Manuskripte

aus den

Instituten für Betriebswirtschaftslehre

der Universität Kiel

No. 425 – revised

Scheduling Resource-Constrained Projects Competitively at Modest Memory Requirements [1]

Arno Sprecher

December 1996, revised November 1997

©Do not copy, publish or distribute without authors' permission.

Arno Sprecher, Christian-Albrechts-Universität zu Kiel, Institut für Betriebswirtschaftslehre, Lehrstuhl für Produktion und Logistik, 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

We consider the resource-constrained project scheduling problem. The purpose of this paper is to direct the focus to a branch-and-bound concept that can, by simple adaptations, operate on a wide range of problem settings. The general approach can, e.g., deal with multi-mode problems, resource availability varying with time, and a wide range of objectives. Even the simple assembly line balancing problem of type-1 can be competitively approached with some modifications.

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., 256 KB versus 24 MB, for solving the standard benchmark set with projects consisting of 32 activities within comparable time. If both approaches are allowed to make limited use of memory, i.e. 256 KB, then more than 97% of the benchmark instances can be solved within fractions of the time required by the current state-of-the-art procedure. The truncated version of our algorithm achieves at 256 KB approximately the results of the truncated version of the state-of-the-art approach at 24 MB. Since, in general, the memory requirements exponentially grow with the number of activities the project consists of, memory will become a critical resource, and the strategy to access previously stored information will gain fundamental importance, when solving larger projects.

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

1 Introduction

In the early beginnings of project scheduling the Critical Path Method (CPM) and the Metra Potential Method (MPM) have been developed to support the project scheduler 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. Limitations of the resources required to execute the activities were not taken into account.

Since the limitation of the resource availability cannot be relaxed in the major part of business applications the research community has answered the more realistic assumptions of the resources' limitation by intensive research. The resource-constrained project scheduling problem (RCPSP) is known as an NP-hard problem (cf. [6]). 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. [3], [4]; DH92 and DH97 for short), feasible completion times (cf. [19]), feasible extensions (cf. [17]), feasible posets (cf. [11]), and feasible subsets (cf. [9]), in order to find an optimal, i.e. makespan minimal, solution. The currently most advanced procedure is DH97 which enhances their earlier work DH92 by a variant of a bound introduced by Mingozzi et al. (cf. [9]) and fully exploits nowadays available 32-bit architectures of personal computers. The procedure has been tested on a set of benchmark problems generated by ProGen (cf. [8]). The projects consist of 32 activities (including two dummy activities) and 4 resources. The CPU-time for solving 479 of 480 instances on a personal computer

(80486, 25 MHz, 32 MB) under Windows NT amounts on average to 12.33 seconds at the cost of 24 MB memory used. The remaining instance has been solved within three hours (9,515 sec. employing the MPM-bound, 10,261 sec. employing a modification of the Mingozzi et al. bound LB3 (cf.[9])).

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 modeled. The problem derived is the multi-mode resource-constrained project scheduling problem (MRCPSP), which is commonly considered with makespan minimization as objective (cf. [18]).

In this paper we will return to a procedure originally developed for the single-mode problem (cf. [19]), then generalized to the multi-mode case (cf. [18]), and substantially simplified to the precedence tree guided scheme (cf. [10]). Later on, Sprecher and Drexl (cf. [12], [15]) employed the precedence tree to form the currently most simple, general, and powerful algorithm for the multi-mode resource-constrained project scheduling problem. Compared with a revised (cf. [12]) version of [10], the size of the problems that can be solved to optimality has been nearly doubled.

Beyond its application in resource-constrained project scheduling the basic branch-and-bound concept can be employed in related areas as well. The simple assembly line balancing problem of type-1 can be solved competitively with some problem specific adaptations (cf. [13]).

We study the application of the general branch-and-bound concept to the RCPSP, we introduce, extend, simplify and accelerate concepts to reduce the effort of enumeration.

We proceed as follows: In Section 2 we describe the problem more precisely. In Section 3 we present the algorithm. The basic algorithm is summarized in Subsection 3.1. The search tree reduction is discussed in Subsection 3.2. The elements to increase efficiency are portrayed in Subsection 3.3 and Subsection 3.4. In Section 4 we reveal our computational results. The algorithm is compared with the ones of Demeulemeester and Herroelen (cf. [4]), Mingozzi et al. (cf. [9]), and Brucker et al. (cf. [1]). In Section 5 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, $1 = 2, \ldots, J$, may not be started before all its predecessors h, $h \in \mathcal{P}_j$, are finished. The set of successors of an activity j is abbreviated to $\mathcal{S}_j$, $j = 1, \ldots, J$. 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. Without

loss of generality, 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 resources. Given a horizon, that is, an upper bound $\overline{T}$ on the project's makespan, K_r units of resource r, $r \in R$, are available in each period t, $t = 1, \ldots, \overline{T}$. Performing an activity j, $j = 1, \ldots, J$, requires k_{jr} units of resource r, $r \in R$, each period activity j is in process. The objective is to find a makespan minimal schedule among the precedence and resource feasible ones.

Presuming feasibility, 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. 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 bounds can be used by a branch-and-bound algorithm to speed up convergence.

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., [12], pp. 10, [13]). Thus, the problem is a member of the class of NP-hard problems (cf. [6]).

Moreover, the model can be easily generalized to include different modes to perform the activities, time--varying requests and generalized temporal constraints. Additionally, beside the minimization of the makespan other objectives can be modeled as well (cf., e.g.,[12], [15]).

3 The Branch-and-Bound Algorithm

In this section we describe the algorithm for solving the RCPSP. The basic algorithm is summarized in Subsection 3.1. The search tree reduction is discussed in Subsection 3.2. The elements to increase efficiency are portrayed in Subsection 3.3 and Subsection 3.4.

3.1 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. [10]). 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 [3] and [17], an activity is called eligible, if all its predecessors are scheduled. 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 g_j, $j = 1, \ldots, i$, associated with the nodes, we obtain the set of eligible activities on level i, namely Y_i, as follows: $Y_1 := \{g_1\} = \{1\}$, $\mathcal{ACS}_1 := \{g_1\} = \{1\}$, $Y_{i+1} := Y_i \backslash \{g_i\} \cup \{k \in \mathcal{S}_{g_i}; \mathcal{P}_k \subseteq \mathcal{ACS}_i\}$, $i = 1, \ldots, J-1$, $\mathcal{ACS}_{i+1} := \mathcal{ACS}_i \cup \{g_{i+1}\}$, $i = 1, \ldots, J-1$. The

number of elements of the eligible set Y_i is denoted by $\hat{N}_i$, the number of the element currently selected is N_i, i.e., it is $g_i = Y_{i,N_i}$.

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 ST_{g_i} of activity g_i considered for scheduling on level i is the lowest feasible start time, which (a) is not less than the start time of the activity most recently scheduled, (b) does not violate the precedence or resource constraints, and (c) does not exceed the latest start time LS_{g_i}. In the sequel we will refer to the determination of the lowest feasible start time following (a), (b) and (c) 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 [12].

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.

After finding an improved solution with makespan B^*, the bounds imposed by latest start times LS_i (finish times LF_j) of activities j, are reduced to $LS_j - (LF_J - B^* + 1)$ $(LF_j - (LF_J - B^* + 1))$, $j = 1, \ldots, J$. Denoting the completion time of activity g_j through CT_{g_j}, $j = 1, \ldots, J$, the variable backtracking level is then, via $i^* := \min\{k \in \{1, \ldots, J\}; CT_{g_k} > LF_{g_k}\}$, defined by $i := i^* - 1$.

Note, the variable 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 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. Furthermore, the single-mode algorithm terminates if level $i = 0$ is reached via backtracking.

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.

3.2 The Bounding Rules

In this subsection we summarize, extend, simplify and accelerate concepts to reduce the effort for enumeration. Due to scheduling strategy (*) the (partial) schedule $\mathcal{PS}_i$ related to a sequence of activities $Seq_i = [g_1, \ldots, g_i]$, $i \leq J$, to be scheduled consecutively is uniquely determined.

The **Extended and Simplified Single Enumeration Rule** excludes multiple enumeration of one and the same (partial) schedule induced by different sequences, and leaves out schedules from continuation that are dominated due to feasible left-shifts. It extends and simplifies the Single Enumeration Rule presented in [15]. The earlier version of the rule required a multi-dimensional array to verify the assumptions via previously

stored information. In the extended and simplified rule the necessity of storing and accessing to previously stored information has been eliminated.

Theorem 1 *(Extended and Simplified 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)* $ST_{g_{i+1}} = ST_{g_i}$*, 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 $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$. □

The following methods of search tree reduction base on the concept of left-shift of activities (cf., e.g., [16]). A left-shift of an activity derives a feasible (partial) schedule from a given feasible (partial) schedule by reducing the start time of the activity while preserving the start times of the remaining activities. One distinguishes a one-period, a local and a global left-shift of an activity of a (partial) schedule. Given a feasible (partial) schedule. A one-period left-shift derives a feasible (partial) schedule through reducing the start time of one activity by one period. A local left-shift is a left-shift that consists of a series of one-period left-shift of one and the same activity. Schedules in which no activity can be locally left-shifted are called semi-active. A global left-shift is a left-shift that cannot be obtained by a series of one-period left-shifts of one and the same activity. Schedules in which no activity can be locally or globally left-shifted are called active. The set of semi-active and the set of active schedules built dominant sets with respect to any regular measure of performance, like, e.g., the makespan minimization (cf., e.g., [16]).

The **Local Left-Shift Rule** reduces the enumeration to semi-active schedules. If the start time $ST_{g_{i+1}}$ of an activity g_{i+1}, selected to continue a given sequence $Seq_i = [g_1, \ldots, g_i]$ on level $(i+1)$, can be reduced to $\overline{ST}_{g_{i+1}} = ST_{g_{i+1}} - 1$ without violating the precedence or resource constraints, then the current selection can be skipped. An example is given in Figure 2 of the Appendix.

The concept has been successfully employed in, e.g., [3], [9], and [17]. However, there are fundamental differences in the effect; the implementation provided in [3] does not reduce the enumeration to semi-active schedules (cf. [14]).

The **Extended Global Left-Shift Rule**, commonly applied with the Local Left-Shift Rule, reduces the enumeration to active schedules. If the start time $ST_{g_{i+1}}$ of an activity g_{i+1}, selected to continue a given sequence $Seq_i = [g_1, \ldots, g_i]$ on level $(i+1)$, can be reduced to $\overline{ST}_{g_{i+1}} \leq ST_{g_i} - d_{g_{i+1}}$ without violating the precedence or resource constraints, then all the continuations of Seq_i can be skipped. An example is given in Figure 3 of the Appendix.

Note, in contrast to the Local Left-Shift Rule, where we consider only a one-period left-shift, the Global Left-Shift Rule has a much stronger effect. The Global Left-Shift Rule induces backtracking. In contrast to

[15], where we suggested to track a single level back, we improve the effect and now return to a level k with $k = \min\{j = 1, \ldots, i; \overline{ST}_{g_{i+1}} + d_{g_{i+1}} = \overline{CT}_{g_{i+1}} \leq ST_{g_j}\}$.

Note, the left-shifts that are considered in the Global Left-Shift Rule are not necessarily global ones. If we have, $ST_{g_i} = ST_{g_{i+1}}$ and $\overline{ST}_{g_{i+1}} = ST_{g_i} - d_{g_{i+1}}$ then we obtain the partial schedule by a local left-shift. However, the effect is stronger if considered as feasible left-shift that induces backtracking.

As easily to be seen, the Extended and Simplified Single Enumeration Rule covers dominance resulting from feasible left-shifts. The partial schedules obtained as a result of the shift have been previously evaluated. The Local and the Global Left-Shift Rule, additionally, identify partial schedules that are dominated by schedules to be found later in the enumeration process.

We can learn more from the Local and 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 of 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 activity g_{i+1}^{min} producing the minimum completion time on level $(i+1)$.

The **Contraction Rule** employs previously determined start times obtained from (a) scheduling in accordance with strategy (*), (b) feasible local left-shifts, and (c) feasible global left-shifts, to determine an upper bound on the start time of an activity to be scheduled. The bound is used to avoid – at least partly – feasibility of left-shifts on later levels. We consider a sequence $Seq_i = [g_1, \ldots, g_i]$ considered to be continued on level $(i+1)$. When incrementing the level from i to $(i+1)$ we initialize the bound BST_{i+1} of the start time of an activity to be scheduled on level $(i+1)$ by a large integer, e.g., MaxInt. The bound is reduced with the trials of scheduling an activity g_{i+1} on level $(i+1)$ to $BST_{i+1} := \min\{BST_{i+1}, CT_{g_{i+1}}\}$. If activity g_{i+1} can be locally left-shifted to start at $\overline{ST}_{g_{i+1}}$ (finish at $\overline{CT}_{g_{i+1}}$), then bound can be reduced to $BST_{i+1} := \min\{BST_{i+1}, \overline{CT}_{g_{i+1}}\}$. If activity g_{i+1} can be globally left-shifted to start at $\overline{ST}_{g_{i+1}}$ (finish at $\overline{CT}_{g_{i+1}}$) then bound can be reduced to $BST_k := \min\{BST_k, \overline{CT}_{g_{i+1}}\}$, where level k is the level to be revisited after detecting feasibility of the global left-shift. An example for the bound determination by the Contraction Rule is obtained through the example of the local left-shift displayed in Figure 2, and the example of the global left-shift displayed in Figure 3 of the Appendix. In the examples the bound of the start time can be reduced to $BST_6 = 14$ and $BST_3 = 3$, respectively. If, now, the start time $ST_{g_{i+1}}$ of an activity g_{i+1} to be scheduled on a certain level $(i+1)$ fulfills $ST_{g_{i+1}} \geq BST_{i+1}$, then activity g_{i+1} can be skipped on level $(i+1)$. Note, the rule can be employed in the multi-mode case only if no nonrenewable resources have to be considered.

The **Set-Based Dominance** compares the compactness of the current partial schedule with the compactness of a schedule previously studied. It relies on a multi-mode suitable concept that detects dominated heads of partial schedules (cf. [15]), and in the single-mode case covers portions of the Dominance Pruning suggested by Stinson et al. (cf. [17]), the Network Cuts studied by Talbot and Patterson (cf. [19]) and the Cutset Dominance Rule employed by DH92 and DH97. Mathematically more appropriate would be the name "ideal"-based dominance. However, to avoid excessive use of mathematical terms and to enunciate the

difference to the Cutset Dominance we use Set-Based Dominance.

Let $Seq_i = [g_1, \ldots, g_i]$ be the sequence currently considered to be continued, and $\overline{Seq}_i = [\overline{g}_1, \ldots, \overline{g}_i]$ be a sequence previously studied with identical sets of scheduled activities, i.e., $\bigcup_{j=1}^{i}\{g_j\} = \mathcal{ACS}_i = \overline{\mathcal{ACS}}_i = \bigcup_{j=1}^{i}\{\overline{g}_j\}$. If the start time $ST_{g_{i+1}}$ of activity g_{i+1} currently considered to continue the sequence Seq_i fulfills $ST_{g_{i+1}} \geq CT^{max}(\overline{Seq}_i) = \max_{j=1}^{i}\{\overline{CT}_{\overline{g}_i} = \overline{ST}_{\overline{g}_j} + d_{\overline{g}_j}\}$, then $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ is dominated by the previously evaluated sequence $\overline{Seq}_{i+1} = [\overline{g}_1, \ldots, \overline{g}_i, g_{i+1}]$. An example is given in Figure 4 of the Appendix. Clearly, the Set-Based Dominance Rule can be reformulated to realize the Cutset Dominance Rule by DH92. However, the strengthened effect can only be achieved at the cost of excessive use of memory (cf. Subsection 3.4). Therefore, we stay with our implementation, and instead, enhance our algorithm by the Simple Permutation Rule.

The **Simple Permutation Rule** covers further portions of the Cutset Dominance Rule by DH without substantial increase of memory requirements. Let $Seq_i = [g_1, \ldots, g_i]$ be the sequence currently considered to be continued by scheduling activity g_{i+1} at $ST_{g_{i+1}}$ on level $(i+1)$. If there is an activity g_k, with $k \leq i$ and $CT_{g_k} \leq ST_{g_{i+1}}$ that can be feasibly 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. An example for the Simple Permutation Rule is given in Figure 5 of the Appendix. However, to preserve optimality, it is necessary to consider combinations with the Local Left-Shift and the Set-Based Dominance Rule, and adjust the stand-alone implementation accordingly.

The **Non-Optimality Rule** is, to the best of our knowledge, the first one that offers a necessary condition for optimality of a continuation of a partial schedule. Let $Seq_i = [g_1, \ldots, g_i]$, be the sequence currently considered to be continued by scheduling g_{i+1}. If (a) $ST_{g_{i+1}} = CT^{max}(Seq_i)$, (b) $\Delta_{g^{max}}$ periods of the activity g^{max} inducing the maximum completion time $CT^{max}(Seq_i)$ can be feasibly left-shifted, and (c) the difference Δ^{max} between the largest completion time $CT^{max}(Seq_i)$ and second largest completion time $CT^{\overline{max}}(Seq_i)$ of the activities already scheduled is larger than the non-left-shiftable portion of activity g^{max}, i.e., $\Delta^{max} = CT^{max}(Seq_i) - CT^{\overline{max}}(Seq_i) > d_{g^{max}} - \Delta_{g^{max}}$, then a continuation of $Seq_{i+1} = [g_1, \ldots, g_i, g_{i+1}]$ cannot be makespan minimal. The maximal left-shiftable portion $\Delta_{g^{max}}$ can be obtained as a by-product of the Extended Global Left-Shift Rule. An example for the Non-Optimality Rule is given in Figure 6 of the Appendix.

3.3 Bound Computation and Application

In this subsection we present a variant of the bound LB3 from Mingozzi et. al. (cf. [9]). In a preprocessing routine we generate a priority list of the set of activities. Each activity g in the list is assigned an individual 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. The activity is marked. Afterwards the elements from its individual list are marked in the priority list. The procedure continues as far as unmarked elements can be found in the priority list. The finally valid LB is then a lower bound of the minimal time necessary to complete the partial schedule.

Clearly, in general, the quality of the bound obtained depend on the priority rule used to built the list. In our implementation we have built three priority lists and determined the bound LB3Mod 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 non-decreasing length of individual list as tie-breaker, (b) vice versa, and (c) first listing the activities of a critical path, and, second, appending the remaining activities; the latter ones are arranged with respect to non-decreasing length of individual lists. Taking the maximum of the bounds related to the priority lists the bound is at least as good as the MPM-bound.

Since the bounds obtained obviously depend on the set of scheduled activities only, we have to calculate them only once, when the set is generated for the first time and it is intended to store it together with the set of scheduled activities and the maximum completion time (cf. Subsection 3.4). We contrast our implementation with the ones of Mingozzi et al. (cf. [9]) and DH97:

Mingozzi et al. use a circular list, which is in some respect equivalent to using J priority lists. The priority list of activities is obtained through extension of a list of activities of a critical path through the activities that do not belong to the critical path. The activities that do not belong to the critical path are arranged with respect to non-decreasing length of individual lists. The computation is performed as described above, considering each element of the circular list as the beginning of one of the J priority lists. The maximum of the J bounds determines the bound LB3. Note, first, since a circular list is used, the length of the individual list cannot be reduced. Second, computational overhead through multiple calculation of the bound for the same set of scheduled (unscheduled) activities, is not taken into account. The bounds are repetitively calculated for identical sets of scheduled (unscheduled) activities. However, the bound is at least as good as the MPM-bound.

DH97 uses only one priority list. The priority list is obtained through ordering the activities with respect to non-decreasing length of the individual lists. Ties are broken with respect to non-increasing duration. Note, first, the length of the individual list is not reduced. Second, this variant of the bound LB3 can be worse than the precedence-based bound (MPM-duration). Third, computational overhead through multiple calculation of the bound for the same set of scheduled (unscheduled) activities is not taken into account. The differences of the variants are summarized in Table 1. Note, a pre-study has shown that the implementation of the original HWNP (cf. [9]) is slower than the implementation of LB3 as outlined above.

Algorithm	Name	Number of Lists	Size of Individual Lists	Improves MPM-Bound	Multiple Computation
Mingozzi et al.	LB3	J(implicitly)	full length	yes	yes
DH97	LB3DH	1	full length	no	yes
Sprecher	LB3Mod	3	reduced length	yes	no

Table 1: Variants of the Bound LB3

3.4 Data Storage and Retrieval

In the sequel we will contrast the implementation of our Set-Based Dominance Rule with DH97's implementation of the Cutset Dominance Rule.

Name	Type	Components	Size Specification	Interpretation
SetTree	array	SetNode	input parameter	array of all SetNodes selected to be stored during computation
SetNode	structure	SetPtr	4 bytes	pointer to set of scheduled activities
Set	array	unsigned	$1 \leq J \leq 32$ 4 bytes	binary coded set of scheduled activities
		long integer	$33 \leq J \leq 64$ 8 bytes	
			$65 \leq J \leq 96$ 12 bytes	
		MaxCT	4 bytes	maximum completion time of scheduled activities
		LBCompSet	4 bytes	lower bound of the minimum makespan required to schedule the activities out of the complement set
		Left	4 bytes	pointer to left SetNode
		Right	4 bytes	pointer to right SetNode

Table 2: Data Structure for Implementing Set-Based Dominance

The data structure implemented to realize the Set-Based Dominance Rule is described in Table 2. The array "SetTree" is used to store all the information related to previously generated partial schedules necessary to apply the Set-Based Dominance Rule. Its size is determined through an input parameter. The components of the array are structures with six components. The structure consists of a pointer to a set of scheduled activities. The sets are, as usual, binary coded, that is, assuming $J = 32$, the binary digits 0000 0000 0010 1011 coincide with the integer $1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^3 + 1 \cdot 2^5 = 43$, and the set $\{1, 2, 4, 6\}$. Note, first, since exactly four bits are set to 1, the set can only correspond to a set of scheduled activities related to a partial schedule of level 4. Second, we have selected a pointer, to a "Set" instead of an array of fixed size, to enable the algorithm to deal with projects of any size without manual adaptation of the data structures. The component "MaxCT" contains the maximum completion time of the activities that are scheduled. The component "LBCompSet" contains a lower bound of the makespan required to schedule the unscheduled activities. Note, by storing the bound related to a set of scheduled (unscheduled) activities the lower bound has to be computed only once, and then can be taken from the memory if a partial schedule with same set of scheduled activities, more precisely, set of unscheduled activities, is studied. The components "Left" and "Right" are pointer to (addresses of) "SetNode" that are used to built the level dependent binary trees. The pointers are initialized by NULL. The roots of the binary trees are stored in the components SetTree[j], $j = 1, \ldots, J$, of the array "SetTree". For $J \leq 32$ ($33 \leq J \leq 64$) one element of the array requires 24 (28) bytes.

We consider Table 3 with some sample entries required to apply the Set-Based Dominance Rule. We assume $J = 8$. During the initialization first according to the number of "SetNode" the array "SetTree" is build through allocation of memory. Subsequently the cells are initialized, and memory to store the sets is allocated to the set pointers "SetPtr"– indicated by $\downarrow$, and, afterwards, the sets are initialized, too. The first J cells of

	SetTree[j]											
j	1	...	4	...	8	...	x	...	y	...	z	...
SetPtr			↓				↓		↓		↓	
Set			$\{1,2,3,4\}$				$\{1,2,3,6\}$		$\{1,2,4,5\}$		$\{1,2,4,6\}$	
int(Set)			15=1111				39=100111		27=11011		43=101011	
MaxCT			13				12		13		9	
LBCompSet			6				7		8		7	
Left			NULL				& SetTree[y]		NULL		NULL	
Right			& SetTree[x]				& SetTree[z]		NULL		NULL	

Table 3: Sample Entries for Data Structure SetTree

the array "SetTree" are then reserved for the roots of the level-dependent trees. So, e.g., SetTree[4] is root of the binary tree related to level 4. For the memory location that stores the set of scheduled activities, we have specified beside the interpretation as a set, the actually stored interpretation as an integer. The first set of scheduled activities of level 4 that has been stored, is the set $\{1,2,3,4\}$. Later, the set $\{1,2,3,6\}$ is stored at SetTree[x] the address of which is given in cell "Right" of SetTree[4] (since $15 < 39$). Thereby, the position x is determined as the lowest still un-used component of array "SetTree". Continuing leads to the definition of SetTree[y] and SetTree[z] with the sets $\{1,2,4,5\}$ and $\{1,2,4,6\}$ stored to the left (since $27 < 39$) and right (since $39 < 43$) of SetTree[x]. Since lower bounds, like the resource-based bound $LBRes$ (cf. [17]), $LB3$ and $LB3Mod$, of the makespan required to schedule the activities of the complement set, depend on the set of activities that is scheduled, they only have to be computed once. After computation the bound is stored together with the set of scheduled activities, and the maximum completion time.

During enumeration, after it is decided to schedule activity g_i, i.e., before incrementing the level index from i to $(i+1)$, the "SetTree" is searched. Beginning in the root SetTree[i] of the binary tree of level i the previously stored sets are compared with the current set of scheduled activities $\mathcal{ACS}_i$. If the set is found then the completion time found CTF_i and the lower bound found LBF_i is defined through the maximum completion time and the lower bound saved earlier, otherwise we define $CTF_i = MaxInt$ and $LBF_i = 0$. If now, on level $(i+1)$, the start time $ST_{g_{i+1}}$ of an activity g_{i+1} selected to be scheduled fulfills $ST_{g_{i+1}} \geq CTF_i$, then in accordance with the Set-Based Dominance Rule the current selection on level $(i+1)$ can be skipped. Additionally, lower bounding allows to skip the current selection, if $ST_{g_{i+1}} + LBF_i > LF_J$.

When tracking back from level $(i+1)$ to level i, the current set node information of level i replaces previously stored information if $CT^{max}(Seq_i) < CTF_i$, and is stored in a new leave of the binary tree, if the start time $ST_{g_{i+1}}$ of at least one activity g_{i+1} tested on level $(i+1)$ fulfills $CT^{max}(Seq_i) \leq ST_{g_{i+1}}$.

DH (cf. [3]) define for a time instant m a *cut-set* C_m as the set of all unscheduled activities for which all predecessor activities are in the partial schedule PS_m. Of course, since a cut-set does not depend on the time instant but on the set of scheduled activities only, there is a one-to-one correspondence between a cutset and the set of scheduled activities. The Cutset Dominance Rule is stated as follows:

Theorem 2 *(Cutset Dominance Rule, cf. [3]) Consider a cutset C_m at time m which contains the same activities as a cutset C_k, which was previously saved during the search of another path in the search tree. If time k was not greater than time m and if all activities in progress at time k did not finish later than the maximum of m and the finish time of the corresponding activities in PS_m then the partial schedule PS_m is dominated.*

DH97 employed two matrices to implement the rule. One of a size of 256 KB, to store the pointers to the cutsets. The cutsets are stored in the cutset matrix together with the "additional information". We need to store

1. the pointer to the cutset – 4 bytes,
2. the cutset CS – 4 bytes for $J \leq 32$, 8 bytes for $33 \leq J \leq 64$,
3. the decision point CS_t – 4 bytes,
4. the number A of activities in progress at time CS_t – 4 bytes;
5. the pointer to the next block of cutset information – 4 bytes,
6. the number of the a'th activity in progress at time CS_t – 4 bytes, $a = 1, \ldots, A$.
7. the completion time of the a'th activity in progress at time CS_t – 4 bytes, $a = 1, \ldots, A$.

Consequently, for $J \leq 32$, the requirements for storing the first block of information related to one specific cutset sums up to 20 bytes if no activity is in progress at the decision point, and $20 + (2 \cdot A) \cdot 4$ bytes, if A activities are in process at the decision point. Since the pointer in the pointer matrix and the cutset in the cutset matrix have to be stored only once, further blocks of information related to a cutset already stored, require $12 + (2 \cdot A) \cdot 4$ bytes. Note, it is not explicitly stated how the authors deal with previously stored blocks of cutset information that are dominated by the current block of cutset information.
Since (a) the Set-Based Dominance Rule requires only one block of information per set of scheduled activities, (b) the memory requirements for one block of information for the Set-Based Dominance Rule are, as a rule, less the requirements for one block of information of the Cutset Dominance Rule, (c) not all the blocks of information related to partial schedules are stored, the memory requirements for Set-Based Dominance are not that excessive as for the Cutset Dominance Rule implemented by DH97.
Apart from the low memory requirements (cf. Section 4), the Set-Based Dominance retrieves the information more efficiently than the Cutset Dominance Rule. The former rule employs binary search, the latter rule employs linear search. Increasing the number of activities of the project, obviously induces (1) that the need of memory for the Cutset Dominance Rule substantially exceeds the need for the Set-Based Dominance Rule, (2) that the need of memory for the Cutset Dominance Rule rises much faster than the need for the Set-Based Dominance Rule, (3) that a substantial increase of memory usage slows down the linear search – performed in the Cutset Dominance Rule – more drastically than the binary search – performed in the Set-Based Dominance Rule. Moreover, storing the bound obviously increases the performance substantially.

Table 4 summarizes all the theoretical developments of this paper. The practical impact is presented in the next section.

Rule	Reference	Special Features
(1) Extended and Simplified Single Enumeration Rule	[15]	(a) extension, (b) simplification
(2) Local Left-Shift Rule	[3], [4], [9], [15], [17]	–
(3) Extended Global Left-Shift Rule	[15]	reduced backtrack level
(4) Contraction Rule	—–	upper bounds on start time through (a) scheduling, (b) local left-shifts, (c) global left-shifts
(5) Set-Based Dominance	[3], [4], [9], [15], [17], [19]	(a) binary search, (b) low memory requirements, (c) combined with lower bounding
(6) Simple Permutation Rule	—–	(a) low memory substitute of Cutset Rule and Set-Based Dominance, (b) extension of Set-Based Dominance
(7) Non-Optimality Rule	—–	necessary condition for optimality
(8) Lower Bound LB3	[4], [9]	increased efficiency

Table 4: Summary of Developments

4 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 LINUX. The data structures allow to deal with projects of any size through parameter adjustment. The multi-mode version studied in [15] served as the basis of our implementation. The following modifications 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. [4]). 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 implemented, extended, simplified and accelerated as described in Subsection 3.2, Subsection 3.3, and Subsection 3.4. (e) Assuming constant resource availability, like DH97, 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.

Table 5 describes the different variants of the algorithm that have been implemented. (+) indicates that the respective option is incorporated, and (–) that it is not. All variants include (1) the Contraction Rule, (2) the Extended and Simplified Single Enumeration Rule, (3) the Local Left-Shift Rule, and (4) the Extended

Variant	Permutation Rule	LBRes	LB3	LB3Mod	Set-Based Dominance	Non-Optimality	LB3Mod-Red
1	+	−	−	−	−	+	−
2	+	+	−	−	−	+	−
3	+	−	+	−	−	+	−
4	+	−	−	+	−	+	−
5	+	−	−	+	+	+	−
GSA	+	−	−	+	+	+	+

Table 5: Variants Implemented

Global Left-Shift Rule. Three different bounds have been studied. First, the resource-based bound LBRes (cf. Stinson et al. [17]), second, the bound LB3 (cf. [9], Subsection 3.3), and third, the variant LB3Mod of the bound LB3 (cf. Subsection 3.3). Note, for the Set-Based Dominance Rule it did not pay, to store all the sets that relate to a sequence once generated during enumeration. A set $\mathcal{ACS}_i$ induced by a sequence Seq_i is stored only, if the start time of at least one activity g_{i+1} considered to be scheduled on level $(i+1)$, fulfills $CT^{max}(Seq_i) \leq ST_{g_{i+1}}$. It is stored when tracking back from level $(i+1)$ to level i. Of course, if the maximum completion time $CT^{max}(Seq_i)$ of a current sequence Seq_i is less than the one produced by a previously studied sequence $\overline{Seq_i}$ with same set of scheduled activities, then $CT^{max}(Seq_i)$ replaces $CT^{max}(\overline{Seq_i})$. However, when combining the Set-Based Dominance with lower bound computations, the effort necessary can be reduced further by storing the bounds once computed. If the set has been previously stored, then the bound is retrieved from the memory, otherwise it is (newly) computed. The option, LB3Mod-Red, refers to the combination of the Set-Based Dominance and lower bounding by LB3Mod, where the bounds are only computed when it is intended to store the information for set based dominance, and, consequently, the bound computed. The variant that employs all the options is referred to as general sequencing algorithm (GSA). The bounding rules presented are applied, in the following order: (1) Contraction Rule, (2) Local Left-Shift Rule, (3) Extended and Simplified Single Enumeration Rule (4) Set-Based Dominance Rule, (5) Non-Optimality Rule, (6) Simple Permutation Rule, (7) Extended Global Left-Shift Rule.

The procedure has been tested on the standard benchmark set produced by the project generator ProGen (cf. [8]). The projects consist of 32 activities (including two dummy activities), and 4 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 DH97. Unfortunately, an executable code of DH97 was not available to us, so we compare with the results presented in [4]. The average CPU-times and the frequency distributions are displayed in Table 6 and Table 7.

We allowed 3600 seconds for enumeration. The activities have been relabeled in accordance with minimum

No of	DH97		Variant							
instances	Mem.	CPU	1	2	3	4	5	GSA	GSA	GSA
		[sec.]	–	–	–	–	256 KB	128 KB	256 KB	512 KB
466	256 KB	56.46	121.22	74.78	45.32	15.44	2.32	1.58	1.41	1.51
470	1 MB	17.13	150.82	104.79	73.14	26.07	2.97	2.36	1.78	1.87
475	4 MB	6.23	187.13	141.58	110.26	56.70	4.82	5.82	2.88	2.94
478	16 MB	10.44	208.55	163.28	132.16	78.94	14.41	25.30	10.38	9.44
479	24 MB	12.33	215.63	170.46	139.40	86.29	21.90	32.77	17.87	15.28

Table 6: Average CPU-Times [sec.] of Different Variants versus DH97

Variant	Memory	Range of CPU-Times [sec.]							
		[0;0.5]	(0.5-1]	(1-5]	(5-10]	(10-100]	(100-500]	(500-3600)	≥ 3600
1		323	25	32	14	32	18	12	24
2	–	321	21	36	19	35	14	19	15
3	–	256	39	58	28	53	18	16	12
4	–	316	32	46	11	42	16	10	7
5	256 KB	323	43	55	16	32	6	3	2
GSA	128 KB	364	36	37	7	25	6	1	4
GSA	256 KB	354	38	45	7	29	2	4	1
GSA	512 KB	335	50	52	6	30	2	4	1

Table 7: Frequency Distribution of CPU-Times of Different Variants

earliest finish time (MinEFT). The first column of Table 6 shows the number of problems that could be solved by DH97 within 3600 seconds on a IBM personal computer (PS/2 Model P75, 25 MHz) using Microsoft Visual C++ 2.00 under Windows NT 3.50. The memory DH97 used to store the cutset matrix is given in the second column. Additional 256 KB are required to store the pointer matrix. The average CPU-time to solve these problems is listed in the third column. The column 4 through column 11 show the average CPU-times for the different variants of our algorithm. If the variants employ set-based dominance then the memory required, i.e. 128 KB, 256 KB, 512 KB, is given in the third row. Unfortunately, an executable version of DH97 was not available to us. We assume a comparison factor of approximately 2.7 to balance the difference clockpulses (DH97/25 MHz, GSA/66 MHz) of the different machines. From the tables we can deduce

1. by comparing variant 1 with variants 2, 3, and 4: The bounds reduce the average CPU-times.
2. by comparing variant 2 with variant 3: The bound $LB3$ has a stronger effect than the resource-based bound $LBRes$.
3. by comparing variant 3 with variant 4: The bound $LB3Mod$ has a stronger effect than the bound $LB3$.
4. by comparing DH97 with variant 4: On 97% of the problems, the sequencing algorithm achieves even without extensive use of memory a lower average CPU-time than DH97 using more than 256 KB.

5. by comparing variant 4 with variant 5: The set-based dominance and bound-storing policy reduce the average CPU-time significantly.

6. by comparing variant 5 with GSA at 256 KB: Computing the bound $LB3Mod$ only if it will be stored, reduces the average CPU-time.

7. by comparing DH97 with GSA: Allowing GSA and DH97 to use up to 256 KB, then GSA can solve 97% of the problems on average in a fraction of the CPU-time required by DH97,

8. by comparing DH97 with GSA: Allowing GSA to use only 256 KB and DH97 to use up to 4 MB, does not favor the use of DH97,

9. by comparing DH97 with GSA: Allowing DH97 to use 16 or 24 MB slightly favors the use of DH97.

Since, in general, the memory requirements exponentially grow with number of activities the project consists of, memory will become a critical resource. Additionally, the high memory requirements might slow down the computation, when the previously stored information is retrieved through linear search (DH97) instead of binary search (GSA).
Moreover, we have studied several priority rules, like, MinJobNr, MinSlack, MinEST, MinEFT, MinLST, MinLFT, MinDur, etc. The average CPU-times vary by some ten percent for 466 (470) problems, and by some twenty percent for 475 (479) problems. Note, similar to DH97, one instance was hard to solve. Whereas DH97 could solve j3029-1 (new name j3013-1) within three hours (9,515 sec. employing the MPM-bound, 10,261 sec. employing LB3DH), GSA could solve j3029-5 (new name j3013-5) depending on the priority rule within three hours (MinEST 19,006, MinEFT 13,651 sec., maximum cumulated resource usage of reflexive, transitive successors 9,968 sec.).

GSA,66 MHz, 256 KB, CPU [sec.]	0.03	0.07	0.18	0.20	0.37	0.5	1.0	1.85	5.0	10.00
$\overline{\Delta}$ [%]	–	–	2.513	2.222	1.447	1.110	0.741	0.571	0.317	0.206 –
DH97, 25 MHz, 24 MB, CPU [sec.]	0.10	0.20	0.50	–	1.00	–	–	5.0	–	–
$\overline{\Delta}$ [%]	1.865	1.320	0.839	–	0.623	–	–	0.326	–	–
GSA,66 MHz, 256 KB, CPU [sec.]	11.11	22.22	60	111.11	300	444.44	1200	1333.33	3600	
$\overline{\Delta}$ [%]	0.175	0.130	0.060	0.034	0.022	0.015	0.012	0.012	0.003	
DH97, 25 MHz, 24 MB, CPU [sec.]	30.0	60.0	–	300	–	1200	–	3600	–	
$\overline{\Delta}$ [%]	0.120	0.057	–	0.021	–	0.00	–	0.00	–	

Table 8: Average Deviations from Optimality at Different Time-Limits [sec.] for GSA (256 KB) versus DH97 (24 MB)

The heuristic capabilities of our algorithm have been studied too. Table 8 compares the average percentage deviations $\overline{\Delta}$ of the GSA-determined (80486, 66 MHz, 256 KB) makespan from minimum makespan with the average percentage deviations $\overline{\Delta}$ of the DH97-determined (80486, 25 MHz, 24 MB) makespan from minimum makespan at different time-limits. In order to balance the different clockpulses of the different machines a factor of 2.7 has been considered in the comparison. That is, we allotted DH97 2.7 times of the CPU-time

that is allotted to GSA when searching for a solution. Note, whereas GSA could not determine a single feasible solution within 0.03 or 0.07 (66 MHz) seconds DH97 seemingly solved a number of problems within 0.1 (25 MHz) seconds. This seems to be caused through initialization time. In our implementation of GSA the allocation, initialization, and deallocation of the entire storage employed by set-based dominance, the MPM-calculations, and the relabeling of the activities are part of the CPU-time. These basic calculations require 0.12 (66 MHz) seconds. However, if GSA and DH97 are allotted less than a second than DH97 produces slightly better results. As the allotted CPU-time increases (more than 1 second) the differences of average deviations of GSA (256 KB) and DH97 (24 MB) reduce to fractions of a percent. Consequently, the truncated version of GSA (256 KB) compares as good as DH97 (24 MB) with the heuristics developed for the RCPSP.

Furthermore, we have compared our algorithm with the branch-and-bound algorithm BB LB3 of Mingozzi et al. (cf. [9]). This new branch-and-bound concept studies continuations of a list of feasible subsets. The algorithm employs (a) a reduction of the number of feasible subsets to be selected for continuation, (b) a left-shift rule, and (c) the Cutset Dominance Rule, (d) the bound LB3. It has been implemented in Fortran 77 on an IBM PS/2 55 sx (80386 sx, 15 MHz) running under DOS. The memory requirements are not known, but – due to the use of the Cutset Dominance Rule – should amount at least at several hundred kilo bytes. We compare the average CPU-times of GSA (128 KB) and BB LB3 on different parameter groups (cf. [7]) of the instance set j30. The results are displayed in Table 9. Note, the average CPU-times of GSA for parameter group 29 (new 13) includes four instances that could not be solved within the time limit of 3600 seconds. Taking a comparison factor of less than twenty (due to architecture, clockpulse, and operating system), we see that GSA produces a lower average CPU-time on the majority of the parameter groups (indicated by *). BB LB3 outperforms GSA on parameter group 29 (new 16, $C = 1.5$, $RF = 1.0$, $RS = 0.2$) and parameter group 45 (new 29, $C = 1.8$, $RF = 1.0$, $RS = 0.2$). That is, the problems that have been identified as extremely hard to solve, can be solved faster by BB LB3 than by GSA.

label	Parameter Group																	
old	21	22	23	24	25	26	29	30	31	37	38	41	45	46	53	57	61	62
new	5	6	7	8	9	10	13	14	15	21	22	25	29	30	37	41	45	46
BB LB3	186.4	76.5	31.1	11.6	1001.4	274.8	10728.7	803.0	184.0	40.6	64.5	384.8	2753.3	181.6	26.3	59.8	83.7	112.3
GSA	3.7	0.8	0.2	0.1	37.3	3.7	1595.2	26.3	9.2	0.4	0.7	5.9	231.9	5.8	0.2	0.6	0.6	2.3
factor	50.4	95.6	155.5	116.0	26.8	74.3	6.7	30.5	20.0	101.5	92.1	65.2	11.9	31.3	131.5	99.7	139.5	48.8
	*	*	*	*	*	*	—	*	=	*	*	*	—	*	*	*	*	*

Table 9: CPU-Times [sec.] of GSA (128 KB) and BB LB3 for Different Parameter Groups

In the last experiment we have applied the GSA (Pentium, 166 MHz, LINUX) to projects that consist of 60 (j60) and 90 (j90) non-dummy activities. Using the network complexity $C = 1, 5, 1.8, 2.1$; the resource factor $RF = 0.25, 0.50, 0.75, 1.00$; and the resource strength $RS = 0.2, 0.5, 0.7, 1.0$, 480 instances have been generated per set (cf. [7]). The only authors that have published results on the set j60 are Brucker et al. (cf. [1]). Brucker et al. (BKST) developed 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 20/801) operating under Solaris 2.5 with 64 MB general storage (80 MHz + 1 MB SC). Within an imposed time limit of 3600 seconds the algorithm could find and verify an optimal solution for 425 of the 480 test problems with 30 non-dummy activities. The average percentage deviation from optimality was 1.2%. The comparison of GSA and BKST for the set j60 is displayed in Table 10. Note, for the GSA it paid to determine a first feasible solution of larger projects as the earliest start schedule induced by the priority values. The priority rules employed by GSA are displayed in the third row. However, since all the optimal solutions of the larger projects are not known, we utilized the lower bounds $LB\ BKST$ provided by Brucker et al. in order to assess the quality of the solutions determined. The bound implements the LB2 concept of Mingozzi et al. (cf. [9]) through delayed column generation technique. We have listed (1) the number of problems of the set (# problems), (2) the allotted CPU-time (Time Limit), (3) the number of problems (# solv. prob.) solved within the allotted time, (4) the average CPU-times CPU [sec.] for the entire set, (5) the average percentage deviation $\overline{\Delta}(LB\ BKST)$ of the makespan determined from $LB\ BKST$, (6) the maximum percentage deviation $\Delta^{max}(LB\ BKST)$ of the makespan determined from $LB\ BKST$, (7) the average percentage deviation $\overline{\Delta}(MPM)$ of the makespan determined from MPM-bound, (8) the maximum percentage deviation $\Delta^{max}(MPM)$ of the makespan determined from MPM-bound, (9) the memory required by the algorithms.

		j60 BKST	j60 GSA			j90 GSA	
			MinEST	MinEFT	MinEST	MinEST	MinEFT
(1)	# problems	480	480	480	480	480	480
(2)	Time Limit [sec.]	3600	300	300	1800	300	300
(3)	# solv. prob.	326	349	343	364	295	274
(4)	CPU [sec.]	—	88.07	92.67	472.69	120.26	137.41
(5)	$\overline{\Delta}(LB\ BKST)$	4.8 %	5.72 %	6.40 %	5.27 %	8.25 %	9.94 %
(6)	$\Delta^{max}(LB\ BKST)$	30.8 %	45.76 %	52.54 %	40.68 %	58.67 %	64.00 %
(7)	$\overline{\Delta}(MPM)$	—	13.60 %	14.38 %	13.04 %	15.73 %	17.61 %
(8)	$\Delta^{max}(MPM)$	—	131.17 %	131.17 %	128.57 %	123.73 %	138.9 %
(9)	Memory	$<$ 10 MB	$<$ 6 MB	$<$ 6MB	$<$10 MB	$<$ 8 MB	$<$11 MB

Table 10: GSA versus BKST

We observe (1) that GSA solves more problems to optimality than BKST, (2) that the average deviation $\overline{\Delta}(LB\ BKST)$ and the maximum deviation $\Delta^{max}(LB\ BKST)$ of GSA exceed the ones of BKST. This phenomenon might be reasoned by the different "perspectives" from which the approaches study the problems. That is, if the allotted CPU-time does not suffice to find an optimal solution then the branch-and-bound concepts that rely on the continuation of partial schedules, e.g. the ones of Demeulemeester and Herroelen [3], [4], Stinson et al. [17], Mingozzi et al. [9], and the GSA, tend to spend their time on the arrangement of activities scheduled on high levels of the branch-and-bound tree. The arrangement of activities on low levels – although it might substantially improve the quality – is considered less. The approach developed

by Brucker et al. has a more global perspective of the schedules to be analyzed. Even on low levels of the branch-and-bound tree the arrangement of activities of the "beginning" and the arrangement of the activities of the "end" of the schedules can be considered.

5 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. [8]). 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 encourage to study further variants of the algorithm.

The approach can compete with the best solution procedures currently available. The algorithm uses far less memory than the state-of-the-art procedure DH97, i.e., 256 KB versus 24 MB, for solving the standard benchmark set with projects consisting of 32 activities within comparable time. If both approaches are allowed to make limited use of memory, i.e. 256 KB, then more than 97% of the benchmark instances can be solved within fractions of the time required by the current state-of-the-art procedure DH97. The truncated version of our algorithm achieves at 256 KB approximately the results of the truncated version of the state-of-the-art approach DH97 at 24 MB. Since, in general, the memory requirements exponentially grow with the number of activities the project consists of, memory will become a critical resource. Additionally, the high memory requirements might slow down the computation, when the previously stored information is retrieved through linear search (DH97) instead of binary search (GSA). Moreover, since comparison pruning is nearly exhausted in the approach by DH97, we believe that our algorithm will gain more by the development of new dominance rules and bounds.

Acknowledgments: I would like to thank Andreas Drexl and Sönke Hartmann as well as the anonymous referees for helpful comments and suggestions.

A Appendix

For illustrational purposes, we will use the AON representation of a project as given in Figure 1 (cf. [5], p. 179).

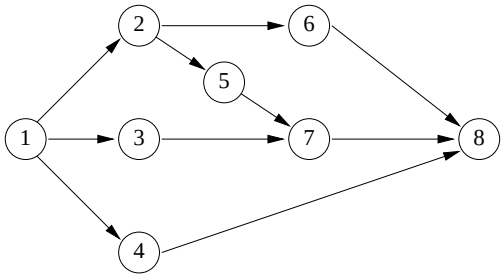

Figure 1: Example Network

We assume that two renewable resources with constant availability of $K_1 = 2$ and $K_2 = 3$, 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, activities 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. Furthermore, we assume the eligible set to be examined with respect to increasing labels.

An example for the Local Left-Shift Rule is given in Figure 2. It is $i = 5$, $Seq_5 = [1, 2, 3, 5, 4]$, $g_6 = 6$, and $ST_{g_6} = ST_6 = 11$, $\overline{ST}_6 = 10$.

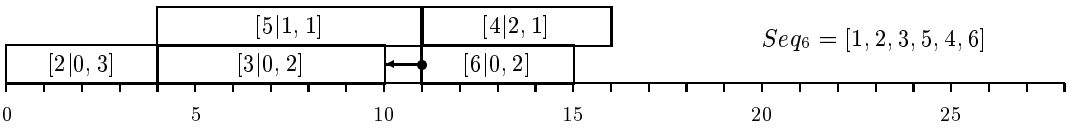

Figure 2: Local Left-Shift Rule

An example for the Global Left-Shift Rule is given in Figure 3. It is $i = 5$, $Seq_5 = [1, 2, 3, 5, 6]$, $g_6 = 4$ and $\overline{ST}_{g_6} = \overline{ST}_4 = 0$. The backtrack level is $k = 3$. Note, activity 4 can not be locally left-shifted.

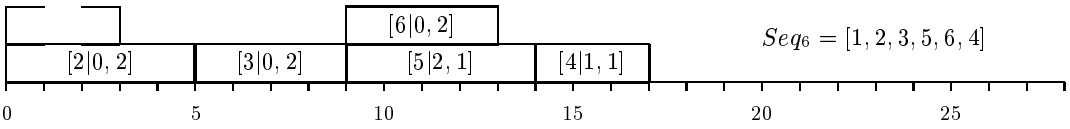

Figure 3: Global Left-Shift Rule

An example for the Set-Based Dominance Rule is given in Figure 4. It is $i = 6$, $Seq_6 = [1, 2, 4, 6, 3, 5]$, $\overline{Seq}_6 = [1, 2, 3, 5, 4, 6]$, $\overline{\mathcal{ACS}}_6 = \{1, 2, 3, 4, 5, 6\} = \mathcal{ACS}_6$, $CT^{max}(\overline{Seq}_6) = 13$, $g_7 = 7$, and $ST_7 = 13$.

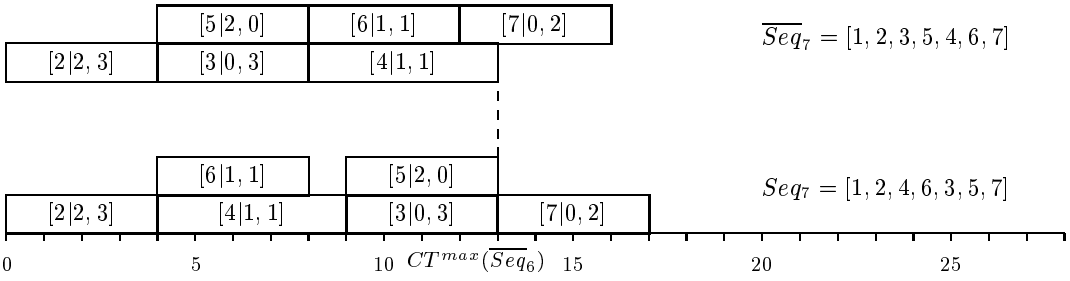

Figure 4: Set-Based Dominance

An example for the Simple Permutation Rule is given in Figure 5. It is $i = 5$, $Seq_i = Seq_5 = [1, 2, 4, 3, 5]$, $g_6 = 6$, $ST_{g_6} = ST_6 = 6$, $k = 4$, $g_4 = 3$, $CT_{g_4} = CT_3 = 6$, i.e., $CT_3 \leq ST_6$, $l = 3$, $g_3 = 4$, $ST_{g_3} = 0$. Activity $g_4 = 3$ and activity $g_3 = 4$ can be interchanged such that activity 3 starts at $\overline{ST}_3 = ST_4 = 0$ and activity 4 finishes at $\overline{CT}_4 = CT_3 = 6 \leq ST_6$. Interchanging both activities neither violates the precedence nor the 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 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 are at most equal to ones of the previously studied sequence $\overline{Seq_6}$, the current sequence is dominated.

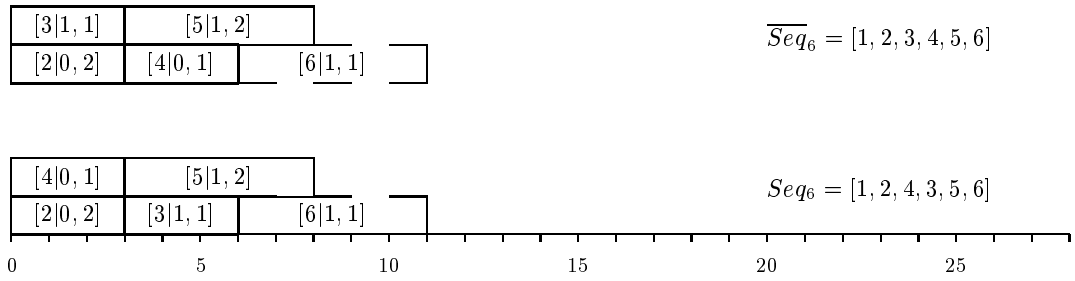

Figure 5: Simple Permutation Rule

An example for the Non-Optimality Rule is given in Figure 6. It is $Seq_i = Seq_6 = [1, 2, 3, 4, 5, 6]$ – the partial schedule is displayed in the lower part of the figure – $g_{i+1} = g_7 = 7$, (a) $ST_{g_7} = CT^{max}(Seq_6) = 14$, (b) $g^{max} = 6$, $\Delta^{max} = CT^{max}(Seq_6) - CT^{\overline{max}}(Seq_6) = CT_6 - CT_5 = 14 - 12 = 2 > d_{g^{max}} - \Delta_{g^{max}} = d_6 - \Delta_6 = 4 - 3 = 1$. Activity 6 as a whole cannot be left-shifted, but 3 periods of it could be left-shifted to start at period $ST = 3$, which

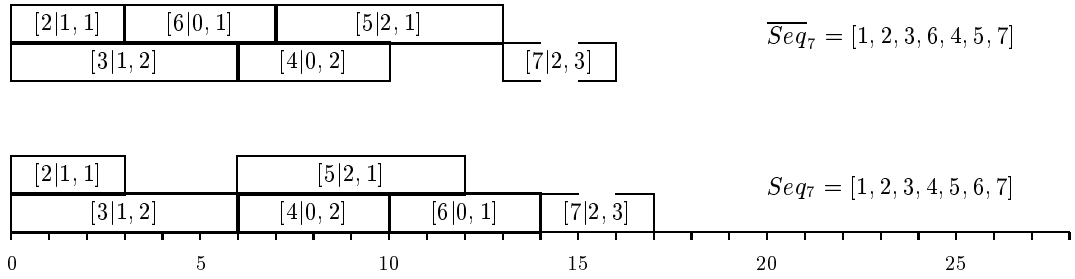

Figure 6: Non-Optimality Rule

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 $\Delta^{max} - \Delta_{g^{max}} = 2 - 1 = 1$ period(s).

References

[1] Brucker, P.; S. Knust; A. Schoo and O. Thiele (1997): A branch & bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research (to appear).

[2] 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.

[3] 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.

[4] Demeulemeester, E. and W. Herroelen (1997): New benchmark results for the resource-constrained project scheduling problem. Management Science, Vol. 43 (to appear).

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

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

[7] Kolisch, R. and A. Sprecher (1996): PSPLIB – A project scheduling problem library. European Journal of Operational Research, Vol. 96, S. 205–216.

[8] 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.

[9] Mingozzi, A.; V. Maniezzo; S. Ricciardelli and L. Bianco (1997): An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation. Management Science (to appear).

[10] 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.

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

[12] 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.

[13] Sprecher, A. (1997): A Competitive Exact Algorithm for Assembly Line Balancing. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 449, Kiel, Germany.

[14] Sprecher, A. and A. Drexl (1996): Minimal Delaying Alternatives and Semi-Active Time Tabling in Resource-Constrained Project Scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 426, Kiel, Germany.

[15] Sprecher, A. and A. Drexl (1997): Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. European Journal of Operational Research (to appear).

[16] 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.

[17] 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.

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

[19] 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.