A GENETIC ALGORITHM FOR THE RESOURCE CONSTRAINED MULTI-PROJECT SCHEDULING PROBLEM

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

ABSTRACT. This paper presents a genetic algorithm (GA) 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.

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 near-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 networking tools, such as the *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.

In this paper, we present a new genetic algorithm (GA) approach to solve the *Resource Constrained Multi-Project Scheduling Problem* (RCMPSP). The remainder of the paper is organized as follows. Section 2 describes the problem and presents the conceptual model and Section 3 presents a literature review. Section 4 describes our approach to solve the RCMPSP and Section 5 introduces the model used. Section 6 presents a newly developed

Date: October 29, 2004. Revised January 20, 2006 and July 26, 2006.
Key words and phrases. Project management, metaheuristics, genetic algorithm, scheduling.
AT&T Labs Research Technical Report: TD-668LM4.

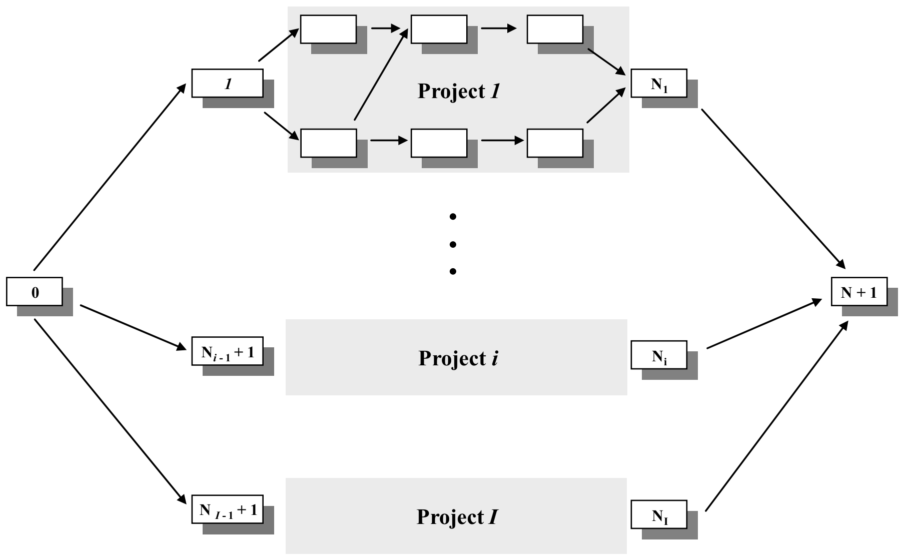


FIGURE 1. Multi-project network example. Artificial (or dummy) activities mark the start and end of each project as well as of the multi-project.

schedule generation procedure and Section 7 describes the genetic algorithm. Section 8 details the problem instance generator and Section 9 reports the computational experiments. Concluding remarks are make in Section 10, along with a discussion about further research.

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 I projects, where each project $i \in I$ is composed of activities $j = \{N_{i-1}+1, \dots, N_i\}$, where activities $N_{i-1}+1$ and N_i are dummy and represent the initial and final activities of project i. $\mathcal{J}$ is the set of activities. There exists a set of renewable resources types $\mathcal{K} = \{1, \dots, k\}$. The activities are interrelated by two kinds of constraints. First, the precedence constraints, which force each activity $j \in \mathcal{J}$ to be scheduled after all predecessor activities, $\mathcal{P}_j$, are completed. Second, processing of the activities is subject to the availability of resources with limited capacities. While being processed, activity $j \in \mathcal{J}$ requires $r_{j,k}$ units of resource type $k \in \mathcal{K}$ during every time instant of its non-preemptable duration d_j. Resource type $k \in \mathcal{K}$ has a limited availability of R_k at any point in time. Parameters d_j, $r_{j,k}$, and R_k are assumed to be non-negative and deterministic. For start and end activities of project i, we have, for all $i \in I$, that

$$d_{(N_{i-1}+1)} = d_{N_i} = 0 \text{ and } r_{N_{i-1}+1,k} = r_{N_i,k} = 0 \ (\forall k \in \mathcal{K}).$$

Activities 0 and $N+1$ are dummy activities, have no duration, and correspond to the start and end of all projects (see Figure 1).

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 and precedence constraints, while minimizing some performance measure. Let F_j represent the finish time of activity $j \in \mathcal{J}$. A schedule can be represented by a vector of finish times $(F_1, \ldots, F_{N+1})$. Let $\mathcal{A}(t)$ be the set of activities being processed at time instant t. The conceptual model of the RCMPSP can be described as

$$\text{Minimize } \textit{performance measure}\,(F_1, \ldots, F_N) \tag{1}$$

Subject to:

$$F_l \leq F_j - d_j, \quad j = 1, \ldots, N+1 \;;\; l \in \mathcal{P}_j, \tag{2}$$

$$\sum_{j \in \mathcal{A}(t)} r_{j,k} \leq R_k, \quad k \in K \;;\; t \geq 0, \tag{3}$$

$$F_j \geq 0, \quad j = 1, \ldots, N+1. \tag{4}$$

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 available capacity. 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, 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 (Garey and Johnson, 1979). The RCMPSP, as a generalization of the RCPSP, is therefore also NP-hard.

Exact methods to solve the RCMPSP are proposed in the literature. The pioneering work of multi-project scheduling by Pritsker et al. (1969) proposed a zero-one programming approach. Mohanthy and Siddiq (1989) studied the problem of assigning due dates to the projects in a multi-project environment. That study presents an integer programming model and simulation mechanism. The integer program generates the schedules. The simulation allows testing some heuristic rules and the system chooses the best schedule. Drexl (1991) considered a non-preemptive variant of the resource constrained assignment problem using a hybrid branch and bound / dynamic programming algorithm with a Monte Carlo-type upper bounding heuristic. Deckro et al. (1991) formulated the multi-project scheduling problem as a block angular general integer programming model and employed a decomposition approach to solve large problems. Vercellis (1994) describes a Lagrangean decomposition technique for solving multi-project planning problems with resource constraints and alternative modes of performing each activity in the projects. The decomposition can be useful in several ways, such as providing bounds on the optimum so that the quality of approximate solutions can be evaluated. Furthermore, in the context of branch-and-bound algorithms, it can be used for more effective fathoming of the tree nodes. Finally, in the modeling perspective, the Lagrangean optimal multipliers can provide insights to project managers as prices for assigning the resources to different projects.

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 in this class have been proposed in the literature. For example, Fendley (1968) used multi-projects with three and five projects and considered three efficiency measurements in the computational analysis: project slippage, resource utilization, and in-process inventory. The most important conclusion of Fendley is that the priority rule *Minimum Slack First* (MINSLK) obtained the best efficiency with the three response variables. Kurtulus and Davis (1982) designed multi-project instances whose projects have between 34 and 63 activities and resource requirements for each activity between 2 and 6 units. They show six new priority rules and *Maximum Total Work Content* (MAXTWK) and *Shortest Activity from the Shortest Project* (SASP) were the best algorithms to schedule multi-projects when the objective was to minimize the mean project delays, where the delays were measured in relation to the unconstrained critical path duration.

Kurtulus and Narula (1985) studied penalties due to project delay. They analyze this problem with multi-project instances of three projects in which activities number between 24 and 33 for small-sized problems and between 50 and 66 activities for large-sized problems. The priority rules used in previous papers were modified by adding penalties to the delays. Six penalty functions and four new priority rules based on penalties were analyzed: *Maximum Duration and Penalty*, *Maximum Penalty*, *Maximum Total Duration Penalty*, and simultaneously *Slack and Penalty*. As one of the most important conclusions, the priority rule Maximum Penalty was considered the best algorithm to minimize the sum of the project weight delay.

Dumond and Mabert (1988) studied the problem of assigning due dates to the projects in a multi-project environment. Each project has between 6 and 49 activities with 24 activities on average whose resource requirements were between one and three types of resources simultaneously. In that paper, five resource allocation heuristics and four strategies to assign due dates to the projects were analyzed: *Mean Flow*, *Number of Activities*, *Critical Path Time*, and *Scheduled Finish Time*. The computational results show that the priority rule *First Come First Served* (FCFS) with the strategy *Scheduled Finish Time Due Date* rule was the best algorithm for minimizing the mean completion time, the mean lateness, the standard deviation of lateness and minimizing the total tardiness. Tsubakitani and Deckro (1990) proposed a heuristic for multi-project scheduling with resource constraints using the Kurtulus and Davis (1982) approach to select appropriate heuristic decision rules. They coded the SASP priority rule to schedule multi-projects with more than 50 projects which could have more than 100 activities. The model has an UPDATE routine that allows the project manager to update the projects when they are in execution. Bock and Patterson (1990) designed a computational experiment based on the work of Dumond and Mabert (1988) with three factors: *Due Date Setting Strategy*, *Algorithm based on Priority Rule*, and *Resource Preemption Strategy*. That paper shows that the priority rules FCFS and MINSLK had the best performance, minimizing mean weighted lateness and mean absolute lateness.

Lawrence and Morton (1993) studied the due date setting problem of scheduling multiple resource-constrained projects with the objective of minimizing weighted tardiness costs. They develop an efficient and effective means of generating low cost schedules requiring multiple resources and a *cost-benefit* scheduling policy with resource pricing which balances the marginal cost of delaying the start of an eligible activity with the marginal benefit of such a delay. A central part of this policy is the heuristic estimation of implicit resource prices, which forms the basis for calculating marginal delay costs. The resulting policies are tested against a number of dispatch scheduling rules taken from the literature, and against several new scheduling rules with good results. Shankar and Nagi (1996) proposed a two-level hierarchical approach consisting of the planning and scheduling stages.

The planning stage was formulated as a linear program, which gives the choice of selecting among multiple objective functions. The scheduling stage uses simulated annealing to calculate the solution.

Wiley et al. (1998) developed a method utilizing *Work Breakdown Structure* (WBS) and Dantzig-Wolfe decomposition to generate feasible aggregate level multi-project program plans and schedules. The Dantzig-Wolfe procedure provides a means of generating interim solutions and their appropriate funding profile. The decision maker may then choose any one of these solutions based upon their own experience and risk tolerance. Ozdamar et al. (1998) examined different dispatching rules for the tardiness and the net present value objective embedded in a multi-pass heuristic. Ash (1999) proposed a deterministic simulation scheme using available project data to choose an activity scheduling heuristic which not only allows for the establishment of good project schedules, but determines a priori which resources will be assigned to specific project activities. A graphical interface is updated as the simulation runs and the ability to stop and modify decision-making while a simulation is in progress could be useful to project managers. Such extensions might lead to insights into project progress and resource utilization, while allowing project schedulers to apply judgment that a pure heuristic approach lacks.

Lova et al. (2000) developed a multi-criteria heuristic that improves lexicographically two criteria: one time type (mean project delay in relation to the unconstrained critical path duration or multi-project duration increase) and one no time type (project splitting, in-process inventory, resource leveling or idle resources) that can be chosen by the user. The multi-criteria heuristic algorithm consists of several algorithms based on the improvement of multi-project feasible schedules. Through an extensive computational study, they have shown that this method improves the feasible multi-project schedule obtained from heuristic methods based on the priority rules coded *Maximum Total Work Content* (MAX-TWK) and *Minimum Latest Finish Time* (MINLFT) as well as project management software – *Microsoft Project*, *CA-SuperProject*, *Time Line*, and *Project Scheduler*. Lova and Tormos (2002) developed combined random sampling and backward-forward heuristics for the objectives of mean project delay and multi-project duration increase.

Mendes (2003) presents a genetic algorithm that uses a random key representation and a modified parallel *Schedule Generation Scheme* (SGS). The modified parallel SGS determines all activities to be eligible which can be started up to the schedule time plus a delay time. This genetic algorithm minimizes simultaneously the tardiness, earliness, and flow time deviation criteria.

4. NEW APPROACH

The new approach presented in this paper combines a new measure of performance, a genetic algorithm based on random keys, and, as described Section 6, a new schedule generation procedure that creates parameterized active schedules (Gonçalves and Beirão, 1999; Gonçalves et al., 2005). In general terms, the approach innovates in the following two fundamentals areas:

1. *The model.* A new measure of performance is developed. This measure attempts to capture reality by integrating due dates, work in process, and inventory. Constraints enforcing the release date concept are also introduced.
2. *Solution method.* Considering the difficulty 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.

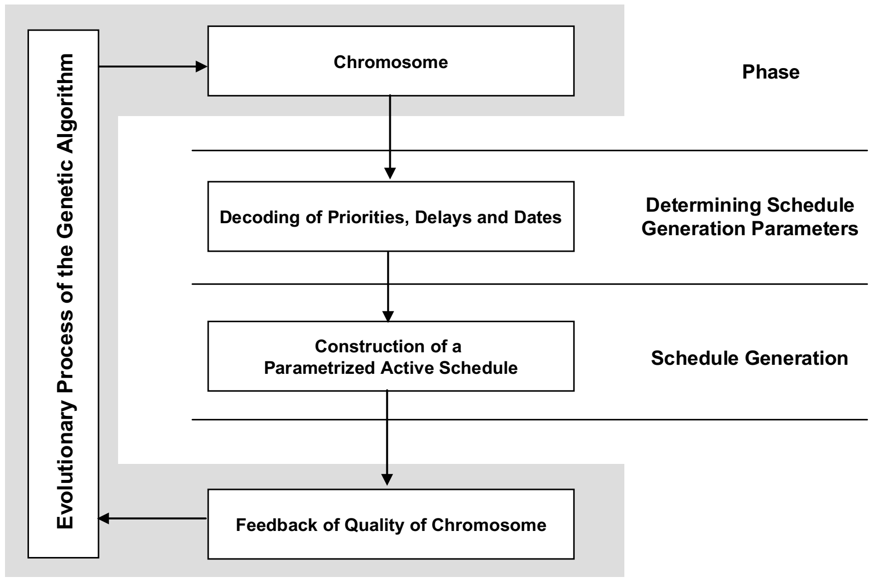

Evolutionary Process of the Genetic Algorithm
Chromosome
Phase
Decoding of Priorities, Delays and Dates
Determining Schedule Generation Parameters
Construction of a Parametrized Active Schedule
Schedule Generation
Feedback of Quality of Chromosome

FIGURE 2. Architecture of the new approach. Genetic algorithm evolves chromosomes, which are passed to the decoder. Decoder determines schedule generation parameters (priorities, delays, and dates) and passes them to solution generator, which builds parametrized active schedules. Schedules are evaluated and fitnesses are fedback to the genetic allgorithm.

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:

1. *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.
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 quality (performance measure) is fed back to the genetic algorithm. Figure 2 illustrates the sequence of steps applied to each chromosome generated by the genetic algorithm.

Unlike with tabu search or simulated annealing, genetic algorithms, in general, require that the search space be connected. Since our genetic algorithm uses random numbers in the interval $[0,1]$, it searches an n-dimensional hypercube, which is, of course, connected. The mutation procedure guarantees that, if a sufficiently large number of generations are carried out, the genetic algorithm will sample the entire hypercube and consequently find the best set of random keys.

5. THE NEW 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 subsections describe the details of the refinements of the model.

5.1. **Performance measure.** Project management is a complex decision making process involving due date (tardiness), start (earliness), and work in process (flow time) constraints. The new performance measure incorporates simultaneously three criteria: tardiness, earliness, and flow time. The following notation will be used:

D_i:	Ideal duration for project i.
DD_i:	Due date for project i.
CD_i:	Conclusion date for project i in generated schedule.
BD_i:	Start date for project i in generated schedule.
T_i:	Tardiness of project $i = \max\ \{CD_i\ -\ DD_i\,,\ 0\}$.
E_i:	Earliness of project $i = \max\ \{DD_i\ -\ CD_i\,,\ 0\}$.
FD_i:	Flow time deviation for project $i = \max\ \{CD_i\ -\ BD_i - D_i,\ 0\}$.
CPD_i:	Critical path duration of project i.

The new performance measure is defined as

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

where a, b, and c are parameters defined by the decision maker. To overcome the problem of not knowing the ideal duration of a project in a real-world situation, we replace

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

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 constraints

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

to the model, where MDL_i represents earliest release date for project i. These constraints are enforced in the model implicitly by assigning a duration $DL_i \geq MDL_i$ to the initial activity of each project, i.e.,

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

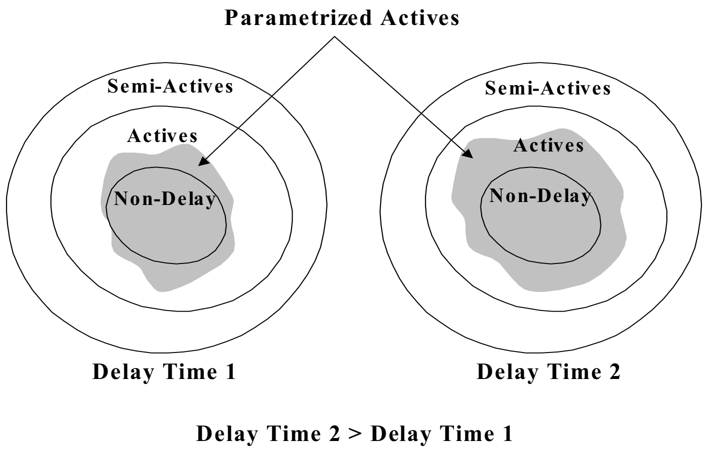


FIGURE 3. Parameterized active schedules for different values of the delay time.

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. (2005). 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 3 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay 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 t_g. The active set comprises all activities which are active at t_g, i.e.

$$\mathcal{A}(t_g) = \left\{ j \in \mathcal{J} \,|\, F_j - d_j \le t_g < F_j \right\}.$$

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

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

The set 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 the set E_g comprise all activities which are precedence-feasible in the interval $[t_g, t_g + Delay_g]$, i.e.

$$E_g = \left\{ j \in \mathcal{J} \setminus S_{g-1} \,|\, F_i \le t_g + Delay_g \ (i \in \mathcal{P}_j) \right\}.$$

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

```
procedure CONSTRUCT-PARAMETRIZED-ACTIVE-SCHEDULES
1  Initialization: $g \leftarrow 1$; $t_1 \leftarrow 0$; $A_0 \leftarrow \{0\}$; $\Gamma_0 \leftarrow \{0\}$;
                   $S_0 \leftarrow \{0\}$; $RD_k(0) \leftarrow R_k$, $(k \in \mathcal{K})$;
2  while $|S_g| < n+2$ repeat
3      Update: $E_g$;
4      while $E_g \neq \{\}$ repeat
5          Select activity with highest priority:
           $j^* \leftarrow \underset{j \in E_g}{\operatorname{argmax}} \{PRIORITY_j\}$;
6          Calculate earliest finish time (in terms of precedence only):
           $EF_{j^*} = \max_{i \in \mathcal{P}_j}\{F_i\} + d_{j^*}$;
7          Calculate earliest finish time (in terms of precedence and capacity):
           $F_{j^*} \leftarrow d_{j^*} + \min\{t \in [FMC_{j^*} - d_{j^*}, \infty] \cap \Gamma_g \mid r_{j^*,k} \leq RD_k(\tau),$
               $k \in \mathcal{K} \mid r_{j^*,k} > 0, \tau \in [t, t + d_{j^*}]\}$;
8          Update: $S_g \leftarrow S_{g-1} \cup \{j^*\}$; $\Gamma_g \leftarrow \Gamma_{g-1} \cup \{F_{j^*}\}$;
9          Iteration increment: $g \leftarrow g+1$;
10         Update: $A_g$, $E_g$, $RD_k(t) \mid t \in [F_{j^*} - d_{j^*}, F_{j^*}]$, $k \in \mathcal{K} \mid r_{j^*,k} > 0$;
11     end while;
12     Determine the time associated with activity g;
       $t_g \leftarrow \min\{t \in \Gamma_{g-1} \mid t > t_{g-1}\}$;
13 end while;
end CONSTRUCT-PARAMETRIZED-ACTIVE-SCHEDULES;
```

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

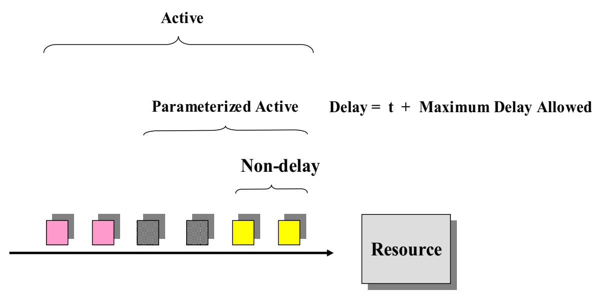
parameterized active schedules is incorporated in the selection step of the procedure,

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

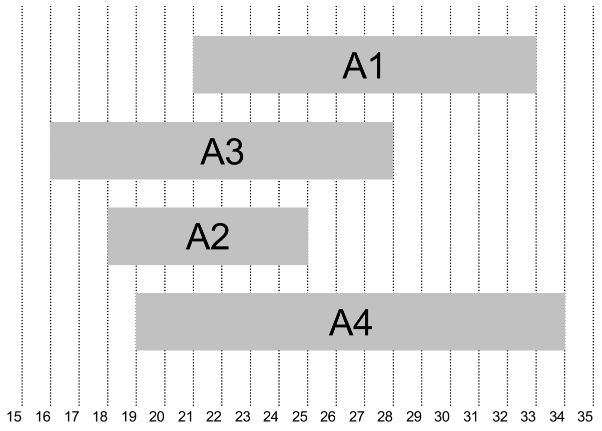
The set E_g is responsible for forcing the selection to be made only amongst activities which will have a delay smaller or equal to the maximum allowed delay. Figure 5a illustrates the different queue sizes in the selection step according to the type of schedule, i.e., non-delay, parameterized active, and active schedules. Figure 5b depicts a Gantt chart where activities A1, A2, A3, and A4 are being processed. Let A1-N, A2-N, A3-N, and A4-N denote the activities that depend on the end of activities A1, A2, A3, and A4, respectively, and are to be processed on the same resource. The table below the Gantt chart shows the queue of eligible activities at the resource when $t = 25$ and when the delay parameter is equal to 0, 3, 8, and 9 time units. The parameters $PRIORITY_j$ (priority of activity j) and $Delay_g$ (delay 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



(a) Eligible activities for the selection step at time



Activities in queue at *t = 25*	**Delay**
A2-N	0 (Non-delay)
A3-N A2-N	3
A1-N A3-N A2-N	8
A4-N A1-N A3-N A2-N	9

(b) Queue of eligible activities at ***t*** =25 for different values of the delay parameter

FIGURE 5. Eligible activities for different types of schedules.

(1859) in *The Origin of Species by Natural Selection*. By mimicking this process, genetic algorithms, if suitably encoded, are able to *evolve* solutions to real world problems. Before a genetic algorithm can be run, an *encoding* (or *representation*) for the problem must be devised. A *fitness function*, which assigns a figure of merit to each encoded solution, is also required. During the run, parents are *selected* for reproduction and *recombined* to generate offspring (see high-level pseudo-code in Figure 6).

```
procedure GENETIC-ALGORITHM
1  Generate initial population $P_0$;
2  Evaluate population $P_0$;
3  Initialize generation counter $g \leftarrow 0$;
4  while stopping criteria not satisfied repeat
5      Select some elements from $P_g$ to copy into $P_{g+1}$;
6      Crossover some elements of $P_g$ and put into $P_{g+1}$;
7      Mutate some elements of $P_g$ and put into $P_{g+1}$;
8      Evaluate new population $P_{g+1}$;
9      Increment generation counter: $g \leftarrow g+1$;
10 end while;
end GENETIC-ALGORITHM;
```

FIGURE 6. Pseudo-code of a standard genetic algorithm.

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.

7.1. **Chromosome representation.**

The genetic algorithm described in this paper uses a random key alphabet which is comprised of 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 usually called *genotype*, while in an indirect representation it does not and special procedures are needed to derive a solution from it usually 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 (phenotype) we use the parameterized active schedule generator described in Section 6. Each solution chromosome is made of $2n+m$ genes, where n is the number of activities and m is the number of projects:

$$Chromosome = (\underbrace{gene_1, \dots, gene_n}_{\text{Priorities}}, \underbrace{gene_{n+1}, \dots, gene_{2n}}_{\text{Delay Times}}, \underbrace{gene_{2n+1}, \dots, gene_{2n+m}}_{\text{Release Dates}})$$

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 which schedules one activity per iteration. The last m genes are used to determine the release dates of each of the m projects.

7.2. **Decoding.** We next 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:

1. *GA-Basic:* A basic decoding procedure;
2. *GA-SlackNd:* 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;
3. *GA-SlackMod:* A more sophisticated decoding procedure in which problem specific information is included.

The next subsection 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 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.

GA-Basic: For this solution alternative, the priority of each activity $j \in \mathcal{J}$ is given by the gene value, i.e.

$$Priority_j = Gene_j.$$

GA-SlackNd: For this solution alternative, the priority of each activity $j \in \mathcal{J}$ is given by the normalized slack calculated by the expression

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

where *MaxSlack* is the maximum slack for all activities amongst all projects, $Slack_j = DD_{i \mid j \in i} - LLP_j$, where $DD_{i \mid j \in i}$ is the due date of the project i to which activity j belongs and LLP_j is the longest length path from the beginning of activity j to the end of the project i to which activity j belongs.

GA-SlackMod: 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 expression

$$Priority_j = \frac{Slack_j}{MaxSlack} \times (0.7 + 0.3 \times Gene_j).$$

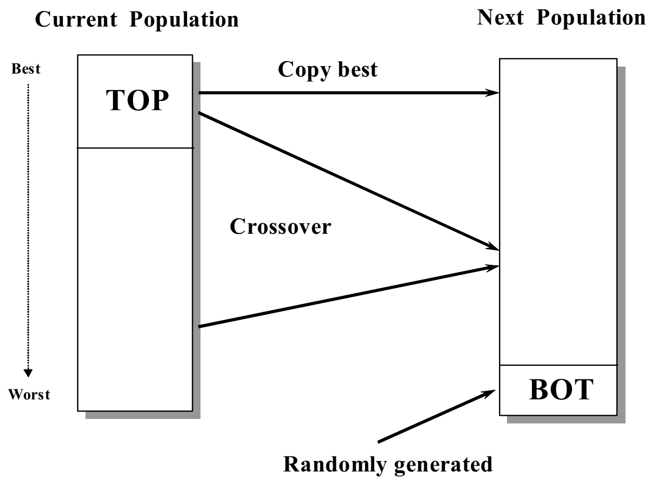


FIGURE 7. Transitional process between consecutive generations. Current population is sorted from best to worst. Top (TOP) individuals from current population are copied unchanged to next population. Bottom (BOT) individuals in current population are replaced by randomly generated individuals in the next population. The remaining individuals of the next population are generated by applying crossover operator to randomly selected individual from TOP individuals of current population and randomly selected individual from entire current population.

7.2.2. *Decoding of the delays.* The genes between $n+1$ and $2n$ are used to determine the delay times $Delay_g$, 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.

GA-Basic and GA-SlackMod: For these solutions alternatives, the delay schedules generated are given by

$$Delay_g = Gene_g \times 1.5 \times MaxDur,$$

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.

GA-SlackNd: For this solution alternative, the delay schedules generated are non-delay. Therefore, all delays are zero, i.e.

$$Delay_g = 0.$$

7.2.3. *Decoding of the release dates.* The last m genes of each the chromosome, (genes $2n+1$ to $2n+m$) are used to determine the release dates of each project $i \in I$. All of the above solution alternatives (*GA-Basic*, *GA-SlackNd*, and *GA-SlackMod*) use the following decoding expression to obtain the release date of each project $i \in I = \{1,\dots,m\}$:

$$DL_i \ = \ MDL_i + Gene_{2n+i} \times (DD_i - MDL_i)\,.$$

7.3. **Evolutionary strategy.** To breed good solutions, the random key vector population is operated upon by a genetic algorithm. Many variations of GAs 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.

Each individual of the initial population is initialized with a random key vector. Given a current population, we perform the following three steps in order to obtain the next generation (see Figure 7):

1. *Reproduction:* Some of the best individuals are copied from the current generation into the next (see TOP in Figure 7). 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 (DeJong and Spears, 1991) are used as opposed to the traditional one-point or two-point crossover. Two individuals are randomly chosen to act as parents to produce an offspring. One of the parents is chosen amongst the best individuals in the population (TOP in Figure 7), while the other is randomly chosen from the whole current population (including TOP). For each gene, a random number in the interval $[0,1]$ is generated. If the random number obtained is smaller than a threshold value (for example 0.7) called crossover probability (*CProb*), then the allele of the first parent is inherited (or selected) by the offspring. Otherwise, the allele selected is that of the second parent. An example of a crossover outcome is given in Figure 8.
3. *Mutation.* Mutation, in this scheme, 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 8). 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. One can think of these mutants as being immigrants. 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.

8. PROBLEM INSTANCE GENERATOR

In this section, we describe the problem generator used to produce test problems for the computational experiments. 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 for which the optimal value (for the measure of performance described in Section 5.1) is zero (i.e. *tardiness* $= 0$, *earliness* $= 0$, and *flow time deviation*$= 0$). It is possible that these instances are easy to solve. Nevertheless, we use them to measure the performance of our approach. The problem generator has the following input parameters:

- Number of problems to generate;

Chromosome 1	**0.32**	**0.77**	**0.53**	**0.85**

Chromosome 2	0.26	0.15	0.91	0.44

Random Number	**0.58**	0.89	**0.68**	**0.25**
Relation to crossover probability of 0.7	$<$	$>$	$<$	$<$

Offspring chromosome	**0.32**	0.15	**0.53**	**0.85**

FIGURE 8. Example of parameterized uniform crossover with crossover probability equal to 0.7. For each allele, a random number in the interval $[0,1]$ is generated. With probability 0.7 the offspring inherits the allele of Chromosome 1 and with probability 0.3, it inherits the one of Chromosome 2.

- Number of projects to include in each problem;
- Average number of projects 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 (`http://129.187.106.231/psplib/`);
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 expression
$$\frac{\textit{Sum of ideal duration of each single-project instance}}{\textit{Number of single-projects to be simultaneously in execution}};$$
4. The *start date* of each project is randomly generated in the interval
$$[0, \textit{problem due date} - \textit{ ideal duration}].$$
5. The resource capacity in the interval $[0, \textit{due date upper bound}]$ is calculated by the adding the resource capacities of each single-project instance from its start 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 (and since there are no benchmark problems instances available for the RCMPSP where the measure of performance includes tardiness, earliness, and flowtime deviation) we used multi-project instances generated by the problem instance generator described in the previous section.

Five types of multi-project instances where generated, respectively, with 10, 20, 30, 40, 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, 4800, and 6000 activities, respectively. Each activity can use up to four resources. The average number of overlapping projects in execution can be 3, 6, 9, 12, and 15. Table 2 shows the combinations of the number of overlapping projects used for the problems with 10, 20, 30, 40, and 50 single-projects.

9.1. **GA configuration.** Though there is no straightforward way to configure the parameters of a genetic algorithm, our past experience with genetic algorithms based on the same evolutionary strategy (see Gonçalves and Almeida (2002), Ericsson et al. (2002), Gonçalves and Resende (2004), Gonçalves et al. (2005), and Buriol et al. (2005)) has shown that good results can be obtained with the values of TOP, BOT, and Crossover Probability (*CProb*) shown in Table 3.

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 factorial analysis on a small pilot set of 12 problem instances not included in the experimental tests. The factorial analysis combines 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 0.5 (in intervals of 0.1) times the number of activities in the multi-project problem instance. The total factorial analysis included 180 possible configurations of the GA

For all the 60 possible combinations of TOP, BOT, and population sizes and for all problem instances in the pilot set the GA obtained the same fitness values for all tested values of *CProb* = (0.70, 0.75, 0.80). Also, for population sizes greater than 0.2 times the number of activities in the multi-project problem instance, the results were equal for all combinations of TOP, BOT and *CProb*. This last point reduced our statistical analysis only to population sizes smaller or equal to 0.2 times the number of activities in the multi-project problem instance. Table 1 presents summary results of factorial analysis for the GA parameters. Each configuration is represented by the tuple

$$\textit{population size factor} - \textit{TOP} - \textit{BOT},$$

where the population size equals the population size factor times the number of activities in the multi-project problem instance. We exclude the *CProb* value from the configuration representation since the GA obtained the same fitness values for all tested values of *CProb*.

Configuration 0.2–10–20 is the best in terms of the sum of fitness values and the number of best results (7 out 12). To test if the differences between configuration 0.2–10–20 and the other 23 configurations were statistically significant, we used the Wilcoxon's signed rank test since the differences between the configurations were not normally distributed. The last column in Table 1 presents the *P*-value of the signed rank test for each the 23 paired comparisons. The five *P*-values starting with an $*$ indicate that we cannot consider the results obtained by two configurations to be significantly different at a 5% confidence level. Nevertheless, and since configuration 0.2–10–20- obtains the best number of best results, we used it to configure our GA for the experimental tests.

The configuration shown in Table 4 was held constant for all experiments and all problem instances. The experimental results demonstrate that this configuration provides high-quality solutions and that it is very robust.

9.2. **Experimental results.** Table 2 summarizes the experimental results. It lists the fitness, earliness, tardiness, and flow time deviation. Let N be the number of projects in each problem instance. Averages and standard deviation were computed for 20 problem

TABLE 1. Results of the factorial analysis for the GA parameters. For each parameter configuration, the GA was run on 12 test problems. The fitness of the best solution found for each configuration/problem pair is given in the table as are the sums of the fitness values for each configuration, the number of best solutions found by each configuation, and the *P*-values found by applying Wilcoxon's signed rank test to the differences between configuration 0.2–10–20 and the other 23 configurations.

	Problems												Sum of	Num. of	
Configuration	P10-3-1	P10-3-2	P20-3-1	P20-3-2	P20-6-1	P20-6-2	P30-3-1	P30-3-2	P30-6-1	P30-6-2	P30-9-1	P30-9-2	Fitness	Best	*P*-Value
0.1-10-15	44.00	28.70	88.80	206.00	0.45	2.10	1915.73	1325.99	45.29	0.33	0.57	0.50	3658.46	0	0.00326385
0.1-10-20	42.10	2.70	1.65	187.25	0.50	65.40	3314.63	1438.43	42.24	0.37	0.57	0.50	5096.33	1	0.00533770
0.1-10-25	10.00	1.50	94.80	63.85	0.45	32.65	1254.23	1317.76	48.63	0.30	0.57	0.53	2825.27	1	0.00535537
0.1-10-30	10.00	0.40	6.75	63.60	0.40	42.65	3142.03	1992.04	67.13	0.30	0.53	0.43	5326.27	4	0.02514460
0.1-15-15	78.00	37.10	33.00	49.35	0.50	29.35	972.40	1246.43	43.49	0.33	0.57	0.50	2491.02	1	∗ 0.107666
0.1-15-20	9.90	24.70	22.70	689.05	0.45	6.30	1117.13	1465.65	43.56	0.33	0.53	0.43	3380.74	2	0.02514460
0.1-15-25	0.40	19.10	26.25	508.00	0.40	2.70	1306.13	2543.34	56.17	0.40	0.57	0.50	4463.96	1	0.00326385
0.1-15-30	10.10	3.10	22.25	348.10	0.50	0.55	1590.63	1269.31	33.86	0.37	0.57	0.50	3279.84	0	0.01078750
0.1-20-15	10.20	11.90	204.50	363.55	0.45	4.05	1574.37	1506.65	50.41	0.40	0.57	0.47	3727.51	0	0.00252631
0.1-20-20	5.30	4.10	173.95	606.85	0.50	55.55	1448.10	1219.93	29.49	0.37	0.53	0.50	3545.17	3	∗ 0.0542413
0.1-20-25	14.30	0.30	139.05	436.20	0.45	4.80	1270.83	2339.89	59.27	0.33	0.57	0.50	4266.49	1	0.00535537
0.1-20-30	70.20	0.50	237.15	261.30	0.45	4.05	1887.03	2341.21	54.18	0.37	0.53	0.50	4857.47	1	0.00533770
0.2-10-15	1.00	0.90	154.75	59.90	0.50	0.60	1915.73	1325.99	45.29	0.33	0.57	0.50	3506.06	0	0.01078750
0.2-10-20	**0.10**	**0.30**	**1.65**	**49.35**	**0.40**	**0.95**	**1117.13**	**1246.43**	**29.49**	**0.37**	**0.53**	**0.43**	**2447.13**	**7**	**–**
0.2-10-25	12.00	0.60	163.75	128.60	0.45	0.95	1254.23	1317.76	48.63	0.30	0.57	0.53	2928.37	1	0.00859001
0.2-10-30	2.30	0.30	8.15	431.55	0.45	3.60	3142.03	1992.04	29.49	0.30	0.5	30.43	5658.62	5	0.03733890
0.2-15-15	3.10	3.20	104.70	129.60	0.45	0.35	972.40	1246.43	43.49	0.33	0.57	0.50	2505.12	2	∗ 0.16981
0.2-15-20	0.10	13.80	31.55	542.50	0.45	126.40	1117.13	1465.65	43.56	0.33	0.53	0.43	3342.44	3	∗ 0.0642234
0.2-15-25	0.20	1.10	34.20	119.70	0.45	0.35	1306.13	2543.34	56.17	0.40	0.57	0.50	4063.11	1	0.01078750
0.2-15-30	0.30	2.30	91.20	310.50	0.45	0.40	1590.63	1269.31	33.86	0.37	0.57	0.50	3300.39	0	0.01347120
0.2-20-15	10.00	3.00	566.45	1040.60	0.45	60.40	1574.37	1506.65	50.41	0.40	0.57	0.47	4813.76	0	0.00252631
0.2-20-20	1.20	14.70	15.95	304.95	0.40	2.65	1448.10	1219.93	29.49	0.37	0.53	0.50	3038.77	4	∗ 0.145151
0.2-20-25	0.40	2.10	43.10	308.40	0.45	8.50	1270.83	2339.89	59.27	0.33	0.57	0.50	4034.34	0	0.00326385
0.2-20-30	0.20	13.10	27.55	379.90	0.45	3.10	1887.03	2341.21	54.18	0.37	0.53	0.50	4708.12	1	0.00533770

TABLE 2. Experimental results. Each of the three algorithms (*GA-SlackMod*, *GA-SlackND*, and *GA-Basic*) was run on 20 instances of five problem types (having 10, 20, 30, 40, and 50 projects). For each problem type, algorithm, and different value of overlapping projects, the table shows averages and standard deviations for fitness, tardiness, earliness, and flow time deviations, as well as the number of best solutions found and the percentage improvement of the average last generation fitness values to those of the first generation, i.e. $100 \times$ (Fitness at 1st gen. $-$ Fitness at last gen.)$/$(Fitness at 1st gen.)

	Overl'ng		Fitness		Tardiness		Earliness		Flow Deviation		Best	Percentage
Proj's.	Proj's.	Algorithm	Avg^1	SD^1	Avg^2	SD^2	Avg^3	SD^3	Avg^4	SD^4	sol'n	Improvement
		GA-SlackMod	**10.35**	**18.56**	**0.00**	**0.00**	**1.20**	**1.41**	**0.38**	**0.54**	**17**	**99.98900**
10	3	GA-SlackND	89.79	334.90	0.02	0.07	1.97	1.19	0.32	0.49	3	
		GA-Basic	621.40	691.98	0.12	0.18	8.91	5.63	3.43	3.47	1	
		GA-SlackMod	**73.14**	**117.52**	**0.00**	**0.00**	**2.57**	**2.91**	**1.07**	**1.97**	**17**	**99.99947**
	3	GA-SlackND	131.27	117.33	0.01	0.02	5.25	3.22	0.76	0.74	3	
20		GA-Basic	27400.19	949504.86	2.37	2.54	20.91	10.89	5.84	4.49	0	
		GA-SlackMod	**0.95**	**2.10**	**0.00**	**0.00**	**0.42**	**0.27**	**0.03**	**0.07**	**20**	**99.99998**
	6	GA-SlackND	21.12	59.53	0.00	0.00	1.25	1.31	0.16	0.32	0	
		GA-Basic	612.51	882.51	0.08	0.16	8.38	4.48	0.66	0.81	0	
		GA-SlackMod	**210.13**	**202.81**	**0.01**	**0.02**	**3.92**	**2.88**	**1.74**	**1.45**	**18**	**99.99688**
	3	GA-SlackND	459.93	295.66	0.10	0.08	6.97	3.97	2.21	1.62	2	
		GA-Basic	279192.26	256884.49	4.34	1.66	30.77	13.03	5.29	3.51	0	
		GA-SlackMod	**3.89**	**7.11**	**0.00**	**0.00**	**0.60**	**0.40**	**0.09**	**0.20**	**20**	**99.99998**
30	6	GA-SlackND	29.47	49.86	0.00	0.01	1.50	0.98	0.24	0.35	0	
		GA-Basic	3164.16	3766.42	0.35	0.34	14.25	9.21	0.64	0.53	0	
		GA-SlackMod	**0.48**	**0.37**	**0.00**	**0.00**	**0.38**	**0.12**	**0.02**	**0.05**	**19**	**99.99999**
	9	GA-SlackND	11.23	33.13	0.00	0.01	0.60	0.38	0.03	0.07	2	
		GA-Basic	304.92	311.37	0.01	0.02	5.00	3.78	0.23	0.26	0	
		GA-SlackMod	**1324.14**	**1282.69**	**0.06**	**0.06**	**9.45**	**7.29**	**6.15**	**4.77**	**15**	**99.99949**
	3	GA-SlackND	1741.61	1044.50	0.20	0.08	13.31	14.64	6.50	3.20	5	
		GA-Basic	1670572.27	1890484.11	7.74	3.29	37.49	14.64	5.18	3.11	0	
		GA-SlackMod	**6.18**	**15.00**	**0.00**	**0.00**	**0.59**	**0.35**	**0.11**	**0.22**	**18**	**99.99998**
	6	GA-SlackND	41.00	43.02	0.02	0.02	1.47	0.84	0.19	0.34	2	
40		GA-Basic	7736.26	590.32	0.73	0.44	15.02	9.61	0.74	0.75	0	
		GA-SlackMod	**4.48**	**16.52**	**0.00**	**0.00**	**0.50**	**0.25**	**0.06**	**0.21**	**18**	**99.99999**
	9	GA-SlackND	11.41	29.82	0.01	0.02	0.74	0.30	0.02	0.06	2	
		GA-Basic	919.64	921.75	0.07	0.12	7.83	4.95	0.13	0.12	0	
		GA-SlackMod	**2.00**	**4.28**	**0.00**	**0.00**	**0.52**	**0.26**	**0.04**	**0.08**	**17**	**100.00000**
	12	GA-SlackND	6.36	8.63	0.01	0.01	0.62	0.22	0.05	0.11	2	
		GA-Basic	223.80	250.23	0.00	0.02	4.00	3.89	0.08	0.11	2	
		GA-SlackMod	**2584.49**	**2887.14**	**0.07**	**0.04**	**14.68**	**5.68**	**7.40**	**6.42**	**11**	**99.91177**
	3	GA-SlackND	2441.51	691.81	0.22	0.10	14.81	5.68	6.60	6.54	9	
		GA-Basic	2613944.26	3101743.13	8.79	3.72	38.15	16.38	5.96	2.80	0	
		GA-SlackMod	**25.87**	**57.23**	**0.00**	**0.00**	**0.87**	**0.60**	**0.23**	**0.39**	**17**	**99.99996**
	6	GA-SlackND	68.61	47.68	0.04	0.03	1.69	0.76	0.31	0.31	3	
		GA-Basic	25674.75	30128.67	1.33	0.70	19.41	13.10	0.86	0.65	0	
		GA-SlackMod	**0.73**	**0.79**	**0.00**	**0.00**	**0.43**	**0.11**	**0.02**	**0.05**	**20**	**99.99999**
50	9	GA-SlackND	11.43	25.26	0.00	0.00	0.81	0.35	0.13	0.13	0	
		GA-Basic	1580.02	1190.59	0.22	0.20	8.90	4.79	0.20	0.18	0	
		GA-SlckMod	**1.35**	**2.16**	**0.00**	**0.00**	**0.50**	**0.17**	**0.02**	**0.05**	**18**	**100.00000**
	12	GA-SlackND	4.22	8.07	0.00	0.00	0.62	0.20	0.04	0.05	1	
		GA-Basic	412.25	355.36	0.03	0.05	4.53	2.74	0.10	0.09	1	
		GA-SlackMod	**1.07**	**1.98**	**0.00**	**0.00**	**0.50**	**0.15**	**0.01**	**0.04**	**13**	**100.00000**
	15	GA-SlackND	1.45	2.47	0.00	0.00	0.52	0.11	0.02	0.02	8	
		GA-Basic	214.77	243.85	0.01	0.02	3.57	3.25	0.08	0.11	3	

TABLE 3. Range of parameters that produced good results in previous implementations of this evolutionary strategy.

Parameter	**Interval**
TOP	0.10 – 0.20
BOT	0.15 – 0.30
Crossover Probability (*CProb*)	0.70 – 0.80

TABLE 4. GA configuration for computational experiments. These values were used in the computational experiments on all three algorithms on all test instances.

Population size:	min (0.2 × 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

instances. Columns Avg^1 and SD^1 list averages and standard deviations for the expression

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

Avg^2 and SD^2 list averages and standard deviations for the expression

$$\frac{\sum_{i=1}^{N} E_i}{N};$$

Avg^3 and SD^3 list averages and standard deviations for the expression

$$\frac{\sum_{i=1}^{N} T_i}{N};$$

and Avg^4 and SD^4 list averages and standard deviations for the expression

$$\frac{\sum_{i=1}^{N} FD_i}{N}.$$

Algorithm *GA-SlackMod* was the winner in all aspects relative to the other two. For all instances, in absolute terms, algorithm *GA-SlackMod* obtained earliness, tardiness, and flow time deviation close to the optimum value (i.e. near zero). To test if the differences between *GA-SlackMod* and the other two (*GA-Basic* and *GA-SlackND*) were statistically significant, we again used Wilcoxon's signed rank test. Table 5 presents the *P*-values of the paired comparisons. As can be observed, *GA-SlackMod* was significantly better than any of the other two at a confidence level smaller than 1%.

TABLE 5. *P*-values of the paired comparisons. Obtained by applying Wilcoxon's signed rank test to the differences between algorithm *GA-SlackMod* and the other two.

Paired comparison	*P*-value
GA-Basic vs *GA-SlackMod*	0.000000
GA-SlackND vs *GA-SlackMod*	0.000000

The last column of Table 2 presents percentage improvement of the average last generation fitness values to those of the first generation. It is clear that the GA is responsible for a big improvement in the quality of the solution. Sometimes the average percentage improvement is as large as 100%. As expected, the fitness obtained gets smaller, and is thus better, as the number of overlappings of projects increases. This is due to the fact that as the number of overlappings of projects increases so does the flexibility in terms of capacity, therefore allowing for more possibilities of finding a good schedule.

TABLE 6. Experimental results for the algorithm *GA-SlackMod*. For each problem type (10, 20, 30, 40, and 50 projects) and number of overlapping projects, the table lists averages for fitness, tardiness, earliness, and flow time deviations for two performance measures.

	Overlapping	Fitness		Tardiness		Earliness		Flow Time	
Projects	Projects	1)	2)	1)	2)	1)	2)	1)	2)
10	3	10.35	53.58	0.00	0.00	1.20	0.71	105.63	106.60
20	3	73.14	118.62	0.00	0.00	2.57	4.52	103.84	116.01
	6	0.95	41.03	0.00	0.00	0.42	0.36	96.57	96.89
30	3	210.16	23.41	0.01	0.02	3.92	12.67	104.30	143.77
	6	3.89	34.25	0.00	0.00	0.60	0.57	97.01	97.44
	9	0.48	40.38	0.00	0.00	0.38	0.41	95.14	95.20
40	3	1324.14	2637.71	0.06	0.04	9.45	30.17	109.26	187.46
	6	6.18	48.24	0.00	0.00	0.59	0.84	96.19	96.57
	9	4.48	33.94	0.00	0.00	0.50	0.39	96.01	95.89
	12	2.00	41.19	0.00	0.00	0.52	0.44	96.89	5.26
50	3	2584.49	6638.50	0.07	0.03	14.68	55.21	110.10	262.52
	6	25.87	43.04	0.00	0.00	0.87	1.01	96.14	97.04
	9	0.73	29.06	0.00	0.00	0.43	0.39	95.67	95.83
	12	1.35	42.65	0.00	0.00	0.50	0.54	95.24	95.21
	15	1.07	39.40	0.00	0.00	0.50	0.43	94.56	94.60

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

TABLE 7. Average elapsed time for 50 generations. This table lists average elapsed running time for all three algorithms on all test instances in each test problem class.

Problem instance type (number of projects):	10	20	30	50
Average elapsed time for 50 generations:	178 s	449 s	840 s	1860 s

Table 6 presents the results obtained by algorithm *GA-SlackMod* with the performance measure (5), where $\sum_i FD_i^2$ is replaced by

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

The algorithms were 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 7.

10. CONCLUSIONS AND FURTHER RESEARCH

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 of test problem with 10, 20, 30, 40, and 50 projects (having 1200, 2400, 3600, 4800, and 6000 activities, respectively). In the computational experiments, the algorithm *GA-SlackMod* had 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.

Further work could be conducted to explore the possibility of using activities with multi-mode usage of resources. Furthermore, the genetic algorithm could be used to determine the level of availability of each resource within a certain predefined range of values.

REFERENCES

R. Ash. Activity scheduling in the dynamic, multi-project setting: Choosing heuristics through deterministic simulation. In *Proceedings of the 1999 Winter Simulation Conference*, pages 937–941, Pheoenix, USA., 1999.

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.

D.B. Bock and J.H. Patterson. A comparison of due date setting, resource assignment, and job preemption heuristics for the multi-project scheduling problem. *Decision Sciences*, 21:387–402, 1990.

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.

C.R. Darwin. *On the origin of species through natural selection*. John Murray, London, 1859.

R.F. Deckro, E.P. Winkofsky, J.E. Hebert, and R. Gagnon. A decomposition approach to multi-project scheduling. *European Journal of Operational Research*, 51:110–118, 1991.

K. DeJong and W. Spears. On the virtues of parameterised uniform crossover. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236. Morgan Kaufman, San Mateo, CA, 1991.

A. Drexl. Scheduling of project networks by job assignment. *Management Science*, 37 (12):1590–1602, 1991.

J. Dumond and V.A. Mabert. Evaluating project scheduling and due date assignment procedures: An experimental analysis. *Management Science*, 34(1):101–118, 1988.

M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.

L.G. Fendley. Towards the development of a complete multiproject scheduling system. *Journal of Industrial Engineering*, 19(10):505–515, 1968.

M.R. Garey and D.S. Johnson. *Computers and intractability. A Guide to the theory of NP-completeness*. W.H. Freeman and Company, New York, 1979.

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. *Jornal of Heuristics*, 8:629–642, 2002.

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, 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 and M.G.C. Resende. An evolutionary algorithm for manufacturing cell formation. *Computers & Industrial Engineering*, 47:247–273, 2004.

R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark instances for scheduling problems. In J.Weglarz, editor, *Handbook on recent advances in project scheduling*, pages 197–212. Kluwer, Amsterdam, 1998.

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

I.S. Kurtulus and S.C. Narula. Multi-project scheduling: Analysis of project performance. *IIE Transactions*, 17:58–66, 1985.

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

A. Lova, C. Maroto, and P. Tormos. A multicriteria heuristic method to improve resource allocation in multiproject scheduling. *European Journal of Operational Research*, 127: 408–424, 2000.

A. Lova and P. Tormos. Combining random sampling and backward-forward heuristics for resource-constrained multi-project scheduling. In *Proceedings of the Eight International Workshop on Project Management and Scheduling*, pages 244–248, Valencia, Spain, 2002.

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.

R.P. Mohanthy and M.K. Siddiq. Multiple projects multiple resources-constrained scheduling: Some studies. *International Journal of Production Research*, 27(2):261–280, 1989.

L. Ozdamar, G. Ulusoy, and M. Bayyigit. A heuristic treatment of tardiness and net present value criteria in resource constrained project scheduling. *International Journal of Physical Distribution & Logistics Management*, 28:805–824, 1998.

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

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

S. Tsubakitani and R.F. Deckro. A heuristic for multi-project scheduling with limited resources in the housing industry. *European Journal of Operational Research*, 49:80–91, 1990.

C. Vercellis. Constrained multi-project planning problems: A Lagrangean decomposition approach. *European Journal of Operational Research*, 78:267–275, 1994.

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

(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

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

(Mauricio G. C. Resende) INTERNET AND NETWORK SYSTEMS RESEARCH CENTER, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.
E-mail address: mgcr@research.att.com