ELSEVIER European Journal of Operational Research 90 (1996) 334–348

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem

Erik L. Demeulemeester *, Willy S. Herroelen

Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium

Received July 1994; revised November 1994

Abstract

In this paper a branch-and-bound procedure is described for scheduling project activities subject to precedence and resource constraints, where activities can be preempted at any discrete time instant and where the objective is to minimize the project duration. The procedure is based on a depth-first solution strategy in which nodes in the solution tree represent resource and precedence feasible partial schedules. Branches emanating from a parent node correspond to exhaustive and minimal combinations of activities, the delay of which resolves resource conflicts at each parent node. A precedence based lower bound and several dominance rules are introduced in order to restrict the growth of the solutions tree. The solution procedure has been programmed in the C language and extensive computational experience is reported.

Keywords: Project management; Resource constraints; Preemption; Programming; Branch-and-bound; Networks/graphs; Applications

1. Introduction

The planning and control of large projects is a difficult and important problem of modern enterprises that many network planning techniques have tried to handle. In the late 1950's, the development of the now common network planning techniques, such as PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method), made it possible to find minimum duration schedules for projects, assuming that the various resources required for project completion were not a constraining factor. In practice, however, project completion requires the use of various resources, whose often limited availability directly influences planning objectives, time estimations, scheduling and progress control. As a result, research efforts commenced to expand upon these techniques to include the allocation of resources.

One of the important resource allocation problems is the *resource-constrained project scheduling problem* (RCPSP), which involves the scheduling of a project to minimize its total duration subject to precedence constraints and constant availability constraints on the required set of resources. Extensive surveys can be found in Davis [1], Herroelen [7] and Herroelen and Demeulemeester [8].

A fundamental assumption of most optimal and suboptimal procedures for solving the RCPSP is that activities in progress are non-preemptable. As a result very little is known about the potential benefits of activity preemption in the presence of constant

* Corresponding author. E-mail (earn/bitnet): Erik.Demeulemeester@econ.kuleuven.ac.be

0377-2217/96/$15.00 © 1996 Elsevier Science B.V. All rights reserved
SSDI 0377-2217(95)00358-4

and variable resource availabilities. The objective of this paper is to bridge this gap. A branch-and-bound procedure is described for the RCPSP problem under constant and variable resource availabilities, where activities can be preempted at any discrete time instant and resumed later with zero set-up time.

In the next section we formally describe the *preemptive resource-constrained project scheduling problem* (PRCPSP). In Section 3 we offer a review of the literature and show by counterexample that one of the dominance rules used by the only optimal procedure that is specifically developed for the PRCPSP [9] is incorrect, which may cause the procedure to miss the optimum on certain problem instances. In Section 4 we describe our branch-and-bound procedure and illustrate it on a problem example. In Section 5 we report the computational results obtained on a set of test problems with constant and varying resource availabilities. Section 6 is then reserved for our overall conclusions.

2. Problem statement

The classical *resource-constrained project scheduling problem* (RCPSP) is commonly based on the following assumptions:

(a) A project consists of different activities which are represented in the activity-on-the-node format (i.e. a directed, acyclic graph in which the nodes represent the activities and where the arcs denote the precedence constraints). Two dummy activities are introduced: activity 1 represents the start activity of the project and is a (direct or indirect) predecessor of every other activity in the project, while activity n denotes the end activity of the project and is a (direct or indirect) successor of every other activity in the project.

(b) The activities are related by a set of finish–start precedence relations with a time lag of 0, implying that no activity can be started before all its predecessors have completed.

(c) No ready times or due dates are imposed on any of the project activities.

(d) Each activity i $(i = 1,\dots, n)$ has a constant duration d_i (setup times are negligible or are included in the fixed duration).

(e) Each activity i requires a constant number of units r_{ik} of a renewable resource type k $(k = 1,\dots, K)$. The resource requirements r_{ik} are known constants over the processing interval of the activity.

(f) The availability a_k of the renewable resource type k is also a known constant throughout the project duration interval.

(g) No activity can be interrupted once begun (activity preemption is not allowed).

(h) The objective is to complete the project as soon as possible without violating any resource or precedence constraints.

Conceptually, the problem can be formulated as follows [5]:

$$\text{minimize } f_n \tag{1}$$

subject to:

$$f_i \leqslant f_j - d_j \quad \text{for all } (i, j) \in H, \tag{2}$$

$$f_1 = 0, \tag{3}$$

$$\sum_{i \in S_t} r_{ik} \leqslant a_k \quad k = 1,\dots, K \text{ and } t = 1,\dots, f_n, \tag{4}$$

where:

n	=	the number of activities in the project;
K	=	the number of resource types;
f_i	=	the finish time of activity i;
d_i	=	the duration of activity i;
H	=	the set of pairs of activities indicating finish–start precedence relations
S_t	=	the set of activities in progress during time interval $(t-1, t]$
	=	$\{i \mid f_i - d_i < t \leqslant f_i\}$;
r_{ik}	=	the amount of resource type k that is required by activity i;
a_k	=	the total availability of resource type k.

In this formulation the objective function is given as Eq. (1): the project completion time is minimized by minimizing the finish time of the unique dummy end activity n. Constraint set (2) ensures that no activity can be started until all its predecessors have been completed (the finish time of the dummy start activity 1 is assigned a value of 0 as a result of Eq. (3)), while constraint set (4) specifies that the resource utilization during any time interval $(t-1, t]$ does not exceed the resource availability levels for any of the resource types. A variety of integer linear programming formulations for the RCPSP can be found in Wiest [18], Patterson and Huber [12], Patterson and Roth [13] and Pritsker et al. [14], among others.

Numerous researchers have tried to develop efficient solution procedures for the RCPSP by applying integer linear programming, dynamic programming or implicit enumeration (branch-and-bound) methods. To the best of our knowledge, the depth-first branch-and-bound procedure by Demeulemeester and Herroelen [4] is the fastest exact solution method that is capable of solving the RCPSP.

In this paper we will relax the nonpreemption condition of the RCPSP. The *preemptive resource-constrained project scheduling problem* (PRCPSP) allows activities to be preempted at any integer time instant and restarted later on at no additional cost, i.e. the fixed duration d_i of an activity may be split in d_i duration units. Each duration unit $j = 1, 2, \dots, d_i$ of activity i is then assigned a finish time $f_{i,j}$. In order to ease the conceptual formulation a variable $f_{i,0}$ will be used to denote the earliest time that an activity i can be started (which equals the latest finish time of all its predecessors as only finish–start relations with a time lag of 0 are allowed). An activity i belongs to the set S_t of activities in progress at time t if one of its duration units $j = 1, \dots, d_i$ finishes at time t (i.e., if $f_{i,j} = t$). With this variable definition in mind, the PRCPSP can be formulated conceptually as follows:

$$\text{minimize } f_{n,0} \tag{5}$$

subject to:

$$f_{i,d_i} \leqslant f_{j,0} \quad \text{for all } (i, j) \in H, \tag{6}$$

$$f_{i,j-1} + 1 \leqslant f_{i,j} \quad i = 1, \dots, n \text{ and } j = 1, \dots, d_i, \tag{7}$$

$$f_{1,0} = 0, \tag{8}$$

$$\sum_{i \in S_t} r_{ik} \leqslant a_k \quad k = 1, \dots, m \text{ and } t = 1, \dots, f_{n,0}. \tag{9}$$

The objective function (5) minimizes the project length by minimizing the earliest start time of the dummy end activity which by assumption has a duration of 0. Constraint set (6) ensures that all precedence relations are satisfied: the earliest start time of an activity j cannot be smaller than the finish time for the last unit of duration of its predecessor i. In constraint set (7) it is specified that the finish time for every unit of duration of an activity has to be at least one time unit larger than the finish time for the previous unit of duration. Activity 1 is assigned an earliest start time of 0 in Eq. (8), while constraint set (9) stipulates that the resource constraints cannot be violated.

The solution procedure that will be presented in Section 3 is based on the assumption that only two dummy activities with a duration of 0 exist in the project, being the dummy start and end activities. However, as the projects are described using the activity-on-the-node scheme, there is no reason whatsoever to introduce dummy activities except to denote the start and end of a project.

3. Literature review

The introduction of preemption into the RCPSP increases the number of feasible solutions enormously: at every time instant decisions have to be made on what activities have to be scheduled for one time period. Moreover and in contrast with a solution procedure for the RCPSP, the scheduling of part of an activity in the previous period does not restrict the set of possible scheduling decisions in the current period. In view of this increased complexity, most research on the PRCPSP was directed towards finding good heuristic solutions. However, Slowinski [16] and Weglarz [17] have turned their attention towards optimal solutions for the preemptive case, when *continuous* (i.e. non-integer) processing times are assumed for the different activities. This assumption, however, may not be valid in actual situations as practical considerations may dictate *discrete* processing activities.

Davis and Heidorn [2] developed an implicit enumeration scheme, capable of solving the RCPSP under the assumption that the resource requirements can be variable over the duration of the activities. Their solution procedure required activity durations to be represented as a series of *unit duration* tasks, connected by precedence relations of the type 'must immediately precede'. The splitting of activities in unit duration tasks seduced them to claim [2] (p. 815):

"The assumption that jobs must be performed continuously once started is not a necessary requirement of the algorithm. Instead, this assumption was imposed to conform with those generally associated with the version of the project network scheduling problem most often discussed in the literature. If it can be assumed that jobs may be interrupted and restarted later at no cost, then such job "splitting" can easily be handled, at little or no increase in computational requirements for a given problem."

This claim was never verified in later publications. The reader should realize, however, that there is a fundamental difference between job splitting in combination with 'must immediately precede'-precedence relations and preemption. In the former case, the scheduling of a unit duration task of an activity at the last decision point automatically induced the scheduling of the next unit duration task of that activity at the current decision point, thus severely restricting the number of possible scheduling decisions at each decision point and hence also the growth of the branch-and-bound tree. However, if preemption is allowed, the number of possible scheduling decisions at the current decision point only depends on what unit duration tasks have been scheduled previously and not on what unit duration tasks were scheduled during the last time period before the current decision point. Therefore the above claim cannot be corroborated. On the contrary, we are convinced that an enormous increase in the computation times will be observed if the solution procedure is tried on the preemptive case.

To the best of our knowledge, the literature on the preemptive resource-constrained project scheduling problem is almost void. The only solution method that was specifically developed for the PRCPSP as defined in Section 1 is described in the doctoral dissertation of Kaplan [9]. There the PRCPSP is formulated as a dynamic program and is solved using a reaching procedure, incorporating dominance properties and upper/lower bounds on the optimal project duration in order to decrease the amount of computational effort. In a recent paper [10] computational experience is given on 41 projects (problems 20–60) of the well-known Patterson problem set [11]. For these 41 projects the classical resource-constrained project scheduling problems (no preemption allowed) were solved in an average computation time of 50 seconds on an IBM PS/2 Model 70 with an Intel 80386 processor running at 16 Mhz along with a numeric coprocessor. Allowing preemption the average computation time increased with a factor of 8.5 and amounted to 7 minutes and 5 seconds. In that paper the following extension of the PRCPSP was also described: whenever an activity is split, an additional setup time is added to the remaining duration of the activity (in the computational experience section of that paper the setup time was put equal to half the duration of the activity involved). As could be expected, a significant increase in the solution times was found for this extension (for 5 out of the 41 problems an optimal solution could not be found or confirmed after two hours of computation).

The solution procedure presented by Kaplan makes use of the following dominance rule [9] (page 55):

"THEOREM 3.14. Suppose the available amount per period of each resource type remains constant over time. Then there exists an optimal schedule with the following property:

preemption of an activity in progress at state δ can occur only if another activity is completed at δ (and not completed at $p(\delta)$)."

In this theorem, state δ is defined to be the vector of the scheduled activity durations $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $p(\delta)$ represents the last state prior to δ in the shortest path from 0 to δ (the meaning of these variables will be clarified below). It can be shown by counterexample that this dominance rule is incorrect and may cause the Kaplan procedure to miss the optimal solution. Consider the project in Fig. 1 and assume that this project is subject to a constant resource availability of two units for a single resource type.

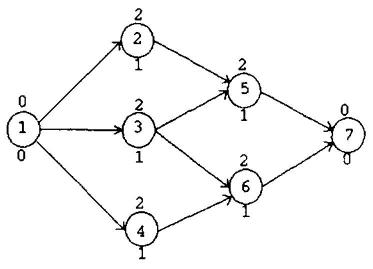

Fig. 1. Counterexample to Kaplan's theorem 3.14. (Circles denote activities with duration given above and resource requirements below the activity.)

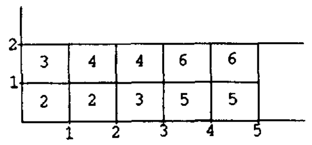

Fig. 2. An optimal solution for the counterexample.

For this project several optimal solutions can be found with a project length of 5. The resource profile for one of them is shown in Fig. 2. This optimal solution is constructed along the following path in the dynamic program:

$\delta_1 = (0, 0, 0, 0, 0, 0, 0) \to$
$\delta_2 = (0, 1, 1, 0, 0, 0, 0) \to$
$\delta_3 = (0, 2, 1, 1, 0, 0, 0) \to$
$\delta_4 = (0, 2, 2, 2, 0, 0, 0) \to$
$\delta_5 = (0, 2, 2, 2, 1, 1, 0) \to$
$\delta_6 = (0, 2, 2, 2, 2, 2, 0)$

Along this path six states are considered, one of them being state $\delta_3 = (0, 2, 1, 1, 0, 0, 0)$. For this state $p(\delta_3)$ equals the state $\delta_2 = (0, 1, 1, 0, 0, 0, 0)$. This optimal schedule, however, does not satisfy the property of theorem 3.14: in constructing state δ_3 from δ_2 a preemption of activity 3 (which was in progress at state δ_2) occurred while no other activity was completed at state δ_2.

For this example project there exist only three semi-active schedules that satisfy the property of theorem 3.14. The paths in the dynamic programming solution for these three schedules are as follows:

(a) $(0, 0, 0, 0, 0, 0, 0) \to (0, 1, 1, 0, 0, 0, 0) \to$
$(0, 2, 2, 0, 0, 0, 0) \to (0, 2, 2, 1, 1, 0, 0) \to$
$(0, 2, 2, 2, 2, 0, 0) \to (0, 2, 2, 2, 2, 1, 0) \to$
$(0, 2, 2, 2, 2, 2, 0)$

(b) $(0, 0, 0, 0, 0, 0, 0) \to (0, 1, 0, 1, 0, 0, 0) \to$
$(0, 2, 0, 2, 0, 0, 0) \to (0, 2, 1, 2, 0, 0, 0) \to$
$(0, 2, 2, 2, 0, 0, 0) \to (0, 2, 2, 2, 1, 1, 0) \to$
$(0, 2, 2, 2, 2, 2, 0)$

(c) $(0, 0, 0, 0, 0, 0, 0) \to (0, 0, 1, 1, 0, 0, 0) \to$
$(0, 0, 2, 2, 0, 0, 0) \to (0, 1, 2, 2, 0, 1, 0) \to$
$(0, 2, 2, 2, 0, 2, 0) \to (0, 2, 2, 2, 1, 2, 0) \to$
$(0, 2, 2, 2, 2, 2, 0)$

These are the only three paths that are generated if theorem 3.14 is applied. As all three solutions have a project length of 6, we are led to believe that the optimal project length equals 6. However, as already indicated, there exist various solutions with a project length of 5, but for these solutions we have to preempt activities at time instants where no other activity completes. Hence, theorem 3.14 is clearly incorrect. The optimal solutions obtained by Kaplan for the selected 41 problem instances from the Patterson set seem to indicate, however, that on larger problem instances the described anomaly will only occur at rare occasions.

4. Solution methodology

4.1. The introduction of subactivities

In order to ease the description of the solution procedure we will distinguish between activities and subactivities. At the start of the procedure we will create a new project network in which all activities are replaced by one or more subactivities. The dummy start and end activities are replaced by dummy start and end subactivities with a duration of 0. All other activities (with, by assumption, a nonzero duration) are split into subactivities, the number of these subactivities being equal to the duration of the original activity. Each of these subactivities has a duration of 1 and resource requirements that are equal to those of the corresponding activity.

The nodes in the branch-and-bound search tree correspond with partial schedules in which finish times have been assigned to a number of subactivities. The partial schedules are feasible, satisfying both precedence and resource constraints. *Partial schedules* PS_t are only considered at those time instants t which correspond with the completion of one or more subactivities. A partial schedule PS_t at time t thus consists of the set of scheduled subactivities. An *unfinished activity* at time t is an activity for which not all subactivities have been scheduled.

Partial schedules are built up starting at time 0 and proceed systematically throughout the search process by adding at each decision point subsets of activities until a complete feasible schedule is obtained. In this sense, a complete schedule is a *continuation* of a partial schedule. At every time instant t we define the *eligible set* E_t as the set of activities for which one of the subactivities is eligible to start given the precedence constraints. These eligible activities can start at time t if the resource constraints are not violated.

In the next section we describe a number of dominance rules which may be used to restrict the growth of the search tree.

4.2. Dominance rules

A first dominance rule heavily restricts the number of feasible schedules that have to be considered during the solution procedure. This dominance rule is based on the notion of a semi-active timetabling schedule, which was already defined by French [6]: a schedule is constructed by *semi-active timetabling* if no activity can be started earlier in the schedule without violating the precedence or resource constraints. However, for the PRCPSP we have to consider semi-active timetabling at the level of the subactivities, as indicated in Theorem 1:

Theorem 1. *In order to solve the PRCPSP it is sufficient to construct the partial schedules by semi-active timetabling at the level of the subactivities.*

Proof. See Appendix.

The branch-and-bound procedure for the RCPSP, developed by Demeulemeester and Herroelen [4], uses two dominance rules which determine at certain time instants t which eligible activities must be scheduled by themselves and which pair of eligible activities are to be scheduled concurrently to start at time t in an optimal continuation of the corresponding partial schedule. In this paper, we extend the dominance rules to apply to the PRCPSP. For that purpose, some further definitions are in order. Consider an eligible, unfinished activity i at time instant t. Denote the z unfinished subactivities of activity i at time t as $i_1, i_2, \ldots, i_z$. We say that activity i is *scheduled immediately* at time t if all its remaining subactivities i_x $(x = 1, \ldots, z)$ are scheduled such that $f_{i_x} = t + x$. The second theorem can then be formulated as follows.

Theorem 2. *If for a partial schedule* PS_t *at time instant t, there exists an eligible activity i that cannot be scheduled together with any other unfinished activity j at any time instant $t' \geqslant t$ without violating the precedence or resource constraints, then an optimal continuation of* PS_t *exists with all remaining subactivities $i_1, i_2, \ldots, i_z$ of activity i scheduled immediately at time t.*

Proof. See Appendix.

It should be observed that it is not necessary to check whether an activity is in progress at time t or not. This is possible because the preemption assumption allows us to forget the scheduling decisions in the previous period and to consider only those possibilities implied by the set of eligible subactivities. In addition, it is important to note that the theorem implies that it is necessary to check whether an eligible activity can be scheduled together with any other unfinished activity at any time instant t', greater than or equal to t. Theorem 2 can easily be extended to the case where an eligible activity can be scheduled concurrently with only one other activity, which is also eligible. The exact statement of this dominance rule can be found in Theorem 3:

Theorem 3. *If for a partial schedule* PS_t *at time instant t, there exists an eligible activity i that can be scheduled together with only one other unfinished activity j at any time instant $t' \geqslant t$ without violating the precedence or resource constraints and if activity j is eligible, then an optimal continuation of* PS_t *exists with all remaining subactivities of activity i scheduled immediately and with as many subactivities of activity j as possible scheduled concurrently with the subactivities of activity i.*

Proof. See Appendix.

It should be noticed that in Theorem 3 no test needs to be performed to check whether the remaining duration of activity j is larger than that of activity i. Indeed, if the remaining duration of activity j is larger, we will schedule as many subactivities of activity j as there are unscheduled subactivities in activity i. If, however, the remaining duration of activity j is smaller or equal, we will schedule all remaining subactivities of activity j concurrently with those of activity i. Again, the time instant t' may be greater than or equal to t.

If it is impossible to schedule all eligible subactivities at time t, a so-called *resource conflict* occurs. Such a conflict will produce a new *branching* in the branch-and-bound tree. The branches describe ways to resolve the resource conflict; i.e. decisions about which combinations of subactivities are to be delayed. We define a *delaying set* $D(p)$ which consists of all subsets D_q of eligible subactivities, the delay of which would resolve the current resource conflict at level p of the search tree. A delaying alternative D_q is *minimal* if it does not contain other delaying alternatives $D_v \in D(p)$ as a subset. The next dominance rule allows the procedure to delete from explicit consideration a large number of redundant delaying alternatives.

Theorem 4. *In order to resolve a resource conflict, it is sufficient to consider only minimal delaying alternatives.*

Proof. See Appendix.

A last dominance rule is based on the cutset dominance rule that was described in Demeulemeester [3] and Demeulemeester and Herroelen [4]. This theorem compares partial schedules at identical decision points and determines whether the current partial schedule is a subset of a partial schedule that has already been considered. If this is the case, the current partial schedule can be dominated as stated more exactly in Theorem 5.

Theorem 5. *Consider a partial schedule* PS_t *at time* t. *If there exists a partial schedule* PS'_t *that was previously saved at a same time instant* t *and if* PS_t *is a subset of* PS'_t, *then the current partial schedule* PS_t *is dominated.*

Proof. Immediate.

It might be interesting to observe that Theorems 1 and 4 have their counterparts in the algorithm developed by Kaplan [9]. However, Kaplan does not incorporate the dominance rules described in our Theorems 2, 3 and 5, although there is no objection to their use: we think that these three dominance rules could easily be introduced into her procedure.

4.3. Lower bound calculation

Demeulemeester [3] and Demeulemeester and Herroelen [4] have used three types of lower bound calculations for the RCPSP. Their extension to the PRCPSP [3], however, comes at the expense of increased computational requirements. Therefore, only the *remaining critical path lower bound* has been included in the procedure for the PRCPSP. For each delaying alternative D_q that is constructed at decision point t the remaining critical path lower bound l_q is computed as $l_q = t + 1 + \max_{i \in D_q}\{q_i\}$, where q_i denotes the remaining critical path length for subactivity i.

Kaplan [9] tested the introduction of a Lagrangean relaxation based lower bound in her solution procedure. The computational results indicated that this lower bound was 10 percent closer to the optimum than the critical path lower bound, but this at the expense of extra computation time. On average, the Lagrangean relaxation algorithm took over 3.5 times as long as the algorithm based on the critical path lower bound. These results confirm our experience on the use of lower bounds [3] and inspired us to rely on the simple, but efficient calculation of the remaining critical path lower bound.

The dominance rules that were described in Section 4.2 and the critical path lower bound that was presented above will be combined into the branch-and-bound solution procedure of Section 4.4. An example of the application of this implicit enumeration procedure will be presented in Section 4.5.

4.4. The branch-and-bound procedure

The detailed algorithmic steps of the branch-and-bound algorithm are described below. For simplicity of notation the subscripts t for the sets PS, S and E are omitted.

Step 1. Initialisation

– Construct a new project network in which all activities are replaced by subactivities (the durations d_i and the resource requirements r_{ik} that will be mentioned in the sequel of this procedure now refer to subactivities instead of activities).

All subactivities of a non-dummy activity have a duration of 1, while the subactivities of a dummy activity have a duration of 0.
- Let $T = 9999$ be an upper bound on the project duration.
- Set the level of the branch-and-bound tree $p = 0$.
- Initialize the decision point: $t = -1$.
- For every subactivity i compute the remaining critical path length q_i.
- Schedule the dummy start subactivity: $f_1 = 0$, PS $= \{1\}$ and $S = \{1\}$.
- Compute the critical path lower bound at level 0: $\text{lb}_0 = q_1$.
- Compute the eligible set: $E = \{i \mid \text{subactivity } i \text{ has subactivity 1 as a single predecessor}\}$.

Step 2. Augmentation
- If the dummy end subactivity n' has been scheduled, the schedule is completed. Update the schedule length: $T = f_{n'}$. If T equals lb_0, then STOP (with the optimal solution), else update the branching level: $p = p - 1$ and go to Step 7 (backtracking).
- Update the decision point: $t = t + 1$.
- Check if the current partial schedule is dominated by a previously saved partial schedule (Theorem 5). If it is dominated, go to Step 7 (backtrack), else save the current partial schedule and the decision point t.

Step 3. Separation
- For each eligible activity i count the number of unfinished activities that can be scheduled concurrently with activity i at any time instant $t' \geqslant t$ without violating the precedence or resource constraints.
- If no unfinished activity can be scheduled concurrently with activity i, apply Theorem 2: schedule all remaining subactivities i_x $(x = 1, \ldots, z)$ of activity i as soon as possible: $\text{PS} = \text{PS} + \{i_x\}$, $S = \{i_z\}$ and $f_{i_x} = t + x$. Update the decision point $t = f_{i_z} - 1$ and the eligible set $E = E - \{i_1\} + \{y \mid \text{subactivity } y \text{ is a successor of } i_z \text{ and all predecessors of } y \text{ have been scheduled}\}$. There is, however, one small exception: if the dummy end subactivity n' (with a duration of 0) is scheduled during this step, its finish time is put equal to the current decision point: $f_{n'} = t$.
- If only one unfinished activity j can be scheduled together with activity i and if activity j is eligible, apply Theorem 3: schedule all remaining subactivities $i_x(x = 1,.., z)$ of activity i as soon as possible: $\text{PS} = \text{PS} + \{i_x\}$, $S = \{i_z\}$ and $f_{i_x} = t + x$. Let z' be the number of remaining subactivities of activity j at time t. Put $z'' = \min\{z, z'\}$. Schedule the z'' first remaining subactivities j_x $(x = 1,.., z'')$ of activity j as soon as possible: $\text{PS} = \text{PS} + \{j_x\}$, $S = S + \{j_{z''}\}$ and $f_{j_x} = t + x$. Update the decision point $t = f_{i_z} - 1$ and the eligible set $E = E - \{i_1, j_1\} + \{y \mid \text{subactivity } y \text{ is a successor of } i_z \text{ or } j_{z''} \text{ and all predecessors of } y \text{ have been scheduled}\}$.
- If no subactivity has been scheduled during this step, go to Step 4. Otherwise, test whether these scheduling decisions have increased the critical path of the project beyond the current best feasible solution: if $t + 1 + \max_{i \in E}\{q_i\} \geqslant T$, go to Step 7. Otherwise, go to Step 2.

Step 4. Scheduling
- For each resource type k check if $\sum_{i \in E} r_{ik} \leqslant a_k$. If there is at least one resource type k for which the sum of the resource requirements of all eligible activities exceeds the resource availability, we have a resource conflict: go to Step 5.
- If no resource conflict occurred, schedule all eligible subactivities: $\text{PS} = \text{PS} + E$, $S = E$ and for all $i \in E$: $f_i = t + 1$.
- Compute the eligible set $E = \{j \mid j \text{ is a successor of } i \in S \text{ and all predecessors of } j \text{ have been scheduled}\}$ and go to Step 2.

Step 5. Resource conflict
- Update the branch level of the search tree: $p = p + 1$.
- Determine for each resource type k how many units have to be released in order to resolve the current resource conflict: $c_k = \sum_{i \in E} r_{ik} - a_k$.
- Determine the delaying set $D(p) = \{D_q \mid D_q \text{ is a subset of } E, \sum_{i \in D_q} r_{ik} \geqslant c_k \text{ for each resource type } k \text{ and } D_q \text{ does not contain other } D_v \in D(p) \text{ as a subset}\}$.
- For every $D_q \in D(p)$ compute the lower bound: $l_q = t + 1 + \max_{i \in D_q}\{q_i\}$.
- Select the $D_q \in D(p)$ with the smallest l_q (ties are broken arbitrarily) and call this delaying alternative D_b.
- Update the delaying set: $D(p) = D(p) - D_b$.
- Compute $\text{lb}_p = \max\{\text{lb}_{p-1}, l_b\}$. If $\text{lb}_p \geqslant T$, decrease the branching level: $p = p - 1$ and go to Step 7.

– Store the current partial schedule and the eligible set.

Step 6. Delaying

– Schedule all eligible subactivities that do not belong to the chosen delaying alternative: $PS = PS + E - D_b$, $S = E - D_b$ and for all $i \in (E - D_b)$: $f_i = t + 1$.
– Update the eligible set: $E = D_b + \{j \mid \text{subactivity } j$ is a successor of subactivity $i \in S$ and all predecessors of j have been scheduled$\}$.
– Go to Step 2.

Step 7. Backtracking

– If the branching level $p \leqslant 0$, then STOP.
– If $D(p) = \emptyset$, set $p = p - 1$ and repeat Step 7.
– Select the delaying alternative $D_b \in D(p)$ with the smallest lower bound l_b and update the delaying set: $D(p) = D(p) - D_b$.
– Compute $\text{lb}_p = \max\{\text{lb}_{p-1}, l_b\}$. If $\text{lb}_p \geqslant T$, update the branching level: $p = p - 1$ and repeat Step 7.
– Restore the partial schedule and the eligible set and go to Step 6.

4.5. Numerical example

The formal description of the branch-and-bound algorithm for the PRCPSP will be clarified on the example of Fig. 3. This example assumes a constant resource availability of five units for a single resource type. The different steps of the branch-and-bound procedure can now be described as follows.

Step 1. Construct the subactivity project network that is shown in Fig. 4. Initialize the upper bound on the project length $T = 9999$, the branching level $p = 0$ and the decision point $t = -1$. Compute the remaining critical path length for every subactivity

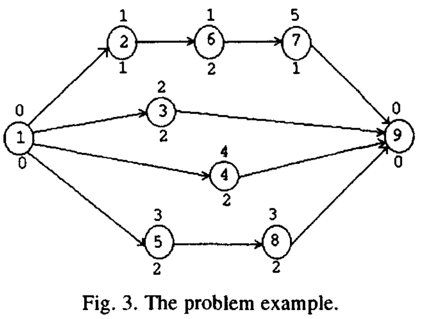

Fig. 3. The problem example.

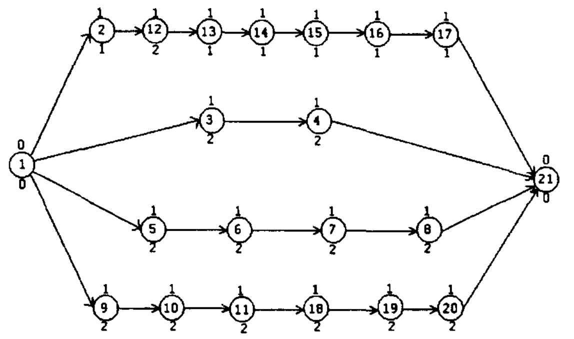

Fig. 4. A subactivity project network.

and schedule the dummy start (sub)activity: $f_1 = 0$, $PS = \{1\}$ and $S = \{1\}$. Update $\text{lb}_0 = q_1 = 7$ and compute $E = \{2, 3, 5, 9\}$.

Step 2. Update $t = 0$. The current partial schedule cannot be dominated: save the current partial schedule and the decision point.

Step 3. Every eligible activity can be scheduled with at least two other unfinished activities (e.g., activity 2 could be scheduled together with activities 3, 4, 5 and 8): no subactivities can be scheduled during this step.

Step 4. The resource requirements for all eligible subactivities amount to 7, whereas only 5 units are available: a resource conflict occurs.

Step 5. Update p = 1 and compute $c_1 = 2$. Determine $D(1) = \{D_1 = \{3\}$ with $l_1 = 3$, $D_2 = \{5\}$ with $l_2 = 5$, $D_3 = \{9\}$ with $l_3 = 7\}$. Select D_1 and update $D(1) = \{\{5\}, \{9\}\}$. Compute $\text{lb}_1 = 7 < T$: store the current partial schedule and the eligible set.

Step 6. Update $PS = \{1, 2, 5, 9\}$, $S = \{2, 5, 9\}$ and $f_2 = f_5 = f_9 = 1$. Compute $E = \{3, 6, 10, 12\}$.

Step 2. Update $t = 1$. The current partial schedule is not dominated: save the current partial schedule and the decision point.

Step 3. Every eligible activity can be scheduled with at least two other unfinished activities: no subactivities can be scheduled during this step.

Step 4. The resource requirements for all eligible subactivities amount to 8, whereas only 5 units are available: a resource conflict occurs.

Step 5. Update $p = 2$ and compute $c_1 = 3$. Determine $D(2) = \{D_4 = \{3, 6\}$ with $l_4 = 5$, $D_5 = \{3, 10\}$ with $l_5 = 7$, $D_6 = \{3, 12\}$ with $l_6 = 8$, $D_7 = \{6, 10\}$ with $l_7 = 7$, $D_8 = \{6, 12\}$ with $l_8 = 8$, $D_9 = \{10, 12\}$

with $l_9 = 8$}. Select D_4 and update $D(2) = \{\{3, 10\}, \{3, 12\}, \{6, 10\}, \{6, 12\}, \{10, 12\}\}$. Compute $lb_2 = 7 < T$: store the current partial schedule and the eligible set.

Step 6. Updates PS = {1, 2, 5, 9, 10, 12}, $S = \{10, 12\}$ and $f_{10} = f_{12} = 2$. Compute $E = \{3, 6, 11, 13\}$.

Step 2. Update $t = 2$. The current partial schedule is not dominated: save the current partial schedule and the decision point.

Continuing the procedure in this manner leads to the delay of subactivity 4 at level $p = 6$ of the solution tree. The algorithm then proceeds along the following steps:

Step 6. Update PS = {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20}, $S = \{8, 16, 20\}$ and $f_8 = f_{16} = f_{20} = 6$. Compute $E = \{4, 17\}$.

Step 2. Update $t = 6$. The current partial schedule is not dominated: save the current partial schedule and the decision point.

Step 3. Activity 3 (for which subactivity 4 is eligible) can be scheduled concurrently with only one other unfinished activity, namely activity 7 (for which subactivity 17 is eligible). We therefore schedule all remaining subactivities of activity 3: PS = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20}, $S = \{4\}$ and $f_4 = 7$. As there were as many unscheduled subactivities in activity 7 as in activity 3 ($z = z' = z'' = 1$), we schedule that single subactivity: PS = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 12, 13, 14, 15, 16, 17, 18, 19, 20}, $S = \{4, 17\}$ and $f_{17} = 7$. As only one subactivity was scheduled for each activity, the decision point remains unchanged: $t = 6$. Update the eligible set: $E = \{21\}$. As $6 + 1 + \max\{0\} = 7 < T$, go to Step 2.

Step 2. Update $t = 7$. The current partial schedule is not dominated: save the current partial schedule and the decision point.

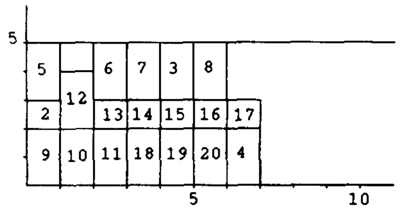

Fig. 5. The optimal schedule at the level of the subactivities.

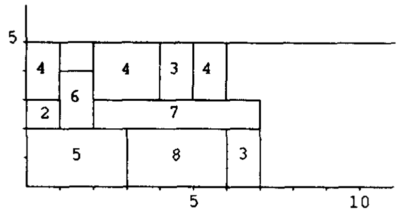

Fig. 6. An optimal preemptive schedule at the level of the activities.

Step 3. Activity 9 (for which subactivity 21 is eligible) cannot be scheduled concurrently with any other unfinished activity (all other activities have completed). Schedule all remaining subactivities of activity 9: PS = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21}, $S = \{21\}$ and $f_{21} = 7$ (the dummy end subactivity has been scheduled). As only one subactivity was scheduled for activity 9, the decision point remains unchanged: $t = 7$. Update the eligible set: $E = \emptyset$. As $7 < T$, go to Step 2.

Step 2. The dummy end subactivity has finished: update $T = 7$. T equals lb_0: STOP with the optimal solution. This optimal schedule of the subactivities is shown in Fig. 5.

Translating this optimal schedule in terms of the original activities, we obtain the optimal preemptive schedule that is shown in Fig. 6. There we see that in this optimal schedule activities 3 and 4 are preempted.

In this small example problem little could be gained by allowing activities to be preempted: the same optimal project length was obtained as when no preemption was allowed [3], while more branching levels were created during the search procedure. In the next section we will explore the costs and benefits of activity preemption in more depth.

5. Computational experience

5.1. Constant resource availabilities

The branch-and-bound algorithm of Section 4.4 has been coded in the C language. We have tested this program on the 41 Patterson problems used by Kaplan [10] on a similar computer, running at 16

Table 1
Computational results for Patterson problems 20–60

Solution procedure	Kaplan	PRCPSP–b & b
Average computation time (seconds)	425	4.9863
Standard deviation	713	9.2932

Mhz. The computational results are summarized in Table 1.

These computational results indicate that for this restricted problem set of 41 problems our procedure is on the average 85.23 times faster than Kaplan's procedure. Despite the erroneous use of the dominance rule mentioned earlier in this paper, the Kaplan procedure manages to generate the optimal solution for all 41 problems.

Using an IBM PS/2 running at 25 Mhz, we have tested the algorithm on all 110 problems in the Patterson set of test problems [11]. All problems could be solved within 5 minutes of computation time: a summary of the computational results can be found in Table 2.

Comparing these results with the computational requirements obtained by the branch-and-bound procedure of Demeulemeester and Herroelen [4] for the nonpreemptive RCPSP, we notice an increase in the average computation time with a factor of 33.82. The question remains whether this increase in the computational requirements can be justified by a corresponding decrease in the optimal project lengths. The effect on the project length reduction is shown in Table 3.

The results in Table 3 indicate that the 33-fold increase in the average computation time from RCPSP to PRCPSP is only compensated by a one time unit reduction in the optimal project duration for 30 of the 110 Patterson problems, constituting an average relative decrease in the optimal project length of 0.7823%. This small improvement in the project length was already recognized by Kaplan. Moreover,

Table 2
Computational results on all 110 Patterson problems

Solution procedure	PRCPSP–b & b
Average computation time (seconds)	6.8985
Standard deviation	25.8149

Table 3
Decrease in the optimal project length if preemption is allowed

Decrease in optimal project length	Number of problems
0	80
1	30

she expects [9, p. 104] that the optimal schedules found by her dynamic programming solution procedures will show greater reductions in project duration (over the nonpreemptive case) in situations where the availability of resources does not remain constant over the course of the project.

In Section 5.2 we will describe what minor adaptations to the branch-and-bound algorithm are necessary in order to make it applicable for the PRCPSP under the assumption of variable resource availabilities. We will subsequently test the Kaplan hypothesis that more significant improvements in the optimal project length can be found if preemption is introduced into a project scheduling problem with variable resource availabilities.

5.2. Variable resource availabilities

When variable resource availabilities are introduced into the PRCPSP, Theorems 2 and 3 are no longer valid. Indeed, as the resource availabilities vary over time, no pairwise interchange can be performed under the guarantee that no resource constraint is violated. Therefore, we cannot apply Step 3 of the solution procedure for this case. All other dominance rules, as well as the critical path lower bound calculation, are still valid for this generalized PRCPSP. The only other adaptation needed is that in Steps 4 and 5 every reference to the resource availability a_k is changed into a_{kt} (its equivalent when variable resource availabilities are allowed).

These adaptations being made, we are ready to apply this solution procedure on the Simpson prob-

Table 4
Computational results for the Simpson problems

Solution procedure	PRCPSP–b & b
Number of problems solved	107
Average computation time (seconds)	12.6321
Standard deviation (seconds)	36.9071

Table 5
Decrease in the optimal project length if preemption is allowed

Decrease in optimal project length	Number of problems
0	38
1	45
2	17
3	8
4	1
5	0
6	0
7	1

lem set that was described in Simpson [15] and in Demeulemeester and Herroelen [5]. This problem set consists of the 110 Patterson problems where the resource availability has become variable over the project horizon. Computational experience with the nonpreemptive case of this problem set can be found in Demeulemeester and Herroelen [5]. The results for the preemptive case are presented in Table 4 (the limit on the computation time was set to 5 minutes).

From Table 4 we can see that not all 110 Simpson problems can be solved to optimality within the allowed time limit. There are three problems for which the optimal project length was not found or confirmed after 5 minutes of computation time. For the 107 problems that were solved optimally, the average solution time of 12.6321 seconds is 12.34 times larger than the average computation time needed by the Demeulemeester and Herroelen procedure [5] when no preemption was allowed. The results of Table 5 clearly show that the improvements in the optimal project length are more significant in the case of variable resource availabilities than when constant resource availabilities are considered (for the variable resource availability case the average relative decrease in the optimal project length is 2.8802%), thus confirming the Kaplan hypothesis.

6. Conclusions

In this paper we have described an efficient depth-first branch-and-bound procedure for optimally solving the preemptive resource-constrained project scheduling problem. The solution procedure is equipped with five powerful dominance rules, which are extensions of the dominance rules that were described in Demeulemeester [3] and Demeulemeester and Herroelen [4] for the basic RCPSP.

The only publication in the literature that deals with the PRCPSP is the doctoral dissertation of Kaplan [9]. She formulates the PRCPSP as a dynamic programming problem and solves this problem by using a reaching procedure which incorporates dominance properties and upper/lower bounds on the optimal project length. We show by counterexample that one of the dominance rules used by the Kaplan procedure is incorrect, which may cause the procedure to miss the optimum on certain problem instances. Moreover, computational experience with 41 Patterson problems indicates that our procedure is on average 85 times faster than Kaplan's.

Solving all 110 Patterson problems with our algorithm, we noticed a 33-fold increase in the computation times when compared to the solution times of the solution procedure for the nonpreemptive RCPSP [3,4]. Notwithstanding this significant increase in the computational requirements a relative decrease in the optimal project lengths of only 0.7823% was found.

Subsequently, the branch-and-bound algorithm for the PRCPSP was extended to the PRCPSP with variable resource availabilities. Testing this algorithm on the Simpson problem set, only 107 of the 110 problems could be solved optimally within a time limit of 5 minutes. For these 107 problems the average computation time was 12 times larger than the time needed for the nonpreemptive version of this problem. The reductions in the optimal project length for this problem set, however, were much larger than in the constant resource availability case (a relative decrease in the optimal project lengths of 2.8802% was observed). These computational results indicate that in project scheduling the introduction of preemption has little effect, except when variable resource availability levels are defined.

Appendix

Proof of Theorem 1. Consider an optimal schedule which has not been constructed by semi-active timetabling at the level of the subactivities. Then there is at least one subactivity which could be started earlier. Of all such subactivities choose one with the earliest finish time (ties are broken arbitrarily). Retimetable this subactivity to start as early as possible without violating any precedence or resource constraints. In this new schedule no finish time has increased and thus the schedule length remains unchanged.

Repeat this process of retimetabling subactivities, which could have started earlier, until there are no such subactivities left. Note that, because we always pick a subactivity which has the earliest finish time, no subactivity can be retimetabled more than once. There are a finite number of subactivities so this retimetabling process must terminate. The final schedule is the result of semi-active timetabling, since no subactivity could be started earlier. Moreover, since the schedule length didn't increase at any stage, the final schedule is at least as good as the original. Hence, there exists an optimal schedule that can be found by semi-active timetabling at the level of the subactivities. □

Proof of Theorem 2. Let schedule G be an optimal continuation of the partial schedule PS_t with finish times f_y for all subactivities y. Let activity i satisfy the assumptions of the theorem. If all subactivities of activity i are scheduled immediately at time t in G or no such i exists, we are done.

Otherwise, let $i_1, i_2, \dots, i_z$ denote the remaining subactivities of activity i that can finish (if scheduled immediately) at time instants $t+1, t+2, \dots, t+z$, respectively. Let i_x be the first remaining subactivity of i that is not scheduled immediately in G (i.e., $f_{i_x} \neq t+x$). By hypothesis, no other subactivity y is scheduled concurrently with subactivity i_x between $f_{i_x}-1$ and f_{i_x}. Let J be the set of subactivities that are scheduled during the interval $(t+x-1, f_{i_x}-1]$. Construct schedule G' by interchanging subactivity i_x with the set J such that subactivity i_x finishes at time instant $t+x$ and the set J is scheduled during the interval $(t+x, f_{i_x}]$. By hypothesis, subactivity i_x was eligible at time instant $t+x-1$ so that no predecessor of i_x belongs to J. No resource constraint is violated as a pairwise interchange is performed under the assumption of constant resource availabilities. Hence, G' is feasible. Since the project length of G' is equal to that of G, schedule G' is also an optimal continuation of PS_t.

A new schedule G'' can now be constructed by repetitively interchanging the first remaining subactivity i_x that is not scheduled to complete at time $t+x$ with the set of subactivities that is scheduled during the interval $(t+x-1, f_{i_x}-1]$ until all remaining subactivities of activity i are scheduled immediately. This new schedule is feasible and has the same project length as schedule G. Thus, G'' is an optimal continuation of the partial schedule PS_t. □

Proof of Theorem 3. Let schedule G be an optimal continuation of the partial schedule PS_t with finish times f_y for all subactivities y. Let activities i and j satisfy the hypotheses of the theorem. If no such i and j exist, we are done.

Assume that at time t activity i has z remaining subactivities and activity j has z' remaining subactivities. Let $z'' = \min\{z, z'\}$ be the number of subactivities of activity j that, according to Theorem 3, should be scheduled concurrently with the z remaining subactivities of activity i. If all z remaining subactivities of i are scheduled immediately and if the first z'' subactivities of j are scheduled immediately, we are done.

Otherwise, the proof will consist of two main parts. First we will prove that there exists a schedule G' which is an optimal continuation of PS_t and in which z'' subactivities of both i and j are scheduled immediately. Afterwards we will only have to prove that there exists a schedule G'' in which the remaining $z-z''$ subactivities of i are scheduled immediately from time instant $t+z''$ on and that this schedule G'' is also an optimal continuation of PS_t.

Let $i_1, i_2, \dots, i_z$ denote the remaining subactivities of activity i that can finish (if scheduled immediately) at time instants $t+1, t+2, \dots, t+z$, respectively. Let $j_1, j_2, \dots, j_{z''}$ denote the remaining subactivities of activity j that can finish (if scheduled immediately) at time instants $t+1, t+2, \dots, t+z''$, respectively. Let $t+x \leqslant t+z''$ be the first time instant at which not both subactivities i_x and j_x finish in G (i.e., $f_{i_x} \neq t+x$ and/or $f_{j_x} \neq t+x$). If no such x exists, we can proceed with the second part of this proof.

Otherwise, we proceed as follows: as activity i could only be scheduled concurrently with activity j at any time instant greater than or equal to t, we have to consider two separate possibilities:

(a) Subactivity i_x is scheduled by itself in G to finish at time f_{i_x}. Let J be the set of subactivities that are scheduled during the interval $(t+x-1, f_{i_x}-1]$. Construct schedule Q by interchanging subactivity i_x with the set J such that i_x finishes at $t+x$ and the set J is scheduled during the interval $(t+x, f_{i_x}]$. By hypothesis, i_x was eligible at time instant $t+x-1$ so that no predecessor of i_x belongs to J. No resource constraint is violated as a pairwise interchange is performed under the assumption of constant resource availabilities. Hence, Q is feasible and has the same project length as G. Now we construct schedule Q' from Q by left-shifting subactivity j_x to finish at time instant $t+x$. By hypothesis, subactivity j_x was eligible at time instant $t+x-1$ and no resource constraint is violated as subactivity j_x can be scheduled concurrently with subactivity i_x. Hence, Q' is a feasible schedule. Since the project length of Q' is equal to that of Q, it is also optimal.

(b) Subactivity i_x is scheduled concurrently with subactivity j_y ($y \geqslant x$). Let J be the set of subactivities that are scheduled during the interval $(t+x-1, f_{i_x}-1]$. Construct schedule Q' by interchanging subactivities i_x and j_y with the set J such that i_x and j_y finish at time instant $t+x$ and J is scheduled during the interval $(t+x, f_{i_x}]$. By hypothesis, i_x was eligible at time instant $t+x-1$ so that no predecessor of i_x belongs to J. However, if $y > x$, one or more predecessors of j_y (all of them subactivities of activity j) belong to J. Simply renumbering the subactivities of activity j such that these are again in the correct sequence solves this precedence violation. No resource constraint is violated as a pairwise interchange is performed under the assumption of constant resource availabilities. Since the project length of Q' is equal to that of G, it is also optimal.

A new schedule G' can be constructed by each time repeating one of these possibilities until the first z'' remaining subactivities of both i and j are scheduled immediately. Since this schedule G' is feasible and has the same project length as schedule G, it is also an optimal continuation of PS_t. Hence, the first part of the proof is completed.

For the second part of the proof we have to consider two possibilities:

(a) If $z = z''$ (i.e., at time instant t there were at least as many remaining subactivities of activity j as there were remaining subactivities of activity i), we have scheduled all remaining subactivities of activity i during the first part of the proof. Thus, schedule G' already satisfies the conclusion of the theorem and G'' is simply a copy of G'.

(b) If $z > z''$ (i.e., at time instant t there were fewer remaining subactivities of activity j than there were remaining subactivities of activity i), we can apply Theorem 2 at time instant $t+z''$. Indeed, at that time instant activity i is eligible and cannot be scheduled together with any other unfinished activity (activity j has finished). Thus, we can construct an optimal schedule G'' in which all subactivities of activity i that remain unscheduled at time $t+z''$ are scheduled immediately, as indicated in the proof of Theorem 2.

Hence, in all cases we can construct a feasible schedule G'' which has the same project length as schedule G. Thus G'' is also an optimal continuation of the partial schedule PS_t. □

Proof of Theorem 4. Consider an optimal schedule that is constructed by at least one non-minimal delaying alternative. Then, by definition of a minimal delaying alternative, at least one subactivity could be removed from that delaying alternative and thus be scheduled earlier on in the project schedule. Hence, the schedule is not a semi-active schedule. By Theorem 1 we know that there exists an optimal schedule in the set of semi-active schedules and this set now can be constructed by using minimal delaying alternatives only because of the fact that all subactivities have a duration of 1. □

References

[1] Davis, E.W., "Project scheduling under resource constraints: Historical review and categorisation of procedures", *AIIE Transactions* 5/4 (1973) 297–313.

[2] Davis, E.W., and Heidorn, G.E., "An algorithm for optimal project scheduling under multiple resource constraints", *Management Science* 17/12 (1971) 803–816.

[3] Demeulemeester, E.L., "Optimal algorithms for various classes of multiple resource-constrained project scheduling problems", Unpublished Phd Dissertation, Katholieke Universiteit Leuven, Belgium (1992).

[4] Demeulemeester, E.L., and Herroelen, W.S., "A branch-and-bound procedure for the multiple resource-constrained

project scheduling problem", *Management Science* 38/12 (1992) 1803–1818.

[5] Demeulemeester, E.L., and Herroelen, W.S., "A branch-and-bound procedure for the generalized resource-constrained project scheduling problem", Research report nr 9206, Departement Toegepaste Economische Wetenschappen, Katholieke Universiteit Leuven, Belgium (1992).

[6] French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop*, Ellis Horwood (1982).

[7] Herroelen, W.S., "Resource-constrained project scheduling – The state of the art", *Operational Research Quarterly* 23 (1972) 261–275.

[8] Herroelen, W.S., and Demeulemeester, E.L., "Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems", Paper presented at the Summer School on Scheduling Theory and Its Applications, Château de Bonas, France, September 28–October 2, 1992.

[9] Kaplan, L.A., "Resource-constrained project scheduling with preemption of jobs", Unpublished Phd Dissertation, University of Michigan (1988).

[10] Kaplan, L.A., "Resource-constrained project scheduling with setup times", Unpublished paper, Department of Management, University of Tenessee, Knoxville (1991).

[11] Patterson, J.H., "A comparison of exact procedures for solving the multiple constrained resource project scheduling problem", *Management Science* 30/7 (1984) 854–867.

[12] Patterson, J.H., and Huber, W.D., "A horizon-varying zero–one approach to project scheduling", *Management Science* 20/6 (1974) 990–998.

[13] Patterson, J.H., and Roth, G.W., "Scheduling a project under multiple resource constraints: A zero–one programming approach", *AIIE Transactions* 8/4 (1976) 449–456.

[14] Pritsker, A.B., Watters, L.J., and Wolfe, P.M., "Multiproject scheduling with limited resources: A zero–one programming approach", *Management Science* 16/1 (1969) 93–108.

[15] Simpson, W.P., "A parallel exact solution procedure for the resource-constrained project scheduling problem", Unpublished Phd Dissertation, Indiana University (1991).

[16] Slowinski, R., "Two approaches to problems of resource allocation among project activities – A comparative study", *Journal of the Operational Research Society* 31/8 (1980) 711–723.

[17] Weglarz, J., "Project scheduling with continuously-divisible, doubly constrained resources", *Management Science* 27/9 (1981) 1040–1053.

[18] Wiest, J.D., "Some properties of schedules for large projects with limited resources", *Operations Research* 12/3 (1964) 395–418.