Sprecher, Arno

Solving the RCPSP efficiently at modest memory requirements

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 425

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Sprecher, Arno (1996) : Solving the RCPSP efficiently at modest memory requirements, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 425, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/177305

Manuskripte

aus den

Instituten für Betriebswirtschaftslehre

der Universität Kiel

No. 425

Solving the RCPSP Efficiently at Modest Memory Requirements [1]

Arno Sprecher

December 1996

Arno Sprecher, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstraße 40, 24098 Kiel, Germany.
Email: sprecher@bwl.uni-kiel.de, WWW: http://www.wiso.uni-kiel.de/bwlinstitute/prod, FTP: ftp://www.wiso.uni-kiel.de/pub/operations-research.

[1] Supported by the Deutsche Forschungsgemeinschaft

Abstract

As a natural extension of the classical MPM, algorithms for the resource-constrained project scheduling problem have attracted much attention over the last two decades. Enumerating delay alternatives, extension alternatives, feasible completion times, feasible posets, or feasible subsets, all the methods aim at finding a makespan minimal schedule among the precedence and resource feasible ones. Applicability is mainly limited to problems with constant resource availability and/or problems allowing only a single processing alternative per activity.

The purpose of this paper is to direct the focus to a procedure that can, by simple adaptations, operate on more general problem settings. The general approach can, e.g., deal with multi-mode problems, resource availability varying with time, and a wide range of objectives. Although the algorithm is the most general and simple one currently available for resource-constrained project scheduling, the computational performance can compete with the best approaches available for the single-mode problem. The algorithm uses far less memory than the state-of-the-art procedure, i.e., 400 KB versus 24 MB, for solving the standard benchmark set with projects consisting of 32 activities. It definitely outperforms the state-of-the-art procedure if both approaches are allowed to make limited use of memory. Since, in general, the memory requirements exponentially grow with the number of activities the project consists of, memory will become a critical resource. Conservative estimates of memory requirements for the state-of-the-art approach have to be settled at least at 500 MB and for our approach at most at 5 MB for 62-activity projects. Additionally, heuristic capabilities of the truncated version of our algorithm are extremely encouraging.

The procedure has been coded in C and implemented on a personal computer. The computational results, show that, beside of the theoretical benefits (1) ease of description, (2) ease of implementation, and (3) ease of generalization, practical advantages as (1) reasonable performance and (2) low memory requirements will make its use favorable when attacking larger problems or variants of the resource-constrained project scheduling problem.

Keywords: Project Scheduling, Resource Constraints, Single-Mode, Branch-and-Bound, Heuristic, Computational Results.

1 Introduction

In the early beginnings of project scheduling CPM and MPM have been developed to support the project manager in doing his work. Assuming deterministic durations of the activities that build up the project, both methods mainly determine time-windows, i.e., intervals, where the activities can be performed in without violating given precedence relations and a given project completion time, i.e., makespan. The limitation of the resources required to execute the activities are not taken into account.

Since the limitation of the resource availability cannot be relaxed in the major part of business applica-

tions the research community has answered the more realistic assumptions of the resources' limitation by intensive research. As a generalization of the flow-shop, job-shop, and open shop problem the resource-constrained project scheduling problem (RCPSP) is known as an NP-hard problem (cf. [10]). Therefore the main focus is on the development of branch-and-bound algorithms where different ideas have been presented to build the tree guiding the enumeration of the schedules. The schemes enumerate minimal delaying alternatives (cf. [5], [6]), feasible completion times (cf. [31]), feasible extensions (cf. [29]), feasible posets (cf. [21]), and feasible subsets (cf. [16]), in order to find an optimal, i.e. makespan minimal, solution. The currently most advanced procedure has been developed by Demeulemeester and Herroelen (cf. [6]) which enhances their earlier work (cf. [5]) by a bound introduced by Mingozzi et al. (cf. [16]) and fully exploits nowadays available 32-bit architectures of personal computers. The procedure solved the entire set of benchmark problems generated by ProGen (cf. [15]) for the first time. The projects consist of 32 activities (including two dummy activities) and 4 renewable resources. The CPU-time on a personal computer (80486, 25 MHz, 32 MB) under Windows NT averages at some 34 seconds at the cost of 24 MB memory used.

Recent advances again follow the requirements of practice. Alternative process plans allow to fulfill the task related to an activity in different ways, called modes. The activities can be executed in one out of several modes. The modes reflect alternative combinations of resources and belonging quantities employed to fulfill the tasks related to the activities. The activity duration is a discrete function of the employed quantities, that is, using this concept e.g. working-off an activity can be accelerated by raising the quantities coming into operation (time-resource-tradeoff). Moreover, by raising the quantities of some resources and reducing the quantities of others the resource substitution (resource-resource-tradeoff) can be realized. The problem derived is the multi-mode resource-constrained project scheduling problem (MRCPSP), which is commonly considered with makespan minimization as objective (cf. [30]).

In this paper we will return to a procedure originally developed for the single-mode problem (cf. [31]), then generalized to the multi-mode case (cf. [30]), and substantially simplified to the precedence tree guided scheme (cf. [19], [20]). Later on, Sprecher and Drexl (cf. [24], [25], [26]) employed the precedence tree to form the currently most simple, general, and powerful algorithm for the multi-mode resource-constrained project scheduling problem. The size of the problems that can be solved to optimality has been nearly doubled. Projects with up to 22 activities (including of two dummy activities) with 3 modes per activity can be solved. The CPU-time for the 22-activity projects on a personal computer (80486, 66 MHz, 16 MB) under OS/2 averages at some 240 seconds if 2 renewable and 2 nonrenewable resources are taken into account, and 12 seconds if only 2 renewable resources are limited. In the former case at

most 8 MB memory have been used.

We will adapt the algorithm developed for the MRCPSP to the RCPSP and enhance it by some RCPSP specific rules. The computational experience reveals that the algorithm can compete with the state-of-the-art-approach from Demeulemeester and Herroelen (cf. [6]) at far less memory requirements. The approach we present uses only 400 KB instead of the 24 MB required by Demeulemeester and Herroelen to solve the 32-activity projects.

We proceed as follows: In Section 2 we describe the problem more precisely. In Section 3 we give a brief summary of solution procedures proposed for the RCPSP. In Section 4 we present the single-mode version of the algorithm and the bounding rules used to accelerate its convergence. In Section 5 we reveal our computational results. In Section 6 we draw the conclusions for future research.

2 The Model

We consider a project which consists of J activities (jobs, tasks). Due to technological requirements, precedence relations between some of the activities enforce that an activity j, $j = 2, \ldots, J$, may not be started before all its predecessors h, $h \in \mathcal{P}_j$, are finished. The structure of the project is depicted by a so-called activity-on-node (AON) network where the nodes and the arcs represent the activities and precedence relations, respectively. The network is acyclic and numerically labeled, that is an activity j has always a higher number than all its predecessors. W.o.l.o.g. activity 1 is the only start activity (source) and activity J is the only finish activity (sink).

The activities may not be preempted, i.e., an activity once started has to be completed without interruption. Performing activity j takes d_j periods and is supported by a set R of renewable resources (cf. [32], [33]). Given a horizon, that is, an upper bound $\overline{T}$ on the project's makespan, K_{rt} units of renewable resource r, $r \in R$, are available in a period t, $t = 1, \ldots, \overline{T}$. Performing an activity j, $j = 1, \ldots, J$, requires k_{jr} units of renewable resource r, $r \in R$, each period activity j is in process. The parameters are summarized in Table 1 and assumed as integer-valued. The objective is to find a makespan minimal schedule that meets the constraints imposed by the precedence relations and the limited availability of the renewable resources.

Presuming feasibility and a constant per-period availability of the renewable resources, an upper bound on the minimum makespan is given by the sum of the activity durations. Given an upper bound $\overline{T}$ on the project's makespan we can use the precedence relations to derive time windows, i.e. intervals $[EF_j, LF_j]$, with earliest finish time EF_j and latest finish time LF_j, containing the precedence feasible completion times of activity j, $j = 1, \ldots, J$, by traditional forward and backward recursion as performed in MPM.

J	:	number of activities
d_j	:	duration of activity j
R	:	set of renewable resources
$\overline{T}$	:	upper bound on the project's makespan
$K_{rt} \geq 0$	:	number of units of renewable resource r, $r \in R$, available in period t, $t = 1, \ldots, \overline{T}$
$k_{jr} \geq 0$	:	number of units of renewable resource r, $r \in R$, used by activity j each period the activity is in process
$\mathcal{P}_j$ $(\mathcal{S}_j)$	:	set of immediate predecessors (successors) of activity j
ES_j (EF_j)	:	earliest start time (finish time) of activity j, calculated by neglecting resource usage
LS_j (LF_j)	:	latest start time (finish time) of activity j, calculated by neglecting resource usage and taking into account the upper bound $\overline{T}$ on the project's duration

Table 1: Symbols and Definitions

Analogously, the interval $[ES_j, LS_j]$ bounded from below and above by the earliest start time ES_j and the latest start time LS_j, respectively, can be calculated to reflect the precedence feasible start times. The benefit is twofold: First, the number of variables used in the integer (binary) programming formulation is reduced substantially. Second, within a branch-and-bound algorithm the bounds can be efficiently used to speed up the convergence.

Using the time windows derived we can now state the problem as a linear program as presented by Talbot (cf. [30]). We use binary decision variables x_{jt}, $j = 1, \ldots, J$, $t = EF_j, \ldots, LF_j$,

$$x_{jt} = \begin{cases} 1 & , \text{ if activity } j \text{ is completed at the end of period } t \\ 0 & , \text{ otherwise.} \end{cases}$$

The model is presented in Table 2 and referred to as the (single-mode) resource-constrained project scheduling problem (RCPSP).

Since there is exactly one finish activity, the objective function (1) represents the minimization of the project's makespan. Constraints (2) ensure that exactly one completion time is assigned to each activity. The precedence relations are taken into account by (3). (4) guarantees, that the per-period availabilities of the renewable resources are not exceeded.

Obviously, the well-known flow-shop, job-shop-, open-shop and assembly line balancing problem are included in the model outlined above (cf., e.g., [24], pp. 10). Thus, the problem is a member of the class of NP-hard problems (cf. [10]).

Moreover, the model presented above can be easily extended to include different modes to perform the activities , time-varying request (cf. [8]) and generalized temporal constraints (cf. [1], [24], pp. 19).

$$\text{Minimize } \Phi(x) = \sum_{t=EF_J}^{LF_J} t \cdot x_{Jt} \tag{1}$$

s.t.

$$\sum_{t=EF_j}^{LF_j} x_{jt} = 1 \qquad j = 1, \ldots, J \tag{2}$$

$$\sum_{t=EF_h}^{LF_h} t \cdot x_{ht} \leq \sum_{t=EF_j}^{LF_j} (t - d_j) x_{jt} \qquad j = 2, \ldots, J, h \in \mathcal{P}_j \tag{3}$$

$$\sum_{j=1}^{J} k_{jr} \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} \leq K_{rt} \qquad r \in R, t = 1, \ldots, \overline{T} \tag{4}$$

$$x_{jt} \in \{0, 1\} \qquad j = 1, \ldots, J, t = EF_j, \ldots, LF_j$$

Table 2: The Model of the RCPSP

However, note, if negative minimal time-lags are incorporated then the objective function has to be adapted to reflect the minimization of the makespan. Additionally, beside the makespan other objectives as the minimization of the weighted delays, the minimization of the total number of tardy activities, the minimization of the mean weighted flow time, the maximization of the net present value as well the smoothness of the resource profile can be easily modeled (cf. [22], [23]).

3 Literature Review

In this section we will briefly summarize enumeration procedures proposed for solving the RCPSP. Stinson et al. (1978) (cf. [29]) provided a branch-and-bound algorithm that enumerates the extension alternatives (cf. [11]) of partial schedules. The partial schedules reflect scheduling decisions already made for a subset of the set of activities. The partial schedules are always feasible with respect to precedence and resource constraints. Given a partial schedule, the next decision point is determined by the time incrementing scheme introduced by Johnson (cf. [12]). That is, assuming constant resource availability, it is sufficient to study the completion times of the activities already scheduled as candidates for start times of new activities. Consequently, the decision point can be determined as the minimal completion time of the activities contained in the partial schedule. At the decision point it is considered to extend the partial schedule by activities that are eligible, i.e., by activities all the predecessors of

which are in the partial schedule and finish at or before the decision point. From the set of eligible activities, the subset of schedulable activities is built. The schedulable activities are the ones that can be selected on their own to be started at the decision point without violating the resource constraints. The extension alternatives are built by all the subsets of the schedulable activities that can extend the partial schedule without violating the resource constraints. They form the descendants of the current node of the branch-and-bound tree. The scheme starts with the empty partial schedule and successively extends the current partial schedule by determining the next decision point and selecting an extension alternative. Backtracking is performed if all the alternatives are evaluated at the current node of the branch and bound tree. The basic scheme is enhanced by dominance pruning as initially developed by Johnson (cf. [12], quoted in accordance with [29], and Subsection 4.2). Moreover, (i) a precedence-based bound relying on MPM calculations of latest finish times of the activities for a given project duration, and (ii) a resource-based bound is used. Combining the effects of precedence and resource constraints Stinson et al. propose the critical sequence lower bound. Finally, left-shifts are studied to detect further dominance (cf. Section 4).

Demeulemeester and Herroelen (1992) (cf. [5]) presented an enumeration scheme relying on the idea of resolving resource conflicts by delaying some of the activities causing the conflict (cf. [4]). The procedure continues a given partial schedule by temporarily scheduling all the eligible activities at the decision point as determined by Johnson (cf. [12]). If the cumulated resource requests at the decision point exceed the availability then it is branched to the next level. At this level the delaying alternatives are considered to resolve the conflict. Thereby, a delaying alternative is a subset of the set of activities in process at the decision point, the delay of which makes the partial schedule resource feasible. If no resource conflict occurs then the next decision point is determined. The algorithm tracks back if at the current level all the delaying alternatives are evaluated. Demeulemeester and Herroelen prove that it is sufficient to study only the minimal delaying alternatives. Moreover, they make use of the precedence-based bound and the critical sequence bound as well as additional dominance concepts. The concepts make use of left-shift dominance (cf. Subsection 4.2) and, moreover, a cut-set rule similar to the dominance pruning used by Stinson et al. and the network-cuts as employed by Talbot and Patterson (cf. [31]). Finally, Demeulemeester and Herroelen introduce two immediate selection strategies (cf. [2]). The procedure has been tested on the Patterson-Set (cf. [18]) and compared with the approach developed by Stinson et al. (cf. [29]). The 110 problems have up to 51 activities. On the problem set the solution time of the enumeration scheme by Demulemeester and Herroelen averages at 0.21 seconds (IBM PS/2 Model 70 A21, 25 MHz) and outperforms the one by Stinson et al. by a factor of nearly 12. The procedure has been generalized to the multi-mode case by Sprecher et al. (cf. [27]).

Kolisch et al. (1995) (cf. [15]) developed the parameter-driven project generator ProGen as a tool for the evaluation of algorithms proposed for resource-constrained project scheduling. Allowing to specify, e.g., the number of activities, the number of resources, activity durations, number of modes, resource requests and availabilities by defining lower and upper bounds, ProGen randomly generates instances with a given network-complexity, resource factor and resource strength. A set of 480 projects with 32 activities (two dummy activities are included) has been generated and used to test the approach by Demeulemeester and Herroelen (1992) on a personal computer (IBM PS/2 Model 55sx, 80386sx, 15 MHz). Whereas the Patterson-set has been solved within 1.06 seconds on average only 415 of the 480 problem instances have been solved within a time limit of 1000 seconds per problem.

Mingozzi et al. (1996) (cf. [16]) developed an enumeration procedure relying on the concept of feasible subsets. They define a feasible subset as a subset of the set of activities, where (i) the sum of the resource requirements does not exceed the availability for any resource, and (ii) there is no precedence relation between any pair of activities out of the subset. The algorithm starts with two empty (ordered) lists, the subset list and the duration list. The former one to store the feasible subsets and the latter one to store the processing times of the feasible subsets. The algorithm seeks to complete the lists to represent solutions for the RCPSP by adding feasible subsets and processing times of the subsets to the lists. The lists finally obtained represent a feasible schedule, if (a) every activity is executed without interruption, (b) the processing times of the feasible subsets coincide with the durations of the activities, and (c) the starting times respect the precedence constraints imposed. At each node of the branch-and-bound tree the current lists are continued by adding a feasible subset and a processing time to the lists, such that, (a) no non-completed activity is interrupted, and (b) the predecessors of the activities within the feasible subset considered to be added, are completed. The processing time of the chosen feasible subset is defined by the minimum processing time required to complete an activity out of the feasible subset. The basic enumeration scheme has been enhanced by a rule reducing the set of feasible subsets $N(\alpha)$ (cf. [16]) to be tested at a certain node α, a variant of the left-shift rule (cf. [29], and Subsection 4.2), and the cut-set rule (cf. [5], and Subsection 4.2). Moreover, five bounds are derived from the new mathematical programming formulation. The authors report that the bounds perform better than the critical sequence bound introduced by Stinson et al. (cf. [29]), and that their algorithm is competitive to the procedure presented by Demeulemeester and Herroelen (cf. [5]), the best one known up to then.

Demeulemeester and Herroelen (1996) (cf. [6]) enhanced their approach from [5] by a variant of the bound LB3 from Mingozzi et al. (cf. [16]). Moreover, they optimized their code by changing for$(i = 1; i \leq n; i++)$-loops to for$(i = n; i \geq 1; i--)$-loops, and representing four resources of 8 bit size

through one 32 bit unsigned integer. Allowing as much as 24 MB Demeulemeester and Herroelen could solve the entire standard benchmark-set (cf. [15]) for the first time. The CPU-time of their Microsoft Visual C++ 2.00 implementation averages at some 34 seconds on a personal computer (80486, 25 MHz) operating under Windows NT.

In contrast to the approaches sketched above the one developed by Talbot and Patterson (1978) (cf. [31]) allows to consider resource availabilities varying with time. Moreover, since only one activity is scheduled per node of the branch-and-bound tree, only simple data structures are necessary to implement the algorithm. Using an upper bound on the project's makespan the time-windows from MPM-calculations are employed to limit the start times of the activities. As already mentioned the procedure schedules one activity per node of the branch-and-bound tree. Assuming a numerically labeled network always the lowest indexed activity not in the partial schedule can be selected for assigning a start time to it. The first start time assigned to the activity is the lowest resource and precedence feasible one. If an activity cannot be scheduled within the time limits without violating the resource or precedence constraints, then backtracking to the previous level is performed. There the next precedence and resource feasible start time which is greater than the one previously assigned is selected. To accelerate the enumeration the concept of network-cuts is introduced.

Talbot (1982) (cf. [30]) extends the approach to the multi-mode case, where the next untested mode is selected and evaluated before tracking back. Later on Patterson et al. (1989) (cf. [19]) developed the precedence tree to guide the search for an optimal solution of the multi-mode problem.

Sprecher and Drexl (1994, 1996) (cf. [24], [25], [26]) made use of the precedence tree to guide the search for an optimal solution of the multi-mode resource-constrained project scheduling problem. The size of the problems that can be solved to optimality has been nearly doubled. Projects with up to 22 activities (inclusively of two dummy activities) with 3 modes per activity can be solved. The CPU-time for the 22-activity projects on a personal computer (80486, 66 MHz, 16 MB) under OS/2 averages at some 240 seconds if 2 renewable and 2 nonrenewable resources are taken into account, and 12 seconds if only 2 renewable resources are limited. In the former case at most 8 MB memory have been used.

As we will see in the following sections the concepts used for the multi-mode algorithms can be successfully employed for the single-mode problem as well. Although the approach is more general, the algorithm produces results competitive to the one of the best special purpose codes with respect to solution times, and uses far less memory.

Brucker et al. (1996) (cf. [3]) present a branch-and-bound algorithm basing on a generalization of the disjunctive graph model. The nodes of the branch-and-bound tree correspond to so-called schedule schemes (sets of disjunctions, conjunctions, parallelity, and flexibility relations). The algorithm has

been tested on a workstation (SUN/Sparc 10/512) operating under Solaris 2.4 with 64 MB general storage (2x50 MHz + 1 MB SC) and compared with the algorithm by Demeulemeester and Herroelen (cf. [5]). Within an imposed time limit of 3600 seconds the algorithm could not find and verify an optimal solution for 20 so-called easy instances and 31 so-called hard instances of the ProGen set.

4 The Branch-and-Bound Algorithm

In this section we describe the single-mode variant of our algorithm. In Subsection 4.1 we briefly summarize the basic scheme for the RCPSP. In Subsection 4.2 we describe the bounding-rules by illustrations. Since their validity mainly can be deduced from the illustrations, we will give proofs only, if the statements generalize the concepts introduced in [25], [26], or if they are entirely new. The remaining proofs can be found in [25], [26].

4.1 The Basic Scheme

As for the MRCPSP the search for an optimal solution is guided by the precedence tree introduced by Patterson et. al. (cf. [19]). The nodes of the precedence tree correspond to the nodes of the branch-and-bound tree. The root node 1 of the tree is given by the single start activity and the leaves are copies of the only finish activity J. The descendents of a node j within the precedence tree are built by the activities that are eligible after scheduling the activities on the path leading from the root node 1 to node j. Thereby, in contrast to [5] and [29], an activity is called eligible, if all its predecessors are scheduled. Analogously to the algorithm for the MRCPSP we use the set $\mathcal{ACS}_i$ to denote the set of activities currently scheduled up to level i. Assuming that passing the nodes of the precedence tree means scheduling the activities associated with the nodes we obtain the set of eligible activities on level i, namely Y_i, as follows:

$$
\begin{aligned}
Y_1 &:= \{g_1\} = \{1\} \\
\mathcal{ACS}_1 &:= \{g_1\} = \{1\} \\
Y_{i+1} &:= Y_i \backslash \{g_i\} \cup \{k \in S_{g_i}; \mathcal{P}_k \subset \mathcal{ACS}_i\} && i = 1, \ldots, J-1 \\
\mathcal{ACS}_{i+1} &:= \mathcal{ACS}_i \cup \{g_{i+1}\} && i = 1, \ldots, J-1
\end{aligned}
$$

Using the preliminaries presented we can concisely state the algorithm: The algorithm schedules one activity per node of the branch-and-bound tree. An activity is firstly considered for scheduling when all of its predecessors are scheduled. The start time of the activity under consideration is the lowest feasible start time, which (a) is not less than the start time of the activity most recently scheduled, and (b) does

not violate the precedence or resource constraints. In the sequel we will refer to the determination of the lowest feasible start time following (a) and (b) as the strategy (*). Note, employing scheduling strategy (*) reduces the number of schedules to be examined substantially. The correctness of this reduction – compared to the enumeration of all the feasible start times – is proven in [25].

However, if scheduling of the current activity is not feasible then backtracking is performed. On this level the next untested eligible activity is selected. If there is no untested eligible activity left then backtracking to the previous level is performed. At this level the next eligible activity is chosen.

Using the notation displayed in Table 3 the algorithm is formally described in Table 4. Note, in contrast to [25], the basic description includes a simple bounding-rule, the so-called Non-Delayability Rule. It states that an activity that cannot be scheduled on the current level with respect to a given partial schedule cannot be scheduled on higher levels either, if the same schedule is continued, that is, after identifying that an activity cannot be scheduled, backtracking can be performed.

Clearly, the ordering of the eligible set, i.e. the decision which activity to select when, has an influence on the solution time. However, for the present, we assume the eligible sets to be arranged with respect to increasing labels, i.e., activity numbers. Surely, all the priority rules allowing to relabel the activities before the enumeration is started can be implemented in any case.

i	: level index
i^*	: lowest index which produces a time window violation after the recalculation
Y_i	: set of eligible activities on level i
$\hat{N}_i$	: cardinality of the set Y_i
N_i	: index of the element from the eligible set Y_i which is currently under consideration
Y_{iN_i}	: the N_i'th element of the set of eligible activities on level i
$\mathcal{ACS}_i$	: set of activities currently scheduled up to level i
g_i	: activity currently scheduled or under consideration on level i
ST_{g_i} (CT_{g_i})	: start (completion) time of activity g_i scheduled on level i
t_P	: lowest feasible start time of activity g_i with respect to the precedence relations
$Seq_i = [g_1, \ldots, g_i]$	: sequence of activities scheduled on level $j = 1, \ldots, i$
$\mathcal{PS}_i$	: partial schedule induced by sequence Seq_i and employing strategy (*)

Table 3: Notation Used in the B&B-Algorithm

For sake of simplicity we have focused on the minimization of the makespan, although, as described in [25], any regular measure of performance as well as multi-mode problems and other variants can be solved with a slight modification of the scheme.

Step 1: **(Initialization)**

$\mathcal{ACS}_0 := \emptyset$; $g_0 := 0$; $ST_{g_0} := 0$; $CT_{g_0} := 0$; $\mathcal{P}_1 := \{0\}$; $i := 1$; $Y_1 := \{1\}$; $\hat{N}_1 := 1$; $N_1 := 0$;

Step 2: **(Select next untested descendant)**

If $N_i < \hat{N}_i$ then $N_i := N_i + 1$; $g_i := Y_{iN_i}$; goto Step 4;

Step 3: **(One-level backtracking)**

$i := i - 1$; if $i = 0$ then STOP, else remove job g_i from partial schedule; readjust resource arrays and goto Step 2;

Step 4: **(Find feasible start time)**

$t_P := \max\{CT_k; k \in \mathcal{P}_{g_i}\}$; $t^* := \max\{t_P, ST_{g_{i-1}}\}$; determine the **earliest** resource feasible start time $\bar{t}$, $t^* \leq \bar{t} \leq LF_{g_i} - d_{g_i}$, of job g_i; if scheduling is impossible goto Step 2, else set $ST_{g_i} := \bar{t}$; $CT_{g_i} := \bar{t} + d_{g_i}$; $\mathcal{ACS}_i := \mathcal{ACS}_{i-1} \cup \{g_i\}$ and adjust resource arrays;

Step 5: **(Check completeness)**

If $i = J$ then goto Step 7;

Step 6: **(Update the eligible set)**

$i := i + 1$; calculate the new descendant set $Y_i := Y_{i-1} \backslash \{g_{i-1}\} \cup \{k \in \mathcal{S}_{g_{i-1}}; \mathcal{P}_k \subseteq \mathcal{ACS}_{i-1}\}$; $\hat{N}_i := |Y_i|$; $N_i := 0$; goto Step 2;

Step 7: **(Store solution and adjust time bounds)**

Store solution g_j, ST_{g_j}, $\quad j = 1, \ldots, J$;

Set $LS_j := LS_j - (LF_J - CT_J + 1)$, $\quad j = 1, \ldots, J$;

and $LF_j := LF_j - (LF_J - CT_J + 1)$, $\quad j = 1, \ldots, J$;

Step 8: **(Calculate lowest indexed level violating the time window)**

$i^* := \min\{k \in \{1, \ldots, J\}; CT_{g_k} > LF_{g_k}\}$;

Step 9: **(Variable-level backtracking)**

Readjust resources used by jobs g_k, $k = J, \ldots, i^* - 1$;

$i := i^* - 1$; goto Step 2.

Table 4: Minimizing the Project's Makespan

Note, after finding an improved feasible solution the backtrack level in the single-mode case differs from the one of the multi-mode case. In the multi-mode case the level that has to be visited is the lowest indexed level where the completion time of the activity scheduled on this level violates the new bound imposed by the adapted latest finishing time. On this level there might be another mode allowing the activity to be scheduled within the bounds. In the single-mode case the bound violation means, since scheduling strategy (*) is used, that the activity cannot be scheduled on this level, and we can track

another step back.

4.2 The Bounding Rules

In this subsection we will briefly summarize the bounding rules employed to accelerate the basic algorithm. Due to scheduling strategy (*) the (partial) schedule $\mathcal{PS}_i$ related to a sequence of activities $Seq_i = [g_1, \ldots, g_i]$, $i \leq J$, is uniquely determined. However, in the representation of the rules, we sometimes need the starting times as well as the completion times of activities $g_1, \ldots, g_i$, we will denote them by $ST_{g_1}, \ldots, ST_{g_i}$ and $CT_{g_1}, \ldots, CT_{g_i}$, respectively.

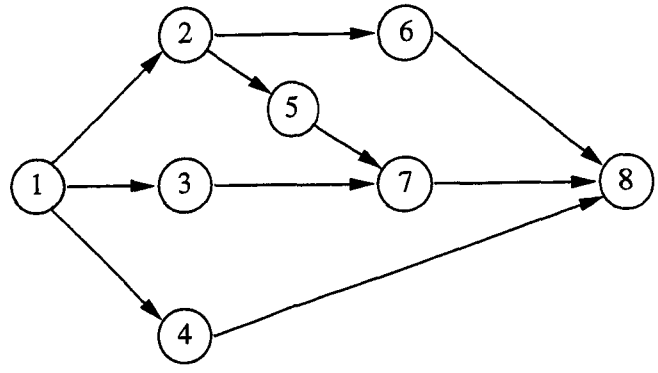

Figure 1: Example Network

For illustrational purposes, we will use the AON representation of a project as given in Figure 1 (cf. [9], p. 179). We assume that two renewable resources with constant availability of $K_{1t} = 2$ and $K_{2t} = 3$, $t = 1, \ldots, \overline{T}$, units per period have to be taken into account. The notation $[j|k_{j1}, k_{j2}]$ is chosen to represent the per-period usages of the resources of an activity j. Note, activity 1 and 8 are dummy activities, i.e., they have a zero duration and do not request any resource. Moreover, in order to simplify the description of the rules, the requests and durations of the remaining activities may vary from instance to instance. Moreover, we assume the eligible set to be examined with respect to increasing labels.

In the multi-mode case we distinguished static and dynamic search tree reduction schemes. The static schemes are preprocessing rules especially designed for the multi-mode problem with nonrenewable resources. For obvious reasons they do not play a role for the RCPSP. Moreover, the Non-Delayability Rule is already incorporated into the basic scheme. It states that an activity that cannot be scheduled within the bounds on the current level cannot be scheduled on later levels, either.

The first rule is the so-called Single-Enumeration Rule. It excludes multiple enumeration of one and the same (partial) schedule. As to be seen from Figure 2, the sequences $\overline{Seq_3} = [1, 2, 3]$ and $Seq_3 = [1, 3, 2]$

produce the same starting time for activities 2 and 3, i.e., $ST_2 = ST_3 = 0$. Therefore Seq_3 which is, due to ordering of the eligible set, analyzed after $\overline{Seq_3}$ needs not be continued.

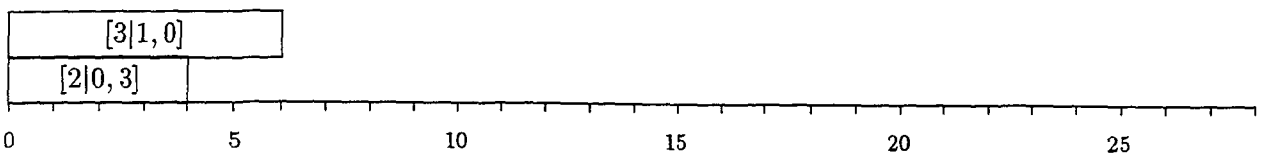

Figure 2: Single Enumeration Rule

The Single Enumeration Rule presented in [24] is already efficient and avoids duplicate enumeration, but it can be further extended and simplified. First, the modification does not require an additional array for storing the start times. Second, further dominance due to feasible left-shifts (cf. [28]) is taken into account. We stress the assumption that the eligible sets are arranged with respect to increasing labels.

Theorem 1 *(Extended Single Enumeration Rule)*
Let $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ *be the currently considered sequence. If (a)* $g_{i+1} < g_i$ *and (b)* $ST_{g_i} = ST_{g_{i+1}}$, *then the current sequence* Seq_{i+1} *is dominated by the (previously evaluated) sequence* $\overline{Seq}_{i+1} = [g_1, \ldots, g_{i-1}, g_{i+1}, g_i,]$.

Proof: Due to numerically labeling of the network, (a) $g_{i+1} < g_i$ implies that the sequence $\overline{Seq}_{i+1} = [g_1, \ldots, g_{i-1}, g_{i+1}, g_i]$ can be scheduled precedence feasibly by scheduling strategy (*). The equation (b) $ST_{g_{i+1}} = ST_{g_i}$ implies that start times $\overline{ST}_{g_{i+1}}$ and $\overline{ST}_{g_i}$ of activity g_{i+1} and g_i in the partial schedule associated with the sequence $\overline{Seq}_{i+1} = [g_1, \ldots, g_{i-1}, g_{i+1}, g_i]$ fulfill $\overline{ST}_{g_{i+1}} \leq \overline{ST}_{g_i} = ST_{g_i}$. That is, the previously evaluated continuations of the sequence $\overline{Seq}_{i+1} = [g_1, \ldots, g_{i-1}, g_{i+1}, g_i]$ dominate the ones of $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$. □

Let use assume that the start times of the three activities 2, 3 and 4 of the network given in Figure 1 are $ST_2 = ST_3 = ST_4$ independently of the sequence they are scheduled in. Applying Theorem 1 shows, that, although only pairwise comparisons are used, the more general concept with k activities having sequence independent starting times $ST_{g_{i+1}} = \cdots = ST_{g_{i+k}}$ is realized. That is, from originally $k!$ continuations to be examined only one has to be selected, that is, $(k! - 1)$ continuations can be saved (cf. [25]).
As already mentioned, the Extended Single Enumeration Rule covers dominance due to feasible left-shifts, producing partial schedules that have been previously evaluated. The Local and the Global

Left-Shift Rule, which we will summarize in the sequel, additionally identify partial schedules that are dominated by schedules to be found later in the enumeration process. The Local and the Global Left-Shift Rule make use of the fact that the set of semi-active and active schedules, respectively, is a dominant set with respect to any regular measure of performance (cf., e.g., [28]).

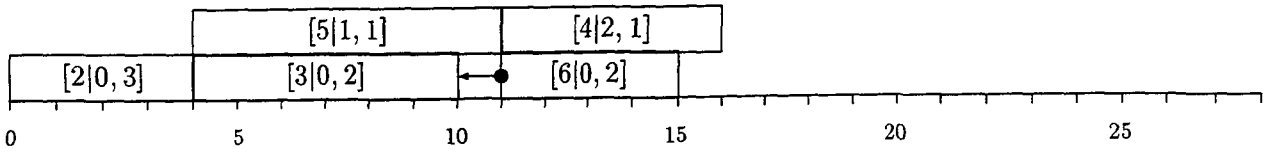

Figure 3: Local Left-Shift Rule

In Figure 3 we see that in the partial schedule associated with the current sequence $Seq_6 = [1, 2, 3, 5, 4, 6]$ activity 6 can be locally left-shifted, i.e., started one period before $ST_6 = 11$, which results from (*), i.e., $ST_{g_i} \leq ST_{g_{i+1}}$, without violating the precedence or resource constraints. Doing so frees resources in period $ST_6 + d_6 = 15$, for scheduling the remaining activities. The rule has been successfully used in, e.g., [5], [29] and is stated in the following theorem:

Theorem 2 *(Local Left-Shift Rule)*
Let $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ be the currently considered sequence. If activity g_{i+1} has the same start time as activity g_i, i.e., $ST_{g_i} = ST_{g_{i+1}}$, and can, by neglecting (a) of scheduling strategy (*), be started at $\overline{ST} = ST_{g_{i+1}} - 1$ then the sequence Seq_{i+1} is dominated by the sequence $\overline{Seq}_{i+1} = [g_1, \ldots, g_{i-1}, g_{i+1}, g_i]$.

Note, since we have considered only a one-period left-shift, the Local Left-Shift Rule does not imply that all the continuations of $Seq_i = [g_1, \ldots, g_i]$ are dominated. The stronger implication is only valid if activity g_{i+1} can be started at a time $\overline{ST}$, $\overline{ST} \leq ST_{g_i} - d_{g_{i+1}}$. We use Figure 4 for illustration, and analyze the sequence $Seq_6 = [1, 2, 3, 5, 6, 4]$. Note, activity 4 can not be locally left-shifted. Since

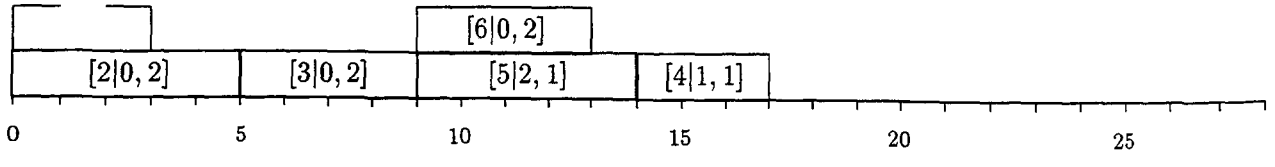

Figure 4: Global Left-Shift Rule

activity 4 can be started at $\overline{ST}_4 = 0$ without violating the constraints, we can free resources in periods t, $t = ST_4+1, \ldots, ST_4+d_4 = 15, \ldots, 17$, as a result of which the availability of the resources for scheduling the remaining activities is increased. Note, whereas in [25] we suggested backtracking to the previous level, we will now strengthen the effect by some additional considerations. Rescheduling an activity on

the previous level, i.e., level 5, does not eliminate the feasibility of the global left-shift of activity 4. For achieving this we have to track further steps back. The level which has to be revisited to eliminate the feasibility of the left-shift of activity 4 is the minimum level where the activity currently scheduled on, has a start time which is at least equal to the completion time obtained from the left-shift, i.e., $\overline{CT}_4 = \overline{ST}_4 + d_4 = 4$. That is, for the instance, we have to track back to level 3 to eliminate feasibility of the left-shift. We include this aspect and summarize as a theorem:

Theorem 3 *(Global Left-Shift Rule)*
Let $Seq_i = [g_1, \ldots, g_i]$ be the sequence currently considered. If, by ignoring condition (a) of scheduling strategy (*), the sequence Seq_i can be feasibly continued by scheduling activity g_{i+1} with a minimal start time $\overline{ST}_{g_{i+1}}$ and related completion time $\overline{CT}_{g_{i+1}}$, such that for a $k \in \{1, \ldots, i\}$

$$\overline{CT}_{g_{i+1}} \leq ST_{g_k}$$

then all the continuations of $Seq_k = [g_1, \ldots, g_k]$ are dominated.

We can learn more from the Global Left-Shift Rule. That is, by recording the minimum completion time CT_{i+1}^{min} of the activities g, $g = Y_{i+1,N}$, $N = 1, \ldots N_{i+1} - 1$, already tested to continue the sequence $Seq_i = [g_1, \ldots, g_i]$, we obtain a bound on the start time $ST_{g_{i+1}}$ of an activity g_{i+1} from Y_{i+1}. If the start time of the currently considered activity g_{i+1} fulfills $ST_{g_{i+1}} \geq CT_{i+1}^{min}$ then a continuation of $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ is dominated by a continuation $Seq_{i+2} = [g_1, \ldots, g_i, g_{i+1}^{min}, g_{i+1}]$ with g_{i+1}^{min} producing the minimum completion time on level $(i+1)$. We capture the following remark:

Remark 1 *(Extended Global-Left Shift Rule)*
Let $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ be the sequence currently considered with $g_{i+1} = Y_{i+1,N_{i+1}}$. Let CT_{i+1}^{min} be the minimum completion time of the activities g, $g = Y_{i+1,N}$, $N = 1, \ldots, N_{i+1} - 1$ in $Seq_{i+1} = [g_1, \ldots, g_i, g]$. If $ST_{g_{i+1}} \geq CT_{i+1}^{min}$, then Seq_{i+1} cannot be continued to an active schedule.

Note, by commonly applying the Local and the (Extended) Global Left-Shift Rule only active schedules are generated. The Local as well as the (Extended) Global-Left Shift Rule can be employed when optimizing any regular measure of performance (cf. [25]).
The next rule we present is derived from the Multi-Mode Cut-Set Rule I presented in [25]. In the single-mode version, on the one hand, it covers portions of the cut-set rule presented by Demeulemeester and Herroelen (cf. [5]), but, on the other hand, it requires only a fraction of the memory used by Demeulemeester and Herroelen.
For a given sequence $Seq_i = [g_1, \ldots, g_i]$ we refer to the set of currently scheduled activities $\mathcal{ACS}_i = \bigcup_{j=1}^{i} \{g_j\}$ as the cut-set associated with sequence Seq_i and denote the maximum completion time as

$CT^{max}(Seq_i)$, i.e., $CT^{max}(Seq_i) = \max_{j=1}^{i}\{ST_{g_j} + d_{g_j}\}$. Using the definition we can now illustrate a multi-mode suitable version of the cut-set rule. We study the partial schedules induced by the sequences $\overline{Seq}_7 = [1,2,3,5,4,6,7]$ and $Seq_7 = [1,2,4,6,3,5,7]$ (cf. Figure 5), respectively. We note, (a) on level 6 both sequences have the same cut-sets, i.e., $\overline{\mathcal{ACS}}_6 = \mathcal{ACS}_6$, and (b) the start time of activity 7 currently considered to continue the sequence Seq_6 fulfills $ST_7 = 13 \geq CT^{max}(\overline{Seq}_6) = 13$, that is, the left-over capacities associated with $Seq_7 = [1,2,4,6,3,5,7]$, in periods $t = 13,\ldots,\overline{T}$, at most match with the ones of $\overline{Seq}_7 = [1,2,3,5,4,6,7]$. Therefore, since scheduling strategy (*) is used, the continuations of Seq_7 are dominated by previously evaluated continuations of $\overline{Seq}_7$. The rule is captured in Theorem 4.

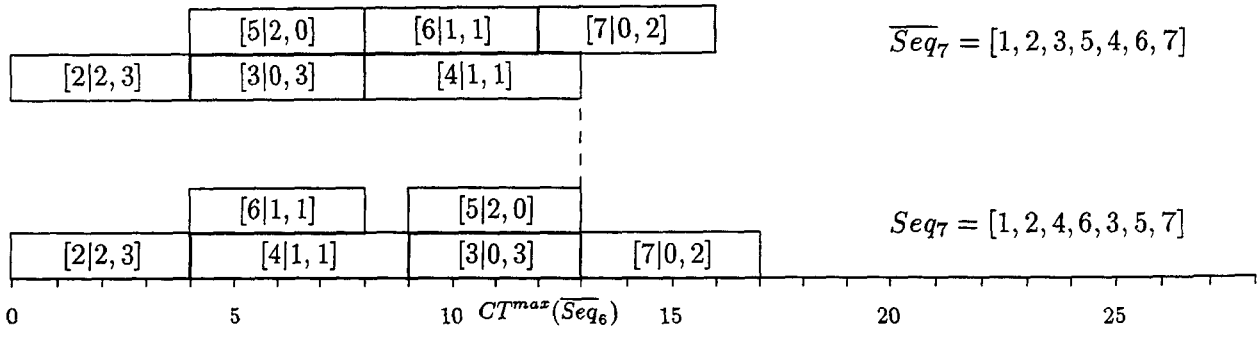

Figure 5: Cut-Set Rule

Theorem 4 *(Cut-Set Rule, Dominated Heads)*
Let $Seq_i = [g_1,\ldots,g_i]$ be the sequence currently under consideration to be continued by feasibly scheduling activity g_{i+1}. If the start time $ST_{g_{i+1}}$ is at least equal to the maximum completion time $CT^{max}(\overline{Seq}_i)$ obtained from a previously evaluated sequence $\overline{Seq}_i = [\overline{g}_1,\ldots,\overline{g}_i]$ with same cut-set, i.e. $\bigcup_{j=1}^{i}\{\overline{g}_j\} = \bigcup_{j=1}^{i}\{g_j\}$, then $Seq_{i+1} = [g_1,\ldots,g_i,g_{i+1}]$ is dominated by $\overline{Seq}_{i+1} = [\overline{g}_1,\ldots,\overline{g}_i,g_{i+1}]$.

The rule can be efficiently realized by coding the cut-sets through integers and storing them in binary level dependent trees. For dealing with other (regular) measures of performance only minor modifications are necessary.

Clearly, our variant of the cut-set rule can be reformulated to the variant implemented by Demeulemeester and Herroelen (cf. [5]): We compare the current sequence Seq_i, which is to be extended by starting activity g_{i+1} at $ST_{g_{i+1}}$, with the sequence $\overline{Seq}_i$ previously evaluated. If (a) $\mathcal{ACS}_i = \overline{\mathcal{ACS}}_i$, and (b) $\overline{CT}_{g_k} \leq CT_{g_k}$ for all $g_k \in \overline{\mathcal{ACS}}_i$ with $\overline{CT}_{g_k} > ST_{g_{i+1}}$, then the sequence $Seq_{i+1} = [g_1,\ldots,g_{i+1}]$ is dominated by a continuation of $\overline{Seq}_i$. An instance is given in Figure 6. It is $\overline{Seq}_i = \overline{Seq}_5 = [1,2,3,4,5]$,

$Seq_i = Seq_5 = [1,2,4,3,5]$, and $g_6 = 6$. That is, $\overline{Seq_6} = [1,2,3,4,5,6]$ dominates $Seq_6 = [1,2,4,3,5,6]$.

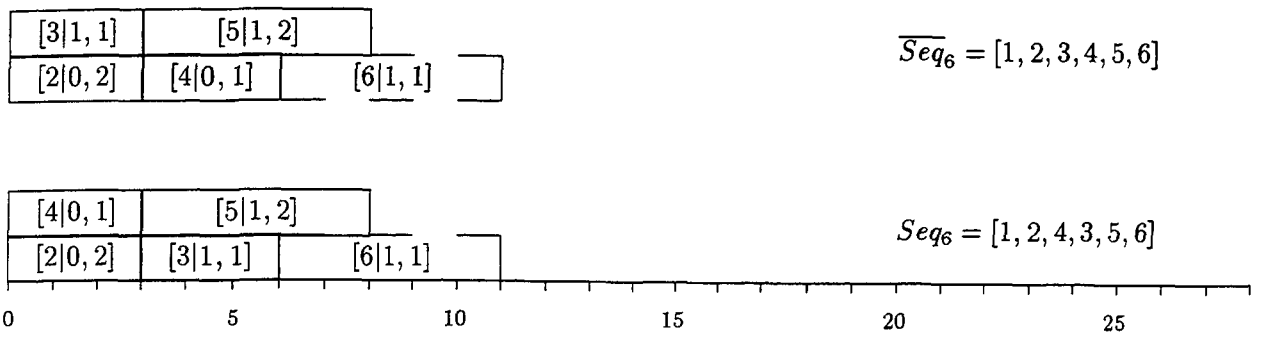

Figure 6: Simple Permutation Rule

Since, for a given sequence, beside the cut-set, additionally those activities that finish after the start time of the finally scheduled activity have to be stored together with their completion times, this formulation requires far more memory (cf. Section 5 and [6]). Consequently, the strengthened effect can only be realized on very well equipped and expensive computers. Even for solving problems with the modest size of 32 activities 24 MB are necessary for full exploitation. Consequently, we stay with our implementation, and enhance our algorithm, instead, by the Simple Permutation Rule that covers other portions of the Demeulemeester/Herroelen variant of the cut-set rule without requiring a substantial increase of memory utilization.

In the lower part of the Gantt chart displayed in Figure 6 we see that activity $g_4 = 3$ finishes before activity g_6 is started, and, moreover, that activity $g_4 = 3$ can be interchanged with the higher labeled activity $g_3 = 4$, which is scheduled on a lower level. Interchanging both activities does not violate the precedence or resource constraints. Taking into account that the eligible sets are ordered with respect to increasing labels, we note that the sequence $\overline{Seq_6} = [1,2,3,4,5,6]$ has been analyzed in an earlier phase of the enumeration, i.e., before $Seq_6 = [1,2,4,3,5,6]$. Since the left-over capacities in periods $t = ST_6 + 1, \ldots, \overline{T}$ of the current sequence Seq_6 at most match with the one of the previously studied sequence $\overline{Seq_6}$, the current sequence is dominated.

Theorem 5 *(Simple Permutation Rule)*

Let $Seq_i = [g_1, \ldots, g_i]$ be the sequence currently considered to be continued by scheduling activity g_{i+1} at $ST_{g_{i+1}}$. If there is an activity g_k, with $k \leq i$ and $CT_{g_k} \leq ST_{g_{i+1}}$ that can be interchanged with an activity g_l with $l < k$ and $g_l > g_k$, such that, g_k starts at ST_{g_l} and g_l finishes at CT_{g_k} then the continuations of the current sequence $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ are dominated by a previously evaluated sequence.

The next rule we present is, to the best of our knowledge, the first one that offers a necessary condition for optimality of a continuation of a partial schedule. We assume the resource availabilities to be constant. We study the sequence $Seq_6 = [1, 2, 3, 4, 5, 6]$ with the partial schedule given in the lower part of Figure 7. We currently consider activity 7 for scheduling. We will denote the largest completion time of the activities already scheduled by $CT^{max}(Seq_6)$ and the second largest completion time by $CT^{\overline{max}}(Seq_6)$, i.e., it is $CT^{max}(Seq_6) = CT_6 = 14$ and $CT^{\overline{max}}(Seq_6) = CT_5 = 12$. The start time of activity 7 is equal $CT^{max}(Seq_6) = 14$.

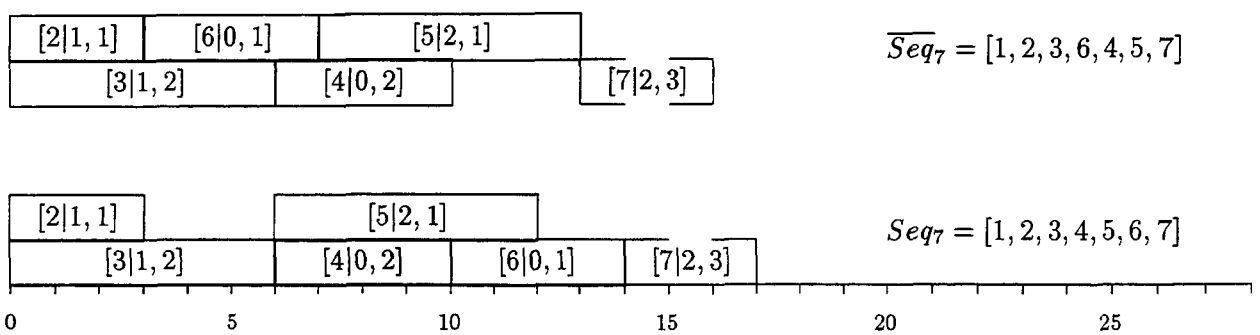

Figure 7: Non-Optimality Rule

Activity 6 as a whole cannot be left-shifted, but 3 periods of it could be left-shifted to start at period $ST = 3$, and, which would leave $\Delta = 1$ period unscheduled. That is, inserting activity 6 at the named position would require a delay of at most $\Delta = 1$ period(s) of activity 4 and 5. On the other hand, if activity 6 is removed from the partial schedule, then activity 7 could be started $\Delta^{max} = CT^{max}(Seq_6) - CT^{\overline{max}}(Seq_6) = 14 - 12 = 2$ periods earlier than in the current partial schedule. That is, as to be seen in the upper part of the graphic, inserting activity 6 at $ST = 3$ would reduce the makespan of the continuations of $Seq_7 = [1, 2, 3, 4, 5, 6, 7]$ by at least $\Delta^{max} - \Delta = 2 - 1 = 1$ period(s). We capture the following theorem:

Theorem 6 *(Non-Optimality-Rule)*

Let $Seq_i = [g_1, \cdots, g_i]$, be the sequence currently considered to be continued by scheduling g_{i+1}. If (a) $ST_{g_{i+1}} = CT^{max}(Seq_i)$, (b) Δ periods of the activity g^{max} which induces the maximum completion time $CT^{max}(Seq_i)$ can be left-shifted, and (c) the difference Δ^{max} between the largest and second largest completion time of the activities already scheduled is larger than the non-left-shiftable portion of activity g^{max}, i.e., $\Delta^{max} = CT^{max}(Seq_i) - CT^{\overline{max}}(Seq_i) > d_{g^{max}} - \Delta$, then a continuation of $Seq_{i+1} = [g_1, \cdots, g_i, g_{i+1}]$ cannot be makespan minimal.

Finally, we have implemented the following variant of the bound LB3 from Mingozzi et. al. (cf. [16]).

The bound is valid when resource availability is constant. In a preprocessing routine we generate a priority list of the set of activities. Each activity g in the list is assigned another list of activities containing all the activities that can be performed simultaneously with activity g without violating the precedence and resource constraints, and which are not placed before activity g in the priority list. During the enumeration at each node of the branch-and-bound tree an activity which is scheduled is eliminated from the priority list, and again added to the list through backtracking. The remaining activities in the priority list are considered from the beginning to the end of the priority list. Starting with $LB = 0$, and no activity marked, the first non-eliminated and non-marked activity is selected and its duration is added to LB. Afterwards the elements from its list are marked. The procedure continues as far as unmarked elements can be found in the priority list. The finally valid LB is then a bound on the minimal time necessary to complete the partial schedule.

Clearly, in general, the quality of the bound obtained depends on the priority rule used to built the list. In our implementation we have built three priority lists and determined the bound LB3 as the maximum of the bounds obtained from all the priority lists. The lists have been determined by (a) using maximum duration as first criterion and size of individual list as tie-breaker (cf. [6]), (b) vice versa and (c) minimum slack with arbitrarily broken ties. Since the quality of the bound obtained from a priority list obviously depends on the cut-set only, we have calculated them only once, when the cut-set is generated for the first time and stored them in the related structure.

5 Computational Results

In this section we present the results of our computational analysis. The algorithm has been coded in GNU C and implemented on a personal computer (80486, 66 MHz, 16 MB) operating under OS/2. The multi-mode version studied in [25] and [26] served as the basis of our implementation. Only minor changes have been performed: E.g., (a) mode-indices have been eliminated, (b) the data structures have been adapted due to the absence of nonrenewable resources, (c) the renewable resources have been merged into global resources (cf. [6]). That is, assuming per-period availability and requests of less than 256 units, the availability of and the requests for four resources can be represented by a 32 bit unsigned integer. Doing so resource profiles can be checked and adapted for four resources simultaneously. (d) The bounding-rules have been enhanced as described in Subsection 4.2. Note, all the rules but the variant of the rule proposed by Mingozzi et al. (cf. [16]) can be fitted to serve in the multi-mode case as well. (e) Assuming constant resource availability, like Demeulemeester and Herroelen, the determination of the earliest precedence and resource feasible start time can be simplified. Due to scheduling strategy

(*) it suffices to determine a single period where the resource availabilities allow scheduling of the current activity to establish resource feasibility within the remaining periods. This change reduces the average computation time of the more general version taking into account time varying resource availability by some ten percent.

The procedure has been tested on the standard benchmark set produced by the project generator ProGen (cf. [15]). The projects consist of 32 activities (including two dummy activities), and 4 renewable resources. The network complexity NC, which is defined as the number of non-redundant arcs per activity is 1.5, 1.8, and 2.1. The resource strength RS as a normalized measure of scarceness of the resources is 0.2, 0.5, 0.7, and 1.0. The resource factor RF reflecting the average number of resources requested by an activity has been set to 0.25, 0.50, 0.75, and 1.00. Ten instances per combination of NC, RS, and RF have been generated. Giving a total of $3 \cdot 4 \cdot 4 \cdot 10 = 480$ instances.

The results are compared with the state-of-the-art algorithm developed by Demeulemeester and Herroelen (cf. [6]), hereafter named DH96. They are displayed in Table 5. The first column of the table

	DH96		Priority Rule													
	Mem.	CPU [sec.]	Mem.	JobNr min	SLK min	LFT min	LST min	EFT min	EST min	DUR min	RU min	RK min	$\|\mathcal{S}\|$ min	RU$\mathcal{S}$ min	$\|\overline{\mathcal{S}}\|$ max	RU$\overline{\mathcal{S}}$ max
466	256 KB	56.46	400 KB	1.08	1.25	1.11	1.15	1.05	0.98	1.21	1.25	1.06	1.17	1.08	1.07	1.17
470	1 MB	17.13	400 KB	1.41	1.65	1.46	1.55	1.36	1.30	1.55	1.60	1.37	1.52	1.37	1.42	1.62
475	4 MB	6.23	400 KB	2.69	3.18	2.88	3.01	2.25	2.29	2.49	2.47	2.50	2.93	2.28	2.83	3.04
478	16 MB	10.44	400 KB	9.78	11.50	8.99	12.71	7.73	11.31	9.49	10.30	9.33	11.76	10.33	10.93	11.90
479	24 MB	12.33	400 KB	16.17	17.29	16.41	18.62	12.85	17.33	14.74	14.64	15.26	18.03	17.84	17.21	18.36

Table 5: Average CPU-Times [sec.] on 480 ProGen Instances

shows the number of problems that could be solved by DH96 within 3600 seconds on a IBM personal computer (PS/2 Model P75, 25 MHz) allowing the memory used to be as much as given in the second column. The average CPU-time to solve these problems is listed in the third column. The fourth column shows the memory requirements of our algorithm. The remaining columns exhibit the average CPU-times for the different priority rules employed to order the sets of eligible activities. We have implemented (1) minimum job number minJobNr, (2) minimum slack minSLK, (3) minimum latest finish time minLFT, (4) minimum latest start time minLST, (5) minimum earliest finish time minEFT, (6) minimum earliest start time minEST,(7) minimum duration minDUR, (8) minimum cumulated resource usage minRU, (9) minimum rank minRK, (10) minimum number of (direct) successors min$|\mathcal{S}|$, (11) minimum cumulated resource usage of (direct) successors minRU$\mathcal{S}$, (12) maximum number of (re-

flexive transitive) successors max$|\overline{\mathcal{S}}|$, (13) maximum cumulated resource maxRU$|\overline{\mathcal{S}}|$ usage of (reflexive transitive) successors. From the table one can observe, the difference between the priority rules are only marginal. The rule that performed best is minimum earliest finish time minEFT. Our algorithm clearly outperforms DH96 when making only modest use of memory, i.e., up to 1 MB. Taking a comparison factor of 2.7 to balance the difference clockpulses (DH96/25 MHz, 66 MHz) of the different machines, our algorithm is as fast as DH96, if we allow DH96 to use 4 MB of memory. If memory use can be excessive, i.e., 16 MB and more, than DH96 is slightly faster. Since, in general, the memory requirements exponentially grow with number of activities the project consists of, memory will become a critical resource. Conservative estimations of memory requirements for DH96 have to be settled at least at 500 MB and for our approach at most at 5 MB for 62-activity projects.

Priority-	Range of CPU-Times [sec.]							
Rule	[0;0.5]	(0.5-1]	(1-5]	(5-10]	(10-100]	(100-500]	(500-3600]	> 3600
minJobNr	375	34	29	15	20	3	3	1
minSLK	366	35	39	10	22	3	4	1
minLFT	379	30	33	7	24	3	3	1
minLST	380	27	34	9	22	3	4	1
minEFT	379	25	36	10	23	2	4	1
minEST	382	26	33	10	22	2	4	1
minDUR	371	27	39	11	26	1	4	1
minRU	365	31	40	10	28	1	4	1
minRK	378	29	33	11	22	3	3	1
min$\|\mathcal{S}\|$	372	36	28	11	25	3	4	1
minRU$\mathcal{S}$	375	27	38	11	23	1	3	2
max$\|\overline{\mathcal{S}}\|$	375	31	35	9	22	4	3	1
maxRU$\overline{\mathcal{S}}$	374	28	36	12	22	4	3	1

Table 6: Frequency Distribution of CPU-Times on 480 ProGen Instances (80486, 66 MHz)

The heuristic capabilities of our algorithm have been studied too. Table 6 shows the frequency distribution of solution times of the exact algorithm. Table 7 reveals the capabilities of the truncated exact method in more detail. For the evaluation the quality of the solution after an allotted CPU-time of 0.2, 0.5, 1, 5 and 10 seconds has bee studied. We recorded the number of instances for which an optimal

solution has been found within the time limit (#opt), the number of instances for which no solution could be found within the time limit (#n. sol.), and the average percentage deviation of the best solution found from the optimal makespan ($\overline{\Delta}$ [%]).

		Priority Rule												
		JobNr	SLK	LFT	LST	EFT	EST	DUR	RU	RK	$\lvert\mathcal{S}\rvert$	RU$\lvert\mathcal{S}\rvert$	$\lvert\overline{\mathcal{S}}\rvert$	RU$\lvert\overline{\mathcal{S}}\rvert$
CPU-Time		min	min	min	min	min	min	min	min	min	min	min	max	max
0.2 sec.	#opt.	335	343	351	356	333	351	321	314	339	330	340	344	333
	#n. sol.	0	1	1	1	0	0	0	0	0	0	1	2	2
	$\overline{\Delta}$ [%]	1.94	2.01	1.71	1.69	2.24	1.60	3.09	3.51	1.96	2.62	2.33	1.71	2.18
0.5 sec.	#opt.	400	394	399	401	392	408	393	382	403	395	398	407	401
	# n. sol.	0	1	0	0	0	0	0	0	0	0	0	1	0
	$\overline{\Delta}$ [%]	0.94	0.98	0.95	0.94	1.17	0.85	1.53	1.68	0.88	1.24	1.16	0.85	1.03
1 sec.	#opt.	422	423	426	427	418	425	413	409	426	422	421	427	418
	# n. sol.	0	0	0	0	0	0	0	0	0	0	0	0	0
	$\overline{\Delta}$ [%]	0.65	0.65	0.63	0.65	0.78	0.56	0.99	1.11	0.58	0.80	0.71	0.59	0.72
5 sec.	#opt.	452	450	450	449	451	453	449	446	453	447	452	451	449
	# n. sol.	0	0	0	0	0	0	0	0	0	0	0	0	0
	$\overline{\Delta}$[%]	0.30	0.31	0.30	0.32	0.31	0.28	0.43	0.45	0.28	0.36	0.28	0.29	0.30
10 sec.	# opt.	462	458	457	455	462	461	459	456	462	459	459	459	460
	# n. sol.	0	0	0	0	0	0	0	0	0	0	0	0	0
	$\overline{\Delta}$ [%]	0.21	0.22	0.23	0.24	0.19	0.20	0.31	0.30	0.19	0.22	0.21	0.20	0.22

Table 7: Truncated Exact Method – Quality of Solution vs. CPU-Time on 480 ProGen Instances (80486, 66 MHz)

Within the time limit of 0.2 and 0.5 seconds the priority rules minSLK, minLFT, minLST, minRU$\mathcal{S}$, max$|\overline{\mathcal{S}}|$, and maxRU$|\overline{\mathcal{S}}|$ could not find a feasible solution for some of the problems. This is mainly reasoned by the use of the shift-rules and the non-optimality rule which can exclude branches from further continuation due to dominance without having found any complete schedule. Especially the minSLK rule produces in the beginning of the enumeration schedules of low quality. Due to the serial scheduling strategy it first schedules the critical activities and later on the non-critical. This leaves available capacity in early periods unused and induces feasibility of left-shift when non-critical activities are considered, or excessive use of resources by precedence independent activities at the end. Therefore,

the results seem to confirm the evaluation of serial and parallel single- and multi-pass priority rule based heuristics from Kolisch (cf. [13]). Kolisch found out that minSLK performs bad within in a serial approach and good in a parallel one. Note, since the procedure developed by Demeulemeester and Herroelen (cf. [5], [6]) can be considered as a scheme following parallel strategy in contrast to the one presented here using a serial strategy, there is a fundamental difference in the effect of the minSLK rule. However, the differences of the performance of the priority rules are reduced the more time the algorithm is allotted. Moreover, alike for the exact approaches, a comparison of the truncated exact method employing, e.g., the minEST rule, with the truncated version of DH96 reveals the quality of the results. Although our algorithm requires only 400 KB memory, compared to 24 MB used by DH96, the average deviations are nearly identical. On a personal computer (IBM PS/2 Model P75, 80486dx, 25 MHz) the average deviation of DH96 has been 0.84% (0.62%, 0.33%, 0.12%) for an allotted CPU-time of 0.5 (1.0, 5.0, 30.0) seconds.

The quality of the results of the truncated exact method is additionally emphasized by a comparison with the heuristics, e.g., from Kolisch/Drexl and Naphade et al. developed for the RCPSP.

Kolisch/Drexl (cf. [14]) tested their parameterized regret-based sampling algorithm on those resource-constrained projects, i.e., with $RS < 1$, the optimal solution of which are known from the analysis in [15]. Using a sample size of 500 (100, 10) requiring 3.09 (0.69, 0.11) seconds of CPU-time on a personal computer (80486dx, 60 MHz) the problem instances have been solved with an average deviation of 0.71% (1.22%, 2.53%) from optimum.

Naphade et al. (cf. [17]) made use of the optimal makespans found during 3600 seconds by DH92 (cf. [5]) within the evaluation of ProGen (cf. [15]). Their local search method that builds up on their ideas for the job shop problem determines an optimal solution for 371 of the 428 problem instances optimally solved by DH92 and achieves an average deviation of 0.56% within 3.33 seconds on a workstation (IBM RISC 340-400 M).

6 Conclusions

We have presented an algorithm for the resource-constrained project scheduling problem. The algorithm has been tested on the standard benchmark set generated by ProGen (cf. [15]). The computational results show that, beside of the theoretical benefits (1) ease of description, (2) ease of implementation, and (3) ease of generalization, practical advantages as (1) reasonable performance and (2) low memory requirements make its use favourable especially when attacking larger problems or variants of the resource-constrained project scheduling problem (cf., e.g., [25], [26]).

The approach can compete with the best solution procedures currently available. The algorithm uses far less memory than the state-of-the-art procedure DH96, i.e., 400 KB versus 24 MB, for solving the standard benchmark set with projects consisting of 32 activities. It definitely outperforms DH96 if both approaches are allowed to make limited use of memory. Since, in general, the memory requirements exponentially grow with number of activities the project consists of, memory will become a critical resource. Conservative estimates of memory requirements for DH96 approach have to be settled at least at 500 MB and for our approach at most at 5 MB for 62-activity projects. Therefore, we conjecture that the use of our algorithm will be especially beneficial when dealing with larger problems. Moreover, since comparison pruning is nearly exhausted in the approach by Demeulemeester and Herroelen, we believe that our algorithm will gain more by the development of new dominance rules and bounds.

Acknowledgements: I would like to thank Andreas Drexl and Sönke Hartmann for helpful comments and suggestions.

References

[1] BARTUSCH, M.; R.H. MÖHRING AND F.J. RADERMACHER (1988): Scheduling project networks with resource constraints and time windows. Annals of Operations Research, Vol. 16, pp. 201-240.

[2] BRUCKER, P.; B. JURISCH AND A. KRÄMER (1994): The job-shop problem and immediate selection. Annals of Operations Research, Vol. 50, pp. 73-114.

[3] BRUCKER, P.; A. SCHOO AND O. THIELE (1996): A branch & bound algorithm for the resource-constrained project scheduling problem. Research Report, No. 178, Department of Mathematics and Computer Sciences, Osnabrück University, Germany.

[4] CHRISTOFIDES, N.; R. ALVAREZ-VALDES AND J.M. TAMARIT (1987): Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research, Vol. 29, pp. 262-273.

[5] DEMEULEMEESTER, E. AND W. HERROELEN (1992): A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science, Vol. 38, pp. 1803-1818.

[6] DEMEULEMEESTER, E. AND W. HERROELEN (1996): New benchmark results for the resource-constrained project scheduling problem. Management Science, to appear.

[7] DREXL, A. (1991): Scheduling of project networks by job assignment. Management Science, Vol. 37, pp. 1590-1602.

[8] DREXL, A. AND J. GRÜNEWALD (1993): Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions, Vol. 25, No. 5, pp. 74-81.

[9] ELMAGHRABY, S.E. (1977): Activity networks: Project planning and control by network models. Wiley, New York.

[10] GAREY, M.R. AND D.S. JOHNSON (1979): Computers and intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA.

[11] HARTMANN, S. AND A. DREXL (1997): Project scheduling with multiple modes: A comparison of exact algorithms. Manuskripte aus den Instituten für Betriebswirtschaftslehre, Kiel, Germany, in preparation.

[12] JOHNSON, T.J.R. (1967): An algorithm for the resource-constrained project scheduling problem. PhD Dissertation, Massachusets Institute of Technology, USA.

[13] KOLISCH, R. (1996): Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research, Vol. 90, pp. 320-333.

[14] KOLISCH, R. AND A. DREXL (1996): Adaptive search for solving hard project scheduling problems. Naval Research Logistics, Vol. 43, pp. 23-40.

[15] KOLISCH, R.; A. SPRECHER AND A. DREXL (1995): Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, Vol. 41, pp. 1693-1703.

[16] MINGOZZI, A.; V. MANIEZZO; S. RICCIARDELLI AND L. BIANCO (1996): An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation. Management Science, to appear.

[17] NAPHADE, K.S.; S.D. WU AND R.H. STORER (1995): Problem space search algorithms for the resource-constrained project scheduling problem. Research Report, Department of Industrial & Manufacturing Systems Engineering, Lehigh University.

[18] PATTERSON, J.H. (1984): A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. Management Science, Vol. 30, pp. 854-867.

[19] PATTERSON, J.H.; R. SLOWINSKI; F.B. TALBOT AND J. WEGLARZ (1989): An algorithm for a general class of precedence and resource constrained scheduling problems. In: Slowinski, R. and J. Weglarz (Eds.): Advances in project scheduling. Elsevier, Amsterdam, pp. 3-28.

[20] PATTERSON, J.H.; R. SLOWINSKI; F.B. TALBOT AND J. WEGLARZ (1990): Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. European Journal of Operational Research, Vol. 49, pp. 68-79.

[21] RADERMACHER, F.J. (1985/86): Scheduling of project networks. Annals of Operations Research, Vol. 4, pp. 227-252.

[22] SLOWINSKI, R. (1989): Multiobjective project scheduling under multiple-category resource constraints. In: Slowinski, R. and J. Weglarz (Eds.): Advances in project scheduling. Elsevier, Amsterdam, pp. 151-167.

[23] SLOWINSKI, R.; B. SONIEWICKI AND J. WEGLARZ (1994): DSS for multiobjective project scheduling. European Journal of Operational Research, Vol 79, pp. 220-229.

[24] SPRECHER, A. (1994): Resource-constrained project scheduling: Exact methods for the multi-mode case. Lecture Notes in Economics and Mathematical Systems, No. 409. Springer, Berlin.

[25] SPRECHER, A. AND A. DREXL (1996a): Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part I: Theory. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 385, Kiel, Germany.

[26] SPRECHER, A. AND A. DREXL (1996b): Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part II: Computation. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 386, Kiel, Germany.

[27] SPRECHER, A.; S. HARTMANN AND A. DREXL (1996): An exact algorithm for project scheduling with multiple modes. OR Spektrum, to appear.

[28] SPRECHER, A.; R. KOLISCH AND A. DREXL (1995): Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research, Vol. 80 , pp. 94-102.

[29] STINSON, J.P.; E.W. DAVIS AND B.M. KHUMAWALA (1978): Multiple resource-constrained scheduling using branch and bound. AIIE Transactions, Vol. 10, pp. 252-259.

[30] TALBOT, F.B. (1982): Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science, Vol. 28, pp. 1197-1210.

[31] TALBOT, F.B. AND J.H. PATTERSON (1978): An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. Management Science, Vol. 24, pp. 1163-1174.

[32] WEGLARZ, J. (1979): Project scheduling with discrete and continuous resources. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, pp. 644-650.

[33] WEGLARZ, J. (1980): On certain models of resource allocation problems. Kybernetics, Vol. 9, pp. 61-66.