A Biased Random-Key Genetic Algorithm with Forward-Backward Improvement for the Resource Constrained Project Scheduling Problem*

José Fernando Gonçalves

LIAAD, Faculdade de Economia do Porto, Universidade do Porto

Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal

jfgoncal@fep.up.pt

Mauricio G. C. Resende

Algorithms and Optimization Research Department, AT&T Labs Research,

180 Park Avenue, Room C241, Florham Park, NJ 07932 USA

mgcr@research.att.com

Jorge J. M. Mendes

Instituto Superior de Engenharia do Porto, Instituto Politécnico do Porto

Rua Dr. António Bernardino de Almeida, 431 4200-072 Porto, Portugal

jjm@isep.ipp.pt

This paper presents a biased random-keys genetic algorithm for the Resource Constrained Project Scheduling Problem. The chromosome representation of the problem is based on random keys. Active schedules are constructed using a priority-rule heuristic in which the priorities of the activities are defined by the genetic algorithm. A forward-backward improvement procedure is applied to all solutions. The chromosomes supplied by the genetic algorithm are adjusted to reflect the solutions obtained by the improvement procedure. The heuristic is tested on a set of standard problems taken from the literature and compared with other approaches. The computational results validate the effectiveness of the proposed algorithm.

* Supported by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006.
AT&T Labs Research Technical Report.
Date: 2009-03-11.

Keywords: Project management, scheduling, genetic algorithms, random keys, forward-backward improvement, resource constrained project scheduling problem.

1 Introduction

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

1. *Precedence constraints* force each activity j to be scheduled after all predecessor activities P_j are completed;

2. Activities require resources with *limited capacities*.

While being processed, activity j requires $r_{j,k}$ units of resource type $k \in K$ during every time instant of its non-preemptable duration d_j. Resource type k has a limited capacity R_k at any point in time. The parameters d_j, $r_{j,k}$, and R_k are assumed to be integer, non-negative, and deterministic. For the project start and end activities, we impose the boundary conditions $d_0 = d_{n+1} = 0$ and $r_{0,k} = r_{n+1,k} = 0$, for all $k \in K$. The RCPSP consists in finding a schedule of the activities, taking into account the resources and the precedence constraints, that minimizes the makespan $C_{\max}$, i.e. that minimizes the finish time of the last activity to be processed.

Let F_j represent the finish time of activity j. A schedule can be represented by a vector of finish times $(F_1, \ldots, F_m, \ldots, F_{n+1})$. Figure 1 shows an example of a project comprising $n = 9$ activities which have to be scheduled, subject to one renewable resource type with a capacity of two units. Two solutions for this example are shown in Figure 2, an infeasible solution that violates the resource constraint with a makespan of 24 and an optimal feasible solution with a makespan of 36.

Several exact methods to solve the RCPSP are proposed in the literature. Currently, the most competitive exact algorithms seem to be the ones of Demeulemeester and Herroelen (1997), Brucker et al. (1998), Klein and Scholl (1998a;b), Mingozzi et al. (1998), and Sprecher (2000). Stork and Uetz (2005) present several complexity results related to generation and counting of all circuits of an independence system and study their relevance in the solution of RCPSP.

It was shown by Blazewicz et al. (1983) that the RCPSP, as a generalization of the classical job shop scheduling problem, is NP-hard, therefore justifying the use of heuristics to solve large problem instances.

Several authors propose procedures for computing lower bounds on the makespan of the RCPSP. Demassey et al. (2005) propose a cooperation method between constraint programming and integer programming. Brucker and Knust (2003) present a destructive lower bound for the multi-mode resource-constrained project scheduling problem with

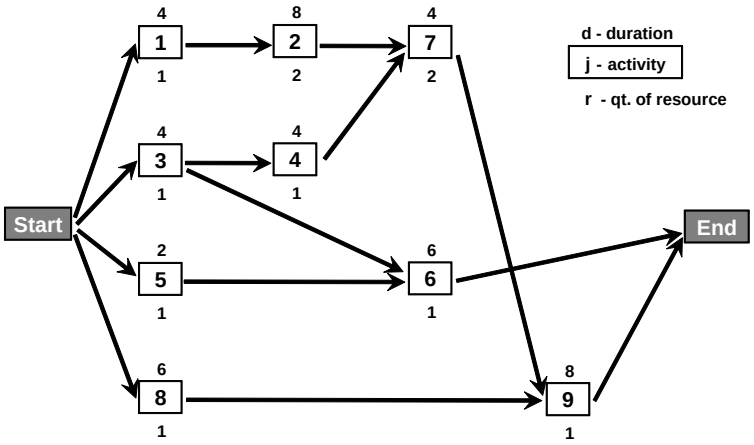

Figure 1: Project network example. Activities are represented as boxes and precedences by directed arcs. Parameters d_j/r_j are given for each activity j.

minimal and maximal time-lags. Brucker and Knust (2000) develop a destructive lower bound for the RCPSP, where the lower bound calculations are based on two methods for proving infeasibility of a given threshold value for the makespan. The first uses constraint propagation techniques, while the second is based on a linear programming formulation.

Most of the heuristic methods used for solving resource-constrained project scheduling problems either belong to the class of priority rule based methods or to the class of metaheuristic based approaches (Kolisch and Hartmann, 1999). The first class of methods starts with none of the jobs being scheduled. Subsequently, a single schedule is constructed by selecting a subset of jobs in each step and assigning starting times to these jobs until all jobs have been considered. This process is controlled by the scheduling scheme as well as priority rules with the latter being used for ranking the jobs. Several approaches of this class have been proposed in the literature, e.g. Alvarez-Valdez and Tamarit (1989), Boctor (1990), Cooper (1976; 1977), Davis and Patterson (1975), Lawrence (1985), Kolisch (1996a;b), Kolisch and Hartmann (1999), and Tormos and Lova (2001; 2003). The second class of methods improves upon an initial solution. This is done by successively executing operations which transform one or several solutions into others. Several approaches of this class have been proposed in the literature, e.g. genetic algorithms (Leon and Ramamoorthy (1995), Lee and Kim (1996), Hartmann (1998), Kohlmorgen et al. (1999), Hartmann (2002), Kochetov and Stolyar (2003), Mendes (2003), Valls et al. (2003; 2005), and Mendes et al. (2009)), simulated annealing (Slowinski et al. (1994), Boctor (1996), and Bouleimen and Lecocq (2003)), tabu search (Pinson et al. (1994), Baar et al. (1998), Thomas and Salhi (1998), Nonobe and Ibaraki (2002), and Gagnon et al. (2004)), local search-oriented approaches (Fleszar and Hindi (2004) and Palpant et al. (2004)), and population-based approaches (Debels et al. (2006) and Valls et al. (2003)).

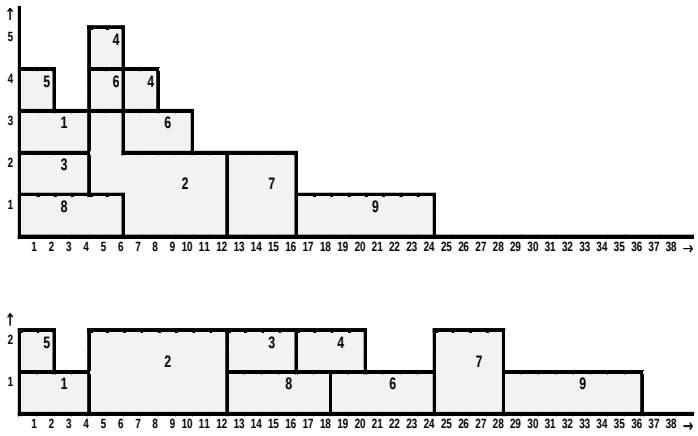

Figure 2: Two solutions for the project network example of Figure 1. On top, an infeasible schedule that ignores the resource constraint and results in a makespan of 24. On the bottom, a feasible schedule with an optimal makespan of 36.

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

In this paper, we present a new genetic algorithm for finding cost-effective solutions for the resource constrained project scheduling problem. The remainder of the paper is organized as follows. Section 2 discusses different classes of schedules. In Section 3, we present the new approach, based on a biased random-keys genetic algorithm (Gonçalves and Resende, 2009) and a schedule generation procedure, to find cost-effective solutions to the resource constrained project scheduling problem. Section 4 reports computational results and concluding remarks are made in Section 5.

2 Types of schedules

Schedules can be classified into one of following three types:

1. *Semi-active schedules.* These are feasible schedules obtained by sequencing activities as early as possible. In a semi-active schedule, no activity can be started earlier without altering the processing sequences.

2. *Active schedules.* These are feasible schedules in which no activity could be started earlier without delaying some other activity or breaking a precedence constraint. Active schedules are also semi-active schedules. An optimal schedule is always active, so the search space can be safely limited to the set of all active schedules.

3. *Non-delay schedules.* These are feasible schedules in which no resource is kept idle when it could start processing some activity. Non-delay schedules are active and hence are also semi-active.

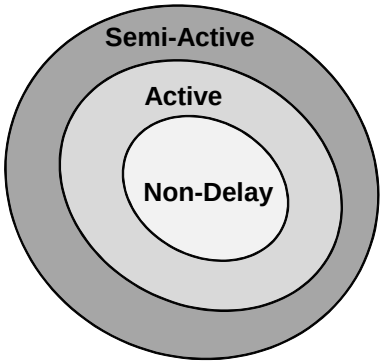

Figure 3: Types of schedules.

Figure 3 illustrates where the set of active schedules are located relative to the class of semi-active and non-delay schedules. As mentioned above, an optimal schedule is in the set of all active schedules. Section 3.3 presents pseudo-code to generate active schedules.

3 New approach

3.1 Overview of the new approach

The new approach proposed in this paper combines a biased random-keys based genetic algorithm, a schedule generation scheme, an improvement procedure, and a chromosome adjustment procedure.

The role of the genetic algorithm is to evolve the encoded solutions, or *chromosomes*, which represent the priorities of the activities. For each chromosome, the following four phases are applied:

1. *Decoding of priorities.* In this phase the chromosome supplied by the genetic algorithm is transformed into the priorities of the activities.

2. *Schedule generation.* This phase makes use of the priorities defined in the first phase and constructs an active schedule using a *Serial Schedule Generation Scheme (Serial SGS)* described in Section 3.3.

3. *Schedule improvement.* This phase tries to improve the solution obtained in the previous phase using an improvement procedure called *Forward-Backward Improvement.*

4. *Chromosome adjustment.* This phase adjusts the chromosome genes given by the GA to reflect the solution obtained after the schedule improvement.

After a schedule is obtained, the corresponding measure of quality (*makespan*) is fed back to the genetic algorithm. Figure 4 illustrates the sequence of steps applied to each chromosome generated by the genetic algorithm. Each of these phases will be described in detail in the following sections.

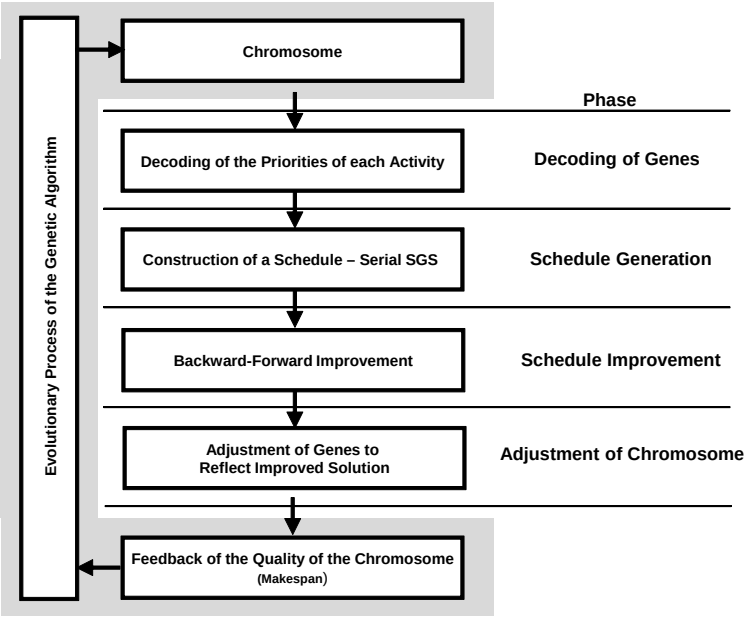

Figure 4: Architecture of the new approach.

3.2 Genetic algorithm

Genetic algorithms are adaptive methods that are used to solve search and optimization problems (Goldberg, 1989; Beasley et al., 1993). They are based on the genetic process of biological systems. 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 (1859). By mimicking this process, genetic algorithms, if suitably encoded, are able to *evolve* solutions to real-world problems. To build a genetic algorithm, an *encoding* (or *representation*) for the problem must first be devised. A *fitness function*, which assigns a figure of merit to each encoded solution, is also required. While executing the genetic algorithm, parents are *selected* for reproduction and *recombined* to generate offspring (see high-level pseudo-code in Figure 5).

It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as *genes*) are joined together to form a string of values (*chromosome*). In genetic terminology, the set of parameters represented by a particular chromosome is referred to as an *individual*. The fitness of an individual depends on its chromosome and is evaluated by the fitness function. The individuals, during the reproductive phase, are selected from the population and *recombined*, producing

```
procedure GENETIC-ALGORITHM
1   Generate initial population P_0;
2   Evaluate population P_0;
3   Initialize generation counter g ← 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 ← g + 1;
10  end while;
end GENETIC-ALGORITHM;
```

Figure 5: Pseudo-code of a standard genetic algorithm.

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

3.2.1 Chromosome representation

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

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

In the present context, the direct use of schedules as chromosomes is too complicated to represent and manipulate. In particular, it is difficult to develop corresponding crossover and mutation operations. Instead, solutions are represented indirectly by parameters that are later used by a schedule-generation scheme to obtain a solution. To build the solution, we use an active schedule generator described in Section 3.3.

Each solution chromosome is made of n genes, where n is the number of activities:

$$chromosome = (\underbrace{gene_1, \ldots, gene_n}_{\text{priorities}}).$$

The n genes are used to determine the priority of each of the n activities.

3.2.2 Decoding the priorities of the activities

The priorities of the activities are given directly by the genetic algorithm, i.e.

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

These priorities are used by the decoding algorithm presented in Section 3.3.

3.2.3 Evolutionary strategy

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

Given a current population, we perform the following three steps to obtain the next generation:

1. *Reproduction.* Some of the best individuals are copied from the current generation into the next (see TOP in Figure 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 local minima. Nevertheless, this can be overcome by using high mutation rates as described below.

2. *Crossover.* Regarding the crossover operator, parametrized uniform crossover (Spears and Dejong, 1991) is used as opposed to the traditional one-point or two-point crossover. Two individuals are randomly chosen to act as parents. One of the parents is chosen amongst the best individuals in the population (TOP in Figure 7), while the other is randomly chosen from the whole current population (including TOP). For each gene, a real random number in the interval $[0, 1]$ is generated. If the random number obtained is smaller than a threshold value, called *crossover probability* ($CProb > 0.5$), for example 0.7, then the allele of the first parent is used. Otherwise, the allele used is that of the second parent. An example of a crossover outcome is given in Figure 6.

3. *Mutation.* In this scheme, mutation is used in a broader sense than usual. The operator we define acts like a mutation operator and its purpose is to prevent premature convergence of the population. Instead of performing gene-by-gene mutation, with very small probability at each generation, we introduce some new individuals into the next generation (see BOT in Figure 7). These new individuals (mutants) are randomly generated from the same distribution as the original population and thus,

no genetic material of the current population is brought in. This process prevents premature convergence of the population, like in a mutation operator, and leads to a simple statement of convergence, i.e. if a sufficiently large number of generations is carried out, then the entire solution space will be sampled.

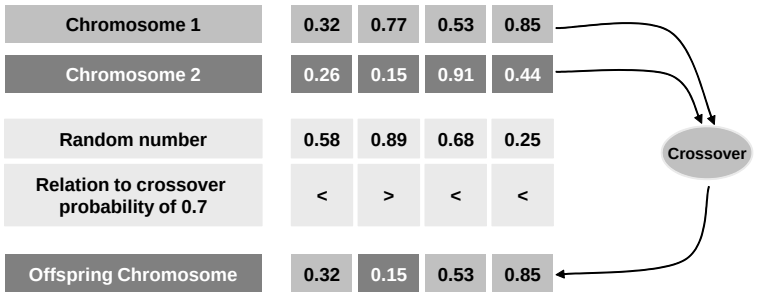

Figure 6: Example of parametrized uniform crossover with crossover probability equal to 0.7.

The initial population is randomly generated. Figure 7 depicts the transitional process between two consecutive generations.

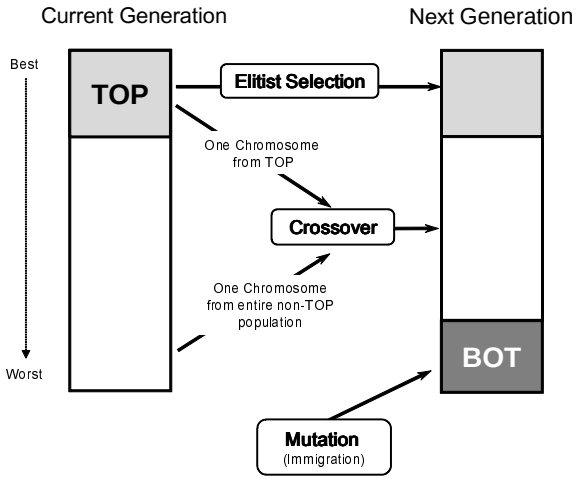

Figure 7: Transitional process between consecutive generations.

3.3 Schedule generation procedure

The procedure used to construct active schedules is based on a scheduling generation scheme which consists of $g = 1, ..., n$ stages, in each of which one activity is selected and scheduled at the earliest precedence and resource feasible completion time. There are two disjoint activity sets associated with each stage. The schedule set S_g, which includes all the activities that been already scheduled and the eligible set D_g, which comprises all

the activities eligible for scheduling. Let $A(t)$ represent the set of activities active a time t and let $RD_k(t)$ be remaining capacity of resource type k at time t, given by

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

The algorithmic description of the scheduling generation scheme used to build parametrized active schedules is shown in the pseudo-code in Figure 8. The initialization assigns a completion time of 0 to the dummy source (activity 0) and places it in the partial schedule. At the start of every step g, the eligible set D_g, the set of finish times Γ_g, and the remaining capacities $RD_k(t)$ are calculated. Step 4 selects from eligible set D_g the activity which has the highest priority (the priorities of the activities are supplied by the GA). Afterwards, the finish time of j is calculated by first computing earliest precedence feasible finish time EF_j and then calculating the earliest precedence and resource-feasible finish time LF_j within the interval $[EF_j\,,\,LF_j]$, where LF_j denotes the latest finish time as calculated by backward recursion (cf. Elmaghraby (1977)) from an upper bound of finish time T of the project. The makespan of the solution is given by the maximum of all predecessor activities of activity $n+1$, i.e. $F_{n+1} = \max\{F_l \mid l \in P_{n+1}\}$. The time complexity of the serial SGS presented in Figure 8 is $\mathcal{O}(n^2 \cdot K)$.

The basic idea consists in letting the GA evolve the priorities used in the selection step of the procedure (step 4),

$$j^* \leftarrow \text{argmax}\{PRIORITY_j \mid j \in D_g\},$$

i.e. the parameters $PRIORITY_j$ (priority of activity j used at each g) are supplied by the GA.

For $nGen$ generations this procedure generates

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

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

procedure CONSTRUCT - ACTIVE SCHEDULE

1 Initialize $F_0 = 0,\ S_0 = \{0\}$

2 **for** $g = 1$ **to** n **do**

3 Calculate D_g, Γ_g and $RD_k(t)$ $(k \epsilon K;\ t \epsilon \Gamma_g)$

4 *// select activity with highest priority*
· $j^* \leftarrow \text{argmax}\{PRIORITY_j \mid j \in D_g\}$

5 *// compute earliest finish time (in terms of precedence only)*
· $EF_{j^*} \leftarrow \max\{F_i \mid i \in P_j\} + d_{j^*}$

6 *// compute earliest finish time (in terms of precedence and capacity)*
· $F_{j^*} \leftarrow \min\{t \in [EF_{j^*} - d_{j^*},\ LF_j - d_{j^*}] \cap \Gamma_g \mid$
· $r_{j^*,k} \leq RD_k(\tau),\ k \in K \mid r_{j^*,k} > 0, \tau \in [t,\ t + d_{j^*}]\} + d_{j^*}$

7 *// update* S_g
· $S_g \leftarrow S_{g-1} \cup \{j^*\}$

8 **end for**

9 // compute makespan (equal to finish time of activity $n+1$)
· $F_{n+1} = \max_{j \in P_{n+1}} \{F_j\}$

end CONSTRUCT - ACTIVE SCHEDULE;

Figure 8: Pseudo-code for the active schedule construction procedure.

Figure 9 presents an example of the application of the serial SGS to the problem given in Figure 1. The numbers in bold for the row labeled D_g indicate the selected activity.

g	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
D_g	**1**, 3, 5, 8	3, **5**, 8, 2	3, 8, **2**	3, **8**	**3**	**4**, 6	**6**, 7	**7**	**9**
Act. Selected	**1**	**5**	**2**	**8**	**3**	**4**	**6**	**7**	**9**

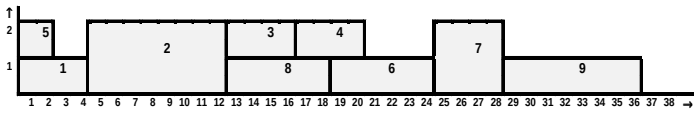

Figure 9: Example for serial SGS.

3.4 Forward-Backward Improvement

After obtaining a solution by the SGS proposed in Section 3.3 our heuristic attempts to reduce the makespan of the project through the use of a procedure called Forward–Backward Improvement (FBI). Forward-backward scheduling methods employ an SGS to iteratively schedule the project by alternating between forward and backward scheduling.

The FBI procedure employs an SGS to iteratively schedule the project by alternating between forward and backward scheduling. This multi-pass heuristic scheduling procedure was proposed by Li and Willis (1992). The forward and backward passes are based on the concepts of forward and backward free slack of the activities. The forward (backward) free slack of an activity in a feasible schedule is the amount of time that the activity can be shifted right (left) allowing the remaining activities to start on their scheduled dates. Neither forward nor backward free slack take into account resource constraints. Pseudo-code for forward scheduling is shown in Figure 8. Backward scheduling applies the procedure outlined in Figure 8 to the reverse precedence network where the former end activity $n + 1$ becomes the new start activity. The priority values used in step 4 of the algorithm are obtained from the start or completion times of the last generated schedule.

Figure 10 depicts the application of FBI to an initial solution given in Figure 10-a. In Figure 10-b all the activities are moved forward (to the right) reducing the makespan from 36 to 32 time units. In Figure 10-c all the activities are moved backward (to the left) reducing the makespan from 32 to 30 time units.

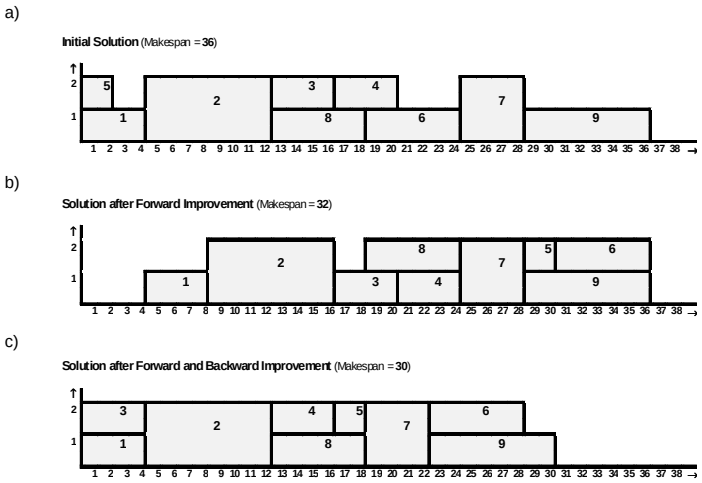

Figure 10: Example of Forward Backward Improvement

3.5 Chromosome Adjustment

The solutions produced by FBI are usually not in accordance with the priorities that were initially supplied by the GA chromosome. Since the GA has no knowledge of the changes in priorities that occur in the final solution, the heuristic adjusts the chromosome to reflect these changes. To make the chromosome supplied by the GA agree with the solution, the heuristic adjusts the order of the genes according to the starting times. Figure 11 shows an example of the adjustment process.

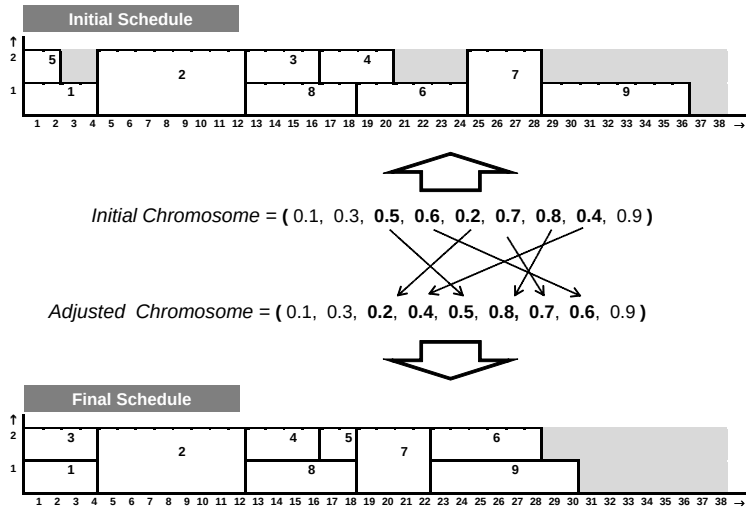

Figure 11: Example of the chromosome adjustment process.

4 Numerical Experiments

In this section, we report results obtained on a set of experiments conducted to evaluate the performance of the algorithm proposed in this paper. We call this algorithm *GA-FBI-PS* (***G***enetic ***A***lgorithm with ***F***orward ***B***ackward ***I***mprovement for ***P***roject ***S***cheduling). The algorithm was implemented using Microsoft Visual Basic 6.0 and the tests were carried out on a computer with a Intel Core 2 CPU running at 2.4 GHZ on the Windows XP operating system.

4.1 Benchmark Instances and Algorithms

To illustrate the effectiveness of *GA-FBI-PS* we consider a total of 1560 instances from three classes of standard RCPSP test problems: *J30* (480 instances each with 30 activities), *J60* (480 instances each with 60 activities), and *J120* (600 instances each with 120 activities). All problem instances require four resource types. Instance details are described by Kolisch et al. (1995) and can be obtained at `http://129.187.106.231/psplib/datasm.html`.

The proposed algorithm is compared with the following algorithms:

1. Local search-oriented approaches:
   - Fleszar and Hindi (2004)
   - Palpant et al. (2004)
2. Population-based approaches:
   - Debels et al. (2006)
   - Valls et al. (2004)
3. Problem and heuristic space method:
   - Leon and Ramamoorthy (1995)
4. Priority-rule based sampling methods:
   - Tormos and Lova (2003) – sampling LFT, FBI
   - Schirmer and Riesenberg (1998)
   - Kolisch and Drexl (1996)
   - Kolisch (1996b) – single pass LFT (serial)
   - Kolisch (1996b) – single pass LFT (parallel)
   - Kolisch (1996a;b) – single pass WCS
   - Kolisch (1995) – random (serial)
   - Kolisch (1995) – random (parallel)
5. Genetic algorithms:
   - Mendes et al. (2009) – BRKGA

- Valls et al. (2005) – GA – FBI
- Debels and Vanhoucke (2005) – GA – DBH
- Valls et al. (2003) – GA – hybrid, FBI
- Kochetov and Stolyar (2003) – GA, tabu search, path-relinking
- Hartmann (2002) – GA self adapting
- Hartmann (1998) – GA activity list
- Hartmann (1998) – GA random key
- Hartmann (1998) – GA priority rule

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

4.2 GA configuration

In our past experience with genetic algorithms based on the same evolutionary strategy (see e.g. Gonçalves and Almeida (2002), Gonçalves et al. (2005), Gonçalves and Resende (2004), Buriol et al. (2005), and Gonçalves (2006)), we obtained good results with values of TOP, BOT, and crossover probability ($CProb$) in the following ranges:

Parameter	Interval
TOP	0.10 – 0.20
BOT	0.15 – 0.30
$CProb$	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. To fine tune these parameters, we conducted a small pilot experiment with combinations of the following values $TOP \in \{0.10, 0.15, 0.20\}$, $BOT \in \{0.15, 0.20, 0.25, 0.30\}$, and $CProb \in \{0.70, 0.75, 0.80\}$. We tried population sizes with 1, 2, 5, 10, and 15 times the number of activities in the project.

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

Population Size	$10 \times$ number of activities in the problem
CProb	0.7
TOP	The chromosomes of the 10% best fit solutions from the previous population are copied to the next generation.
BOT	The number of mutant chromosomes that are randomly generated and added to the next generation is 20% of the population size.
Fitness	Makespan (to minimize)
Stopping Criterion	250 generations

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

4.3 Computational results

Tables 1 (for algorithms for which papers reported the number of schedules generated) and Table 2 (for algorithms for which papers did not report the number of schedules generated) summarize the average percentage deviation D_{OPT} from the optimal makespan for instance set *J30*. The best value obtained by *GA-FBI-PS* was $D_{OPT} = 0.01$. *GA-FBI-PS* obtains the optimal solution for 478 of the 480 instances (99.58% of the instances). The number of generated schedules as a function of the number of generations is equal to $10 \times 30 + (nGer - 1) \times (0.90 \times 5 \times 30)$. *GA-FBI-PS* ranks first for 5,000 schedules and ranks second for 50,000 schedules or more.

Table 1: Average percent deviations from optimal makespan – ProGen set $J30$.

	Maximum number of schedules / Average CPU time (s) Generations / (Number of schedules of *GA-FBI-PS*)				
	1000 / 0.36	**5000 /1.8**	**50000 /18**	**100000 / 36**	**500000 / 180**
Reference	1 / (900)	6 / (4950)	61 / (49500)	123 / (99720)	617 / (499860)
Kochetov and Stolyar (2003)	0.10	0.04	**0.00**	-	-
Mendes et al. (2009)	0.06	**0.02**	0.01	-	-
This paper	0.32	**0.02**	0.01	0.01	0.01
Debels et al. (2006)	0.27	0.11	0.01	0.01	0.01
Debels and Vanhoucke (2005)	0.15	0.04	0.02	-	-
Valls et al. (2003)	0.27	0.06	0.02	-	-
Valls et al. (2005)	0.34	0.20	0.02	-	-
Tormos and Lova (2003)	0.25	0.13	0.05	-	-
Nonobe and Ibaraki (2002)	0.46	0.16	0.05	-	-
Hartmann (2002)	0.38	0.22	0.08	-	-
Hartmann (1998)	0.54	0.25	0.08	-	-
Bouleimen and Lecocq (2003)	0.38	0.23	-	-	-
Schirmer and Riesenberg (1998)	0.65	0.44	-	-	-
Baar et al. (1998)	0.86	0.44	-	-	-
Kolisch and Drexl (1996)	0.74	0.53	-	-	-
Kolisch (1996b)	0.83	0.53	0.27	-	-
Hartmann (1998)	1.03	0.56	0.23	-	-
Kolisch (1995)	1.44	1.00	0.51	-	-
Hartmann (1998)	1.38	1.12	0.88	-	-
Kolisch (1996a;b)	1.40	1.28	-	-	-
Kolisch (1996b)	1.40	1.29	1.13	-	-
Kolisch (1995)	1.77	1.48	1.22	-	-
Leon and Ramamoorthy (1995)	2.08	1.59	-	-	-

Table 2: Average percent deviations from optimal makespan – ProGen set $J30$.

		CPU time		
Reference	**Avg. % deviation**	**Avg.**	**Max.**	**CPU freq.**
Palpant et al. (2004)	0.00	10.26s	123.0s	2.3 GHz
Fleszar and Hindi (2004)	0.01	0.64s	5.9s	1.0 GHz
Valls et al. (2003)	0.06	1.61s	6.2s	400 MHz
Valls et al. (2004)	0.10	1.16s	5.5s	400 MHz

Tables 3 (for algorithms for which papers reported the number of schedules generated) and Table 4 (for algorithms for which papers did not report the number of schedules generated) summarize the average percentage deviation from the well-known critical path-based lower bound (DLB) for instance set *J60*. These lower bound values are reported by

Stinson et al. (1978). For the instances of class *J60*, *GA-FBI-PS* obtained $DLB = 10.49$. The number of generated schedules as a function of the number of generations is equal to$10 \times 60 + (nGer - 1) \times (0.90 \times 5 \times 60)$. *GA-FBI-PS* ranks second for 5,000 schedules and ranks first for 50,000 schedules or more.

Table 3: Average percent deviations from critical path lower bound – ProGen set *J*60.

	Maximum number of schedules / Average CPU time (s) Generations / (Number of schedules of *GA-FBI-PS*)				
	1000 / 0.11	**5000 / 0.53**	**50000 / 5.25**	**100000 / 10.5**	**500000 / 52.5**
Reference	1 / (1800)	3 / (5040)	30 / (48700)	61 / (99000)	308 / (499140)
This paper	-	11.56	**10.57**	**10.51**	**10.49**
Mendes et al. (2009)	11.72	**11.04**	10.67	10.67	-
Debels and Vanhoucke (2005)	**11.45**	10.95	10.68	-	-
Debels et al. (2006)	11.73	11.10	10.71	-	10.53
Valls et al. (2003)	11.56	11.10	10.73	-	-
Kochetov and Stolyar (2003)	11.71	11.17	10.74	-	-
Valls et al. (2005)	12.21	11.27	10.74	-	-
Hartmann (2002)	12.21	11.70	11.21	-	-
Hartmann (1998)	12.68	11.89	11.23	-	-
Tormos and Lova (2003)	11.88	11.62	11.36	-	-
Bouleimen and Lecocq (2003)	12.75	11.90	-	-	-
Nonobe and Ibaraki (2002)	12.97	12.18	11.58	-	-
Schirmer and Riesenberg (1998)	12.94	12.58	-	-	-
Kolisch and Drexl (1996)	13.51	13.06	-	-	-
Baar et al. (1998)	13.80	13.48	-	-	-
Hartmann (1998)	14.68	13.32	12.25	-	-
Hartmann (1998)	13.30	12.74	12.26	-	-
Kolisch (1996b)	13.59	13.23	12.85	-	-
Kolisch (1996b)	13.96	13.53	12.97	-	-
Kolisch (1995)	14.89	14.30	13.66	-	-
Kolisch (1996a;b)	13.66	13.21	-	-	-
Kolisch (1995)	15.94	15.17	14.22	-	-
Leon and Ramamoorthy (1995)	14.33	13.49	-	-	-

Table 4: Average percent deviations from critical path lower bound – ProGen set $J60$.

		CPU time		
Reference	**Avg. % deviation**	**Avg.**	**Max.**	**CPU freq.**
Palpant et al. (2004)	10.81	38.8s	223.0s	2.3 GHz
Valls et al. (2004)	10.89	3.7s	22.6s	400 MHz
Valls et al. (2003)	11.45	2.8s	14.6s	400 MHz
Möhring et al. (2003)	15.60	6.9s	57s	200 MHz

Tables 5 (for algorithms for which papers report the number of schedules generated) and Table 6 (for algorithms for which papers do not report the number of schedules generated) summarize the average percentage deviation from the well-known critical path-based lower bound (DLB) for instance set *J120*. This lower bound values are reported by Stinson et al. (1978). For the *J120* instances, *GA-FBI-PS* obtained $DLB = 30.08$. The number of generated schedules as a function of the number of generations is equal to$10 \times 120 + (nGer - 1) \times (0.90 \times 5 \times 120)$. *GA-FBI-PS* ranks seventh for 1,000 schedules and ranks third for 5,000 and 50,000 schedules.

Table 5: Average percent deviations from critical path lower bound – ProGen set $J120$.

	Maximum number of schedules / Average CPU time (s) Generations / (Number of schedules of *GA-FBI-PS*)				
	1000 / 0.6	**5000 / 1.8**	**50000 / 18**	**100000 / 36**	**500000 / 180**
Reference	1 / (3600)	2 / (6840)	15 / (48960)	30 / (97560)	154 / (490320)
This paper	-	35.94	32.76	31.63	**30.08**
Debels and Vanhoucke (2005)	34.19	**32.34**	**30.82**	-	-
Valls et al. (2003)	**34.07**	32.54	31.24	-	-
Mendes et al. (2009)	35.87	33.03	31.44	**31.32**	31.20
Debels et al. (2006)	35.22	33.10	31.57	-	30.48
Valls et al. (2005)	35.39	33.24	31.58	-	-
Kochetov and Stolyar (2003)	34.74	33.36	32.06	-	-
Hartmann (2002)	37.19	35.39	33.21	-	-
Tormos and Lova (2003)	35.01	34.41	33.71	-	-
Hartmann (1998)	39.37	36.74	34.03	-	-
Bouleimen and Lecocq (2003)	42.81	37.68	-	-	-
Nonobe and Ibaraki (2002)	40.86	37.88	35.85	-	-
Hartmann (1998)	39.93	38.49	36.51	-	-
Schirmer and Riesenberg (1998)	39.85	38.70	-	-	-
Kolisch (1996b)	39.60	38.75	37.74	-	-
Kolisch (1996a;b)	39.65	38.77	-	-	-
Kolisch and Drexl (1996)	41.37	40.45	-	-	-
Leon and Ramamoorthy (1995)	42.91	40.69	-	-	-
Hartmann (1998)	45.82	42.25	38.83	-	-
Kolisch (1996b)	42.84	41.84	40.63	-	-
Kolisch (1995)	44.46	43.05	41.44	-	-
Kolisch (1995)	49.25	47.61	45.60	-	-

Table 6: Average percent deviations from critical path lower bound – ProGen set $J120$.

			CPU time		
Reference	**Avg. % deviation**		**Avg.**	**Max.**	**CPU freq.**
Valls et al. (2004)	31.58		59.4s	264.0s	400 MHz
Palpant et al. (2004)	32.41		207.9s	501.0s	2.3 GHz
Valls et al. (2003)	34.53		17.0s	43.9s	400 MHz
Möhring et al. (2003)	36.00		72.9s	654s	200 MHz

From the above results it is clear that no algorithm dominates *GA-FBI-PS*. The approach of Debels et al. (2006) is the one that seems to have similar performance. Given that *GA-FBI-PS* uses an evolutionary strategy that depends on the number of generations, it is not surprising that, for problems with large number of activities, it does

not perform so well when only a small number of schedules generated is allowed. With our heuristic, we improved[1] the best known solution for 11 instances in test problem repository PSPLIB `http://129.187.106.231/psplib/files/j120hrs.sm`.

5 Concluding remarks

This paper presented a biased random-keys genetic algorithm for the resource constrained project scheduling problem. The chromosome representation of the problem is based on random keys. The schedules are constructed using a priority rule in which the priorities are defined by the genetic algorithm. Schedules are constructed using a procedure that generates active schedules. The approach was tested on a set of 1560 standard instances taken from the literature and compared with results of 25 other algorithms taken from the literature. In extensive computational testing, our algorithm compared well with the other algorithms and produced new best known solutions for a number of benchmark test instances. Overall, the experiments validate the effectiveness of the proposed algorithm.

Acknowledgment

This work has been supported by funds granted by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006.

References

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

Baar, T., P. Brucker, and S. Knust: 1998, 'Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem'. In: S. Voss, S. Martello, I. Osman, and C. Roucairol (eds.): *Meta-heurisitics: Advances and trends in local search paradigms for optimization*. Kluwer, pp. 1–8.

Bean, J. C.: 1994, 'Genetics and random keys for sequencing and optimization'. *ORSA Journal on Computing* **6**, 154–160.

Beasley, D., D. R. Bull, and R. R. Martin: 1993, 'An overview of genetic algorithms: Part 1, Fundamentals'. *University Computing* **15**(2), 58–69. Department of Computing Mathematics, University of Cardiff, UK.

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

[1] As of March 8, 2009.

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

Boctor, F. F.: 1996, 'An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems'. *International Journal of Production Research* **34**, 2335–2351.

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

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

Brucker, P. and S. Knust: 2000, 'A linear programming and constraint propagation-based lower bound for the RCPSP'. *European Journal of Operational Research* **127**, 355–362.

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

Brucker, P. and S. Knust: 2006, *Complex scheduling*. Springer.

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

Buriol, L. S., M. G. C. Resende, C. C. Ribeiro, and M. Thorup: 2005, 'A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing'. *Networks* **46**(1), 36–56.

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

Cooper, D. F.: 1977, 'A note on serial and parallel heuristics for resource-constrained project scheduling'. *Foundations of Control Engineering* **2**, 131–133.

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

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

Debels, D., B. De Reyck, R. Leus, and M. Vanhoucke: 2006, 'A hybrid scatter search / electromagnetism meta-heuristic for project scheduling'. *European Journal of Operational Research* **169**, 638–653.

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

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

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

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

Elmaghraby, S.: 1977, *Activity networks: Project planning and control by network models*. Wiley.

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

Gagnon, M., F. F. Boctor, and G. d'Avignon: 2004, 'A tabu search algorithm for the resource-constrained project scheduling problem'. In: *Proceedings of Administrative Sciences Association of Canada Annual Conference (ASAC 2004)*.

Goldberg, D. E.: 1989, *Genetic algorithms in search optimization and machine learning*. Addison-Wesley.

Gonçalves, J. F.: 2006, 'A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem'. *European Journal of Operational Research*. To appear.

Gonçalves, J. F. and J. R. Almeida: 2002, 'A hybrid genetic algorithm for assembly line balancing'. *Jornal of Heuristics* **8**, 629–642.

Gonçalves, J. F., J. J. M. Mendes, and M. G. C. Resende: 2005, 'A hybrid genetic algorithm for the job shop scheduling problem'. *European Journal of Operational Research* **167**, 77–95.

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

Gonçalves, J. F. and M. G. C. Resende: 2009, 'Biased random-keys genetic algorithms for combinatorial optimization'. Technical report, AT&T Labs Research, Florham Park, New Jersey, USA.

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

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

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

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

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

Klein, R.: 1999, *Scheduling of resource-constrained projects*. Kluwer.

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

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

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

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

Kolisch, R.: 1995, *Project scheduling under resource constraints: Efficient heuristics for several problem classes*. Physica-Verlag.

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

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

Kolisch, R. and A. Drexl: 1996, 'Adaptative search for solving hard project scheduling problems'. *Naval Research Logistics* **43**, 43–23.

Kolisch, R. and S. Hartmann: 1999, 'Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis'. In: J. Weglarz (ed.): *Handbook on recent advances in project scheduling*. Kluwer, pp. 147–178.

Kolisch, R. and S. Hartmann: 2005, 'Experimental investigation of heuristics for resource-constrained project scheduling: An update'. *European Journal of Operational Research*. To appear.

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

Kolisch, R., A. Sprecher, and A. Drexl: 1995, 'Characterization and generation of a general class of resource-constrained project scheduling problems'. *Management Science* **41**, 1693–1703.

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

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

Leon, V. J. and B. Ramamoorthy: 1995, 'Strength and adaptability of problem-space based neighborhoods for resource constrained scheduling'. *Operations Research Spektrum* **17**, 173–182.

Li, K. Y. and R. J. Willis: 1992, 'An iterative scheduling technique for resource-constrained project scheduling'. *European Journal of Operational Research* **56**(3), 370 – 379.

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, Porto, Portugal. In Portuguese.

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

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

Möhring, R. H., A. S. Schulz, F. Stork, and M. Uetz: 2003, 'Solving project scheduling problems by minimum cut computations'. *Management Science* **49**, 330–350.

Nonobe, K. and T. Ibaraki: 2002, 'Formulation and tabu search algorithm for the resource constrained project scheduling problem'. In: C. C. Ribeiro and P. Hansen (eds.): *Essays and surveys in metaheuristics*. Kluwer Academic Publishers, pp. 557–588.

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

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

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

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

Spears, W. M. and K. A. Dejong: 1991, 'On the virtues of parameterized uniform crossover'. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. pp. 230–236.

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

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

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

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

Tormos, P. and A. Lova: 2001, 'A competitive heuristic solution technique for Resource-Constrained Project Scheduling'. *Annals of Operations Research* **102**, 65–81.

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

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

Valls, V., F. Ballestin, and M. S. Quintanilla: 2005, 'Justification and RCPSP: A technique that pays'. *European Journal of Operational Research* **165**, 375–386.

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