

A Note on “Hierarcical Models for Multi-Project Planning and Scheduling”

Sönke Hartmann, Arno Sprecher

Institut für Betriebswirtschaftslehre
Lehrstuhl für Produktion und Logistik
Christian-Albrechts-Universität zu Kiel
24098 Kiel, Germany

e-mail: hartmann@bwl.uni-kiel.de, sprecher@bwl.uni-kiel.de

Abstract: We consider the multi-mode resource-constrained project scheduling problem. The focus of our analysis is on an algorithm recently proposed by Speranza and Vercellis for finding makespan minimal solutions. The correctness of the algorithm is examined. By counterexamples we illustrate that the algorithm does not generally find (existing) optimal solutions.

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

1 Introduction

In a recent paper, Speranza and Vercellis [1] have suggested a branch-and-bound algorithm for solving the multi-mode resource-constrained project scheduling problem with makespan minimization as objective.

Introducing tight schedules, Speranza and Vercellis show that the set of tight schedules is dominating the set of schedules, that is, an enumeration procedure can be reduced to the set of tight schedules without losing optimality. Since non-tight schedules are excluded from the search space, such an algorithm is claimed to be very efficient. The authors suggest an algorithm which uses *maximal extensions* of partial schedules to derive tight schedules.

Unfortunately, the algorithm presented sometimes excludes all the optimal tight schedules from evaluation. We will show by counterexamples that the algorithm does not always find an optimal solution. In addition, for some problems the procedure does not even find an existing feasible solution.

The remainder of the paper is organized as follows: After the description of the model in Section 2, we present a formally revised version of the algorithm in Section 3. Section 4 provides two examples the algorithm cannot solve to optimality. 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 $Pred_j$ 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} , $t = 1, \dots, T$, where T denotes an upper bound on the project's makespan. For each nonrenewable resource r , $r \in N$, the overall capacity for the entire 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, $w_{jmr t}$ denotes the usage of renewable resource r , $r \in R$, in the t 'th period activity j is in process, $t = 1, \dots, d_{jm}$. 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 a single mode, that is, $M_1 = M_n = \{1\}$, with a duration of zero periods and no requests for 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 project's makespan.

V	: set of activities in the project
M_j	: set of modes of activity j
$Pred_j$: 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 process
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. Let \mathbf{Z}_+ denote the set of the nonnegative integers. A *schedule* is a set

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

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 $T_j, T_j \in \mathbf{Z}_+$, and the mode $m_j, m_j \in M_j$.

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 feasible schedule $S = \{ (j, T_j, m_j) \mid j \in V \}$ is called *tight* if there does not exist an activity $j, j \in V$, a mode $m'_j, m'_j \in M_j$, and a start time $T'_j, T'_j \in \mathbf{Z}_+$, 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 feasible schedule is tight if there does not exist an activity $j, j \in V$, the finish time of which can be reduced without violating the constraints or changing the start times or modes of any of the remaining activities.

Proposition 1. (Cf. [1], Proposition 4.1.) If there exists an optimal schedule for a given MRCPS, 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 $T'_j \in \mathbf{Z}_+$ 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 $j \neq |I|$. 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 the tight schedules for 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 is claimed to be more efficient than an algorithm that additionally examines non-tight schedules.

Using the idea to enumerate only tight schedules, the algorithm seeks to evaluate and extend partial schedules by adding so-called maximal extensions. For thoroughly studying the extension procedure, we need some definitions:

Definition 3. (Cf. [1].) Let PS be a partial schedule, and let τ be a time instant. An unscheduled activity $j, j \in V$, 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 $m_j, m_j \in M_j$, 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 , respectively. 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 modes (cf. [1]). 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,

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

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

$$P = A \setminus F .$$

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

$$L = \{ (j, T_j, m_j) \mid j \in I, T_j \in \mathbf{Z}_+ \text{ and } m_j \in M_j \}$$

let $PS' := PS + L$.

(a) The assignment of start times and modes 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 $m'_j, m'_j \in M_j$, and a start time $T'_j, T'_j \in \mathbf{Z}_+$, such that

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

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

(b) A *maximal extension* \bar{H} is a dominating assignment of start times and modes for which the related set of activities H is a maximal subset of $B \cup P$.

Remark 1. In [1], it is not explicitly mentioned how to determine the start times T_j of the activities $j, j \in I$, 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 would contain all the activities of $I = B \cup P$. We will return to this question in Section 4.

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 and clarifications.

The algorithm described in [1] is a depth first search branch-and-bound algorithm. First, the earliest start time est_j for each activity $j, j \in V$, 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 to start at time $\tau_1 = 0$.

In Step 2, the algorithm checks whether all activities have been scheduled. If this holds true the current solution is stored in Step 6, and backtracking is performed in Step 5 by decrementing g . If thereby level $g = 0$ is not reached, then Step 3 is executed.

Otherwise, if there are unscheduled activities left in Step 2, the set of the free activities, B_g , at the current level g of the branch-and-bound tree and the set of the maximal extensions, H_g , are determined.

In Step 3, a maximal extension \bar{H}_g is chosen and removed from H_g . If no maximal extension is available, backtracking is performed.

In Step 4, the algorithm branches to the next level, that is, g is incremented. The partial schedule at level g , PS_g , is obtained by adding the selected maximal extension \bar{H}_{g-1} to PS_{g-1} . Moreover, the set of the activities scheduled in PS_g , A_g , and the set of the activities currently in process, P_g , are obtained by adding the set of activities contained in \bar{H}_{g-1}, H_{g-1} , to A_{g-1} and P_{g-1} , respectively.

New earliest start times are computed as follows: If activity j was either free or in process at the previous level but not in the current maximal extension (case (a)), $e_{jm}(g)$ is defined as the maximum of $e_{jm}(g-1)$ 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)), $e_{jm}(g)$ is determined as the maximum of $e_{jm}(g-1)$ and time τ_{g-1} .

Furthermore, a new time instant τ_g is determined as the minimum of the earliest finish time of the activities in process and the earliest possible start time of any unscheduled activity the predecessors of which are finished at time τ_{g-1} . Then the set of the activities that finish at or before time τ_g , F_g , is computed and the set of the activities currently in process, P_g , is updated. Now the next set of maximal extensions is determined in Step 2.

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 process 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)
\bar{H}_g	: maximal extension of the current partial schedule
H_g	: set of the activities contained in the corresponding maximal extension \bar{H}_g
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: (Initialization)

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

Step 3: (Next maximal extension)

If $H_g = \emptyset$ then go to Step 5,
otherwise select a maximal extension \bar{H}_g from H_g and remove it from H_g .

Step 4: (Branching)

$g := g + 1$;
 $A_g := A_{g-1} \cup H_{g-1}$;
 $PS_g := PS_{g-1} + \bar{H}_{g-1}$;
 $P_g := P_{g-1} \cup H_{g-1}$;
for each activity-mode combination (j,m) , $j \in B_{g-1} \cup P_{g-1} \setminus H_{g-1}$, $m \in M_j$, let (a)
 $e_{jm}(g) := \max \{ e_{jm}(g-1), \min \{ T_j \mid PS_g \cup \{ (j, T_j, m) \} \text{ is feasible } \} \}$;
for each activity-mode combination (j,m) , $j \in (V \setminus A_{g-1}) \setminus B_{g-1}$, $m \in M_j$, let (b)
 $e_{jm}(g) := \max \{ e_{jm}(g-1), \tau_{g-1} \}$;

$K_g := \{ j \in V \setminus A_g \mid \text{for every } i, i \in Pred_j \text{ it is } i \in A_g \text{ and } T_i + d_{im_i} \leq \tau_{g-1} \}$;

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

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

$P_g := A_g \setminus F_g$;

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. If the nonrenewable resources are scarce, the algorithm might not find any existing feasible solution.

Instance 1. We consider the instance given in Table 4. The corresponding network can be found in Figure 1. 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.

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

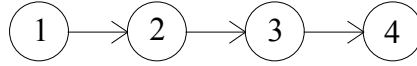


Figure 1: Network of Instance 1

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

Step 1: (Initialisation)

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

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

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. $H_1 = \{ \{(2,0,1)\} \}$.

Step 3: (Next maximal extension)

$\bar{H}_1 = \{(2,0,1)\}$; $H_1 = \{2\}$; $H_1 = \emptyset$.

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));

$K_2 = \emptyset$; $\tau_2 = 1$ (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. $H_2 = \emptyset$, 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. If at least two renewable resources have to be taken into account, the algorithm might not find an existing optimal solution.

Instance 2. Consider the instance given in Table 5. The network of this instance is shown in Figure 2. If activity 2 is scheduled in mode 1, it cannot be in process at the same time as activity 4. If activity 2 is accomplished in mode 2, it cannot be in process at the same time as activity 3 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 .

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

Table 5: Durations, Requests and Availabilities of Instance 2

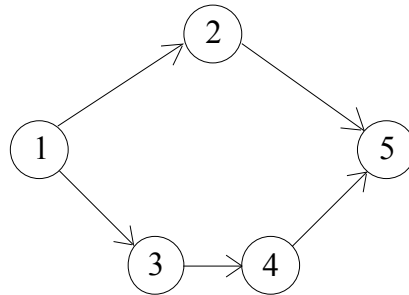


Figure 2: Network of Instance 2

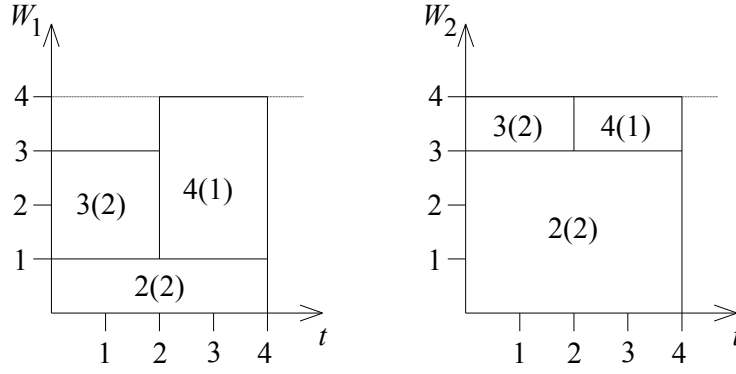


Figure 3: Resource Usages of the Unique Optimal Solution

Step 1: (Initialization)

$g = 1$; $PS_1 = \{(1,0,1)\}$; $A_1 = \{1\}$; $\tau_1 = 0$; $F_1 = \{1\}$; $P_1 = \emptyset$; $Best = 9999$;
 $e_{1,1}(1) = 0$; $e_{2,1}(1) = 0$; $e_{2,2}(1) = 0$; $e_{3,1}(1) = 0$; $e_{3,2}(1) = 0$; $e_{4,1}(1) = 1$; $e_{5,1}(1) = 3$.

Step 2: (Maximal extensions)

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

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

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

Note, we do not have to consider assignments which contain only one element because they cannot be maximal (cf. Remark 1). 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

$$H_1 = \{\alpha\} = \{ \{(2,0,1), (3,0,1)\} \}.$$

Second, we assume $T_j = \tau_g$ for each activity j , $j \in I$, in the definition of dominating assignments. We have four assignments all of which contain one activity:

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

and we have three assignments which contain two activities each:

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

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' . That is, α' , γ' and ε' are the only dominating assignments. But since $(3,0,1)$ could be added to α' and $(2,0,1)$ could be added to γ' , ε' is the only maximal extension, i. e. $H_1 = \{ \{(2,0,1), (3,0,1)\} \}$.

Thus, for this example it makes no difference whether $T_j = \tau_g$ is assumed or not. In both cases, we have $H_1 = \{ \{(2,0,1), (3,0,1)\} \}$.

Step 3: (Next maximal extension)

$$\bar{H}_1 = \{(2,0,1), (3,0,1)\}; \quad H_1 = \{2, 3\}; \quad H_1 = \emptyset.$$

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));

$K_2 = \emptyset$; $\tau_2 = 1$ (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 computed at this level consist of a set of activities $I, I \subseteq P_2 \cup B_2 = \{2, 4\}$, the mode 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). That is, we can ignore this part of the branch-and-bound tree. Since there are no more maximal extensions left at this level, backtracking to level 1 occurs. At level 1, again, no more maximal extensions remain, and we track back to level 0, where the algorithm terminates. That is, the algorithm does not find the optimal solution. Since activity 3 is scheduled in mode 1, only a suboptimal solution for this problem can be found by the algorithm. ■

In both counterexamples the algorithm cannot find an optimal schedule. As already mentioned, Proposition 4.2 (cf. [1]) states that the algorithm proposed separates the set of tight schedules in mutually exclusive sets. Therefore, according to Proposition 1, if all the tight schedules are examined, an optimal schedule will be obtained. The enumeration of all tight schedules should be realized by scheduling the dummy source activity and successively adding maximal extensions to the current partial schedule. Adding a maximal extension to a tight partial schedule without delaying an activity brings out another tight partial schedule. Consequently, in this case, the algorithm produces only tight schedules. However, the counterexamples show that the algorithm does not enumerate *all* the tight schedules. This is due to the fact that in some cases a partial schedule which is not tight can be completed to a tight schedule. In fact, in the examples above, an optimal (tight) schedule can only be generated if at least one intermediate partial schedule is not tight. However, in these examples, any intermediate partial schedule is tight, and the algorithm fails to determine an optimal solution.

5 Conclusions

We have analyzed an algorithm proposed for finding makespan minimal solutions of the multi-mode resource-constrained project scheduling problem. The basic framework of the algorithm, the set of tight schedules and the maximal extensions have been thoroughly studied. Thereby, it has been shown that the exclusive consideration of maximal extensions of the partial schedules might leave (the optimal) tight schedules out of evaluation.

References

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