Sprecher, Arno; Drexl, Andreas

Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequeacing Algorithm. Part II: Computation

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 386

No. 386

Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part II: Computation

Arno Sprecher and Andreas Drexl

January 1996

Arno Sprecher, Andreas Drexl
Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel,
Olshausenstr. 40, 24098 Kiel, Germany
Tel.&Fax ++49(0)431-880-1531, Email: Drexl@bwl.uni-kiel.de.

Abstract

This is the second part of a paper considering the multi-mode resource-constrained project scheduling problem. The first part presents an exact solution procedure for solving several variants of the multi-mode resource-constrained project scheduling problem including the commonly known makespan minimization problem. The basic scheme is enhanced by powerful search tree reduction schemes considerably useful when optimizing any regular measure of performance.

The second part reports the results of a thorough computational study and the experience gained with the algorithm's realization.

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 ten thousand problem instances have been systematically generated to evaluate the algorithm's performance. The experimental investigations demonstrate a superior performance of the exact method and reasonable capabilities of the truncated exact approach. The size of the projects that can be solved to optimality has been nearly doubled.

The main objectives of this part are: First, to give additional information concerning implementation of the algorithm and the bounding rules. Second, to illustrate the effect of the bounding rules. Third, to show the impact of the variation of several project characteristics on solution time and quality. Fourth, to enlarge the set of project instances available for testing multi-mode procedures.

Keywords: Project Scheduling, Resource Constraints, Multiple Modes, Branch-and-Bound, Heuristic, Instance Generation, Computational Results.

1 Introduction

This paper is a continuation of Sprecher and Drexl (1996) (cf. [18]). There we have presented, analyzed and extended a precedence tree guided enumeration scheme for solving the multi-mode resource-constrained project scheduling problem. Starting with the general description of the problem and its mathematical programming formulation we have worked out the precedence tree guided enumeration scheme and reduced it to its key mechanism. Doing so several quite general search tree reduction schemes have been developed and illustrated by an example. Limitations and adaptations for dealing with slightly changed assumptions and objectives have been discussed.

Several benefits the approach bears make it outstanding: (1) Ease of description, (2) ease of implementation, (3) ease of generalization, and as it will turn out as a result of our study, (4) superior performance of the exact method and (5) reasonable heuristic capabilities of the truncated method.

Although the problem is known as NP-hard the strength of the influence of the variation of project characteristics on solution times is still unknown. We will close this gap. The main objectives of

this part are: (1) To give additional information concerning implementation of the algorithm and the bounding rules. (2) To illustrate the separate and common effect of the bounding rules. (3) To show the impact of the variation of several project characteristics on solution times. (4) To enlarge the set of project instances available for testing multi-mode approaches.

We proceed as follows: Section 2 gives a brief summary of the standard project generator ProGen. Section 3 reports about the computational experience we gained with the exact approach. The seperate 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 exact approach are presented. Section 4 draws the conclusions.

2 Generation of Instances

In this section we describe an algorithm for generating problem instances of the precedence- and resource-constrained project scheduling problem presented in the first part. It will provide us the instances we need for properly evaluating the sequencing algorithm proposed.

Certainly, a solution procedure's cababilities (efficiency) can be preliminarily tested on a set of hand-made problem instances, but the deficiencies associated with that kind of specific problem collection are thorouhly studied in the literature (cf. e.g. [6]).

On the other hand, inspite of the lack of realism, using the systematic generation scheme has multiple benefits: (1) Since most of the combinatorial problems under consideration are only known as NP-hard, a worst case solution time estimation can be seldomly reported. Using a systematic scheme the estimates can be improved. (2) Analyzing the performance of an algorithm on parameter characterized problem settings may bring out information for data specific solution strategy development (as e.g. in [7], [16]). The constraints may by classified into groups of simplifying and impeding constraints, thus giving rise to study relaxations and bound calculations. (3) A systematic examination of the computational time and quality of solutions can turn out benefits for particular problem classes and disadvantages for others.

Therefore, we summarize the project generator ProGen developed by Kolisch et al. (cf. [8]) which serves as the basis of our computational studies presented in Section 3.

The algorithm consists of four steps. Namely, Step 1, the base data generation, Step 2, the network construction, Step 3, the resource demand generation, and finally, Step 4, the resource availability generation.

Within the generation process we use the functions round$(\cdot)$ and trunc$(\cdot)$ where the former rounds the

argument to an integer and the latter truncates the decimal fraction of the argument.

Furthermore, the random functions $\text{rand}(n_1, n_2)$ and $\overline{\text{rand}}(n_1, n_2)$ indicate randomly drawing of an integer and a real, respectively, out of the integer bounded interval $[n_1, n_2]$. The (pseudo) random numbers itselves are obtained by transforming $[0, 1)$ uniformly distributed random numbers. The $[0, 1)$ uniformly distributed random numbers are calculated through the congruence-generator developed by Lehmer using the constants and implementation as given by Schrage (cf. [14]).

The generation of the project's base data, that is, the generation of the number of non-dummy activities $\overline{J}$, the number of renewable resources $|R|$, the number of nonrenewable resources $|N|$, the number of modes $|M_j|$, and the activity durations d_{jm} is straightforward as displayed in Table 1. That is, the quantaties are determined by randomly drawn integers out of the related, user specified intervals.

$\overline{J}$	:=	$\text{rand}[J^{min}, J^{max}]$	(number of non-dummy activities)
M_j	:=	$\text{rand}[M^{min}, M^{max}]$	(number of modes of activity j)
d_{jm}	:=	$\text{rand}[d^{min}, d^{max}]$	(duration of activity j in mode m)
$\|R\|$	:=	$\text{rand}[R^{min}, R^{max}]$	(number of renewable resources)
$\|N\|$	:=	$\text{rand}[N^{min}, N^{max}]$	(number of norenewable resources)

Table 1: Base Data Generation

The activities are labelled consecutively from 1 to J, $J := \overline{J} + 2$, where activity 1 and J is dummy source and dummy sink activity, respectively. Both activities have a unique mode with a zero duration and do not request any resource. The modes of the remaining non-dummy activities are labelled with respect to non-decreasing durations.

If the per-period availability of the renewable resources is constant and the problem is feasible with respect to the nonrenewable resources, an upper bound on the optimal makespan $\overline{T}$ can be obtained by adding up maximal durations of the activities, i.e. $\overline{T} = \sum_{j=2}^{J-1} \max_{m=1}^{M_j} \{d_{jm}\}$.

A lower bound on the project's makespan is given by the MPM-duration obtained by using the modes of shortest duration and taking into account the project network the construction of which is presented in the following subsection.

2.1 Network Generation

The underlying idea of the network construction relies on a simple implication of the definition of a network. By definition, we know, that for every node v out of the vertex set V, $V = \{1, \ldots, J\}$, and

arc set A, there is a (directed) path from the single source, node 1, to v and a directed path from v to the single sink, node J. That is, every node except of the sink (source) has at least one successor (predecessor). Therefore, the network construction is performed in three steps: First, each node is assigned a predecessor, second, each node is assigned a successor and, finally, further arcs are added. Clearly, for characterizational purpose, the network finally generated should not contain arcs giving no additional information about scheduling the activities. That is, an arc considered for introducing to the current subgraph should not be redundant or cause a redundant precedence relation.

By a more formal consideration, we can state that an arc (i,j) of a network G is redundant, if it is an element of the transitive closure $\overline{G}^+$ of $\overline{G} = (V, A\backslash\{(i,j)\})$. Doing so, four cases of redundancy produced by an arc currently considered for introducing to the current subgraph can be identified (cf. [8]). As a result, arcs inducing redundant precedence relations are excluded from inclusion.

However, for a given cardinality of the set of nodes the minimal and maximal number of non-redundant arcs can be determined as follows (cf. [8]): Let $G = (V, A)$ be a network with $|V| = n$. (a) Since a network is connected, the minimal number of non-redundant arcs A^{min} is given by $A^{min} = n - 1$. (b) The maximal number of non-redundant arcs A^{max} in a network with $n \geq 6$ is given by $A^{max} = n - 2 + \left(\frac{n-2}{2}\right)^2$ if n is even, and $A^{max} = n - 2 + \left(\frac{n-1}{2}\right)\left(\frac{n-3}{2}\right)$ if n is odd.

$\mathcal{S}_1^{min}$ ($\mathcal{S}_1^{max}$)	:	minimal (maximal) number of start activities
$\mathcal{P}_J^{min}$ ($\mathcal{P}_J^{max}$)	:	minimal (maximal) number of finish activities
$\mathcal{S}_j^{max}$ ($\mathcal{P}_j^{max}$)	:	maximal number of successor (predecessor) activities of activity j, $j = 2, \ldots, J-1$
C	:	network complexity, i.e. the average number of non-redundant arcs per node (including the super-source and -sink)
ϵ_{NET}	:	tolerated complexity deviation

Table 2: Input Network Generation

For the characterization of the network we use the parameters given in Table 2. The complexity as the average number of (non-redundant) arcs per node has been introduced by Pascoe (cf. [10]) for activity-on-arc networks and adopted by Davis (cf. [3]) for the AON-representation. Thereby, an increasing complexity reflects for a fixed number of jobs, a stronger interconnectedness of the network. It has already been shown by Alvarez-Valdes and Tamarit (cf. [1], and confirmed by Kolisch et al. [8]) that

problems become easier with an increasing complexity. This makes the term complexity somewhat confounding. Nevertheless, since the term has been used in a number of computational studies (cf. [5], [9], [11], [20], and [21]) and, therefore, has become a wellknown project summary measure we retain it.

We now briefly summarize the usage of the parameters given above within the four steps of the construction process. In Step 1 the number of start and finish activities are drawn randomly out of the interval $[\mathcal{S}_1^{min}, \mathcal{S}_1^{max}]$ and $[\mathcal{P}_J^{min}, \mathcal{P}_J^{max}]$, respectively. Arcs which connect the dummy source with the start activities and the finish activities with the dummy sink are then incorporated into the network. In Step 2, beginning with the lowest indexed non-start activity, each activity is assigned a predecessor activity at random. Similar in Step 3, each activity which has no successor is assigned one. Finally, in Step 4, further arcs are added until the complexity is reached. During the whole procedure it is ensured that (1) introducing an arc does not cause redundant precedence relation, (2) the limitation on the number of predecessor and successor activities are met. Moreover, by an appropriate reduction of the selectable predecessors and successors a numerically labelled network is generated.

At the end of the generation process the number of arcs ActAcrcs of the network fulfills $(1 - \epsilon_{NET}) \cdot J \cdot C \leq \text{ActArcs} \leq (1 + \epsilon_{NET}) \cdot J \cdot C$.

2.2 Resource Demand and Availability Generation

We consider a resource type τ, $\tau \in \{R, N\}$.

2.2.1 Resource Demand Generation

The parameters characterizing the demand generation are displayed in Table 3. Their use will be summarized in the following.

The resource demand generation requires two decisions to be made. First, the resources used or consumed by the job-mode combinations $[j, m]$ have to be determined. Second, for a job-mode combination using or consuming a resource, the number of units to be used or consumed have to be fixed. The first step is referred to as *request generation* and the latter as *generation of demand level.*

(a) Requested Resources

The requested resources are characterized by the resource factor (RF), which measures the density of the array k_{jmr}. It has been introduced for the single-mode case by Pascoe (cf. [10]) and utilized in

$Q_\tau^{min}(Q_\tau^{max})$	:	minimal (maximal) number of resources of type τ used by a job-mode combination $[j,m]$
$U_\tau^{min}(U_\tau^{max})$	:	minimal (maximal) demand for a resource of type τ
$P_\tau(F=1)(P_\tau(F=2))$	:	probability that demand for a resource of type τ is duration constant (F=1) (monotonically decreasing with the duration (F=2))
RF_τ	:	resource factor of type τ
RS_τ	:	resource strength of type τ
ϵ_{RF}	:	tolerated resource factor deviation

Table 3: Input Demand Generation

studies by Cooper (cf. [2]) and Alvarez-Valdes and Tamarit (cf. [1]). Kolisch et al. (cf. [8]) generalized it to the multi-mode case:

$$RF_\tau := \frac{1}{J-2}\frac{1}{|\tau|}\sum_{j=2}^{J-1}\frac{1}{M_j}\sum_{m=1}^{M_j}\sum_{r\in\tau}\begin{cases}1 & , \text{ if } k_{jmr}>0\\ 0 & , \text{ otherwise}\end{cases}$$

Roughly speaking, the resource factor reflects the average portion of resources requested per job/mode combination. It is normalized to the interval $[0,1]$, where, obviously, a resource factor $RF = 0$, expresses the unconstrained MPM-case, and $RF = 1$ indicates that each job/mode combination uses and consumes all the resources, respectively.

Since, by definition, the resource factor is a discrete measure for a specific base data assingnment not all the parameters within the interval $[0,1]$ can be met exactly. Therefore the actual resource factor ARF_τ is controlled by a deviation constant ϵ_{RF} guaranteeing the realized resource factor is out of the interval $[RF_\tau \cdot (1-\epsilon_{RF}), RF_\tau \cdot (1+\epsilon_{RF})]$.

Beside the resource factors the minimal and maximal number of requested resources Q_τ^{min} and Q_τ^{max} are taken into account when generating the resource requests. These parameters ensure that a minimal and maximal number of resources of type τ requested per job/mode combination is established.

(b) Level of Demand

In the previous subsection we have discussed the resource request generation, that is, the decision concerning the resources which are requested by the job/mode combinations. The problem now arising is to determine level of demand. That is, for the renewable resources the per-period demand and for the nonrenewable resources the total demand has to be generated. The interrelation between the durations of the modes and the demand for a resource r is reflected by two types of functions.

One of which is duration independent ($F = 1$) and the other one is decreasing with the (increasing) duration ($F = 2$). For each resource $r \in \tau$ the interrelation is defined in accordance with the type dependent probabilities $P_\tau(F = 1)$ and $P_\tau(F = 2)$, that is, $F_\tau(r) := 1$ if $\overline{\text{rand}}[0,1] < P_\tau(F = 1)$ and $F_\tau(r) := 2$ otherwise.

If $F_\tau(r) = 1$, then for each job a demand U' can be randomly drawn out of the integer interval $[U_\tau^{min}, U_\tau^{max}]$ and assigned to all the modes requesting this resource. If $F_\tau(r) = 2$, then for each job j two levels U^1 and U^2 are drawn randomly out of the parameter specified interval, i.e. $U^1 := \text{rand}[U_\tau^{min}, U_\tau^{max}]$ and $U^2 := \text{rand}[U_\tau^{min}, U_\tau^{max}]$. Subsequently U^{low} and U^{high} is defined as the minimum and maximum of both levels, respectively.

Given $\overline{M_j}$, the number of modes of job j with different durations requesting resource r, we calculate

$$\Delta := \frac{U^{high} - U^{low}}{\overline{M_j}}$$

and yield $\overline{M_j}$ intervals I_k as follows:

$$I_k := [round(U^{high} - \Delta k), round(U^{high} - \Delta(k-1))] \qquad k = 1, \ldots, \overline{M_j}.$$

Since the modes are labelled with respect to nondecreasing durations, we can now draw the demand randomly out of the intervals corresponding to the durations. If two different modes with same duration request a resource then their level of demand is drawn out of the same interval.

During the construction process inefficiency of the modes (cf. [8] and Part I, Section 4) is controlled and excluded. That is, the case of two different modes m and $\overline{m}$ with $d_{jm} \leq d_{j\overline{m}}$ and $k^\rho_{jmr} \leq k^\rho_{j\overline{m}r}$ for all $r \in R$ and $k^\nu_{jmr} \leq k^\nu_{j\overline{m}r}$ for all $r \in N$ cannot occur.

2.2.2 Resource Availability Generation

In order to analyze the relationship between the resource demand of the jobs and the resource availability Cooper (cf. [2]) introduced the resource strength (RS). Unfortunately, it is non-normalized and does not take into account the underlying network structure. Thus, simple and difficult problems are assigned identical characterizations of availability. To overcome these problems Kolisch et al. (cf. [8]) proposed a new definition of the resource strength.

Using a minimal and maximal demand level K_r^{min} and K_r^{max} the actual availability is expressed by a convex combination $K_r := K_r^{min} + RS_\tau(K_r^{max} - K_r^{min})$ with normalized scaling parameter RS_τ. $RS_\tau = 0$ represents the lowest resource availability that can make the problem feasible with respect to one resource, and $RS_\tau = 1$ results in the unconstrained MPM-case.

More precisely, for nonrenewable resources r, $r \in N$, the minimal and maximal availabilities to complete the project are calculated as follows:

$$K_r^{min} := \sum_{j=2}^{J-1} \min_{m=1}^{M_j} \{k_{jmr}^{\nu}\} \quad , \quad K_r^{max} := \sum_{j=2}^{J-1} \max_{m=1}^{M_j} \{k_{jmr}^{\nu}\}$$

For a given type dependent resource strength $RS_\tau \in [0,1]$ the availability is then obtained by

$$K_r^{\nu} := K_r^{min} + \text{round}(RS_\tau \, (K_r^{max} - K_r^{min})).$$

If the considered resource is renewable the minimal demand is

$$K_r^{min} := \max_{j=2}^{J-1} \left\{ \min_{m=1}^{M_j} \{k_{jmr}^{\rho}\} \right\}.$$

The maximal demand is calculated as the peak period demand of the precedence preserving earliest start schedule when performing each job in the lowest indexed mode that employs maximal per-period demand with respect to the resource under consideration. That is, the maximal per-period demand of job j with respect to resource r is given by

$$k_{jr}^{*} := \max_{m=1}^{M_j} \{k_{jmr}^{\rho}\}$$

and the corressponding mode with shortest duration:

$$m_{jr}^{*} := \min_{m=1}^{M_j} \{m | k_{jmr}^{\rho} = k_{jr}^{*}\}$$

Given the precedence relations of the project the earliest start schedule with the modes determined above can be derived. We obtain the resource dependent start time ST_j^{τ} and completion time CT_j^{τ} of job j, $j = 2, \ldots, J-1$. The peak period demand is then defined by

$$K_r^{max} := \max_{t=1}^{\overline{T}} \left\{ \sum_{\substack{j=2 \\ ST_j^{\tau}+1 \le t \le CT_j^{\tau}}}^{J-1} k_{jm_{jr}^{*}r} \right\}$$

and, thus, using the type dependent resource strength RS_τ the available amount is given by

$$K_r^{\rho} := K_r^{min} + \text{round}(RS_\tau (K_r^{max} - K_r^{min})).$$

By construction we can state the following:

Remark 1

(a) If $|\tau| = 1$ and $RS_\tau = 0$, then the lowest resource feasible level with respect to τ will be generated.

(b) IF $RS_\tau << 1$ and $M_j > 1$, then feasibility of the problem can not be assured, because of mode coupling via resource constraints.

3 Computational Results

In this section we present the results of our computational studies. One of the main results will be that, although the efficieny of the algorithm has been substantially increased by the proposed bounding rules, the multi-mode resource-constrained project scheduling problem is less tractable than reported in the literature. Patterson et al. (cf. [13]) 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 has been 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 ten and thirty jobs.

3.1 Some Details of Implementation

Before we start to present our computational experience with the exact method in Subsection 3.2 and the truncated exact method in Subsection 3.3 we will air some of the secrets and problems concerning the implementation of the algorithm. We hope others will follow. Especially, hints on the implementation of the bounding rules are given, to ease the reimplementation of the algorithm.

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 Sep 4 and Step 4' of the algorithm given in Tables 4 and 5 of Part I, respectively. Due to nondecreasing leftover capacities $K_r^{\varrho}(\mathcal{PS}_i)$, $t = ST_{g_i}+1,\ldots,\overline{T}$, $r \in R$, resource feasibility in a period $\bar{t}+1$ implies resource feasibility in periods $t, \bar{t}+2,\ldots,\bar{t}+d_{jm}$. That is, only the first period has to be checked for establishing (renewable) resource feasibilty. Doing so some ten percent of computation time reported can be saved. Moreover,

with the same argument, the start time of an activity currently considered for scheduling is determined by the minimum precedence and resource feasible start time one of the activities already scheduled finishes at. 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. As already mentioned in the theoretical part the way of the bounding rules' implementation is critical to success. For getting deeper insight we briefly report our experience..

Bounding Rule 1 (input data reduction) and **Bounding Rule 2** (input data adjustment), the static preprocessing rules, can be straightforwardly implemented. However, if they are commonly realized, it is useful to apply Bounding Rule 1 first, and then Bounding Rule 2 for maximizing the common effect. Removing modes non-executable with respect to renewable resources, via Bounding Rule 1, may increase the subtractable amount of minimal consumptions in Bounding Rule 2.

Bounding Rule 3 (non-delayability) which induces backtracking if an activity g_{i+1} cannot be successfully scheduled, that is, without violating the constraints, is realized by an internal variable. It counts the number of unsuccessful consecutive trials of scheduling that specific job on a certain level. Backtracking occurs if the number of consecutive trials is equal to the number of modes. Therefore, it has to be reinitialized even if a job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be successfully scheduled, but the truncation of the related branch is enforced by another bounding rule (cf. e.g. with the discussion of local left-shift rule, Bounding Rule 5).

Bounding Rule 4 (single enumeration), which checks if from a current partial schedule the same completions are obtainable as from partial schedules considered earlier, can be straightforwardly implemented by the use of Theorem 6, Part I. However, the internal variable mentioned in the discussion of Bounding Rule 3 has to be reinitialized when a branch is truncated in accordance with Bounding Rule 4.

Bounding Rule 5 (local left-shift), that excludes partial schedules from further extension if the currently considered job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be locally left-shifted, involves two cases to be distinguished. First, from the currently assigned start time $ST_{g_{i+1}}$ determined with respect to Theorem 1 (a) and (b), Part I, (only) a one-period left-shift is feasible. Second, a complete local left-shift can be performed, that is, activity/mode combination $[g_{i+1}, m_{g_{i+1}}]$ can be started at $ST_{g_i} - d_{g_{i+1}m_{g_{i+1}}}$ without violating the constraints. Whereas in the former case the internal variable mentioned in the discussion of Bounding Rule 3 has to be reinitialized it does not have to be in the latter. That is, in first case an (the) optimal solution might be found by extending the current partial schedule by scheduling $[g_{i+1}, m_{g_{i+1}}]$ on a level higher than the current one. The second case means,

that even if an optimal schedule can be found by extending the current partial schedule through scheduling $[g_{i+1}, m_{g_{i+1}}]$ on a level higher than the current there is a corresponding schedule where $[g_{i+1}, m_{g_{i+1}}]$ is finished at or before ST_{g_i}. Note, both implementations will ensure that only semi-active schedules are generated (cf. [19]). Since the second case will be realized through the global and multi-mode left-shift rule we have implemented the first case.

Bounding Rule 6 (global and multi-mode left shift), which examines if a global or multi-mode left-shift can be performed on the activity considered for scheduling can be realized in several ways. First, we distinguish between projects with *unlimited* overall budgets, i.e. $|N| = 0$, and projects with *limited* overall budgets, i.e. $|N| > 0$.

In the *former case* we have implemented two variants. *First*, if, for a given i-partial schedule $\mathcal{PS}_i$, an activity g_{i+1} is considered for scheduling for the first time on level $(i+1)$, then we have checked all the modes $m_{g_{i+1}}$, $m_{g_{i+1}} = 1, \ldots, M_{g_{i+1}}$, if they can be scheduled in the interval determined by the precedence feasible start time $ST^{prec}_{g_{i+1}}(\mathcal{PS}_i)$ of activity g_{i+1} and the start time ST_{g_i} currently assigned to activity g_i in $\mathcal{PS}_i$. If any of the modes can be worked off in the specified interval, then backtracking is performed. *Second*, when stepping forward from a level i to a level $(i+1)$, we examine all the activity/mode combinations $[g, m]$, $g \in Y_{i+1}$, $m = 1, \ldots, M_g$. If any of the combinations can be processed in the interval $[\max\{ST^{prec}_g(\mathcal{PS}_i), ST_{g_{i-1}} - d_{gm} + 1\}, ST_{g_i}]$ then we track back. Clearly, it is rational to search for a feasible left-shift only if $ST_{g_{i-1}} < ST_{g_i}$. The benefit of the second implementation is twofold: First, a possible dominance is detected on a lower level than by the first variant, and, second, the length of the intervals to be considered are reduced. Inspite of the theoretical benefits the computational experience makes us favour the first variant.

In the *latter case* we have tested numerous variants, unfortunately, none of the implementations could improve the performance of the algorithm employing all the bounding rules together. We will describe only six implementations. The *first* one implements Theorem 8 without mode change, that is, if an activity/mode combination $[g_{i+1}, m_{g_{i+1}}]$ is considered for scheduling on level $(i+1)$ we scan the interval $[ST^{prec}_{g_{i+1}}(\mathcal{PS}_i), ST_{g_i}]$ for a sub-intervall where $[g_{i+1}, m_{g_{i+1}}]$ can be additionally scheduled in. If we are successful we select the next activity/mode combination. The *second* one extends the first to all the modes that dominate the mode currently under consideration w.r.t. consumption of nonrenewable resources. That is, if there is a mode m, $1 \leq m \leq M_{g_{i+1}}$, with $k^{\nu}_{g_{i+1}mr} \leq k^{\nu}_{g_{i+1}m_{g_{i+1}}r}$, $r \in N$, that can be processed in $[ST^{prec}_{g_{i+1}}(\mathcal{PS}_i), ST_{g_i}]$, then $[g_{i+1}, m_{g_{i+1}}]$ can be skipped. Feasibility of a left-shift is considered consecutively for all the modes if the acitivity g_{i+1} is considered for scheduling on level $(i+1)$ with respect to i-partial schedule $\mathcal{PS}_i$ for the first time. The next activity/mode is selected if a dominating mode can be (globally) left-shifted. The *third* and *fourth* variant enhance

the first and second one by a blocking mechanism, that is, an activity/mode combination that can be globally or multi-mode left-shifted to finish at or before ST_{g_i} with respect to i-partial schedule $\mathcal{PS}_i$ can be left-shifted exactly the same way when considering an $(i+k)$-partial schedule $\mathcal{PS}_{i+k} = \mathcal{PS}_i \oplus [h_1, m_{h_1}] \oplus \cdots \oplus [h_k, m_{h_k}]$. The activity/mode combination can be blocked. In the third variant, an activity/mode combination blocked on level $(i+1)$ is skipped on levels $i+k$, $k \geq 1$. In the fourth variant, an activity/mode combination a dominating mode of which is blocked on level $(i+1)$ is skipped on levels $i+k$, $k \geq 1$. The *fifth* variant makes use of the blocking mechanism and, additionally, reduces the periods to be examined. Using a priority rule with attributes given in Remark 4, Part I, two cases are distinguished: If $N_i \leq N_{i+1}$ then it seeks to schedule the unblocked activity/mode combination $[g_{i+1}, m_{g_{i+1}}]$ in the interval $[ST^{prec}_{g_{i+1}}(\mathcal{PS}_i), ST_{g_i}]$. Otherwise the interval $[\max\{ST^{prec}_{g_{i+1}}(\mathcal{PS}_i), ST_{g_{i-1}} - d_{g_{i+1}m_{g_{i+1}}} + 1\}, ST_{g_i}]$ is scanned. If the trial is successful then the job/mode combination is blocked for scheduling on a level $i+k$, $k \geq 1$. The *sixth* variant examines all the job/mode combinations $[g_{i+1}, m]$, $m = 1, \ldots, M_{g_{i+1}}$, when g_{i+1} is tried to be scheduled for the first time on level $i+1$ with i-partial schedule $\mathcal{PS}_i$. The intervals are determined like in variant 5. A job/mode combination $[g_{i+1}, m_{g_{i+1}}]$ is skipped if a dominating combination is blocked on level $(i+1)$ or a lower one.

However, independently in whatever way of the ones described above, the global and multi-mode left-shift rule has been implemented, the schedules finally obtained need not be tight (cf. [15]).

Note, when implementing the global (and multi-mode) left-shift rule as decribed above the local left-shift rule is partly included. However, as already mentioned, the former rule would truncate a larger part of the branch-and-bound tree.

Bounding Rule 7 (multi-mode rule) can be straightforwardly implemented. On each level of the branch-and-bound tree we store for the activity g_{i+1} currently under consideration the completion times obtained when using the modes m, $m = 1, \ldots, m_{g_{i+1}}$. However, one has to take into acount the comments on combining the local (global) left-shift, the multi-mode and the cut-set rule. That is, the completion time of an activity/mode combination has to be set to infinity, if one of the shift rules is applied successfully in order to preserve optimality.

Bounding Rule 8 (multi-mode cut-set rule I) can be realized in several variants as well. Clearly, it can be generalized to the following remark:

Remark 2

Let $\mathcal{PS}_i$ and $\overline{\mathcal{PS}_j}$ be an i-partial and a j-partial schedule, respectively, that have been detected on different paths of the branch-and-bound tree. Let g_{i+1} be an activity neither scheduled in $\mathcal{PS}_i$ nor

in $\overline{\mathcal{PS}}_j$. *Furthermore, let* $ST_{g_{i+1}}$ *denote the start time of activity/mode combination* $[g_{i+1}, m_{g_{i+1}}]$ *in* $\mathcal{PS}_{i+1} = \mathcal{PS}_i \oplus [g_{i+1}, m_{g_{i+1}}]$ *and* $\overline{ST}_{\overline{g}_j}$ *the start time of activity/mode combination* $[\overline{g}_j, \overline{m}_{\overline{g}_j}]$ *in* $\overline{\mathcal{PS}}_j$. *If*

(a) $\mathcal{CS}(\mathcal{PS}_i) \subseteq \mathcal{CS}(\overline{\mathcal{PS}}_j)$ (b) $K_r^\nu(\mathcal{PS}_i) \leq K_r^\nu(\overline{\mathcal{PS}}_j), r \in N,$

(c) $\overline{ST}_{\overline{g}_j} \leq ST_{g_{i+1}},$ (d) $K_{rt}^\rho(\mathcal{PS}_i) \leq K_{rt}^\rho(\overline{\mathcal{PS}}_j), r \in R, t = ST_{g_{i+1}} + 1, \ldots, \overline{T},$

(e) $ST_{g_{i+1}}^{prec}(\overline{\mathcal{PS}}_j) \leq ST_{g_{i+1}}$

then, by the use of the scheduling strategy given in Theorem 1, Part I, $\mathcal{PS}_i \oplus [g_{i+1}, m_{g_{i+1}}]$ *is dominated by* $\overline{\mathcal{PS}}_j \oplus [g_{i+1}, m_{g_{i+1}}]$.

Proof: Obvious □

However, obviously, in contrast to the single-mode case (cf. e.g. [4]), in the multi-mode case condition (d) of Remark 2 cannot be deduced from

(d1) $\overline{CT}_g \leq CT_g,$ if $g \in \mathcal{CS}(\mathcal{PS}_i)$ with $\overline{CT}_g > ST_{g_{i+1}}$

(d2) $\overline{CT}_g \leq ST_{g_{i+1}},$ otherwise.

That is, in order to prevent from excessive use of storage and expensively proving the assumptions of Remark 2 we strengthened them to the ones given in Theorem 10, Part I. Doing so simple data structures are sufficient for efficiently checking for dominance. First, by demanding identity in condition (a) of Remark 2, i.e. $i = j$ and $\mathcal{CS}(\mathcal{PS}_i) = \mathcal{CS}(\overline{\mathcal{PS}}_i)$, we can use a binary tree for storing the binary coded (integer-valued) cut-sets obtained from already evaluated partial schedules. Doing so fast binary search techniques can be employed. Second, by using $CT^{max}(\overline{\mathcal{PS}}_i) \leq ST_{g_{i+1}}$ we can deduce condition (c), (d) and (e) of Remark 2 with low memory requirements and computational effort.
Note, the fact that the proofs of Theorems 4, 5, 7, 8, 9, and 10 mainly rely on Remark 2 does not necessarily mean that the partial schedules which are excluded from further continuation in accordance with the theorems would be excluded by Remark 2 as well. This is due to the fact that, roughly speaking, on the one hand Remark 2 makes use of the history, that is the set of already evaluated partial schedules, and on the other hand the theorems make use of the history and the future of the enumeration process.
As already mentioned a cut-set $\mathcal{CS}(\mathcal{PS}_i)$, the related maximum completion time $CT^{max}(\mathcal{PS}_i)$ and the related left-over capacities $K_r^\nu(\mathcal{PS}_i)$ are stored when tracking back from level $(i+1)$ to level i. Thereby, it pays to store the information only if on level $(i+1)$ an activity/mode combintion has been found having a start time at least equal to $CT^{max}(\mathcal{PS}_i)$.

For keeping the effort of verification of cut-set dominance low we focussed on dominating cut-set information. That is, considering i-partial schedules $\mathcal{PS}_i$ and $\overline{\mathcal{PS}_i}$ with same cut-sets, the cut-set information of $\mathcal{PS}_i$ dominates the one of $\overline{\mathcal{PS}_i}$, if $K_r^{\nu}(\mathcal{PS}_i) \geq K_r^{\nu}(\overline{\mathcal{PS}_i})$, $r \in N$, and $CT^{max}(\mathcal{PS}_i) \leq CT^{max}(\overline{\mathcal{PS}_i})$. The newly obtained cut-set information is only stored if it is not dominated be previously stored cut-set information. Additionally, previously stored informations are eleminated if they are dominated by the new one.

3.2 Exact Methods

The summary of the different variants implemented is given in Table 4, where (+) denotes that the related bounding rule is employed and (−) that it is not. The basic variant V0 matches, except for the corrections mentioned in Section 3, Part I, the algorithm proposed in [12] and [13]. In Bounding Rule 5 (local left-shift rule) we have only checked feasibility of a one-period local left-shift. From the different variants of Bounding Rule 6 (global and multi-mode left-shift) the first one performed best when employing one rule at a time. The extension of the basic variant V0 by the first variant of Bounding Rule 6 is denoted as V61. However, when combined with the other bounding rules its use has not been beneficial. Bounding Rule 9 (cut-set rule II) has not been tested seperately since the calculations required can be directly employed for the implementation of Bounding Rule 8 (cut-set rule I). We abbreviated the extension of the basic variant V0 by Bounding Rule 8 to V8 and the enhancement through Bounding Rule 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 bounding rules but Bounding Rule 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 bounding rules is denoted as V101. Thereby, Bounding Rule 6 is implemented in its first variant suitable when considering only renewable resources.

In order to evaluate the effect of the bounding rules presented in Part I, Section 4, we have made use of the standard project generator ProGen.

The first set of instances comes from the evaluation of ProGen. Beside the constant parameter specification given in Table 5 we have chosen monotonically decreasing functions, i.e. $P_R(F = 2) = 1$ ($P_N(F = 2) = 1$) to define a negative correlation of the levels of usage (consumption) and the activity duration. Using the network and availability tolerances ϵ_{NET} and ϵ_{AVL} of 0.05 we allowed at most 200 trials to meet the requirements.

The variable settings, as the resource factor of the renewable and nonrenewable resources RF_R and RF_N and the resource strength of the renewable and nonrenewable resources RS_R and RS_N are given

Bounding Rule	Variant												
	V0	V1	V2	V3	V4	V5	V61	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)	–	–	–	–	–	–	–	–	–	+	–	–	–

Table 4: Variants of the Algorithm of Table 4, Part I

	J	M_j	d_j	$\|R\|$	U_R	Q_R	$\|N\|$	U_N	Q_N	S_1	S_j	$\mathcal{P}_J$	$\mathcal{P}_j$
min	10	3	1	2	1	1	2	1	1	3	1	3	1
max	10	3	10	2	10	2	2	10	2	3	3	3	3

Table 5: Constant Parameter Levels under Full Factorial Design

in Table 6. For each combination of the variable levels ten instances have been generated. As outlined in [8] feasibility of all the problems cannot be guaranteed. Only 536 of the 640 problems have a feasible solution.

Parameter	Levels			
RF_R	0.50	1.00		
RS_R	0.20	0.50	0.70	1.00
RF_N	0.50	1.00		
RS_N	0.20	0.50	0.70	1.00

Table 6: Variable Parameter Levels under Full Factorial Design

Table 7 displays the average computation times in seconds for the different levels of RF_R, RS_R, RF_N and RS_N employing one bounding 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

			Variant									
		Feas.	V0	V1	V2	V3	V4	V5	V61	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
$\max_{CPU}$			1,174.94	599.91	47.69	1,138.75	298.25	390.16	580.88	681.00	93.63	53.13

Table 7: Separate Effect of the Bounding Rules – J = 10 (80486dx, 66MHz)

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 seconds.

Table 8 compares the average CPU-times of the basic variant V0 with the variant V99, employing all the bounding rules but the global and multi-mode left-shift, on the set of problems previously described (J=10). Moreover, 12-job problems with the parameter specification given above have been generated and solved. A substantial improvement within all the problem 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 (1,489.43) times faster than the original variant. The frequencies of the solution times are given in Table 9.

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 non-dummy activities J, the number of modes per job M_j, the complexity C, the number of nonrenewable resources $|N|$, and the number of renewable resources $|R|$ as given in

		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	2,060.06
	1.00	277	35.87	0.18	199.27	1.00	282	681.68	0.52	1,310.92
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	1,503.40
	0.70	138	36.48	0.10	364.80	0.75	140	611.97	0.17	3,599.82
	1.00	140	37.20	0.08	465.00	1.00	132	478.35	0.13	3,679.61
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	2,059.95
RS_N	0.20	76	184.61	0.19	971.63	0.25	72	3,773.89	0.74	5,099.85
	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	1,489.43
σ_{CPU}			88.73	0.21	422.52			1,939.95	0.72	2,694.37
$\max_{CPU}$			1,174.94	2.31	508.63			22,925.88	9.18	2,497.37

Table 8: Common Effect of the Bounding Rules – J= 10, 12 (80486dx, 66MHz)

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	–	–	–	–	–

Table 9: Frequencies of Solution Times, J= 10, 12 (80486dx, 66MHz)

Table 10. The underbar denotes the standard setting for studying the remaining variations. For each parameter combination we have varied the resource factor RF and the resource strength RS, i.e. $RF_N, RF_R \in \{0.50, 1.00\}$ and $RS_N, RS_R \in \{0.25, 0.50, 0.75, 1.00\}$, and generated ten instances per combination. That is 640 instances per combination of the size parameters have been produced. Clearly, minor changes of the constant parameter setting have to be made, that is, Q_τ^{min}, Q_τ^{max}, $\tau \in \{R, N\}$, have been adapted. The results of the experiment are given in Table 10.

The third and the nineth column show the number of problems that have a feasible solution. Columns four through six and ten through twelve give the average CPU-time, the standard deviation of the CPU-time and the maximum CPU-time in seconds. As to be expected, (1) the strongest influence on

		Feas.	μ_{CPU}	σ_{CPU}	max_{CPU}			Feas.	μ_{CPU}	σ_{CPU}	max_{CPU}
J	10	536	0.14	0.21	2.31	M_j	1	640	0.05	0.05	0.53
	12	547	0.37	0.72	9.18		2	551	1.06	3.81	48.68
	14	551	1.87	4.81	51.09		3	550	7.40	22.59	272.97
	16	550	7.40	22.59	272.97		4	555	45.09	123.76	1,061.37
	18	552	47.29	170.30	1,853.06		5	558	264.65	895.37	14,165.88
	20	554	238.67	955.68	11,579.20						
C	1.5	551	19.97	71.36	828.10	$\|N\|$	1	637	3.54	15.18	275.90
	1.8	550	7.40	22.59	272.97		2	550	7.40	22.59	272.97
	2.1	552	3.30	8.80	78.94		3	600	17.27	59.34	739.47
$\|R\|$	1	553	4.42	17.85	301.00						
	2	550	7.40	22.59	272.97						
	3	557	10.87	35.03	428.03						
	4	552	10.28	26.23	283.69						
	5	546	15.36	56.46	979.09						

Table 10: Variation of Project Size - Variant 99 (80486dx, 66MHz)

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 complexitiy. (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. left-over capacities, related to a cut-set. That is the effect of the cut-set rule is more and more consumed by the effort spended on checking the assumptions. Finally, a detailed analysis of the influence of the resource factor and the resource strength on solution times has confirmed, (5) the higher the resource factor RF_R or RF_N is, that is, the higher the average portion of the resources used (consumed) per acivity/mode combination the more time on average is necessary to solve the problem. (6) Except for minor deviations, a neagtive

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. $|N| = 0$. Using the fixed parameter setting of Table 5 with minor adaptations due to $|N| = 0$, that is, $|N|^{min} = |N|^{max} = 0$, $Q_N^{min} = Q_N^{max} = 0$ and $U_N^{min} = U_N^{max} = 0$, ten instances per combination of the variable setting $RF_R = 0.50, 1.00$ and $RS_R = 0.25, 0.50, 0.75, 1.00$ have been generated. Moreover, 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 11 through 15. The problems have been solved by the basic variant V100 and the bounding rule 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 ten- and twelve-job problems are given in the fourth and nineth column of Table 11. The average CPU-times required to solve the feasible ten-job problems are given in column five and six, whereas column seven compares the average solution times of variant V100 and variant V101. Similarly, columns eight through twelve show the results concerning the twelve-job problems. 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 12, where the average CPU-times of the

RF_R	RS_R	J	Feas.	V100	V101	Factor	J	Feas.	V100	V101	Factor
0.50	0.25	10	5	1.85	0.03	61.66	12	7	3.32	0.08	41.50
	0.50		10	0.25	0.02	12.50		10	0.52	0.02	26.00
	0.75		10	0.02	0.00	—		10	0.01	0.01	1.00
	1.00		10	0.00	0.01	—		10	0.01	0.02	0.50
1.00	0.25		10	4.41	0.08	55.12		10	75.55	0.20	377.75
	0.50		10	0.37	0.04	9.25		10	1.51	0.07	21.57
	0.75		10	0.02	0.02	1.00		10	8.82	0.03	294.00
	1.00		10	0.00	0.01	—		10	0.01	0.00	—
μ_{CPU}			75	0.80	0.02	40.00		77	11.53	0.05	230.60
σ_{CPU}				3.03	0.03	101.00			50.38	0.07	719.71
$\max_{CPU}$				24.12	0.13	185.53			411.94	0.38	1,084.05

Table 11: Computational Results - Only Renewable Resources – J =10, 12 (80486dx, 66MHz)

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 13. 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, 3,322). The worst-case acceleration, i.e. the comparison factors of maximum solution times, is about 186 (1,084, 1,307, 3,722).

RF_R	RS_R	J	Feas.	V100	V101	Factor	J	Feas.	V100	V101	Factor
0.50	0.25	14	9	112.74	0.24	469.75	16	9	232.92	0.35	665.48
	0.50		10	1.89	0.04	47.25		10	175.31	0.15	1,168.73
	0.75		10	2.76	0.03	92.00		10	0.10	0.05	2.00
	1.00		10	0.01	0.03	0.33		10	0.02	0.02	1.00
1.00	0.25		10	253.54	0.44	576.22		10	22,062.41	4.27	4,166.84
	0.50		10	10.39	0.08	129.87		10	1,171.03	2.15	544.66
	0.75		10	0.62	0.03	20.66		10	3.53	0.10	35.30
	1.00		10	0.01	0.02	0.50		10	0.04	0.06	0.66
μ_{CPU}			79	46.92	0.11	426.54		79	2,990.14	0.90	3,322.37
σ_{CPU}				232.17	0.21	1,105.57			12,665.44	3.28	3,861.41
$\max_{CPU}$				1,882.19	1.44	1,307.07			73,635.87	19.78	3,722.74

Table 12: Computational Results - Only Renewable Resources – J =14, 16 (80486dx, 66MHz)

Var.	J	[0;0.5]	(0.5;5]	(5;20]	(20;100]	(100;500]	(500;2,000]	(2,000;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 13: Frequencies of Solution Times [*sec.*] (80486dx, 66MHz)

Finally, Tables 14 and 15 show the computional results for the projects consisting of eighteen or twenty non-dummy activities. Comparing the average and worst case performance if nonrenewable resources are taken into account (Table 6 through Table 10) or not (Table 11 through Table 15) 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 incompletable 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.

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

Table 14: Computational Results - Only Renewable Resources – J =18, 20 (80486dx, 66MHz)

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

Table 15: Frequencies of Solution Times [*sec.*] (80486dx, 66MHz)

3.3 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 the former subsection by variant V99. The results are displayed in Table 16. For the majority of the problems an optimal solution can be found and verified within twenty seconds.

Beside the frequency distributions of solution time the quality of the solutions that are found after spending a certain amount of time on the enumeration process is of interest. Therefore, we have let the enumeration process run ten seconds and one minute, respectively. The truncated exact approach is again based on variant V99. The results are given in Table 17. 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 problems no feasible solution could

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

Table 16: Frequencies of Solution Times by Variant V99 [*sec.*] (80486dx, 66MHz)

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

	$J = 10$	$J = 12$	$J = 14$		$J = 16$		$J = 18$		$J = 20$	
	10 sec.	10 sec.	10 sec.	1min.	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
$\overline{\Delta}$	0%	0%	0.23%	0%	0.82%	0.03%	2.87%	0.73%	6.05%	2.38%
Δ^{max}	0%	0%	20.83%	0%	24.32%	5.56%	58.06%	29.73%	78.95%	51.61%

	$M_j = 1$		$M_j = 2$		$M_j = 3$		$M_j = 4$		$M_j = 5$	
	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
$\overline{\Delta}$	0%	0%	0.04%	0%	0.82%	0.03%	3.53%	1.00%	6.24%	2.81%
Δ^{max}	0%	0%	9.52%	0%	24.32%	5.56%	41.03%	35.90%	57.89%	35.71%

	$\|R\| = 1$		$\|R\| = 2$		$\|R\| = 3$		$\|R\| = 4$		$\|R\| = 5$	
	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
$\overline{\Delta}$	0.36%	0.01%	0.82%	0.03%	1.10%	0.24%	1.40%	0.15%	1.57%	0.24%
Δ^{max}	22.73%	3.57%	24.32%	5.56%	37.50%	31.03%	31.58%	17.39%	51.52%	18.18%

	$\|N\| = 1$		$C = 1.8$ $\|N\| = 2$		$\|N\| = 3$		$C = 1.5$		$C = 2.1$	
	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
$\overline{\Delta}$	0.39%	0.03%	0.82%	0.03%	1.56%	0.38%	1.35%	0.23%	0.31%	0%
Δ^{max}	19.44%	13.89%	24.32%	5.56%	36.84%	37.50%	33.33%	16.67%	19.23%	0%

Table 17: Truncated Exact Approach by Variant V99 – 10 Seconds and 1 Minute (80486dx, 66MHz)

4 Conclusions

We have studied the computational effect of the acceleration schemes presented in the first part of the paper. The results indicate a substantial improvement of computational tractability of the multi-mode resource-constrained project scheduling problem. The size of the projects that can be solved to optimality has been nearly doubled. However, although the rules presented can mainly be implemented in other multi-mode suitable generalizations (cf. e.g. [17]) of single-mode algorithms (cf. [4], [20]) 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.

By the use of the standard project generator ProGen we have established a wide range of more than ten thousand problem instances that can serve as a basis for the experimental investigation and evaluation of new approaches. The results of the computational experiment reflect the intuitive expectation and thus confirm the quality of ProGen. 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).

References

[1] Alvarez-Valdes, R. and J.M. Tamarit (1989): Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. In: Slowinski, R. and J. Weglarz (Eds.): Advances in project scheduling. Elsevier, Amsterdam, pp. 113-134.

[2] Cooper, D.F. (1976): Heuristics for scheduling resource-constrained projects: An experimental investigation. Management Science, Vol. 22, pp. 1186-1194.

[3] Davis, E.W. (1975): Project network summary measures constrained-resource scheduling. AIIE Transactions, Vol. 7, pp. 132-142.

[4] 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.

[5] Drexl, A. and J. Grünewald (1993): Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions, Vol. 25, No. 5, pp. 74-81.

[6] JACKSON, H.F.; P.T. BOGGS; S.G. NASH AND S. POWELL (1991): Guidelines for reporting results of computational experiments. Mathematical Programming, Vol. 49, pp. 413-425.

[7] KOLISCH, R. (1995): Project scheduling under resource constraints: Efficient heuristics for several problem classes. Physica-Verlag, Heidelberg.

[8] KOLISCH, R.; A. SPRECHER AND A. DREXL (1995): Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, Vol. 41, No. 11.

[9] KÜRTULUS, I.S. AND S.C. NARULA (1985): Multi-project scheduling: Analysis of project performance. IIE Transactions, Vol. 17, pp. 58-66.

[10] PASCOE, T.L. (1966): Allocation of resources C.P.M. Revue Francaise Recherche Operationelle, No. 38, pp. 31-38.

[11] PATTERSON, J.H. (1976): Project scheduling: The effects of problem structure on heuristic performance. Naval Research Logistics Quarterly, Vol. 23, pp. 95-123.

[12] 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.

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

[14] SCHRAGE, L. (1979): A more portable Fortran random number generator. ACM Transactions on Mathematical Software, Vol. 5, pp. 132-138.

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

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

[17] SPRECHER, A.; S. HARTMANN AND A. DREXL (1994): Project scheduling with discrete time-resource and resource-resource tradeoffs. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 357, Kiel.

[18] SPRECHER, A. AND A. DREXL (1996): Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part I: Theory.

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

[20] STINSON, J.P. (1976): A branch and bound algorithm for a general class of multiple resource-constrained scheduling problems. PhD Dissertation, Graduate School of Business Administration, University of North Carolina, USA.

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