Hybrid Local Search Techniques for the Resource-Constrained Project Scheduling Problem

Igor Pesek[1], Andrea Schaerf[2], and Janez Žerovnik[1,3]

[1] IMFM, Jadranska 19, 1000 Ljubljana, Slovenia
[2] DIEGM, University of Udine, via delle Scienze 208, 33100 Udine, Italy
[3] FS, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia

Abstract. This paper proposes a local search algorithm that makes use of a complex neighborhood relation based on a hybridization with a constructive heuristics for the classical resource-constrained project scheduling problem (RCPSP).

We perform an experimental analysis to tune the parameters of our algorithm and to compare it with a tabu search based on a combination of neighborhood relations borrowed from the literature. Finally, we show that our algorithm is also competitive with the ones reported in the literature.

1 Introduction

The Resource-Constrained Project Scheduling Problem (RCPSP) is a classical scheduling problem that has received a lot of attention from the community on metaheuristics (see, e.g., [1]). In addition, a large and widely-accepted dataset is available publicly for RCPSP [14], so that algorithms can be compared on a common ground.

We propose a local search algorithm that makes use of a complex neighborhood relation based on a hybridization with a constructive heuristics. More precisely, at each iteration, we create a list of activities that are first removed altogether from the schedule, and then reinserted one by one (in their order) in the possible best position in the schedule. This neighborhood relation takes inspiration from the good results that were achieved with a similar approach for the TSP problem [3].

We perform an experimental analysis to understand the behavior of the algorithm and to tune its parameters. In addition, we compare, in a statistically-principled way, our algorithm with our implementation of a tabu search algorithm based on a combination of neighborhood relations borrowed from the literature. Finally we place our best results within the ones reported in the literature.

The outcome is that our algorithm performs favourably with the tabu search, which was reported to be among the most competitive methods. In addition, although for a definitive answer a more complete analysis is needed, we can claim that our solver is competitive with those in the literature.

T. Bartz-Beielstein et al. (Eds.): HM 2007, LNCS 4771, pp. 57–68, 2007.

The paper is organized as follows. In Section 2 we describe the RCPSP by providing the mathematical formulation, describing the public broadly-accepted dataset, and discussing related work on the problem. In Section 3 we illustrate our main local search solver and the TS-based "competitor", both in terms of search space, neighborhood relations, and specific metaheuristics. In Section 4 we report the results of our experimental analysis and we make the comparisons. Finally, in Section 5 we draw some conclusions and we discuss future work.

2 Resource-Constrained Project Scheduling Problem

There are two versions of the RCPSP, namely the single-mode and multi-mode one. In the multi-mode version of the problem, an activity has to be performed in one of the prescribed ways (modes) using specified amount of the resources, whereas in the single-mode version there is exactly one execution mode for each activity. In this paper we focus on the single-mode version of the problem.

2.1 Problem Formulation

The resource-constrained project scheduling problem (RCPSP) can be stated as follows. Given are n activities $a_1, \ldots, a_n$ and r renewable resources. A constant amount R_k of units of resource k is available at any time. Activity a_i must be processed for p_i time units; preemption is not allowed. During this time period a constant amount of r_{ik} units of resource k is occupied. All the values R_k, p_i, and r_{ik} are non-negative integers.

Furthermore, there are precedence relations defined between activities. That is, we are given a directed graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ such that if $(i, j) \in E$ then activity j cannot start before the end of activity i.

The objective is to determine starting times s_i for the activities a_i, $i = 1, \ldots, n$ in such a way that:

- at each time t the total resource demand is less than or equal to the resource availability for each resource,
- the precedence constraints are fulfilled and,
- the makespan $C_{max} = \max_{i=1}^{n} c_i$, where $c_i = s_i + p_i$, is minimized.

As a generalization of the job-shop scheduling problem the RCPSP is NP-hard in the strong sense.

Let us illustrate the above definitions with the example on Fig. 1 which is taken from [22]. There are eleven activities and one single resource. The numbers associated with each node give the length of the activity and the units of resource it uses.

Clearly, a schedule is represented by the assignment of starting times of all activities. However, it can also be represented indirectly by an *activity list*, which is a permutation of all the activities.

An activity list is called feasible if it satisfies all precedence constraints, i.e. each activity has all its network predecessors as predecessors in the list.

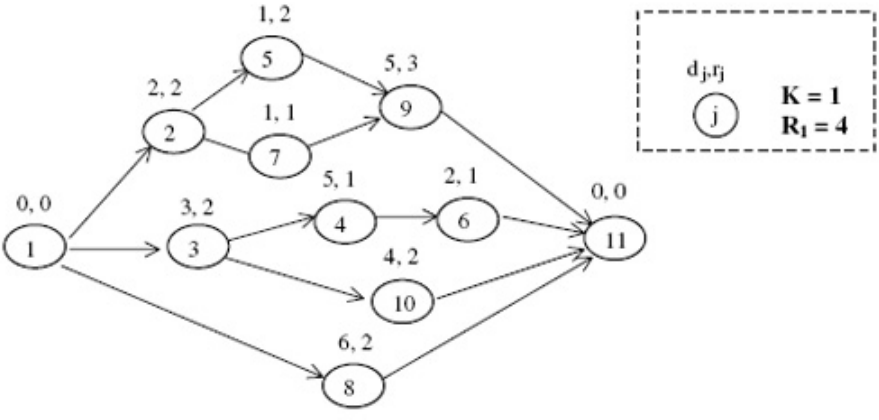

Fig. 1. Activity network

It is easy to see that given a feasible activity list there is a unique way to build a feasible schedule of minimal makespan, i.e., the so-called left-justified one. This is simply obtained by placing the activities in the given order one by one at the earliest possible starting time according to precedences and resource constraints. It can also be shown that there always exists an activity list that generates an optimal schedule [12].

Given the network depicted in Fig. 1, some of the (feasible) activity lists are: $\lambda = (1, 2, 3, 5, 7, 9, 8, 4, 6, 10, 11)$ $\alpha = (1, 2, 3, 7, 10, 4, 8, 5, 9, 6, 11)$ and $\beta = (1, 2, 3, 7, 4, 10, 8, 5, 6, 9, 11)$.

For example, the makespan of β is 18, as the activity list β gives rise to a schedule depicted on Fig. 2.

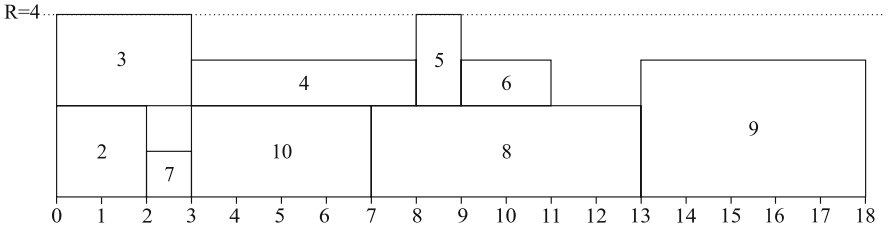

Fig. 2. Feasible schedule for activity list β

2.2 Datasets

PSPLIB [14] is a large dataset for RCPSP that is composed of 480 instances for each $n = 30, 60, 90$, and 120, for the single-mode version of the problem. Virtually all papers dealing with RCPSP have considered this dataset as the ground for the experimental analysis.

We investigated the performance of our algorithm on various PSPLIB instances. In this work, we give preliminary results on the instances with 60 and

90 activities. We have selected ten instances for $n = 60$ and ten for $n = 90$. We selected the instances randomly, but only among the "hard" ones; that is, instances for which the known best solution is not equal to the lower bound reported.

2.3 Related Work

In recent years, many papers have been published on RCPSP, as reported for example in the surveys by Özdamar and Ulusoy [17] and by Brucker *et al* [4]. A great progress have been made in the solving procedures which take into account two different approaches: optimal and heuristic.

The optimal approach includes methods such as 0-1 linear programming (Mingozzi *et al* [15]) and implicit enumeration with branch and bound (Brucker *et al* [5]).

Nevertheless, the NP-hard nature of the problem makes it difficult to solve large-size projects [2], in such a way that, in practice, the use of heuristics is necessary. Therefore, besides exact algorithms many authors have developed heuristics for the RCPSP as the only feasible method of handling practical RCPSP instances (for a survey see the work by Kolisch and Hartmann [10,13]).

Among other, the one that most resembles our approach is the insertion technique [6], which is also based on insertion of activities in a schedule, but it is not based on local search and it makes use of a different representation of the search space. Furthermore this method inserts only one activity in one iteration, whereas ours inserts multiple ones in the same iteration.

3 Local Search for RCPSP

In order to apply the local search paradigm we need to define the search space, the cost function, the selection rule for the initial solution (Section 3.1), and the neighborhood structure (Section 3.2). Finally, we present our metaheuristic procedures (Section 3.3).

3.1 Search Space, Cost Function, and Initial Solution Construction

As already discussed in Section 2.1, an activity list represents a schedule. Therefore, the search space is simply given by the set of possible activity lists, i.e., permutations of the set $\{1, \ldots, n\}$. Among all activity lists only the feasible ones, i.e. those satisfying all precedence constraints, are considered as possible search states.

The cost function is simply given by the makespan of the schedule. In fact, this is the sole objective, and we do not consider schedules that violate some constraints.

The initial solution is constructed from scratch in a stepwise extension of the partial schedule (this approach is thoroughly investigated also in [12]). In each step we randomly choose one activity from the set of activities that have all predecessors already scheduled and put it at the end of the partial activity list.

3.2 Neighborhood Relations

We first introduce the new neighborhood relation that we call RaR (Remove and Reinsert). Then we describe the literature-based one that we use for comparison.

The main idea of RaR is to remove a fixed number m of activities from the schedule and insert them back into the schedule, where m is a parameter of the method.

A move M is thus identified by a sequence of m activities $M = \{a_{M_1}, a_{M_2}, \ldots, a_{M_m}\}$, and it is executed in a state S leading to state S' in the following way:

1. $S' = S - M$ *// remove all activities in M from S*
2. for each $i = 1, \ldots, m$, add activity a_{M_i} to S' in the following way:
   2.1. search for the position j in which the makespan of S' plus a_{M_i} is minimized (breaking ties randomly)
   2.2 insert activity a_{M_i} in position j in S'

For example, let $m = 2$ and consider the network of Fig. 1 and the state β corresponding to the schedule of Fig. 2, and the RaR move $M = \{5, 9\}$. Starting from β, the activity list obtained in Step 1 is $\beta^{(0)} = 1, 2, 3, 7, 4, 10, 8, 6, 11$. In Step 2, we first reinsert the activity 5 in order to construct $\beta^{(1)} = 1, 2, 3, 5, 7, 4, 10, 8, 6, 11$. Activity 5 can only be reinserted after its predecessor 2, and before its successor 11 (not 9, because it was also deleted). Among the feasible positions, the one chosen gives the minimal makespan. Second, after reinsertion of 9 we get $\beta^{(2)} = 1, 2, 3, 5, 7, 9, 4, 10, 8, 6, 11$ with makespan 15 (see Fig. 3). □

The other neighborhood relation that we use, which we call MS (for MultiShift), is actually the set-union of two neighborhoods. For the first one, the main idea is to move an activity i behind some other activity j, which is not in precedence relation to i. In the literature this move is called *shift move* and it has been introduced in [1]. More formally, a move M is identified by a pair $\{i, j\}$ (with $i, j \in \{1, \ldots, n\}$), such that a_i is placed after a_j in the schedule L. The move can be illustrated by

$$L = \ldots i \ldots u \ldots j \ldots \Rightarrow \; L' \; = \; \ldots j \; i \; u \ldots$$

where u is an activity that must be scheduled after i. The shift of i thus is accompanied by a set of other shifts so as to prevent to break precedence constraints.

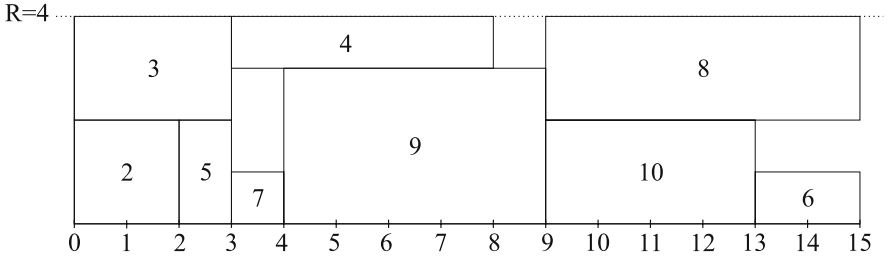

Fig. 3. Schedule for activity list $\beta^{(2)}$

The second component of our neighborhood works in the reversed way. Instead of moving activity i after j, we move activity j before i. Again, if necessary, we also make accompanying set of shifts in order to keep all precedence constraints satisfied.

These neighborhoods together are designed in such a way, that they make room for some activities to be scheduled earlier and therefore possibly shorten the makespan of the schedule.

3.3 Local Search Metaheuristics

The neighborhood RaR, for most of the values of m that we consider, is a large neighborhood, and its exhaustive exploration turned out to be rather impractical. Therefore for this neighborhood, we resort to a simple hill-climbing strategy, based on a randomized non-ascending selection. In details, at each iteration, we draw a random move and compute its cost. If the move is improving or *sideways* it is executed, otherwise the state is unchanged. The procedure stops after a fixed number of iterations. We call this algorithm HC(RaR).

It is worth noticing that HC(RaR) can also be seen as a form of min-conflict hill-climbing (MCHC) [16], since activities are selected at random, but then are reinserted in the optimal way. In addition, like MCHC, it accepts sideways moved (for diversification).

On the other hand, for the smaller neighborhood MS, more "aggressive" metaheuristics seem to be more effective (based on preliminary experiments). Therefore, we make use of a Tabu Search [9], with a dynamic-size tabu list to implement a short term prohibition mechanism (called Robust Tabu Search in [11]). In addition, we use the standard aspiration criterion and we search for the next state by exploring the full neighborhood at each iteration. We call this algorithm TS(MS).

4 Experimental Analysis

In order to determine how successful our new approach is, we first explain our experimental settings (Section 4.1) and then experimental parameter tuning for both methods (Section 4.2). Next we compare both methods with best parameters for each method (Section 4.3) and finally we present comparison with methods and results reported in the literature (Section 4.4).

4.1 Experimental Setting

Experiments were performed on an Intel Pentium 4 (3.4 GHz) processor running Linux Suse v. 10. The algorithms have been coded in C++ exploiting the framework EasyLocal++ [7], and the executables were obtained using the GNU C/C++ compiler (v. 4.0.1). The statistical tests are performed using the software environment for statistical computing R [20].

For HC(RaR) the stop criterion in our experiments is based on the number of iterations, i.e. the number of feasible schedules generated. Given that the

running time of a single iteration depends on m, in order to make a comparison fair, we normalize the number of iterations so that the total running times are approximately the same.

A typical number of iterations used in the literature (see [10,13]) is 5000. We decided to grant 5000 iterations to the largest value of m and to augment the others accordingly.

In particular, for the instances with 60 activities, we have tested HC(RaR) with $m = 2, \ldots, 20$, and the running times for $m = 20$ of 5000 iteration turned out to be about 150s. All settings for both HC(RaR) and TS(MS) are made in order to run for 150s per trial. For HC(RaR), this means that the number of iterations allowed was larger for smaller m, reaching the highest value 23333 at $m = 2$. More precisely, we have used a heuristic formula for the number of iterations such that the runs needed approximately the same time. The formula is $max\_iter = 10500/(m+1)(1+\delta)$, where $\delta = (20-m)/36$.

4.2 Experiments for Parameter Tuning for HC(RaR) and TS(MS)

Our first set of experiments aims at identifying the best value of m for HC(RaR). Table 1 shows the results for the selected instances of size $n = 60$, for 30 trials for each instance. The table reports the average deviation w.r.t. the lower bound (obtained through the critical-path method [1]), and its standard deviation.

The outcome of the experiments show that the best results are obtained for $m = 7$ (in bold). Note that for values around 7 the results are very close and the standard deviation is relatively small for all of them.

Table 1. Results of HC(RaR) for $m = 2, \ldots, 20$ with time limit 150s for n=60

m	avg. diff. (%)	std. dev.
2	11.668	1.0497
3	11.364	0.9448
4	11.340	1.0158
5	11.252	0.9548
6	11.130	0.9713
7	**11.107**	0.9636
8	11.122	0.8810
9	11.242	0.8473
10	11.294	0.9244
11	11.388	0.9131
12	11.583	0.8689
13	11.742	0.9826
14	11.867	0.9330
15	11.918	0.8784
16	12.152	0.9073
17	12.345	0.8866
18	12.598	0.9920
19	12.797	0.8610
20	12.880	0.9419

Table 2. Results of TS(MS) for different tabu lengths for $n = 60$

Tabu length	avg. diff. (%)	std. dev.
5 10	14.526	1.3679
5 15	14.473	1.3468
5 20	14.470	1.4237
10 15	**14.339**	1.3703
10 20	14.587	1.4881
10 25	14.364	1.4140
10 30	14.453	1.4528
15 20	14.519	1.5012
15 30	14.370	1.3830
20 30	14.485	1.4563
20 40	14.435	1.5786

Table 3. Results of HC(RaR) for $m = 2, \ldots, 20$ with time limit 210s for $n = 90$

m	avg. diff. (%)	std. dev.
2	13.398	1.4224
3	12.842	1.4536
4	12.661	1.3023
5	12.662	1.4095
6	12.721	1.3402
7	**12.654**	1.2768
8	12.782	1.3863
9	13.143	1.2843
10	13.283	1.3608
11	13.333	1.3961
12	13.516	1.3693
13	13.753	1.5144
14	13.922	1.3413
15	14.061	1.3176
16	14.306	1.3292
17	14.215	1.2872
18	14.478	1.3614
19	14.654	1.2652
20	14.956	1.3357

Similarly, Table 2 shows that the best configuration for TS(MS) is with tabu length [10,15] for $n = 60$, although results for other tabu lengths are very similar.

The experiments for $n = 90$ presented in Table 3 show that the best results are obtained when $m = 7$, although results for other m that are near 7 are also very competitive.

The choice of a suitable value for m is clearly crucial for the behavior of any local search based on RaR. In our preliminary work [19], we conjectured that the best value of m is linearly dependent on n, and in particular it is about $m = n/10$. In the experiments reported here, we provide a more statistically-principled comparison,

and we show that the conjecture was not very precise, and, surprisingly, the best choice is the fixed value $m = 7$ for both datasets.

To test this hypothesis we did a smaller experiment on the dataset with $n = 120$, which again seems to confirm that the best choice for m is 7.

4.3 Comparison Between HC(RaR) and TS(MS)

Table 4 shows a comparison between our two methods with problemset $n = 60$. We compared only the best performing choice of parameters for both methods, more precisely for HC(RaR) we used $m = 7$ and for TS(MS) we used tabu length [10,15]. For each instance it shows the results for 30 trials in terms of: the best value obtained, the average percentage difference w.r.t. the lower bound, and the standard deviation. The last column shows the p-value for the Wilcoxon test [23], which represents the confidence in the statistical difference of the two sequences of the results, in favour of HC(RaR).

The last line shows the comparison for all instances together, using the paired Wilcoxon test.

Table 4. Comparison between HC(RaR) and TS(MS), $n = 60$

			HC(RaR)			TS(MS)			
Instance	LB	UB	best	avg. dev. (%)	std. dev.	best	avg. dev. (%)	std. dev.	p
j605_3	75	80	81	9.16	0.9568	82	12.76	1.8740	2.848e-08
j605_10	79	81	81	5.11	1.0482	83	8.65	1.3437	3.727e-09
j609_4	79	87	88	15.27	1.3148	88	17.38	1.5902	4.176e-05
j609_5	77	85	88	14.29	0.8563	89	19.87	1.4410	2.928e-11
j6013_7	80	87	88	12.25	0.9451	91	16.33	1.1527	1.156e-10
j6025_5	86	98	98	16.01	0.8034	100	18.8	1.2671	1.095e-09
j6025_7	83	90	91	11.24	1.1925	93	14.9	1.3287	1.531e-09
j6029_5	102	110	112	11.86	1.2476	112	14.71	1.8073	1.274e-07
j6029_8	96	103	104	9.48	0.7895	106	12.15	1.2472	3.401e-10
j6037_8	88	93	93	6.4	0.4818	94	7.84	0.6506	3.916e-09
ALL	–	–	–	11.107	0.9636	–	14.339	1.3702	4.193e-06

Table 4 shows that HC(RaR) is clearly superior to TS(MS) in means of average deviation, namely mean of the average deviations on all the instances is 11.10% against 14.33%. Also mean of the standard deviations for HC(RaR) is significantly better than TS(MS), and the p values are very close to 0, showing a confidence in the statistical difference of almost 100%.

Table 5 presents the same data for $n = 90$, and it shows similar results.

4.4 Comparison with Previous Results

In [13] experimental results of almost all the state-of-the-art algorithms are reported. Comparison is made with number of iterations limited to $i = 1000$, 5000 and 50000. Since we have high running times, we will compare our method with results reported for 50000 iterations.

Table 5. Comparison between HC(RaR) and TS(MS) for $n = 90$

			HC(RaR)			TS(MS)			
Instance	LB	UB	best	avg. dev. (%)	std. dev.	best	avg. dev. (%)	std. dev.	p
j905_3	82	87	90	12.68	1.2	91	16.75	1.6519	3.493e-09
j905_5	107	111	114	8.72	1.1926	118	13.43	2.714	6.515e-11
j909_3	97	102	106	11.96	1.2543	111	17.8	1.9482	3.343e-11
j909_7	102	109	113	13.27	1.2579	117	19.02	1.7814	3.673e-11
j9013_3	104	108	113	10.29	0.8226	116	14.2	1.5638	3.076e-11
j9021_6	95	106	108	17.79	1.8321	111	20.56	1.9619	8.481e-06
j9029_5	114	121	125	11.78	1.0858	127	15.5	1.66	2.808e-10
j9037_2	103	115	116	15.08	1.384	119	18.71	1.9137	1.394e-09
j9037_7	112	123	123	11.85	1.34	126	15.98	2.0712	1.816e-10
j9045_7	127	136	140	13.12	1.3984	145	17.11	1.7114	5.869e-11
ALL	–	–	–	12.654	1.2767	–	16.906	1.8977	8.487e-07

Table 6. Comparison between the best methods for solving RCPSP, $n = 60$

Method	Author	avg. dev.(%)
Scatter search	Debels et al	10.71
GA-hybrid	Valls et al.	10.73
GA, TS-path relinking	Kochetov and Stolyar	10.74
GA-FBI	Valls et al.	10.74
GA-forw.-backw., FBI	Alcaraz et al.	10.84
HC(RaR)	**Pesek, Schaerf, Žerovnik**	**11.10**
GA-self-adapting	Hartmann	11.21
GA-activity list	Hartmann	11.23
Sampling-LFT, FBI	Tormos and Lova	11.36
TS-activity list	Nonobe and Ibaraki	11.58

We should however remark that, at this stage, a fair comparison is not possible for two reasons: First, in the most relevant literature, results are reported only as the average on all 480 instances, and we haven't completed such a massive computation yet. Secondly, the computing power granted to the solvers is always expressed only in terms of visited schedules, and it is not clear how this metric applies to our solver. The most straightforward application is by granting this number of iteration to our solvers, but we have to admit that this would be too biased on our side, as our iteration is computationally much more expensive than the equivalent step in the literature (in some cases even more than 2 orders of magnitude).

Having in mind these limits of the comparison, we summarize the current results in Table 6, which shows the best ten algorithms reported (rest of the algorithms reported is in [13]). In first column shows the method used, then the author of the method and in the last column the average deviation from the critical path lower bound.

We can assert that our results are well in-line with the best solvers, also having in mind that it regards only a set of "hard" instances (which are the minority),

whereas the others are averaged on all of them, but, on the other hand, we have much longer runs.

5 Conclusions and Future Work

We have proposed a local search solver for RCPSP based on a large neighborhood coming from a hybridization with a constructive heuristic. In our experimental analysis we have identified the best parameter setting for the solver, and compared it with our implementation, using the same technologies, of a state-of-the-art solver, namely a tabu search based on a classical neighborhood relation. The results, although preliminary, are quite encouraging.

Obviously, the first future work will be to perform a comprehensive analysis on all PSPLIB instances, so as to place the results within the relevant literature, by identifying a common ground of comparison that takes into account both results and running times.

We also plan to find new ways to improve our solver on the various levels of intervention: metaheuristic strategy, neighborhood exploration, and implementation.

Finally, in this paper we focused on the single-mode problem, whereas the multi mode problem is actually more natural and describes real life problems more accurately. Therefore, we plan to work on this other version, because we believe that our approach can be adapted to it and also provide good results.

References

1. Baar, T., Brucker, P., Knust, S.: Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem. In: Voss, S., Martello, S., Osman, I., Roucairol, C. (eds.) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp. 1–18. Kluwer Academic Publishers, Dordrecht (1998)
2. Blazewicz, J., Lenstra, J., Kan, A.R.: Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5, 11–24 (1983)
3. Brest, J., Žerovnik, J.: An approximation algorithm for the asymmetric traveling salesman problem. Ricerca Operativa 28, 59–67 (1999)
4. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(1), 3–41 (1999)
5. Brucker, P., Knust, S., Schoo, A., Thiele, O.: A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107(2), 272–288 (1998)
6. Christian, A., Michelon, P., Reusser, S.: Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research 149, 249–267 (2003)
7. Di Gaspero, L., Schaerf, A.: EasyLocal++: An object-oriented framework for flexible design of local search algorithms. Software—Practice and Experience 33(8), 733–765 (2003)

8. Gendreau, M., Hertz, A., Laporte, G., Stan, M.: A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research 46(3), 330–335 (1998)
9. Glover, F., Laguna, M.: Tabu search. Kluwer Academic Publishers, Dordrecht (1997)
10. Hartmann, S., Kolisch, R.: Experimental evaulation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127(2), 394–407 (2000)
11. Hoos, H.H., Stützle, T.: Stochastic Local Search Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA (USA) (2005)
12. Kolisch, R., Hartmann, S.: Heuristic algorithms for solving the resource-constrained project scheduling problem - classification and computational analysis. In: Weglarz, J. (ed.) Handbook on recent advances in project scheduling, pp. 147–178. Kluwer Academic Publishers, Dordrecht (1999)
13. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 174(1), 23–37 (2006)
14. Kolisch, R., Sprecher, A.: PSPLIB – a project scheduling library. European Journal of Operational Research 96(1), 205–216 (1997) Data available from `http://129.187.106.231/psplib/`
15. Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L.: An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Management Science 44(5), 714–729 (1998)
16. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence 58, 161–205 (1992)
17. Özdamar, L., Ulusoy, G.: A survey on the resource-constrained project scheduling problem. IIE transactions 27(5), 574–586 (1995)
18. Palpant, M., Artigues, C., Michelon, P.: Lssper: Solving the resource-constrained project scheduling problem with large neighbourhood search. Annals of Operations Research 131(1-4), 237–257 (2004)
19. Pesek, I., Žerovnik, J.: Best insertion algorithm for resource-constrained project scheduling problem (preprint, 2006) available on `http://arxiv.org/abs/0705.2137v1`
20. R Development Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria (2005) ISBN 3-900051-07-0.
21. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)
22. Valls, V., Quintanilla, S., Ballestín, F.: Resource-constrained project scheduling: A critical activity reordering heuristic. European Journal of Operational Research 149, 282–301 (2003)
23. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1, 80–83 (1945)