Sprecher, Arno; Hartmann, Sönke; Drexl, Andreas

Project scheduling with discrete time-resource and resource-resource tradeoffs

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 357

No. 357

Project Scheduling with Discrete Time-Resource and Resource-Resource Tradeoffs

Arno Sprecher / Sönke Hartmann / Andreas Drexl

December 1994

Arno Sprecher, Sönke Hartmann, Andreas Drexl, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstraße 40, 24118 Kiel, Germany.

Abstract

We consider an extension of the classical resource-constrained project scheduling problem (RCPSP), which covers discrete resource-resource and time-resource tradeoffs. As a result a project scheduler is permitted to identify several alternatives or modes of accomplishment for each activity of the project.

The solution procedure we present is a considerable generalization of the branch-and-bound algorithm proposed by Demeulemeester and Herroelen, which is currently the most powerful method for optimally solving the RCPSP. More precisely, we extend their concept of delay alternatives by introducing mode alternatives. The basic enumeration scheme is then enhanced by dominance rules which highly increase the performance of the algorithm. The computational results obtained by solving the standard ProGen instances indicate that the new method outperforms the most rapid procedure reported in the literature by a factor of four. Additionally and more important than the average reduction of the solution time is the substantial decrease of the variance of the solution times.

Keywords: Project Management / Scheduling, Discrete Resource-Resource / Time-Resource Tradeoffs, Delay and Mode Alternatives, Branch-and-Bound, Dominance Rules, Computational Results.

1 Introduction

The scheduling of projects has its beginnings in the early fifties when the Metra Potential Method (MPM) and the Critical Path Method (CPM) were developed. By the use of technological precedence constraints and deterministic activity durations both methods mainly determine the minimal project length, time windows, i.e. intervals for the latest start and finish times of the activities, and the set of critical activities. Resource constraints are not taken into account explicitly.

By the introduction of resources we obtain as a generalization the resource-constrained project scheduling problem (RCPSP). The resources involved are available in a discrete and constant amount each period. Each activity has a unique prespecified duration and uses a constant amount of the resources involved each period it is in progress. Once started the activity may not be interrupted (no preemption allowed). The commonly considered objective is the minimization of the makespan.

Since the problem subsumes the wellknown job-shop and flow-shop problem it is an NP-hard problem as well (cf. [9]). Consequently, the RCPSP has caused numerous publications dealing with the developement of suboptimal (cf. e.g. [2], [11]) and optimal (cf. [1], [3], [13], [15], [16], [21]) solution procedures, where Demeulemeester and Herroelen's exact solution procedure (cf. [4], [5]) outperforms all the other approaches.

By considering the activity durations as a discrete function of the resources and/or amounts of the resources allocated, we obtain a more realistic model. The time-resource and resource-resource tradeoff can be implemented and the multi-mode resource-constrained project scheduling problem (MRCPSP) can be stated: Each activity can be performed in one out of a set of prescribed ways, called modes, having a mode specific duration and mode specific resource requirements.

Three different categories of resources are distinguished, that is, renewable, nonrenewable and doubly constrained ones (cf. [17]). Renewable resources are limited on a per-period-basis. Machines and manpower are examples of this resource category. For nonrenewable resources, the availability for the entire project is

limited. An example of this resource category is money if the budget of the project is limited. Doubly constrained resources are limited both on a per-period-basis and on a total-project-basis. Money is an example of this resource category if not only the budget of the project but also the per-period-cashflow is limited.
The purpose of the paper is to present a new efficient procedure for solving the MRCPSP. It considerably extends the currently most powerful exact solution procedure of Demeulemeester and Herroelen from the RCPSP to the MRCPSP. The outline of the remaining is as follows: Section 2 provides the precise description and mathematical programming formulation of the problem. Section 3 discusses sets of schedules the enumeration can be reduced to without loosing optimality. Section 4 outlines the basic enumeration scheme and Section 5 bears bounding rules that increase the performance of the enumeration procedure. Section 6 reports our computational experience. Finally, the conclusions are drawn in Section 7.

2 Problem Description

The MRCPSP can be stated as follows: We consider a project which consists of J activities (jobs). Due to technological requirements the activities are partially ordered, that is, there are precedence relations between some of the jobs. These precedence relations are imposed by sets of immediate predecessors $\mathcal{P}_j$, $1 \leq j \leq J$, indicating that an activity j may not be started before all of its predecessors are completed. With analogous interpretation for each activity j, $1 \leq j \leq J$, the set of the immediate successors $\mathcal{S}_j$ is defined. The precedence relations can be represented by an acyclic activity-on-node network. Job 1 is the only source activity while job J is the unique sink activity of the network. Furthermore, we assume that the activities are numerically labeled, that is, a predecessor of job j has a smaller number than j.
Each activity requires certain amounts of resources to be performed. We distinguish three different categories of resources, that is, renewable, nonrenewable and doubly constrained ones. The set of renewable resources is referred to as R. For each renewable resource r, $r \in R$, the per-period-availability is constant and given by K_r^ρ. For nonrenewable resources, the availability for the entire project is limited. The set of nonrenewable resources is denoted as N. For each nonrenewable resource r, $r \in N$, the overall availability for the entire project is given by K_r^ν. Since doubly constrained resources can simply be incorporated by the enlargement of the sets of the renewable and nonrenewable resources, we do not consider them explicitly.
Each activity can be performed in one of several modes of accomplishment. A mode represents a way of combining different resources and/or levels of resource requests. M_j denotes the number of modes of activity j, $1 \leq j \leq J$. The duration of job j being performed in mode m, $1 \leq m \leq M_j$, is given by d_{jm}. The modes of each activity are labeled with respect to non-decreasing duration. Once an activity is started in one of its modes, it is not allowed to be interrupted. Furthermore, job j, $1 \leq j \leq J$, being performed in mode m, $1 \leq m \leq M_j$, uses k_{jmr}^ρ units of renewable resource r, $r \in R$, each period it is in process, where w. l. o. g. we assume $k_{jmr}^\rho \leq K_r^\rho$ for each renewable resource r, $r \in R$. Note, otherwise activity j could not be performed in mode m. Moreover, it consumes k_{jmr}^ν units of nonrenewable resource r, $r \in N$. 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 under consideration is the minimization of the project's makespan. A summary of the notation introduced can be found in Table 1. We assume the parameters to be nonnegative and integer valued.

Given the precedence relations and an upper bound $\overline{T}$ on the project's makespan, which is e.g. given by the sum of the maximal durations of the activities, we use the modes of shortest duration and derive time

J	:	number of activities
$j = 1\,(J)$	:	unique source (sink) activity
$\mathcal{P}_j\ (\mathcal{S}_j)$	:	set of immediate predecessors (successors) of activity j
M_j	:	number of modes of activity j
d_{jm}	:	(non preemptable) duration of activity j being performed in mode m
$R\,(N)$	:	set of renewable (nonrenewable) resources
k^{ρ}_{jmr}	:	per-period usage of renewable resource r required to perform activity j in mode m
k^{ν}_{jmr}	:	total consumption of nonrenewable resource r required to perform activity j in mode m
K^{ρ}_{r}	:	per-period availability of renewable resource r
K^{ν}_{r}	:	total availability of nonrenewable resource r

Table 1: Symbols and Definitions

windows by traditional forward and backward recursion as performed in MPM. These time windows are represented by intervals $[EF_j, LF_j]$, where EF_j and LF_j denote the earliest and the latest finish time of activity j, $1 \leq j \leq J$, respectively. Analogously, time windows $[ES_j, LS_j]$ can be determined, where ES_j and LS_j denote the earliest and the latest start time of activity j, $1 \leq j \leq J$, respectively.
The benefit of the time-windows is twofold: First, they can be used in the mathematical programming formulation to reduce the number of variables substantially. Second, they can be utilized in several enumeration procedures to speed up the convergence (cf. e.g. [5], [19], and Section 5).
For the formal description of the MRCPSP we define a binary variable for each combination of an activity j, $1 \leq j \leq J$, a mode m, $1 \leq m \leq M_j$, and a period t, $t = 0, \ldots, \overline{T}$ (cf. [22]):

$$x_{jmt} = \begin{cases} 1, & \text{if job } j \text{ is performed in mode } m \text{ and completed in period } t \\ 0, & \text{otherwise.} \end{cases}$$

The objective function and the constraints of the model are shown in Table 2. Since activity J is the only finish activity, the objective function (1) reflects the project's makespan which has to be minimized. Constraints (2) indicate that each activity is assigned exactly one mode and exactly one finish time. (3) ensures that no activity is started until all its predecessors are finished. Furthermore, (4) secures that the per-period-levels of the renewable resources are met. The consumption of the nonrenewable resources is limited to their availabilities by (5).
By simple adaptations the constraints of the model can be extended to time varying supply and usage of the renewable resources (cf. [7]). Moreover, minimal and maximal time-lags can be included (cf. [1]).

3 Dominating Sets of Schedules

The classification of schedules offers a basic framework of the project scheduling theory and the improvement of algorithmic tractability. Clearly, the computational performance of an enumeration procedure can be

$$\text{Minimize} \quad \sum_{t=EF_J}^{LF_J} t \cdot x_{J1t} \tag{1}$$

s. t.

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

$$\sum_{m=1}^{M_h} \sum_{t=EF_h}^{LF_h} t \cdot x_{hmt} \leq \sum_{m=1}^{M_j} \sum_{t=EF_j}^{LF_j} (t - d_{jm}) \cdot x_{jmt} \qquad j = 2, \ldots, J,\ h \in \mathcal{P}_j \tag{3}$$

$$\sum_{j=2}^{J-1} \sum_{m=1}^{M_j} k_{jmr}^{\rho} \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_{jm}-1, LF_j\}} x_{jmq} \leq K_r^{\rho} \qquad r \in R,\ t = 1, \ldots, \overline{T} \tag{4}$$

$$\sum_{j=2}^{J-1} \sum_{m=1}^{M_j} k_{jmr}^{\nu} \sum_{t=EF_j}^{LF_j} x_{jmt} \leq K_r^{\nu} \qquad r \in N \tag{5}$$

$$x_{jmt} \in \{0, 1\} \qquad j = 1, \ldots, J,\ m = 1, \ldots, M_j,\ t = 0, \ldots, \overline{T} \tag{6}$$

Table 2: The Model of the MRCPSP

substantially enhanced if the enumeration is reduced from the entire set of schedules $\mathcal{S}$ to a proper subset $\mathcal{DS}$ without loosing optimality, that is, the subset $\mathcal{DS}$ dominates $\mathcal{S}$.
For the MRCPSP a schedule is properly defined when each activity is assigned a mode and a start or finish time. More precisely, we define:

Definition 3.1 *Let $\mathbf{Z}_+$ denote the set of the nonnegative integers.*

(a) A schedule is a set of triplets $S = \{(1, m_1, f_1), \ldots, (J, m_J, f_J)\}$, where each activity j, $1 \leq j \leq J$, is assigned a mode m_j, $1 \leq m_j \leq M_j$, and a finish time f_j, $f_j \in \mathbf{Z}_+$.

(b) A schedule S is called feasible if the precedence constraints are maintained and if the resource contraints are met.

(c) A partial schedule PS is a subset of a schedule S.

Given a schedule S for the MRCPSP, we can prove whether a local or global left-shift (cf. [20]) can be performed on S. Consequently, the sets of semi-active and active schedules can be defined for the multi-mode

problem. That is, a feasible schedule of the MRCPSP is semi-active (active) if the schedule is semi-active (active) w.r.t. to the (single-mode) RCPSP obtained by fixing the modes to the ones selected in the schedule. Analogously to the RCPSP we can capture the following (cf. [8], p. 27): If there is an optimal schedule for a given instance, then there is an optimal semi-active (active) schedule.
Note, within a local or global left shift a mode change is not allowed to reduce the finish time of the corresponding activity. If we additionally permit a mode change we obtain the notion of tight schedules introduced by Speranza and Vercellis (cf. [18]):

Definition 3.2 *Let* $S = \{(j, m_j, f_j) \mid j = 1, \ldots, J\}$ *be a feasible schedule.*

(a) A multi-mode left shift of an activity j, $1 \leq j \leq J$, *is an operation on* S *the result of which is a feasible schedule* S' *with* $S' = S \backslash \{(j, m_j, f_j)\} \cup \{(j, m'_j, f'_j)\}$ *such that* $1 \leq m'_j \leq M_j$ *and* $f'_j < f_j$.

(b) A schedule S *is called tight if no multi-mode left shift of any of the activities can be performed on* S.

That is, a schedule is tight if there does not exist an activity the finish time of which can be reduced without violating the constraints or changing the finish time or mode of any of the remaining activities. The definition is illustrated in the following example. Consider the project instance given in Table 3. The corresponding network is shown in Figure 1. Two solutions for this instance can be found in Figure 2, where $j(m)$ denotes that job j is processed in mode m. Note, both schedules are active. Schedule (a) is not tight because a multi-mode left-shift can be performed: Activity 3 which is accomplished in mode 1 and finishes at time 4 could be executed in mode 2 and finished at time 3. The result of this multi-mode left shift is displayed in Figure 2 (b). Note, schedule (b) is tight because no multi-mode left shift can be performed.

j	$\mathcal{S}_j$	m	d_{jm}	k^{ρ}_{jm1}	R	K^{ρ}_1	N
1	{2,3}	1	0	0	{1}	3	∅
2	{4}	1	2	2			
3	{5}	1	2	2			
		2	3	1			
4	{6}	1	1	3			
		2	2	1			
5	{6}	1	2	3			
6	∅	1	0	0			

Table 3: Project Instance

Note, whereas for the RCPSP the set of tight schedules coincides with the set of active schedules, the set of tight schedules of a multi-mode problem is only a subset of the active ones. Nevertheless, since we can obtain an optimal tight schedule by iteratively applying multi-mode left shifts to an optimal schedule, optimality is not lost if only tight schedules are enumerated (cf. [18], Proposition 4.1).
If instead of the finish time reduction associated with a multi-mode left-shift a mode reduction (without changing the finish time) is considered we obtain the set of mode-minimal schedules. More precisely, we define:

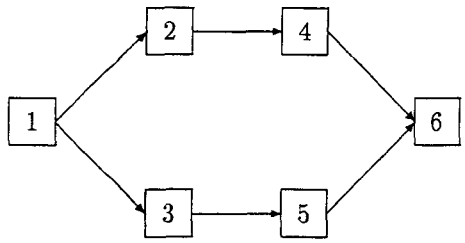

Figure 1: Project Network

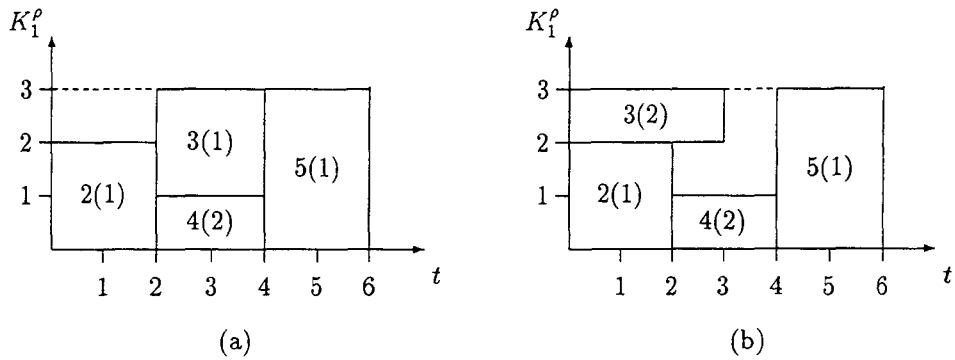

Figure 2: Non-Tight and Tight Schedule of the Project Instance

Definition 3.3 *Let* $S = \{(j, m_j, f_j) \mid j = 1, \ldots, J\}$ *be a feasible schedule.*

(a) A mode reduction of an activity j, $1 \leq j \leq J$, *is an operation on* S *the result of which is a feasible schedule* S' *with* $S' = S \backslash \{(j, m_j, f_j)\} \cup \{(j, m'_j, f_j)\}$ *such that* $1 \leq m'_j < m_j$.

(b) The schedule S *is called mode-minimal if no mode reduction of any activity can be performed on* S.

Again, we consider the project instance given in Table 3 and the two solutions shown in Figure 2. It can be easily verified that the schedule illustrated in Figure 2 (a) is mode-minimal. In contrast, schedule (b) is not mode-minimal because a mode reduction can be performed on activity 4 which is accomplished in mode 2 and ends at time 4: Being executed in mode 1 and started at time 3 instead of 2, job 4 could be finished at time 4 as well.

This example shows that there are tight schedules which are not mode-minimal and, moreover, there are mode-minimal schedules which are not tight. Nevertheless, it can easily be verified that the search space can be reduced to the set of mode-minimal schedules without loosing optimality.

Note, in accordance with the remarks stated above the search space of an enumeration scheme can only be restricted to schedules which are either tight or mode-minimal. The following theorem, however, states that it is sufficient to examine schedules which are both tight and mode-minimal.

Theorem 3.1

If there is an optimal schedule for a given instance, then there is an optimal schedule which is both tight and mode-minimal.

Proof: Consider an optimal tight schedule. This schedule can be transformed into a mode-minimal schedule by applying mode reductions. If the resulting schedule is not tight, it can be turned into a tight schedule by performing multi-mode left shifts. Note, since a mode reduction does not affect the finish times of the activities and since a multi-mode left shift can only reduce a finish time, only a finite number of mode reductions and multi-mode left shifts can be performed. That is, iteratively applying the substitutions as described above results in an optimal schedule which is both tight and mode-minimal. □

During the enumeration process the objective is to exclude dominated schedules from evaluation as early as possible. That is, one has to decide whether a partial schedule can be completed to a semi-active, active, tight and mode-minimal schedule or not. The decision rules can be used in an enumeration scheme to enhance the performance (cf. Section 5).
Most of the solution procedures presented in the literature (cf. e.g. [5], [19], [21]) employ bounding rules which reduce the search space to the sets of semi-active and active schedules. The notion of tight schedules is only used by Speranza and Vercellis (cf. [18], [10]), the notion of mode-minimal schedules is entirely new.

4 Branching on Mode- and Delay-Alternatives

In this section we present a branch-and-bound algorithm for solving the MRCPSP to optimality. The fundamental approach is a considerable extension of the concept of delay alternatives used by Demeulemeester and Herroelen (cf. [4], [5]) as well as Christofides et al. (cf. [3]) for the (single-mode) RCPSP. To describe the algorithm accurately we need the following definitions:

Definition 4.1 *Let t_g be a time instant.*

(a) An activity j, $1 \leq j \leq J$, which is scheduled in mode m_j with finish time f_j is in process at time t_g, if we have $f_j - d_{jm} < t_g \leq f_j$. The related set is the set of activities that are in process. It is denoted as JIP_g.

(b) A currently unscheduled activity j, $1 \leq j \leq J$, is called eligible at time t_g if all of its predecessors i, $i \in \mathcal{P}_j$, are scheduled with finish time $f_i \leq t_g$. The related set is the set of eligible activities which is denoted as EJ_g.

Recall, in Demeulemeester's and Herroelen's algorithm (cf. [4], [5]), all eligible activities are started at the current time instant (decision point) in their only possible mode. If a resource conflict occurs, one has to decide which of the activities in process to delay in order to resolve the conflict. In the multi-mode case, the eligible activities have to be assigned a mode before they can be put into process. Subsequently, the concept of delay alternatives can be used. In order to reflect the fixing of the modes, we introduce the notion of a mode alternative.

Definition 4.2
Let $\mathbf{Z}_+$ denote the set of the nonnegative integers. Furthermore, let t_g be a time instant at some level g of the branch-and-bound tree, and let EJ_g denote the set of the activities that are eligible at time t_g. If $EJ_g \neq \emptyset$, then a mode alternative is a mapping $\mathcal{MA}_g : EJ_g \rightarrow \mathbf{Z}_+$ which assigns each eligible activity j, $j \in EJ_g$, a mode $\mathcal{MA}_g(j) \in \{1, \ldots, M_j\}$.

Definition 4.3

(a) Let t_g be a time instant at level g of the branch-and-bound tree, and let JIP_g be the set of activities in process at time t_g. Let m_j denote the mode of activity j, $j \in JIP_g$. A delay alternative $\mathcal{DA}_g$ is a subset of JIP_g such that for each resource r, $r \in R$, it is

$$\sum_{j \in JIP_g \setminus \mathcal{DA}_g} k^{\rho}_{jm_jr} \leq K^{\rho}_r .$$

(b) A delay alternative $\mathcal{DA}_g$ is called minimal if no proper subset of $\mathcal{DA}_g$ is a delay alternative.

Using the definitions given above the proceeding at a current time instant (decision point) is as follows: If the current eligible set is empty, we proceed as outlined by Deemulemeester and Herroelen for the single-mode case (cf. [4], [5]), that is, we test a delay alternative. Otherwise, we select a mode alternative in order to fix the modes of the eligible activities. Subsequently, if necessary, we test the delay alternatives. If we return, via backtracking, to a level of the branch-and-bound tree at which all delay alternatives have been examined, we select the next mode alternative; if there is no more we track another stage back. The variables and symbols used in the algorithm are summarized in Table 4 while a formal description of the algorithm is provided in Table 5.

g	:	level of the branch-and-bound tree
m_j	:	mode of activity j
f_j	:	finish time of activity j
$f^{old}_{g,j}$	:	stored finish time of activity j before delay at level g
t_g	:	decision point at level g
JIP_g	:	set of activities that are in process at time t_g
FJ_g	:	set of activities that finish at time t_g
$\overline{FJ}_g$	:	set of activities that finish at or before time t_g
SJ_g	:	set of activities that are scheduled up to level g
EJ_g	:	set of activities that are eligible at time t_g
$\mathcal{SOMA}_g$	:	set of mode alternatives remaining at level g
$\mathcal{MA}_g$	:	selected mode alternative at level g
$\mathcal{SODA}_g$	:	set of delay alternatives remaining at level g
$\mathcal{DA}_g$	:	selected delay alternative at level g

Table 4: Notation used in the Algorithm for the MRCPSP

In Step 1 of the algorithm level $g = 0$ of the branch-and-bound tree is initialized. That is, the dummy source activity 1 is started at the time instant at level 0, $t_0 = 0$. Therefore, the set of the activities in process at level 0, JIP_0, and the set of the scheduled jobs at level 0, SJ_0, is initialized with $\{1\}$. The set of the activities that are finished at or before time t_0, $\overline{FJ}_0$, is initialized with $\emptyset$. Moreover, the finish time of the

Step 1: **(Initialization)**

$g := 0$; $t_0 := 0$; $JIP_0 := \{1\}$; $SJ_0 := \{1\}$; $\overline{FJ}_0 := \emptyset$; $f_1 := 0$; $m_1 := 1$; $\mathcal{DA}_0 := \emptyset$;

Step 2: **(Reschedule Delayed Activities and Compute Eligible Activities)**

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

$FJ_g := \{j \in JIP_{g-1} \mid f_j = t_g\}$; $\overline{FJ}_g := \overline{FJ}_{g-1} \cup FJ_g$;

$f_j := t_g + d_{jm_j}$, $j \in \mathcal{DA}_{g-1}$; $JIP_g := (JIP_{g-1} \backslash FJ_g) \cup \mathcal{DA}_{g-1}$; $SJ_g := SJ_{g-1} \cup \mathcal{DA}_{g-1}$;

$EJ_g := \{j \in \{1, \ldots, J\} \backslash SJ_g \mid \mathcal{P}_j \subseteq \overline{FJ}_g\}$; $JIP_g := JIP_g \cup EJ_g$; $SJ_g := SJ_g \cup EJ_g$;

if $J \in EJ_g$ then store current solution and go to Step 8;

Step 3: **(Compute Mode Alternatives)**

if $EJ_g = \emptyset$ then $\mathcal{SOMA}_g := \emptyset$ and go to Step 6,

else $\mathcal{SOMA}_g := SetOfModeAlternatives(EJ_g)$;

Step 4: **(Select Mode Alternative)**

if $\mathcal{SOMA}_g = \emptyset$ then go to Step 8,

else select $\mathcal{MA}_g \in \mathcal{SOMA}_g$; $\mathcal{SOMA}_g := \mathcal{SOMA}_g \backslash \{\mathcal{MA}_g\}$;

for each $j \in EJ_g$ update $m_j := \mathcal{MA}_g(j)$ and $f_j := t_g + d_{jm_j}$;

Step 5: **(Check for Nonrenewable Resource Conflict)**

if any nonrenewable resource produces a resource conflict then goto Step 4;

Step 6: **(Compute Delay Alternatives)**

$\mathcal{SODA}_g := SetOfMinimalDelayAlternatives(JIP_g)$;

Step 7: **(Select Delay Alternative)**

If $\mathcal{SODA}_g = \emptyset$ then go to Step 4,

else select $\mathcal{DA}_g \in \mathcal{SODA}_g$; $\mathcal{SODA}_g := \mathcal{SODA}_g \backslash \{\mathcal{DA}_g\}$;

$f_{g,j}^{old} := f_j$, $j \in \mathcal{DA}_g$; $JIP_g := JIP_g \backslash \mathcal{DA}_g$; $SJ_g := SJ_g \backslash \mathcal{DA}_g$; go to Step 2;

Step 8: **(Backtracking)**

$g := g - 1$; if $g = 0$ then STOP,

else $f_j := f_{g,j}^{old}$, $j \in \mathcal{DA}_g$; $JIP_g := JIP_g \cup \mathcal{DA}_g$; $SJ_g := SJ_g \cup \mathcal{DA}_g$; go to Step 7.

Table 5: A Branch-and-Bound Algorithm for the MRCPSP

source activity is defined as $f_1 := 0$, and its mode is defined as $m_1 := 1$. Finally, since no activity is delayed at level 0, the only minimal delay alternative at level 0, $\mathcal{DA}_0$, is the empty set.

In Step 2 we first branch to the next level of the branch-and-bound tree by incrementing g. Then the next time instant t_g (decision point) is computed. Since the per-period availability of the renewable resources is constant it is defined by the earliest finish time of the activities which are in process at the previous level. We determine FJ_g, that is, the set of the activities that finish at time t_g. The set of the activities that finish at or before time t_g, $\overline{FJ}_g$, is computed as the union of $\overline{FJ}_{g-1}$ and FJ_g. The jobs that have been delayed at the previous level are rescheduled to start at time t_g by updating their finish times w. r. t. their fixed modes. Then we obtain the set of the activities that are in process at level g, JIP_g, by eliminating the jobs that finish at time t_g from JIP_{g-1} and, moreover, adding the previously delayed activities. The set of the scheduled activities at level g, SJ_g, is computed by adding the previously delayed activities to SJ_{g-1}. Then the eligible set EJ_g is given by the set of those unscheduled jobs the predecessors of which are finished at or before time t_g. The eligible jobs are put in process, that is, they are added to JIP_g and SJ_g. If the dummy sink activity J is eligible, we have obtained a new complete schedule which is stored, and backtracking occurs. Otherwise, Step 3 is performed.

In Step 3, we determine the set of the mode alternatives $\mathcal{SOMA}_g$. If no job is eligible at the current level, there are no mode alternatives to be considered. Therefore, the set $\mathcal{SOMA}_g$ is empty. Furthermore, since there is no activity that has not been (temporarily) scheduled at the previous level, a resource conflict concerning a nonrenewable resource cannot occur. Thus, we skip to Step 6 in order to compute the set of delay alternatives. Otherwise, if there are eligible activities, we compute the set of the mode alternatives $\mathcal{SOMA}_g$ and proceed with Step 4.

In Step 4, a mode alternative $\mathcal{MA}_g$ is selected and removed from $\mathcal{SOMA}_g$. The modes m_j of the eligible activities j, $j \in EJ_g$, are fixed w. r. t. the selected mode alternative. Moreover, they are started at time t_g by the definition of their finish times f_j w. r. t. $\mathcal{MA}_g$. If no mode alternative is available, backtracking in Step 8 is performed.

Step 5 controls the consumption of the nonrenewable resources. If the availability of any nonrenewable resource is exceeded, the next mode alternative is selected in Step 4. Otherwise, in Step 6, the set of the minimal delay alternatives, $\mathcal{SODA}_g$, is computed.

In Step 7, a delay alternative $\mathcal{DA}_g$ is selected and then removed from the set of delay alternatives $\mathcal{SODA}_g$. Each activity j selected to be delayed is removed from JIP_g and SJ_g and, moreover, its finish time f_j is stored in $f_{g,j}^{old}$. The delay alternative has resolved the current resource conflict and Step 2 is performed, that is, the delayed activities are rescheduled to start at the next decision point. However, if no delay alternative is available, Step 4 is executed for selecting the next mode alternative.

In Step 8, backtracking is performed. If, after decrementing g, the level of the branch-and-bound tree is zero, the algorithm stops. Otherwise, the activities that have previously been delayed at this level are put in process by restoring their finish times. After the adjustment of JIP_g and SJ_g, Step 7 is executed for selecting the next delay alternative.

Using the correctness of the algorithm proposed by Demeulemeester we can state the following theorem:

Theorem 4.1

The algorithm of Table 5 is correct, that is, it finds an (existing) optimal (i. e. makespan-minimal) solution for a given MRCPSP.

Proof: Obviously, the algorithm of Table 5 terminates if the given problem is feasible w.r.t. the renewable resources. Since a feasible schedule is especially feasible w. r. t. the nonrenewable resources and, moreover, it is $k^{\nu}_{jmr} \geq 0$, $j = 1, \ldots, J$, $m = 1, \ldots, M_j$, $r \in N$, there is no feasible completion of a partial schedule which is infeasible w. r. t. a nonrenewable resource. Therefore, Step 5 does not exclude feasible schedules from evaluation, and we can assume w. l. o. g. $|N| = 0$.
Let $\{(1, m_1), \ldots, (J, m_J)\}$, $1 \leq m_j \leq M_j$, $1 \leq j \leq J$, be a set of job/mode combinations from which we can obtain an optimal solution by the use of Demeulemeester's algorithm. Since for each activity j, $1 \leq j \leq J$, the related mode m_j can be selected by a mode alternative when the predecessors of activity j are finished, the algorithm reduces to the branching scheme proposed by Demeulemeester. □

5 Bounding Rules

This section provides static and dynamic search tree reduction techniques. Whereas the static bounding rules can be realized by an adaptation of the input data, the assumptions of the dynamic bounding rules have to be checked during the enumeration process. The rules are stated as theorems and, if necessary, additional information about their algorithmic realization is given.
The first bounding rule bears both static and dynamic features. The pruning rule and variants of it have been successfully employed for solving the RCPSP (cf. e.g. [5], [21]) as well as the MRCPSP (cf. [14], [19]). Given an upper bound on the project's makespan, the latest finish times LF_j of the activities j, $j = 1, \ldots, J$, as described in Section 2 can be used to reduce the search space. They are updated after finding an improved feasible solution.

Theorem 5.1 *(Bounding Rule 1)*
Let T denote an upper bound on the project's makespan. If there is an activity j, $1 \leq j \leq J$, the assigned finish time of which exceeds the latest finish time LF_j, then the current partial schedule cannot be completed with a makespan less than or equal to T $(= LF_J)$.

The enumeration procedure starts with a predefined upper bound and successively adapts it, when the first or an improved solution is found. That is, if T denotes a newly obtained makespan related to a complete schedule, then we can recalculate the latest finish times by

$$LF_j := LF_j - (LF_J - T + 1), \qquad j = 1, \ldots, J.$$

The underlying idea of the first static acceleration scheme is to exclude modes and/or nonrenewable resources from the input data. The effect is a reduction of the number of modes or constraints to be considered. For notational convenience, we need the following definitions:

Definition 5.1

(a) The minimal request of activity j, $1 \leq j \leq J$, for nonrenewable resource r, $r \in N$, is given by

$$kmin^{\nu}_{jr} := \min\{k^{\nu}_{jmr} \mid m = 1, \ldots, M_j\}.$$

(b) The maximal request of activity j, $1 \leq j \leq J$, for nonrenewable resource r, $r \in N$, is given by

$$kmax^{\nu}_{jr} := \max\{k^{\nu}_{jmr} \mid m = 1, \ldots, M_j\}.$$

Definition 5.2 *Let j, $1 \leq j \leq J$, be an activity, and let m_j, $1 \leq m_j \leq M_j$, be a mode of activity j.*

(a) Mode m_j is called non-executable w. r. t. a renewable resource r, $r \in R$, if we have $k^{\rho}_{jm_jr} > K^{\rho}_r$.

(b) Mode m_j is called non-executable w. r. t. a nonrenewable resource r, $r \in N$, if we have

$$\sum_{\substack{i=1 \\ i \neq j}}^{J} kmin^{\nu}_{ir} + k^{\nu}_{jm_jr} > K^{\nu}_r.$$

(c) Mode m_j is called inefficient, if there exists another mode m'_j, $1 \leq m'_j \leq M_j$, of activity j with $d_{jm_j} \geq d_{jm'_j}$ and $k^{\rho}_{jm_jr} \geq k^{\rho}_{jm'_jr}$ for each renewable resource r, $r \in R$, and $k^{\nu}_{jm_jr} \geq k^{\nu}_{jm'_jr}$ for each nonrenewable resource r, $r \in N$ (cf. [12]).

(d) A nonrenewable resource r, $r \in N$, is called redundant, if we have

$$\sum_{j=1}^{J} kmax^{\nu}_{jr} \leq K^{\nu}_r.$$

Using the previous definitions, we can state the following theorem:

Theorem 5.2 *(Bounding Rule 2)*

(a) Within a feasible schedule for a given instance, no activity can be performed in a non-executable mode.

(b) If there is an optimal schedule for a given instance, then there is an optimal schedule in which no activity is accomplished in an inefficient mode.

(c) Excluding a redundant nonrenewable resource from a project instance does not affect the set of the feasible (optimal) schedules.

Note, if an activity has not a single executable mode then there is no feasible schedule for the given instance. Recall, we have assumed in Section 2 that no mode is non-executable w. r. t. a renewable resource. This is necessary in particular to secure that the enumeration algorithm terminates. However, since the project generator ProGen (cf. [12]) which has been used in the computational studies may generate instances including modes that are non-executable w. r. t. a renewable resource, they have to be eliminated from the project data before the algorithm is applied. Furthermore, according to Theorem 5.2, modes that are inefficient or non-executable w. r. t. a nonrenewable resource and, moreover, redundant nonrenewable resources may also be deleted.

In the following example, we discuss the process of erasing modes and/or nonrenewable resources in more detail. We consider the project instance given in Table 6. Mode 1 of activity 2 is non-executable w. r. t. renewable resource 1 and can therefore be deleted from the input data. This induces that mode 1 of activity 4 becomes non-executable w. r. t. nonrenewable resource 2. Removing this mode from the project data causes redundancy of nonrenewable resource 2 which, therefore, can be neglected. Now mode 2 of activity 5 becomes inefficient. Eliminating this mode turns nonrenewable resource 3 redundant. The reduced project instance with adjusted mode numbers is provided in Table 7.

This example shows that deleting a mode which is non-executable w. r. t. a renewable resource may force a mode of another activity to become non-executable w. r. t. a nonrenewable resource. Moreover, removing a non-executable mode from the project data may cause redundancy of a nonrenewable resource. Finally, erasing a redundant nonrenewable resource may lead to inefficiency of a mode while eliminating an inefficient

j	$\mathcal{S}_j$	m	d_{jm}	k^{ρ}_{jm1}	k^{ν}_{jm2}	k^{ν}_{jm3}	R	K^{ρ}_1	N	K^{ν}_2	K^{ν}_3
1	{2,3}	1	0	0	0	0	{1}	4	{2,3}	13	14
2	{4}	1	2	5	2	1					
		2	4	2	4	1					
3	{5}	1	3	3	3	3					
		2	5	1	2	4					
4	{6}	1	2	2	8	3					
		2	3	1	2	3					
5	{6}	1	3	2	3	2					
		2	4	2	1	7					
6	$\emptyset$	1	0	0	0	0					

Table 6: Project Instance

j	$\mathcal{S}_j$	m	d_{jm}	k^{ρ}_{jm1}	R	K^{ρ}_1	N
1	{2,3}	1	0	0	{1}	4	$\emptyset$
2	{4}	1	4	2			
3	{5}	1	3	3			
		2	5	1			
4	{6}	1	3	1			
5	{6}	1	3	2			
6	$\emptyset$	1	0	0			

Table 7: Reduced Project Instance

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.

Table 8: Implementation of Theorem 5.2

mode may cause redundancy of a nonrenewable resource. Therefore, the projects input data should be prepared as described in Table 8.

The next bounding rule to be presented is especially designed for instances with nonrenewable resources, that is, $|N| > 0$. Its dynamic variant, as given in Theorem 5.3, has been proposed by Drexl (cf. [6]) for a less general framework. Sprecher (cf. [19], p. 62) adapted the rule to the MRCPSP and substantially improved

the effect by reformulating it as a static rule.

Theorem 5.3 *(Bounding Rule 3)*
Let SJ_g be the set of the activities that are scheduled up to level g of the branch-and-bound tree, and let $\overline{SJ}_g := \{1, \ldots, J\} \setminus SJ_g$ denote the set of the currently unscheduled activities. If there is a resource r, $r \in N$, with

$$\sum_{j \in SJ_g} k^{\nu}_{jm_jr} + \sum_{j \in \overline{SJ}_g} kmin^{\nu}_{jr} > K^{\nu}_r,$$

then the current partial schedule cannot be completed.

By the following remark the time consuming calculations during enumeration phase can be reduced to simple adaption of the input data (cf. [19], Remark 5.7). Note, this adaption should be made after the input data has been prepared according to Bounding Rule 2.

Remark 5.1
The bounding rule of Theorem 5.3 can be implemented via preprocessing by adjusting the input data as follows:

$$\bar{k}^{\nu}_{jmr} := k^{\nu}_{jmr} - kmin^{\nu}_{jr}, \qquad j = 1, \ldots, J,\ m = 1, \ldots, M_j,\ r \in N$$

and

$$\bar{K}^{\nu}_r := K^{\nu}_r - \sum_{j=2}^{J-1} kmin^{\nu}_{jr}, \qquad r \in N.$$

The first dynamic bounding rule bases on the fact that, according to Theorem 3.1, optimality is preserved if the enumeration is reduced to schedules that are tight and mode-minimal. Therefore, whenever it is certain that no tight or mode minimal schedule can be obtained from the current partial schedule backtracking may be performed.

Theorem 5.4 *(Bounding Rule 4)*
Let t_g be the decision point at the current level of the branch-and-bound tree, and let FJ_g denote the set of those jobs that finish at time t_g. Furthermore, let PS_g denote the current partial schedule. If there exists an activity j, $j \in FJ_g$, such that a multi-mode left shift or a mode reduction of j with resulting mode m'_j, $1 \leq m'_j \leq M_j$, can be performed on PS_g and, moreover, if $k^{\nu}_{jm'_jr} \leq k^{\nu}_{jm_jr}$ holds for each nonrenewable resource r, $r \in N$, then PS_g needs not be completed.

Proof: Let PS' denote the result of the multi-mode left shift or mode reduction performed on PS_g. Let PS_g be completable w. r. t. the resources. Then PS' is completable, too. Furthermore, if a multi-mode left shift or mode reduction of an activity that finishes at time t_g can be performed at level g, then the same multi-mode left shift or mode reduction can be performed at any later stage $\bar{g}$, $\bar{g} \geq g$. That is, scheduling or delaying activities at any stage $\bar{g}$, $\bar{g} \geq g$, cannot prevent the possibility of performing a multi-mode left shift or mode reduction of any activity out of FJ_g. Consequently, any obtainable schedule cannot be tight and mode-minimal. However, in accordance with Theorem 3.1 there is an optimal schedule which is tight and mode-minimal. Therefore, optimality is preserved if PS_g is not completed. □

Obviously, this bounding rule can be applied in two steps of the algorithm, in Step 2 and Step 7. Whereas the interpretation of the former includes that all the delay alternatives are dominated, the latter one means that only the currently selected delay alternative can be excluded from further consideration. Although the delay of an activity j, $1 \leq j \leq J$, that has been started on a lower level, frees resources in periods t, $t = f_j - d_{jm_j} + 1, \ldots, t_g$, the effect of the application in Step 7 is consumed by the additional effort.
Moreover, clearly, one can extend the theorem to $\overline{FJ}_g$ instead of FJ_g. This would guarantee that only tight and mode-minimal schedules are generated which does not hold if only FJ_g is considered. But again, the additional effect is consumed by the additional effort for checking the assumptions.
Note, in general Theorem 5.4 cannot be applied to an activity which is not finished at the current decision point without losing optimality (cf. [10]).
Although the infeasibility of a multi-mode left shift, via Bounding Rule 4, includes that a local left shift is infeasible too (cf. Section 3), it is useful to check for the latter one seperately. This is reasoned by the fact that a feasible local left shift of an activity considered for starting on the current level g at time instant t_g cannot be averted by scheduling or delaying activities on levels $\bar{g}$, $\bar{g} > g$. That is, the exclusion of partial schedules due to a feasible local left shift of an activity can be detected on a lower stage than the same feasible (mode-preserving) multi-mode left-shift.
Consequently, the bounding rule presented next is the so-called left shift rule which seeks to exclude partial schedules from consideration if it is certain that no complete semi-active schedule can be obtained. The left shift rule has been successfully used in several algorithms for the single-mode case as well as for the multi-mode case (cf. e.g. [21], [19], p. 61). We employ a formulation which is similar to the one used by Demeulemeester for the single-mode case (cf. [4], p. 52).

Theorem 5.5 *(Bounding Rule 5, Left Shift Rule)*
Let $\mathcal{DA}_g$ and $\mathcal{DA}_{g-1}$ denote the delay alternatives selected at the current and at the previous level of the branch-and-bound tree, respectively. If there exists an activity j, $j \in \mathcal{DA}_{g-1} \backslash \mathcal{DA}_g$, which can be locally left shifted without changing its mode, then the current partial schedule needs not be completed.

Proof: Since fixing the modes of the scheduled activities reduces the algorithm of Table 5 to Demeulemeester's procedure for the RCPSP (cf. the proof of Theorem 4.1), the proof for the left shift rule given in [4], p. 52, holds. □

Nevertheless, delaying an activity at a stage $\bar{g}$ higher than the current stage g can make a local left-shift feasible which is not feasible at the current stage g. Thus, inspite of the search tree reduction in accordance with Bounding Rule 5 one can obtain schedules that are not semi-active.
The following two bounding rules are generalizations of rules used by Demeulemeester and Herroelen in their algorithm for the RCPSP (cf. [5]). Roughly speaking, they reduce the examination of the set of the delay alternatives to a single alternative. We adapt these ideas to the multi-mode case. The following definition will simplify the formal description of the bounding rules.

Definition 5.3
Two activities i and j, $i, j \in \{1, \ldots, J\}$, $i \neq j$, with related modes m_i, $1 \leq m_i \leq M_i$, and m_j, $1 \leq m_j \leq M_j$, are simultaneously performable if they are independent with respect to the precedence relations and the sum of their resource usages, $k^{\varrho}_{im_ir} + k^{\varrho}_{jm_jr}$, does not exceed the availability K^{ϱ}_r of any renewable resource r, $r \in R$.

We use a four-dimensional array $\pi[i, m_i, j, m_j]$ to reflect whether job i in mode m_i and job j in mode m_j are simultaneously performable or not. We obtain:

$$\pi[i, m_i, j, m_j] := \begin{cases} 1, & \text{if job } i \text{ in mode } m_i \text{ and job } j \text{ in mode } m_j \text{ are simultaneously performable,} \\ 0, & \text{otherwise.} \end{cases}$$

Consider the project instance shown in Table 9. The corresponding network is shown in Figure 3. Activity 3 can only be performed in mode 1. Since activity 3 is not simultaneously performable with the activities 1, 2, 6 and 8 due to the precedence constraints, we have $\pi[3,1,1,1] = \pi[3,1,2,1] = \pi[3,1,6,1] = \pi[3,1,6,2] = \pi[3,1,8,1] = 0$. Moreover, activity 3 in mode 1 cannot be performed at the same time as activities 4 in mode 1, 5 in mode 1, 7 in mode 1, and 7 in mode 2 without violating the resource constraints. Therefore, it is $\pi[3,1,4,1] = \pi[3,1,5,1] = \pi[3,1,7,1] = \pi[3,1,7,2] = 0$. However, performing activity 3 in mode 1 together with job 5 in mode 2 does not violate the constraints, that is, $\pi[3,1,5,2] = 1$. Finally, we have $\pi[3,1,3,1] = 0$ by definition.

j	$\mathcal{S}_j$	m	d_{jm}	k^{ρ}_{jm1}	R	K^{ρ}_1	N	$\pi[3,1,j,m]$
1	$\{2\}$	1	0	0	$\{1\}$	4	$\emptyset$	0
2	$\{3,4,5\}$	1	2	1				0
3	$\{6\}$	1	5	3				0
4	$\{6\}$	1	4	2				0
5	$\{7\}$	1	2	2				0
		2	3	1				1
6	$\{8\}$	1	3	2				0
		2	4	1				0
7	$\{8\}$	1	1	3				0
		2	2	2				0
8	$\emptyset$	1	0	0				0

Table 9: Project Instance

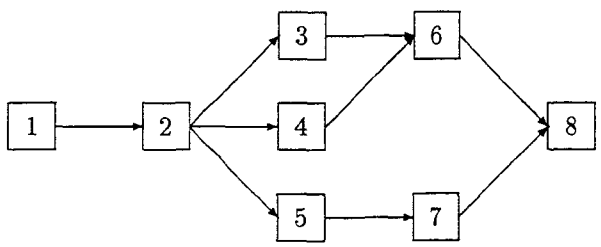

Figure 3: Network of the Project Instance

Now the basic idea of the rule can be described as follows (cf. [5], Theorem 1): We assume that all of the activities currently in process have either been previously delayed or have become eligible, that is, they start at the current decision point. Moreover, we can find an activity j which is in process and which cannot be simultaneously processed with any other activity temporarily started at t_g. Furthermore, job j cannot be

performed at the same time as any currently unscheduled activity in any mode. Then optimality is preserved if we start activity j at the current decision point, delay all the other activities which are currently in process, and do not examine any other delay alternative at the current decision point t_g. More precisely, we have:

Theorem 5.6 *(Bounding Rule 6)*
Let $\mathcal{SODA}_g$ be the newly computed set of minimal delay alternatives at level g of the branch-and-bound tree, that is, no delay alternative has been selected or examined yet. Let SJ_g denote the set of the jobs scheduled up to level g, and let $\overline{SJ}_g := \{1,\ldots,J\}\backslash SJ_g$ be the set of the unscheduled jobs. Furthermore, let JIP_g be the set of the activities in process at the current decision point t_g. If all the activities in process at time t_g start at time t_g and, moreover, there exists an activity h, $h \in JIP_g$, such that

(a) $\pi[h, m_h, j, m_j] = 0$ for all activities j, $j \in JIP_g\backslash\{h\}$,

(b) $\pi[h, m_h, j, m] = 0$ for all activities j, $j \in \overline{SJ}_g$, and all modes m, $1 \leq m \leq M_j$,

then $\mathcal{DA}_g = JIP_g\backslash\{h\}$ is the only minimal delay alternative that has to be considered when extending the current partial schedule.

Proof: We assume that activity h, $h \in JIP_g$, scheduled in mode m_h, fulfills the assumptions of Theorem 5.6. Let $\mathcal{DA}_g := JIP_g\backslash\{h\}$, and let $\bar{S} := \{(1, \bar{f}_1, \bar{m}_1), \ldots, (J, \bar{f}_J, \bar{m}_J)\}$ denote a schedule obtained by branching from the current node using a minimal delay alternative $\overline{\mathcal{DA}}_g$ other than $\mathcal{DA}_g$. Note, it is $h \in \overline{\mathcal{DA}}_g$ and, moreover, $m_j = \bar{m}_j$ for each scheduled job j, $j \in SJ_g$. Especially, we have $m_h = \bar{m}_h$. We show that there is a schedule obtainable by branching from the current node using the delay alternative $\mathcal{DA}_g$ which has a makespan less than or equal to the one of $\bar{S}$. Using Figure 4 (a) for illustration we define:

$$\bar{S}_1 := \{(j, \bar{m}_j, \bar{f}_j) \mid j = 1,\ldots,J,\ \bar{f}_j \leq t_g\},$$

$$\bar{S}_2 := \{(j, \bar{m}_j, \bar{f}_j) \mid j = 1,\ldots,J,\ j \neq h,\ t_g < \bar{f}_j \leq \bar{f}_h - d_{h\bar{m}_h}\},$$

$$\bar{S}_3 := \{(j, \bar{m}_j, \bar{f}_j) \mid j = 1,\ldots,J,\ j \neq h,\ \bar{f}_j - d_{j\bar{m}_j} \geq \bar{f}_h\}.$$

By the assumptions on π we have

$$\bar{S} = \bar{S}_1 \dot{\cup} \bar{S}_2 \dot{\cup} \{(h, \bar{m}_h, \bar{f}_h)\} \dot{\cup} \bar{S}_3.$$

Furthermore, we define:

$$S_1 := \bar{S}_1, \qquad S_2 := \{(j, \bar{m}_j, \bar{f}_j + d_{hm_h}) \mid (j, \bar{m}_j, \bar{f}_j) \in \bar{S}_2\}, \qquad S_3 := \bar{S}_3.$$

and obtain with

$$S := S_1 \dot{\cup} \{(h, m_h, t_g + d_{hm_h})\} \dot{\cup} S_2 \dot{\cup} S_3$$

a feasible schedule the makespan of which is equal to the one of $\bar{S}$. An illustration of schedule S is displayed in Figure 4 (b). Obviously, if we select delay alternative $\mathcal{DA}_g$ for branching from the current node, we can obtain a schedule with makespan less than or equal to the one of S. □

The following example illustrates the previously described bounding rule. We consider again the instance given in Table 9. Furthermore, we consider the partial schedule shown in Figure 5 (a). Obviously, activity 3

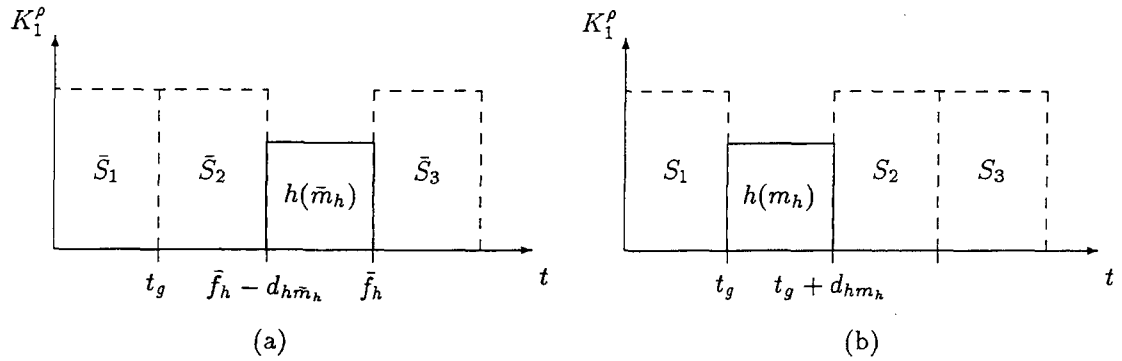

Figure 4: Illustration of Bounding Rule 4

cannot be scheduled together with any activity in process at time 2. Note, activity 3 could be in process at the same time as activity 5 being performed in mode 2. However, activity 5 has already been assigned mode 1 which cannot be changed at the current or a higher level. Moreover, job 3 cannot be in process together with any of the unscheduled jobs 6, 7, and 8. Consequently, activity 3 can only be in process on its own. Thus, according to Bounding Rule 4, the only minimal delay alternative to be examined is {4,5}.
In contrast, considering the partial schedule displayed in Figure 5 (b), activity 3 can be simultaneously processed with activity 5 which is scheduled in mode 2. In fact, none of the activities in process at time 2 can only be processed on its own. Thus, Bounding Rule 4 cannot be applied to this partial schedule.

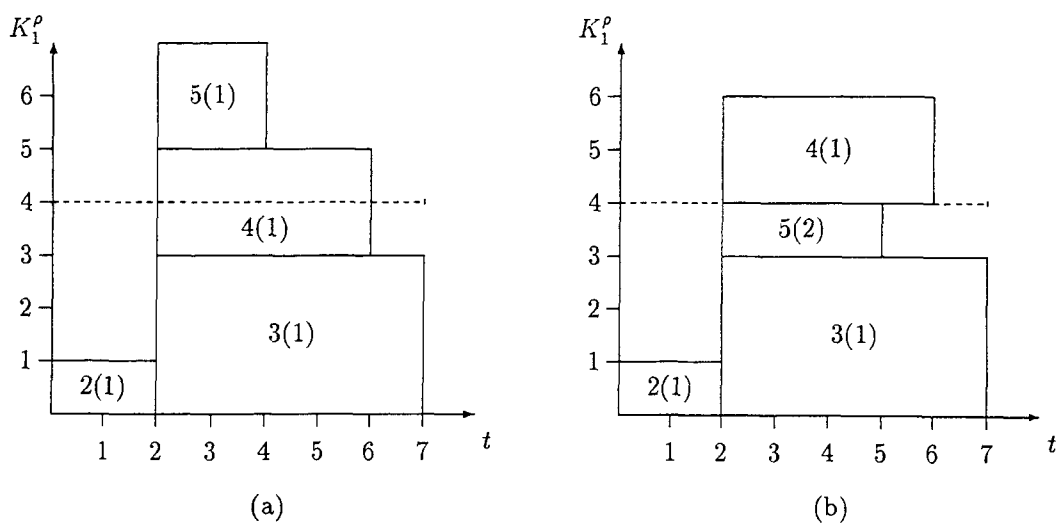

Figure 5: Partial Schedules for the Project Instance

The following bounding rule (cf. [5], Theorem 2) is closely related to the previous one. However, it seeks to start two activities at the current decision point which are simultaneously performable.

Theorem 5.7 *(Bounding Rule 7)*
Let $\mathcal{SODA}_g$ be the newly computed set of minimal delay alternatives at level g of the branch-and-bound tree, that is, no delay alternative has been selected or examined yet. Let SJ_g denote the set of the jobs scheduled

up to level g, and let $\overline{SJ}_g := \{1,\ldots,J\}\backslash SJ_g$ *be the set of the unscheduled jobs. Furthermore, let* JIP_g *be the set of the activities in process at the current decision point* t_g*. If all activities in process at time* t_g *start at time* t_g *and, moreover, there exists an activity* h, $h \in JIP_g$, *such that*

(a) $\pi[h, m_h, i, m_i] = 1$ *for exactly one activity* i, $i \in JIP_g\backslash\{h\}$,

(b) $\pi[h, m_h, j, m] = 0$ *for all activities* j, $j \in \overline{SJ}_g$ *and all modes* m, $1 \leq m \leq M_j$,

and if we additionally have $d_{im_i} \leq d_{hm_h}$, *then* $\mathcal{DA}_g = JIP_g\backslash\{h,i\}$ *is the only minimal delay alternative that has to be considered when extending the current partial schedule, and the next decision point to be considered is given by* f_h.

Proof: We assume that activities h and i, $h, i \in JIP_g$, with related modes m_h and m_i, respectively, fulfill the assumptions of Theorem 5.7. Let $\mathcal{DA}_g := JIP_g\backslash\{h,i\}$, and let $\bar{S} := \{(1,\bar{f}_1,\bar{m}_1),\ldots,(J,\bar{f}_J,\bar{m}_J)\}$ denote a schedule obtained by branching from the current node using a minimal delay alternative $\overline{\mathcal{DA}}_g$ other than $\mathcal{DA}_g$. Note, it is $h \in \overline{\mathcal{DA}}_g$ and, moreover, $m_j = \bar{m}_j$ for each scheduled job j, $j \in SJ_g$. Especially, we have $m_h = \bar{m}_h$ and $m_i = \bar{m}_i$. We show that there is a schedule obtainable by branching from the current node using the delay alternative $\mathcal{DA}_g$ which has a makespan less than or equal to the one of $\bar{S}$. We define:

$$\bar{S}_1 := \{(j,\bar{m}_j,\bar{f}_j) \mid j = 1,\ldots,J,\ \bar{f}_j \leq t_g\},$$

$$\bar{S}_2 := \{(j,\bar{m}_j,\bar{f}_j) \mid j = 1,\ldots,J,\ j \neq h,\ j \neq i,\ t_g < \bar{f}_j \leq \bar{f}_h - d_{h\bar{m}_h}\},$$

$$\bar{S}_3 := \{(j,\bar{m}_j,\bar{f}_j) \mid j = 1,\ldots,J,\ j \neq h,\ j \neq i,\ \bar{f}_j - d_{j\bar{m}_j} \geq \bar{f}_h\}.$$

We have

$$\bar{S} = \bar{S}_1 \dot{\cup} \bar{S}_2 \dot{\cup} \{(h,\bar{f}_h,\bar{m}_h)\} \dot{\cup} \bar{S}_3 \dot{\cup} \{(i,\bar{f}_i,\bar{m}_i)\}.$$

Furthermore, we define:

$$S_1 := \bar{S}_1, \qquad S_2 := \{(j,\bar{m}_j,\bar{f}_j + d_{hm_h}) \mid (j,\bar{m}_j,\bar{f}_j) \in \bar{S}_2\}, \qquad S_3 := \bar{S}_3.$$

and we obtain with

$$S := S_1 \dot{\cup} \{(h,m_h,t_g + d_{hm_h})\} \dot{\cup} \{(i,m_i,t_g + d_{im_i})\} \dot{\cup} S_2 \dot{\cup} S_3$$

a feasible schedule with a makespan equal to the one of $\bar{S}$. Obviously, if we select delay alternative $\mathcal{DA}_g$, we can consider f_h as the next decision point and obtain a schedule with makespan less than or equal to the one of S. □

To illustrate this bounding rule, we consider again the instance given in Table 9 and the partial schedule displayed in Figure 5 (b). Activity 3 in mode 1 may be processed simultaneously with activity 5 in mode 2, but not with job 4 in mode 1 (recall, we have $\pi[3,1,5,2] = 1$ and $\pi[3,1,4,1] = 0$). Note, it is $d_{5,2} = 3 \leq 5 = d_{3,1}$. Moreover, job 3 cannot be in performed at the same time as any of the unscheduled jobs 6, 7 and 8. Therefore, activity 3 can only be processed together with activity 5. Thus, according to Bounding Rule 5, the only minimal delay alternative to be examined is $\{4\}$. Note, activity 4 can be delayed up to time 7, that is, the finish time of activity 3.

6 Computational Results

In this section we present the results of the computational studies concerning the algorithm proposed in the previous sections. The experiments have been performed on an IBM-compatible 386-DX personal computer with 40 MHz clockpulse. The algorithm has been coded in Borland C.

We have made use of a set of test problems constructed by the project generator ProGen (cf. [12]). These instances have been generated for the validation of ProGen and, moreover, used by Sprecher to evaluate his algorithm (cf. [19], p. 91). Each of the project instances consists of 12 activities (including dummy source and sink activity). Each of the non-dummy activities may be performed in one out of three modes of accomplishment. The duration of a mode varies between 1 and 10 periods. We have two renewable and two nonrenewable resources. The detailed fixed parameter settings can be found in [12].

In order to design the set of instances four parameters, the resource factor and the resource strength of each resource category, have been varied. The resource factor RF is a measure of the average portion of resources requested per job. The resource strength RS reflects the scarceness of the resources. Table 10 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 combination of the parameters, ten instances have been generated. Following [19], we have considered only those 536 of the 640 resulting instances which have a feasible solution.

Parameter	Levels			
RF_R	0.5	1.0		
RS_R	0.2	0.5	0.7	1.0
RF_N	0.5	1.0		
RS_N	0.2	0.5	0.7	1.0

Table 10: Variable Parameter Levels under Full Factorial Design

The outline of this section is as follows: In Subsection 6.1, the effect of the bounding rules presented in Section 5 is examined. Subsection 6.2 provides a comparison of our algorithm with the one proposed by Sprecher (cf. [19]), which is currently the most powerful exact method available to solve the MRCPSP.

6.1 Effect of the Bounding Rules

This section focuses on the computational results concerning the algorithm of Table 5 in accordance with the bounding rules presented in Section 5. Recall, Bounding Rule 1 is a precedence based lower bound. Bounding Rule 2 provides a reduction of the input data before the algorithm is started. Bounding Rule 3 checks whether the current partial schedule is completable w. r. t. the nonrenewable resources or not. Bounding Rule 4 induces backtracking if it is certain that the current partial schedule cannot lead to a complete schedule which is tight and mode-minimal. Bounding Rule 5 excludes some partial schedules from consideration which cannot be completed to semi-active schedules. Bounding Rules 6 and 7 reduce the examination of the set of delay alternatives to a single alternative.

We have tested a basic variant of the procedure, that is, the algorithm of Table 5 enhanced by Bounding

Rule 1. Moreover, we have separately extended the basic variant by each of the remaining bounding rules. Doing so, we can compare the results with the ones obtained by Sprecher (cf. [19], p. 91). Note, the precedence based lower bound (Bounding Rule 1) is already included in the basic version of Sprecher's procedure. Finally, we have tested the combined effect of all bounding rules.

Table 11 displays the average computation times μ_{CPU}, the standard deviations σ_{CPU}, and the maximum computation times $\max_{CPU}$ obtained by solving the 536 instances with the different variants of the algorithm. Furthermore, the comparison factor is given in the last column. It reflects the comparison of the average CPU-time of the basic variant including, Bounding Rule 1, with the average CPU-time of the variant under consideration. Bounding Rule 3 is the most powerful one with a comparison factor of approximately five. Bounding Rules 4 and 6 have a comparison factor of approximately two. The left shift rule (Bounding Rule 5) with a comparison factor of 1.3 is less effective than reported by Sprecher for his algorithm. This is partly due to the different structures of the enumeration schemes. Furthermore, Sprecher's formulation of the left shift rule reduces the search space to semi-active schedules while the one presented here does not. The effect of Bounding Rule 7 is almost completely consumed by its additional computational effort for checking the assumptions.

Table 12 displays the average computation times for the different levels of RF_R, RS_R, RF_N, and RS_N. The last row contains the comparison factors of the basic variant with the variant using all bounding rules. Sprecher reported a lack of bounding rules for instances with a high resource factor RF_R and a low resource strength RS_R. It seems that this gap has been closed by employing Bounding Rule 6 which cannot be used in Sprecher's algorithm. The new Bounding Rule 2 is most effective when the resource strength of the nonrenewable resources RS_N is either very high or very low. If it is very high, the nonrenewable resources are likely to be redundant. Recall, this may force modes to become inefficient. In contrast, if it is very low, some modes may be non-executable w. r. t. a nonrenewable resource. Furthermore, for some parameter levels, the algorithm is slowed down by the Bounding Rules 5 and 7. Finally, the frequency distributions of the computation times are shown in Table 13.

6.2 Comparison with Sprecher's Algorithm

In this section, we compare our algorithm with the one suggested by Sprecher (cf. [19]). Note, our procedure has been designed for the MRCPSP while Sprecher's approach additionally covers resource availability varying with time. For the computational studies reported in this section we used the accelerated versions of both algorithms. That is, the seven bounding rules described in Section 5 have been employed to speed up the algorithm of Table 5. Sprecher's procedure includes the four bounding rules designed for instances with nonrenewable resources (cf. [19], p. 53). We used the original implementation provided by Sprecher. In order to obtain comparable results, we have recompiled Sprecher's source code with the same compiler (Borland C) that has been used for our algorithm.

Table 14 shows the average computation times μ_{CPU}, the standard deviations σ_{CPU}, and the maximum computation times $\max_{CPU}$ of both algorithms. Moreover, the comparison factor which is given in the last column shows that our algorithm is approximately four times faster than Sprecher's.

Table 15 displays the average computation times and the comparison factors for the different levels of RF_R, RS_R, RF_N, and RS_N. It can be observed that our algorithm is faster for all parameter levels; the comparison factors range between 2.06 and 22.88.

Finally, the frequency distributions of the computation times, given in Table 16, show the superiority of our algorithm. While our procedure has solved 87% of the problems within one second, Sprecher's approach has

Bounding Rule(s)	μ_{CPU}	σ_{CPU}	$\max_{CPU}$	Factor
1	12.84	72.17	1439.34	1.00
1,2	8.04	32.42	485.55	1.60
1,3	2.48	13.92	290.33	5.17
1,4	6.09	24.02	339.45	2.11
1,5	9.89	52.03	982.58	1.30
1,6	5.96	21.67	300.17	2.15
1,7	11.71	54.76	964.07	1.10
1–7	0.53	1.33	13.68	24.08

Table 11: Effect of the Bounding Rules — All Instances

Bounding	RF_R		RS_R				RF_N		RS_N			
Rule(s)	0.5	1.0	0.2	0.5	0.7	1.0	0.5	1.0	0.2	0.5	0.7	1.0
1	2.62	22.39	43.02	8.04	3.69	0.95	3.28	20.13	64.62	6.30	3.14	3.41
1,2	1.94	13.75	25.96	5.55	2.55	0.70	1.99	12.66	35.74	6.22	3.13	1.01
1,3	0.33	4.49	9.65	0.83	0.36	0.11	3.21	1.93	1.45	2.61	1.97	3.40
1,4	1.34	10.53	20.28	3.67	1.76	0.69	2.64	8.72	23.92	4.42	2.45	2.56
1,5	1.80	17.46	35.18	4.60	2.52	0.92	3.31	14.91	45.29	5.53	3.07	3.53
1,6	2.28	9.39	12.16	7.94	3.68	0.95	0.75	9.93	31.67	3.49	1.09	0.54
1,7	2.49	20.33	38.02	7.98	3.68	0.95	3.59	17.90	56.04	6.15	3.19	3.82
1–7	0.20	0.84	1.37	0.54	0.24	0.09	0.38	0.65	0.52	0.92	0.59	0.08
Factor	13.10	26.65	31.40	14.89	15.38	10.56	8.63	30.97	124.27	6.85	5.32	39.25

Table 12: Effect of the Bounding Rules — Full Factorial Design

Bounding Rule(s)	[0,0.1]	(0.1,1]	(1,5]	(5,10]	(10,25]	(25,50]	(50,100]	(100,250]	> 250
1	198	124	87	40	39	19	16	8	5
1,2	205	128	93	43	31	16	12	6	2
1,3	224	188	71	22	24	4	2	-	1
1,4	193	149	94	40	33	15	8	2	2
1,5	200	135	80	45	37	18	12	5	4
1,6	198	142	96	39	33	13	9	5	1
1,7	198	125	86	41	38	19	16	9	4
1–7	240	227	58	8	3	-	-	-	-

Table 13: Effect of the Bounding Rules — Frequency Distribution

only been capable to solve 69% within the same time.

Algorithm	μ_{CPU}	σ_{CPU}	$\max_{CPU}$	Factor
Sprecher	2.18	6.96	112.03	1.00
MRCPSP	0.53	1.33	13.68	4.11

Table 14: Comparison with Sprecher's Algorithm — All Instances

	RF_R		RS_R				RF_N		RS_N			
Algorithm	0.5	1.0	0.2	0.5	0.7	1.0	0.5	1.0	0.2	0.5	0.7	1.0
Sprecher	0.84	3.43	7.24	1.11	0.62	0.47	1.97	2.33	2.21	2.94	1.75	1.83
MRCPSP	0.20	0.84	1.37	0.54	0.24	0.09	0.38	0.65	0.52	0.92	0.59	0.08
Factor	4.20	4.08	5.28	2.06	2.58	5.22	5.18	3.58	4.25	3.20	2.97	22.88

Table 15: Comparison with Sprecher's Algorithm — Full Factorial Design

Algorithm	[0,0.1]	(0.1,1]	(1,5]	(5,10]	(10,25]	(25,50]	(50,100]	(100,250]	> 250
Sprecher	114	255	117	28	16	4	1	1	-
MRCPSP	240	227	58	8	3	-	-	-	-

Table 16: Comparison with Sprecher's Algorithm — Frequency Distribution

7 Conclusions

In this paper we have presented a new branch-and-bound algorithm for minimizing the project's makespan of a multi-mode resource-constrained project scheduling problem. By the use of mode alternatives the algorithm considerably extends the concept of delay alternatives which is used in the currently most powerful solution procedure for single-mode resource-constrained project scheduling.

The basic enumeration scheme has been enhanced by static and dynamic search tree reduction techniques. The static rules can be used by any algorithm for solving a multi-mode resource-constrained project scheduling problem. The dynamic rules substantially extend the dominance concepts from the single-mode to the multi-mode case.

Computational experience gained with the algorithm applied to the standard sample of project instances generated by ProGen indicates a substantial reduction of the solution times obtainable with the currently most powerful procedure published in the literature. Beside the mean CPU time the variance is substantially decreased.

References

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

[2] BOCTOR, F.F. (1993): Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research, Vol. 31, pp. 2547-2558.

[3] 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.

[4] DEMEULEMEESTER, E. (1992): Optimal algorithms for various classes of multiple resource-constrained project scheduling problems. PhD Dissertation, Katholieke Universiteit Leuven, Belgium.

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

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

[7] DREXL, A. AND J. GRÜNEWALD (1993): Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions, Vol. 25, pp. 74-81.

[8] FRENCH, S. (1982): Sequencing and scheduling: An introduction to the mathematics of the job-shop. Wiley, New York.

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

[10] HARTMANN, S. AND A. SPRECHER (1993): A note on "Hierarchical models for multi-project planning and scheduling". Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 338, Kiel.

[11] KOLISCH, R. (1995): Project scheduling under resource constraints – Efficient heuristics for several problem classes. Physica, Heidelberg (to appear).

[12] KOLISCH, R.; A. SPRECHER AND A. DREXL (1995): Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science (to appear).

[13] MINGOZZI, A.; V. MANIEZZO; S. RICCIARDELLI AND L. BIANCO (1994): An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. University of Bologna, Department of Mathematics, Technical Report No. 32, Bologna.

[14] 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.

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

[16] SCHRAGE, L. (1971): Solving resource-constrained network problems by implicit enumeration - non-preemptive case. Operations Research, Vol. 18, pp. 263-278.

[17] SLOWINSKI, R. (1980): Two approaches to problems of resource allocation among project activities: A comparative study. Journal of the Operational Research Society, Vol. 31, pp. 711-723.

[18] SPERANZA, M.G. AND C. VERCELLIS (1993): Hierarchical models for multi-project planning and scheduling. European Journal of Operational Research, Vol. 64, pp. 312-325.

[19] SPRECHER, A. (1994): Resource-constrained project scheduling: Exact methods for the multi-mode case. Lecture Notes in Economics and Mathematical Systems, Vol. 409, Springer, Berlin et al.

[20] 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 (to appear).

[21] 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.

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