A Bi-population Based Genetic Algorithm for the Resource-Constrained Project Scheduling Problem

Dieter Debels[1] and Mario Vanhoucke[1,2]

[1] Ghent University, Faculty of Economics and Business Administration, Hoveniersberg 24, 9000 Ghent, Belgium
[2] Vlerick Leuven Gent Management School, Operations & Technology Management, Centre, Reep 1, 9000 Ghent, Belgium
{dieter.debels, mario.vanhoucke}@ugent.be

Abstract. The resource-constrained project scheduling problem (RCPSP) is one of the most challenging problems in project scheduling. During the last couple of years many heuristic procedures have been developed for this problem, but still these procedures often fail in finding near-optimal solutions for more challenging problem instances. In this paper, we present a new genetic algorithm (GA) that, in contrast of a conventional GA, makes use of two separate populations. This bi-population genetic algorithm (BPGA) operates on both a population of left-justified schedules and a population of right-justified schedules in order to fully exploit the features of the iterative forward/backward scheduling technique. Comparative computational results reveal that this procedure can be considered as today's best performing RCPSP heuristic.

1 Introduction

We study the resource-constrained project scheduling problem (RCPSP), denoted as $m,1|\text{cpm}|C_{max}$ using the classification scheme of [9]. The RCPSP can be stated as follows. In a project network in AoN format G($\boldsymbol{N}$,$\boldsymbol{A}$), we have a set of nodes $\boldsymbol{N}$ and a set of pairs $\boldsymbol{A}$, representing the direct precedence relations. The set $\boldsymbol{N}$ contains n activities, numbered from 0 to $n-1$ ($|\boldsymbol{N}| = n$). Furthermore, we have a set of resources $\boldsymbol{R}$, and for each resource type $k \in \boldsymbol{R}$, there is a constant availability a_k throughout the project horizon. Each activity $i \in \boldsymbol{N}$ has a deterministic duration $d_i \in$ IN and requires $r_{ik} \in$ IN units of resource type k. We assume that $r_{ik} \leq a_k$ for $i \in \boldsymbol{N}$ and $k \in \boldsymbol{R}$. The dummy start and end activities 0 and n - 1 have zero duration and zero resource usage. A schedule S is defined by an n-vector of start times $\boldsymbol{s}(S) = (s_0, \ldots, s_{n-1})$, which implies an n-vector of finish times $\boldsymbol{f}(S)$ where $f_i = s_i + d_i, \forall\, i \in \boldsymbol{N}$. A schedule S is said to be feasible if it is nonpreemptive and if the precedence and resource constraints are satisfied. If none of the activities can be scheduled forwards (backwards) due to precedence or resource constraints, then the schedule is said to be left-justified (right-justified). The objective of the RCPSP is to find a feasible schedule that minimizes the project makespan f_{n-1}.

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3483, pp. 378–387, 2005.

2 Representation and Generation of Left- and Right-Justified Schedules

Each RCPSP meta-heuristic relies on a *schedule representation* to encode a schedule and a *schedule generation scheme* (SGS) to decode the schedule representation into a schedule S. For both the representation and generation of a schedule various approaches exist.

Table 1. Incorporation of TO condition

Justification of schedule	TO condition	implementation of TO in AL
Right-justified schedule	$s_i < s_j \Rightarrow p < q$	sort activities in increasing order of their start times
Left-justified schedule	$f_i > f_j \Rightarrow p < q$	sort activities in decreasing order of their finish times

Although five different methods are given in the literature [13], a schedule representation is simply a representation of a priority-structure between the activities. For our procedure we use the most frequently used [13] *activity list* (AL) representation where a sequence of non-dummy activities $\lambda = [\lambda_1, \ldots, \lambda_{n-2}]$ is used to determine the priority of each activity. When $\lambda_p = i$, we say that activity i is at position p in the AL. An activity i has a lower priority than all preceding activities in the sequence and a higher priority than all succeeding activities. An AL is said to be *precedence-feasible* if an activity never comes after the position of one of its successors (predecessors) in the list used for the generation of a left-justified (right-justified) schedule. In the current paper, we rely on the topological ordering (TO) condition [5, 28] for our AL representation. Our version of the TO condition and its implementation in the AL is described in Table 1, with p and q the positions of activity i and j in an AL. The table illustrates that the TO condition and the implementation depends on the justification of the schedule (left or right). Since the TO condition is based on start and finish times, and hence uses information from the corresponding schedule, we can only incorporate the TO condition after the schedule generation. In Sect. 3.3, the advantages of the TO condition will be illustrated on a project example.

Besides various schedule representations, there exist also two different types of SGSs in the literature; the serial SGS and the parallel SGS. As it is sometimes impossible to reach an optimal solution with the parallel SGS [17], we opt for the serial SGS where all activities are scheduled one-in-a-time and in the sequence of the AL. Each activity is scheduled as soon (as late) as possible within the precedence and resource constraints to construct a left-justified (right-justified) schedule. We introduce the example project depicted in Fig. 1, with a single renewable resource type with availability $a_1 = 2$. The problem is represented by an activity-on-the-node network. Corresponding to each activity we depicted the duration on top of the node and the resource demand below the node. Figure 2 represents a left-justified schedule 1, obtained by applying a serial SGS on the activity list [1, 2, 8, 5, 3, 4, 6, 7, 9]. The incorporation of the TO-condition on this schedule, leads to the activity list AL_1, depicted at the bottom of Fig. 2.

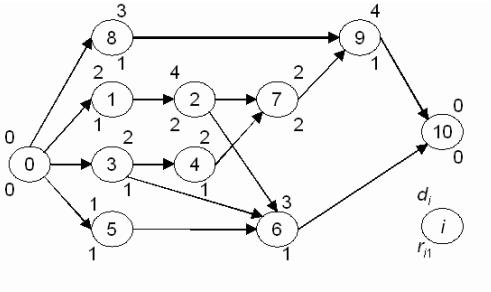

Fig. 1. Example project

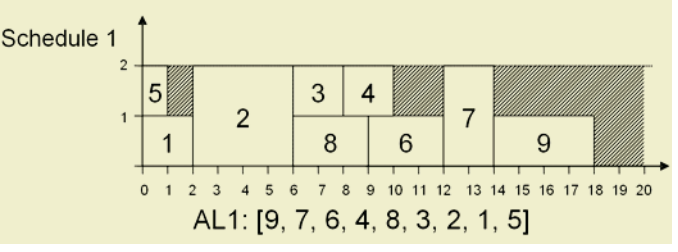

Fig. 2. A left-justified schedule and the corresponding AL after incorporation of the TO-condition

A well-known local search technique for RCPSP meta-heuristics is the iterative forward/backward scheduling technique. This technique is introduced by Li and Willis [20] and successfully implemented in various meta-heuristic procedures [1, 5, 11, 24, 25, 26, 27, 29]. The technique is based on the serial SGS and uses schedule information to determine the AL. Starting from a left-justified schedule, the procedure creates an AL by sorting the activities in decreasing order of their finish times (i.e. the TO condition for left-justified schedules of Table 1). Then, the serial SGS is used to build a right-justified schedule. In a following iteration, the activities are sorted in increasing order of the start times in the right-justified schedule (i.e. the TO condition for right-justified schedules of Table 1) and the serial SGS is used to generate a left-justified schedule. In doing so, only improvements can occur for each iteration. The procedure stops when no further improvements can be obtained. Assume that schedule 1 of Fig. 2 is our start left-justified schedule with an activity list AL_1 in which the activities are sorted in decreasing order of the finish times. The iterative forward/backward procedure uses this list to construct a right-justified schedule with corresponding activity list AL_2. In this list, the activities have been sorted in increasing order of their start times. This iteration (see schedule 2 of Fig. 3) leads to a makespan improvement of 2 time units. In a next iteration, the procedure constructs the left-justified schedule 3 with a corresponding activity list AL_3. The procedure could continue this process by using the activity list AL_3 to construct a right-justified schedule, but it is easy to see that no further makespan improvement can be achieved.

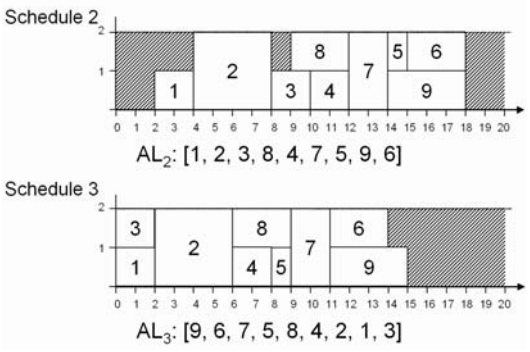

Fig. 3. The iterative forward/backward scheduling technique

3 The Bi-population Genetic Algorithm

The evolution of living beings motivated Holland [10] to solve complex optimization problems by using algorithms that simulate biological evolution. This approach gave rise to the technique known as a genetic algorithm (GA). In a GA, processes loosely based on natural selection, crossover and mutation are repeatedly applied to one population that represents potential solutions. In contrast to a regular GA, we use the bi-population genetic algorithm (BPGA) that makes use of two different populations: a population ***LJS*** that only contains left-justified schedules and a population ***RJS*** that only contains right-justified schedules. Both populations have the same population size. The procedure starts with the generation of an initial ***LJS***, followed by an iterative process that continues until the stop criterion is satisfied. The iterative process consecutively adapts the population elements of ***RJS*** and ***LJS***. ***RJS*** (***LJS***) is updated by feeding it with combinations of population elements taken from ***LJS*** (***RJS***) that are scheduled backwards (forwards) with the serial SGS. The remainder of this section reveals some further algorithmic details about the construction of the initial population, parent-selection, crossover-operator, diversification and selection mechanism of the BPGA.

3.1 Construction of the Initial Population

We start the genetic algorithm by building an initial population ***LJS*** of left-justified schedules. Each population element is created by randomly generating an AL, constructing the corresponding left-justified schedule and finally incorporating the TO condition of Table 1.

3.2 Parent Selection

For each population element a of ***LJS*** (***RJS***) we create a set of nrc right-justified (left-justified) children that are candidates to enter ***RJS*** (***LJS***). To create a child out of a, we select another parent b from ***LJS*** (***RJS***) by using the 2-tournament selection procedure. In this selection procedure two population-

elements are chosen randomly, and the element with the lowest makespan is selected. Afterwards, we determine randomly whether a or b represents the father S_f. The other parent represents the mother S_m.

3.3 Generation of a Child

A right-justified (left-justified) child is created from two parents from ***LJS*** (***RJS***) in two phases. In both phases, the advantages of our TO-condition implementation are fully exploited.

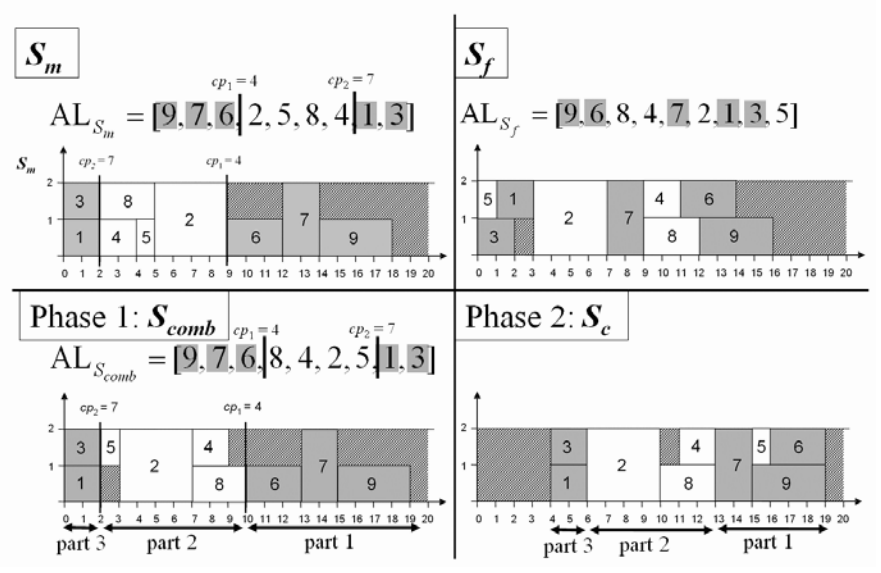

Fig. 4. Crossover operator

Phase 1: The Construction of a Combined Activity List $\mathbf{AL}_{S_{comb}}$. Based on two parents from ***LJS*** (***RJS***), we use a 2-point crossover operator to generate the combined activity list $\mathrm{AL}_{S_{comb}}$ which is used in phase 2 to construct a right-justified (left-justified) child S_c. To that purpose we select two crossover points cp_1 and cp_2 as follows. First, we randomly generate a crossover interval Δcp from $[1/4.f_{n-1}(S_m),\ 3/4.f_{n-1}(S_m)]$, where $f_{n-1}(S_m)$ denotes the makespan of the mother schedule. Then, we randomly generate cp_1 from $[0,\ f_{n-1}(S_m) - \Delta cp]$ and set $cp_2 = cp_1 + \Delta cp$. The TO condition allows the construction of $\mathrm{AL}_{S_{comb}}$ and the combined schedule S_{comb} by simply copying parts from the AL of the mother and the father. More precisely, we copy all activity positions from the mother from the intervals $[1,\ cp_1[$ and $]cp_2,\ n]$. The remaining activities from the interval $[cp_1,\ cp_2]$ are copied in $\mathrm{AL}_{S_{comb}}$ according to the AL ranking of the father. In Fig. 4, we have set cp_1 and cp_2 to 4 and 7, and AL_{S_f} and AL_{S_m} represent the activity lists of the parents in TO-format. The dark-colored activities from the interval $[1,\ 4[$ (i.e. 9, 7 and 6) and $]7,\ 9]$ (i.e. 1 and 3) are copied from AL_{S_m} to

$\mathrm{AL}_{S_{comb}}$ and correspond to part 1 and part 3 in S_{comb}. The remaining activities (i.e. 2, 5, 8 and 4), displayed in white, are copied into $\mathrm{AL}_{S_{comb}}$ according to the sequence of AL_{S_f}, i.e. 8, 4, 2 and 5 and represent part 2 in S_{comb}.

Phase 2: The Construction of a Right-Justified (Left-Justified) Child. The combined schedule S_{comb} is often neither a left- or a right-justified schedule. Therefore, we transform this combined schedule into a left-justified (right-justified) schedule, when the parents belong to ***RJS*** (***LJS***), using the SGS. This can be done by running the iterative forward/backward scheduling procedure on the combined schedule S_{comb}. In doing so, only improvements can occur for each part of S_{comb}. In our example of Fig. 4, we transform S_{comb} in a right-justified schedule S_c, resulting in a makespan improvement of 3 time-units for part 1 and 1 time-unit for part 2.

3.4 Diversification

Diversification is necessary in every genetic algorithm to avoid the creation of a homogeneous population. We use a reactive method that only applies diversification to a child when it comes from two not mutually diverse parents. To define whether the parents are sufficiently diverse, we need a threshold τ and a distance measure. Our distance measure simply takes the sum of absolute deviations between the positions in the activity list of the father and the activity list of the mother for each activity and divides the obtained value by the number of non-dummy activities as defined in (1). Diversification is desirable if the distance exceeds the threshold τ and is exerted on $\mathrm{AL}_{S_{comb}}$ by randomly swapping the positions of two activities for nrd times. In our example we calculate a distance of 2.0 between S_f and S_m as the sum of position differences for all activities is 18 and the number of non-dummy activities is 9.

$$\mathrm{dist}(S_f, S_m) = \frac{1}{n-2} \sum_{i=1}^{n-2} |\text{position of } i \text{ in } \mathrm{AL}_{S_f} - \text{position of } i \text{ in } \mathrm{AL}_{S_m}| \quad (1)$$

3.5 Selection Mechanism

The selection mechanism determines the way in which the new generation replaces the old generation. In order to make the genetic algorithm successful, the 'survival of the fittest'-principle should be embedded. Good children should have a higher chance to enter the new generation than inferior ones in order to improve the quality of the population. The population ***RJS*** (***LJS***) is fed by children generated from ***LJS*** (***RJS***). In the following we will explain how we update ***RJS***. The way in which we update ***LJS*** is analogue. As mentioned previously, we generate nrc children for each element of ***LJS***. From the set that is created by the x^{th} population-element, we select the child with the lowest makespan. This child will replace the x^{th} element of ***RJS***, even if this leads to a deterioration of the makespan. But, in order to prevent that we loose high-quality schedules, we do not automatically replace the x^{th} element if this corresponds with the best-found schedule so far. In this case, we only perform replacement when the child represents a new best-found solution.

4 Comparative Computational Results

We have coded the procedure in Visual C++ 6.0 and performed computational tests on an Acer Travelmate 634LC with a Pentium IV 1.8 GHz processor using the well-known PSPLIB dataset [15] which we use to compare our procedure with other existing procedures from the literature. This dataset contains the subdatasets J30, J60 and J120 that contain problem-instances of 30, 60 and 120 activities respectively. We predefined the settings of the parameters as follows. The number of children *nrc* is fixed at 2, the diversification-parameter *nrd* is fixed at the number of non-dummy activities divided by 10 and the threshold τ for applying diversification is set equal to 2. The population size has been fine-tuned to an optimal value.

Table 2. Comparative results for J30, J60 and J120

J30	max #schedules		
	1,000	5,000	50,000
[11]	0.10	0.04	0.00
[5]	0.27	0.11	0.01
BPGA	**0.17**	**0.06**	**0.02**
[27]	0.27	0.08	0.02
[1]	0.33	0.12	-
[30]	0.34	0.20	0.02
[26]	0.25	0.13	0.05
[22]	0.46	0.16	0.05
[24]	0.30	0.16	0.07
[7]	0.38	0.22	0.08
[6]	0.54	0.25	0.08
[25]	0.30	0.17	0.09
[30]	0.46	0.28	0.11
[3]	0.38	0.23	-
[4]	0.74	0.33	0.16
[23]	0.65	0.44	-
[2]	0.86	0.44	-
[12]	0.74	0.52	-
[8]	1.03	0.56	0.23
[17]	0.83	0.53	0.27
[4]	0.81	0.54	0.28
[16]	1.44	1.00	0.51
[17]	1.05	0.78	0.56
[8]	1.38	1.12	0.88
[17, 18]	1.40	1.28	-
[17]	1.40	1.29	1.13
[16]	1.77	1.48	1.22
[19]	2.08	1.59	-

J60	max #schedules		
	1,000	5,000	50,000
BPGA	**11.45**	**11.00**	**10.69**
[5]	11.73	11.10	10.71
[27]	11.58	11.10	10.73
[11]	11.71	11.17	10.74
[30]	12.21	11.27	10.74
[7]	12.21	11.70	11.21
[6]	12.68	11.89	11.23
[26]	11.88	11.62	11.36
[25]	12.14	11.82	11.47
[1]	12.57	11.88	-
[24]	12.18	11.87	11.54
[3]	12.75	11.90	-
[22]	12.97	12.18	11.58
[30]	12.73	12.35	11.94
[23]	12.94	12.58	-
[4]	13.28	12.63	11.94
[6]	14.68	13.32	12.25
[6]	13.30	12.74	12.26
[12]	13.51	13.08	-
[17, 18]	13.88	13.21	-
[4]	13.80	13.31	12.83
[17]	13.75	13.34	12.84
[17]	13.59	13.23	12.85
[2]	13.80	13.48	-
[19]	14.33	13.49	-
[17]	13.98	13.53	12.97
[16]	14.89	14.30	13.66
[16]	15.94	15.17	14.22

J120	max #schedules		
	1,000	5,000	50,000
BPGA	**34.29**	**32.34**	**30.75**
[27]	34.07	32.54	31.24
[5]	35.22	33.10	31.57
[30]	35.39	33.24	31.58
[11]	34.74	33.36	32.06
[30]	35.18	34.02	32.81
[7]	37.19	35.39	33.21
[26]	35.01	34.41	33.71
[21]	-	35.43	-
[6]	39.37	38.74	34.03
[25]	36.24	35.56	34.77
[24]	36.49	35.81	35.01
[1]	39.36	36.57	-
[22]	40.86	37.88	35.85
[4]	39.97	38.41	38.44
[30]	38.21	37.47	38.46
[3]	42.81	37.08	-
[6]	39.93	38.48	38.51
[23]	39.85	38.70	-
[17]	39.60	38.75	37.74
[17, 18]	39.65	38.77	-
[6]	45.82	42.25	38.83
[17]	41.27	40.38	39.34
[12]	41.37	40.45	-
[4]	41.36	40.46	39.41
[19]	42.81	40.69	-
[17]	42.84	41.84	40.62
[16]	44.46	43.05	41.44
[16]	49.25	47.81	45.60

To be able to compare procedures for the RCPSP, Hartmann and Kolisch [8] presented a methodology in which all procedures can be tested on the PSPLIB-datasets by using 1,000 and 5,000 generated schedules as a stop condition. In [14] Hartmann and Kolisch give an update of the results, and also report on 50,000 schedules as a schedule limit. In Table 2 we compare our algorithm with these results for the datasets J30, J60 and J120 respectively. The average deviation from the optimal solution is used as a measure of quality for J30 and the average deviation from the critical path based lower bound for J60 and J120. For each dataset the heuristics are ranked by the corresponding deviation for 50,000 schedules. The results for 5,000 and 1,000 schedules are used as a tie-breaker, if necessary. The table reveals that our procedure is capable to report consistently good results. For the datasets J60 and J120 it outperforms all other procedures.

Table 3. Optimal values of the population size

	1,000 schedules		5,000 schedules		50,000 schedules	
Dataset	*popsize*	Avg.CPU	*popsize*	Avg.CPU	*popsize*	Avg.CPU
J30	55	0.02s	112	0.04s	416	0.39s
J60	30	0.03s	71	0.09s	390	0.89s
J120	20	0.09s	60	0.22s	290	2.19s

Only for J30, [11] and [5] report a slightly better result. Furthermore, our procedure often generates better solutions for the PSPLIB problem instances than the best solutions found so far (based on PSPLIB results on December 3, 2004, see http://www.bwl.uni-kiel.de/bwlinstitute/Prod/psplib/datasm.html). As an example, we obtained 15 improvements for J120 and with a stop condition of 50,000 schedules. In general we conclude that the more challenging the problem-instances are, the better our procedure performs compared to other procedures.

The optimal values of the population size used for the results of Table 2 and the average CPU-time needed to solve one problem-instance of the dataset are depicted in Table 3. This table reveals that the population size is positively related to the schedule limit and negatively related to the number of activities. The use of a large population avoids, similar to diversification, a homogeneous population, and this becomes more important for small problem instances and high values for the stop criterion.

5 Conclusion

In this paper we presented a genetic algorithm for the resource-constrained project scheduling problem (RCPSP) that operates on two separate populations. The first population only contains left-justified schedules and the second population only contains right-justified schedules. Our bi-population genetic algorithm (BPGA) combines schedules of the first population to create children that are candidate to enter the second population and vice versa. In this way the procedure is able to exploit the advantages of a local search technique denoted as the iterative forward/backward scheduling technique. The comparative computational results on the well-known PSPLIB dataset illustrated that the BPGA is currently the best meta-heuristic procedure for the RCPSP.

References

1. Alcaraz, J., Maroto, C.: A robust genetic algorithm for resource allocation in project scheduling, Annals of Operations Research, 102, 83-109 (2001).
2. Baar, T., Brucker, P., Knust, S.: Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem, Meta-heuristics: Advances and trends in local search paradigms for optimization, 1-8 (1998).
3. Bouleimen, K., Lecocq, H.: A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, European Journal of Operational Research, 149, 268-281 (2003).

4. Coelho, J., Tavares, L.: Comparative analysis of meta-heuricstics for the resource constrained project scheduling problem, Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal (2003).
5. Debels, D., De Reyck, B., Leus, R., Vanhoucke, M.: A scatter-search meta-heuristic for the resource-constrained project scheduling problem, European Journal of Operational Research, forthcoming.
6. Hartmann, S.: A competitive genetic algorithm for the resource-constrained project scheduling, Naval Research Logistics, 45, 733-750 (1998).
7. Hartmann, S.: A self-adapting genetic algorithm for project scheduling under resource constraints, Naval Research Logistics, 49, 433-448 (2002).
8. Hartmann, S., Kolisch, R.: Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, European Journal of Operational Research, 127, 394-407 (2000).
9. Herroelen, W., Demeulemeester, E., De Reyck, B.: A classification scheme for project scheduling. In: Weglarz, J. (Ed.), Project Scheduling - Recent Models, Algorithms and Applications, International Series in Operations Research and Management Science, Kluwer Academic Publishers, Boston, 14, pp. 77-106 (1998).
10. Holland, J.H., 1975. Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor.
11. Kochetov, Y., and Stolyar, A.: Evolutionary local search with variable neighbourhood for the resource constrained project scheduling problem, Proceedings of the 3rd International Workshop of Computer Science and Information Technologies (2003).
12. Kolisch, R., Drexl, A.: Adaptive search for solving hard project scheduling problems, Naval Research Logistics, 43, 23-40 (1996).
13. Kolisch, R., Hartmann, S.; Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling - Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, pp. 147-178 (1999).
14. Kolisch, R., Hartmann, S.: Experimental investigation of Heuristics for resource-constrained project scheduling: an update, working paper, Technical University of Munich (2004).
15. Kolisch, R., Sprecher, A.: PSPLIB - A project scheduling library, European Journal of Operational Research, 96, 205-216 (1996).
16. Kolisch, R.: Project scheduling under resource constraints - Efficient heuristics for several problem classes, Physica (1995).
17. Kolisch, R.: Serial and parallel resource-constrained project scheduling methods revisited: theory and computation, European Journal of Operational Research, 43, 23-40 (1996).
18. Kolisch, R.: Efficient priority rules for the resource-constrained project scheduling problem, Journal of Operations Management, 14, 179-192 (1996).
19. Leon V. J., Ramamoorthy, B.: Strength and adaptability of problem-space based neighbourhoods for resource-constrained scheduling, OR Spektrum, 17, 173-182 (1995).
20. Li, K.Y., Willis, R.J.: An iterative scheduling technique for resource-constrained project scheduling, European Journal of Operational Research, 56, 370-379 (1992).
21. Merkle, D., Middendorf, M., Schmeck, H.: Ant colony optimization for resource constrained project scheduling, IEEE Transaction on Evolutionary Computation, 6(4), 333-346 (2002).

22. Nonobe, K., Ibaraki, T.: Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP). In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Meta-heuristics, Kluwer Academic Publishers, Boston, pp. 557-588 (2002).
23. Schirmer, A.: Case-based reasoning and improved adaptive search for project scheduling, Naval Research Logistics, 47, 201-222 (2000).
24. Tormos, P., Lova, A.: A competitive heuristic solution technique for resource-constrained project scheduling, Annals of Operations Research, 102, 65-81 (2001).
25. Tormos, P., Lova, A.: An efficient multi-pass heuristic for project scheduling with constrained resources, International Journal of Production Research, 41, 1071-1086 (2003).
26. Tormos, P., and Lova, A.: Integrating heuristics for resource constrained project scheduling: One step forward, Technical report, Department of Statistics and Operations Research, Universidad Politecnica de Valencia (2003).
27. Valls, V., Ballestin, F., Quintanilla, S.: A hybrid genetic algorithm for the Resource-constrained project scheduling problem with the peak crossover operator, Eighth International Workshop on Project Management and Scheduling, 368-371 (2002).
28. Valls, V., Quintanilla, S., Ballestin, F.: Resource-constrained project scheduling: a critical activity reordering heuristic, European Journal of Operational Research, 149, 282-301 (2003).
29. Valls, V., Ballestin, F., Quintanilla, S.: A population-based approach to the resource-constrained project scheduling problem, Annals of Operations Research, 131, 305-324 (2004).
30. Valls, V., Ballestin, F.: Quintanilla, S.: Justification and RCPSP: A technique that pays, European Journal of Operational Research, Forthcoming.