A GRASP APPROACH TO THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

YURI KOCHETOV, ARTEM STOLYAR

Sobolev Institute of Mathematics, Siberian Branch of Russian Academy of Sciences, prospekt Akademika Koptyuga 4, 630090 Novosibirsk, RUSSIA, jkochet@math.nsc.ru, asto@math.nsc.ru

Abstract. In this paper we consider a well-known NP-hard resource constrained project scheduling problem. We develop an algorithm based on the new type of greedy heuristics. The heuristic supposes the solving of an auxiliary bottleneck problem forming the probabilistic strategy based on the optimal solution of the auxiliary problem. We use a forward-backward technique to improve the schedule. The algorithm contains the local descent phase. Two neighborhood structures are used in the standard local descent procedure. Computational results for the Patterson and PSPLib benchmark instances are presented. The results show that the new algorithm outperforms many existing methods for solving the resource constrained project scheduling problem.

Key Words. Project Scheduling, Sampling Methods, Bottleneck problem.

1 Introduction

In the resource-constrained project scheduling problem (RCPSP) we consider a single project which consists of m activities with non-preemptable duration. The activities are interrelated by precedence and resource constraints. All resources are renewable. The first (last) activity $j = 1$ ($j = m$) is defined to be the unique dummy source (sink). The objective is to schedule the activities such the precedence and resource constraints are obeyed and the makespan of the project is minimized.

We use the following notations [8]:

$J = \{1, \ldots, m\}$: the set of activities,

R: the set of resources,

d_j: processing time of activity j,

K_r: the total amount available of the resource r for each time period,

k_{jr}: the amount of the resource r for activity j per-period usage,

P_j: the set of the activity j immediate predecessors,

T_{max}: time horizon of the project.

Problem variables:

FT_j: the finish time of the activity $j \in J$,

A_t: a set of of activities which are in progress at the time t:

$$A_t = \{j \in J \mid FT_j - d_j \leq t < FT_j\}.$$

We present below the problem statement:

$$\text{Minimize } \max_{j \in J} FT_j,$$

$$\text{s.t. } FT_i \leq FT_j - d_j,\ i \in P_j,\ j \in J,$$

$$\sum_{j \in A_t} k_{jr} \leq K_r,\ \forall r \in R,\ t = 0, \ldots, T_{max},$$

The RCPSP as generalization of the well-known job-shop problem is NP-hard. Moreover, there is no polynomial-time approximation algorithm with a performance guarantee of $m^{1-\varepsilon}$ for any $\varepsilon > 0$ unless NP=ZPP [12].

There are either exact or heuristic approaches for solving the problem. Exact methods proposed are dynamic programming and branch and bound procedures [3, 11]. However, the algorithms are able to find an optimal solution for instances with up to 30 activities in a suitable computational time. The first heuristics proposed were priority-rule based scheduling methods. The heuristics have the advantage to be robust, intuitive and fast from the point of view the computational effort.

The most part of heuristics consist of two components, a schedule generation scheme (SGS) and a priority rule. Single-pass methods construct one deterministic schedule using one SGS and one priority rule. There are many possibilities to combine SGS and the priority rules to a multi-pass method. The most common ones are multi priority rule methods, forward-backward scheduling methods and sampling methods [9].

The greedy randomized adaptive search procedure (GRASP) [4] presented in this paper blends together two last classes and uses the optimal solution of the auxiliary bottleneck problem to generate a feasible schedule. The algorithm consists of three phases. At the first phase a sampling of feasible schedules is generated. We use the parallel method to construct the schedule. At the second phase each schedule from the sampling is improved by the forward-backward technique. The third phase is a local descent.

GRASP method

For $i = 1, \ldots, MaxIter$ do

1 Construct a heuristic solution

2 Apply forward-backward procedure

3 Apply local descent procedure

4 Update best found solution.

The paper is organized in the following manner. In the Section 2 we present the basis of the parallel scheduling method. In Section 3 we discuss the schedule construction using the auxiliary bottleneck problem. Section 4 is devoted to the forward-backward phase. Local descent phase is presented

2 The parallel method

The parallel method is one of the famous fast heuristic for the RCPSP [6]. It consists of at most m stages. At the stage n, we have a schedule time t_n and three sets of activities:

complete set $C_n = \{j \in J \mid FT_j \leq t_n\}$,

active set $A_n = \{j \in J \mid FT_j - d_j \leq t_n < FT_j\}$,

decision set $D_n = \{j \in J \setminus (C_n \cup A_n) \mid P_j \subseteq C_n,\ k_{jr} \leq K_r - \sum_{i \in A_n} k_{ir},\ \forall r \in R\}$.

The complete set C_n comprises all activities which have been completed up to time t_n. The active set A_n comprises all activities which are in progress at time t_n. The decision set D_n comprises all activities which are sequence feasible at time t_n. Our aim is to select a subset $D' \subseteq D_n$ of activities which will start at the time t_n.

Parallel method

1. Put $n := 0,\ t_n := 0,\ C_n := \emptyset,\ A_n := \{1\},\ FT_1 := 0$.

2. While $|C_n + A_n| < m$ do

2.1 $n := n + 1,\ t_n := \min\{FT_j \mid j \in A_{n-1}\}$.

2.2 Compute $C_n,\ A_n,\ D_n$.

2.3 $\widetilde{K}_r(t_n) := K_r - \sum_{j \in A_n} k_{jr},\ \forall r \in R$.

2.4 Select a subset $D' \subseteq D_n$ such that $\sum_{j \in D'} k_{jr} \leq \widetilde{K}_r(t_n),\ \forall r \in R$

2.6 Put $FT_j := t_n + d_j,\ \forall j \in D'$.

Step 2.4 is the most important part of the method. To select the subset $D' \subseteq D_n$ we propose a probabilistic strategy based on an optimal solution of an bottleneck problem.

3 Auxiliary bottleneck problem

Suppose that we know the optimal solution of RCPSP and for each activity $j \in D_n$ we may com-

a_j: the lateness of activity j if it starts at the time moment t_n,

b_j: the lateness of activity j if it starts at the earliest time moment $t > t_n$.

For a given schedule time t_n and a partial schedule $\{FT_j \mid j \in C_n \cup A_n\}$ we need to select a subset $D' \subseteq D_n$. It can be done by solving the following bottleneck problem:

$$\text{Minimize } \{\max_{j \in D_n}(a_j x_j + b_j(1 - x_j))\}$$

$$\text{s.t. } \sum_{j \in D_n} k_{jr} x_j \le \widetilde{K}_r(t_n),\ r \in R$$

$$x_j \in \{0, 1\},\ j \in D_n.$$

The objective function evaluates a lateness of the project. Variables $x_j,\ j \in D_n$ allow us to get desirable set $D' = \{j \in D_n \mid x_j = 1\}$.

Proposition. The bottleneck problem is polynomial solvable.

In order to define the parameters of the bottleneck problem we need to analyse the schedule at the time t_n. Suppose that the set D' is given. We assume that D' is complete in the sense that there is no activity $j \in D_n \backslash D'$ which could be added to the set D' without violating of the resource constraints at time t_n. Consider an activity $j \in D_n \backslash D'$ and define its earliest resource-feasible starting time taking into account the set D'. The desirable time t'_j is a minimal time among finish times of activities from the set $A_n \cup D'$ such that the available capacity of resources is enough for the activity j:

$$t'_j = \min \gamma,\ \gamma \in \{t_n + d_i \mid i \in A_n \cup D'\}$$

$$\text{s.t. } k_{jr} \le K_r - \sum_{i \in A_n \cup D',\ t_n + d_i > \gamma} k_{ir}, \forall r \in R.$$

The value t'_j is calculated with time complexity $O(|A_n \cup D'| \cdot |R|)$.

Note that the value t'_j depends on the set D'. Let $\widetilde{t}_j$ be a maximum among all values t'_j corresponding to all possible sets D'. Unfortunately, it's impossible to compute it in polynomial time. So, we use the upper bound $\overline{t}_j \ge \widetilde{t}_j$ which is a pessimistic value for the earliest resource-feasible starting time of the activity j for any set D'. We define it by the following way:

$$\overline{t}_j = \min\{\gamma = t_n + d_i,\ i \in A_n \cup D_n \setminus \{j\}\}$$

Let LF_j be the latest finish time of the activity j as determined by backward recursion omitting resource constraints when T is used as the critical path method based lower bound of the optimal solution. The differences $t_n - (LF_j - d_j)$ and $\overline{t}_n - (LF_j - d_j)$ characterize the project delay depending on the activity j finish time. So, we can define the parameters of the bottleneck problem as:

$$a_j = t_n - (LF_j - d_j), \quad b_j = \overline{t}_n - (LF_j - d_j).$$

It is known that probabilistic multi-pass methods show better results than deterministic single-pass methods [8]. If a one pass is not time-consuming, we may decrease the relative error by increasing the number of trials.

Let x^*_j be an optimal solution of the bottleneck problem. Followed by the probabilistic multi-pass strategy, we select a random subset of the set $D^1 = \{j \in D_n \mid x^*_j = 1\}$. Afterward, we add several activities from the set $D^0 = \{j \in D_n \mid x^*_j = 0\}$ subject to the resource constraints. For this purpose consider a probability p. Sort the activities in D^1 and D^0 according to the *resource utilization ratio*, $(1/|R|)\sum_{r \in R} k_{jr}/K_r$. Consider the first activity from the D^1 according to the order predefined and insert it into the set D' with probability p or cancel in and consider the next activity with probability $1-p$ and so on. When the set D^1 is exhausted repeat the same operation with the set D^0 while the resource constraints are fulfilled. If the the value p is too small the set D' may be empty. In this case we fill the set D' randomly taking into account the resource constraints. As a result, we have a random subset D' at the step 2.4 of the parallel method. So, we obtain a random schedule. In order to improve the makespan we apply the forward-backward strategy [9] to the schedule.

4 Forward-backward phase

Consider a schedule S_0. Following the forward-backward procedure we consequently transform the left-active schedule into the right-active schedule an vice-versa. Consider list $L = (j_1, \ldots, j_m)$. Serial method construct an active schedule for the list L.

Serial method(L)

2. Calculate the earliest precedence- and resource-feasible starting time t for the activity j_n.

3. Put $FT_{j_n} = t + d_{j_n}$.

4. Put $n := n + 1$;

5. If $n \leq m$ then goto 2.

In the same manner we can construct the right-active schedule considering the activities in reverse order.

Inverse Serial method(L, T)

1. Put $n := m,\ t := T,\ FT_{j_n} := t$.

2. Calculate the latest precedence- and resource-feasible finish time t for the activity j_n.

3. Put $FT_{j_n} = t$.

4. Put $n := n - 1$;

5. If $n \geq 1$ then goto 2.

Let $T = T(S_0)$ is a schedule S_0 makespan. Construct a list L_1 of activities in non-decreasing order of its finish times: $FT_{j_n} \leq FT_{j_{n+1}},\ n = 1, \ldots, m-1$. Apply to the list L_1 the serial decoder in reverse manner. As result we obtain a *right-active* schedule S_1. Construct a list L_2 of activities in non-decreasing order of its start times: $FT_{j_n} - d_{j_n} \leq FT_{j_{n+1}} - d_{j_{n+1}}, n = 1, \ldots, m-1$. Apply to the list L_2 the serial decoder. Obtain an active schedule S_2. If $T > T(S_2)$ then repeat the previous actions.

Below we present the formal statement of the Forward-Backward phase.

Forward-Backward method

1. Put $T := \max_{j \in J} FT_j$ for the schedule S_0.

2. Construct the list $L_1 = (j_1, \ldots, j_m)$, such that $FT_{j_n} \leq FT_{j_{n+1}},\ n = 1, \ldots, m-1$.

3. Construct a right-active schedule S_1 by Inverse Serial(L_1).

4. Construct the list $L_2 = (j_1, \ldots, j_m)$, such that $FT_{j_n} - d_{j_n} \leq FT_{j_{n+1}} - d_{j_{n+1}},\ n = 1, \ldots, m-1$.

5. Construct an active schedule S_2 by Serial(L_2).

6. If $T > \max_{j \in J} FT_j$ then put $S_0 := S_2$ and goto 1.

of returns to the step 1 could be quite large. Unfortunately we cannot evaluate it by polynomial function of initial data. On the other hand, computational results show small number of returns to the step 1. As a matter of fact, the number is much less than m.

5 Local descent

We apply standard local descent procedure to the schedule obtained at the previous stage. We consider two neighborhoods of linear cardinality.

5.1 Neighborhood N_1

The neighborhood N_1 was proposed by Brucker et al. [1] for tabu search algorithm. It is based on critical arcs in a special digraph. Consider a list of activities L and correspondent active schedule S. Let $G = (V, E)$ be a digraph where $V = J$ and $E = \{(i, j) \mid FT_i = FT_j - d_j\}$. The longest path in G is called the *critical path*. The arc (i, j) is called the *critical arc* if it belongs to a critical path and $i \notin P_j$. Define three move operators for a critical arc (i, j).

Let activity i is listed before than activity j in the list L. Operator $Shift_{ij}(L)$ moves the activity i immediately after than activity j together with all its successors listed before than activity j.

$Shift_{ij}(L)$:

$$(\ldots i \ldots\ t \ldots\ j \ldots) \rightarrow (\ldots\ j\ i\ t \ldots).$$

Let activity i is listed after than activity j in the list L. Operator $BShift_{ij}(L)$ moves the first activity l which is not a successor of the activity i immediately before than activity i.

$BShift_{ij}(L)$:

$$(\ldots j \ldots\ i \ldots\ l \ldots u \ldots) \rightarrow (\ldots\ j \ldots l\ i \ldots u \ldots).$$

Symmetrically, Operator $FShift_{ij}(L)$ moves the last activity l which is not a predecessor of the activity j immediately after than activity j.

$FShift_{ij}(L)$:

$$(\ldots u \ldots\ l \ldots\ j \ldots i \ldots) \rightarrow (\ldots\ u \ldots j\ l \ldots i \ldots).$$

The neighborhood N_1 is a set of lists obtained by applying three move operators to all critical arcs

5.2 Neighborhood N_2

The neighborhood N_2 is constructed using a solution (not necessary optimal) of the multi-dimensional knapsack problem. Consider a list of activities L and correspondent active schedule S. By $Block(j)$ we denote the set of activities which are overlapped together with the activity j in the schedule S. We assume that the activities finishing immediately before and starting immediately after than the activity j also belong to the set $Block(j)$. Let $\widehat{G} = (V, E)$ be a digraph where $V = J$ and $E = \{(i, j) \mid FT_i = FT_j - d_j,\ i \in P_j\}$. The maximal connected subgraph of $\widehat{G}$ with a source in the vertex j is called *outcoming network* of the activity j. Define two positions $First$ and $Last$. The $First$ is a minimal position in the list L among all activities of the $Block(j)$. The $Last$ is a maximal position in the list L among all activities of the $Block(j)$ and the outcoming network.

The move operator is defined for any activity j for which the $Block(j)$ does not contain any predecessor of the activity j. The neighbor schedule S' is constructed in three steps. At the first step we construct a partial schedule by serial SGS for the first $(First - 1)$ activities in the list L. At the second step we extend the partial schedule by inserting the next $(Last - First + 1)$ activities into the partial schedule. For this purpose we use the parallel SGS where the selection of the set D' at the step 2.5. is done using the greedy solution of the multi-dimensional knapsack problem. Finally, we construct the complete schedule by inserting the remaining activities into the partial schedule following the serial SGS.

The neighborhood N_2 consists of all neighbor schedules. The cardinality of N_2 is $O(m)$.

6 Computational experiments

We test the GRASP algorithm for the Patterson and PSPLib benchmark instances.

In Table 1 we present a comparison of the new algorithm without local descent phase against the ASP algorithm [8] on the Patterson instances [13]. We use the sampling size $Z = 1, 10, 100, 500$. The that the new approach allows to decrease relative error and get optimal solutions quite often. The running time of the both algorithms is extremely small.

		1	10	100	500
Average deviation	ASP	3,7	1,8	0,8	0,4
	GRASP	2,3	1,6	0,4	0,1
Standard deviation	ASP	4,3	2,4	1,6	1,2
	GRASP	3,3	2,7	1,1	0,4
Maximal deviation	ASP	16,7	12,1	9,4	6,5
	GRASP	13,9	12,9	5,0	2,6
Percent of opt. solved	ASP	38,2	51,8	76,4	85,5
	GRASP	55,6	62,7	85,5	96,4

Table 1. Comparison of algorithms on the Patterson instances.

Algorithm	30	60	120
GRASP	0,28	11,74	34,46
Genetic Algorithm [5]	0,54	12,68	39,37
sampling - LFT [6]	1,40	13,59	39,60
sampling - WCS [7]	1,40	13,66	39,65
CBR-SAR [14]	0,71	13,02	40,08
ASP [8]	0,74	13,51	41,37
Simulated Annealing [2]	0,38	12,75	42,81
Genetic Algorithm [10]	2,08	14,33	42,91

Table 2. Average percent deviation, 1000 iterations, PSPLib instances.

In Table 2 we compare our algorithm with several algorithms from the literature. We present average percent deviation from the optimum for the PSPLib instances with 30 activities and deviation from CPM-based lower bound for the instances with 60 and 120 activities. Computational results show that new approach has better performance for these benchmarks.

Dimension	FB	N_1	N_2
30	0,39	0,28	0,30
60	12,11	11,79	11,74
120	36,25	34,68	34,46

Table 3. Local descent effort.

In table 3 we present the average relative error for the GRASP algorithm with local descent. First column (FB) corresponds to the GRASP without

to the neighborhoods N_1 and N_2 respectively. Table 3 shows that using of local descent allows us to decrease the relative error. Note, that the first neighborhood is better for small instances, while the second one is better for more hard problems. The computational time for both neighborhoods are the same. The depth of the descent not exceed 10 iterations.

7 Conclusions

A GRASP algorithm is presented for the resource constrained project scheduling problem is considered. The heuristic solution is constructed using the optimal solution of an auxiliary bottleneck problem. To improve the schedule the algorithm uses a forward-backward technique and standard local descent procedure. Two neighborhood structures of linear cardinality are considered. Computational results show the high performance of the algorithm.

Acknowledgments

This research is supported by Russian Foundation for Basic Research grant 01-03-00455.

References

[1] Baar T., Brucker P., Knust S. Tabu-search algorithms for the resource-constrained project scheduling problem. *Working Paper*, Universität Osnabrück, 1997.

[2] Bouleimen, K. and Lecocq, H.: A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem. *Technical Report*, Université de Liège, (1998).

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

[4] Feo, T. A., Resende, M. G. C.: Greedy randomized adaptive search procedures. *J. Glob. Optim.* 6 (1995) 109–133.

[5] Hartmann, S.: A competitive genetic algorithm for resource-constrained project

[6] Kolisch, R.: Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2)(1996) 320–333.

[7] Kolisch, R.: Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14(3) (1996) 179–192.

[8] Kolisch, R. and Drexl, A.: Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43(1) (1996), 23–40.

[9] Kolisch, R. and Hartmann, S.: Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In: J. Węglarz (ed.): *Project Scheduling: Recent Models, Algorithms and Applications*. Kluwer Acadimic Publishers, Boston, (1999), 147–178.

[10] Leon, V. and Ramamoorthy, B.: Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spektrum*, 17(2/3) (1995), 173-182.

[11] Mingozzi A., Maniezzo V., Ricciardelli S., Bianco L. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation *Manag. Sci.* 44 (1998), 714–729.

[12] Möhring R. H., Schulz A. S., Stork F., Uetz M. Solving project scheduling problems by minimum cut computations *Manag. Sci.*, 49(3) (2003), 330–350.

[13] Patterson, J.H.: A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30 (1984), 854–867.

[14] Schirmer, A.: Case-based reasoning and improved adaptive search for project scheduling. *Technical Report 472*, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1998.