J Heuristics (2011) 17:467–486
DOI 10.1007/s10732-010-9142-2

A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem

José Fernando Gonçalves · Mauricio G.C. Resende · Jorge J.M. Mendes

Received: 11 March 2009 / Revised: 20 July 2010 / Accepted: 29 July 2010 / Published online: 31 August 2010

Abstract This paper presents a biased random-key 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.

Keywords Project management · Scheduling · Genetic algorithms · Random keys · Forward-backward improvement · Resource constrained project scheduling problem

Supported by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006.
AT&T Labs Research Technical Report.

J.F. Gonçalves
LIAAD, Faculdade de Economia do Porto, Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal
e-mail: jfgoncal@fep.up.pt

M.G.C. Resende (✉)
Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932, USA
e-mail: mgcr@research.att.com

J.J.M. Mendes
Instituto Superior de Engenharia do Porto, Instituto Politécnico do Porto, Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto, Portugal
e-mail: jjm@isep.ipp.pt

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 $j \in J$ to be scheduled after all its predecessor activities (i.e. all activities in set P_j) are completed;
2. Activities require resources with *limited capacities*.

While being processed, activity $j \in J$ requires $r_{j,k}$ units of resource type $k \in K$ during every time instant of its non-preemptable duration d_j. Resource type $k \in K$ has a limited capacity R_k at any point in time. 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 resource and precedence constraints, that minimizes the makespan C_{max} , i.e., the finish time of the last activity 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 Fig. 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

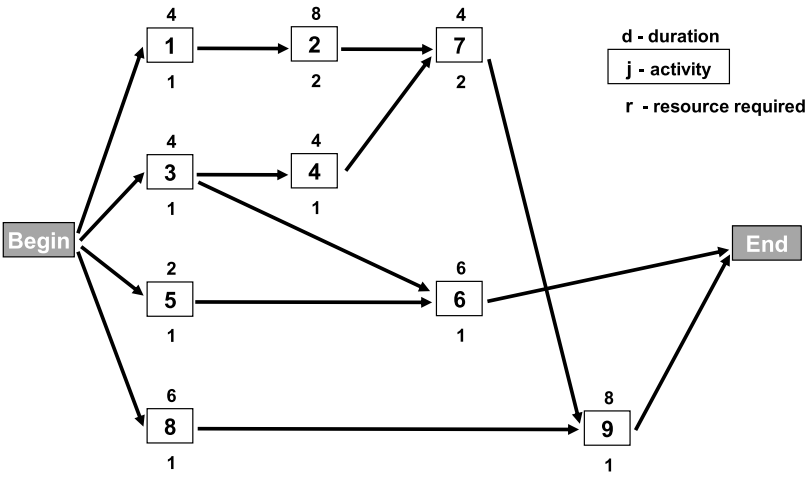

Fig. 1 Project network example. Activities are represented as boxes and precedences by directed arcs. Parameters d_j and r_j are given for each activity j

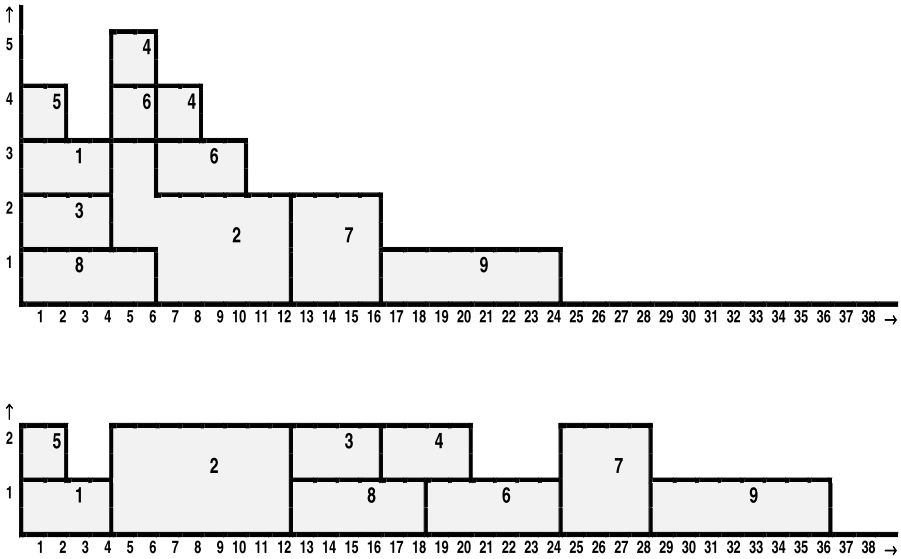

Fig. 2 Two solutions for the project network example of Fig. 1. *On top*, an infeasible schedule that ignores the resource constraint and results in a makespan of 24. *On the bottom*, a feasible schedule with an optimal makespan of 36

Herroelen (1997), Brucker et al. (1998), Klein and Scholl (1998a, 1998b), 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.

Blazewicz et al. (1983) showed 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 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 approach of Brucker and Knust 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 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 in 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, 1996b), 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; Valls et al. 2003, 2005, and Mendes et al. 2009), simulated annealing (Slowinski et al. 1994; Boctor 1996; 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-based approaches (Fleszar and Hindi 2004 and Palpant et al. 2004), and population-based approaches (Debels et al. 2006 and Valls et al. 2003).

Surveys are presented by Hartmann and Kolisch (2000), Kolisch and Padman (2001), and Demeulemeester and Herroelen (2002). Kolisch and Hartmann (2006) and Brucker and Knust (2006) describe models and algorithms for complex scheduling problems and discuss the RCPSP.

In this paper, we present a new biased random-key genetic algorithm for finding optimal or near optimal solutions for the resource constrained project scheduling problem. The remainder of the paper is organized as follows. Section 2 presents the new approach based on a biased random-key genetic algorithm (Gonçalves and Resende 2009), a schedule generation procedure, and an improvement procedure. Section 3 reports experimental results. Concluding remarks are made in Sect. 4.

2 The new approach

2.1 Overview of the new approach

The new approach proposed in this paper combines a biased random-key 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 Sect. 2.3.
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 genetic algorithm to reflect the solution obtained after the schedule improvement.

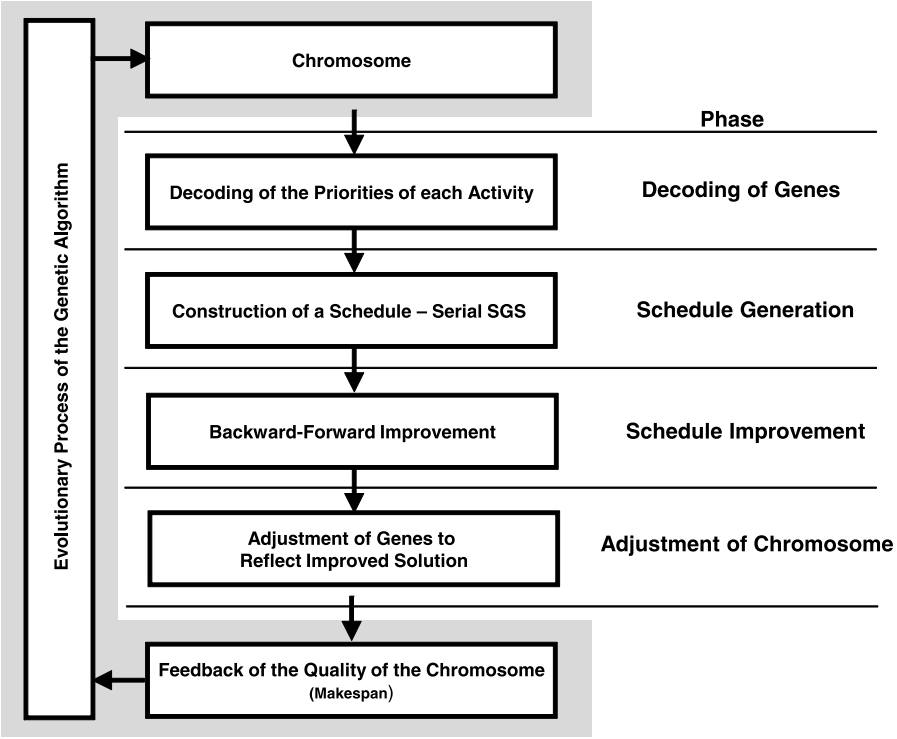

Fig. 3 Architecture of the new approach

After a schedule is obtained, the corresponding measure of quality (*makespan*) is fed back to the genetic algorithm. Figure 3 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.

2.2 Biased random-key genetic algorithm

This section presents the chromosome representation, the decoding procedure, and the evolutionary process of the genetic algorithm.

A chromosome represents a solution to the problem. 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. Such a solution is called a *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.

The genetic algorithm described in this paper uses a random-key alphabet 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 resulting 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.

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

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

These priorities are used by the decoding algorithm presented in Sect. 2.3.

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 Fig. 5). 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.
2. *Crossover.* Parametrized uniform crossover (Spears and Dejong 1991) is used as opposed to the traditional one-point or two-point crossover. After two parents are chosen, the first chosen randomly from the *TOP* (unlike Bean 1994 we always choose one parent from *TOP* (Gonçalves and Resende 2009)) and the second chosen randomly from the entire old population (including chromosomes copied to the next generation in the elitist selection). 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*), then the allele of the first parent is inherited by the offspring solution. Otherwise, the inherited allele is that of the second parent. An example of a crossover outcome is given in Fig. 4.
3. *Mutation.* In biased random-key genetic algorithms, 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 to local minima. Instead of performing gene-by-gene mutation at each generation with very small probability, we introduce a few new individuals into the next generation (represented by *BOT* in Fig. 5). These new individuals (called *mutants*) are randomly generated from the same distribution as the original population. Therefore, no genetic material of the current population is brought in. This process prevents premature convergence of the population and leads to a simple statement of convergence, i.e. if a sufficiently large number of generations are carried out, then the entire solution space will be sampled.

The initial population is randomly generated. Figure 5 depicts the transitional process between two consecutive generations.

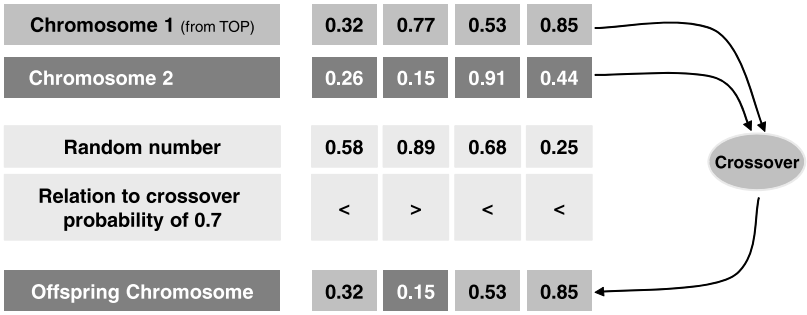

Fig. 4 Example of parametrized uniform crossover with crossover probability equal to 0.7

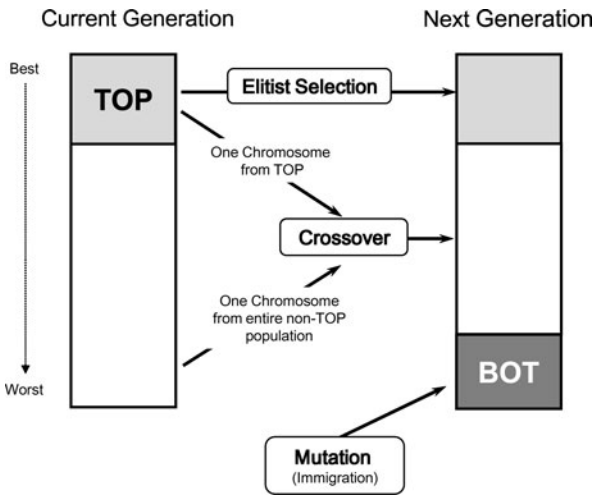

Fig. 5 Transitional process between consecutive generations

2.3 Schedule generation procedure

The procedure used to construct active schedules is based on a scheduling generation scheme that consists of $g = 1, \ldots, n$ 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 the activities not scheduled for which all predecessor activities have already been scheduled. Let $A(t)$ represent the set of activities that are active a time t and let $RD_k(t)$ be the 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 Fig. 6. The initialization assigns a completion time of 0 to the dummy source (activity 0) and places it

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 $RD_k(t)$ $(k \in K;\ t \in \Gamma_g)$
4 *// select activity with highest priority*
· $j^* \leftarrow \text{argmin}\{PRIORITY_j \mid j \in D_g\}$
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)*
· $F_{j^*} \leftarrow \min\{t \in [EF_{j^*} - d_{j^*},\ LF_j - d_{j^*}] \cap \Gamma_g \mid .$
· $.r_{j^*,k} \leq RD_k(\tau),\ k \in K,\ \tau \in [t,\ t + d_{j^*}]\} + d_{j^*}$
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;

Fig. 6 Pseudo-code for the active schedule construction procedure

in the partial schedule. At the start of every step g, the eligible set D_g, the set of finish times for each scheduled activity Γ_g, and the remaining capacities $RD_k(t)$ are computed. Step 4 selects from eligible set D_g the activity that has the highest priority (recall that the priorities of the activities are supplied by the genetic algorithm). Afterwards, the finish time of activity j is calculated by first computing the earliest precedence feasible finish time EF_j and then calculating the earliest precedence and resource feasible finish time F_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. Note that when checking for the availability of capacity we only need to check for the time periods in Γ_g because between activity finish times the capacity availability remains unchanged. 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 Fig. 6 is $\mathcal{O}(n^2 \cdot K)$.

The genetic algorithm evolves the priorities used in the selection step of the procedure (step 4),

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

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

Figure 7 presents an example of the application of the serial SGS to the problem given in Fig. 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**

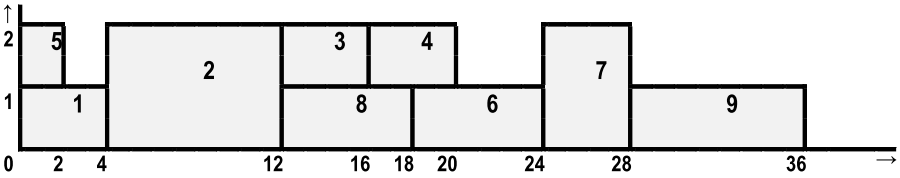

Fig. 7 Example of serial SGS

2.4 Forward-backward improvement

After a solution is obtained by the SGS proposed in Sect. 2.3, our heuristic attempts to reduce the makespan of the project through the use of a procedure called *forward–backward improvement* (FBI).

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. Pseudo-code for backward scheduling is shown in Fig. 6.

As mentioned in Sect. 2.3, the initial application of procedure CONSTRUCT ACTIVE SCHEDULE uses the priority values supplied by the genetic algorithm. To execute the forward pass, procedure CONSTRUCT ACTIVE SCHEDULE is applied to the reverse precedence network in which the former end activity $n + 1$ becomes the new start activity and the priority values of each activity are made equal to the makespan minus the completion time of the previous schedule. A final backward pass is then executed by applying procedure CONSTRUCT ACTIVE SCHEDULE to the original precedence network with the priority values of each activity made equal to the makespan minus the completion time of the previous (forward pass) schedule.

Figure 8 depicts the application of FBI to an initial solution. In Fig. 8(b) all the activities are moved forward (to the right) reducing the makespan from 36 to 32 time units. In Fig. 8(c) all the activities are moved backward (to the left) reducing the makespan from 32 to 30 time units.

2.5 Chromosome adjustment

The solutions produced by FBI are usually not in agreement with the priorities 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

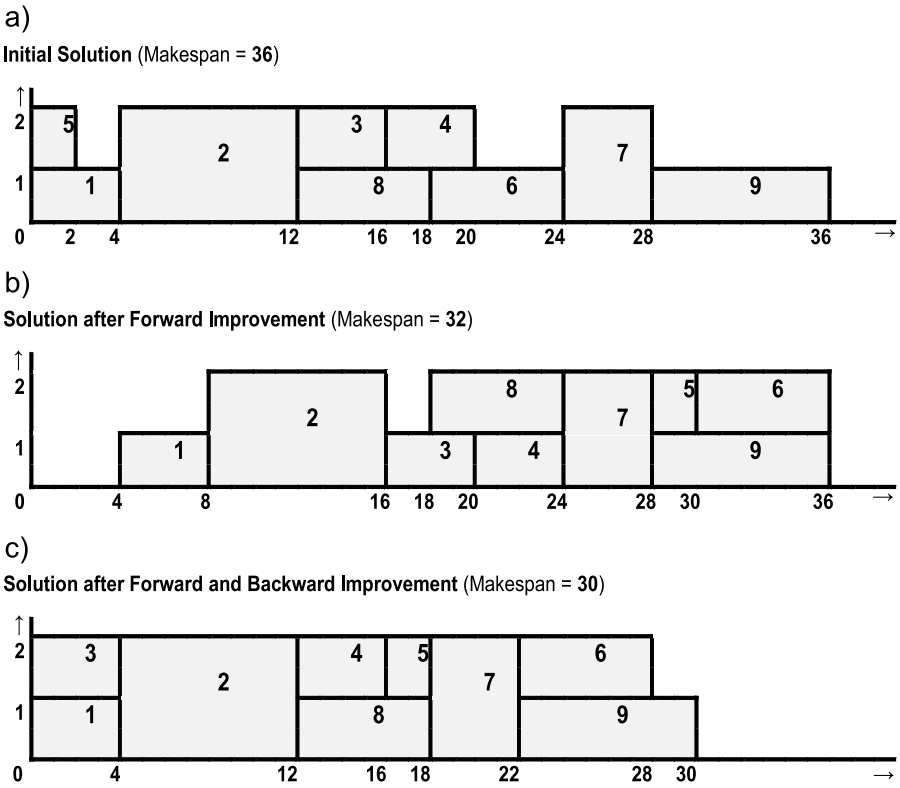

Fig. 8 Example of forward backward improvement

times. Figure 9 shows an example of the adjustment process. This chromosome adjustment improves not only the quality of the solutions but also decreases the number of iterations necessary to obtain the best values.

3 Experimental results

In this section, we report results obtained on a set of experiments conducted to evaluate the performance of the genetic algorithm proposed in this paper. We call this algorithm *GA-FBI*. 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.

3.1 Benchmark instances and algorithms

To illustrate the effectiveness of *GA-FBI* 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

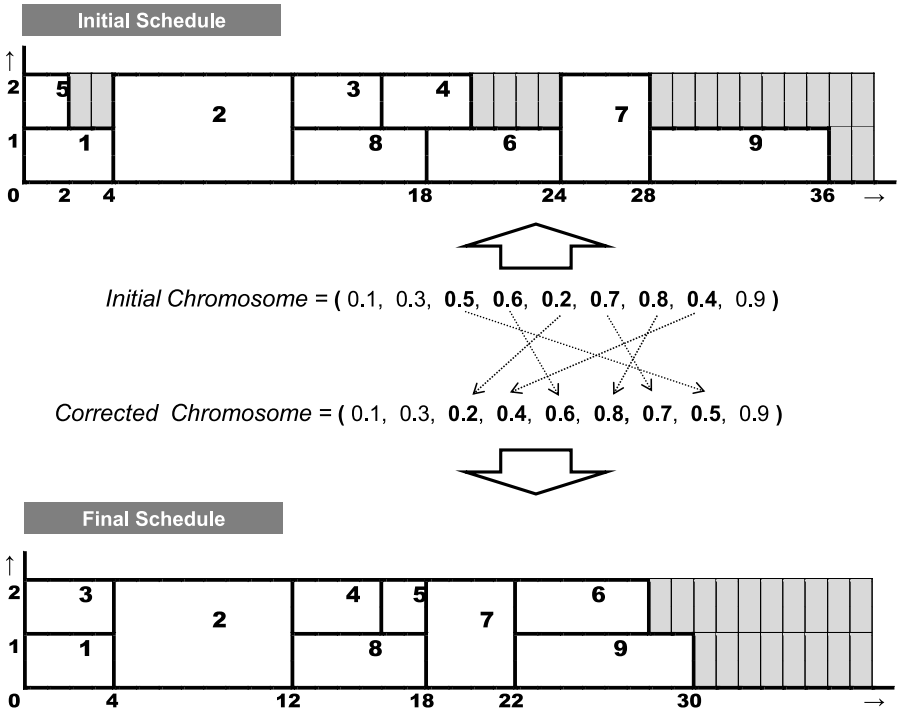

Fig. 9 Example of the chromosome adjustment process

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, 1996b)—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.

3.2 GA configuration

In our past experience with biased random-key genetic algorithms (see e.g. Gonçalves and Almeida 2002; Gonçalves et al. 2005; Gonçalves and Resende 2004; Buriol et al. 2005, and Gonçalves 2007), 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

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 obtain good results by using a population size proportional to the number of activities. We therefore experimented with population sizes having 1, 2, 5, 10, and 15 times the number of activities in the project.

The pilot experiment resulted in the following configuration, which was held constant for all experiments and all problem instances:

Population Size	10 × 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 randomly generated and added to the next generation is 20% of the population size.
Fitness	Makespan (to minimize)
Stopping Criterion	Maximum number of generations

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

3.3 Results

In our results we compare the solutions obtained by the algorithms as a function of the number of schedules generated. In the case of the *GA-FBI* the number of schedules generated is given by

$$3 \times [PopSize + (nGen - 1) \times PopSize \times (1 - TOP)]$$

where *nGen* is the number of generations, *PopSize* is the population size, and *TOP* is the proportion of the previous population copied to the next generation. Notice that we multiply by three to account for one schedule produced by the GA and two

Table 1 Average percent deviations from optimal makespan—ProGen set *J*30

Reference	Maximum number of schedules/Average CPU time(s) Generations / (Number of schedules of GA-FBI)				
	1000/0.36 1/(900)	5000/1.8 6/(4950)	50,000/18 61/(49,500)	100,000/36 123/(99,720)	500,000/180 617/(499,860)
This paper	0.32	**0.02**	0.01	0.01	0.01
Kochetov and Stolyar (2003)	0.10	0.04	**0.00**	–	–
Mendes et al. (2009)	**0.06**	**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, 1996b)	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 *J*30

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

schedules produced by the FBI procedure. A single run of *GA-FBI* is performed per problem instance.

Table 1 (for algorithms in which papers report the number of schedules generated) and Table 2 (for algorithms in which papers do not report the number of sched-

Table 3 Average percent deviations from critical path lower bound—ProGen set *J*60

Reference	Maximum number of schedules/Average CPU time(s) Generations / (Number of schedules of GA-FBI)				
	1000/0.11 1/(800)	5000/0.53 3/(5040)	50,000/5.25 30/(48,780)	100,000/10.5 61/(99,000)	500,000/52.5 308/(499,140)
This paper	–	11.56	**10.57**	**10.51**	**10.49**
Mendes et al. (2009)	11.72	11.04	10.67	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, 1996b)	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 *J*60

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

ules generated) summarize the average percentage deviation D_{OPT} from the optimal makespan for instance set *J30*. The best value obtained by *GA-FBI* was $D_{OPT} = 0.01$. *GA-FBI* obtains the optimal solution for 478 of the 480 instances (99.58% of the in-

Table 5 Average percent deviations from critical path lower bound—ProGen set *J*120

Reference	Maximum number of schedules/Average CPU time(s) Generations / (Number of schedules of GA-FBI)				
	1000/0.6 1/(3600)	5000/1.8 3/(6840)	50,000/18 15/(48960)	100,000/36 30/(97560)	500,000/180 154/(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, 1996b)	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 *J*120

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

stances). *GA-FBI* ranks first for 5,000 schedules and ranks second for 50,000 schedules or more.

Table 3 (for algorithms in which papers report the number of schedules generated) and Table 4 (for algorithms in which papers do not report the number of schedules generated) summarize the average percentage deviation from the well-known critical

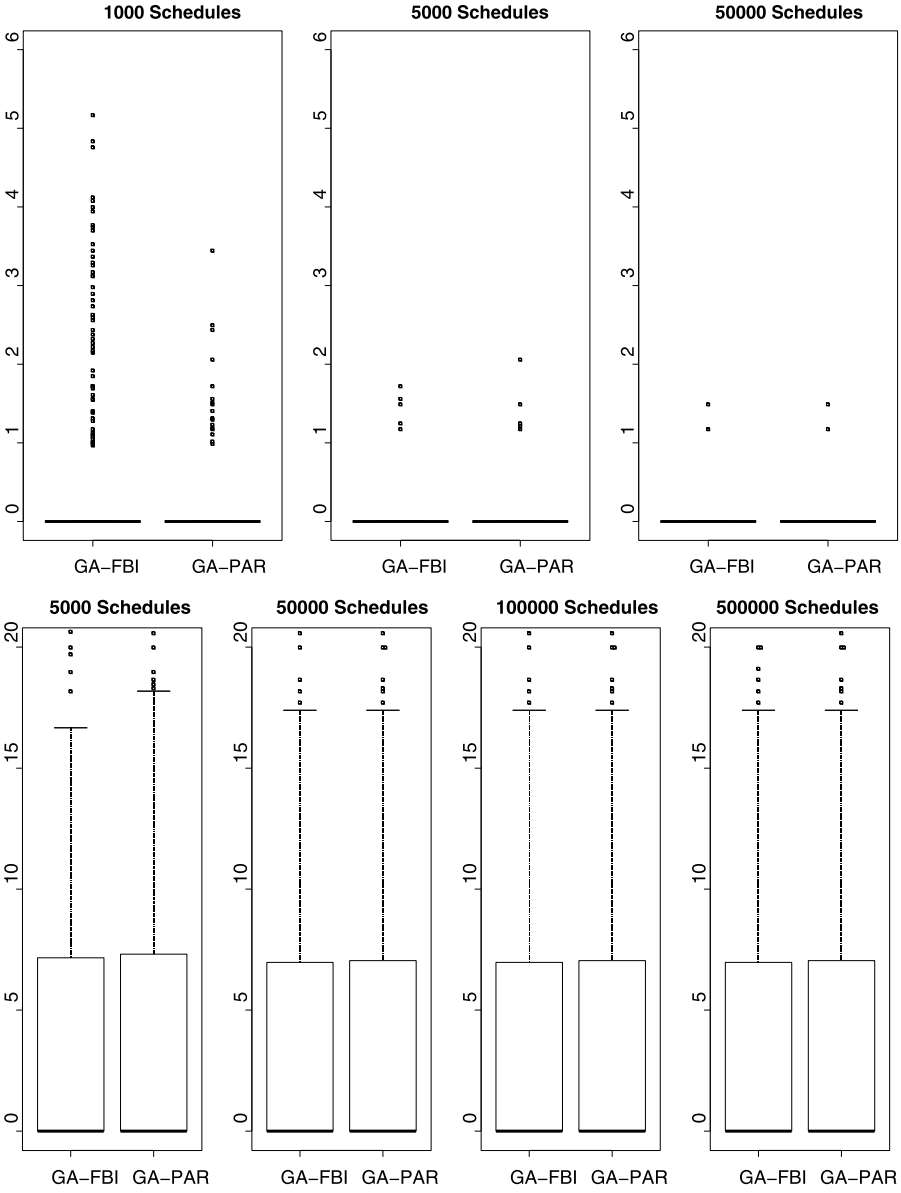

Fig. 10 Box-plots comparing average percent deviation from optimal or lower bound for GA-FBI and GA-PAR

path-based lower bound (*DLB*) for instance set *J60* (Stinson et al. 1978). *GA-FBI* obtained $DLB = 10.49$. *GA-FBI* ranks seventh for 5,000 schedules and ranks first for 50,000 schedules or more.

Table 5 (for algorithms in which papers report the number of schedules generated) and Table 6 (for algorithms in which papers do not report the number of schedules

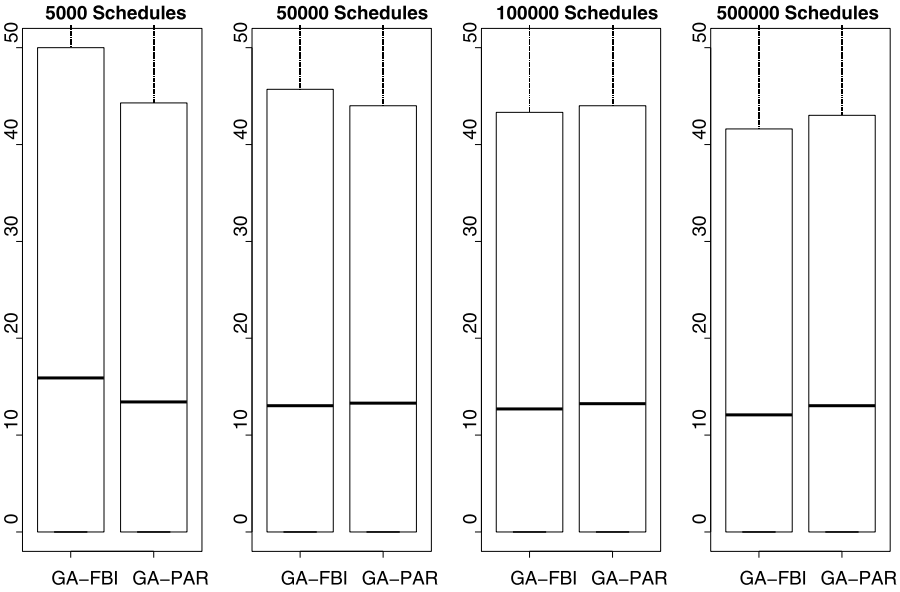

Fig. 10 (*Continued*)

generated) summarize the average percentage deviation from the well-known critical path-based lower bound (*DLB*) for instance set *J120* (Stinson et al. 1978). *GA-FBI* obtained $DLB = 30.08$. *GA-FBI* ranks ninth for 5,000 schedules, seventh for 50,000 schedules and first for 500,000 schedules.

Figure 10 presents a box-plot comparison between *GA-FBI* and *GA-PAR*, the biased random-key genetic algorithm of Mendes et al. (2009). The encoding of a solution in *GA-PAR* consists of two parts, both of which are used by a novel SGS parametrized active scheduler to construct schedules. On the other hand, in *GA-FBI*, chromosome has only one part, directly used by a serial SGS to construct active schedules, followed by an improvement phase (FBI) and a chromosome adjustment phase. This chromosome adjustment cannot be done in *GA-PAR* because of the encoding and the adopted scheduler. As can be seen in Fig. 10 for the J120 class of instances, as the number of allowed schedules increases, heuristic *GA-FBI* becomes better than *GA-PAR*.

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

[1] As of March 8, 2009.

4 Concluding remarks

This paper presents a biased random-key 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 is 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.

Acknowledgements 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., Tamarit, J.M.: Heuristic algorithms for resource-constrained project scheduling: a review and empirical analysis. In: Slowinski, R., Weglarz, J. (eds.) Advances in Project Scheduling, pp. 113–134. Elsevier, Amsterdam (1989)

Baar, T., Brucker, P., Knust, S.: Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In: Voss, S., Martello, S., Osman, I., Roucairol, C. (eds.) Meta-Heuristics: Advances and Trends in Local Search paradigms for optimization, pp. 1–8. Kluwer, Dordrecht (1998)

Bean, J.C.: Genetics and random keys for sequencing and optimization. ORSA J. Comput. **6**, 154–160 (1994)

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G.: Scheduling subject to resource constraints: classification and complexity. Discrete Appl. Math. **5**, 11–24 (1983)

Boctor, F.F.: Some efficient multi-heuristic procedures for resource-constrained project scheduling. Eur. J. Oper. Res. **49**, 3–13 (1990)

Boctor, F.F.: An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems. Int. J. Prod. Res. **34**, 2335–2351 (1996)

Bouleimen, K., Lecocq, H.: A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. Eur. J. Oper. Res. **149**, 268–281 (2003)

Brucker, P., Knust, S.: Lower bounds for resource-constrained project scheduling problems. Eur. J. Oper. Res. **149**, 302–313 (2003)

Brucker, P., Knust, S.: Complex Scheduling. Springer, Berlin (2006)

Brucker, P., Knust, S., Schoo, A., Thiele, O.: A branch and bound algorithm for the resource-constrained project scheduling problem. Eur. J. Oper. Res. **107**, 272–288 (1998)

Brucker, P., Knust, S.: A linear programming and constraint propagation-based lower bound for the RCPSP. Eur. J. Oper. Res. **127**, 355–362 (2000)

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

Cooper, D.F.: Heuristics for scheduling resource-constrained projects: an experimental investigation. Manag. Sci. **22**, 1186–1194 (1976)

Cooper, D.F.: A note on serial and parallel heuristics for resource-constrained project scheduling. Found. Control Eng. **2**, 131–133 (1977)

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

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

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M.: A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. Eur. J. Oper. Res. **169**, 638–653 (2006)
Demassey, S., Artigues, C., Michelon, P.: Constraint-propagation-base cutting planes: an application to the resource-constrained project scheduling problem. INFORMS J. Comput. **17**, 52–65 (2005)
Demeulemeester, E., Herroelen, W.: New benchmark results for the resource-constrained project scheduling problem. Manag. Sci. **43**, 1485–1492 (1997)
Demeulemeester, E., Herroelen, W.: Project Scheduling—A Research Handbook. Kluwer Academic, Dordrecht (2002)
Elmaghraby, S.: Activity Networks: Project Planning and Control by Network Models. Wiley, New York (1977)
Fleszar, K., Hindi, K.S.: Solving the resource-constrained project scheduling problem by a variable neighbourhood search. Eur. J. Oper. Res. **155**, 402–413 (2004)
Gagnon, M., Boctor, F.F., d'Avignon, G.: A tabu search algorithm for the resource-constrained project scheduling problem. In: Proceedings of Administrative Sciences Association of Canada Annual Conference (ASAC 2004) (2004)
Goldberg, D.E.: Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley, Reading (1989)
Gonçalves, J.F.: A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. Eur. J. Oper. Res. **183**, 1212–1229 (2007)
Gonçalves, J.F., Almeida, J.R.: A hybrid genetic algorithm for assembly line balancing. J. Heuristics **8**, 629–642 (2002)
Gonçalves, J.F., Resende, M.G.C.: An evolutionary algorithm for manufacturing cell formation. Comput. Ind. Eng. **47**, 247–273 (2004)
Gonçalves, J.F., Resende, M.G.C.: Biased random key genetic algorithms for combinatorial optimization. Tech. rep., AT&T Labs Research Technical Report, Florham Park, NJ 07733, USA. J. Heuristics (2009, to appear)
Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. Eur. J. Oper. Res. **167**, 77–95 (2005)
Hartmann, S.: A competitive genetic algorithm for resource-constrained project scheduling. Nav. Res. Logist. **45**, 279–302 (1998)
Hartmann, S.: A self-adapting genetic algorithm for project scheduling under resource constraints. Nav. Res. Logist. **49**, 433–448 (2002)
Hartmann, S., Kolisch, R.: Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur. J. Oper. Res. **127**, 394–407 (2000)
Klein, R., Scholl, A.: Progress: optimally solving the generalized resource-constrained project scheduling problem. Tech. rep., University of Technology, Darmstadt (1998a)
Klein, R., Scholl, A.: Scattered branch and bound: an adaptive search strategy applied to resource-constrained project scheduling problem. Tech. rep., University of Technology, Darmstadt (1998b)
Kochetov, Y., Stolyar, A.: Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the 3rd International Workshop of Computer Science and Information Technologies (2003)
Kohlmorgen, U., Schmeck, H., Haase, K.: Experiences with fine-grained parallel genetic algorithms. Ann. Oper. Res. **90**, 203–219 (1999)
Kolisch, R.: Project Scheduling under Resource Constraints: Efficient Heuristics for Several Problem Classes. Physica-Verlag, Heidelburg (1995)
Kolisch, R.: Efficient priority rules for the resource-constrained project scheduling problem. J. Oper. Manag. **14**, 179–192 (1996a)
Kolisch, R.: Serial and parallel resource-constrained project scheduling methods revisite: theory and computation. Eur. J. Oper. Res. **90**, 320–333 (1996b)
Kolisch, R., Drexl, A.: Adaptive search for solving hard project scheduling problems. Nav. Res. Logist. **43**, 43–23 (1996)
Kolisch, R., Hartmann, S.: Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (ed.) Handbook on recent advances in project scheduling, pp. 147–178. Kluwer, Dordrecht (1999)
Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur. J. Oper. Res. **174**, 23–37 (2006)
Kolisch, R., Padman, R.: An integrated survey of deterministic project scheduling. Int. J. Manag. Sci. **29**, 249–272 (2001)

Kolisch, R., Sprecher, A., Drexl, A.: Characterization and generation of a general class of resource-constrained project scheduling problems. Manag. Sci. **41**, 1693–1703 (1995)
Lawrence, S.R.: Resource constrained project scheduling—a computational comparison of heuristic scheduling techniques. Tech. rep., Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh (1985)
Lee, J.K., Kim, Y.D.: Search heuristics for resource constrained project scheduling. J. Oper. Res. Soc. **47**, 678–689 (1996)
Leon, V.J., Ramamoorthy, B.: Strength and adaptability of problem-space based neighborhoods for resource constrained scheduling. OR Spektrum **17**, 173–182 (1995)
Li, K.Y., Willis, R.J.: An iterative scheduling technique for resource-constrained project scheduling. Eur. J. Oper. Res. **56**(3), 370–379 (1992). doi:10.1016/0377-2217(92)90320-9
Mendes, J.J.M., Gonñalves, J.F., Resende, M.G.C.: A random key based genetic algorithm for the resource constrained project scheduling problem. Comput. Oper. Res. **36**, 92–109 (2009)
Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L.: An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. Manag. Sci. **44**, 714–729 (1998)
Möhring, R.H., Schulz, A.S., Stork, F., Uetz, M.: Solving project scheduling problems by minimum cut computations. Manag. Sci. **49**, 330–350 (2003)
Nonobe, K., Ibaraki, T.: Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Ribeiro, C.C., Hansen, P. (eds.) Essays and Surveys in Metaheuristics, pp. 557–588. Kluwer Academic, Dordrecht (2002)
Palpant, M., Artigues, C., Michelon, P.: LSSPER: solving the resource–constrained project scheduling problem with large neighbourhood search. Ann. Oper. Res. **131**, 237–257 (2004)
Pinson, E., Prins, C., Rullier, F.: Using tabu search for solving the resource constrained project scheduling problem. Tech. rep., Universite Catholique de l'Ouest, Angers (1994)
Schirmer, A., Riesenberg, S.: Case-based reasoning and parameterized random sampling for project scheduling. Tech. rep., University of Kiel, Germany (1998)
Slowinski, R., Soniewicki, B., Weglarz, J.: DSS for multiobjective project scheduling. Eur. J. Oper. Res. **79**, 220–229 (1994)
Spears, W.M., Dejong, K.A.: On the virtues of parameterized uniform crossover. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236 (1991)
Sprecher, A.: Scheduling resource-constrained projects competitively at modest memory requirements. Manag. Sci. **46**, 710–723 (2000)
Stinson, J.P., Davis, E.W., Khumawala, B.M.: Multiple resource-constrained scheduling using branch and bound. AIIE Trans. **10**, 252–259 (1978)
Stork, F., Uetz, M.: On the generation of circuits and minimal forbidden sets. Math. Program. **102**, 185–203 (2005)
Thomas, P.R., Salhi, S.: A tabu search approach for the resource constrained project scheduling problem. J. Heuristics **4**, 123–139 (1998)
Tormos, P., Lova, A.: A competitive heuristic solution technique for resource-constrained project scheduling. Ann. Oper. Res. **102**, 65–81 (2001)
Tormos, P., Lova, A.: Integrating heuristics for resource constrained project scheduling: one step forward. Tech. rep., Department of Statistics and Operations Research, Universidad Politecnica de Valencia (2003)
Valls, V., Ballestin, J., Quintanilla, M.S.: A hybrid genetic algorithm for the RCPSP. Tech. rep., Department of Statistics and Operations Research, University of Valencia (2003)
Valls, V., Ballestin, F., Quintanilla, M.S.: A population-based approach to the resource-constrained project scheduling problem. Ann. Oper. Res. **131**, 305–324 (2004)
Valls, V., Ballestin, F., Quintanilla, M.S.: Justification and RCPSP: a technique that pays. Eur. J. Oper. Res. **165**, 375–386 (2005)