Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 185 (2008) 495–508

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Discrete Optimization

A hybrid genetic algorithm for the resource-constrained project scheduling problem [☆]

Vicente Valls [a], Francisco Ballestín [b], Sacramento Quintanilla [c,*]

[a] *Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain*

[b] *Dpto. de Estadística e Investigación Operativa, Facultad de Ciencias Económicas y Empresariales, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006 Pamplona, Spain*

[c] *Dpto. de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Avda. de los Naranjos, s/n, Edificio Departamental Oriental, 46071 Valencia, Spain*

Received 23 September 2004; accepted 5 December 2006
Available online 25 January 2007

Abstract

In this paper we propose a Hybrid Genetic Algorithm (HGA) for the Resource-Constrained Project Scheduling Problem (RCPSP). HGA introduces several changes in the GA paradigm: a crossover operator specific for the RCPSP; a local improvement operator that is applied to all generated schedules; a new way to select the parents to be combined; and a two-phase strategy by which the second phase re-starts the evolution from a neighbour's population of the best schedule found in the first phase. The computational results show that HGA is a fast and high quality algorithm that outperforms all state-of-the-art algorithms for the RCPSP known by the authors of this paper for the instance sets j60 and j120. And that it is competitive with other state-of-the-art heuristics for the instance set j30.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Project management; Scheduling; Heuristics

1. Introduction

The Resource-Constrained Project Scheduling Problem (RCPSP) may be stated as follows: A project consists of a set activities $V = \{1, \ldots, n\}$ where each activity has to be processed without interruption. The dummy activities 1 and n represent the beginning and end of the project. The duration of an activity j is denoted by d_j where $d_1 = d_n = 0$. There are K renewable resource types. The availability of each resource type k in each time period is R_k units, $k = 1, \ldots, K$. Each activity j requires r_{jk} units of resource k during each period of its duration where $r_{1k} = r_{nk} = 0$, $k = 1, \ldots, K$. All parameters are assumed to be non-negative integer valued. There are precedence relations of the finish–start

[☆] This research was partially supported by the Ministerio de Ciencia y Tecnología under contract TIC2002-02510 and the Agencia Valenciana de Ciencia y Tecnología, GRUPOS03/174.

[*] Corresponding author. Tel.: +34 963828396; fax: +34 963828370.

E-mail addresses: Vicente.Valls@uv.es (V. Valls), Francisco.Ballestin@unavarra.es (F. Ballestín), Maria.Quintanilla@uv.es (S. Quintanilla).

type with a zero parameter value defined between the activities. $S_j(P_j)$ is the set of successors (predecessors) of activity j. It is assumed that $1 \in P_j$, $j = 2, \ldots, n$, and $n \in S_j$, $j = 1, \ldots, n-1$. The objective of the RCPSP is to find a schedule S of the activities, i.e., a set of starting times $(s_1, s_2 \ldots, s_n)$ where $s_1 = 0$ and the precedence and resource constraints are satisfied. In this way, the schedule duration $T(S) = s_n$ can be minimised.

The RCPSP is an important and challenging problem that has been widely studied over the past few decades. We refer to the surveys provided by Herroelen et al. (1998), Brucker et al. (1999) and Kolisch and Padman (2001), and the books on Project Scheduling by Weglarz (1999) and Demeulemeester and Herroelen (2002). As a job shop generalisation, the RCPSP is NP-hard in the strong sense (see Blazewicz et al., 1983). Only small-sized problem instances with up to 60 activities can be solved exactly in a satisfactory manner, at least for the KSD set (Kolisch et al., 1995). Therefore, heuristic solution procedures remain as the only feasible method of handling practical resource-constrained project scheduling problems. Recent overviews of heuristic procedures for the RCPSP can be found in Kolisch and Hartmann (1999), Hartmann and Kolisch (2000), Valls et al. (2004, 2005), and Kolisch and Hartmann (2006).

In a recent study, Valls et al. (2005) showed that the double justification (DJ) is a simple technique that can be easily incorporated into many diverse RCPSP algorithms, producing significant improvements in the quality of the schedules generated without generally requiring more computing time. One of these algorithms was the GA activity list of Hartmann (1998), a straightforward implementation of a GA with a general crossover operator. The new algorithm (termed DJGA), obtained after applying the double justification to each generated schedule of the GA, clearly outperformed the other state-of-the-art heuristics with an upper limit of 5000 generated schedules. These promising results have motivated us to develop a <u>h</u>ybrid <u>g</u>enetic <u>a</u>lgorithm (HGA), streamlining ideas and strategies already applied in the DJGA and other metaheuristics (Valls et al., 2003, 2004).

Double justification is called forward–backward improvement by some authors (e.g. Kolisch and Hartmann, 2006). This technique has been used with the name of forward/backward scheduling to develop multi-pass heuristic scheduling procedures (e.g. Li and Willis, 1992; Özdamar and Ulusoy,

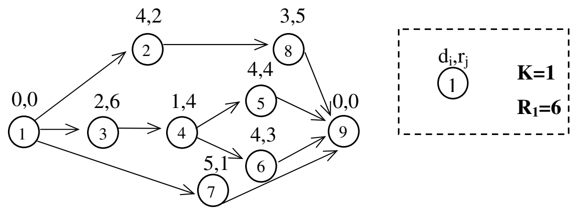

Fig. 1. Activity network for illustrative example.

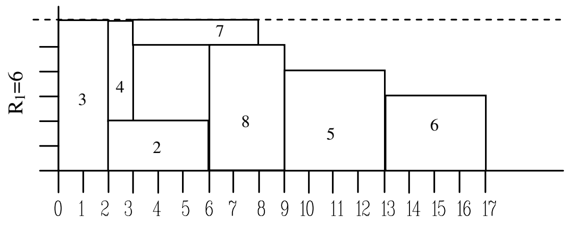

Fig. 2. Feasible schedule S for the illustrative example.

1996a,b; Tormos and Lova, 2001). The interest of forward–backward scheduling has also been sustained by the experiments reported in Palpant et al. (2004).

In this paper we use the following example in Fig. 1 to illustrate the definitions. A feasible schedule S of the above problem is depicted in Fig. 2.

The rest of the paper is organized as follows: Section 2 is devoted to the description of the main elements of HGA. Computational tests are included in Section 3. Finally, a summary and some concluding remarks appear in Section 4.

2. A Hybrid Genetic Algorithm

Genetic Algorithms (GAs) simulate the biological evolution. For an introduction into GAs, we refer to Goldberg (1989).

In this paper we introduce several changes in the GA paradigm based on the knowledge of the problem. The resulting algorithm is a Hybrid Genetic Algorithm (HGA). Below, we briefly introduce the new elements; present the HGA scheme and finally, explain in detail the components of the algorithm.

HGA uses the *peak crossover operator*, whose objective is to exploit the knowledge of the problem to identify and combine those good parts of the solutions that have really contributed to its quality. This is opposed to general crossover operators, which combine randomly selected parts of good solutions without any guarantee, or even indication, of their quality.

Several researchers have extended the original GA approach with the use of local search strategies to improve the solutions obtained using genetic operators – see for example, Mühlenbein et al. (1988) or Ulder et al. (1991). With the same objective, we use the *double justification operator*, which is a local improvement operator that, thanks to its simplicity, speed, and efficiency, can be applied to all RCPSP solutions generated in the evolutionary process.

Most genetic algorithms are based on applying the evolutionary scheme to one population, the initial set of solutions. We consider that this approach is restrictive for two reasons. Firstly, as the number of generations increases, it becomes generally increasingly difficult to obtain global improvements. Secondly, new information is available throughout the search, which can be used to create new populations to which the evolutionary process can be applied. The genetic algorithm can therefore be guided to promising or new areas of the search space. One of the possibilities is to generate a new population by using a biased random sampling of the neighbourhood of the best solution found so far (*a neighbour's population*). Experience seems to show (Valls et al., 2003, 2004) that good candidate schedules are usually found 'fairly close' to other good schedules. Therefore, applying the evolutionary scheme to a sample of the neighbourhood of the best solution found so far may lead to better solutions.

Once the principal elements of the algorithm have been introduced, HGA can be summarised as shown in Fig. 3 where *niter* is a prefixed parameter.

HGA has two phases: an initial phase (steps 1, 2 and 3) of general searching and a second phase (steps 4 and 5) of searching in the neighbourhood of the best solution. In each generation of the first phase, the size of the population is constant and equal to *POPsize* where *POPsize* is an even integer. In each generation of the second phase, the size of the population is constant and equal to $\lfloor POPsize/2 \rfloor$.

In order to facilitate the comparison with state-of-the-art RCPSP heuristics, HGA generates a maximum of *nsche* schedules. For simplicity's sake, the number of schedules generated in each phase is upwardly limited by *nsche*/2. Given *nsche* and *POPsize*, it is easy to calculate *niter* so that this limit is not exceeded. Another number of iterations in each phase– depending on the state of the search – may produce better results.

We describe the components of HGA in more detail below.

2.1. Individuals and fitness

In HGA, individuals are activity list representations of schedules. An *activity list* (Kolisch and Hartmann, 1999) is any precedence feasible permutation $\lambda = (j_1, j_2, \ldots, j_n)$ of the activities. If $i = j_h$ we can say that the activity i is in the position $p(i) = h$. Given a schedule S, any activity list $\lambda = (j_1, j_2, \ldots, j_n)$ that satisfies: $s_i < s_j$ implies $p(i) < p(j)$ – is called an *activity list* (*AL*) *representation* of S. The *serial schedule generation scheme* transforms an activity list λ in a schedule $S(\lambda)$ by taking the activities one by one in the order of the list and scheduling them at their earliest precedence and resource-feasible start time. This procedure generates the so-called *active schedules*, where each activity is scheduled as early as possible. There is always an optimal active schedule. A schedule S may be represented by several

```
1. Create an initial population.
2. Compute fitness for individuals.
3. For [i =1,niter]
    3.1. Select pairs of individuals (parents).
    3.2. Produce children by applying:
        3.2.1. The peak operator to each pair of parents.
        3.2.2. The mutation and the double justification operators to each child.
    3.3. Evaluate the fitness of children.
    3.4 Add the children to the population.
    3.5. Reduce the population by selection.
4. Create a neighbour's population from the best solution so far.
5. Apply step 3 to the new population.
6. Return the best solution found.
```

Fig. 3. The HGA scheme.

activity lists that only differ in the positions assigned to those activities with the same start-times. However, if S is active and $\lambda(S)$ is any activity list representation of S, then $S(\lambda(S)) = S$. If the activity label is used to tie-break then the representation is unique.

In the rest of the paper, we assume that the tie-break rule is to choose the activity with the smallest activity label and we denote the unique AL representation of S as $\lambda(S)$. We also assume that all considered schedules are active and use S, and $\lambda(S)$ interchangeably.

The fitness of an individual λ is given by the makespan of the associated schedule $S(\lambda)$.

The vector $\lambda = (1,3,2,4,7,8,5,6,9)$ is an activity list representation of the active schedule S in Fig. 2. The vector $\lambda' = (1,3,2,4,8,7,5,6,9)$ is not an activity list representation of the schedule S as it is incompatible with the start times, p. e. $s_7 < s_8$ and activity 8 is before activity 7 in λ'.

2.2. The peak crossover operator

Let us look at schedule S in Fig. 2 whose makespan is 17. For each of the 17 periods of the makespan we can add the units of the only resource used by S and show this information in the resource-utilisation graph in Fig. 4. In this graph, the periods of time can be seen on the horizontal axis and the units of resource used appear on the vertical axis. We can see in the figure that there are peaks and valleys that correspond to the higher and lower use of resources, respectively. There are time intervals (e.g. $[0,3]$) with a high use of resources (100% of the six units available to the resource), and others (e.g. $[13,17]$) with a much lower utilisation of resources (only 50%). Other intervals such as $[8,9]$ or $[9,12]$ can be considered periods of high or low use depending on the definition of high/low utilisation that we adopt. In any case, the high use of resources is a desirable characteristic in a schedule for transmitting to descendents in a crossover process.

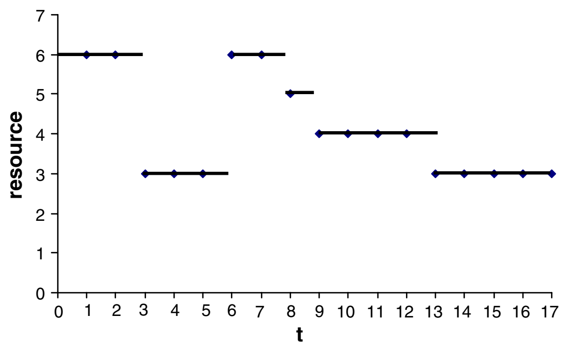

Fig. 4. Graphic showing the use of resources for schedule S.

Returning to schedule S in Fig. 2, we can see that the high use of resources in interval $[0,3]$ has occurred because activities 3, 2, and 4 have been scheduled in this order. $(3,2,4)$ is a sublist of the activity list representation of S that we call a peak. If we establish that 80% is the limit for an interval to be considered of high resource utilisation, then $[6,9]$ is the only other interval with a high utilisation of resources and so $(7,8)$ is the only other peak of S. Both peaks are naturally ordered for their position in the time scale. Given two schedules, father and mother, the peak crossover operator generates two new schedules, son and daughter, in such a way that the son (daughter) inherits the peaks of the father (mother) and the mother (father) determines the position of the remaining activities that are put before and after each peak. Now that we have looked at the ideas behind the peak crossover operator, we need to make some definitions before making a precise definition.

S is an active schedule and t is a period of time. Let Scheduled(t) be the set of activities that according to S are scheduled in the period t, that is to say, $\text{Scheduled}(t) = \{i \in \mathbf{V}; [t-1, t[\cap[s_i, f_i[\neq \emptyset\}$. A grouped measurement of the proportion of resources used by schedule S in period t is the Resource Utilisation Ratio:

$$\text{RUR}(t) = (1/K) * \sum_{j \in \text{Scheduled}(t)} \sum_{k=1}^{K} \frac{r_{j,k}}{R_k}.$$

Note that $0 \leqslant \text{RUR}(t) \leqslant 1$.

Given a threshold δ, we can say that a period t is of high resource utilisation for S, if $\text{RUR}(t) \geqslant \delta$. We say that time interval $I = [I_s, I_f[$, with $I_s < I_f$, is of high resource utilisation for S if all the periods included in I are of high resource utilisation for S. Let us say λ is an activity list representation and I is a time interval of high resource utilisation for $S(\lambda)$. Then we can say that the sublist of λ formed by the activities scheduled in a period of I is a peak of λ for the threshold δ. Notice that a peak P of λ is an ordered list of activities. Therefore, we can define $P(h)$ as the activity that occupies position h in P. Also notice that the positions in λ of the activities of a peak do not have to be consecutive. If i is the first activity of a peak P in λ then we can define the position of P in λ as the position of i in λ, that is to say $p(P) = p(i)$.

Given a threshold δ, the ordered list of disjointed peaks, $P_1, P_2, \ldots, P_q$, of a given activity list representation is built in the following way. P_1 is the first maximal peak of λ with respect to the inclusion. P_j is the first maximal peak after $t_j = \max\{s_i + d_i;\ i \in P_{j-1}\}$. The list may be empty.

Given $\delta = 0.8$, the list of peaks of the schedule S in Fig. 2 is $P_1 = (3, 2, 4)$, $P_2 = (7, 8)$.

Let us say λ (the father) and λ' (the mother) are two activity list representations in the current population that we wish to cross. The peak crossover operator generates two new activity lists: the son λ^s and the daughter λ^d. We will later see how the son is generated. The daughter is generated analogically by just interchanging the roles of the mother and father.

The value of the threshold δ is selected randomly in the interval [*lthreshold*, *uthreshold*], where *lthreshold* and *uthreshold* are parameters of the system and $0 \leqslant lthreshold < uthreshold \leqslant 1$. Let us say $P_1, P_2, \ldots, P_q$ is the list of disjointed peaks of λ with the threshold δ and $npeak_i$ is the cardinal of P_i. $Pred_i$ is the list of activities that precede any activities of the peak P_i, ordered according to λ' and $npred_i$ is the cardinal of $Pred_i$. $Pred_i(j)$ denotes the activity that occupies position j in the $Pred_i$ list.

The procedure shown in Fig. 5 builds the son λ^s and assigns n activities to n positions one by one and in growing position order. At each moment, an activity is candidate if it does not belong to any peak and has not been yet assigned – while all its predecessors have already been assigned.

If the ordered list of disjointed peaks of λ is empty then $\lambda^s = \lambda$.

2.3. The double justification operator

Given a schedule S, to *justify an activity* $j \neq n$ (1) *to the right* (*left*) consists of obtaining a schedule S' such that $s'_i = s_i$, $i \neq j$, $s'_j \geqslant s_j$ ($s'_j \leqslant s_j$) and s'_j is as large (small) as possible. The double justification of a schedule S, $DJ(S)$, is the schedule obtained after first justifying to the right the activities of S in decreasing order of their ends and then, justifying to the left the activities of the resulting schedule S' in decreasing order of their beginnings in S'. $DJ(S)$ is an active schedule which depends on the tie-breaking rule used. In this paper, we assume that the tie-break rule is to choose the activity with the smallest activity label. It is not difficult to see that $T(DJ(S)) \leqslant T(S)$. For further details, we refer to Valls et al. (2005).

2.4. Initial population

We employ the regret based biased random sampling method, Drexl (1991), using the LFT priority rule and fixing $\alpha = 1$ to obtain *POPsize* schedules. The initial population is obtained by first applying the double justification operator to each of the schedules and then obtaining their activity list representations. This method without double justification has been used in Hartmann (1998).

2.5. Parent selection

The parent selection is also a novelty in this paper. If *size* is the size of the current population (the value of *size* depends on the phase) and π, $0 < \pi \leqslant 1$, is a parameter of the system, then $\lfloor \pi * size/2 \rfloor$ couples of individuals (parents) are generated in the following manner. The couples are generated in pairs and each time a couple is formed it is temporally eliminated from the population. The first individual in the couple is the fittest individual in the population, and the second individual is chosen randomly from the rest of the individuals in the

```
1. h = 1.
2. For [i =1,q]                                              For each peak i
    2.1. For [j =1,n] and while [j< p(P_i)]    Put mother's activities before the peak
        2.1.1. If $\lambda'$ (j) is a candidate, then $\lambda^s$ (h) = $\lambda'$ (j) and h = h + 1.
    2.2. For [j = 1, $npred_i$]                         Complete the peak predecessors
        2.2.1. If $Pred_i$(j) is a candidate, then $\lambda^s$ (h) = $Pred_i$(j) and h = h + 1.
    2.3. For [j = 1, $npeak_i$]                                         Put the peak
        2.3.1 Do $\lambda^s$(h) = $P_i$(j) and h = h+1.
3. For [j = 1,n]                     Put the remaining activities behind the last peak
3.1 If $\lambda'$(j) is a candidate, then $\lambda^s$ (h) = $\lambda'$ (j) and h = h+1.
```

Fig. 5. Generation of a son.

current population. This way, we make sure that in each iteration the fittest individuals are used once as parents, but without fixing their couples. So, almost certainly different crosses are performed from iteration to iteration even if the population has not change.

2.6. Mutation operator

We apply the mutation operator used by Hartmann (1998). Given an individual λ the mutation operator works as follows: for all positions $i = 2$, $n - 2$, activities $\lambda(i)$ and $\lambda(i+1)$ are exchanged with probability *pmutation* if both activities are not precedence related. *pmutation* is an input parameter for the procedure.

2.7. Selection

The selection approach is a simple ranking method already used in Hartmann (1998). We keep the POPsize best individuals and remove the remaining individuals from the population (ties broken arbitrarily).

2.8. The 2-phase strategy

Given an activity list λ, the procedure we have implemented to construct a population of nearby activity lists is a simplified version of that used in Valls et al. (2003). The procedure generates $\lfloor POPsize/2 \rfloor$ schedules with $\beta = 1 - 20/n$ by applying the *β-biased random sampling method*, *β-BRSM*, to λ. The neighbour's population is obtained by first applying the double justification operator to each of the generated schedules and then obtaining their activity list representations.

β-BRSM can be described as follows. Given an activity list λ, the method makes use of the serial SGS. The following strategy is employed to select, in each iteration, the next activity j to be scheduled. A number $p \in (0,1)$ is randomly generated. If $p < \beta$, then j is the eligible activity with the lowest position. Otherwise, j is one of the other eligible activities selected by biased random sampling (Kolisch and Hartmann, 1999), and employing the positions as priority values in order to obtain the selection probabilities. Note that the parameter β controls the extent to which the new vector differs from the original.

For the interested reader, it is worth noting that the 2-phase strategy implemented in HGA differs from that implemented in the algorithm FPBP (Valls et al., 2004) in three points although both strategies are motivated by the same idea of exploring the neighbourhoods of good solutions. The first point concerns the nature of the algorithms. Algorithm FPBP differs from HGA in that it is a population-based algorithm where the individuals of the population evolve independently from one another by applying an improvement procedure to each solution. The second point refers to the implementation issue. In the phase two of FPBP, each time an improved solution is generated a new neighbour's population is generated and the whole procedure restarts from it. In HGA, only one neighbour's population is generated. The third point refers to the actual procedure to generate a neighbour's population. The procedure we have implemented in HGA is a simplified version of that used in Valls et al. (2003). We now generate only POP_size/2 individuals with a unique value of β and apply DJ to them instead of generating, with two different values of β, 600 new activity lists from which the 10 best are selected to form the neighbour's population. We think we have proposed a different implementation of a common and very vague starting idea and that we have shown that this idea also works in a GA environment.

3. Computational experiments

In this section we present the results of the computational studies concerning the algorithm introduced in previous sections. The experiments have been performed on a personal computer AMD at 400 MHz. The algorithm has been coded in C.

3.1. Test instances

For test instances, we have used the standard sets j30, j60, j90 and j120 that have been generated using ProGen, Kolisch et al. (1995). The projects in these test sets consist of 30, 60, 90, and 120 activities, respectively. These instances were generated under a full factorial experimental design with the following three independent problem parameters: network complexity (NC), resource factor (RF), and resource strength (RS). The sets j30, j60, and j90 consist of 480 instances each (10 replications of 48 treatments) and the set j120 consists of 600 projects (10 replications of 60 treatments). Details of these problem instances are given in Kolisch et al. (1995) and Kolisch and Sprecher (1997). The instances have been used by many researches and they are available in the Project Scheduling Problem Library in the internet (PSPLIB: http://129.187.106.231/psplib) along with the optimum, or best known, values that have been obtained by various authors over the years.

Several authors (e.g. Hartmann and Kolisch, 2000) have pointed out that the parameter RS is a very influential parameter as far as solution quality of exact and heuristic algorithms is concern. For the instance sets j30, j60, and j90, the parameter RS can take the values 0.2, 0.5, 0.7, or 1, whereas for the instance set j120 the values are 0.1, 0.2, 0.3, 0.4, or 0.5. So, the average value of RS is much lower for the set j120 than for the others and the instances with lower values of RS are the most difficult to solve, as several researches have indicated, e.g. Dorndorf et al. (2000). In fact, any instance with RS = 1 is trivial as the earliest start schedule is optimal for it. It is also worth noting that the state-of-the-art exact methods for the RCPSP solve the j30 instances in a very short computation time. For example, the branch-and-bound algorithm DH2 (Herroelen et al., 1998) solves the j30 instances to optimality in an average runtime of 0.37 seconds on a Pentium Pro/200 computer. This result raises the question as to whether it is still appropriated to use j30 to test the performance of heuristic algorithms. Therefore, the computational results regarding the relative quality of heuristic algorithms that are based on computational experiments performed on the sets j30, j60, and j90 are not comparable to those based on the set j120: The first class of results refers to instances that in general are easier than those considered in the second class of experiments. For this reason, we can expect that high quality algorithms would obtain similar results on the sets j30, j60, and j90 but that they might differentiate on the set j120. Furthermore, the percentage of instances of j30, j60, j90 and j120 for which the optimum solution is known is approximately 100%, 74%, 73% and 35%, respectively and according to PSPLIB. These figures further indicate that the set j120 is the most difficult to resolve for the current algorithms.

3.2. Parameter setting

We have set the values *lthreshold* = 0.75, *uthreshold* = 0.9, and *pmutation* = 0.05 for all the instances. The value of the parameter *pmutation* is that used in Hartmann (1998). The first computational tests have been made for values of *nsche* = 2500, 5000, 10,000, 25,000 and 100,000. Notice that the total number *nsche* accounts for *nsche*/3 schedules generated by the GA and 2 ∗ *nsche*/3 generated by the DJ. For each value of *nsche* we have performed some preliminary experiments with a small subset of instances from set j120 to fix the values of *POPsize* and π for the j120 instances. Concretely, we have selected the first instance in each treatment. The values of *POPsize* selected were 24, 50, 100, 200, and 400 for *nsche* = 2500, 5000, 10,000, 25,000 and 100,000, respectively. The value of π is 0.6 for the smallest value of *nsche*, 0.4 for the next three values and 0.9 for the largest value. For 5000 schedules and the instance sets j30, j60, and j90, the values of *POPsize* (π) were fixed to 100 (0.8), 50 (0.7), and 50 (0.9) respectively. For each instance set, the training test bed consists of the first instance in each treatment.

Furthermore, Table 1 shows the values of *POPsize* (π) selected for the HGA computational results shown in Tables 9–11.

Table 1
Values of parameters *POPsize* and π

Nsche	1000	5000	50,000
j30	50 (1)	100 (0.8)	1000 (1)
j60	50 (0.2)	50 (0.7)	300 (1)
j120	24 (0.4)	50 (0.4)	400 (0.4)

3.3. Main results

We first tested HGA on the instance sets j30, j60, j90, and j120 by limiting the number of generated schedules to a maximum of 5000. Table 2 summarises the results. The first column indicates the instance set referred to in the results shown in each row. The average (standard deviation) percentage deviation from the critical path makespan lower bound – which is obtained by computing the length of a critical path in the resource relaxation of the problem, is reported in the second (third) column, labelled CP_dev (sd). Finally, the average (maximal)

Table 2
HGA5000 in j30, j60, j90 and j120

	CP_dev	sd	av_cpu	max_cpu
j120	32.54	0.44	2.03	3.46
j90	10.46	0.21	0.61	2.31
j60	11.10	0.22	0.46	1.42
j30	13.47	0.22	0.31	0.66

Table 3
HGA in j120 with different limits on the number of schedules

	CP_dev	sd	av_cpu	max_cpu
HGA2500	33.18	0.45	1.02	1.76
HGA5000	32.54	0.44	2.03	3.46
HGA10000	32.04	0.43	4.01	6.97
HGA25000	31.52	0.43	10.02	17.69
HGA100000	30.95	0.42	39.51	69.26

computation time in seconds is shown in column fourth (fifth), labelled av_cpu (max_cpu). We have also run HGA on the instance set j120 while changing the number of permitted schedules to 2500, 10,000, 25,000 and 1,000,000. Table 3 shows the obtained results.

We can observe that HGA5000 produces high quality solutions with low computational times. As we have mentioned above, optimal solutions for all instances in j30 are known. The average deviation of HGA5000 with respect to the optimal solutions is 0.06. We can observe that the quality of HGA increases with the limiting number of schedules; although the computational time also increases linearly. It is interesting to see that max_cpu is less than twice av_cpu in all versions of HGA in j120. We would also like to add that the algorithm breaks the barrier of 31% of average percentage deviation from the critical path makespan in an average of just 40 seconds.

We can observe that HGA produces very stable results: The standard deviation is very small in all cases (Tables 2 and 3). Furthermore, we can also observe that the standard deviation steadily decreases as the limiting number of schedules increases (Table 3).

3.4. The influence of problem parameters

To further analyse the performance of HGA we focus on the set j120 with a limit of 5000 schedules. Tables 4 and 5 show the influence of the generation parameters RS, RF, and NC on CP_dev. HGA($\pi = 1, 1$phase) is a special version of HGA with a limit of 5000 schedules, with $\pi = 1$ and only the first phase. This is equivalent to running DJGA with the peak crossover operator.

From the last column in Table 4, the CP_dev improvement obtained when the two point crossover operator is changed by the peak crossover operator in DJGA can be calculated ($33.24 - 32.96 = 0.28$). A further improvement is obtained when the combined effect of the parent selection and two-phase strategies is added ($32.96 - 32.54 = 0.42$).

We can also observe in Tables 4 and 5 that characteristics RS and RF have a greater impact on CP_dev than NC as it has already been noticed by several authors, e.g. Hartmann and Kolisch (2000). Although one has to be careful when interpreting these results because the quality of the lower bound is unknown it seems that the more difficult instances are those with low RS and large RF in concordance with previous computational results, e.g. Dorndorf et al. (2000).

The computational times for HGA shown in Table 6 provide further evidence. We can also see

Table 4
CP_dev values by RS values in j120

	RS					All instances
	0.1	0.2	0.3	0.4	0.5	
DJGA	98.21	44.13	16.56	6.30	0.99	33.24
HGA($\pi = 1$,1phase)	96.83	43.98	16.62	6.39	0.98	32.96
HGA	95.69	43.47	16.41	6.21	0.93	32.54

Table 5
CP_dev values by RF and NC values in j120

	RF				NC		
	0.25	0.5	0.75	1	1.5	1.8	2.1
DJGA	8.62	26.63	42.76	54.93	32.35	31.24	36.12
HGA($\pi = 1$,1phase)	8.52	26.43	42.52	54.37	32.07	30.97	35.84
HGA	8.59	26.17	41.85	53.54	31.61	30.53	35.49

Table 6
HGA5000 cpu times by RS, RF, and NC values in j120

	RS					RF				NC		
	0.1	0.2	0.3	0.4	0.5	0.25	0.5	0.75	1	1.5	1.8	2.1
av_cpu	2.96	2.78	2.3	1.58	0.54	1.42	1.87	2.30	2.55	1.89	1.98	2.22
max_cpu	3.46	3.25	3.08	3.02	2.80	2.64	3.08	3.30	3.46	3.46	3.35	3.30

that HGA obtains the best result for all parameter values but one (RF = 0.25) and that HGA($\pi = 1$,1-phase) improves DJGA for all parameter values but two (RS = 0.3, 0.4).

Table 7 provides further insight on the influence of the peak crossover operator and the combined effect of the parent selection and two-phase strategies. It can be observed that the number of instances for which HGA($\pi = 1$,1phase) obtains better solutions than DJGA (N_better_sched) is greater than the number of instances for which it obtains worse solutions (N_worse_sched). This global result is true for RS = 0.1, 0.2, and 0.5 but not for RS = 0.3 and 0.4. It seems that the peak crossover operator is more efficient when dealing with the difficult instances than with the easy ones. However, when the combined effect of the parent selection and two-phase strategy is added (HGA versus DJGA) then the difference between N_better_sched and N_worse_sched increases for each value of RS and for the total. Furthermore, all these differences are positive.

Table 8 shows the influence of parameter RF on CP_dev for instances of low RS. It seems that instances with (RS = 0.1 and RF = 0.5, 0.75, 1; RS = 0.2 and RF = 0.75, 1) are more difficult than the other instances.

3.5. Further analysis

In Table 9, two new special versions of HGA are considered. HGA($\pi = 1$) is algorithm HGA but

Table 7
Number of better and worse schedules by RS values in j120

	HGA($\pi = 1$,1phase) versus DJGA		HGA versus DJGA	
	N_better_sched	N_worse_sched	N_better_sched	N_worse_sched
RS = 0.1	72	25	88	13
RS = 0.2	49	32	56	19
RS = 0.3	26	28	31	19
RS = 0.4	10	19	20	10
RS = 0.5	4	3	10	3
Total	161	107	205	64

Table 8
CP_dev values by RF values for low RS instances in j120

	RS = 0.1				RS = 0.2			
	RF				RF			
	0.25	0.5	0.75	1	0.25	0.5	0.75	1
DJGA	26.07	81.56	127.75	157.47	12.79	36.74	56.10	70.88
HGA($\pi = 1$,1phase)	25.57	80.14	126.41	155.21	12.78	36.55	56.30	70.29
HGA	25.94	79.52	124.58	152.74	12.86	36.14	55.37	69.50

Table 9
Single and combined influence of both strategies on CP_dev

	DJGA	HGA($\pi = 1$,1phase)	HGA($\pi = 1$)	HGA(1phase)	HGA
CP_dev	33.24	32.96	32.65	32.64	32.54

with the same parent selection strategy as in DJGA and Hartmann (1998). HGA(1phase) is a special version of HGA with a limit of 5000 schedules and only the first phase. The difference $32.96 - 32.65 = 0.31$ shows the effect on CP_dev of applying the two-phase strategy after having incorporated the peak crossover operator into DJGA whereas the difference $32.96 - 32.64 = 0.32$ shows the effect on CP_dev of applying the new parent selection strategy after having incorporated the peak crossover operator into DJGA. The two effects are almost the same. However, the effect of applying both strategies together is $32.96 - 32.54 = 0.42$ which is greater than both single effects but smaller than their sum.

We have also performed some tests to see the influence on CP_dev of changes on a single parameter value. To test the influence of parameter π we have run HGA with $\pi = 0.3$, 0.5, and 1 obtaining values of CP_dev = 32.58, 32.56, and 32.65, respectively. These values compare unfavourably with the value CP_dev = 32.54 obtained by the standard HGA ($\pi = 0.4$). We have also run HGA with fixed values of $\delta = 0.6$, 0.7, 0.8, 0.9, and 0.95 obtaining values of CP_dev = 34.84, 34.00, 33.93, 32.70, and 33.84, respectively. However, a better value of CP_dev can be obtained if δ is allowed to randomly vary within an interval. Among the few intervals tested, the interval [0.75, 0.9] is the one that yields the best CP_dev (32.54).

We have already compared the proposed parent selection strategy to that used by Hartmann (1998). Now, we compare it with the tournament selection strategy Goldberg et al. (1989), a standard selection strategy of which the selection pressure and loss of diversity depend on the parameter 'tournament size' (K). We have run HGA with the tournament selection strategy with $K = 2$, 3, 4, and 5 obtaining a CP_dev = 32.63, 32.88, 32.92, and 33.01, respectively. Again, these values compare unfavourably with the value CP_dev = 32.54 obtained by the standard HGA.

3.6. Comparison to state-of-the-art algorithms

In this subsection we compare HGA to state-of-the-art heuristic algorithms. As far as the paper is concerned we have looked at all the algorithms referred to in Kolisch and Hartmann (2006), and have chosen to include only those which rank first. In this paper, the computational results are shown in Tables 1–3 for the instance sets j30, j60, and j120, respectively. Each of the three tables is divided into three blocks. The first block includes algorithms for which it is reasonable to accept the assumption that the computational effort for constructing any schedule is similar within the algorithm and between algorithms in the block. This assumption is not acceptable, for example, if backtracking steps or mixed integer program-based methods are included. This assumption justifies the decision to take the number of generated schedules as the basis for the comparison of all the algorithms considered in this block. Three limits on the number of generated schedules are selected: 1000, 5000 and 50,000 schedules. Some algorithms do not fit into the first block but they are included in the blocks two and three of the tables. The second block contains algorithms where the average deviation together with the average and maximal number of schedules (Max_N_sched) required is given. The third block reports the results of heuristics together with the average and maximal computation time as well as the clockpulse of the computer used. In each block, the heuristics are sorted with respect to increasing deviation. In the first block, the methods are sorted with respect to the results for 50,000 schedules and then for 5000 schedules. For each instance set and from the corresponding table we have selected the following algorithms to be included in our comparison Tables 9–11: The first 10 algorithms in block one and the algorithms in blocks two and three which average deviation is less or equal than the greatest deviation for the algorithms selected from block one for 1000 schedules. We have also borrowed the corresponding computational results. Tables 9–11 show the computational results of the selected heuristic algorithms for the instance sets j30, j60, and j120, respectively. They are structured into two blocks. The first (second) block in each table contains the algorithms selected from the first (second and third) block(s) in the corresponding table in Kolisch and Hartmann. The computation times not provided in Kolisch and Hartmann have been obtained from the original papers. The average deviations in Table 10 are calculated with respect to the optimal solutions.

The results indicate that HGA is capable of providing near optimal solutions with very small computation times. It ranks second among the algorithms in block one. In block 2, the algorithm of Palpant et al. (2004), obtains better results in terms of solution quality, but at the expense of

Table 10
Average deviations from optimal makespan in j30

Author(s)	Algorithm type	Max_N_sched			
		1000	5000	50,000	
Kochetov and Stolyar (2003)	GA, TS, path reli.	0.10	0.04	0.00	
Valls et al.	Hybrid GA	0.27	0.06	0.02	
Alcaraz and Maroto (2001)	GA	0.33	0.12	–	
Valls et al. (2005)	DJGA	0.34	0.20	0.02	
Tormos and Lova (2003a)	Sampling + BF/FB	0.25	0.13	0.05	
Nonobe and Ibaraki (2002)	Tabu Search	0.46	0.16	0.05	
Tormos and Lova (2001)	Sampling + BF	0.30	0.16	0.07	
Hartmann (2002)	Self-adapting GA	0.38	0.22	0.08	
Hartmann (1998)	Activity list GA	0.54	0.25	0.08	
Tormos and Lova (2003b)	Sampling + BF	0.3	0.17	0.09	
		Av. Dev.	CPU-time (seconds)		Computer
			Average	Max.	
Palpant et al. (2004)	Decomp. + local opt.	0.00	10.26	123.0	2.3 GHz
Fleszar and Hindi (2004)	VNS-activity list	0.01	0.64	5.9	1.0 GHz
Palpant et al. (2003)	LNS	0.02	4.05	67	2.1 GHz
Valls et al. (2003)	Extended TS	0.06	1.61	6.2	400 MHz
Valls et al. (2004)	Population-based	0.10	1.16	5.5	400 MHz
Sprecher (2002)	Network decomp.	0.12	2.75	39.7	166 MHz
Hindi et al. (2002)	Activity list GA	0.37	0.17	0.49	400 MHz

Table 11
CP_dev values in j60

Author(s)	Algorithm type	Max_N_sched			
		1000	5000	50,000	
Valls et al.	Hybrid GA	11.56	11.10	10.73	
Kochetov and Stolyar (2003)	GA, TS, path reli.	11.71	11.17	10.74	
Valls et al. (2005)	DJGA	12.21	11.27	10.74	
Hartmann (2002)	Self-adapting GA	12.21	11.70	11.21	
Hartmann (1998)	Activity list GA	12.68	11.89	11.23	
Tormos and Lova (2003a)	Sampling + BF/FB	11.88	11.62	11.36	
Tormos and Lova (2003b)	Sampling + BF	12.14	11.82	11.47	
Alcaraz and Maroto (2001)	GA	12.57	11.86	–	
Tormos and Lova (2001)	Sampling + BF	12.18	11.87	11.54	
Bouleimen and Lecocq (2003)	SA	12.75	11.90	–	
		CP_dev	CPU-time (seconds)		Computer
			Average	Max.	
Palpant et al. (2003)	LNS	10.79	13.39	82	2.1 GHz
Palpant et al. (2004)	Decomp. + local opt.	10.81	38.8	223.0	2.3 GHz
Valls et al. (2004)	Population-based	10.89	3.7	22.6	400 MHz
Fleszar and Hindi (2004)	VNS-activity list	10.94	8.89	80.7	1.0 GHz
Valls et al. (2003)	Extended TS	11.45	2.8	14.6	400 MHz
Sprecher (2002)	Network decomp.	11.61	460.2	4311.5	166 MHz
Artigues et al. (2003)	Tabu Search	12.05	3.2	–	450 MHz

greater computational time. The algorithm of Fleszar and Hindi outperforms HGA both in solution quality and computation time. The algorithm of Palpant et al. (2003), obtains the same solution quality as HGA with 50,000 schedules, but at the expense of greater computational time. The remainder of the algorithms is outperformed by HGA.

Table 11 shows that our algorithm outperforms all algorithms in block one for all schedule limits. HGA with 50,000 schedules obtains better solution quality than the last three algorithms in block two in a shorter average computation time. HGA with 50,000 schedules obtains better solution quality than the first four algorithms, in a shorter average computation time in all cases except for possibly the population-based algorithm in Valls et al. (2004).

Table 12 shows that HGA outperforms all algorithms in block 1 for all schedule limits. HGA with 1000, 5000, and 25,000 schedules outperforms the five last, fourth and three first algorithms in block two, respectively, both in terms of solution quality and average computation time.

We have also compared HGA to the algorithm of Debels et al. (2006), not included in the computational study by Kolisch and Hartmann (2006). Table 13 shows average deviations with respect to the optimal solutions (j30) and with respect to critical path lengths (j60 and j120) for 1000, 5000, and 50,000 schedules. We can observe that in j30 and j60 there is one case where an algorithm obtains better results than the other, another case where the opposite occurs and yet another case where both algorithms obtain the same result. However, the differences in favour of HGA are of greater magnitude than those against it. In j120, HGA consistently obtains better results. Additionally, HGA requires a greater average computation time for j30 and a smaller average computation time for j60 and j120 than the algorithm of Debels et al., if we take into account the difference in processor speed. For example, for 5000 schedules the average computation times required by the algorithm of Debels et al. are 0.06, 0.18, and 0.65 seconds (1.8 GHz) for j30, j60, and j120, respectively, whereas HGA requires 0.31, 0.46, and 2.03 seconds (400 MHz), respectively. Another example is the following: The algorithm of Debels et al. obtains a CP_dev value of 31.57 in 6.66 seconds (1.8 GHz) for 50,000 schedules in j120. HGA is able to do better using 25,000 schedules: CP_dev = 31.51 in 10.02 seconds (400 MHz). This case illustrates the main drawback of taking the number of schedules as the only basis for comparison: different heuristics may require different computation times to compute one schedule.

As a final conclusion we can say that HGA outperforms all state-of-the-art algorithms for the RCPSP known by the authors of this paper – for the instance sets j60 and j120. And that it is competitive with other state-of-the-art heuristics for the instance set j30.

Table 12
CP_dev values in j120

Author(s)	Algorithm type	Max_N_sched			
		1000	5000	50,000	
Valls et al.	Hybrid GA	34.07	32.54	31.24	
Valls et al. (2005)	DJGA	35.39	33.24	31.58	
Kochetov and Stolyar (2003)	GA, TS, path reli.	34.74	33.36	32.06	
Valls et al. (2005)	DJ, population based, changes operator	35.18	34.02	32.81	
Hartmann (2002)	Self-adapting GA	37.19	35.39	33.21	
Tormos and Lova (2003a)	Sampling + BF/FB	35.01	34.41	33.71	
Merkle et al. (2002)	Ant Co. Opt.	–	35.43	–	
Hartmann (1998)	Activity list GA	39.37	36.74	34.03	
Tormos and Lova (2003b)	Sampling + BF	36.24	35.56	34.77	
Tormos and Lova (2001)	Sampling + BF	36.49	35.81	35.01	
		CP_dev	CPU-time (seconds)		Computer
			Average	Max.	
Valls et al. (2004)	Population-based	31.58	59.4	264.0	400 MHz
Palpant et al. (2003)	LNS	32.15	71.89	158	2.1 GHz
Palpant et al. (2004)	Decomp. + local opt.	32.41	207.9	501.0	2.3 GHz
Fleszar and Hindi (2004)	VNS-activity list	33.10	219.86	1126.97	1.0 GHz
Valls et al. (2003)	Extended TS	34.53	17.0	43.9	400 MHz
Möhring et al. (2003)	Lagrangian Heur.	36.00	72.9	654.0	200 MHz
Artigues et al. (2003)	Tabu Search	36.16	67.0	–	450 MHz
Sprecher (2002)	Network decomp.	39.29	458.5	1511.3	166 MHz
Artigues et al. (2003)	Insheur/WCS	39.34	18	–	Sun ultra-4 ws

Table 13
Debels et al. versus HGA results for j30, j60, and j120

	Debels et al.			HGA		
	1000	5000	50,000	1000	5000	50,000
j30	0.27	0.11	0.01	0.27	0.06	0.02
j60	11.73	11.10	10.71	11.56	11.10	10.73
j120	35.22	33.10	31.57	34.07	32.54	31.24

4. Summary and concluding remarks

In this paper we have presented a Hybrid Genetic Algorithm, HGA, for the RCPSP. It differs from a 'standard' implementation of the GA ideas in three fundamental aspects. Firstly, the (peak) crossover operator used in HGA is not a pure random nor a context-free operator. It has been designed to combine useful problem-specific information extracted from the parents with the purpose of generating high quality children. Secondly, HGA systematically uses double justification as a simple, fast, and powerful mechanism to improve schedules. And thirdly, HGA incorporates a two-phase strategy by which the second phase re-starts the evolution from a neighbour's population of the best schedule found in the first phase. These three mechanisms, with the help of a new selection method, accelerate the evolution of the population towards higher quality regions. The computational results show that HGA is a fast and high quality algorithm. HGA outperforms all state-of-the-art algorithms for the RCPSP known by the authors of this paper for the instance sets j60 and j120. And it is competitive with other state-of-the-art heuristics for the instance set j30. For practical purposes it is interesting to note that the solution quality steadily increases with increasing number of generated schedules and that the computation time linearly increases with the number of schedules so it is easily predictable. It is our opinion that the success of HGA is mainly due to the incorporation of specific knowledge of the problem into an otherwise powerful context-free metaheuristic.

References

Alcaraz, J., Maroto, C., 2001. A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research 102, 83–109.

Artigues, C., Michelon, P., Reusser, S., 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research 149, 249–267.

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

Bouleimen, K., Lecocq, H., 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 (2), 268–281.

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

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M., 2006. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research 169 (2), 638–653.

Demeulemeester, E., Herroelen, W., 2002. Project scheduling – A research handbookInternational Series in Operations Research and Management Science, vol. 49. Kluwer Academic Publishers.

Dorndorf, U., Pesch, E., Phan-Huy, T., 2000. A branch-and-bound algorithm for the resource-constrained project scheduling problem. Mathematical Methods of Operations Research 52, 413–439.

Drexl, A., 1991. Scheduling of project networks by job assignment. Management Science 37, 1590–1602.

Fleszar, K., Hindi, K.S., 2004. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. European Journal of Operational Research 155 (2), 402–413.

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

Goldberg, D.E., Korb, B., Deb, K., 1989. Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems 3, 493–530.

Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45, 733–750.

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

Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127 (2), 394–407.

Herroelen, W., De Reyck, B., Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research 25 (4), 279–302.

Hindi, K.S., Yang, H., Fleszar, K., 2002. An evolutionary algorithm for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation 6, 512–518.

Kochetov, Y., Stolyar, A., 2003. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceeding of Workshop on Computer Science and Information Technologies.

Kolisch, R., Hartmann, S., 1999. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling, Recent Models, Algorithms and Applications. Kluwer Academic Publishers, Boston, pp. 147–178.

Kolisch, R., Hartmann, S., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 174, 23–37.

Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega 29, 249–272.

Kolisch, R., Sprecher, A., 1997. PSPLIB – A project scheduling library. European Journal of Operational Research 96, 205–216. Available from: <http://129.187.106.231/psplib>.

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

Li, K.Y., Willis, R.J., 1992. An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56, 370–379.

Merkle, D., Middendorf, M., Schmeck, H., 2002. Ant colony optimization for resource-constrained project scheduling. IEEE Transaction on Evolutionary Computation 6 (4), 333–346.

Möhring, R., Schulz, A., Stork, F., Uetz, M., 2003. Solving project scheduling problems by minimum cut computations. Management Science 49 (3), 330–350.

Mühlenbein, H., Gorges-Schleuter, M., Krämer, O., 1988. Evolution algorithms in combinatorial optimization. Parallel Computing 7, 65–88.

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP). In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, pp. 557–588.

Özdamar, L., Ulusoy, G., 1996a. A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis. European Journal of Operational Research 89, 400–407.

Özdamar, L., Ulusoy, G., 1996b. An iterative local constraint based analysis for solving the resource-constrained project scheduling problem. Journal of Operations Management 14, 193–208.

Palpant, M., Artigues, C., Michelon, P., 2003. A LNS method for solving the resource-constrained project scheduling problem. In: Proceedings of the 5th Metaheuristic International Conference MIC.

Palpant, M., Artigues, C., Michelon, P., 2004. LSSPER solving the Resource-Constrained Project Scheduling Problem with large neighbourhood search. Annals of Operations Research 31 (1), 237–257.

Sprecher, A., 2002. Network decomposition techniques for resource-constrained project scheduling. Journal of the Operational Research Society 53 (4), 405–414.

Tormos, P., Lova, A., 2001. A competitive heuristic solution technique for resource-constrained project scheduling. Annals of Operations Research 102, 65–81.

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

Tormos, P., Lova, A., 2003b. An efficient multi-pass heuristic for project scheduling with constrained resources. International Journal of Production Research 41, 1071–1086.

Ulder, N.L.J., Pesch, E., van Laarhoven, P.J.M., Bandelt, H.J., Aarts, E.H.L., 1991. Genetic local search algorithm for the traveling salesman problem. In: Maenner, R., Schwefel, H.P. (Eds.), Parallel Problem Solving from Nature. Springer-Verlag, pp. 109–116.

Valls, V., Quintanilla, S., Ballestín, F., 2003. Resource-constrained project scheduling: A critical activity reordering heuristic. European Journal of Operational Research 149, 282–301.

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

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

Weglarz, J., 1999. Project Scheduling. Recent Models, Algorithms and Applications. Kluwer Academic Publishers.