Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms

Sönke Hartmann*, Andreas Drexl**

Christian-Albrechts-Universität zu Kiel, Lehrstuhl für Produktion und Logistik, D-24098 Kiel, Germany. E-mail: hartmann@bwl.uni-kiel.de, drexl@bwl.uni-kiel.de

Abstract. This paper is devoted to a comparison of all available branch-and-bound algorithms that can be applied to solve resource-constrained project scheduling problems with multiple execution modes for each activity. After summarizing the two exact algorithms that have been suggested in the literature, we propose an alternative exact approach based on the concepts of mode and extension alternatives to solve this problem. Subsequently, we compare it to the two procedures available in the literature. Therefore, the three algorithms as well as all available bounding criteria and dominance rules are summarized in a unified framework. In addition to a theoretical comparison of the procedures, we present the results of our computational studies in order to determine the most efficient algorithm.

Keywords. Project Management and Scheduling, Multiple Modes, Branch-and-Bound, Bounding Rules, Computational Results.

1 Introduction

Within the classical resource-constrained project scheduling problem (RCPSP), the activities of a project have to be scheduled such that the makespan of the project is minimized. Thereby, technological precedence constraints have to be observed as well as limitations of the renewable resources required to accomplish the activities. Once started, an activity may not be interrupted.

This problem has been extended to a more realistic model, the multi-mode resource-constrained project scheduling problem (MRCPSP). Here, each activity can be performed in one out of several modes. Each mode of an activity represents an alternative way of combining different levels of resource requirements with a related duration. Following Slowinski [13], renewable, nonrenewable and doubly constrained resources are distinguished. While renewable resources have a limited per-period availability such as manpower and machines, nonrenewable resources are limited for the entire project, allowing to model e. g. a budget for the project. Doubly constrained resources are limited both for each period and for the whole project. However, since they can simply be incorporated by enlarging the sets of the renewable and nonrenewable resources, we do not consider them explicitly. The objective is to find a mode and a start time for each activity such that the schedule is makespan minimal and feasible with respect to the precedence and resource constraints.

A broad variety of branch-and-bound procedures has been proposed for optimally solving the single-mode RCPSP. The approaches of Stinson et al. [21], Talbot and Patterson [23], Christofides et al. [3], Demeulemeester and Herroelen [4], Mingozzi et al. [10], and Sprecher [16] enumerate

partial schedules in different ways. Approaches based on graph representations have been suggested by Radermacher [12], Bartusch et al. [1], and Brucker et al. [2]. For the multi-mode case, all procedures developed up to now utilize the concept of partial schedules, cf. the approaches of Patterson et al. [11], Sprecher and Drexl [17], Speranza and Vercellis [14], and Sprecher et al. [19]. However, Hartmann and Sprecher [7] have shown that the procedure proposed in [14] is not correct, that is, in some cases, it finds only suboptimal solutions or even fails to determine an existing feasible solution.

This paper deals with exact solution methodologies for the MRCPSP. We introduce an alternative branch-and-bound procedure and, moreover, provide a thorough comparison of the algorithms proposed in the literature and the new approach. The procedures are described in a unified way and are compared both theoretically and numerically. In order to provide a fair comparison, we have employed each available acceleration method into each enumeration algorithm. The three procedures have been implemented and compared on a standard set of project instances that has been generated using the problem generator ProGen developed by Kolisch et al. [9]. This enables us to determine the most efficient procedure currently available to solve the MRCPSP.

The remainder is organized as follows: Section 2 provides the description of the problem. Section 3 summarizes the enumeration algorithms known from the literature and introduces the new one. Section 4 contains the available dominance rules as well as a new concept. Section 5 is devoted to a theoretical comparison of the enumeration schemes. Section 6 provides the results of our computational comparison of the three procedures. Finally, Section 7 states some conclusions.

2   Problem Description

We consider a project which consists of J activities (jobs) labeled $j = 1, \ldots, J$. Due to technological requirements the activities are partially ordered, that is, there are precedence constraints between some of the jobs. These precedence constraints are given by sets of immediate predecessors P_j indicating that an activity j may not be started before all of its predecessors are completed. The precedence constraints can be represented by an activity-on-node network which is assumed to be acyclic. We consider additional activities $j = 0$ representing the only source and $j = J + 1$ representing the unique sink activity of the network.

With the exception of the (dummy) source and (dummy) sink activity, each activity requires certain amounts of resources to be performed. The set of renewable resources is referred to as R. For each renewable resource $r \in R$ the per-period-availability is constant and given by K_r^ρ. The set of nonrenewable resources is denoted as N. For each nonrenewable resource $r \in N$ the overall availability for the entire project is given by K_r^ν.

Each activity can be performed in one of several different modes of accomplishment. A mode represents a combinination of different resources and/or levels of resource requests with a related duration. Once an activity is started in one of its modes, it is not allowed to be interrupted, and its mode may not be changed. Activity j may be executed in M_j modes labeled $m = 1, \ldots, M_j$. The duration of job j being performed in mode m is given by d_{jm}. We assume the modes to be labeled with respect to non-decreasing duration, that is, $d_{jm} \leq d_{jm+1}$ for all activities $j = 1, \ldots, J$ and modes $m = 1, \ldots, M_j - 1$. Furthermore, activity j executed in mode m uses k_{jmr}^ρ units of renewable resource r each period it is in process, where we assume w. l. o. g. $k_{jmr}^\rho \leq K_r^\rho$ for each renewable resource $r \in R$. Note, otherwise activity j could not be performed in mode m. Moreover, it consumes k_{jmr}^ν units of nonrenewable resource $r \in N$. W. l. o. g., we assume that the dummy source and the dummy sink activity have only one mode each with a duration of zero periods and no request for any resource.

The objective is to minimize the makespan of the project. We assume the parameters to be nonnegative and integer valued. A mathematical programming formulation of this problem has been given by Talbot [22].

3 Enumeration Schemes

This section is devoted to enumeration procedures for the MRCPSP. In the first two subsections, we summarize the two algorithms that have been proposed in the literature for this problem. Then we present a new algorithm in Subsection 3.3 using a description which points out the similarities and differences to the former procedures.

3.1 The Precedence Tree

Patterson et al. [11] proposed an algorithm guided by the so-called precedence tree. Restructuring this approach, Sprecher [15] and Sprecher and Drexl [17] developed a new procedure based on the precedence tree and improved it by including new bounding criteria (cf. Section 4).

   We present a simplified formulation of the precedence tree algorithm. The procedure begins with starting the dummy source activity at time 0. At each level g of the branch-and-bound tree, we determine the set SJ_g of the currently scheduled activities and the set EJ_g of the eligible activities, that is, those activities the predecessors of which are already scheduled. Then we select an eligible activity j_g and, subsequently, a mode m_{j_g} of this activity. Now we compute the earliest precedence and resource feasible start time s_{j_g} that is not less than the start time assigned on the previous level of the search tree. Then we branch to the next level. If the dummy sink activity is eligible, we have found a complete schedule. In this case, backtracking to the previous level occurs. Here, we select the next untested mode. If none exists, we select the next untested eligible activity. If we have tested all eligible activities in all available modes, we track another step back. More formally, we have:

Algorithm 1 *(Precedence tree)*

Step 1: (Initialization)
   $g := 0$; $j_0 := 0$; $m_{j_0} := 1$; $s_{j_0} := 0$; $SJ_0 := \emptyset$;
Step 2: (Compute eligible activities)
   $g := g + 1$; $SJ_g := SJ_{g-1} \cup \{j_{g-1}\}$;
   $EJ_g := \{j \in \{1, \ldots, J+1\} \backslash SJ_g \mid P_j \subseteq SJ_g\}$;
   if $J + 1 \in EJ_g$ then store current solution and go to Step 5;
Step 3: (Select next activity)
   if no untested eligible activity is left in EJ_g then goto Step 5,
   else select untested activity $j_g \in EJ_g$;
Step 4: (Select next mode and compute start time)
   if no untested mode is left in $\{1, \ldots, M_{j_g}\}$ then goto Step 3,
   else select untested $m_{j_g} \in \{1, \ldots, M_{j_g}\}$;
   if a conflict w.r.t. a nonrenewable resource occurs then go to Step 4;
   compute earliest precedence and resource feasible start time s_{j_g}
   with $s_{j_g} \geq s_{j_{g-1}}$;
   goto Step 2;
Step 5: (Backtracking)
   $g := g - 1$; if $g = 0$ then STOP, else goto Step 4.

   Note that each combination of an eligible activity and a related mode corresponds to a descendant of the current node in the branch-and-bound tree or, as it is called here, precedence tree. Each branch from the root to a leaf of the precedence tree corresponds to a permutation of the set of activities $j_1, \ldots, j_J$ which is precedence feasible in the sense that each predecessor of a job j_g has a smaller index in the sequence than j_g.

3.2 Mode and Delay Alternatives

In this subsection we summarize the branch-and-bound approach proposed by Sprecher et al. [19]. Introducing the notion of a mode alternative, it extends the concept of delay alternatives used by Christofides et al. [3] and Demeulemeester and Herroelen [4] for the (single-mode) RCPSP.

In contrast to Algorithm 1, here each level g of the branch-and-bound tree is associated with a fixed time instant t_g (decision point) at which activities may be started. Consequently, we use a different definition of eligible activities in this algorithm: A currently unscheduled activity j is called eligible at time t_g if all of its predecessors $i \in P_j$ are scheduled with a finish time $f_i \leq t_g$. Furthermore, an activity j scheduled in mode m_j with start time s_j is said to be in process at time t_g if we have $s_j \leq t_g < s_j + d_{jm}$.

The proceeding at the current level g of the branch-and-bound tree is as follows: We determine the new decision point t_g as the earliest finish time of the activities currently in process. Note that, due to the constant availability levels of the renewable resources, only finish times of scheduled activities need to be considered for starting unscheduled ones. Using the set FJ_g of the activities that are finished at or before the decision point, we compute the set EJ_g of the eligible activities. Then we (temporarily) start those eligible activities at the decision point that have already been assigned a mode at a previous level of the search tree. If there are eligible jobs that have not yet been assigned a mode, that is, if $EJ_g \backslash EJ_{g-1}$ is not empty, then we compute the set $\mathcal{SOMA}_g$ of mode alternatives: A *mode alternative* is a mapping $\mathcal{MA}_g$ which assigns each activity $j \in EJ_g \backslash EJ_{g-1}$ a mode $\mathcal{MA}_g(j) = m_j \in \{1, \ldots, M_j\}$. Selecting a mode alternative, we can (temporarily) start the remaining eligible activities at the decision point as well. Having started all eligible activities by adding them to the set JIP_g of the activities in process, we may have caused a resource conflict. Thus, we compute the set $\mathcal{SODA}_g$ of the minimal delay alternatives according to the following definition: A *delay alternative* $\mathcal{DA}_g$ is a subset of JIP_g such that for each renewable resource $r \in R$ it is

$$\sum_{j \in JIP_g \backslash \mathcal{DA}_g} k^\rho_{jm_jr} \leq K^\rho_r.$$

A delay alternative $\mathcal{DA}_g$ is called *minimal* if no proper subset of $\mathcal{DA}_g$ is a delay alternative. We select a minimal delay alternative and remove the activities to be delayed from the current partial schedule. Note, if no resource conflict occurs, the only minimal delay alternative is the empty set. We store the start time t_g of an activity j to be delayed in s^{old}_{gj} because we have to restore the information during backtracking. Then we branch to the next level and compute the next decision point. If we have completed a schedule, we perform a backtracking step and test the next minimal delay alternative or, if all have been tested, the next mode alternative. Formally, the algorithm can be described as follows:

Algorithm 2 *(Mode and delay alternatives)*

Step 1: (Initialization)
$\quad$ $g := 0$; $t_0 := 0$; $JIP_0 := \{0\}$; $FJ_0 := \emptyset$; $m_0 := 1$; $s_0 := 0$; $EJ_0 := \emptyset$; $\mathcal{DA}_0 := \emptyset$;

Step 2: (Compute new decision point and eligible activities)
$\quad$ $g := g + 1$; $t_g := \min\{s_j + d_{jm_j} \mid j \in JIP_{g-1}\}$;
$\quad$ $FJ_g := FJ_{g-1} \cup \{j \in JIP_{g-1} \mid s_j + d_{jm_j} = t_g\}$;
$\quad$ $EJ_g := \{j \in \{1, \ldots, J+1\} \backslash (FJ_g \cup JIP_{g-1}) \mid P_j \subseteq FJ_g\}$;
$\quad$ $JIP_g := JIP_{g-1} \backslash FJ_g \cup EJ_g$;
$\quad$ if $J + 1 \in EJ_g$ then store current solution and go to Step 7;
$\quad$ for each $j \in \mathcal{DA}_{g-1}$ update $s_j := t_g$;

Step 3: (Compute mode alternatives)
$\quad$ if $EJ_g \backslash EJ_{g-1} = \emptyset$ then $\mathcal{SOMA}_g := \emptyset$ and go to Step 5,
$\quad$ else $\mathcal{SOMA}_g := SetOfModeAlternatives(EJ_g \backslash EJ_{g-1})$;

Step 4: (Select next mode alternative)
$\quad$ if no untested mode alternative is left in $\mathcal{SOMA}_g$ then go to Step 7,

else select untested $\mathcal{MA}_g \in \mathcal{SOMA}_g$;
for each $j \in EJ_g \backslash EJ_{g-1}$ update $m_j := \mathcal{MA}_g(j)$ and $s_j := t_g$;
if a conflict w.r.t. a nonrenewable resource occurs then go to Step 4;

Step 5: (Compute delay alternatives)
$\mathcal{SODA}_g := SetOfMinimalDelayAlternatives(JIP_g)$;

Step 6: (Select next delay alternative)
if no untested minimal delay alternative is left in $\mathcal{SODA}_g$ then go to Step 4,
else select untested $\mathcal{DA}_g \in \mathcal{SODA}_g$; $JIP_g := JIP_g \backslash \mathcal{DA}_g$;
for each $j \in \mathcal{DA}_g$ store $s_{gj}^{old} := s_j$; go to Step 2;

Step 7: (Backtracking)
$g := g - 1$; if $g = 0$ then STOP,
else for each $j \in \mathcal{DA}_g$ restore $s_j := s_{gj}^{old}$; $JIP_g := JIP_g \cup \mathcal{DA}_g$; go to Step 6.

Observe that each combination of a mode alternative and a related minimal delay alternative corresponds to a descendant of the current node in the branch-and-bound tree. Clearly, this procedure is different from Algorithm 1 in that sets of activities instead of (single) activities are started at each level of the branch-and-bound tree. Moreover, here the time instant at which activities may be started is determined before the activities themselves are selected. Finally, in contrast to Algorithm 1, this approach allows to withdraw scheduling decisions at the current level that have been made at a lower level.

3.3 Mode and Extension Alternatives

This subsection is devoted to a new branch-and-bound approach for solving the MRCPSP. Using again the concept of mode alternatives developed by Sprecher et al. [19], we introduce extension alternatives to construct partial schedules. A similar way to extend partial schedules has been proposed by Stinson et al. [21] for the single-mode case.

As in Algorithm 2, each level g of the branch-and-bound tree is associated with a decision point t_g, a set JIP_g of the activities in process, a set FJ_g of the finished activities, and a set EJ_g of the eligible activities. Again, we use a mode alternative to fix the modes of those eligible activities that have not yet been assigned a mode. Then we extend the current partial schedule by starting a subset of the eligible activities at the decision point without violating the renewable resource constraints. More precisely, an *extension alternative* $\mathcal{EA}_g$ is a subset of the eligible set for which we have

$$\sum_{j \in JIP_g \cup \mathcal{EA}_g} k_{jm_jr}^{\rho} \leq K_r^{\rho}$$

for each renewable resource $r \in R$ and, moreover, $\mathcal{EA}_g \neq \emptyset$ if $JIP_g = \emptyset$. Note, in order to secure that the algorithm terminates, we may only have nonempty extension alternatives if no activities are in process. However, if there are currently activities in process, the empty set is always an extension alternative which must be tested in order to guarantee optimality.

At the current level g of the branch-and-bound tree we proceed as follows: We determine the new decision point and compute the set of the eligible activities. Then we determine the set of mode alternatives $\mathcal{SOMA}_g$ for fixing the modes of the eligible activities that have not been eligible before, that is, those activities the modes of which have not yet been fixed. After selecting a mode alternative $\mathcal{MA}_g$, we compute the set of extension alternatives $\mathcal{SOEA}_g$. Finally, we select an extension alternative $\mathcal{EA}_g$ and start the corresponding activities before branching to the next level. The backtracking mechanism equals the one of Algorithm 2. Formally, we can describe the algorithm as follows:

Algorithm 3 *(Mode and extension alternatives)*

Step 1: (Initialization)

$g := 0;\ t_0 := 0;\ JIP_0 := \{0\}; FJ_0 := \emptyset;\ m_0 := 1;\ s_0 := 0;\ EJ_0 := \emptyset;$

Step 2: (Compute new decision point and eligible activities)

$g := g + 1;\ t_g := \min\{s_j + d_{jm_j} \mid j \in JIP_{g-1}\};$

$FJ_g := FJ_{g-1} \cup \{j \in JIP_{g-1} \mid s_j + d_{jm_j} = t_g\};$

$EJ_g := \{j \in \{1, \ldots, J+1\} \backslash (FJ_g \cup JIP_{g-1}) \mid P_j \subseteq FJ_g\};$

$JIP_g := JIP_{g-1} \backslash FJ_g;$

if $J + 1 \in EJ_g$ then store current solution and go to Step 7;

Step 3: (Compute mode alternatives)

if $EJ_g \backslash EJ_{g-1} = \emptyset$ then $\mathcal{SOMA}_g := \emptyset$ and go to Step 5,

else $\mathcal{SOMA}_g := SetOfModeAlternatives(EJ_g \backslash EJ_{g-1});$

Step 4: (Select next mode alternative)

if no untested mode alternative is left in $\mathcal{SOMA}_g$ then go to Step 7,

else select untested $\mathcal{MA}_g \in \mathcal{SOMA}_g;$

for each $j \in EJ_g \backslash EJ_{g-1}$ update $m_j := \mathcal{MA}_g(j);$

if a conflict w.r.t. a nonrenewable resource occurs then go to Step 4;

Step 5: (Compute extension alternatives)

$\mathcal{SOEA}_g := SetOfExtensionAlternatives(EJ_g, JIP_g);$

Step 6: (Select next extension alternative)

if no untested extension alternative is left in $\mathcal{SOEA}_g$ then go to Step 4,

else select untested $\mathcal{EA}_g \in \mathcal{SOEA}_g;\ JIP_g := JIP_g \cup \mathcal{EA}_g;$

for each $j \in \mathcal{EA}_g$ update $s_j := t_g;$ go to Step 2;

Step 7: (Backtracking)

$g := g - 1;$ if $g = 0$ then STOP, else $JIP_g := JIP_g \backslash \mathcal{EA}_g;$ go to Step 6.

Each combination of a mode alternative and a related extension alternative corresponds to a descendant of the current node in the branch-and-bound tree. Note that this procedure is different from Algorithm 2: Whereas the latter includes the possibility to delay activities that have been started on a lower than the current level, the new approach does not allow to withdraw a scheduling decision of a lower level. As a consequence, we may not restrict the search to "maximal" extension alternatives while we do not lose optimality when considering only minimal delay alternatives.

4 Bounding Rules

This section summarizes bounding criteria which speed up the enumeration procedures of the previous section. While most of the rules are known from the literature, we also present a new one and transfer some well-known ones to those enumeration schemes they have not yet been defined for. For the sake of shortness, we have omitted the proofs of those rules that are known from the literature.

4.1 Time Window Based Rules

The first bounding criterion makes use of time windows as determined by MPM. Sprecher [15] and Sprecher et al. [19] have employed this rule in their algorithms for solving the MRCPSP. Given the precedence relations and an upper bound on the makespan of the project (which is e.g. given by the sum of the maximal durations of the activities), we use the modes of shortest duration and derive the latest finish time LF_j for each activity j by traditional backward recursion. If a procedure has found the first or an improved schedule with a makespan T, the latest finish times are recalculated by $LF_j := LF_j - (LF_{J+1} - T + 1)$ for $j = 1, \ldots, J+1$. From the definition of the latest finish times we can derive the following bounding rule:

Bounding Rule 1 *(Basic Time Window Rule) If there is a scheduled activity the assigned finish time of which exceeds the latest finish time, then the current partial schedule cannot be completed*

with a makespan less than the best currently known.

Using the definition of the time window and explicitly considering multiple modes, Sprecher [15] has developed the following rule for the precedence tree algorithm:

Bounding Rule 2 *(Non-Delayability Rule for Algorithm 1) If an eligible activity cannot be feasibly scheduled in any mode in the current partial schedule without exceeding its latest finish time, then no other eligible activity needs to be examined on this level.*

Taking into account the differences between the precedence tree procedure on one hand and the algorithms based on mode alternatives on the other, we can adapt Bounding Rule 2 as follows:

Remark 1 *(Non-Delayability Rule for Algorithms 2 and 3) If an eligible activity the mode of which has not yet been fixed cannot be started in the mode with the shortest duration at the current decision point without exceeding its latest finish time, then no mode alternative needs to be examined at the current level.*

4.2 Preprocessing

This subsection is devoted to two bounding rules which can be implemented by preprocessing. The first one has originally been proposed by Sprecher et al. [19]. It uses the following definitions: A mode is called *non-executable* if its execution would violate the renewable or nonrenewable resource constraints in any schedule. A mode is called *inefficient* if its duration is not shorter and its resource requests are not less than those of another mode of the same activity. A nonrenewable resource is called *redundant* if the sum of the maximal requests of the activities for this resource does not exceed its availability. Clearly, non-executable and inefficient modes as well as redundant nonrenewable resources may be excluded from the project data without loosing optimality. Sprecher et al. [19] describe several interaction effects appearing when modes or nonrenewable resources are removed, e.g. eliminating a redundant nonrenewable resource may cause inefficiency of a mode. Hence, they propose the following way to prepare the input data:

Bounding Rule 3 *(Data Reduction) The project data can be adapted as follows:*
Step 1: Remove all non-executable modes from the project data.
Step 2: Delete the redundant nonrenewable resources.
Step 3: Eliminate all inefficient modes.
Step 4: If any mode has been erased within Step 3, go to Step 2.

The next bounding rule has especially been designed for instances with nonrenewable resources. It has been proposed by Drexl [5] for a less general framework.

Bounding Rule 4 *(Nonrenewable Resource Rule) If scheduling each currently unscheduled activity in the mode with the lowest request for a nonrenewable resource would exceed the capacity of this nonrenewable resource, then the current partial schedule cannot be feasibly completed.*

Sprecher [15] adapted the rule to the MRCPSP and improved the effect by reformulating it as a static rule. Before an algorithm is executed, the project data is adjusted as follows: Defining $k_{jr}^{\nu,min}$ as the minimal request of activity j for nonrenewable resource r, we update $k_{jmr}^{\nu} := k_{jmr}^{\nu} - k_{jr}^{\nu,min}$ for $j = 1, \ldots, J$, $m = 1, \ldots, M_j$, and $r \in N$, and

$$K_r^{\nu} := K_r^{\nu} - \sum_{j=1}^{J} k_{jr}^{\nu,min} \qquad \text{for } r \in N.$$

4.3   Dominating Sets of Schedules

The following three bounding rules make use of a classification of the set of schedules. The notions of semi-active and active schedules as formally defined by Sprecher et al. [20] for the single-mode case can be straightforwardly extended to the multi-mode case: A *left shift* of an activity within a given schedule is a reduction of its finish time without changing its mode and without changing the modes or finish times of the other activities, such that the resulting schedule is both precedence and resource feasible. A *local left shift* is a left shift which is obtainable by one or more successively applied left shifts of one period. A schedule is called *semi-active* if none of the activities can be locally left shifted. Following French [6], we can state that if there is an optimal schedule for a given instance, then there is an optimal semi-active schedule. This result is exploited by the following rule which has been employed by Sprecher [15] and Sprecher et al. [19] for the multi-mode case.

Bounding Rule 5 *(Local Left Shift Rule) If an activity that has been started at the current level of the branch-and-bound tree can be locally left shifted without changing its mode, then the current partial schedule needs not be completed.*

Additionally allowing a mode change of the activity to be shifted, Sprecher et al. [19] defined the notion of a multi-mode left shift: Within a given schedule, a *multi-mode left shift* is a reduction of an activity's finish time without changing the modes or finish times of the other activities, such that the resulting schedule is feasible. A schedule is called *tight* if no multi-mode left shift can be performed. The notion of tight schedules has been introduced by Speranza and Vercellis [14].

Another operation on a schedule of an MRCPSP instance has been introduced by Sprecher et al. [19]: A *mode reduction* on an activity is a reduction of its mode number without changing its finish time and without violating the constraints or changing the modes and finish times of the other activities. A schedule is called *mode-minimal* if there is no activity a mode reduction can be performed on.

Note that there are tight schedules which are not mode-minimal and vice versa. Obviously, if there is an optimal schedule for a given instance, then there is an optimal schedule which is both tight and mode-minimal. Consequently, the following rule proposed by Sprecher et al. [19] induces backtracking when it is certain that no tight or mode-minimal schedule can be obtained from the current partial schedule.

Bounding Rule 6 *(Multi-Mode Rule) Assume that no currently unscheduled activity will be started before the finish time of a scheduled activity j when the current partial schedule is completed. If a multi-mode left shift or a mode reduction of activity j with resulting mode m'_j, $1 \leq m'_j \leq M_j$, can be performed on the current partial schedule and, moreover, if $k^{\nu}_{jm'_jr} \leq k^{\nu}_{jm_jr}$ holds for each nonrenewable resource $r \in N$, then the current partial schedule needs not be completed.*

Clearly, if no multi-mode left shift can be applied, then a local left shift cannot be applied either. Nevertheless, it is useful to check for both types of left shifts separately according to the previous two bounding rules. Observe that we check for a local left shift when the corresponding activity has just been started. However, we can only check for a multi-mode left shift if the corresponding activity has already finished. Otherwise, as outlined by Hartmann and Sprecher [7], we would lose optimality. Consequently, the Local Left Shift Rule is not superflous as the exclusion of a partial schedule due to a feasible local left shift can be detected on a lower level of the branch-and-bound tree than the same (mode-preserving) multi-mode left-shift.

The next operation and the related category of schedules are new: Denoting the finish time of a scheduled activity j with $f_j = s_j + d_{jm_j}$, we consider two activities i and j with $i > j$ that are successively processed within a schedule, that is, $f_i = s_j$. Now an *order swap* is defined as the interchange of these two activities by assigning new start and finish times $s'_j := s_i$ and $f'_i := f_j$, respectively. Thereby, the precedence and resource constraints may not be violated, and the modes and start times of the other activities may not be changed. A schedule in which no

order swap can be performed is called *order monotonous*. Clearly, it is sufficient to enumerate only order monotonous schedules. It should be noted that there are schedules which are tight and mode-minimal but not order monotonous and vice versa. We apply the following bounding criterion:

Bounding Rule 7 *(Order Swap Rule) Consider a scheduled activity the finish time of which is less than or equal to any start time that may be assigned when completing the current partial schedule. If an order swap on this activity together with any of those activities that finish at its start time can be performed, then the current partial schedule needs not be completed.*

Proof. Obvious. □

In analogy to the extension of the left shift concept to the multi-mode case, the definition of the order swap can easily be generalized by allowing a mode change of the activities to be swapped. However, preliminary computational results have shown that the additional effort that would be necessary to check the assumptions completely consumes the acceleration effect.

4.4   The Cutset Rule

The following bounding method stores information about already evaluated partial schedules. During the search process, the rule compares the current partial schedule with the stored data. If it can be proven that any solution obtainable from the current partial schedule cannot be better than a solution obtainable from a previously evaluated partial schedule the information of which has been stored, then backtracking may be performed.

Bounding criteria based on stored information of already evaluated partial schedules have been employed by Stinson et al. [21] and Demeulemeester and Herroelen [4] for the single-mode case. Defining a *cutset* of a partial schedule PS as the set of the activities scheduled in PS, Sprecher and Drexl [17] proposed the following rule for their algorithm for the MRCPSP:

Bounding Rule 8 *(Cutset Rule for Algorithm 1) Let $\overline{PS}$ denote a previously evaluated partial schedule with cutset $CS(\overline{PS})$, maximal finish time $f^{max}(\overline{PS})$ and leftover capacities $K_r^\nu(\overline{PS})$ of the nonrenewable resources $r \in N$. Let PS be the current partial schedule considered to be extended by scheduling some activity j with start time s_j. If we have $CS(PS) = CS(\overline{PS})$, $s_j \geq f^{max}(\overline{PS})$ and $K_r^\nu(PS) \leq K_r^\nu(\overline{PS})$ for all $r \in N$, then PS needs not be completed.*

When all continuations of the current partial schedule have been examined, the cutset information related to the partial schedule that is required for Bounding Rule 8 is stored.

If the concept of mode alternatives is used, the rule has to be adapted. Clearly, each scheduling decision made in the current partial schedule has to be reflected in the data to be stored. Having selected an extension alternative in Algorithm 3, the modes of some activities that are not contained in the current partial schedule may be fixed within each of its continuations. Consequently, we must store the set of those activities the modes of which are fixed and the related modes in addition to the data that is stored according to Bounding Rule 8 for the precedence tree procedure. The cutset rule proposed by Demeulemeester and Herroelen [4] can be generalized to the multi-mode case in a similar way and can then be employed in Algorithm 2. Unfortunately, however, adapting the Cutset Rule to Algorithms 2 and 3 does not speed up these procedures. Roughly speaking, this is due to the fact that we have to store much more data while each cutset information unit is less general when the concept of mode alternatives is used. That is, the effort of storing and comparing the data increases while backtracking due to some stored information becomes less probable. Therefore, we do not give the detailed formal descriptions of the variants of Bounding Rule 8 for the procedures based on mode alternatives.

4.5 Immediate Selection

The following bounding rule has been developed by Demeulemeester and Herroelen [4] for the RCPSP and generalized by Sprecher et al. [19] to the multi-mode case. It states assumptions under which we are allowed to consider only one branching alternative instead of testing all. We first give a formulation that can be employed if the decision point and mode alternative concepts are used.

Bounding Rule 9 *(Immediate Selection for Algorithms 2 and 3) We assume the following situation: All activities that start before the current decision point t_g finish at or before t_g. After selecting a mode alternative, there is an eligible activity j with fixed mode m_j which cannot be simultaneously processed with any other eligible activity i in its fixed mode m_i. Moreover, activity j in mode m_j cannot be simultaneously processed with any unscheduled activity h in any mode $m_h \in \{1, \ldots, M_h\}$. Then $\mathcal{DA}_g = JIP_g \backslash \{j\} (= EJ_g \backslash \{j\})$ is the only minimal delay alternative that has to be examined, and $\mathcal{EA}_g = \{j\}$ is the only extension alternative that has to be examined.*

This rule can be adapted to the precedence tree guided enumeration procedure in several ways. We consider the following variant:

Remark 2 *(Immediate Selection for Algorithm 1) Consider an eligible activity j no mode of which is simultaneously performable with any currently unscheduled activity in any mode. If the earliest feasible start time of each other eligible activity in any mode is equal to the maximal finish time of the currently scheduled activities, then j is the only eligible activity that needs to be selected for being scheduled on the current level of the branch-and-bound tree.*

As described by Demeulemeester and Herroelen [4] for the single-mode and Sprecher et al. [19] for the multi-mode case, a similar immediate selection strategy for scheduling two activities and delaying all other eligible activities can be stated. Preliminary computational results, however, revealed that this rule does not speed up the algorithm when the other rules are employed. Consequently, we do not consider it here.

4.6 A Precedence Tree Specific Rule

Due to the construction of the precedence tree, Algorithm 1 may enumerate one schedule several times. This is the case in the following situation: Consider some partial schedule PS which is extended by scheduling some activity i in mode m_i on level $g-1$ and activity j in mode m_j on level g with identical start times $s_i = s_j$. If we return to PS later in the search process, and if scheduling activity j in mode m_j on level $g-1$ and activity i in mode m_i on level g results in the same start times, then we will obtain a schedule that has previously been enumerated. To avoid duplicate consideration of a schedule, Sprecher [15] has proposed the so-called Single Enumeration Rule which uses a three-dimensional array to check the assumptions mentioned above. We present an alternative rule to exclude duplicate enumeration. Clearly, it can only be used within Algorithm 1 since in the other two procedures a schedule can only be considered once.

Bounding Rule 10 *(Precedence Tree Rule for Algorithm 1) Consider two activities i and j scheduled on the previous and on the current level of the branch-and-bound tree, respectively. If we have $s_i = s_j$ and $i > j$ then the current partial schedule needs not be completed.*

Proof. Let PS be a partial schedule extended by scheduling activities i and j on levels $g-1$ and g, respectively, with $i > j$ and $s_i = s_j$. We assume that extending PS by scheduling j before i, both in the same modes as before, results in start times s'_j and s'_i. Clearly, we have $s'_i = s_i$ and $s'_j \leq s_j$. Thus, extending PS by scheduling i before j cannot lead to a schedule with a shorter makespan than by scheduling j before i. It should be observed that the extension of PS obtained from scheduling j before i cannot be excluded by Bounding Rule 10. $\square$
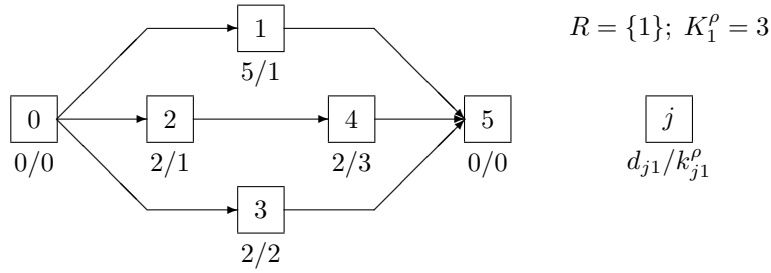
Figure 1: Project Instance

The new rule is not only simpler, but also more general than the original Single-Enumeration Rule in that it additionally contains a portion of the Local Left Shift Rule. This can be seen in the proof given above: If we have $s'_j < s_j$, then the Local Left Shift Rule would also induce backtracking. Nevertheless, the Local Left Shift Rule is still necessary as the Precedence Tree Rule does not exclude partial schedules that are not semi-active if we have $i < j$.

5   Theoretical Comparison of Enumerated Schedules

5.1   Complete Enumeration

In this subsection we compare the sets of schedules enumerated by the algorithms described in Section 3 without considering any of the bounding rules of Section 4. We show that the three branching procedures differ in the related sets of enumerated schedules. For notational convenience, we will refer to the sets of schedules enumerated by Algorithms 1, 2, and 3 with $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$, respectively.

We start our investigation comparing Algorithms 1 and 2. The first theorem states that for some instances schedules that are enumerated by the precedence tree algorithm are not enumerated by the algorithm based on mode and delay alternatives and vice versa.

Theorem 1 *There are instances for which we have $\mathcal{S}_1 \nsubseteq \mathcal{S}_2$ and $\mathcal{S}_2 \nsubseteq \mathcal{S}_1$.*

Proof. We consider the project instance given in Figure 1 as a counterexample. Note that it is a single-mode instance (thus, the mode index and the set of the nonrenewable resources have been omitted). Consequently, the results obtained hold for the single-mode RCPSP as well. It can be easily verified that the schedule shown in Figure 2 (a) is enumerated by Algorithm 1 but not by Algorithm 2. Schedule (b) is enumerated by Algorithm 2 but not by Algorithm 1. $\square$

Next, we compare Algorithms 1 and 3. For each instance, any schedule enumerated by the precedence tree algorithm is also found by the algorithm based on mode and extension alternatives. The reverse, however, does not hold in general.

Theorem 2 *There are instances with $\mathcal{S}_3 \nsubseteq \mathcal{S}_1$, but for all instances it is $\mathcal{S}_1 \subseteq \mathcal{S}_3$.*

Proof. Again, we use the instance displayed in Figure 1 as a counterexample: Schedule (b) of Figure 2 is enumerated by Algorithm 3 but not by Algorithm 1, proving the first part of the theorem.

We consider a partial schedule PS_1 enumerated by Algorithm 1 which is subsequently extended to $\overline{PS}_1 = PS_1 \cup \{(j, m_j, s_j)\}$. Assuming that Algorithm 3 finds a partial schedule PS_3 equal to PS_1, we have to show that it also enumerates a partial schedule $\overline{PS}_3$ equal to $\overline{PS}_1$. Let t be the last decision point in PS_3 at which a nonempty extension alternative $\mathcal{EA}$ has been scheduled. We have $t \leq s_j$ (otherwise s_j would not be a start time considered by Algorithm 1). If we have $t = s_j$,
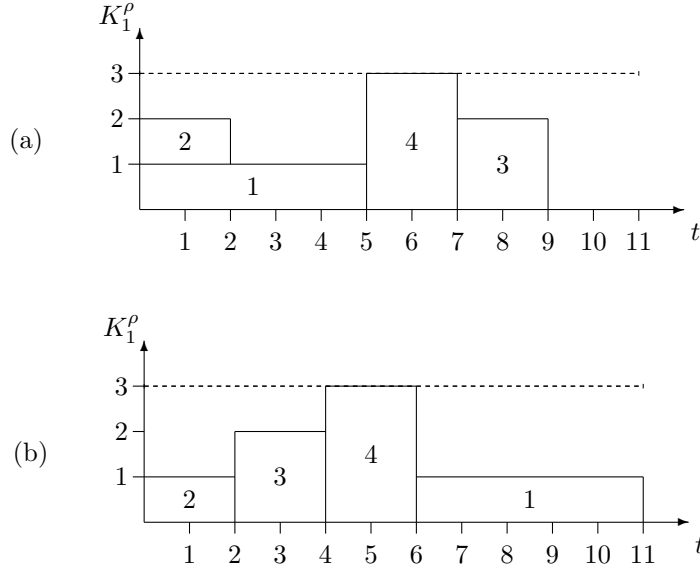
Figure 2: Schedules of the Project Instance

we define $\overline{PS_3}$ as the result of scheduling $\overline{\mathcal{EA}} := \mathcal{EA} \cup \{j\}$ at time t instead of $\mathcal{EA}$. Note that $\overline{\mathcal{EA}}$ is a feasible extension alternative. Otherwise, we extend PS_3 by scheduling the empty extension alternative at all decision points t' with $t < t' < s_j$ (if any). Then we obtain $\overline{PS_3}$ by scheduling extension alternative $\{j\}$ at time s_j which is a decision point (i.e., a finish time of an activity scheduled in PS_3). In both cases, mode m_j can be chosen using a mode alternative, and we have $\overline{PS_3} = \overline{PS_1}$. $\square$

Finally, we compare Algorithm 2 to Algorithm 3. Given an arbitrary instance of the MRCPSP, any schedule enumerated using mode and delay alternatives is also found by using mode and extension alternatives. The inversion does not hold for some instances.

Theorem 3 *There are instances with $\mathcal{S}_3 \nsubseteq \mathcal{S}_2$, but for all instances it is $\mathcal{S}_2 \subseteq \mathcal{S}_3$.*

Proof. Considering again the instance shown in Figure 1, schedule (a) of Figure 2 proves the first part of the theorem as it is enumerated by Algorithm 3 but not by Algorithm 2.

As both algorithms employ the concept of mode alternatives, we may restrict the proof of the second part of the theorem to the single-mode case. We consider an arbitrary project instance and a partial schedule enumerated by both algorithms, that is, $PS_2 = PS_3$. Let t_{g+1} be the next decision point and EJ the set of the eligible activities in both partial schedules (note that the definitions of a decision point and eligible activities are equal in both algorithms). Algorithm 2 schedules the eligible jobs at time t_{g+1} and delays the activities of some minimal delay alternative $\mathcal{DA}$, resulting in partial schedule $\overline{PS_2}$. We have to show that Algorithm 3 finds $\overline{PS_2}$, too. We assume that PS_3 is constructed by a sequence $(t_0, \mathcal{EA}_0), \ldots, (t_g, \mathcal{EA}_g)$ of decision points and extension alternatives. Note that the decision points in PS_2 and PS_3 are equal. We set $A_i := \{j \in \mathcal{DA} \mid s_j = t_i \text{ in } \overline{PS_2}\}$ for $i = 0, \ldots, g+1$ and have $\mathcal{DA} = \cup_{i=0}^{g+1} A_i$. Now we define $\overline{\mathcal{EA}}_{g+1} := EJ \backslash A_{g+1}$ and $\overline{\mathcal{EA}}_i := \mathcal{EA}_i \backslash A_i$ for $i = 0, \ldots, g$. Observe that the sequence of decision points is not affected as all delayed activities have a finish time greater than t_{g+1}. Moreover, each $\overline{\mathcal{EA}}_i$ with $0 \leq i \leq g+1$ is a feasible extension alternative. Hence the extension algorithm enumerates a sequence $(t_0, \overline{\mathcal{EA}}_0), \ldots, (t_{g+1}, \overline{\mathcal{EA}}_{g+1})$ which corresponds to a partial schedule $\overline{PS_3}$. With $\overline{PS_3} = \overline{PS_2}$ by construction we complete the proof. $\square$

The results of this subsection can be summarized as follows: None of the complete enumeration schemes currently available for the MRCPSP is dominant in a sense that the set of the schedules enumerated by one algorithm is less than or equal to the sets enumerated by the other two procedures. However, combining the theorems above, we have

$$\mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathcal{S}_3.$$

5.2 Enumeration with Bounding Rules

In this subsection, we examine the sets of schedules enumerated by the algorithms of Section 3 including the bounding rules of Section 4. The focus will be on the Local Left Shift Rule (Bounding Rule 5) since this allows us to characterize the enumerated set of schedules with respect to the set of semi-active schedules. Similarly to the previous subsection, we denote the sets of schedules enumerated by Algorithms 1, 2, and 3 with the Local Left Shift Rule as $\mathcal{S}_1^{LS}$, $\mathcal{S}_2^{LS}$, and $\mathcal{S}_3^{LS}$, respectively. Finally, the set of the semi-active schedules is referred to as $\mathcal{SAS}$.

We start our investigation with an analysis of the impact of the Local Left Shift Rule. The following theorem states that the precedence tree algorithm in accordance with the Local Left Shift Rule enumerates exactly the set of the semi-active schedules.

Theorem 4 *For all instances, we have $\mathcal{S}_1^{LS} = \mathcal{SAS}$.*

Proof. The Local Left Shift Rule is applied to activity j that is started at time s_j at the current node of the branch-and-bound tree. No scheduling decision at a successor node of the search tree can free renewable resources before s_j, thus, if a local left shift is not possible when starting an activity, it is not possible in any (enumerated) completion of the partial schedule.

Let $S \in \mathcal{SAS}$ be a semi-active schedule for an arbitrary instance. We can construct a sequence $(j_0, m_0, s_0), \ldots, (j_i, m_i, s_i), \ldots, (j_J, m_J, s_J)$ with $s_i \leq s_{i+1}$ for $1 \leq i < J$ representing S. We show by induction that the sequence related to S corresponds to a branch in the search tree built up by Algorithm 1. Let $(j_0, m_0, s_0), \ldots, (j_g, m_g, s_g)$ with $1 \leq g < J$ correspond to a partial schedule PS_1^{LS} found by Algorithm 1. We consider activity j_{g+1} which is eligible in PS_1^{LS} and can therefore be scheduled in mode m_{g+1}. Let t denote the start time assigned to activity j_{g+1} by Algorithm 1 and let $\overline{PS}_1^{LS} = PS_1^{LS} \cup \{(j_{g+1}, m_{g+1}, t)\}$ be the corresponding next partial schedule. We have $t \geq s_{g+1}$, otherwise S could not be semi-active. For the same reason, the left shift rule cannot be applied to activity j_{g+1}. Moreover, it is $t \leq s_{g+1}$ because the precedence tree algorithm assigns the earliest feasible start time and it is $s_g \leq s_{g+1}$. Hence, we deduce $t = s_{g+1}$, that is, $(j_0, m_0, s_0), \ldots, (j_{g+1}, m_{g+1}, s_{g+1})$ corresponds to partial schedule $\overline{PS}_1^{LS}$. $\square$

The next theorem shows that an analogous result cannot be obtained for the algorithm based on mode and delay alternatives including the left shift rule: This one may enumerate schedules which are not semi-active while on the other hand there may exist semi-active schedules which are not enumerated.

Theorem 5 *There are instances for which we have $\mathcal{S}_2^{LS} \not\subseteq \mathcal{SAS}$ and $\mathcal{SAS} \not\subseteq \mathcal{S}_2^{LS}$.*

Proof. We consider the instance shown in Figure 1. Schedule (b) of Figure 2 is not semi-active, but it is enumerated by Algorithm 2 with Local Left Shift Rule: At time 0, activities 1, 2, and 3 are started. Since a resource conflict occurs, we may select $\{3\}$ as minimal delay alternative. At time 2, activities 3 and 4 are started. This resource conflict may be solved by delaying activity 4 which is then rescheduled at time 4. The resulting resource conflict can be solved delaying activity 1. According to the formulation of the Local Left Shift Rule, only activity 4 is tested for a left shift. However, activity 3 may now be locally left shifted to time 0 due to the delay of activity 1. This possible local left shift is not detected by the Local Left Shift Rule. Consequently, activity 1 is started at time 6 completing the (non semi-active) schedule.

Now we consider schedule (a) of Figure 2 which is semi-active. However, it is not enumerated by Algorithm 2 (no matter whether the Local Left Shift Rule is included or not): Starting activities 1, 2, and 3 at time 0, delaying activity 3, and starting activities 3 and 4 at time 2 causes a resource conflict at time 2. It can be solved by the only minimal delay alternatives $\{1,3\}$ and $\{4\}$. None of them will result in schedule (a). $\square$

Theorem 5 states that the Local Left Shift Rule considered here does not prevent the algorithm based on mode and delay alternatives from enumerating schedules which are not semi-active. Note that our formulation of the left shift rule is equivalent to the one used by Demeulemeester and Herroelen [4] for the single-mode case, that is, this observation holds for their procedure as well. Clearly, a possible enumeration of schedules which are not semi-active is due to the delay of activities which start before the previous decision point. Freeing resources before the previous decision point may induce the possibility of a left shift of an activity which starts at the previous (or an earlier) decision point. Such a left shift cannot be detected by this version of the Local Left Shift Rule. However, the rule can be extended to exclude all schedules which are not semi-active:

Remark 3 *(Extended Local Left Shift Rule for Algorithm 2) Let s denote the minimal start time of those activities currently selected to be delayed, that is, $s = \min\{s_j \mid j \in \mathcal{DA}_g\}$. If there is a scheduled activity with a start time greater than s which is not selected to be delayed, and if this activity can be locally left shifted after delaying the currently selected delay alternative, then the current partial schedule needs not be completed.*

With the arguments given above, we can state that the Extended Local Left Shift Rule prevents the algorithm based on mode and delay alternatives from enumerating a schedule which is not semi-active. Thus, denoting the set of schedules enumerated by Algorithm 2 with the Extended Local Left Shift Rule as $\mathcal{S}_2^{ELS}$, the result of Theorem 5 can be improved as follows:

Remark 4 *For all instances, it is $\mathcal{S}_2^{ELS} \subseteq \mathcal{SAS}$, but there are instances for which $\mathcal{SAS} \not\subseteq \mathcal{S}_2^{ELS}$ holds.*

Now we turn to the algorithm based on mode and extension alternatives for which we can obtain the same result as for the precedence tree procedure. When combined with the Local Left Shift Rule, also Algorithm 3 enumerates exactly the set of the semi-active schedules of a given instance.

Theorem 6 *For all instances, we have $\mathcal{S}_3^{LS} = \mathcal{SAS}$.*

Proof. The Local Left Shift Rule is applied to the activities started at the current decision point. As no renewable resources are freed before this decision point when the corresponding partial schedule is completed, the application of the Local Left Shift Rule excludes all schedules which are not semi-active, that is, we have $\mathcal{S}_3^{LS} \subseteq \mathcal{SAS}$.

Using Theorem 4, we have $\mathcal{SAS} = \mathcal{S}_1^{LS}$. Clearly, it is $\mathcal{S}_1^{LS} \subseteq \mathcal{S}_1$. Furthermore we have $\mathcal{S}_1 \subseteq \mathcal{S}_3$ by Theorem 2. Consequently it is $\mathcal{SAS} \subseteq \mathcal{S}_3$. Note that a feasible left shift of a currently started activity is possible in any continuation of the current partial schedule. That is, it cannot be prevented by further scheduling decisions as these do not affect the resource usages before the start time of that activity. Hence the Local Left Shift Rule does not exclude schedules that are not semi-active, and we deduce $\mathcal{SAS} \subseteq \mathcal{S}_3^{LS}$. $\square$

Combining Theorems 4 and 6, we can state that the precedence tree algorithm and the procedure based on mode and extension alternatives both enumerate the same set of schedules when combined with the Local Left Shift Rule, that is, the set of the semi-active schedules. Furthermore, it should be noted that these two theorems can also be used to prove the correctness of Algorithms 1 and 3 since we can find an optimal semi-active schedule for an instance if we can find

an optimal one. Additionally considering Remark 4, we can summarize the results obtained so far as follows:

$$\mathcal{S}_2^{ELS} \subseteq \mathcal{S}_1^{LS} = \mathcal{S}_3^{LS} = \mathcal{S}\mathcal{A}\mathcal{S}.$$

Next, we briefly consider the remaining bounding rules of Section 4. The impact of the first four rules is identical within all branching schemes, that is, including them does not change the relationships stated in Subsection 5.1. This can be explained as follows: The Basic Time Window Rule (Bounding Rule 1) prevents any procedure from completing a schedule with a makespan that is not shorter than the best found so far. If the order in which the schedules are enumerated is the same in the three procedures (which can be achieved by an appropriate branching order), the effect is independent from the specific enumeration scheme. The Non-Delayability Rule (Bounding Rule 2) does not exclude schedules if used together with Bounding Rule 1, it only induces backtracking on lower levels of the branch-and-bound tree. The Data Reduction Rule (Bounding Rule 3) leads to the exclusion of those schedules that contain redundant modes, which is independent from the branching scheme. The Nonrenewable Resource Rule (Bounding Rule 4) does not exclude schedules, it aims at an early detection of infeasible schedules.

We now discuss those two of the remaining rules which make use of dominating sets of schedules. Within Algorithms 1 and 3, the Order Swap Rule (Bounding Rule 7) restricts the enumeration to the set of the order monotonous schedules, while the Multi Mode Rule (Bounding Rule 6) reduces the enumeration to the tight and mode-minimal schedules only if no nonrenewable resources are given. For Algorithm 2 the exclusion of all schedules which are not order monotonous, tight, or mode-minimal can only be obtained if the corresponding tests are performed on those activities which finish at or before the current decision point (cf. the above discussion of the Local Left Shift Rule).

Finally, the Immediate Selection Rule (Bounding Rule 9) has the same effect within the decision point based Algorithms 2 and 3, that is, it does not change the relationship given in Theorem 2. For the formulation of this rule for Algorithm 1 (see Remark 2), an analogous statement cannot be made. The Cutset Rule (Bounding Rule 8), and for obvious reasons also the Precedence Tree Rule (Bounding Rule 10), have only been defined for Algorithm 1 and, therefore, need not be considered in our comparison.

We close this section remarking that although the theoretical results derived here provide a deeper insight into the different solution methodologies, they alone do not allow to predict the solution times required by the algorithms. This is due to the fact that the different operations the procedures consist of may result in different computation times even if the same set of schedules is enumerated. Moreover, the effect of a bounding rule may depend on the algorithmic structure, that is, one algorithm may be accelerated less than another one, cf. the discussion of the different variants of the Cutset Rule. Consequently, the theoretical comparison of this section is completed by the computational comparison provided in the following section.

6   Computational Results

6.1   Experimental Design

In this section we present the results of the computational studies concerning the algorithms discussed in the previous sections. The experiments have been performed on a Pentium-based IBM-compatible personal computer with 133 MHz clock-pulse and 32 MB RAM. The procedures have been coded in ANSI C, compiled with the GNU C compiler and tested under Linux. In order to provide a fair comparison of the algorithms, we have attempted to use identical data structures and related update operations whenever possible. Moreover, we have used the same level of implementational know-how such as use of pointer arithmetics for coding the algorithms. Finally, in contrast to the comparison of Algorithms 1 and 2 provided by Sprecher et al. [19], we have attempted to integrate each bounding criterion of Section 4 into each enumeration procedure (with the exception of the Precedence Tree Rule).

Parameter	levels			
J	10	12	14	16
RF_R	0.50	1.00		
RS_R	0.25	0.50	0.75	1.00
RF_N	0.50	1.00		
RS_N	0.25	0.50	0.75	1.00

Table 1: Variable parameter levels under full factorial design

We used a set of test problems constructed by the project generator ProGen which has been developed by Kolisch et al. [9]. The instances have been used to evaluate the precedence tree algorithm by Sprecher and Drexl [18] as well as the procedure based on mode and delay alternatives by Sprecher et al. [19]. They are available in the project scheduling problem library PSPLIB from the University of Kiel. For detailed information the reader is referred to Kolisch and Sprecher [8].

In our study, we have used the multi-mode problem sets containing instances with 10, 12, 14, and 16 non-dummy activities. Each of the non-dummy activities may be performed in one out of three modes. The duration of a mode varies between 1 and 10 periods. We have two renewable and two nonrenewable resources. For each problem size, a set of instances was generated by systematically varying four parameters, that is, the resource factor and the resource strength of each resource category. The resource factor is a measure of the average portion of resources requested per job. The resource strength reflects the scarceness of the resources. Table 1 displays the variable parameter levels. The resource factors of the renewable and nonrenewable resources are referred to as RF_R and RF_N, respectively. The resource strengths of the renewable and nonrenewable resources are denoted as RS_R and RS_N, respectively. For each problem size and each combination of the resource parameters, ten instances have been generated. Consequently, we have 640 instances for each project size. Those instances for which no feasible solution exists have not been considered. Hence, we have 536 instances with $J = 10$, 547 instances with $J = 12$, 551 instances with $J = 14$, and 550 instances with $J = 16$.[1]

6.2   Effects of the Bounding Rules

As the impact of most of the acceleration methods on the computation times has been thoroughly studied by Sprecher and Drexl [18] as well as Sprecher et al. [19], we only summarize some new insights. The new Order Swap Rule (Bounding Rule 7) accelerates the basic variant of Algorithm 2 (including only the Time Window Rule) by a factor of approximately 1.9. This effect is not totally consumed when the other rules are also included. As already mentioned, none of the tested variants of the Cutset Rule for Algorithms 2 and 3 could accelerate these procedures when the other bounding schemes were employed. However, as reported by Sprecher and Drexl [18], the Cutset Rule can be efficiently used within the precedence tree algorithm. The immediate selection strategy of Bounding Rule 9 accelerates the branching schemes when applied to small instances ($J = 10$), confirming the results obtained by Sprecher et al. [19]. However, it may slow down the procedures if instances with more activities are considered. This is due to the fact that it becomes less probable that the assumptions can be fulfilled while the effort to check them increases with an increasing number of activities. The new formulation of the precedence tree specific rule (Bounding Rule 10) accelerates the basic variant of Algorithm 1 (including the Time Window Rule) by a factor of 8.4 while Sprecher and Drexl [18] report a factor of 3.2 for their formulation. This is mainly due to the fact that the new variant includes a portion of the Local Left Shift Rule. Finally, the Extended Local Left Shift Rule for the algorithm based on mode and delay alternatives (cf. Remark 3 in

---

[1]Due to the history of the project scheduling problem library, some of the parameter settings used to generate the instances with 10 non-dummy activities slightly differ from those given above that have been used to generate the other problems. For more details on the parameter settings cf. Kolisch and Sprecher [8].

Section 5.2) is of rather theoretical interest as it does not yield further acceleration of Algorithm 2.

For the comparison to be summarized in the next subsection we have used the fastest variants of the algorithms. Considering the observations given above, all bounding schemes except for the Cutset Rule, the Immediate Selection Rule, and the Precedence Tree Rule have been included in Algorithms 2 and 3. Clearly, the Cutset Rule as well as the Precedence Tree Rule have been employed in Algorithm 1, omitting only the Immediate Selection Rule. In order to seperate the effect of the Cutset Rule, we have also tested a variant of Algorithm 1 in which the former is not included. The variants of the procedures are summarized in Table 2 where '+' indicates that the corresponding bounding rule is included and '−' means that it is not.

6.3 Comparison of the Algorithms

We start the summary of our numerical results with a comparison of the average computation times given in Table 3. Algorithm 1 with the Cutset Rule is the fastest procedure on the average. It is 2.0 times faster than Algorithm 2 when 10 activities are considered and 7.0 times for projects with 16 activities, that is, the comparison factor increases with an increasing number of jobs. Algorithm 2 is at most 1.4 times faster than Algorithm 3. The precedence tree algorithm is faster than the other two procedures even if the Cutset Rule is not included.

Table 4 shows that Algorithm 1 has the lowest maximal computation times, no matter if the Cutset Rule is employed or not. For two project sizes, the maximal computation times of Algorithm 2 are lower than those of Algorithm 3. In the other two cases, the reverse holds.

Next, we have examined the impact of the resource factor and strength of the renewable resources on the computation times for $J = 16$. The results, summarized in Table 5, show that Algorithm 1 is the fastest procedure for the so-called hard instances with high computation times, that is, if the resource factor is high or the resource strength is low. However, on the easiest instances with a high resource strength Algorithm 2 performs best. This indicates that none of the procedures is dominant in the sense that it is faster than the other two on each instance.

Finally, the distributions of the computation times are listed in Table 6. Algorithm 1 solves 21.5 % of the instances with 16 activities in less then 0.01 seconds while Algorithm 2 solves 23.8 % within this time. On the other hand, Algorithm 2 cannot solve 0.5 % in 1000 seconds while Algorithm 3 fails to solve only 0.2 % within this time.

Summing up the observations above, the new approach based on mode and extension alternatives is outperformed by the other two algorithms with respect to average computation times. This seems to be due to the fact that, as already outlined, branching may not be restricted to "maximal" extension alternatives. This drawback cannot be fully compensated by the Local Left Shift Rule. The procedure based on mode and delay alternatives is the fastest on the easy instances. However, it is outperformed by the precedence tree guided algorithm on the hard instances, and even by the new approach on some hard instances. This is due to the possibility of cancelling previous scheduling decisions by delaying activities: On one branch of the search tree, one activity may be delayed and rescheduled several times. Clearly, the lower the renewable resource strength, the more activities have to be delayed due to the scarceness of the renewable resources, resulting in a high computational effort for this problem class. The precedence tree approach is the fastest with respect to average and maximal computation times. Its main disadvantage, the duplicate enumeration of a schedule, is neutralized by the new efficient precedence tree specific Bounding Rule 10. Moreover, it currently is the only procedure in which an efficient variant of the powerful Cutset Rule can be employed.

7 Conclusions

We have analyzed the branch-and-bound concepts currently available for solving resource-constrained project scheduling problems with multiple modes. The three algorithms, two from

Algorithm	basic scheme	1	2	3	4	5	6	7	8	9	10
1 (a)	precedence tree	+	+	+	+	+	+	+	+	−	+
1 (b)	precedence tree	+	+	+	+	+	+	+	−	−	+
2	mode and delay alt.	+	+	+	+	+	+	+	−	−	−
3	mode and extension alt.	+	+	+	+	+	+	+	−	−	−

Table 2: Accelerated variants of the algorithms to be tested

Algorithm	$J = 10$	$J = 12$	$J = 14$	$J = 16$
1 (a)	0.04	0.12	0.75	3.26
1 (b)	0.05	0.20	1.66	10.60
2	0.08	0.33	4.55	22.81
3	0.11	0.45	4.86	28.08

Table 3: Average computation times (sec) — all instances

Algorithm	$J = 10$	$J = 12$	$J = 14$	$J = 16$
1 (a)	0.77	2.69	22.87	165.11
1 (b)	1.25	5.14	78.91	1601.81
2	2.96	17.29	709.37	4523.44
3	2.87	20.57	529.92	6043.12

Table 4: Maximal computation times (sec) — all instances

Algorithm	RF_R:	0.50	1.00	RS_R:	0.25	0.50	0.75	1.00
1 (a)		0.83	5.58		8.34	2.90	1.20	0.58
1 (b)		1.30	19.51		34.81	5.32	1.77	0.68
2		1.35	43.37		81.24	9.17	1.13	0.23
3		4.07	51.08		94.63	11.75	4.80	1.72

Table 5: Average computation times for resource classes (sec) — $J = 16$

Algorithm	< 0.01	< 0.1	< 1	< 10	< 100	< 1000	< 10000
1 (a)	21.5	43.5	70.2	92.4	99.6	100.0	100.0
1 (b)	21.6	41.8	67.1	90.2	97.8	99.8	100.0
2	23.8	42.3	70.1	88.1	96.8	99.5	100.0
3	16.5	33.4	58.1	82.1	96.5	99.8	100.0

Table 6: Distribution of the computation times (%) — $J = 16$

the literature and one new approach, have been described in a unified framework and accelerated with ten bounding criteria one of which is also new. Subsequently, the procedures have been compared both theoretically and numerically. In our experiments based on a standard set of more than 2000 instances, the precedence tree approach by Sprecher and Drexl [17] outperformed the other two algorithms with respect to the average and maximal computation times. It seems to be well suited to solve hard instances. The procedure based on mode and delay alternatives suggested by Sprecher et al. [19] was shown to be the fastest when applied to easy instances. Furthermore, according to our experience, the precedence tree algorithm seems to be easier to implement as at each node of the branch-and-bound tree (single) activities are scheduled instead of sets of activities. Hence, we conclude that the precedence tree guided enumeration scheme currently is the algorithm of choice when solving larger project instances.

References

[1] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.

[2] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch-and-bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288, 1998.

[3] N. Christofides, R. Alvarez-Valdes, and J. M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.

[4] E. L. Demeulemeester and W. S. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

[5] A. Drexl. Scheduling of project networks by job assignment. *Management Science*, 37:1590–1602, 1991.

[6] S. French. *Sequencing and scheduling: An introduction to the mathematics of the job-shop*. Wiley, New York, 1982.

[7] S. Hartmann and A. Sprecher. A note on "Hierarchical models for multi-project planning and scheduling". *European Journal of Operational Research*, 94:377–383, 1996.

[8] R. Kolisch and A. Sprecher. PSPLIB – a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

[9] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703, 1995.

[10] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.

[11] J. H. Patterson, R. Slowinski, F. B. Talbot, and J. Weglarz. An algorithm for a general class of precedence and resource constrained scheduling problems. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*, pages 3–28. Elsevier, Amsterdam, the Netherlands, 1989.

[12] F. J. Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4:227–252, 1985.

[13] R. Slowinski. Two approaches to problems of resource allocation among project activities: A comparative study. *Journal of the Operational Research Society*, 31:711–723, 1980.

[14] M. G. Speranza and C. Vercellis. Hierarchical models for multi-project planning and scheduling. *European Journal of Operational Research*, 64:312–325, 1993.

[15] A. Sprecher. *Resource-constrained project scheduling: Exact methods for the multi-mode case*. Number 409 in Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, Germany, 1994.

[16] A. Sprecher. Solving the RCPSP efficiently at modest memory requirements. Manuskripte aus den Instituten für Betriebswirtschaftslehre 425, Universität Kiel, Germany, 1996.

[17] A. Sprecher and A. Drexl. 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 385, Universität Kiel, Germany, 1996.

[18] A. Sprecher and A. Drexl. 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 386, Universität Kiel, Germany, 1996.

[19] A. Sprecher, S. Hartmann, and A. Drexl. An exact algorithm for project scheduling with multiple modes. *OR Spektrum*, 19:195–203, 1997.

[20] A. Sprecher, R. Kolisch, and A. Drexl. Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80:94–102, 1995.

[21] J. P. Stinson, E. W. Davis, and B. M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10:252–259, 1978.

[22] F. B. Talbot. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28:1197–1210, 1982.

[23] F. B. Talbot and J. H. Patterson. An efficient integer programming algorithm with network cuts for solving resource-constrained project scheduling problems. *Management Science*, 24:1163–1174, 1978.