Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update

Rainer Kolisch[a], Sönke Hartmann[b]

[a] Chair of Technology–based Services & Operations Management
Technical University of Munich, Germany
e-mail: rainer.kolisch@wi.tum.de
www.som.wi.tum.de

[b] OR Consulting
Bornstraße 6, D-20146 Hamburg, Germany
e-mail: soenke.hartmann@epost.de
www.bwl.uni-kiel.de/bwlinstitute/Prod/mab/hartmann/hartmann.html

April 19, 2004

Abstract

This paper considers heuristics for the well–known resource–constrained project scheduling problem (RCPSP). It provides an update of our survey which was published in 2000. We summarize and categorize a large number of heuristics that have recently been proposed in the literature. Most of these heuristics are then evaluated in a computational study and compared on the basis of our standardized experimental design. Subsequently, we discuss our test design in more detail and give some remarks on its usage by other researchers in future studies. The paper closes with a summary of the recent developments in research on heuristics for the RCPSP.

Keywords: Project Scheduling, Resource Constraints, Heuristics, Computational Evaluation.

1 Introduction

The resource–constrained project scheduling problem (RCPSP) can be stated as follows: A single project consists of a number of n activities where each activity has to be processed in order to complete the project. The activities are interrelated by two kinds of constraints. First, precedence constraints force activity j not to be started before all its immediate predecessors have been finished. Second, performing the activities requires resources with limited capacities. Altogether there is a set of $\mathcal{R}$ resources. While being processed, activity j requires $r_{j,k}$ units of resource $k \in \mathcal{R}$ in every time instant of its non–preemptable duration p_j. Resource k has a limited capacity of R_k at any point in time. The parameters p_j, $r_{j,k}$, and R_k are assumed to be nonnegative and deterministic. The objective of the RCPSP is to find precedence and resource feasible completion times for all activities such that the makespan of the project is minimized.

Since its advent (cf. Pritsker et al. [59]) the RCPSP has been a very popular and frequently studied NP–hard optimization problem (cf. Błażewicz et al. [6]). The last 20 years have witnessed a tremendous improvement of both heuristic and exact solution procedures (cf. e.g. the surveys given in Demeulemeester and Herroelen [16], Hartmann and Kolisch [24, 35], Herroelen et al. [25], Kolisch and Padman [36], and Özdamar and Ulusoy [51]). Due to the fact that the RCPSP "is one of the most intractable problems in Operations Research", it has recently "become a popular playground for the the latest optimization techniques, including virtually all local search paradigms" (cf. Möhring et al. [46] p. 330).

The paper at hand is in line with research that has been performed by the authors in [35] and [24]. In [35] we have classified the multitude of heuristic procedures for the RCPSP with respect to their building blocks such as e.g. schedule generation scheme (SGS), metaheuristic strategy, and solution representation. We have also tested the heuristics on three sets of benchmark instances from the PSPLIB library (cf. Kolisch et al. [38], Kolisch and Sprecher [37]). This research has been continued in [24] where we included new methods and performed a rigorous computational study in order to compare the heuristics, to assess the significance of the building blocks, and to evaluate the impact of problem characteristics such as e.g. the scarcity of resources. The new paper has four goals: First, we summarize new heuristics for the RCPSP which have been presented since our last survey. Second, we extend the computational comparison of [24] by including new heuristics and by adding results which have been obtained by applying a larger computational effort. Third, we try to point out promising approaches which promote the progress in the field. Finally, we provide a critical discussion of the test design and its usage by other authors. To restrict the scope of this survey, we consider only heuristics developed for the classical RCPSP. Nevertheless, we have included papers for generalizations of the RCPSP if computational results for standard test instances of the classical RCPSP are given.

2 New Heuristics for the RCPSP

This section summarizes recent heuristics from the literature. Following the categorization of our study [35] the approaches are grouped into priority rule–based X–pass methods (Section 2.1), classical metaheuristics (Section 2.2), non–standard metaheuristics (Section 2.3), and other heuristics (Section 2.4). For a detailed description of basic components of heuristics such as schedule generation schemes, priority rules, and representations, we refer to [35]. Heuristics that have already been summarized in our previous surveys will be briefly mentioned at the beginning of each Section. For a description the reader is referred to [24, 35].

2.1 X-Pass Methods

X-pass methods which have been summarized in our recent study were presented by Alvarez-Valdes and Tamarit [2], Boctor [7], Cooper [12, 13], Davies [14], Davis and Patterson [15], El-sayed [18], Klein [27], Kolisch [31, 32, 33], Kolisch and Drexl [34], Lawrence [39], Li and Willis [42], Özdamar and Ulusoy [50, 52, 53], Patterson [55, 56], Schirmer [61], Schirmer and Riesenberg [62, 63], Thesen [67], Thomas and Salhi [68], Ulusoy and Özdamar [74], Valls et al. [78], and Whitehouse and Brown [80].

Bidirectional scheduling. Klein [28] proposes a bidirectional approach which employs the serial or the parallel schedule generation scheme and simultaneously constructs a forward and a backward schedule. Each activity is twice in the eligible set: One copy with a forward and one copy with a backward priority value. According to the selected copy the activity is added to the forward or backward schedule. The partial schedules are concatenated by adding the activities of the backward schedule in the order of non–decreasing start times to the forward schedule. The method generates on average 1.5 schedules.

Sampling. Coelho and Tavares [11] suggest a so–called global biased random sampling approach which employs the serial SGS. Whereas previous sampling methods compute probabilities for activity selection, this procedure perturbs the priority values by adding a random value $\in [0,1]$ which is multiplied by a scaling factor. The activites are scheduled in the order prescribed by the modified priority values. An efficient implementation which exploits these "global" perturbations and avoids eligible set computations is also indicated.

2.2 Classical Metaheuristics

In this subsection we present approaches which follow well–known metaheuristic paradigms, namely genetic algorithms, tabu search, simulated annealing, and ant systems. Previously described metaheuristics for the RCPSP include Baar et al. [4], Boctor [8], Bouleimen and Lecocq [9], Cho and Kim [10], Hartmann [22], Kohlmorgen et al. [30], Lee and Kim [40], Leon and Ramamoorthy [41], Naphade et al. [47], Pinson et al. [57] as well as Sampson and Weiss [60].

Genetic algorithms (GAs). Alcaraz and Maroto [1] develop a genetic algorithm based on the activity list representation and the serial SGS. An additional gene decides whether forward or backward scheduling is employed when computing a schedule from an activity list. The crossover operator for activity lists is extended such that a child's activity list can be built up either in forward or in backward direction.

Coelho and Tavares [11] present a genetic algorithm which makes use of the activity list representation and the serial SGS. They suggest a new crossover operator for activity lists called late join function crossover. The current version of the working paper gives only the following rough verbal description of the operator: The late join function crossover constructs a new individual by "adopting the father solution and swapping each adjacent pair that is in reverse order in the mother."

Gonçalves and Mendes [21] use a random key representation and a modified parallel SGS. The modified parallel SGS determines all activities to be eligible which can be started up to the schedule time plus a delay time. The random key has twice the length of the number of activities.

Each entry is a random number. The first half of the entries biases the activity selection and the second half biases the delay time of the SGS.

Hartmann [23] proposes a so-called self–adapting genetic algorithm. This approach extends the activity list representation by adding a gene that determines whether the serial or the parallel SGS is to be used for transforming an activity list into a schedule (thereby, it is defined how the parallel SGS can be used as decoding procedure for activity lists). The choice of the more successful SGS is left to the inheritance and survival–of–the fittest mechanisms.

Hindi et al. [26] suggest a genetic algorithm based on the activity list representation, the serial SGS, and the related order–preserving crossover strategy (similar to Hartmann [22]). The initial population is produced by a pure random mechanism (whereas LFT–based sampling is used in [22]).

Toklu [70] develops a genetic algorithm which operates directly on schedules (i.e., a vector of start times). Since the genetic operators may produce infeasible offspring schedules, a penalty function is used which evaluates the constraint violations.

Valls et al. [75] extend the activity list–based genetic algorithm with forward–backward improvement of Valls et al. [76] (cf. Section 2.4) to what they call a hybrid genetic algorithm. They develop a peak crossover operator which uses properties of the schedule when recombining activity lists. Generally speaking, this operator aims at inheriting those parts of the parents' activity lists that correspond to peaks in the resource usage. Moreover, a second phase of the evolution is started from neighbors of the best individual found in the first phase. The neighbors are constructed with the approach used in Valls et al. [79] (cf. Section 2.3) which is applied to activity lists here.

Tabu search (TS). Artigues et al. [3] employ their insertion technique (cf. Section 2.4) in order to devise a tabu search procedure. Essentially, the method iteratively selects an activity which is first deleted from the schedule and afterwards re–inserted with a network flow–based insertion algorithm. Chosen activities as well as their resource predecessor and successors are elements of the tabu list.

Nonobe and Ibaraki [48] suggest a tabu search approach for a generalized variant of the RCPSP. Considering only the features that are relevant for the standard RCPSP, the heuristic employs the activity list representation, the serial SGS, shift moves, and a specific neighborhood reduction mechanism.

Thomas and Salhi [69] introduce a tabu search method which operates directly on schedules. They define three different moves. Since the resulting neighbor schedules may be infeasible, they employ a repair procedure to turn an infeasible schedule into a feasible one.

Simulated annealing (SA). Valls et al. [76] test a simulated annealing method in a paper that focuses on forward–backward improvement (cf. Section 2.4). The neighborhood definition is taken from Valls et al. [79] (cf. Section 2.3), where a neighbor is constructed by selecting the next activity either in the order of the original solution or by biased random sampling.

Ant systems. Merkle et al. [44] present the first application of ant systems (a metaheuristic strategy developed by Dorigo et al. [17]) to the RCPSP. In their approach, a single ant corresponds to one application of the serial SGS. The eligible activity to be scheduled next is selected using a weighted evaluation of the latest start time (LST) priority rule and so–called pheromones which represent the learning effect of previous ants. A pheromone value τ_{ij} describes how promising it seems to put activity j as the i–th activity into the schedule. Further features of the approach include separate ants for forward and backward scheduling and a 2–opt–based local search phase at the end of the heuristic.

2.3 Non–Standard Metaheuristics

This subsection is devoted to approaches which can be viewed as metaheuristics although they do not follow one of the classic metaheuristic schemes. It summarizes various non–standard local search and population–based methods which have been proposed to solve the RCPSP.

Local search–oriented approaches. Fleszar and Hindi [19] apply a variable neighborhood search (VNS, a metaheuristic strategy introduced by Mladenovic and Hansen [45]) to the RCPSP. They employ the activity list representation, the serial SGS, and an enhanced shift move which allows to shift activities together with their predecessors or successors. During run–time, their approach adds precedence relations on the basis of lower bound calculations.

Palpant et al. [54] embed forward–backward scheduling with the serial SGS and constraint–based optimization of partial schedules in a local search procedure. An initial solution is generated by applying forward–backward scheduling. Afterwards, a so–called block of activities, activities which are processed in parallel or contiguously, is selected randomly and constraint propagation is employed to determine for the selected activities a minimum makespan schedule under the constraints imposed by the non–selected activities. The entire method iterates between the selection of activities, optimization of partial schedules, and forward–backward scheduling until a stopping criterion is met.

Valls et al. [79] propose a two–phase local search method. It is based on the topological order representation (which is a special case of the random key representation) and the serial SGS. Two types of moves (applied either in forward or backward direction) make use of critical activities. A third move employs random sampling within a time window derived from the current solution. The second phase starts from the neighborhood of the best solutions obtained in the first phase. A solution's neighbor is constructed by selecting the next activity either in the order of the original solution or by biased random sampling.

Population–based approaches. Kochetov and Stolyar [29] devise an evolutionary algorithm which combines genetic algorithm, path relinking, and tabu search. Solutions are evolved and diversified in a genetic way. Evolution is done by choosing two solutions from the pool and constructing the path of solutions linking the selected solutions (path relinking, cf. Glover et al. [20]). The best solution from the path is chosen and improved via tabu search. The latter employs a neighborhood where the activity list is divided in three parts. For the first and the last part the serial SGS is employed while for the mid part the parallel SGS is used. The best solution from the tabu search is added to the population and the worst solution is removed from the population.

Valls et al. [77] employ the two–phase framework and the topological order representation of Valls et al. [79] which is described above. They introduce an implicit enumeration–based move to increase the resource utilization within a time interval. A binary (and hence crossover–like) operator is defined to produce convex combinations of solutions in the population.

Valls et al. [76] introduce several non–standard population–based schemes in a study which focuses on forward–backward improvement (cf. Section 2.4). Their schemes differ in the way parents are selected for reproduction and in the number of offspring produced for each pair of parent individuals. They use the priority value representation for which four operators are discussed.

The change operator (which is considered in our computational study) replaces random positions of the first parent with the corresponding positions of the second parent (this can be viewed as a variant of the uniform crossover).

2.4 Other Methods

This subsection summarizes those heristics that can neither be classified as X–pass construction methods nor as metaheuristics. Previously considered methods of this category are those of Alvarez–Valdes and Tamarit [2], Bell and Han [5], Mausser and Lawrence [43], Oguz and Bala [49], Pollack–Johnson [58], Shaffer et al. [64], and Sprecher [65].

Forward–backward improvement (FBI). Tormos and Lova [71] develop an iterative forward–backward heuristic. In each iteration, either the serial or the parallel SGS is used to generate a schedule by regret–based sampling with the latest finish time (LFT) priority rule. The resulting schedule is then improved by a backward–forward pass. That is, the SGS is first applied to the activity list obtained by ordering the activities in non–increasing order of the formerly scheduled finish times (backward pass) and then applied again to the activity list obtained by ordering the activities in non–decreasing order of their start times as obtained from the backward pass (forward pass).

This approach is enhanced in Tormos and Lova [72]. A so–called selective mechanism executes backward–forward improvement passes only if the schedule constructed by sampling is better than the average of the solutions generated by sampling so far. At most two backward–forward improvement passes are applied to such a schedule.

Tormos and Lova [73] present another method which includes some extensions of [72]. In addition to backward–forward improvement passes also forward–backward improvement passes can be executed. At most three passes are applied to the best 25 % of the schedules generated by sampling while to the next best 25 % at most two passes are applied. A separate stage constructs schedules by deterministic application of the LFT priority rule instead of sampling (and then proceeds with improvement passes).

Valls et al. [76] employ a so–called double justification procedure to improve schedules found by other heuristics. It shifts all activities within a schedule to the right and subsequently to the left in order to obtain a better schedule. In order to demonstrate its power and general applicability, it is tested by adding it to various sampling methods and metaheuristics from the literature as well as to new approaches (cf. Sections 2.2 and 2.3). The best results are obtained from extending the activity list–based genetic algorithm of Hartmann [22].

Due to the similarity of the justification procedure of Valls et al. [76] and the backward–forward pass of Tormos and Lova [71], we refer to both approaches by the notion of forward–backward improvement (FBI).

Further heuristics. Artigues et al [3] have devised an insertion technique based on the parallel SGS and the worst case slack (WCS) priority rule. Other piority rules for the parallel SGS are possible. The method is a multi–pass (MP) approach which solves an RCPSP instance n times where at time j ($j = 1, \ldots, n$) activity j is deleted from the instance. Afterwards a network flow–based insertion algorithm is applied in order to perform a makespan–minimal extension of the schedule with activity j. From the at most n different schedules, the schedule with the minimum makespan is selected. The complexity of the entire method is $O(n^3 \cdot m)$ where m denotes the number of renewable resources.

Möhring et al. [46] propose a Lagrange heuristic. The method first generates an upper bound of the project makespan by employing a multi priority rule method. Afterwards, a Lagrange relaxation (LR) and a list scheduling heuristic are invoked iteratively, in order to generate lower and upper bounds for the RCPSP. The Lagrange relaxation of the RCPSP is solved in polynomial time and the precedence feasible start times are employed as input for a list scheduling procedure. The latter generates a number of different precedence and resource feasible schedules by scheduling the activities in non–increasing order of their start time plus the processing times multiplied by a constant δ ($0 \leq \delta \leq 1$).

Sprecher [66] proposes a network decomposition technique which incorporates exact methodologies into heuristic search. In each iteration, an initial schedule is generated by biased random sampling employing the latest finish time (LFT) priority rule. On the basis of the generated schedule, the problem is divided into subproblems which are solved with the truncated version of the branch–and–bound method of Sprecher [65]. The schedules of the subprojects are concatenated to an improved schedule for the overall project.

3 Computational Comparison

3.1 Test Design

This section presents a computational comparison of many of the heuristics that have been reviewed in Section 2. We use the same test sets and the same stopping criterion as in our previous comparison [24]. We employ the three test sets J30, J60, and J120 that have been constructed by the instance generator ProGen (see Kolisch et al. [38]). The projects in these test sets consist of 30, 60, and 120 activities, respectively. Each set has been generated using a full factorial design of parameters which determine the characteristics of the resource and precedence constraints. In total, we have 480 instances with 30 activities, 480 instances with 60 activities, and 600 instances with 120 activities. The instances have been used by many researchers, and they are available from the project scheduling library PSPLIB in the internet (for more detailed information on the test sets, we refer to Kolisch and Sprecher [37]).

As in our last study, we limited the number of generated schedules in the heuristics in order to provide the basis for the comparison. This is based on the assumption that the computational effort for constructing one schedule is similar in most heuristics. This holds in particular for methods which apply the serial or parallel SGS; one pass of an SGS with one start time assignment per activity counts as one schedule. As in our previous comparison, we have selected 1,000 and 5,000 schedules as stopping criteria. Since the speed of computers has increased, larger numbers of schedules can be computed within acceptable run–times. Therefore, we have selected 50,000 schedules as an additional limit.

The advantage of this stopping criterion is that it is independent of the computer platform. Therefore, all heuristics could be tested by their author(s) using the original implementation and the best configuration. Also, future studies can easily make use of the benchmark results presented here by applying the same stopping criterion. Moreover, the tests are independent of compilers and implementational skills, thus we evaluate heuristic concepts rather than program codes. However, the stopping criterion also has a few drawbacks. First, it cannot be applied to all heuristics. For example, it cannot be used if backtracking steps or mixed integer program–based (MIP) methods are included. Nevertheless, the stopping criterion is applicable to most heuristics that have been proposed. Second, different heuristics may require different computation times to compute one schedule. For example, the serial SGS is faster within a metaheuristic based on the activity list representation than within a priority rule–based sampling method (the former simply picks the next activity to be scheduled from the list while the latter has to compute the eligible set, priority values, and selection probabilities). But these differences are rather small—using a time limit with different computers, operating systems, programming languages, and implementational skills would surely lead to much greater inaccuracies. Summing up, we believe that limiting the number of schedules is the best criterion available for such a broad comparison.

After the presentation of the computational results and a summary of the main observations in Subsection 3.2, we close this section with a critical discussion of the stopping criterion in Subsection 3.3.

3.2 Experimental Results

3.2.1 Performance of the Heuristics

The computational results of the tested heuristics can be found in Tables 1, 2, and 3 for the instance sets J30, J60, and J120, respectively. These tables extend those of our previous comparison study [24]. Each heuristic is briefly described by a few keywords, the SGS employed, and the reference. For the J30 set, the results are given in terms of average deviation from the optimal solution. For the other two sets, some of the optimal solutions are unknown. Thus, the average deviation from the well–known critical path–based lower bound is reported.

Each of the three tables is divided into three blocks. The first block considers the test design described in the previous subsection. Here, the results for the three stopping criteria (maximum of 1,000, 5,000, and 50,000 schedules, respectively) are given. Many researchers used our test design in their papers; if results according to our test design were available, we cite them here. Since these papers usually did not cover all test sets and/or stopping criteria, some researchers sent us additional results for this study which are given here as well (note that we accepted only additional results, but no results that improved previously published ones). In order to obtain these additional results, only adjustments of parameter values were allowed but no methodological modifications. These restrictions were necessary to ensure that the results presented here are consistent with the description of the heuristics in the cited papers. In some papers, our stopping criteria were not applied correctly (this was the case for [48], [71], [72], [73]). In these cases, the authors sent us corrected results which are reported here.

Some researchers did not provide results according to our stopping criteria (note that for some methods it is impossible to count the number of schedules in the way required by our criterion, consider in particular the approaches of Möhring et al. [46], Sprecher [66], and Valls et al. [77]). As long as they used the test sets employed here, we have added the results given in their papers in the blocks two and three of the tables. The second block of each table contains algorithms where the average deviation together with the average and maximal number of schedules required is given. Contrary, 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. Of course, for the heuristics reported in blocks two and three, the computational effort has to be taken into account when interpreting the results; observe that the reported effort greatly varies between the different methods.

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 5,000 schedules. Let us also remark that in general only the best performing heuristic of a paper has been considered here. Only if a paper considers substantially different new approaches or if different parameter settings were employed in order to obtain additional insight, more than one heuristic is considered.

In our previous study, the best procedures were those of Hartmann [22] as well as Bouleimen and Lecocq [9]. Hence, these two heuristics are considered as benchmark. Four new approaches from the first block (i.e. where we limited the number of schedules to 1,000, 5,000, and 50,000, respectively), namely the two procedures of Valls et al. [75, 76] as well as the algorithms of Kochetov and Stolyar [29] and Hartmann [23], consistently outperform these two benchmark heuristics in all instance classes (J30, J60, and J120) and for all numbers of schedules (1,000, 5,000, and 50,000). Table 4 gives a survey of these best–performing heuristics. Also several other new methods listed in the first block perform very well as they outperform the two former benchmark heuristics in some of the instance classes and for some of the numbers of schedules. Finally, some of the heuristics which were not tested using our test design (see the second and third block of Tables 1 – 3) produced good results as well, but since their computational effort is not easily comparable (and sometimes obviously very high), we do not consider them in the following discussion.

3.2.2 Characteristics of Good Heuristics

In our previous study [24] we have resumed that "the best heuristics in our computational analysis are metaheuristics based on the activity list representation and the serial SGS." Furthermore, we have proposed that the use of the parallel scheduling scheme might be beneficial for large problems. Based on the new results (cf. Tables 1 – 3) in general and the best performing heuristics (cf. Table 4) in particular we can add the following observations.

Metaheuristic and Population–based Approaches. Again, all heuristics listed in Table 4 are metaheuristics based on activity lists. Furthermore, each of these four methods is a population–based approach (either a classical genetic algorithm or a non–standard metaheuristic which incorporates features of genetic algorithms). The genetic algorithm scheme is enhanced by further components such as other metaheuristics, path relinking, forward–backward improvement, self–adapting mechanisms, or non–standard crossover techniques, which leads to better results than a basic genetic algorithm such as that of Hartmann [22].

The best approach in our study that is not a metaheuristic is the sampling method with forward–backward improvement of Tormos and Lova [73]. It is interesting to note that this approach leads to excellent solutions after computing 1,000 schedules but produces only relatively small improvements when computing more schedules. In contrast, the metaheuristics show much larger improvements of the solution quality when computing more schedules—which is due to the fact that they exploit learning effects during run–time.

Schedule Generation Scheme (SGS). In our last study [24] we have stressed the success of the serial SGS. With all methods given in Table 4 employing the serial SGS this observation still holds. Now several new methods (including two of the four best procedures of Table 4) use both the serial and the parallel SGS. Since neither the serial nor the parallel SGS is consistently superior (which is demonstrated by the results of the sampling methods), it appears to be a good idea to employ both. In fact, the genetic algorithm of Hartmann [23] extends the previous approach of Hartmann [22] by using the parallel SGS in addition to the serial one, which clearly improves the results. While most methods that use both SGS construct a single schedule either with the serial or with the parallel one, Kochetov and Stolyar [29] take a completely different approach. They

			max. #schedules			
Algorithm	SGS	Reference	1,000	5,000	50,000	
GA, TS – path relinking	both	Kochetov, Stolyar [29]	0.10	0.04	0.00	
GA – hybrid, FBI	serial	Valls et al. [75]	0.27	0.06	0.02	
GA – forw.-backward	serial	Alcaraz, Maroto [1]	0.33	0.12	–	
GA – FBI	serial	Valls et al. [76]	0.34	0.20	0.02	
sampling – LFT, FBI	both	Tormos, Lova [73]	0.25	0.13	0.05	
TS – activity list	serial	Nonobe, Ibaraki [48]	0.46	0.16	0.05	
sampling – LFT, FBI	both	Tormos, Lova [71]	0.30	0.16	0.07	
GA – self-adapting	both	Hartmann [23]	0.38	0.22	0.08	
GA – activity list	serial	Hartmann [22]	0.54	0.25	0.08	
sampling – LFT, FBI	both	Tormos, Lova [72]	0.30	0.17	0.09	
sampling – random, FBI	serial	Valls et al. [76]	0.46	0.28	0.11	
SA – activity list	serial	Bouleimen, Lecocq [9]	0.38	0.23	–	
GA – late join	serial	Coelho, Tavares [11]	0.74	0.33	0.16	
sampling – adaptive	both	Schirmer [61]	0.65	0.44	–	
TS – schedule scheme	related	Baar et al. [4]	0.86	0.44	–	
sampling – adaptive	both	Kolisch, Drexl [34]	0.74	0.52	–	
GA – random key	serial	Hartmann [22]	1.03	0.56	0.23	
sampling – LFT, $\alpha = 1$	serial	Kolisch [33]	0.83	0.53	0.27	
sampling – global	serial	Coelho, Tavares [11]	0.81	0.54	0.28	
sampling – random	serial	Kolisch [31]	1.44	1.00	0.51	
sampling – LFT, $\alpha = 3$	serial	Kolisch [33]	1.05	0.78	0.56	
GA – priority rule	serial	Hartmann [22]	1.38	1.12	0.88	
sampling – WCS	parallel	Kolisch [32, 33]	1.40	1.28	–	
sampling – LFT, $\alpha = 1$	parallel	Kolisch [33]	1.40	1.29	1.13	
sampling – random	parallel	Kolisch [31]	1.77	1.48	1.22	
GA – problem space	mod. par.	Leon, Ramamoorthy [41]	2.08	1.59	–	
				#schedules		
			result	average	max.	
GA – activity list	serial	Hindi et al. [26]	0.37	1,683	3,068	
MP – network flow	parallel	Artigues et al. [3]	1.74	30	30	
				CPU-time (sec)		
			result	average	max.	computer
decompos. & local opt.	serial	Palpant et al. [54]	0.00	10.26	123.0	2.3 GHz
VNS – activity list	serial	Fleszar, Hindi [19]	0.01	0.64	5.9	1.0 GHz
local search – critical	serial	Valls et al. [79]	0.06	1.61	6.2	400 MHz
population–based	serial	Valls et al. [77]	0.10	1.16	5.5	400 MHz
network decomposition	–	Sprecher [66]	0.12	2.75	39.7	166 MHz

Table 1: Average deviations (%) from optimal makespan — ProGen set $J = 30$

Algorithm	SGS	Reference	max. #schedules			
			1,000	5,000	50,000	
GA – hybrid, FBI	serial	Valls et al. [75]	11.56	11.10	10.73	
GA, TS – path relinking	both	Kochetov, Stolyar [29]	11.71	11.17	10.74	
GA – FBI	serial	Valls et al. [76]	12.21	11.27	10.74	
GA – self-adapting	both	Hartmann [23]	12.21	11.70	11.21	
GA – activity list	serial	Hartmann [22]	12.68	11.89	11.23	
sampling – LFT, FBI	both	Tormos, Lova [73]	11.88	11.62	11.36	
sampling – LFT, FBI	both	Tormos, Lova [72]	12.14	11.82	11.47	
GA – forw.-backward	serial	Alcaraz, Maroto [1]	12.57	11.86	–	
sampling – LFT, FBI	both	Tormos, Lova [71]	12.18	11.87	11.54	
SA – activity list	serial	Bouleimen, Lecocq [9]	12.75	11.90	–	
TS – activity list	serial	Nonobe, Ibaraki [48]	12.97	12.18	11.58	
sampling – random, FBI	serial	Valls et al. [76]	12.73	12.35	11.94	
sampling – adaptive	both	Schirmer [61]	12.94	12.58	–	
GA – late join	serial	Coelho, Tavares [11]	13.28	12.63	11.94	
GA – random key	serial	Hartmann [22]	14.68	13.32	12.25	
GA – priority rule	serial	Hartmann [22]	13.30	12.74	12.26	
sampling – adaptive	both	Kolisch, Drexl [34]	13.51	13.06	–	
sampling – WCS	parallel	Kolisch [32, 33]	13.66	13.21	–	
sampling – global	serial	Coelho, Tavares [11]	13.80	13.31	12.83	
sampling – LFT, $\alpha = 3$	serial	Kolisch [33]	13.75	13.34	12.84	
sampling – LFT, $\alpha = 1$	parallel	Kolisch [33]	13.59	13.23	12.85	
TS – schedule scheme	related	Baar et al. [4]	13.80	13.48	–	
GA – problem space	mod. par.	Leon, Ramamoorthy [41]	14.33	13.49	–	
sampling – LFT, $\alpha = 1$	serial	Kolisch [33]	13.96	13.53	12.97	
sampling – random	parallel	Kolisch [31]	14.89	14.30	13.66	
sampling – random	serial	Kolisch [31]	15.94	15.17	14.22	
				#schedules		
			result	average	max.	
VNS – activity list	serial	Fleszar, Hindi [19]	10.94	152,503	1.7 mio	
MP – network flow	parallel	Artigues et al. [3]	14.20	60	60	
				CPU-time (sec)		
			result	average	max.	computer
decompos. & local opt.	serial	Palpant et al. [54]	10.81	38.8	223.0	2.3 GHz
population–based	serial	Valls et al. [77]	10.89	3.7	22.6	400 MHz
local search – critical	serial	Valls et al. [79]	11.45	2.8	14.6	400 MHz
network decomposition	–	Sprecher [66]	11.61	460.2	4311.5	166 MHz
TS – network flow	parallel	Artigues et al. [3]	12.05	3.2	–	450 MHz
LR – activity list	both, mod.par.	Möhring et al. [46]	15.60	6.9	57	200 MHz

Table 2: Average deviations (%) from critical path lower bound — ProGen set $J = 60$

			max. #schedules			
Algorithm	SGS	Reference	1,000	5,000	50,000	
GA – hybrid, FBI	serial	Valls et al. [75]	34.07	32.54	31.24	
GA – FBI	serial	Valls et al. [76]	35.39	33.24	31.58	
GA, TS – path relinking	both	Kochetov, Stolyar [29]	34.74	33.36	32.06	
population–based – FBI	serial	Valls et al. [76]	35.18	34.02	32.81	
GA – self-adapting	both	Hartmann [23]	37.19	35.39	33.21	
sampling – LFT, FBI	both	Tormos, Lova [73]	35.01	34.41	33.71	
ant system	serial	Merkle et al. [44]	–	35.43	–	
GA – activity list	serial	Hartmann [22]	39.37	36.74	34.03	
sampling – LFT, FBI	both	Tormos, Lova [72]	36.24	35.56	34.77	
sampling – LFT, FBI	both	Tormos, Lova [71]	36.49	35.81	35.01	
GA – forw.-backward	serial	Alcaraz, Maroto [1]	39.36	36.57	–	
TS – activity list	serial	Nonobe, Ibaraki [48]	40.86	37.88	35.85	
GA – late join	serial	Coelho, Tavares [11]	39.97	38.41	36.44	
sampling – random, FBI	serial	Valls et al. [76]	38.21	37.47	36.46	
SA – activity list	serial	Bouleimen, Lecocq [9]	42.81	37.68	–	
GA – priority rule	serial	Hartmann [22]	39.93	38.49	36.51	
sampling – adaptive	both	Schirmer [61]	39.85	38.70	–	
sampling – LFT, $\alpha = 1$	parallel	Kolisch [33]	39.60	38.75	37.74	
sampling – WCS	parallel	Kolisch [32, 33]	39.65	38.77	–	
GA – random key	serial	Hartmann [22]	45.82	42.25	38.83	
sampling – LFT, $\alpha = 3$	serial	Kolisch [33]	41.27	40.38	39.34	
sampling – adaptive	both	Kolisch, Drexl [34]	41.37	40.45	–	
sampling – global	serial	Coelho, Tavares [11]	41.36	40.46	39.41	
GA – problem space	mod. par.	Leon, Ramamoorthy [41]	42.91	40.69	–	
sampling – LFT, $\alpha = 1$	serial	Kolisch [33]	42.84	41.84	40.63	
sampling – random	parallel	Kolisch [31]	44.46	43.05	41.44	
sampling – random	serial	Kolisch [31]	49.25	47.61	45.60	
				#schedules		
			result	average	max.	
VNS – activity list	serial	Fleszar, Hindi [19]	33.10	1.9 mio	10.8 mio	
MP – network flow	parallel	Artigues et al. [3]	39.34	120	120	
				CPU-time (sec)		
			result	average	max.	computer
population–based	serial	Valls et al. [77]	31.58	59.4	264.0	400 MHz
decompos. & local opt.	serial	Palpant et al. [54]	32.41	207.9	501.0	2.3 GHz
local search – critical	serial	Valls et al. [79]	34.53	17.0	43.9	400 MHz
LR – activity list	both, mod. par.	Möhring et al. [46]	36.00	72.9	654.0	200 MHz
TS – network flow	parallel	Artigues et al. [3]	36.16	67.0	–	450 MHz
network decomposition	–	Sprecher [66]	39.29	458.5	1511.3	166 MHz

Table 3: Average deviations (%) from critical path lower bound — ProGen set $J = 120$

Reference	Type	Representation	SGS	FBI
Valls et al. [75]	genetic algorithm	activity list	serial	●
Valls et al. [76]	genetic algorithm	activity list	serial	●
Kochetov and Stolyar [29]	population–based	activity list	serial & parallel	–
Hartmann [23]	genetic algorithm	activity list	serial & parallel	–

Table 4: Characteristics of the best heuristics

employ each of the SGS to construct just a part of a schedule. Considering their excellent results, we can state that there are different ways to successfully employ both SGS within one heuristic.

Forward–backward improvement (FBI). A noteworthy trend is the use of forward–backward improvement in a surprisingly large number of recent papers (Tormos and Lova [71, 72, 73], Valls et al. [75, 76]). These heuristics are among the best in our study. The power of forward–backward improvement is also demonstrated by Valls et al. [76] who add it to the simplest project scheduling heuristic, i.e. pure random sampling. Tables 1 – 3 show that adding forward–backward improvement to random sampling leads to much better results than adding a priority rule. In fact, on the set J120 this (still remarkably simple) procedure obtains better results than several more complex approaches including that of Bouleimen and Lecocq [9], one of the best methods in our last study [24]. Moreover, as shown by Valls et al. [77], forward–backward improvement can easily be added to any existing heuristic for the RCPSP. This makes forward–backward improvement a promising building block of heuristics.

Sampling. Simple sampling approaches as originally proposed by Kolisch [31, 33] are not competitive anymore. A noteworthy improvement can be obtained for biased random sampling approaches by properly adjusting the parameter α which governs the amount of bias for the activity priorities (cf. also the study of Tormos and Lova [73]). It can be seen for the biased sampling approach with priority rule LFT [33] that for the J30 instance set $\alpha = 1$ gives better results while for the J120 instance set $\alpha = 3$ is superior. I.e. when problems become larger, the amount of bias should be reduced. This is also useful if biased sampling is employed to compute initial solutions for metaheuristics.

3.2.3 Stability of Heuristics

We consider a heuristic stable if it does not change its rank according to the average deviation for different instance and schedule classes as documented in Tables 1 – 3. In order to obtain a measure of stability we calculated for each of the 19 heuristics which have been tested on all 9 instance and schedule classes (3 instance classes $\times$ 3 schedule classes) the variance of the rank, respectively. Table 5 gives the average rank and the rank variance of the heuristics. The heuristics are ordered according to non–decreasing rank variance. Figure 1 gives a graphical representation. Stable heuristics have a small rank variance and hence are situated at the bottom line. We can see that the hybrid genetic algorithm of Valls et al. [75] and the population–based approach of Kochetov and Stolyar [29] are procedures which show consistently superior results (i.e. they have the lowest average rank and the lowest rank variance) whereas the simple sampling heuristics presented in Kolisch [31, 33] show consistently inferior results (i.e. they have very high average ranks and a rank variance which is amongst the lowest). The genetic algorithm of Coelho and Tavares [11]

Algorithm	SGS	reference	Mean Rank	Rank Variance
GA – late join	serial	Coelho, Tavares [11]	10.89	0.32
sampling – random	parallel	Kolisch [31]	18.22	0.40
GA – hybrid, FBI	serial	Valls et al. [75]	1.44	0.47
GA, TS – path relinking	both	Kochetov, Stolyar [29]	1.89	0.54
sampling – LFT, $\alpha = 3$	serial	Kolisch [33]	14.89	0.54
sampling – random, FBI	serial	Valls et al. [76]	9.44	0.91
sampling – LFT, FBI	both	Tormos, Lova [73]	3.78	1.28
sampling – global	serial	Coelho, Tavares [11]	14.22	1.28
GA – self-adapting	both	Hartmann [23]	5.89	1.88
sampling – LFT, FBI	both	Tormos, Lova [71]	6.11	2.10
sampling – random	serial	Kolisch [31]	18.11	2.10
GA – activity list	serial	Hartmann [22]	7.78	2.17
sampling – LFT, FBI	both	Tormos, Lova [72]	6.00	2.22
sampling – LFT, $\alpha = 1$	serial	Kolisch [33]	15.22	3.51
GA – FBI	serial	Valls et al. [76]	3.78	3.73
GA – random key	serial	Hartmann [22]	14.78	4.17
GA – priority rule	serial	Hartmann [22]	13.56	5.14
sampling – LFT, $\alpha = 1$	parallel	Kolisch [33]	14.56	6.91
TS – activity list	serial	Nonobe, Ibaraki [48]	8.44	7.36

Table 5: Mean rank and rank variance of the heuristics

shows consistently mediocre results (i.e. it has an average rank which ranges in the middle while showing a low rank variance). The genetic algorithm of Hartmann [22] extended with the forward–backward improvement approach of Valls et al. [76] has according to its average rank the third best performance but shows at the same time a rather high variance of the performance. We assume that the stability of heuristics is rather a question of careful parameter selection w.r.t. to all instance and schedule classes than an inherent characteristic of the heuristic.

3.3 Critical Remarks on the Use of the Test Design

In the recent years, many researchers have adopted our test design when evaluating their heuristics. This allowed them to compare their approaches with many other heuristics from the literature. Unfortunately, our test design has not been used correctly in a few cases because the number of schedules has not been counted accurately. In [48], this was probably due to a misunderstanding (which may have been caused by the fact that the columns of the tables in our last paper were ambiguously entitled "iterations" instead of "number of schedules"). In [71, 72, 73], the authors counted one backward–forward improvement pass as one schedule although such a pass corresponds to two applications of an SGS (one backward and one forward) and thus to an effort of two constructed schedules. It should be emphasized that the authors of the four papers mentioned above have sent us corrected results (which are displayed in Tables 1 – 3).

In order to avoid misunderstandings or other interpretations in the future, we wish to clarify again the assumptions of our stopping criterion. The heuristic to be evaluated is stopped after a certain maximal number of schedules have been constructed (1,000, 5,000, or 50,000 schedules). Generating one schedule corresponds to (at most) one start time assigment per activity, as done by an SGS (regardless of the computational effort, cf. also the discussion in Section 3.1). Each schedule must be counted (including, e.g., constructed but then rejected neighbor schedules in local search methods). A schedule must be counted as one whole schedule even if it is not

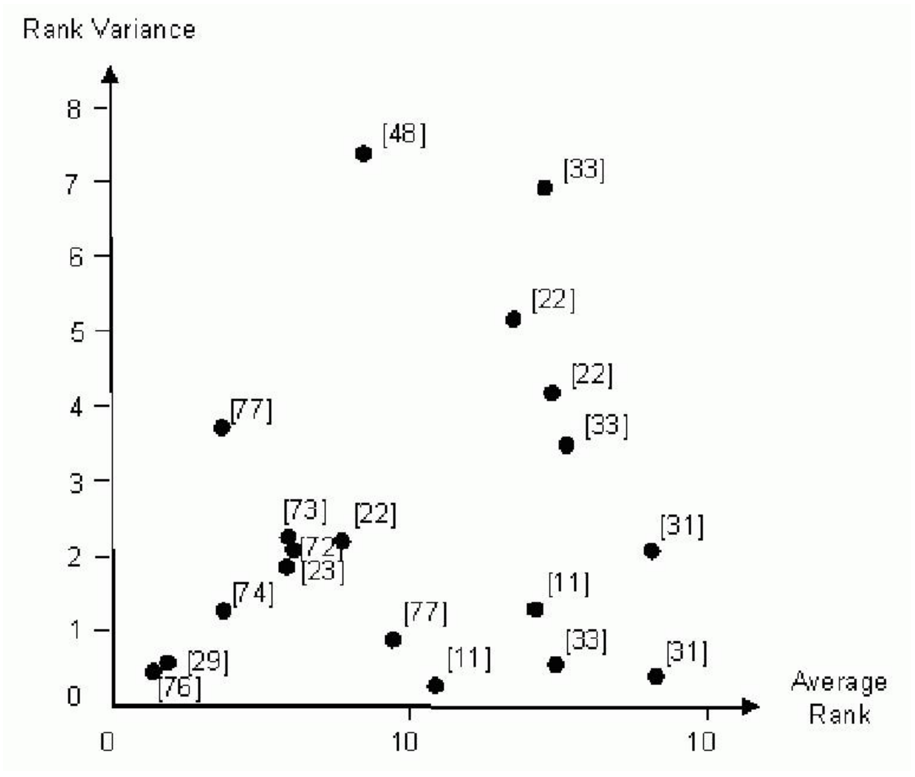

Figure 1: Average rank and rank variance of the heuristics

constructed completely (in fact, methods in our comparison such as [23] employ approaches to abort the completion of inferior schedules to reduce the computation time, but this does not affect the schedule count). We ask that researchers who wish to cite results from our study either observe this stopping criterion accurately when testing their own heuristics or refrain from using our comparison results. Any creative interpretation of the stopping criterion would be unfair.

Researchers who cannot apply the criterion (because their heuristics do not build schedules with an SGS) might still wish to incorporate our comparison into their studies. They may find a suitable way to do so which is, of course, fine. We just request that results are not included in a way that might suggest that a certain number of schedules have been generated if the latter is not true.

4 Conclusions

During the years since our last survey [24], research on project scheduling heuristics has led to both a wide range of new methodological ideas and a substantial improvement of computational results with respect to our test design. In our last comparison, the best performing approaches were two–phase metaheuristics, where the first phase computes initial solutions by a priority rule–based sampling method and the second phase applies a metaheuristic strategy on the basis of the activity list representation (see Bouleimen and Lecocq [9], Hartmann [22]). Now four new papers, namely the two of Valls et al. [75, 76] as well as those of Kochetov and Stolyar [29] and Hartmann [23], report consistently better results. Also, several other new methods produced excellent results.

Starting with the recent work of Tormos and Lova [71], several papers applied a forward–

backward improvement technique (also called justification) to improve schedules constructed by X–pass methods or metaheuristics. This simple procedure—the activities are shifted to the right within the schedule and then to the left—produces excellent results and can be combined with almost any other approach. The best heuristic of this kind is that of Valls et al. [75]. In addition to further features, they incorporated forward–backward improvement to the basic genetic algorithm of Hartmann [22] as a third phase, which leads to much better results than the original genetic algorithm at the same overall computational effort. This three–phase method is one of the best performing heuristics in our comparison. We expect that forward–backward improvement will become an important component in future heuristics for the RCPSP.

Another main research focus has been on metaheuristics. Genetic algorithms and tabu search have been the most popular strategies. Moreover, the first application of ant systems to the RCPSP as well as various non–standard local search– and population–based schemes have been proposed. The activity list has been the most widely used representation. It has usually been employed in its classical form, while a few researchers have extended it. Considering its success, one may in fact wonder why some recent metaheuristics still employ the direct schedule representation with operators that are very likely to produce infeasible solutions.

Priority rule–based X–pass methods have attracted less attention. As already pointed out in our last survey, they are inferior to metaheuristic approaches which are capable of learning. Finally, several heuristics have been developed which can neither be classified as X–pass methods nor as metaheuristics. Such approaches include a Lagrange method, the above–mentioned forward–backward improvement procedures, and strategies based on decomposition and optimization. In the latter approaches, the problem is divided into subproblems which are then optimized separately and concatenated. Although these methods have not yet yielded competitive results, we view these approaches as interesting and promising.

Considering the development during the last years, a general observation is that the recently proposed heuristics contain more components than earlier procedures. Many methods consider both scheduling directions instead of only forward scheduling, both SGS instead of only one, more than one type of local search operator, or even more than one type of metaheuristic strategy. While recombining existing ideas may occasionally seem less creative than developing new ideas, several of the integration efforts have put well–known techniques into a new and promising context, and the results have often been encouraging.

In this paper, we have also given a critical discussion of our comparison study. Unfortunately, our comparison has not been used appropriately in a few papers because the number of schedules (which is the basis for the comparison) was not counted accurately. We expect all researchers either to use our comparison exactly according to the philosophy of the test design or not to use it all.

With the standard test sets which are available on the internet and the computer–independent stopping criterion, the test design and benchmark results of this paper can easily be used by other researchers in future studies. We believe that the test design and the benchmark results from our previous study have already contributed to significant improvements of heuristic results. On the other hand, such easy comparisons might motivate researchers rather to improve the benchmark results with modifications of existing methods than to develop new and innovative ideas. This is not our intention, and we would like to emphasize the value of new methodologies even if they are not fully competitive.

Acknowledgements. We are indebted to the following researchers who provided us with results according to our test design: Javier Alcaraz and Concepcion Maroto (Technial University of Valencia, Spain), Christian Artigues, Philippe Michelon, and Mireille Palpant (University of Avignon, France), José Silva Coelho and Luis Valadares Tavares (Technical University of Lisbon, Portugal), Yuri Kochetov and Artem Stolyar (Sobolev Institute of Mathematics, Russia), Koji Nonobe and Toshihide Ibaraki (University of Kyoto, Japan), Pilar Tormos and Antonio Lova (Technical University of Valencia, Spain), as well as Vincente Valls, Sacramento Quintanilla, and Francisco Ballestin (Technical Universtiy of Valencia, Spain).

Abbreviations

FBI	Forward–backward improvement
GA	Genetic algorithm
LFT	Latest finish time (priority rule)
LR	Lagrange Relaxation
LST	Latest start time (priority rule)
MIP	Mixed Integer Program
MP	Multi–Pass
SGS	Schedule generation scheme
TS	Tabu search
VNS	Variable neighborhood search
WCS	Worst case slack (priority rule)

References

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

[2] R. Alvarez-Valdés and J. M. Tamarit. Heuristic algorithms for resource–constrained project scheduling: A review and an empirical analysis. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*, pages 113–134. Elsevier, Amsterdam, the Netherlands, 1989.

[3] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for statric and dynamic resource–constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[4] T. Baar, P. Brucker, and S. Knust. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: Advances and trends in local search paradigms for optimization*, pages 1–8. Kluwer Academic Publishers, 1998.

[5] C. E. Bell and J. Han. A new heuristic solution method in resource-constrained project scheduling. *Naval Research Logistics*, 38:315–331, 1991.

[6] J. Błażewicz, J. Lenstra, and A. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

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

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

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

[10] J. H. Cho and Y. D. Kim. A simulated annealing algorithm for resource-constrained project scheduling problems. *Journal of the Operational Research Society*, 48:736–744, 1997.

[11] J. Coelho and L. Tavares. Comparative analysis of meta–heuricstics for the resource constrained project scheduling problem. Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal, 2003.

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

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

[14] E. M. Davies. An experimental investigation of resource allocation in mulitactivity projects. *Operations Research Quarterly*, 24:587–591, 1973.

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

[16] E. Demeulemeester and W. Herroelen. *Project scheduling – A research handbook*. Kluwer Academic Publishers, Boston, 2002.

[17] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26:29–41, 1996.

[18] E. A. Elsayed. Algorithms for project scheduling with resource constraints. *International Journal of Production Research*, 20:95–103, 1982.

[19] K. Fleszar and K. Hindi. Solving the resource–constrained project scheduling problem by a variable neighbourhood search. Technical report, Brunel University, Department of Systems Engineering, 2000.

[20] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[21] J. Gonçalves and J. Mendes. A random key based genetic algorithm for the resource–constrained project scheduling problem. Technical report, Departamento de Engenharia, Universidade do Porto, 2003.

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

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

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

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

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

[27] R. Klein. Bidirectional planning — Improving priority rule based heuristics for scheduling resource-constrained projects. Schriften zur Quantitativen Betriebswirtschaftslehre 10/98, Technische Universität Darmstadt, Germany, 1998.

[28] R. Klein. Project scheduling with time–varying resource constraints. *International Journal of Production Research*, 38(16):3937–3952, 2000.

[29] Y. Kochetov and A. Stolyar. 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*, Russia, 2003.

[30] U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:203–319, 1999.

[31] R. Kolisch. *Project scheduling under resource constraints — Efficient heuristics for several problem classes*. Physica, Heidelberg, 1995.

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

[33] R. Kolisch. Serial and parallel resource–constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

[34] R. Kolisch and A. Drexl. Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43:23–40, 1996.

[35] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, *Project scheduling: Recent models, algorithms and applications*, pages 147–178. Kluwer Academic Publishers, 1999.

[36] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *OMEGA International Journal of Management Science*, 29(3):249–272, 2001.

[37] R. Kolisch and A. Sprecher. PSPLIB — A project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

[38] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource–constrained project scheduling problems. *Management Science*, 41(10):1693–1703, 1995.

[39] S. R. Lawrence. Resource constrained project scheduling – A computational comparison of heuristic scheduling techniques. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1985.

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

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

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

[43] H. E. Mausser and S. R. Lawrence. Exploiting block structure to improve resource–constrained project schedules. Technical report, Graduate School of Business Administration, University of Colorado, 1995.

[44] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6:333–346, 2002.

[45] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

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

[47] K. S. Naphade, S. D. Wu, and R. H. Storer. Problem space search algorithms for resource-constrained project scheduling. *Annals of Operations Research*, 70:307–326, 1997.

[48] K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 557–588. Kluwer Academic Publishers, 2002.

[49] O. Oguz and H. Bala. A comparative study of computational procedures for the resource constrained project scheduling problem. *European Journal of Operational Research*, 72:406–416, 1994.

[50] L. Özdamar and G. Ulusoy. A local constraint based analysis approach to project scheduling under general resource constraints. *European Journal of Operational Research*, 79:287–298, 1994.

[51] L. Özdamar and G. Ulusoy. A survey on the resource–constrained project scheduling problem. *IIE Transactions*, 27:574–586, 1995.

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

[53] L. Özdamar and G. Ulusoy. 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, 1996.

[54] M. Palpant, C. Artigues, and P. Michelon. Lssper: Solving the resource–constrained project scheduling problem with large neighbourhood search. Technical Report LIA report 255, Laboratoire d'Informatique d'Avignon, Avignon, 2003.

[55] J. H. Patterson. Alternate methods of project scheduling with limited resources. *Naval Research Logistics Quarterly*, 20:767–784, 1973.

[56] J. H. Patterson. Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly*, 23:95–123, 1976.

[57] E. Pinson, C. Prins, and F. Rullier. Using tabu search for solving the resource-constrained project scheduling problem. In *Proceedings of the fourth international workshop on project management and scheduling*, pages 102–106. Leuven, Belgium, 1994.

[58] B. Pollack-Johnson. Hybrid structures and improving forecasting and scheduling in project management. *Journal of Operations Management*, 12:101–117, 1995.

[59] A. Pritsker, L. Watters, and P. Wolfe. Multiproject scheduling with limited resources: A zero–one programming approach. *Management Science*, 16:93–107, 1969.

[60] S. E. Sampson and E. N. Weiss. Local search techniques for the generalized resource-constrained project scheduling problem. *Naval Research Logistics*, 40:665–675, 1993.

[61] A. Schirmer. Case–based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics*, 47:201–222, 2000.

[62] A. Schirmer and S. Riesenberg. Parameterized heuristics for project scheduling — Biased random sampling methods. Manuskripte aus den Instituten für Betriebswirtschaftslehre 456, Universität Kiel, Germany, 1997.

[63] A. Schirmer and S. Riesenberg. Class-based control schemes for parameterized project scheduling heuristics. Manuskripte aus den Instituten für Betriebswirtschaftslehre 471, Universität Kiel, Germany, 1998.

[64] L. R. Shaffer, J. B. Ritter, and W. L. Meyer. *The critical-path method*. McGraw-Hill, New York, 1965.

[65] A. Sprecher. Scheduling resource-constrained projects competetively at modest memory requirements. *Management Science*, 46:710–723, 2000.

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

[67] A. Thesen. Heuristic scheduling of activities under resource and precedence restrictions. *Management Science*, 23:412–422, 1976.

[68] P. R. Thomas and S. Salhi. An investigation into the relationship of heuristic performance with network–resource characteristics. *Journal of the Operational Research Society*, 48:34–43, 1997.

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

[70] Y. C. Toklu. Application of genetic algorithms to construction scheduling with or without resource constraints. *Canadian Journal of Civil Engineering*, 29:421–429, 2002.

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

[72] P. Tormos and A. Lova. An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal of Production Research*, 41(5):1071–1086, 2003.

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

[74] G. Ulusoy and L. Özdamar. Heuristic performance and network/resource characteristics in resource-constrained project scheduling. *Journal of the Operational Research Society*, 40:1145–1152, 1989.

[75] V. Valls, F. Ballestin, and M. S. Quintanilla. A hybrid genetic algorithm for the RCPSP. Technical report, Department of Statistics and Operations Research, University of Valencia, 2003.

[76] V. Valls, F. Ballestin, and M. S. Quintanilla. Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, 2004. Forthcoming.

[77] V. Valls, F. Ballestin, and M. S. Quintanilla. A population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 2004. Forthcoming.

[78] V. Valls, M. A. Pérez, and M. S. Quintanilla. Heuristic performance in large resource-constrained projects. Technical Report 92-2, Department of Statistics and Operations Research, University of Valencia, 1992.

[79] V. Valls, M. S. Quintanilla, and F. Ballestin. Resource-constrained project scheduling: A critical activity reordering heuristic. *European Journal of Operational Research*, 2004. Forthcoming.

[80] G. E. Whitehouse and J. R. Brown. GENRES: An extension of Brooks algorithm for project scheduling with resource constraints. *Computers & Industrial Engineering*, 3:261–268, 1979.