSA Journal on Computing 6, pp. 154-160.

Beasley, D., Bull, D.R. and Martin, R.R. (1993). An Overview of Genetic Algorithms: Part 1, Fundamentals, University Computing, Vol. 15, No.2, pp. 58-69, Department of Computing Mathematics, University of Cardiff, UK.

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

Deckro, R.F., Winkofsky, E.P., Hebert, J.E., Gagnon, R., (1991). A Decomposition Approach to multi-project scheduling. European Journal of Operational Reserach, Vol. 51, pp. 110-118.

Dumond, J. and Mabert, V.A. (1988). Evaluating Project Scheduling And Due Date Assignment Procedures: An Experimental Analysis. *Management Science*. v. 34, n. 1, pp. 101-118.

Drexl, A., (1991). Scheduling of project networks by job assignment. Management Science, Vol. 37, No. 12, pp. 1590-1602.

Fendley, L.G., (1968). Towards the Development of a Complete Multiproject Scheduling System. Journal of Industrial Engineering, pp. 505-515.

Goldberg, D.E., (1989). Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley.

Gonçalves, J.F. and Beirão, N.C., (1999). Um Algoritmo Genético Baseado em Chaves Aleatórias para Sequenciamento de Operações. Revista Associação Portuguesa Investigação Operacional, Vol. 19, pp. 123-137 (In Portuguese).

Gonçalves, J.F., Mendes, J.M., Resende M.C.G. (2004). Ahybrid gnetic algorithm for the job shop scheduling problem. European Journal of Operational Research. To appear.

Kolisch, R. Schwindt, Sprecher, A. (1998). Benchmark instances for scheduling problems. In J.Weglarz, ed. Handbook on recent advances in project scheduling, Kluwer, Amsterdam, pp. 197-212.

Kurtulus, I.S., and Davis, E.W., (1982). Multi-project scheduling: Categorization of heuristic rules performance. Management Science 28 (2), pp. 161-172.

Kurtulus, I.S., Narula, S.C., (1985). Multi-project scheduling: Analysis of project performance. IIE Transactions 17 (1), pp. 58-66.

Lawrence, S.R., and Morton, T.E., (1993). Resource-constrained multi-project scheduling with tardy costs: Comparing myopic bottleneck and resource pricing heuristics. European Journal of Operational Research Vol. 64, pp. 168-187.

Lova, A., Maroto, C., Tormos, P., (2000). A multicriteria heuristic method to improve resource allocation in multiproject scheduling. European Journal of Operational Research, Vol. 127, pp. 408-424.

Mendes, J. J.M. (2003). "Sistema de Apoio à Decisão para Planeamento de Sistemas de Produção do Tipo Projecto". Ph. D. Thesis. Departamento de Engenharia Mecânica e Gestão Industrial, Faculdade de Engenharia da Universidade do Porto, Portugal (In Portuguese).

Mohanthy, R.P. and Siddiq, M.K. (1989). Multiple Projects Multiple Resources-Constrained Scheduling: Some Studies. International Journal of Production Research. Vol. 27, n. 2, pp. 261-280.

Shankar, V., and Nagi, R. (1996). A flexible optimization approach to multi-resource, multi-project planning and scheduling. Proceedings of 5th Industrial Engineering Research Conference, Minneapolis, May, USA.

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.

Pritsker, A., Allan, B., Watters, L.J., Wolfe, P.M., (1969). Multiproject scheduling with limited resources: A zero-one programming approach. Management Science, 16(1): pp. 93-108.

Tsubakitani, S. and Deckro, R.F. (1990). A Heuristic For Multi-Project Scheduling With Limited Resources In The Housing Industry. European Journal of Operational Research. v. 49. pp. 80-91.

Vercellis, C. (1994). Constrained multi-Project planning problems: a Lagrangean decomposition approach, European Journal of Operational Research 78, pp. 267-275.

Wiley, V.D., Deckro, R.F., Jackson, J.A, (1998). Optimization analysis for design and planning of multi-project programs. European Journal of Operational Research, Vol. 107, pp. 492-506.