Hartmann, Sönke; Sprecher, Arno

A note on "hierarchical models for multi-project planning and scheduling"

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 338

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Nr. 338

A NOTE ON "HIERARCHICAL MODELS FOR MULTI-PROJECT PLANNING AND SCHEDULING"

Sönke Hartmann / Arno Sprecher

December 1993

önke Hartmann, Arno Sprecher, Institut für Betriebswirtschaftslehre,
hristian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, F. R. G.

Abstract: We consider the multi-mode resource-constrained project scheduling problem. The focus is on an algorithm which is supposed to find a makespan optimal solution. This algorithm has been presented in a recent paper by Speranza and Vercellis. The correctness of the algorithm is examined. Moreover, two counterexamples in which the algorithm does not lead to an existing optimal solution are presented.

Keywords: Project scheduling, resource constraints, multiple modes, tight schedules.

1 Introduction

In a recent paper, Speranza and Vercellis [1] suggest a branch-and-bound algorithm for solving the multi-mode resource-constrained project scheduling problem with makespan minimization as objective. Speranza and Vercellis present a new idea for restricting the search space by defining *tight* schedules: According to their proposition, an algorithm that enumerates all tight schedules will find an optimal solution. Since non-tight schedules are excluded from the search space, such an algorithm is supposed to be very efficient. The authors suggest an algorithm which uses *maximal extensions* of partial schedules to produce tight schedules.
Unfortunately, the algorithm does *not* enumerate all tight schedules. We will show by counterexamples that the algorithm does not always find an optimal solution for a given problem. In addition, in some cases it does not find an existing feasible schedule.

The remainder of the paper is organized as follows: After the description of the model in Section 2, we present a formally revised formulation of the algorithm in Section 3. Section 4 provides two examples in which the algorithm does not lead to an optimal solution. Finally, conclusions are drawn in Section 5.

2 The Model

The multi-mode resource-constrained project scheduling problem (MRCPSP) can be stated as follows: We consider a single project which consists of nonpreemptive activities given by the set V. The activities are partially ordered by precedence relations, where P_j is the set of the immediate predecessors of activity j, $j \in V$. The precedence relations can be represented by an acyclic activity-on-node network.
We distinguish two different types of resources: The set of renewable resources is referred to as R while N denotes the set of nonrenewable resources. For each renewable resource r, $r \in R$, the availability in

period t is given by W_{rt}, $t = 1,...,T$, where T denotes an upper bound on the projects makespan. For each nonrenewable resource r, $r \in N$, the overall capacity for the whole project is given by Q_r.

Each activity can be performed in one of several modes of accomplishment, where different modes use different resources and/or have different durations. M_j denotes the set of modes of activity j. For each activity-mode combination (j,m), $j \in V$, $m \in M_j$, the duration d_{jm} is given. Furthermore, w_{jmrt} denotes the usage of renewable resource r, $r \in R$, in each period t, $t = 1,...,d_{jm}$. The consumption of a nonrenewable resource r, $r \in N$, is given by q_{jmr}.

Let $n := |V|$ be the number of activities. Activity $1 \in V$ $(n \in V)$ is the unique dummy source (sink). Both activities are assumed to have only one mode each, that is, $M_1 = M_n = \{1\}$, in which they have a duration of zero periods and do not request any resources.

A summary of the symbols of the model can be found in Table 1. We assume the parameters and the data to be integer valued. The objective is to minimize the projects makespan.

V	:	set of activities in the project
M_j	:	set of modes of activity j
P_j	:	set of immediate predecessors of activity j
d_{jm}	:	(non preemptable) duration of activity j performed in mode m
R (N)	:	set of renewable (nonrenewable) resources
w_{jmrt}	:	usage of renewable resource r required to perform activity j in mode m in the t-th period the activity is in progress
q_{jmr}	:	total consumption of nonrenewable resource r required to perform activity j in mode m
W_{rt}	:	availability of renewable resource r in period t
Q_r	:	total availability of nonrenewable resource r

Table 1: Symbols and Definitions

3 Solution Methodology

In this section we briefly summarize the basic elements used in the algorithm presented in [1].

Definition 1. A *schedule* is a set

$$S = \{ (j, T_j, m_j) \mid j \in V; \text{ there exist exactly one } T_j \in \mathbf{N}_0 \text{ and one } m_j \in M_j \},$$

where each activity j, $j \in V$, is assigned exactly one triplet (j, T_j, m_j) which denotes that activity j is assigned the start time $T_j \in \mathbf{N}_0$ and the mode $m_j \in M_j$.

A schedule is called *feasible* if the precedence and resource constraints are not violated.

A *partial schedule PS* is a subset of a schedule S.

Definition 2. A schedule $S = \{ (j, T_j, m_j) \mid j \in V \}$ is called *tight* if there does not exist an activity j, a mode $m'_j \in M_j$ and a start time $T'_j \in \mathbf{N}_0$, such that

$$S' := S \setminus \{ (j, T_j, m_j) \} \cup \{ (j, T'_j, m'_j) \}$$

is a feasible schedule with $T'_j + d_{jm'_j} < T_j + d_{jm_j}$.

Thus, a schedule S is tight if there does not exist an activity $j \in V$ the finish time of which can be reduced without violating the constraints while the start times and modes of all other activities remain unchanged.

Proposition 1. (Cf. [1], Proposition 4.1.) If there exists an optimal schedule for a given MRCPSP, then there exists an optimal tight schedule.

Proof. Let S be an optimal schedule for a specific instance. If S is tight, we are done. Otherwise, there exists an activity j and a triplet $(j, T_j, m_j) \in S$, for which we can find $T'_j \in \mathbf{N}_0$ and $m'_j \in M_j$ with $T_j \neq T'_j$ and/or $m_j \neq m'_j$, such that $T'_j + d_{jm'_j} < T_j + d_{jm_j}$ holds and

$$S' := S \setminus \{ (j, T_j, m_j) \} \cup \{ (j, T'_j, m'_j) \}$$

is a feasible schedule. Since S is optimal, we have $j \neq |V|$. Thus, S' is of the same length as S, and S' is an optimal schedule, too. Iteratively applying this substitution leads to a tight schedule of the same length as the original schedule S. Therefore, the derived schedule is optimal and tight. ■

According to Proposition 1, an algorithm that enumerates all tight schedules of a given instance will find an optimal solution. The algorithm suggested in [1] is supposed to enumerate all tight schedules (cf. [1], Proposition 4.2) and therefore should be more efficient than an algorithm that additionally examines schedules which are not tight.

The algorithm computes all possibilities to add a maximal set of activities, start times and modes to the current partial schedule without violating the constraints. We need some more definitions to see how the partial schedules are extended in the algorithm:

Definition 3. Let PS be a partial schedule, and let τ be a time instant. An unscheduled activity j, $j \in J$, is called *free at time instant* τ if all predecessors of j are scheduled in PS and completed at or before time τ and, moreover, there exists at least one mode $m_j \in M_j$ in which activity j can be started at or before time τ without violating the constraints.

Definition 4. Let PS_1 and PS_2 be partial schedules. Let A_1 and A_2 denote the sets of those activities that are scheduled in PS_1 and PS_2, repectively. We define the *extension operator for partial schedules* (+) by

$$PS_1 + PS_2 := PS_1 \setminus \{ (j, T_j, m_j) \in PS_1 \mid j \in A_1 \cap A_2 \} \cup PS_2 .$$

This definition ensures that in an extended partial schedule no activity is assigned two different start times or two different mode numbers. Note, in general, we have $PS_1 + PS_2 \neq PS_2 + PS_1$.

Definition 5. Let PS be a partial schedule, let τ be a time instant, let A be the set of the activities scheduled in PS, and let B be the set of those activities which are free at time τ. Furthermore, let F denote the set of those scheduled activities which are finished at time τ, that is,

$$F = \{ j \in J \mid \text{there exists } (j, T_j, m_j) \in PS \text{ with } T_j + d_{jm_j} \leq \tau \}.$$

Let P denote the set of those scheduled activities that are in progress at time τ, that is,

$$P = A \setminus F.$$

For a given subset $I \subseteq B \cup P$ of activities and a related assignment

$$L = \{ (j, T_j, m_j) \mid j \in I,\ T_j \in \mathrm{N}_0 \text{ and } m_j \in M_j \}$$

let $PS' := PS + L$.

The assignment L is called *dominating*, if PS' is a feasible partial schedule, and if there does not exist an activity j, $j \in I$, a mode number $m'_j \in M_j$ and a start time $T'_j \in \mathrm{N}_0$, such that

$$PS'' := PS' \setminus \{ (j, T_j, m_j) \} \cup \{ (j, T'_j, m'_j) \}$$

is a feasible partial schedule with $T'_j + d_{jm'_j} < T_j + d_{jm_j}$.

A *maximal extension* is a dominating assignment for which the addition of any activity j, $j \in B \cup P \setminus I$, to the new partial schedule PS' would cause a resource conflict.

Remark 1. In [1], it is not explicitly mentioned how to determine the start times T_j of the activities $j \in I$ in the above definition of dominating assignments. Note, if T_j is allowed to be greater than the current time instant τ, *each* activity $j \in B \cup P$ can be scheduled in the dominating assignment without causing a resource conflict w. r. t. renewable resources. Therefore, in this case, any maximal extension will contain the activities of $I = B \cup P$. We will return to this question in Section 4.

The algorithm described in [1] is a depth first search branch-and-bound algorithm. First, the earliest start time est_j for each activity j is calculated by traditional forward recursion. The earliest start time $e_{jm}(1)$ for each activity j and each mode m is initialized with est_j. At level $g = 1$ of the branch-and-bound tree, the dummy source activity is scheduled at time $\tau_1 = 0$.

After computing the set $\mathcal{H}_1$ of all maximal extensions, one maximal extension $\overline{H}$ is chosen for branching to the next level $g = 2$ of the tree. Then $\overline{H}$ is removed from the set $\mathcal{H}_1$ and then added to the current partial schedule.

New earliest start times are computed as follows: If activity j was either free or in progress at the previous level but not in the current maximal extension (case (a)), $e_{jm}(2)$ is defined as the maximum between $e_{jm}(1)$ and the earliest time instant at which activity j could start in mode m. If activity j was neither scheduled nor free at level $g = 1$ (case (b)), $e_{jm}(2)$ is determined as the maximum between $e_{jm}(1)$ and time τ_1.

Furthermore, a new time instant τ_2 is determined as the minimum between the earliest finish time of the activities in progress and the earliest possible start time of any unscheduled activity the predecessors of which are finished at time τ_1.

If the dummy sink activity has not been scheduled yet, the next set of maximal extensions of the current partial schedule is computed, and the algorithm proceeds as previously described.

If the dummy sink activity has been scheduled, the current schedule is saved and backtracking to the previous level occurs. Now another maximal extension from the corresponding set at that level is chosen, and the algorithm branches to the next level. However, if the set of maximal extensions is empty, another backtracking step is made.

If level $g = 0$ is reached, the algorithm stops.

A summary of the parameters and variables used in the algorithm can be found in Table 2 while Table 3 provides a presentation of the algorithm including minor changes due to typing errors.

j	:	activity number
m	:	mode number
T_j	:	start time of activity j
est_j	:	earliest start time of activity j
g	:	level of the branch-and-bound tree
$e_{jm}(g)$	:	earliest start time of activity j in mode m at level g
τ_g	:	time instant at level g
PS_g	:	partial schedule at level g
A_g	:	set of activities scheduled in the partial schedule PS_g
P_g	:	set of activities scheduled in PS_g that are still in progress at time τ_g
F_g	:	set of activities scheduled in PS_g that are finished at time τ_g
B_g	:	set of those unscheduled activities that can be started in at least one mode at or before time τ_g (set of free activities)
$\overline{H}$	:	maximal extension of the current partial schedule
H	:	set of the activities contained in the corresponding maximal extension $\overline{H}$
$\mathcal{H}_g$	:	set of the maximal extensions at level g
K_g	:	set of unscheduled activities the predecessors of which are finished at time τ_{g-1}
Best	:	current best makespan
S	:	current best schedule

Table 2: Parameters and Variables used in the Algorithm

Step 1: **(Initialisation)**

$g := 1$; $PS_g := \{(1,0,1)\}$; $A_g := \{1\}$; $\tau_g := 0$; $F_g := \{1\}$; $P_g := \varnothing$; $Best := 9999$;

for every $j \in V$ compute the earliest start time est_j;

for every $j \in V$ and $m \in M_j$ let $e_{jm}(g) := est_j$.

Step 2: **(Maximal extensions)**

If $F_g = V$ then go to Step 6, otherwise compute the set of the free activities B_g and the set of all maximal extensions $\mathcal{H}_g$.

Step 3: **(Next maximal extension)**

If $\mathcal{H}_g = \varnothing$ then go to Step 5,

otherwise select a maximal extension $\overline{H}$ from $\mathcal{H}_g$ and remove it from $\mathcal{H}_g$.

Step 4: **(Branching)**

$g := g + 1$;

$A_g := A_{g-1} \cup H$;

$PS_g := PS_{g-1} + \overline{H}$;

$P_g := P_{g-1} \cup H$;

for each activity-mode combination (j,m), $j \in B_{g-1} \cup P_{g-1} \setminus H$, $m \in M_j$, let (a)

$e_{jm}(g) := \max \{ e_{jm}(g\text{-}1),\ \min \{ T_j \mid PS_{g-1} \cup \{ (j, T_j, m) \} \text{ is feasible} \} \}$;

for each activity-mode combination (j,m), $j \in V \setminus A_{g-1} \setminus B_{g-1}$, $m \in M_j$, let (b)

$e_{jm}(g) := \max \{ e_{jm}(g\text{-}1),\ \tau_{g-1} \}$;

compute K_g;

$\tau_g := \min \{ \min \{ T_j + d_{jm_j} \mid j \in P_g \},\ \min \{ e_{jm}(g) \mid j \in K_g, m \in M_j \} \}$;

$F_g := \{ j \in A_g \mid T_j + d_{jm_j} \le \tau_g \}$;

$P_g := A_g \setminus F_g$;

go to Step 2.

Step 5: **(Backtracking)**

$g := g - 1$;

if $g = 0$ then STOP, otherwise go to Step 3.

Step 6: **(Solution update)**

If $\tau_g < Best$ then $Best := \tau_g$ and $S := PS_g$;

go to Step 5.

Table 3: The Algorithm

4 Discussion of the Algorithm

In this section some problems associated with the algorithm are outlined. It will be shown that in some cases it does not find an optimal solution. Two instances with constant resource availability and consumption are presented.

Remark 2. In case of scarce nonrenewable resources, the algorithm might not lead to any existing feasible solution.

Instance 1. Consider the instance given in Figure 1 and Table 4. Obviously, in every feasible solution activity 2 has to be performed in mode 2, otherwise activity 3 could not be accomplished because of the total request of four units of the nonrenewable resource.

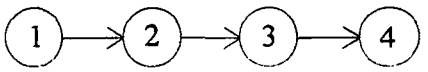

Figure 1: Network of Instance 1

j	m	d_{jm}	q_{jm1}	Q_1
1	1	0	0	3
2	1	1	2	
	2	2	1	
3	1	1	2	
4	1	0	0	

Table 4: Durations, Consumptions and Availability of Instance 1

In the following we will illustrate how the algorithm is dealing with this instance.

Step 1: *(Initialisation)*

$g = 1$; $PS_1 = \{(1,0,1)\}$; $A_1 = \{1\}$; $\tau_1 = 0$; $F_1 = \{1\}$; $P_1 = \varnothing$; $Best = 9999$;

$e_{2,1}(1) = 0$; $e_{2,2}(1) = 0$; $e_{3,1}(1) = 1$; $e_{4,1}(1) = 2$.

Step 2: *(Maximal extensions)*

$B_1 = \{2\}$; assignments: $\{(2,0,1)\}$ and $\{(2,0,2)\}$.

$\{(2,0,1)\}$ dominates $\{(2,0,2)\}$ because if activity 2 is performed in mode 1, it is finished earlier than in mode 2. Therefore, $\{(2,0,1)\}$ is the only dominating assignment. Furthermore, it is the only maximal extension, i. e. $\mathcal{H}_1 = \{\{(2,0,1)\}\}$.

Step 3: *(Next maximal extension)*

$\overline{H} = \{(2,0,1)\}$; $H = \{2\}$; $\mathcal{H}_1 = \emptyset$.

Step 4: *(Branching)*

$g = 2$; $A_2 = \{1, 2\}$; $PS_2 = \{ (1,0,1), (2,0,1) \}$; $P_2 = \{2\}$;

$e_{3,1}(2) = 1$, $e_{4,1}(2) = 2$ (case (b));

$\tau_2 = 1$ (the completion time of activity 2);

$F_2 = \{1, 2\}$; $P_2 = \emptyset$.

Step 2: *(Maximal extensions)*

$B_2 = \{3\}$. We have no maximal extensions at this level, i. e. $\mathcal{H}_2 = \emptyset$, because it is impossible to schedule activity 3 in any mode and any start time with the partial schedule PS_2 due to the total resource request of 4 units.

Since there are no maximal extensions at this level, a complete schedule cannot be found in this part of the tree, and backtracking to level 1 occurs. As we have no more maximal extensions left at level 1, another backtracking step to level 0 is made, and the algorithm stops.

Since Step 6 has not been reached, the algorithm does not find a complete schedule. That is, the algorithm terminates without determining the existing feasible (and optimal) solution. ■

Remark 3. In case of at least two renewable resources, the algorithm might not find an existing optimal solution.

Instance 2. Consider the instance given in Figure 2 and Table 5. If activity 2 is scheduled in mode 1, it cannot be in progress at the same time as activity 4. If activity 2 is accomplished in mode 2, it cannot be in progress at the same time as activity 3 if the latter is scheduled in mode 1. In both cases, the project takes at least five periods. However, we obtain a project duration of four periods by scheduling activity 2 in mode 2 and activity 3 in mode 2. This unique optimal solution is shown in Figure 3, where $j(m)$ stands for activity j being performed in mode m.

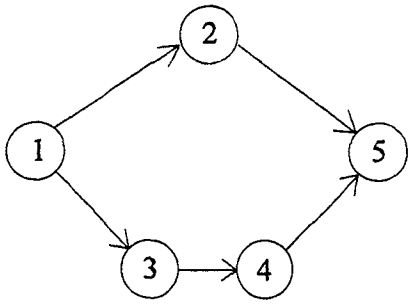

Figure 2: Network of Instance 2

j	m	d_{jm}	w_{jm1}	w_{jm2}	W_1	W_2
1	1	0	0	0	4	4
2	1	3	2	2		
	2	4	1	3		
3	1	1	2	2		
	2	2	2	1		
4	1	2	3	1		
5	1	0	0	0		

Table 5: Durations, Requests and Availabilities of Instance 2

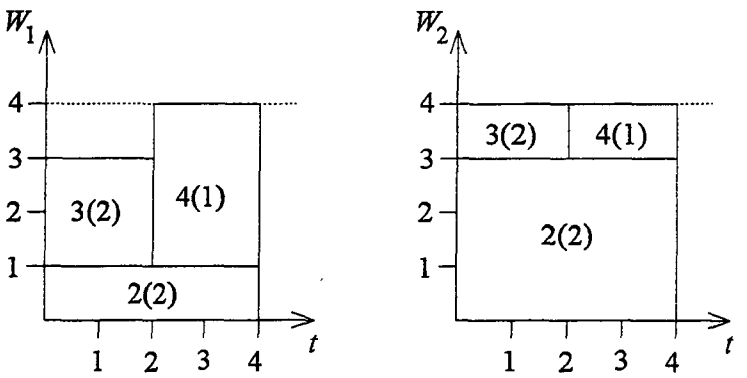

Figure 3: Resource Usages of the Unique Optimal Solution

Step 1: *(Initialisation)*

$g = 1$; $PS_1 = \{(1,0,1)\}$; $A_1 = \{1\}$; $\tau_1 = 0$; $F_1 = \{1\}$; $P_1 = \varnothing$; $Best = 9999$;

$e_{2,1}(1) = 0$; $e_{2,2}(1) = 0$; $e_{3,1}(1) = 0$; $e_{3,2}(1) = 0$; $e_{4,1}(1) = 1$; $e_{5,1}(1) = 3$.

Step 2: *(Maximal extensions)*

$B_1 = \{2, 3\}$.

First, we do not make any assumption on T_j for each activity $j \in I$ in the definition of dominance. In this case, we have to consider the following five assignments:

$\alpha = \{(2,0,1), (3,0,1)\}$, $\beta = \{(2,0,1), (3,0,2)\}$, $\gamma = \{(2,0,2), (3,4,1)\}$,

$\delta = \{(2,1,2), (3,0,1)\}$, $\varepsilon = \{(2,0,2), (3,0,2)\}$.

Note, we do not have to consider assignments which contain only one element because they cannot be maximal (cf. Remark 1). Now we deduce that α dominates β, β dominates ε, ε dominates γ and α dominates δ. Therefore, α is the only dominating assignment, and the set of maximal extensions is given by

$\mathcal{H}_1 = \{\alpha\} = \{ \{(2,0,1), (3,0,1)\} \}$.

Second, we regard the case in which $T_j = \tau$ for each activity $j \in I$ is assumed in the definition of dominating assignments. We have four assignments all of which contain one activity:

$\alpha' = \{(2,0,1)\}$, $\beta' = \{(2,0,2)\}$, $\gamma' = \{(3,0,1)\}$, $\delta' = \{(3,0,2)\}$,

and we have three assignments which contain two activities each:

$\varepsilon' = \{(2,0,1), (3,0,1)\}$, $\varphi' = \{(2,0,1), (3,0,2)\}$, $\kappa' = \{(2,0,2), (3,0,2)\}$.

Note, $\{(2,0,2), (3,0,1)\}$ is an infeasible assignment because in the first period, 5 units of resource 2 would be requested. We deduce that α' dominates β', γ' dominates δ', ε' dominates φ' and φ' dominates κ', so α', γ' and ε' are the only dominating assignments. But since activity 3 (in mode 1) could be added to α' at time 0 and activity 2 (in mode 1) could be added to γ' at time 0, ε' is the only maximal extension in this case, i. e. $\mathcal{H}_1 = \{ \{(2,0,1), (3,0,1)\} \}$.

Thus, it makes no difference for this example whether $T_j = \tau$ is assumed or not. In both cases, we have $\mathcal{H}_1 = \{ \{(2,0,1), (3,0,1)\} \}$.

Step 3: *(Next maximal extension)*

$\overline{H} = \{(2,0,1), (3,0,1)\}$; $H = \{2, 3\}$; $\mathcal{H}_1 = \varnothing$.

Step 4: *(Branching)*

$g = 2$; $A_2 = \{1, 2, 3\}$; $PS_2 = \{(1,0,1), (2,0,1), (3,0,1)\}$; $P_2 = \{2, 3\}$;

$e_{4,1}(2) = 1$, $e_{5,1}(2) = 3$ (case (b));

$\tau_2 = 1$ (because activity 3 ends at time 1); $F_2 = \{1, 3\}$; $P_2 = \{2\}$.

Now Step 2 is performed. We obtain $B_2 = \{4\}$. Since the maximal extensions that are computed at this level consist of the activities $P_2 \cup B_2 = \{2, 4\}$, the mode number of activity 3 remains unchanged in this part of the tree. Therefore, activity 3 will be performed in mode 1, and no maximal extension obtainable from the set of activities $\{2, 4\}$ will lead to an optimal solution (in which activity 3 *must* be performed in mode 2). This is why we may skip this part of the branch-and-bound tree. Since there are no more maximal extensions left at this level, backtracking to level 1 occurs. As we have no more maximal extensions left at level 1, another backtracking step to level 0 is made, and the algorithm stops. The algorithm does not find the optimal solution. Since activity 2 is scheduled in mode 1, only a suboptimal solution for this problem can be found by the algorithm. ■

In both counterexamples the algorithm could not find an optimal schedule. Now we will study the causes of the problem.

As already mentioned in Proposition 4.2 in [1], the algorithm is supposed to enumerate all tight schedules. Then, according to Proposition 1, an optimal schedule will be achieved. This aim should be reached by scheduling the dummy source activity and adding maximal extensions to the current partial schedule.

The definition of dominance which is included in the definition of maximal extensions strongly resembles the definition of tight schedules. It seems that tight schedules should be obtained by

considering 'tight' maximal extensions and therefore 'tight' partial schedules. Since 'tight' has not been defined for partial schedules, we could extend the definition of *tight* to partial schedules as follows:

Definition 6. A partial schedule $PS = \{ (j, T_j, m_j) \mid j \in I \subseteq J \}$ is called *tight* if there does not exist an activity j, a mode number $m'_j \in M_j$ and a start time $T'_j \in \mathbf{N}_0$, such that

$$PS' := PS \setminus \{ (j, T_j, m_j) \} \cup \{ (j, T'_j, m'_j) \}$$

is a feasible partial schedule and $T'_j + d_{jm'_j} < T_j + d_{jm_j}$.

Now, let PS be a partial schedule, and let $\overline{H}$ be a maximal extension of PS. Then we can deduce from the definition of maximal extensions that $PS + \overline{H}$ is tight if PS is tight. Thus it seems that tight schedules shall be produced by considering only tight partial schedules according to the definition above. Therefore, the algorithm produces only tight schedules. However, the counterexamples show that the algorithm does not enumerate *all* tight schedules and therefore fails to find an optimal solution in some cases. Furthermore, the examples show that partial schedules which are not tight have to be accepted in order to obtain all the complete tight schedules.
Consider Instance 1. Obviously, { (1,0,1), (2,0,2), (3,2,1), (4,3,1) } is the unique optimal schedule. Furthermore, it is tight. The partial schedule $PS = \{ (1,0,1), (2,0,2) \}$ is not tight, and it is not accepted by the algorithm. However, PS is a partial schedule of the only complete tight schedule. This example shows that a partial schedule which is not tight can be extended to a tight complete schedule. In this case there does not exist any tight partial schedule containing the activities 1 and 2 which can be extended to a complete schedule.

5 Conclusions

We analyzed an algorithm which is supposed to find an optimal solution for the multi-mode resource-constrained scheduling problem with makespan minimization as objective. The basic framework of the algorithm, the notion of tight schedules and maximal extensions have been thoroughly studied. Thereby, it has been pointed out that (optimal) tight schedules are excluded from consideration if the evaluation of partial schedules is reduced to maximal extensions producing a partial schedule.

Reference

[1] Speranza, M. G. and C. Vercellis (1993): "Hierarchical Models for Multi-Project Planning and Scheduling". *European Journal of Operational Research*, Vol. 64, pp. 312-325.