ELSEVIER

European Journal of Operational Research 107 (1998) 431–450

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm [1]

Arno Sprecher *, Andreas Drexl [2]

Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstraße 40, 24118 Kiel, Germany

Received 1 November 1996; received in revised form 1 May 1997

Abstract

In this paper we present an exact solution procedure of the branch-and-bound type for solving the multi-mode resource-constrained project scheduling problem. The basic enumeration scheme is enhanced by search tree reduction schemes which highly increase the performance of the algorithm. Among the benefits of the approach are ease of description, ease of implementation, ease of generalization, and, additionally, superior performance of the exact approach as well as reasonable heuristic capabilities of the truncated version. The procedure has been coded in C and implemented on a personal computer. Using the standard project generator ProGen we have established a wide range of instances. More than 10,000 problem instances have been systematically generated to evaluate the algorithm's performance. The experimental investigation illustrates: First, the effect of the bounding rules. Second, the superior performance of the exact approach and the capabilities of the truncated version; the size of the projects that can be solved to optimality has been nearly doubled. Third, the impact of the variation of several project characteristics on solution time and quality. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Project scheduling; Resource constraints; Multiple modes; Branch-and-bound; Heuristic; Computational results

1. Introduction

Whereas the standard methods of project scheduling, CPM and MPM, base on the assumption of unlimited availability of the resources involved, the modern concepts include more realistic aspects. In general, the availabilities of the resources involved are limited.

Consequently, numerous publications have dealt with exact methods for solving the so-called single-mode resource-constrained project scheduling problem (SRCPSP) where each of the activities comprising the project has to be performed in one prescribed way (mode) using certain amounts of the resources provided. The objective considered is the minimization of the makespan. Recent advances have incorporated a little more of reality. The activities can be executed in one out of several modes. The modes reflect alternative combinations of resources and belonging quantities employed to

* Corresponding author. E-mail: sprecher@bwl.uni-kiel.de.
[1] Supported by the Deutsche Forschungsgemeinschaft.
[2] E-mail: drexl@bwl.uni-kiel.de.

fulfill the tasks related to the activities. The activity duration is a discrete function of the employed quantities, that is, using this concept e.g. working-off an activity can be accelerated by raising the quantities coming into operation (time-resource-tradeoff). Moreover, by raising the quantities of some resources and reducing the quantities of others resource substitution (resource-resource-tradeoff) can be realized. The problem at hand is the multi-mode resource-constrained project scheduling problem (MRCPSP), which is commonly considered with makespan minimization as objective.

Whereas the exact methods for the single-mode problem are well documented in the literature the multi-mode problem has attracted less attention (cf. [1–7]). We will present an enumeration scheme for optimally solving the MRCPSP with makespan minimization as objective. The basic scheme can be generalized for dealing with any regular measure of performance and enhanced by powerful acceleration schemes. The approach considerably extends the precedence tree guided enumeration scheme proposed by Patterson et al. (cf. [2,3]). The benefits of the approach are (1) ease of description, (2) ease of implementation, (3) ease of generalization, (4) superior performance of the exact approach, and (5) reasonable heuristic capabilities of the truncated version.

The paper is organized as follows: Section 2 is devoted to the detailed description of the model under consideration. Section 3 addresses the basic branch-and-bound procedure and the search tree reduction schemes. Section 4 reports about the computational experience we gained with the exact approach. The separate and common effect of the bounding rules are illustrated. The impact of some project characteristics on the solution times are studied. In addition, results obtained by the truncated version are presented. Section 5 draws the conclusions.

2. The model

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

Each activity j, $j = 1, \ldots, J$, may be executed in one out of M_j modes. The activities may not be preempted and a mode once selected may not change, i.e., a job j once started in mode m has to be completed in mode m without interruption. Performing job j in mode m takes d_{jm} periods and is supported by a set R of renewable, a set N of nonrenewable and a set D of doubly constrained resources. Considering a horizon, that is, an upper bound $\overline{T}$ on the project's makespan, we have an available amount of K^{ρ}_{rt} (K^{δ}_{rt}) units of renewable (doubly constrained) resource r, $r \in R$ ($r \in D$), in period t, $t = 1, \ldots, \overline{T}$. The overall capacity of the nonrenewable resource r, $r \in N$, and doubly constrained resource r, $r \in D$, is given by K^{ν}_{r} and K^{δ}_{r}, respectively. If job j is scheduled in mode m then k^{ρ}_{jmr}, $k^{\rho}_{jmr} \geqslant 0$, units of renewable resource r, $r \in R$, are used, and k^{δ}_{jmr}, $k^{\delta}_{jmr} \geqslant 0$, units of doubly constrained resource r, $r \in D$, are consumed each period job j is in process. Additionally, k^{ν}_{jmr}, $k^{\nu}_{jmr} \geqslant 0$, units of nonrenewable resource r, $r \in N$, are consumed. The parameters are assumed as integer-valued. The objective is to find a makespan minimal schedule that meets the constraints imposed by the precedence relations and the limited resource availabilities. Clearly, since the doubly constrained resources can easily be taken into account by appropriately enlarging the sets of renewable and nonrenewable resources they do not have to be considered explicitly.

Presuming feasibility and a constant per-period availability of the renewable resources, an upper bound on the minimum makespan is given by the sum of maximum activity durations. Given an upper bound $\overline{T}$ on the project's makespan we can use the precedence relations and the modes of shortest duration to derive time windows, i.e. intervals [EF_j,LF_j], with earliest finish time EF_j and latest finish time LF_j, containing the precedence feasible completion times of activity j, $j = 1, \ldots, J$, by traditional forward and backward recursion as performed in MPM. Analogously, the interval [ES_j,LS_j] bounded from below and above by the earliest start time ES_j and the latest start time LS_j, respectively, can be calculated to reflect the precedence feasible start times. The benefit is twofold: First, the number of variables used in the integer (binary) programming formulation is reduced substantially. Second, within a branch-and-bound (BB) algorithm the bounds can be efficiently used to speed up the convergence. Note, since different modes may have different durations, starting an activity j within the time window [ES_j,LS_j] does not necessarily mean that the job is completed in the interval [EF_j,LF_j]. Using the time windows derived and binary decision variables the problem can be stated as a linear program (cf. [7]).

The model can be enhanced to include time-varying requests (cf. [8]) and generalized temporal constraints (cf., e.g., [5], p. 19). Additionally, beside (a) the minimization of the makespan, further objectives, like, e.g., (b) the minimization of the weighted delays, (c) the minimization of the total number of tardy activities, (d) the minimization of the mean weighted flow time, (e) the maximization of the net-present value, (f) the minimization of the total (weighted) resource consumption, and (g) the smoothness of the resource profile can be easily modeled (cf., e.g., [9]).

Considering the definition of a regular performance measure given in [10] for the SRCPSP we can easily extend it to the MRCPSP (cf. [5], p. 22). Roughly speaking, the definition says, a performance measure is called regular, if the objective function value for a given schedule is not made worse, if the completion time of a single activity is reduced without changing the activity's mode. Following the definition, the objectives (a)–(f), are regular and the objective (g) not. Note, more precisely, the maximization of the net-present value is a regular measure of performance if the coefficients of the objective function are nonincreasing with respect to the period index and, moreover, the objective function is multiplied by -1.

The generalized problem we obtain by replacing the makespan minimization by any regular measure of performance is referred to as the generalized resource-constrained project scheduling problem (GRCPSP).

3. The branch-and-bound algorithm

In this section we present and analyze an algorithm for the MRCPSP. The foundation stone of the basic algorithm is laid through a search tree reduction proposed by Patterson et al. (cf. [2,3]). By implicitly using dominating start time assignments, they made a major contribution to improve earlier versions (cf. [7,11]) to a precedence tree guided enumeration scheme. The algorithm is completely restructured and revised. The revised enumeration scheme bears several benefits: (1) ease of description, (2) ease of implementation, (3) ease of generalization.

We proceed as follows: In Section 3.1 the precedence tree which guides the search for an optimal solution is described. In Section 3.2 the detailed realization of the algorithm for minimizing the project's makespan is outlined. Moreover, we briefly summarize adaptations for dealing with generalized problem settings. In Section 3.3 we give a brief summary of the search tree reduction schemes employed to accelerate the algorithm. A more detailed analysis and illustrative examples can be found in [9,12].

3.1. The precedence tree

The main problem to solve in development stage of an exact solution procedure of the branch-and-bound type is the construction of the enumeration tree. Roughly speaking, its responsibility is to guide the search for an optimal solution by the successive decomposition of the problem into subproblems by splitting the feasible region and fixing variables. Obviously, the inclusion of parts of the constraints bears benefits, even if the subproblems are dealt with standard relaxations, e.g. LP relaxation or Lagrangian relaxation. The problem at hand allows a design of the control process which explicitly takes the precedence constraints into account.

The nodes of the precedence tree correspond to the nodes of the branch-and-bound tree. The root node 1 of the tree is given by the single start activity and the leaves are copies of the only finish activity J. The descendants of a node i within the precedence tree are built by the activities that are eligible after scheduling the activities on the path leading from the root node 1 to node i. Thereby, an activity is called eligible if all its predecessors are scheduled. We use the set $\mathscr{ACS}_i$ to denote the set of activities currently scheduled up to level i. Assuming that passing the nodes of the precedence tree means scheduling the activities associated with the nodes we obtain the set of eligible activities on level i, namely Y_i, as follows:

$$
\begin{aligned}
&Y_1 := \{g_1\} = \{1\},\\
&\mathscr{ACS}_1 := \{g_1\} = \{1\},\\
&Y_{i+1} := Y_i \setminus \{g_i\} \cup \{k \in \mathscr{S}_{g_i}; \mathscr{P}_k \subseteq \mathscr{ACS}_i\},\\
&i = 1,\dots,J-1,\\
&\mathscr{ACS}_{i+1} := \mathscr{ACS}_i \cup \{g_{i+1}\}, \quad i = 1,\dots,J-1.
\end{aligned}
$$

Note, only those successors of an activity g_i scheduled on level i become eligible on level $i+1$, the predecessors of which are already scheduled.

The problem now arising is when to start activity g_i on level i. Due to the precedence constraints activity g_i can only be started after the completion of all its predecessors, but, in spite of this limitation several feasible start times can be assigned to the activity currently under consideration (cf. [5], p. 39). Nevertheless, the investigation can be reduced to a single alternative if a regular measure of performance is considered as objective.

We extend our considerations to the multi-mode case and state the theorem, which, roughly speaking, reduces the examination of one path from the root to a leaf of the precedence tree to at most one start time assignment per activity.

Theorem 1 (cf. [9]). *Let (P) be a scheduling problem of the type GRCPSP. If (P) is feasible, then there exists a permutation of the activities* $1,\dots,J$, *denoted as* $g_1,\dots,g_J$, *and accompanying modes* $m_{g_1},\dots,m_{g_J}$, *such that, the schedule with start times* $\mathrm{ST}_{g_1},\dots,\mathrm{ST}_{g_J}$ *fulfilling (a)* $\mathrm{ST}_{g_i} \leqslant \mathrm{ST}_{g_{i+1}}$, $i = 1,\dots,J-1$, *and (b)* ST_{g_i}, $i = 1,\dots,J$, *is the lowest feasible start time of activity* g_i *scheduled in mode* m_{g_i} *with respect to the precedence relations and the leftover capacities after scheduling activities* $g_1,\dots,g_{i-1}$ *in accordance with (a), represents an optimal solution.*

The search space reduction by Theorem 1 is substantial (cf. [5], p. 39). However, note, the start times of the activities in conditions (a) and (b) of Theorem 1 cannot be replaced by their completion times as suggested in [2] without losing optimality (cf. [5], p. 37).

3.2. The enumeration scheme

Using the preliminaries presented in Section 3.1 we can concisely state the algorithm: The algorithm schedules one activity per node of the branch-and-bound tree. An activity is firstly considered for scheduling in its first mode when all of its predecessors are scheduled. The start time of the activity under consideration is the lowest feasible start time which (a) is not less than the start time of the activity most recently scheduled and (b) does not violate the precedence or resource constraints. If scheduling in the current mode is infeasible then the next mode is tested. If there is no untested mode left then the next eligible activity is selected. If there is no untested eligible activity left then backtracking to the previous level is performed. At this level the next mode or eligible activity is chosen.

The summary of the algorithm turns out a degree of freedom concerning the selection of the activity, more precisely the activity/mode combination, to be considered for scheduling on the current level first, second and so forth. Avoiding notational overhead we are leaving priority rules for later discussion and refer to them only if there is the necessity for doing so.

We will now give a more formal description. The notation used to describe the algorithm is displayed in Table 1.

The algorithm for the minimization of the makespan is presented in Table 2. In Step 1 we ini-

Table 1
Notation used in the BB algorithm

i	Level index
i^*	Lowest index which produces a time window violation after the recalculation
Y_i	Set of eligible activities on level i
$\hat{N}_i$	Cardinality of the set Y_i
N_i	Index of the element from the eligible set Y_i which is currently under consideration
Y_{iN_i}	The N_ith element of the set of eligible activities on level i
$\mathcal{ACS}_i$	Set of activities currently scheduled up to level i
g_i	Activity currently scheduled or under consideration on level i
m_{g_i}	Mode activity g_i is currently scheduled in or considered to be scheduled in on level i
ST_{g_i} (CT_{g_i})	Start (completion) time of activity g_i scheduled on level i
t_P	Lowest feasible start time of activity g_i with respect to the precedence relation
t_I	Lowest feasible start time of activity g_i with respect to condition (a) of Theorem 1
Φ^*	Objective function value of the currently best known solution
$\overline{\Phi}(\cdot)$	Function, which assigns each partial schedule a lower bound on the objective function value of a completion; the bound is equal to the objective function value, if a complete schedule is considered.

tialize the variables used. We avoid the case distinction whether any activity is already scheduled or not by defining $g_0 := 0$, $\mathrm{ST}_{g_0} := 0$, $\mathrm{CT}_{g_0} := 0$ and $\mathcal{P}_1 := \{0\}$. The set of eligible activities on the first level ($i = 1$) is defined as $Y_1 := \{1\}$ with corresponding number of eligible activities on the first level $\hat{N}_1 := |Y_1| = 1$. The first and only activity to be tested is the unique start activity, thus we have $g_1 = Y_{11} = 1$. In general, we denote with Y_{iN_i} the N_ith element of the eligible set Y_i where the activities in Y_i, except where stated otherwise, are arranged with respect to increasing job number.

Table 2
Minimizing the project's makespan

```
Step 1:   (Initialization)
          $\mathcal{ACS}_0 := \emptyset$; $g_0 := 0$; $ST_{g_0} := 0$; $CT_{g_0} := 0$; $\mathcal{P}_1 := \{0\}$; $i := 1$; $Y_1 := \{1\}$; $\hat{N}_1 := 1$; $N_1 := 1$;
          $g_1 := 1$; $m_1 := 0$;
Step 2:   (Select next untested mode or descendant)
          If $m_{g_i} < M_{g_i}$ then $m_{g_i} := m_{g_i} + 1$ and goto Step 4;
          If $N_i < \hat{N}_i$ then $N_i := N_i + 1$; $g_i := Y_{iN_i}$; $m_{g_i} := 1$ and goto Step 4;
Step 3:   (Single-level backtracking)
          $i := i - 1$; if $i = 0$ then STOP, else remove job $g_i$ from partial schedule; readjust resource arrays
          and goto Step 2;
Step 4:   (Find feasible start time)
          $t_P := \max\{CT_k; k \in \mathcal{P}_{g_i}\}$ $t_I := ST_{g_{i-1}}$; $t^* := \max\{t_P, t_I\}$; determine the earliest resource
          feasible start time $\bar{t}$, $t^* \leqslant \bar{t} \leqslant LF_{g_i} - d_{g_i m_{g_i}}$, of job $g_i$ in mode $m_{g_i}$; if scheduling is impossible
          goto Step 2, else set $ST_{g_i} := \bar{t}$; $CT_{g_i} := \bar{t} + d_{g_i m_{g_i}}$; $\mathcal{ACS}_i := \mathcal{ACS}_{i-1} \cup \{g_i\}$ and adjust resource arrays;
Step 5:   If $i = J$ then goto Step 7;
Step 6:   (Update the eligible set)
          $i := i + 1$; calculate the new descendant set $Y_i := Y_{i-1} \setminus \{g_{i-1}\} \cup \{k \in \mathcal{S}_{g_{i-1}}; \mathcal{P}_k \subseteq \mathcal{ACS}_{i-1}\}$;
          set $\hat{N}_i := |Y_i|$; $N_i := 1$; $g_i := Y_{i1}$; $m_{g_i} := 0$ and goto Step 2;
Step 7:   (Store solution and adjust time bounds)
          Store solution $g_j, m_{g_j}, ST_{g_j}$,   $j = 1, \ldots, J$;
          Set $LS_j := LS_j - (LF_J - CT_J + 1)$,   $j = 1, \ldots, J$;
          and $LF_j := LF_j - (LF_J - CT_J + 1)$,   $j = 1, \ldots, J$;
Step 8:   (Calculate lowest indexed level violating the time window)
          $i^* := \min\{k \in \{1, \ldots, J\}; CT_{g_k} > LF_{g_k}\}$;
Step 9:   (Multi-level backtracking)
          Readjust resources used (consumed) by jobs $g_k$ in mode $m_{g_k}$, $k = J, \ldots, i^*$;
          $i := i^*$ and goto Step 2.
```

In Step 2 a job/mode combination is selected for assignment. Two cases have to be distinguished. First, if there are further modes for the activity (descendant) currently examined, that is, $m_{g_i} < M_{g_i}$, then the next mode is selected, i.e. $m_{g_i} := m_{g_i} + 1$. Second, if for the current descendant g_i the last mode has been tested, that is, $m_{g_i} = M_{g_i}$, then the next descendant is chosen and the first mode is selected for assignment, i.e. $N_i := N_i + 1$, $g_i := Y_{iN_i}$, and $m_{g_i} := 1$. If no more untested modes and descendants are available, then single-level backtracking (Step 3) has to be performed, otherwise we goto Step 4.

In Step 4 we first calculate the lowest feasible start time t_{P} with respect to the currently scheduled activities and the precedence relations, that is, $t_{\mathrm{P}} := \max\{\mathrm{CT}_k; k \in \mathscr{P}_{g_i}\}$. Subsequently we calculate the minimal feasible start time t_{I} due to condition (a) of Theorem 1 and define $t^* := \max\{t_{\mathrm{P}}, t_{\mathrm{I}}\}$. We then scan the interval $[t^*, \mathrm{LF}_{g_i} - d_{g_i m_{g_i}}]$ for the earliest contiguous interval $d_{g_i m_{g_i}}$ periods long where activity g_i can be scheduled in mode m_{g_i} without violating the renewable resource constraints. That is, we search $\bar{t}$, $t^* \leqslant \bar{t} \leqslant \mathrm{LF}_{g_i} - d_{g_i m_{g_i}}$, such that the leftover capacities of the renewable resources r, $r \in R$, in the periods t, $t = \bar{t} + 1, \ldots, \bar{t} + d_{g_i m_{g_i}}$, are at least equal to the per-period requirements $k^{\rho}_{g_i m_{g_i} r}$. Note, as previously mentioned, since different modes may have different durations, searching the interval $[t^*, \mathrm{LS}_{g_i}]$ for a start time is not equivalent to searching the interval $[t^*, \mathrm{LF}_{g_i} - d_{g_i m_{g_i}}]$. In the former case it might happen that the bounds exposed by the latest finish times are violated without detecting it on the current level, and thus producing computational overhead. Additionally, feasibility with respect to nonrenewable resources has to be checked, i.e. the leftover capacities of the nonrenewable resources r, $r \in N$, are compared with the consumption $k^{\nu}_{g_i m_{g_i} r}$ of the activity currently considered. Obviously, since determining a feasible start time with respect to renewable resources usually consumes more time than comparing consumptions and the remaining availabilities of the nonrenewable resources, it is sensible to check the latter ones first.

If feasibility can be assured, we set the start time ST_{g_i} of activity g_i equal to $\bar{t}$ and the completion time CT_{g_i} equal to $\mathrm{ST}_{g_i} + d_{g_i m_{g_i}}$. Furthermore, the set $\mathscr{ACS}_i$ of activities currently scheduled, is updated, i.e., $\mathscr{ACS}_i := \mathscr{ACS}_{i-1} \cup \{g_i\}$, and the leftover capacities are adjusted. If feasibility cannot be assured, we return to Step 2 and determine the next job/mode combination. Successfully scheduling of activity g_i leads to Step 5.

In Step 5 we check if all the activities are scheduled. If this holds true, we have found the first feasible solution or an improved solution and we can skip to Step 7, where the new solution is stored and the critical path bounds LS_j, LF_j, $j = 1, \ldots, J$, are recalculated. Since our goal is the improvement of the currently best known solution LS_j and LF_j are reduced by the improvement of the former best known solution incremented by one, that is $\mathrm{LF}_J - \mathrm{CT}_J + 1$.

If not all the activities are scheduled, we step over to the next level, $i := i + 1$, and calculate the new descendants, i.e., the set of eligible activities Y_i, the number of descendants $\hat{N}_i := |Y_i|$, select the first descendant $N_i := 1$, $g_i := Y_{i1}$, initialize the mode variable $m_{g_i} := 0$ and go to Step 2.

After the adaptation of the critical path bounds in Step 7 we determine the lowest level index i^*, where the newly derived critical path bounds are violated by the current schedule. Multi-level backtracking is then performed in Step 9, that is, all the activities g_i, which have been scheduled on a level i, $i \geqslant i^*$, are removed from the schedule, the resource availabilities are readjusted and the current level index is set equal to i^*.

The algorithm terminates if in Step 2 no more job/mode assignment is possible and decrementing the level index in Step 3 leads to $i = 0$. Otherwise, if the level index is greater than zero after decrementing it in Step 3, then g_i is removed from the partial schedule and resource availabilities are readjusted.

A thorough study of the enumeration scheme shows that its current state is entirely described by (a) the currently best known solution and bound on the objective function, (b) the sequences of the job/mode combinations the eligible activities have to be examined in, and (c) the job/mode combinations $[g_j, m_{g_j}]$, $j = 1, \ldots, i$, scheduled up to the current level i.

Clearly, in accordance with Theorem 1 and Step 4 of the algorithm it is not necessary to keep the start time of the activities. However, the enhanced representation will simplify the later discussion and the implementation of acceleration schemes.

We can adapt the algorithm presented for optimizing any regular measure of performance. Due to Theorem 1 only minor changes are necessary in order to attack scheduling problems of the type GRCPSP. That is, the enumeration is again guided by the precedence tree and only the evaluation of bounds on the objective function value has to be adapted and explicitly incorporated (Steps 4, 7–9), respectively.

Performing Step 4 as described above follows two intentions. On the one hand, if an activity g_i (in mode m_{g_i}) is assigned a start time $\bar{t}$, $\bar{t} \leqslant \mathrm{LF}_{g_i} - d_{g_i m_{g_i}}$, then the partial schedule $\mathcal{PS}_i$ is at least completable with respect to the precedence relations. On the other hand, since the latest finish times are adjusted after the improvement of the current best solution, the adapted latest start and finish times (Step 7) offer a new bound on the objective function value (makespan) obtainable from a completion of the current partial schedule. That is, if we would assign a start time $\bar{t}$, $\bar{t} > \mathrm{LF}_{g_i} - d_{g_i m_{g_i}}$, to an activity g_i then the partial schedule is not completable with a makespan T, $T \leqslant \mathrm{LF}_J$.

For notational convenience, we denote the objective function with Φ and let Φ^* denote the objective function value of the current best solution. Furthermore, $\overline{\Phi}(\cdot)$ denotes a function which determines a lower bound on the objective function of the completion for a given partial schedule $\mathcal{PS}_i$. The bound matches with the objective function value of $\mathcal{PS}_i$ if a complete schedule is the argument, i.e. $i = J$. With this in mind, we rearrange Steps 4, 7–9 to Steps 4′ and 7′ (cf. Table 3), respectively.

Remark 1. In contrast to [2], p. 13, we included lower bound evaluation explicitly and prevent computational overhead by restarting the procedure after determining an improved best solution.

Although, the (modified) enumeration scheme can deal with a wide variety of scheduling problems, further modifications are necessary if, e.g., (maximum) time-lags between the activities or time-varying resource requests for renewable resources have to be taken into account (cf. [5], p. 67). Both extensions of the model require beside the job/mode combination $[g_i, m_{g_i}]$ currently considered a third dimension which saves the start time ST_{g_i} currently evaluated. That is, before switching to another job/mode combination the remaining feasible start times have to be examined. Another job/mode combination for the current job/mode combination is chosen if no further feasible start times are left. However, doing so, only minor changes are necessary to optimize any objective and taking into account time-varying requests as well as time-lags. In contrast, job specific constraints on the start and finish times imposed by a release date and a deadline can be implemented by simple time window adaptations.

3.3. Search tree reduction

By the exclusion of partial schedules from further continuation we can reduce the enumeration

Table 3
Optimizing a regular measure of performance

Step 4′:	(Find feasible start time)
	$t_P := \max\{CT_k; k \in \mathscr{P}_{g_i}\}$; $t_I := ST_{g_{i-1}}$; $t^* := \max\{t_P, t_I\}$; determine the earliest resource
	feasible start time $\bar{t}$, $t^* \leqslant \bar{t} \leqslant LF_{g_i} - d_{g_i m_{g_i}}$, of job g_i in mode m_{g_i}; if scheduling is impossible or
	$\overline{\Phi}(\mathcal{PS}_i) \geqslant \Phi^*$ then goto Step 2, else set $ST_{g_i} := \bar{t}$;
	$CT_{g_i} := \bar{t} + d_{g_i m_{g_i}}$; $\mathcal{ACS}_i := \mathcal{ACS}_{i-1} \cup \{g_i\}$
	and adjust resource arrays;
Step 7′:	(Store solution and adjust the bounds)
	Store solution g_j, m_{g_j}, ST_{g_j}, $j = 1, \ldots, J$, Φ^*;
	remove job g_J from partial schedule; readjust resource arrays and goto Step 2;

tree guiding the search for an optimal solution. Clearly, for optimizing the given objective we have to assure that the reduction does not make worse the (optimal) solutions obtainable. That is, we have to elaborate conditions on which a partial schedule need not be extended without losing optimality. These conditions build the bounding rules we will present. A very illustrative example, proofs and additional hints concerning the implementation of the rules are given in [9,12].

The rules are, if not otherwise mentioned, applicable when optimizing any regular measure of performance. We distinguish two types of bounding rules, the static and the dynamic ones. Static rules can be implemented by the adaptation of the input data. The enumeration scheme remains unchanged and the rules can be employed by any algorithm solving the problem at hand. In contrast, the dynamic rules are incorporated into the algorithm. As a rule their sensible application depends on the algorithm employed.

The first static rule eliminates modes and/or nonrenewable resources from the input data. That is, the number of (partial) schedules and/or constraints to be considered are reduced before the solution procedure is started. Although the rule is quite simple it can help increasing the algorithm's performance, especially when a large project with a number of activities, hardly to manage manually, is dealt with, or if makespan minimization problems have to be solved iteratively, like, e.g., in resource investment and resource allocation problems.

Rule 1 (Input data reduction rule): Calling a mode m_j nonexecutable if in a feasible schedule activity j cannot be performed in mode m_j, nonexecutable modes can be eliminated without effect on the set of feasible solutions. Calling a mode m_j inefficient with respect to the objective Φ if there is another mode m'_j, $1 \leqslant m'_j \leqslant M_j$, of activity j, such that any feasible schedule $\mathscr{PS}_J$ with activity j being performed in mode m_j has a corresponding feasible schedule $\overline{\mathscr{PS}_J}$ with activity j being performed in mode m'_j with $\Phi(\overline{\mathscr{PS}_J}) \leqslant \Phi(\mathscr{PS}_J)$, inefficient modes can be eliminated without effect on the value of the optimal solution obtainable. Calling a resource redundant if the constraints related to resource r can be neglected without influence on the set of feasible schedules, redundant resources can be eliminated. Criteria for identifying nonexecutable modes, inefficient modes, and redundant resources can be found in [9]. Note, the example given in [6] shows that deleting a mode which is nonexecutable w.r.t. a renewable resource may force a mode of another activity to become nonexecutable w.r.t. a nonrenewable resource. Moreover, removing a nonexecutable 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 mode may cause redundancy of a nonrenewable resource. Therefore, the project's input data should be prepared by the iterations described in [6].

The next bounding rule to be presented is designed for instances with nonrenewable resources, that is, $|N| > 0$. It checks completability of the partial schedule currently considered. A dynamic variant has been proposed by Drexl (cf. [13]) for a less general model. Sprecher (cf. [5]) adapted the rule to the GRCPSP and substantially increased its efficiency by reformulating it as a preprocessing, i.e. static, rule.

Rule 2 (Input data adjustment rule): If a given partial schedule $\mathscr{PS}_i$ cannot be completed with respect to the leftover capacities of the nonrenewable resources then backtracking can be performed. The partial schedule $\mathscr{PS}_i$ is not completable if there is a resource r, $r \in N$, with

$$K_r^v - \sum_{j=1}^{i} k_{g_j m_j r}^v < \sum_{j \in \{1,\ldots,J\}; j \notin \mathscr{ACS}_i} \min\{k_{jmr}^v; m = 1,\ldots,M_j\}.$$

Since the rule checks completability of the current partial schedule it is applicable when optimizing any (regular) objective. The dynamic formulation can be transferred to a static one, by defining

$$kmin_{jr}^v := \min\{k_{jmr}^v; m = 1,\ldots,M_j\},$$
$$j = 1,\ldots,J, \quad r \in N$$

and adjusting the input data as follows:

$$\overline{k}_{jmr}^v := k_{jmr}^v - kmin_{jr}^v, \quad j = 1,\ldots,J,$$
$$m = 1,\ldots,M_j, \quad r \in N$$

and

$$\overline{K}_r^v := K_r^v - \sum_{j=1}^{J} kmin_{jr}^v, \quad r \in N.$$

After the input data adjustment of Rule 2 each pair of a nonrenewable resource r, $r \in N$, and an activity j, $j = 1, \ldots, J$, has at least one mode m with an adapted consumption of zero units, that is, the resource factor (cf. [14]) of the instance is reduced.

Obviously, the effect of commonly employing Rules 1 and 2 is the strongest, if, first, the input data is reduced and then adjusted.

Rule 3 (Non-delayability rule): If a partial schedule $\mathcal{PS}_i$ cannot be continued by scheduling a job g_{i+1} on level $(i+1)$ then backtracking can be performed. The rule can be used when optimizing any (regular) measure of performance.

Surely, the underlying idea is adaptable to a single job/mode combination $[g_{i+1}, m_{g_{i+1}}]$, that is, if a job g_{i+1} is not schedulable in mode $m_{g_{i+1}}$ on level $(i+1)$, with partial schedule $\mathcal{PS}_i$, then $[g_{i+1}, m_{g_{i+1}}]$ is not schedulable on a level $(i+k+1)$ with partial schedule $\mathcal{PS}_{i+k}$ continuing $\mathcal{PS}_i$. Unfortunately, the effect of this adaptation is completely consumed by the additional effort to be performed.

Rule 4 (Single enumeration rule): If the start times of two job/mode combinations $[g, m]$ and $[h, n]$ scheduled on levels $(i+1)$ and $(i+2)$, respectively, do not depend on the sequence the combinations are scheduled in, then only one sequence has to be selected to continue the current partial schedule $\mathcal{PS}_i$.

Using the scheduling strategy given in Theorem 1 the assumptions imply that the start times are identical. Given nonzero durations the assumptions can be verified by storing the start times in a three-dimensional array $\mathrm{PT}[i][g][m]$ as proposed in [9].

Obviously the rule is extendable: We consider an i-partial schedule $\mathcal{PS}_i$ and unscheduled activities $g_{i+1}, \ldots, g_{i+k}$, with accompanying modes $m_{g_{i+1}}, \ldots, m_{g_{i+k}}$. If the start times $\mathrm{ST}_{g_{i+1}}, \ldots, \mathrm{ST}_{g_{i+k}}$ of $g_{i+1}, \ldots, g_{i+k}$, scheduled on levels $i+1, \ldots, i+k$, in modes $m_{g_{i+1}}, \ldots, m_{g_{i+k}}$, do not depend on the permutation of $g_{i+1}, \ldots, g_{i+k}$, i.e. $\mathrm{ST}_{g_{i+1}} = \cdots = \mathrm{ST}_{g_{i+k}}$, then only one permutation has to be examined, i.e. $(k! - 1)$ continuations can be saved. Note, although only pairwise comparison of the start times are used in our implementation the generalized concept is realized (cf. [9]), that is, if the conditions hold only one sequence is continued.

The following bounding rule makes use of the fact that the set of semi-active schedules is a dominant set with respect to any regular measure of performance (cf. [15]). That is, for any regular measure of performance there is an optimal semi-active schedule. The bounding rule is the so-called local left-shift rule. It has already been successfully employed in [16,17].

Rule 5 (Local left-shift rule): If, by ignoring condition (a) of Theorem 1, activity g_{i+1} is additionally schedulable in mode $m_{g_{i+1}}$ with start time $\overline{\mathrm{ST}}_{g_{i+1}}$, $\overline{\mathrm{ST}}_{g_{i+1}} = \mathrm{ST}_{g_i} - 1 = \mathrm{ST}_{g_{i+1}} - 1$, without violating the precedence- and (renewable) resource-constraints, then the job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be skipped on level $(i+1)$ when continuing the partial schedule $\mathcal{PS}_i$.

By applying the local left-shift rule the enumeration is reduced to the set of semi-active schedules. However, we can extend the considerations to global left-shifts (with and without mode changes) to reduce the enumeration to active schedules.

Rule 6 (Global and multi-mode left-shift rule): Let $\mathcal{PS}_i$ be the partial schedule currently to be continued on level $(i+1)$ by scheduling job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ with start time $\mathrm{ST}_{g_{i+1}}$. If, by ignoring condition (a) of Theorem 1, job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be started at $\overline{\mathrm{ST}}_{g_{i+1}}$ with $\overline{\mathrm{ST}}_{g_{i+1}} \leqslant \mathrm{ST}_{g_i} - d_{g_{i+1}, m_{g_{i+1}}}$, then the job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be skipped on level $(i+1)$.

We can combine the global left-shift with a mode change to the multi-mode left-shift: If, by ignoring condition (a) of Theorem 1, a job/mode combination $[g_{i+1}, \overline{m}_{g_{i+1}}]$ with $k^v_{g_{i+1}, \overline{m}_{g_{i+1}}, r} \leqslant k^v_{g_{i+1}, m_{g_{i+1}}, r}$, for $r \in N$, can be started at $\overline{\mathrm{ST}}_{g_{i+1}}$ with $\overline{\mathrm{ST}}_{g_{i+1}} \leqslant \mathrm{ST}_{g_i} - d_{g_{i+1}, \overline{m}_{g_{i+1}}}$, then the job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be skipped on level $(i+1)$. Note, if $\mathrm{ST}_{g_i} = \mathrm{ST}_{g_{i+1}}$, and $\overline{\mathrm{ST}}_{g_{i+1}} = \mathrm{ST}_{g_i} - d_{g_{i+1}, m_{g_{i+1}}}$ then, we, more precisely, have detected a feasible local left-shift (cf. [15]). However, if no nonrenewable resources have to be taken into account, then detecting a feasible (global) left-shift allows to track back immediately.

By commonly applying the local left-shift rule and the global left-shift rule (with or without mode change) only active schedules are generated. However, since the binding effect may or may not be produced by scheduling the last activity tightness (cf. [4,1]) of the schedules cannot be guaranteed. Moreover, if a regular measure of performance, other than (a)–(d) of Section 2, has to be optimized, then a mode change is not allowed in general. Nevertheless, the assumptions can be enhanced in order to allow a mode change.

For notational convenience, we subsequently state rules for the minimization of the project's makespan. By adapting the assumptions they can be used for dealing with any regular measure of performance.

Rule 7 (multi-mode rule): Let $\mathcal{PS}_i$ be the partial schedule which has been continued by scheduling job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ on level $(i+1)$ to $\mathcal{PS}_{i+1}$. If the start time $\mathrm{ST}_{g_{i+2}}$ of job/mode combination $[g_{i+2}, m_{g_{i+2}}]$ currently considered to be scheduled on level $(i+2)$ is at least equal to the completion time of a job/mode combination $[g_{i+1}, \overline{m}_{g_{i+1}}]$ with $\overline{m}_{g_{i+1}} < m_{g_{i+1}}$ and $k^v_{g_{i+1},\overline{m}_{g_{i+1}},r} \leqslant k^v_{g_{i+1},m_{g_{i+1}},r}$, $r \in N$, then job/mode combination $[g_{i+2}, m_{g_{i+2}}]$ can be skipped on level $(i+2)$.

The next rule we present is a multi-mode cut-set rule. It extends in some ways earlier, only single-mode suitable, concepts as the network-cuts from Talbot and Patterson (cf. [11]), the dominance pruning from Stinson et al. (cf. [17]) and the cut-set rule from Demeulemeester and Herroelen (cf. [16]) to the multi-mode case. Roughly speaking, it compares the compactness of the schedule currently to be continued with the compactness of one previously studied.

For ease of representation we define: The cut-set $\mathcal{CS}(\mathcal{PS}_i)$ belonging to $\mathcal{PS}_i$ is defined as the set of activities currently scheduled up to level i, that is

$$\mathcal{CS}(\mathcal{PS}_i) := \bigcup_{j=1}^{i} \{g_j\}$$

and the maximum related completion time $\mathrm{CT}^{\max}(\mathcal{PS}_i)$ of the activities scheduled in $\mathcal{PS}_i$ is defined by

$$\mathrm{CT}^{\max}(\mathcal{PS}_i) := \max_{j=1}^{i} \{\mathrm{ST}_{g_j} + d_{g_j m_{g_j}}\}.$$

Rule 8 (Multi-mode cut-set rule I, dominated heads): Let $\mathcal{PS}_i$ be the partial schedule currently to be continued by scheduling job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ with start time $\mathrm{ST}_{g_{i+1}}$. If a previously evaluated partial schedule $\overline{\mathcal{PS}_i}$ has (a) the same cut-set, i.e., $\mathcal{CS}(\mathcal{PS}_i) = \mathcal{CS}(\overline{\mathcal{PS}_i})$, (b) a maximum completion time $\mathrm{CT}^{\max}(\overline{\mathcal{PS}_i})$ which is at most equal to the start time $\mathrm{ST}_{g_{i+1}}$ of job/mode combination $[g_{i+1}, m_{g_{i+1}}]$, i.e., $\mathrm{CT}^{\max}(\overline{\mathcal{PS}_i}) \leqslant \mathrm{ST}_{g_{i+1}}$, and (c) a consumption of nonrenewable resources at most equal to the consumption induced by $\mathcal{PS}_i$, then job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be skipped on level $(i+1)$.

The "serial" and the "block" structure of the examples given in [9] illustrate that this dominance concept can be successfully used when the renewable resources are scarce or ample. Note, although there is a certain similarity between the multi-mode rule (Rule 7) and the multi-mode cut-set rule I (Rule 8) one can easily build examples where the former rule can be applied and the latter one not, and vice versa (cf. [9]).

Moreover, only minor modifications are necessary for dealing with the performance measures (a)–(e), given in Section 2. That is, by additionally taking into account the partial schedule related partial sums the rule can be extended.

As already mentioned, roughly speaking, the multi-mode cut-set rule I compares the quality of the current partial schedule with the quality of a partial schedule previously evaluated. To distinguish it from the following cut-set-based rule we call it dominated heads. In contrast, the last rule we are going to present bounds the enumeration process by estimating the time necessary to complete the partial schedule under consideration. It offers a criterion for incompletable tails. We assume the resource availability of the renewable resources as constant and the minimization of the makespan as objective.

Rule 9 (Multi-mode cut-set rule II, incompletable tails): We consider a problem of type MRCPSP with constant availabilities of the renewable resources. Moreover, let $\mathcal{PS}'_k$ be any k-partial schedule. By $\mathrm{LF}_J(\mathcal{PS}'_k)$ we denote the latest finish time of activity J valid after all the continuations of $\mathscr{PS}'_k$ have been evaluated by the enumeration scheme given in Table 2. That is, the bounds on the finish times have been adjusted after finding an improved solution. Let $\mathscr{PS}_i$ be the partial schedule currently under consideration to be continued by starting job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ at $\mathrm{ST}_{g_{i+1}}$. If there is a previously evaluated partial schedule $\overline{\mathscr{PS}_i}$ with (a) $\mathscr{CS}(\overline{\mathscr{PS}_i}) = \mathscr{CS}(\mathscr{PS}_i)$, (b) modes selected in $\overline{\mathscr{PS}_i}$ induce leftover capacities at least equal to the ones of $\mathscr{PS}_i$, and (c) $\mathrm{ST}_{g_{i+1}} + \mathrm{LF}_J(\overline{\mathscr{PS}_i}) - \mathrm{CT}^{\max}(\overline{\mathscr{PS}_i}) + 1 > \mathrm{LF}_J$ then a continuation of $\mathscr{PS}_i$ which schedules $[g_{i+1}, m_{g_{i+1}}]$ on level $(i+1)$ cannot be completed to a schedule with a makespan at most equal to the currently valid latest finish time LF_J of activity J. Clearly, combining the rules has to be done with care to guarantee that optimality is not lost. Especially the combinations of the cut-set and the shift-rules need a thorough analysis. The interested reader is referred to [9] where we discuss the problems arising.

4. Computational results

In this section we present the results of our computational studies. One of the main results will be that, although the efficiency of the algorithm has been substantially increased by the proposed bounding rules, the MRCPSP is less tractable than reported in the literature. Patterson et al. (cf. [3]) have generated 91 instances. The number of jobs ranged from 10 to 500, where 75 instances have up to 30 jobs. The instances have been characterized by the mean number of modes, mean activity duration, minimum/maximum activity duration, standard deviation of the activity durations, critical path length (based on minimum activity durations), average fraction of resources used by an activity mode and network density. The procedure has been coded in Fortran and implemented on an IBM 4381 mainframe, which is, as stated, approximately seven times faster than a 386-based, 20 MHz PC with a numeric coprocessor. For an imposed time limit of 1 (10) minutes 30 (33) of the problems with up to 50 activities have been solved to optimality. The preponderance of the problem sizes ranged between 10 and 30 jobs.

We proceed as follows: In Section 4.1 we briefly summarize the parameters used to generate the problem instances by ProGen. In Section 4.2 we reveal some details of implementation. In Section 4.3 we present our computational experience with the exact method, and in Section 4.4 we report about the truncated exact method.

4.1. Project characteristics

In this section we give a brief summary of the characteristics of the project instances we have generated by ProGen. They serve as the input of the project generator. A detailed description of the parameters and their realization can be found in [18,14,19]. The instances as well as the results are available on our ftp-site ftp.bwl.uni-kiel.de under path /pub/operations-research/psplib.

We have utilized minimal and maximal values to limit the number J of nondummy activities, the number of modes M_j of nondummy activities, the duration d_j of nondummy activities, the number of renewable resources $|R|$, the number of nonrenewable resources $|N|$, the number of start activities $|\mathscr{S}_1|$, the number of finish activities $|\mathscr{P}_J|$, the number of successors $|\mathscr{S}_j|$, and the number of predecessors $|\mathscr{P}_j|$. Additionally, we have used the fixed setting of Table 4, the base setting of Table 5, and the variable settings of Table 6.

The first instance set $J10$ has been generated for the evaluation of ProGen (cf. [14]). Using the fixed parameter setting of Table 4, the base parameter setting of Table 5 with redefining $J^{\min} = J^{\max} = 10$, $NC = 1.5$, and the A Levels of the variable parameter setting of Table 6 a total of $2 \cdot 4 \cdot 2 \cdot 4 \cdot 10 = 640$ instances has been produced. 536 problems of the instances set have a feasible solution. Furthermore, by adjusting the base parameters accordingly and utilizing the B Levels of Table 6, we have built instance sets having 12, 14, 16, 18, 20 nondummy jobs, 1, 2, 3, 4, 5 modes per nondummy job, 1, 2, 3, 4, 5 renewable resources, or 1, 2, 3 nonrenewable resources. With 10 replications per combination of the variable parameter setting again a total of 640 projects per instance set has been obtained. Clearly, due to

Table 4
Fixed parameter setting

P_1^R	P_2^R	P_1^N	P_2^N	ϵ_{NET}	ϵ_{RF}
0.00	1.00	0.00	1.00	0.05	0.05

mode-coupling via resource constraints not all the problems have a feasible solution (cf. [14]).

Finally, by redefining base parameters of Table 5 and utilizing the C Levels of Table 6 we have produced instance sets where only renewable resources are limited, i.e., with $|N| = 0$. The instance sets consist of 10, 12, 14, 16, 18, 20 nondummy jobs and $2 \cdot 4 \cdot 10 = 80$ instances per set.

4.2. Some details of implementation

Before we present our computational experience with the exact method in Section 4.3 and the truncated exact method in Section 4.4 we will give some general comments concerning our implementation. More specific information can be found in [9,12]. Throughout the section the point of attack is the most frequently considered makespan minimization problem. The algorithm has been coded in GNU C using ANSI-standard and run under OS/2 on a personal computer (80486dx processor, 66 MHz clockpulse, 16 MB memory). The code requires less than 100 KB and the data structures at most 8 MB. The basic version of the algorithm has been implemented and then accelerated by the use of pointer arithmetic which actually makes the procedure approximately two times faster than the counterpart employing array arithmetic.

If the availability levels of the renewable resources are constant, then computation time can be saved when searching for the lowest (renewable) resource feasible start time within Step 4 of the algorithm. Due to nondecreasing leftover capacities resource feasibility in a period $\bar{t}+1$ implies resource feasibility in periods t, $t = \bar{t}+2, \ldots, \bar{t}+d_{jm}$. That is, only the first period has to be checked for establishing (renewable) resource feasibility. Doing so some 10% of computation time reported could be saved. Moreover, with the same argument, for the determination of the resource feasible start time of an activity currently considered for scheduling, it is sufficient to study the finish times of the activities already scheduled (cf. [20] quoted in accordance with [17]). That is, it is only necessary to keep resource availabilities of the periods an activity finishes at. Although preliminary tests indicate a significant reduction of CPU-time we have ignored these perceptions due to major adaptations necessary for the complete exploitation of this implication.

The summary of the different variants implemented is given in Table 7, where (+) denotes that the related rule is employed and (–) that it is not. The basic variant V0 matches, except for the corrections mentioned in Section 3, the algorithm proposed in [2,3].

Rules 1 (input data adjustment) and 2 (input data adaptation) have been implemented as de-

Table 5
Base parameter setting

	J	M_j	d_j	$\|R\|$	U_R	Q_R	$\|N\|$	U_N	Q_N	$\mathscr{S}_1$	$\mathscr{S}_j$	$\mathscr{P}_J$	$\mathscr{P}_j$	NC
min	16	3	1	2	1	1	2	1	1	3	1	3	1	1.8
max	16	3	10	2	10	2	2	10	2	3	3	3	3	1.8

Table 6
Variable parameter settings

	A Levels				B Levels				C Levels			
RF_R	0.50	1.00			0.50	1.00			0.50	1.00		
RS_R	0.20	0.50	0.70	1.00	0.25	0.50	0.75	1.00	0.25	0.50	0.75	1.00
RF_N	0.50	1.00			0.50	1.00						
RS_N	0.20	0.50	0.70	1.00	0.25	0.50	0.75	1.00				

Table 7
Variants of the algorithm of Table 4, Part I

Rule	Variant												
	V0	V1	V2	V3	V4	V5	V6	V7	V8	V89	V99	V100	V101
B1 (data reduction)	–	+	–	–	–	–	–	–	–	–	+	–	+
B2 (data adjustment)	–	–	+	–	–	–	–	–	–	–	+	–	–
B3 (non-delayability)	–	–	–	+	–	–	–	–	–	–	+	–	+
B4 (single enumeration)	–	–	–	–	+	–	–	–	–	–	+	–	+
B5 (local left-shift)	–	–	–	–	–	+	–	–	–	–	+	–	+
B6 (global left-shift)	–	–	–	–	–	–	+	–	–	–	–	–	+
B7 (multi-mode rule)	–	–	–	–	–	–	–	+	–	–	+	–	+
B8 (cut-set rule I)	–	–	–	–	–	–	–	–	+	+	+	–	+
B9 (cut-set rule II)	–	–	–	–	–	–	–	–	–	+	–	–	–

scribed above. Rule 3 (non-delayability rule) has been realized by an internal variable counting the consecutive trials a job is tested to be assigned a start time without success. Rule 4 (single enumeration rule) is checked through a three-dimensional array as proposed in [9]. Rule 5 (local left-shift rule) has been implemented by checking only feasibility of a one-period local left-shift. Rule 6 (global and multi-mode left-shift) has been tested in several variants as suggested in [12]. The variant without mode change performed best when employing one rule at a time. However, when combined with the other rules its use has not been beneficial. Rule 8 (cut-set rule I) has been realized by storing the cut-sets and related information in binary, level dependent trees. The cut-set information of a level is stored when backtracking leads to this level. Dominated information has been overwritten or deleted. Rule 9 (cut-set rule II) has not been tested separately since the calculations required can be directly employed for the implementation of Rule 8 (cut-set rule I). We abbreviated the extension of the basic variant V0 by Rule 8 to V8 and the enhancement through Rules 8 and 9 to V89. Moreover, due to the comments on the combination of the rules and its restricted use, it has not been combined with the remaining rules. The final version employing all the rules but Rules 6 and 9 is denoted as V99. The variant V100, except for changes possible due to the exclusion of nonrenewable resources, matches with the basic variant V0. Its enhancement by the rules is denoted as V101. Thereby, Rule 6 is implemented with mode change, and tested when an activity g_{i+1} is selected on a level $(i+1)$ for the first time.

4.3. Exact methods

Table 8 displays the average computation times in seconds for the different levels of RF_R, RS_R, RF_N and RS_N employing one rule at a time. The number of feasible problems within the classes is given in the third column. For the calculation of the average CPU-times we have fixed the considered parameter and left the others free. The last three rows of the table show the overall averages, the overall standard deviations and the maximum computation times the different variants required to solve a problem. From the table, independently of a rule being employed or not, a positive correlation of the average solution times and the resource factor of the renewable and nonrenewable resources can be observed. The strongest increase of the average solution times is induced by an increasing resource factor and decreasing the strength of the nonrenewable resources, i.e. RF_N and RS_N. It has to be mentioned that employing the rules has slowed down the basic variant V0 by some hundreds of seconds only on a few instances out of the class of problems solvable within 0.1 s.

Table 9 compares the average CPU-times of the basic variant V0 with the variant V99 on the 10-job problems previously described. Moreover, 12-job problems with the parameter specification given above have been generated and solved. A substantial improvement within all the problem

Table 8
Separate effect of the rules – $J = 10$ (80486dx, 66 MHz)

		Feas.	Variant									
			V0	V1	V2	V3	V4	V5	V6	V7	V8	V89
RF_R	0.50	259	25.93	19.51	2.34	21.97	7.31	5.56	12.34	21.48	3.40	2.23
	1.00	277	35.87	24.08	3.42	29.30	11.80	14.09	21.44	27.51	3.79	2.85
RS_R	0.20	119	29.27	21.66	5.40	22.84	15.38	19.26	21.84	21.77	1.62	1.42
	0.50	139	21.04	14.95	2.09	16.83	6.17	7.37	12.78	17.76	3.41	2.44
	0.70	138	36.48	23.66	2.36	31.36	8.92	8.26	17.49	27.29	4.20	2.97
	1.00	140	37.20	27.16	2.10	31.59	8.88	6.33	16.75	31.12	4.90	3.20
RF_N	0.50	232	1.73	1.27	1.69	1.05	0.80	1.13	1.40	1.53	0.49	0.49
	1.00	304	53.45	37.60	3.82	44.62	16.37	16.71	28.98	42.20	5.98	4.12
RS_N	0.20	76	184.61	122.28	5.61	155.11	56.30	54.58	97.47	144.55	16.97	10.46
	0.50	153	13.17	12.75	4.35	10.47	4.07	5.33	8.27	11.01	3.19	2.75
	0.70	156	2.81	2.69	1.89	2.08	1.06	1.59	1.99	2.43	0.75	0.73
	1.00	151	1.10	0.40	1.11	0.61	0.63	0.86	1.00	0.88	0.24	0.25
μ_{CPU}		536	31.06	21.87	2.90	25.76	9.63	9.97	17.04	24.59	3.60	2.55
σ_{CPU}			88.73	58.53	6.11	78.71	27.13	29.34	46.94	64.28	9.52	6.25
$\max_{CPU}$			1174.94	599.91	47.69	1138.75	298.25	390.16	580.88	681.00	93.63	53.13

classes can be observed. The comparison factor of the variants V0 and V99 indicates that the enhanced algorithm solves the 10-job (12-job) problems on average 221.85 (1489.43) times faster than the original variant. The frequencies of the solution times are given in Table 10.

The purpose of the following experiment is to find out the impact of the variation of the size of the project on the solution times. Using the constant parameter settings mainly given in Table 5 we have varied the number of nondummy activities J, the number of modes per job M_j, the com-

Table 9
Common effect of the rules – $J = 10$, 12 (80486dx, 66 MHz)

	$J = 10$					$J = 12$				
		Feas.	V0	V99	Factor		Feas.	V0	V99	Factor
RF_R	0.50	259	25.93	0.09	288.11	0.50	265	412.12	0.20	2060.06
	1.00	277	35.87	0.18	199.27	1.00	282	681.68	0.52	1310.92
RS_R	0.20	119	29.27	0.27	108.40	0.25	134	553.70	0.80	692.12
	0.50	139	21.04	0.13	161.84	0.50	141	556.26	0.37	1503.40
	0.70	138	36.48	0.10	364.80	0.75	140	611.97	0.17	3599.82
	1.00	140	37.20	0.08	465.00	1.00	132	478.35	0.13	3679.61
RF_N	0.50	232	1.73	0.10	17.30	0.50	241	21.51	0.24	89.62
	1.00	304	53.45	0.17	314.41	1.00	306	968.18	0.47	2059.95
RS_N	0.20	76	184.61	0.19	971.63	0.25	72	3,773.89	0.74	5099.85
	0.50	153	13.17	0.22	59.86	0.50	158	159.04	0.59	269.55
	0.70	156	2.81	0.14	20.07	0.75	158	18.46	0.30	61.53
	1.00	151	1.10	0.03	36.66	1.00	159	10.57	0.05	211.14
μ_{CPU}		536	31.06	0.14	221.85		547	551.09	0.37	1489.43
σ_{CPU}			88.73	0.21	422.52			1939.95	0.72	2694.37
$\max_{CPU}$			1174.94	2.31	508.63			22,925.88	9.18	2497.37

Table 10
Frequencies of solution times, $J = 10$, 12 (80486dx, 66 MHz)

Var.	J	[0;0.5]	(0.5;5]	(5;20]	(20;100]	(100;500]	(500;2,000]	(2,000;10,000]	>10,000
V0	10	230	131	64	54	54	3	–	–
V99	10	509	27	–	–	–	–	–	–
V0	12	192	89	53	77	61	32	37	6
V99	12	439	106	2	–	–	–	–	–

plexity C, the number of nonrenewable resources $|N|$, and the number of renewable resources $|R|$ as given in Table 11. The underbar denotes the standard setting for studying the remaining variations. The results of the experiment are given in Table 11.

The third column shows the number of problems that have a feasible solution. Columns 4–6 give the average CPU-time, the standard deviation of the CPU-time and the maximum CPU-time in seconds. As to be expected, we observe: (1) the strongest influence on CPU-time is exerted by the variation of the number of jobs and modes, respectively. CPU-times seem to increase exponentially with both parameters. (2) Increasing the complexity C reduces the number of precedence feasible schedules and therefore the number of sequences to be examined. That is, the CPU-time required for optimally solving the problems decreases with an increasing complexity. (3) The number of renewable resources seems to influence the CPU-times linearly. On the one hand, checking feasibility is more time consuming if the number of renewable resources is increased, on the other hand, tight constraints can prevent from deeply descending the branch-and-bound tree. (4) The number of nonrenewable resources, too, is positively correlated with the CPU-times. But, though feasibility testing with respect to nonrenewable resources is less lavish than earliest start time computation with respect to renewable resources, the number of nonrenewable resources have a stronger impact. We conjecture that it is mainly reasoned by the fact that increasing the number of nonrenewable resources may increase the number of incomparable request levels, i.e. leftover capacities, related to a cut-set. That is the effect of the cut-set rule is more and more consumed by the effort spent on checking the assumptions. (5) A detailed analysis of the influence of the resource factor and the resource strength on solution times has confirmed the higher the resource factor RF_R or RF_N is, that is, the higher the average portion of the resources used (consumed) per activity/mode combination the more time on average is necessary to solve the problem. (6) Finally, except for minor deviations, a negative correlation of the average CPU-times and the availability of the resources has been established. The lower the resource strength is, the more time is necessary to solve the problem.

In the next experiment we have studied projects solely employing renewable resources, i.e. $|N| = 0$.

Table 11
Variation of project size – Variant 99 (80486dx, 66 MHz)

		Feas.	μ_{CPU}	σ_{CPU}	$\max_{\text{CPU}}$
	10	536	0.14	0.21	2.31
	12	547	0.37	0.72	9.18
	14	551	1.87	4.81	51.09
J	$\underline{16}$	550	7.40	22.59	272.97
	18	552	47.29	170.30	1853.06
	20	554	238.67	955.68	11,579.20
	1.5	551	19.97	71.36	828.10
C	$\underline{1.8}$	550	7.40	22.59	272.97
	2.1	552	3.30	8.80	78.94
	1	553	4.42	17.85	301.00
	$\underline{2}$	550	7.40	22.59	272.97
$\|R\|$	3	557	10.87	35.03	428.03
	4	552	10.28	26.23	283.69
	5	546	15.36	56.46	979.09
	1	640	0.05	0.05	0.53
	2	551	1.06	3.81	48.68
M_j	$\underline{3}$	550	7.40	22.59	272.97
	4	555	45.09	123.76	1061.37
	5	558	264.65	895.37	14,165.88
	1	637	3.54	15.18	275.90
$\|N\|$	$\underline{2}$	550	7.40	22.59	272.97
	3	600	17.27	59.34	739.47

Table 12
Computational results – Only renewable resources – $J = 10$, 12 (80486dx, 66 MHz)

RF_R	RS_R	J	Feas.	V100	V101	Factor
0.50	0.25	10	5	1.85	0.03	61.66
	0.50		10	0.25	0.02	12.50
	0.75		10	0.02	0.00	–
	1.00		10	0.00	0.01	–
1.00	0.25		10	4.41	0.08	55.12
	0.50		10	0.37	0.04	9.25
	0.75		10	0.02	0.02	1.00
	1.00		10	0.00	0.01	–
μ_{CPU}			75	0.80	0.02	40.00
σ_{CPU}				3.03	0.03	101.00
$\max_{\mathrm{CPU}}$				24.12	0.13	185.53
0.50	0.25	12	7	3.32	0.08	41.50
	0.50		10	0.52	0.02	26.00
	0.75		10	0.01	0.01	1.00
	1.00		10	0.01	0.02	0.50
1.00	0.25		10	75.55	0.20	377.75
	0.50		10	1.51	0.07	21.57
	0.75		10	8.82	0.03	294.00
	1.00		10	0.01	0.00	–
μ_{CPU}			77	11.53	0.05	230.60
σ_{CPU}				50.38	0.07	719.71
$\max_{\mathrm{CPU}}$				411.94	0.38	1084.05

Table 13
Computational results – only renewable resources – $J = 14$, 16 (80486dx, 66 MHz)

RF_R	RS_R	J	Feas.	V100	V101	Factor
0.50	0.25	14	9	112.74	0.24	469.75
	0.50		10	1.89	0.04	47.25
	0.75		10	2.76	0.03	92.00
	1.00		10	0.01	0.03	0.33
1.00	0.25		10	253.54	0.44	576.22
	0.50		10	10.39	0.08	129.87
	0.75		10	0.62	0.03	20.66
	1.00		10	0.01	0.02	0.50
μ_{CPU}			79	46.92	0.11	426.54
σ_{CPU}				232.17	0.21	1105.57
$\max_{\mathrm{CPU}}$				1882.19	1.44	1307.07
0.50	0.25	16	9	232.92	0.35	665.48
	0.50		10	175.31	0.15	1168.73
	0.75		10	0.10	0.05	2.00
	1.00		10	0.02	0.02	1.00
1.00	0.25		10	22,062.41	4.27	4166.84
	0.50		10	1171.03	2.15	544.66
	0.75		10	3.53	0.10	35.30
	1.00		10	0.04	0.06	0.66
μ_{CPU}			79	2990.14	0.90	3322.37
σ_{CPU}				12,665.44	3.28	3861.41
$\max_{\mathrm{CPU}}$				73,635.87	19.78	3722.74

We have varied the number of activities, that is, projects consisting of 10, 12, 14, 16, 18, 20 non-dummy activities have been produced. The results are displayed in Tables 12–16. The problems have been solved by the basic variant V100 and the enhanced variant V101. Again, because of the mode coupling constraints, not all the problems have a feasible solution. The number of feasible problems for the 10- and 12-job problems are given in the fourth column of Table 12. The average CPU-times required to solve the feasible problems are given in columns five and six, whereas column seven compares the average solution times of variant V100 and variant V101. The last three rows display the overall average CPU-times, the overall standard deviations and the maximum CPU-times.

We extend our considerations to Table 13, where the average CPU-times of the basic variant V100 and the accelerated variant V101 on 14- and 16-job projects are compared. The frequencies of CPU-times are given in Table 14. Again, the comparison factors give a clear picture of the substantial improvement of the basic variant on average and worst case performance. On average the basic variant is accelerated on 10- (12-, 14-, 16-) job projects by a factor of approximately 40 (230, 427, 3322). The comparison factor of maximum solution times, is about 186 (1084, 1307, 3722).

Finally, Tables 15 and 16 show the computational results for the 18- and 20-job projects. Comparing the average and worst case performance if nonrenewable resources are taken into account (Tables 8–11) or not (Tables 12–16) we can identify the strong impact of overall limitations on resource availability (i.e. nonrenewable resources). The influence is that strong that it cannot be solely reasoned by the additional operations to be performed. We conjecture, that numerous feasible partial schedules cannot be identified as incomple-

Table 14
Frequencies of solution times (s) (80486dx, 66 MHz)

Var.	J	[0;0.5]	(0.5;5]	(5;20]	(20;100]	(100;500]	(500;2000]	(2000;10,000]	>10,000
V100	10	62	10	2	1	–	–	–	–
V101	10	75	–	–	–	–	–	–	–
V100	12	49	15	5	6	2	–	–	–
V101	12	77	–	–	–	–	–	–	–
V100	14	48	11	7	9	2	2	–	–
V101	14	75	4	–	–	–	–	–	–
V100	16	45	3	4	8	6	6	2	5
V101	16	65	11	3	–	–	–	–	–

Table 15
Computational results – only renewable resources – $J = 18$, 20 (80486dx, 66 MHz)

RF_R	RS_R	J	Feas.	V101
0.50	0.25	18	10	1.63
	0.50		10	0.13
	0.75		10	0.14
	1.00		10	0.14
1.00	0.25		10	22.72
	0.50		10	1.31
	0.75		10	0.07
	1.00		10	0.09
μ_{CPU}			80	3.28
σ_{CPU}				15.49
$\max_{\mathrm{CPU}}$				110.94
0.50	0.25	20	9	1.57
	0.50		10	1.11
	0.75		10	0.17
	1.00		10	0.34
1.00	0.25		10	84.94
	0.50		10	2.41
	0.75		10	0.32
	1.00		10	0.51
μ_{CPU}			79	11.42
σ_{CPU}				65.74
$\max_{\mathrm{CPU}}$				576.22

table with respect to nonrenewable resources until further continuations have been evaluated. That is, fast completability checks may cause a substantial improvement of the performance when taking into account nonrenewable resources.

4.4. Truncated exact methods

In the final experiment we have examined the heuristic capabilities of the algorithm. For this purpose we have built the frequency distributions of times necessary for optimally solving the problems of Section 4.3 by variant V99. The results are displayed in Table 17. For the majority of the problems an optimal solution can be found and verified within 20 s. Beside the frequency distributions of solution time the quality of the solutions that is found after spending a certain amount of time on the enumeration process is of interest. Therefore, we have let the enumeration process run 10 s and 1 min, respectively.

The truncated exact approach is again based on variant V99. The results are given in Table 18. Hereby, considering one of the four blocks, the first row displays the number of feasible problems; "Solved" indicates the number of problems a feasible solution of which has been found; "$\emptyset$" gives the number of problems no feasible solution could be determined for, although there is one; "Φ^*" represents the number of problems the optimal solution can be found and verified; "Best" reflects the number of problems the solution determined is optimal. Finally, the last two rows show the average deviation $\overline{\Delta}$ and maximum deviation from optimality $\Delta^{\max}$ of the solutions found. The results show a reasonable improvement of the quality of the solutions by spending more time on the enumeration. Moreover, the number of problems a feasible solution has been found of is rather high.

Table 16
Frequencies of solution times (s) (80486dx, 66 MHz)

Var.	J	[0;0.5]	(0.5;5]	(5;20]	(20;100]	(100;500]	(500;2000]	(2000;10,000]	>10,000
V101	18	58	16	4	1	1	–	–	–
V101	20	45	22	9	2	1	1	–	–

Table 17
Frequencies of solution times by Variant V99 (s) (80486dx, 66 MHz)

	[0;0.5]	(0.5;5]	(5;20]	(20;100]	(100;500]	(500;2,000]	(2000;10,000]	>10,000
J								
10	509	27	–	–	–	–	–	–
12	439	106	2	–	–	–	–	–
14	318	188	43	7	–	–	–	–
$\underline{16}$	254	169	78	41	8	–	–	–
18	197	156	72	77	39	11	–	–
20	166	126	60	78	69	37	16	2
M_j								
1	639	1	–	–	–	–	–	–
2	394	138	14	5	–	–	–	–
$\underline{3}$	254	169	78	41	8	–	–	–
4	213	131	57	98	43	13	–	–
5	188	112	43	61	84	53	16	1
C								
1.5	228	151	90	53	26	3	–	–
$\underline{1.8}$	254	169	78	41	8	–	–	–
2.1	301	166	65	20	–	–	–	–
$\|R\|$								
1	294	158	74	25	2	–	–	–
$\underline{2}$	254	169	78	41	8	–	–	–
3	212	185	97	52	11	–	–	–
4	198	186	95	63	10	–	–	–
5	200	177	94	52	22	1	–	–
$\|N\|$								
1	331	224	63	17	2	–	–	–
$\underline{2}$	254	169	78	41	8	–	–	–
3	218	194	102	62	21	3	–	–

Usually, heuristic approaches fail finding a (good) feasible solution if the consumption of nonrenewable resources is limited.

5. Conclusions

We have discussed an enumeration scheme for solving the MRCPSP. The basic scheme can be easily generalized for additionally handling constraints induced by time-varying requests as well as minimal and maximal time-lags. Moreover, any objective considered can be dealt with a slightly modified version.

The basic enumeration scheme has been extended by powerful search tree reduction schemes which are mainly valid when optimizing any regular measure of performance. The results of our computational experiments indicate a substantial improvement of computational tractability of the MRCPSP. The size of the projects that can be solved to optimality has been nearly doubled.

Table 18
Truncated exact approach by Variant V99 – 10 s and 1 m (80486dx, 66 MHz)

	$J = 10$	$J = 12$	$J = 14$		$J = 16$		$J = 18$		$J = 20$	
	10 sec	10 sec	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min
Feas.	536	547	551	551	550	550	552	552	554	554
Solved	536	547	551	551	550	550	550	552	550	552
$\emptyset$	0	0	0	0	0	0	2	0	4	2
Φ^*	536	547	524	551	467	535	384	481	327	405
Best	536	547	535	551	495	546	417	508	354	437
$\bar{\Delta}$	0%	0%	0.23%	0%	0.82%	0.03%	2.87%	0.73%	6.05%	2.38%
$\Delta^{\max}$	0%	0%	20.83%	0%	24.32%	5.56%	58.06%	29.73%	78.95%	51.61%

	$M_j = 1$		$M_j = 2$		$\underline{M_j = 3}$		$M_j = 4$		$M_j = 5$	
	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min
Feas.	640	640	551	551	550	550	555	555	558	558
Solved	640	640	551	551	550	550	543	551	533	545
$\emptyset$	0	0	0	0	0	0	12	4	25	13
Φ^*	640	640	539	551	467	535	362	458	322	381
Best	640	640	547	551	495	546	402	493	347	415
$\bar{\Delta}$	0%	0%	0.04%	0%	0.82%	0.03%	3.53%	1.00%	6.24%	2.81%
$\Delta^{\max}$	0%	0%	9.52%	0%	24.32%	5.56%	41.03%	35.90%	57.89%	35.71%

	$\lvert R\rvert = 1$		$\underline{\lvert R\rvert = 2}$		$\lvert R\rvert = 3$		$\lvert R\rvert = 4$		$\lvert R\rvert = 5$	
	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min
Feas.	553	553	550	550	557	557	552	552	546	546
Solved	553	553	550	550	555	557	549	552	545	546
$\emptyset$	0	0	0	0	2	0	3	0	1	0
Φ^*	492	547	467	535	449	532	434	528	423	512
Best	527	551	495	546	486	540	468	540	461	526
$\bar{\Delta}$	0.36%	0.01%	0.82%	0.03%	1.10%	0.24%	1.40%	0.15%	1.57%	0.24%
$\Delta^{\max}$	22.73%	3.57%	24.32%	5.56%	37.50%	31.03%	31.58%	17.39%	51.52%	18.18%

			$\underline{C = 1.8}$							
	$\lvert N\rvert = 1$		$\underline{\lvert N\rvert = 2}$		$\lvert N\rvert = 3$		$C = 1.5$		$C = 2.1$	
	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min	10 sec	1 min
Feas.	637	637	550	550	600	600	551	551	552	552
Solved	637	637	550	550	590	599	550	551	552	552
$\emptyset$	0	0	0	0	10	1	1	0	0	0
Φ^*	585	633	467	535	464	558	430	513	503	547
Best	607	635	495	546	508	572	471	531	527	552
$\bar{\Delta}$	0.439%	0.03%	0.82%	0.03%	1.56%	0.38%	1.35%	0.23%	0.31%	0%
$\Delta^{\max}$	19.44%	13.89%	24.32%	5.56%	36.84%	37.50%	33.33%	16.67%	19.23%	0%

Using the standard project generator ProGen we have established a wide range of more than 10,000 problem instances that can serve as a basis for the experimental investigation and evaluation of new approaches. The impact of the variation of problem characteristics on the solution time and quality has been studied. The results of the computational experiment reflect the intuitive expectation and thus confirm the reliability of ProGen.

However, although the rules presented can mainly be implemented in other multi-mode suitable generalizations (cf. e.g. [6]) of single-mode algorithms (cf. [16,17]) the approach presented here remains the most general one currently available. In contrast to the previously mentioned it covers time-varying resource availabilities and can simply be generalized to cover time-varying resources requests as well. Although we are far away from solving problems sized by reality the results give a clear indication what we have to expect when trying to solve large projects to optimality. However, truncated exact approaches seem to be an appropriate tool for the heuristic solution of real problems (of large size).

Acknowledgements

We are grateful to the anonymous referees for extremely helpful comments and suggestions.

References

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

[2] J.H. Patterson, R. Slowinski, F.B. Talbot, J. Weglarz, An algorithm for a general class of precedence and resource constrained scheduling problems, in: R. Slowinski, J. Weglarz (Eds.), Advances in Project Scheduling, Elsevier, Amsterdam, 1989, pp. 3–28.

[3] J.H. Patterson, R. Slowinski, F.B. Talbot, J. Weglarz, Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems, European Journal of Operational Research 49 (1990) 68–79.

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

[5] A. Sprecher, Resource-constrained project scheduling: exact methods for the multi-mode case, Lecture Notes in Economics and Mathematical Systems, no. 409, 1994, Springer, Berlin.

[6] A. Sprecher, S. Hartmann, A. Drexl, An exact algorithm for project scheduling with multiple modes, OR Spektrum vol. 19 (1997) 195–203.

[7] F.B. Talbot, Resource-constrained project scheduling with time-resource tradeoffs: the nonpreemptive case, Management Science 28 (1982) 1197–1210.

[8] A. Drexl, J. Grünewald, Nonpreemptive multi-mode resource-constrained project scheduling, IIE Transactions 25 (5) (1993) 74–81.

[9] A. Sprecher, 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, no. 385, 1996a, Kiel.

[10] F.J. Radermacher, Scheduling of project networks, Annals of Operations Research 4 (1985,1986) 227–252.

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

[12] A. Sprecher, 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, no. 386, 1996, Kiel.

[13] A. Drexl, Scheduling of project networks by job assignment, Management Science 37 (1991) 1590–1602.

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

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

[16] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, Management Science 38 (1992) 1803–1818.

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

[18] R. Kolisch, A. Sprecher, A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems – Easy and hard instances, Manuskripte aus den Instituten für Betriebswirtschaftslehre, no. 301, 1992, Kiel.

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

[20] T.J.R. Johnson, An algorithm for the resource-constrained project scheduling problem, Ph.D. Dissertation, Massachusetts Institute of Technology, USA, 1967.