MANAGEMENT SCIENCE
Vol. 38, No. 12, December 1992
Printed in U.S.A.

A BRANCH-AND-BOUND PROCEDURE FOR THE MULTIPLE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM*

ERIK DEMEULEMEESTER AND WILLY HERROELEN
Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Dekenstraat 2, B-3000 *Leuven, Belgium*

In this paper a branch-and-bound procedure is described for scheduling the activities of a project of the PERT/CPM variety subject to precedence and resource constraints where the objective is to minimize 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. Precedence and resource-based bounds described in the paper are combined with new dominance pruning rules to rapidly fathom major portions of the solution tree. The procedure is programmed in the C language for use on both a mainframe and a personal computer. The procedure has been validated using a standard set of test problems with between 7 and 50 activities requiring up to three resource types each. Computational experience on a personal computer indicates that the procedure is 11.6 times faster than the most rapid solution procedure reported in the literature while requiring less computer storage. Moreover, problems requiring large amounts of computer time using existing approaches for solving this problem type are rapidly solved with our procedure using the dominance rules described, resulting in a significant reduction in the variability in solution times as well.
(PROJECT MANAGEMENT—RESOURCE CONSTRAINTS; PROGRAMMING—BRANCH-AND-BOUND; NETWORKS/GRAPHS—APPLICATIONS)

1. Introduction

The specific resource allocation problem addressed in this paper is the multiple resource-constrained single-project scheduling problem, in which it is assumed that an activity is subject to technological precedence constraints (an activity can only be started if all its predecessor activities have been finished) and cannot be interrupted once begun (no job preemption allowed). Resources are assumed to be available per period in constant amounts and are also demanded by an activity in constant amounts throughout the duration of the activity. The objective is to schedule the activities subject to the precedence and resource constraints in order to minimize the total project duration.

Reviews of this classical resource-constrained project scheduling problem can be found in Herroelen (1972a, b) and Davis (1966, 1973). Comparisons of optimal and suboptimal procedures for the resource-constrained single-project scheduling problem can be found in Cooper (1976), Davis and Patterson (1975), Patterson (1984), Alvarez-Valdés and Tamarit (1989). The multi-project scheduling problem has been dealt with by Kurtulus and Davis (1982) and Kurtulus and Narula (1985). Early attempts to solve the problem concentrated in two areas: the formulation and solution of the problem as a mathematical programming problem and the development of heuristic solution procedures for obtaining satisfying solutions. Early attempts at using integer programming to solve the exact version of the problem were unsuccessful. Consequently, numerous enumerative (branch-and-bound) procedures for solving certain variants of the problem optimally were developed (Balas 1970, Bell and Park 1990, Christofides et al. 1987, Davis and Heidorn 1971, Elmaghraby and Herroelen 1977, Fisher 1973, Gorenstein 1972, Johnson 1967, Patterson and Huber 1974, Patterson and Roth 1976, Pritsker et al. 1969, Schrage 1970, Stinson

* Accepted by L. Joseph Thomas; received August 6, 1990. This paper has been with the authors 5 months for 3 revisions.

0025-1909/92/3812/1803$01.25

et al. 1978, Talbot and Patterson 1978). It should be mentioned that the resource-constrained project scheduling problem is a generalization of the well-known job-shop scheduling problem and as such is NP-complete (Blazewicz et al. 1983).

Computational results obtained by Patterson (1984) on a problem set of 110 test problems seem to indicate that Talbot's solution procedure (Talbot and Patterson 1978) is an effective problem solving technique whenever resource constrainedness in a problem is low and would likely be the preferred solution approach where computer storage is a particularly limiting factor (Patterson 1984). The branch-and-bound solution procedure of Stinson (Stinson et al. 1978) was found to be the fastest in those instances in which computer memory is not limiting. Talbot's implicit enumeration procedure consists of a systematic enumeration of all possible activity finish times (i.e., candidate problems) with the order of candidate problem selection determined before execution of the algorithm. Fathoming rules are employed to eliminate from explicit consideration possible activity finish times that cannot possibly lead to improved schedules. Noteworthy in this regard is the concept of a network cut to eliminate from explicit consideration possible inferior activity completion times earlier in the enumeration phase of the algorithm. Stinson developed a best-first branch-and-bound procedure in which nodes in the solution tree correspond to precedence and resource feasible assignments for a subset of the activities of a project. The order in which the candidate problems are considered is determined during the execution of the algorithm. Partial solutions are considered next for further evaluation based upon a four-element decision vector. Candidate problems can be eliminated from consideration by left-shift dominance pruning and by lower bound pruning.

Computational results obtained by Christofides et al. (1987) on a different set of test problems indicate that their branch-and-bound procedure outperforms Stinson's procedure, at least for projects up to 25 activities and three resource types. Their algorithm is based on the idea of using disjunctive arcs for resolving conflicts that are created whenever sets of activities have to be scheduled whose total resource requirements exceed the resource availabilities in some periods.

The purpose of this paper is to describe a new effective branch-and-bound procedure for the multiple resource-constrained single-project scheduling problem. The underlying theory and the optimal solution procedure are described in the next section. §3 focuses on computational results obtained by the branch-and-bound procedure on the well-known set of 110 test problems assembled by Patterson (1984). The last section is reserved for overall conclusions.

2. A Branch-and-Bound Procedure

The single-project, multiple resource-constrained project scheduling problem can be formulated as follows (for similar notation, see Christofides et al. 1987):

$$\min f_n \tag{1}$$

subject to

$$f_j - f_i \geq d_j, \qquad (i, j) \in H, \tag{2}$$

$$\sum_{S_t} r_{ik} \leq b_k, \qquad t = 1, 2, \ldots, f_n, \tag{3}$$

$$k = 1, 2, \ldots, K, \qquad \text{where}$$

f_i = finish time of activity i, $i = 1, 2, \ldots, n$,
H = set of pairs of activities indicating precedence constraints,
d_i = processing time of activity i,

r_{ik} = amount of resource type k required by activity i,
S_t = set of activities in process in time interval $]t-1, t] = \{i \mid f_i - d_i < t \leq f_i\}$,
b_k = total availability of resource type k.

It is assumed that activity i has a fixed processing time d_i (setup times are negligible or are included in the processing time). We further assume activity-on-the-node networks where activities 1 and n are dummy activities indicating the single start and end nodes of a project, respectively. The resource requirements r_{ik} are known constants over the processing interval of the activity. The availability of resource type k, b_k, is also a known constant throughout the project duration interval.

The precedence constraints given by equation (2) indicate that an activity j can only be started if all predecessor activities i are completed. Once started, activities run to completion (nonpreemption condition). The resource constraints given in equation (3) indicate that for each time period $]t-1, t]$ and for each resource type k, the resource amounts required by the activities in progress cannot exceed the resource availability. The objective function is given as equation (1). The project duration is minimized by minimizing the finish time of the unique dummy ending activity n.

2.1. *The Search Process*

The nodes in the branch-and-bound search tree correspond with partial schedules in which finish times temporarily have been assigned to a subset of the activities of the project. The partial schedules are feasible, satisfying both precedence and resource constraints. *Partial schedules* PS_m are only considered at those time instants m which correspond with the completion time of one or more project activities. A similar time incrementing scheme has been used by Johnson (1967), Stinson et al. (1978) and Christofides et al. (1987) for the case in which resource availabilities b_k are assumed to be constant throughout the duration of a schedule. Since our algorithm minimizes the length of the schedule, the partial schedules are constructed by semiactive timetabling (Bellman et al. 1982, French 1982). In other words, each activity is started as soon as it can within the precedence and resource constraints. A *partial schedule* PS_m at time instant m thus consists of the set of temporarily *scheduled activities*. Scheduling decisions are "temporary" in the sense that temporarily scheduled activities may be delayed as a result of decisions made at later stages in the search process. *Unscheduled activities* at time m are those for which the temporary scheduling decisions have not yet been made. At time m the corresponding partial schedule PS_m will contain some activities which have been finished and others which are still in progress. The former activities have finish times smaller than or equal to m and are placed in the *set* F_m *of finished activities* at time instant m. The latter activities belong to the *set* S_m *of activities in progress* at time instant m. The *set of unfinished activities* U_m at time instant m then contains all activities that do not belong to F_m.

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 (including the empty set) until a complete feasible schedule is obtained. In this sense, a complete schedule is a *continuation of a partial schedule*.

At every time instant m we define the *eligible set* E_m as the set of activities which are not in the partial schedule and whose predecessor activities have finished. These eligible activities can start at time m if the resource constraints are not violated. The next two theorems describe two special cases explored by the algorithm in order to put eligible activities in progress.

THEOREM 1. *If at time m the partial schedule* PS_m *has no activity in progress and an eligible activity i cannot be scheduled together with any other unscheduled activity at any time instant* $m' \geq m$ *without violating the precedence or resource constraints, then there*

exists an optimal continuation of the partial schedule with the eligible activity i put in progress (started) at time m.

PROOF. See Appendix 1.

At each decision point m, this theorem allows the procedure to put in progress the eligible activity which must be scheduled by itself due to precedence and resource constraints.

THEOREM 2. *If at time m the partial schedule PS_m has no activity in progress, if there is an eligible activity i which can be scheduled concurrently with only one other unscheduled activity j at any time instant $m' \geq m$ without violating precedence or resource constraints, and if activity j is both eligible and not longer in duration than activity i, then there exists an optimal continuation of the partial schedule in which both activities i and j are put in progress at time m.*

PROOF. See Appendix 1.

At each decision point m, this theorem allows the procedure to identify the two eligible activities which must be scheduled concurrently.

If it is impossible to schedule all eligible activities at time m, a so-called *resource conflict* occurs (a similar concept is described in Bellman et al. 1982 and Christofides et al. 1987). Such a conflict will produce a new *branching* in the branch-and-bound solution tree. The branches describe ways to resolve the resource conflict; i.e., decisions about which combinations of activities are to be delayed. We define a *delaying set* $D(p)$, which consists of all subsets of activities D_q, either in process or eligible, 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 theorem allows the procedure to delete from explicit consideration the redundant delaying alternatives. The delay of a subset of activities $D_q \in D(p)$ is introduced in our approach by adding extra precedence relations that force them to wait until the completion of some other activities. For every delaying alternative $D_q \in D(p)$ the *set of extra precedence relations* $G_q = \{(j, i)\}$ is constructed by taking as the predecessor activity for every activity $i \in D_q \in D(p)$ the earliest finishing activity j, that is either in progress or eligible to start at time m and that is not delayed (ties are broken arbitrarily).

THEOREM 3. *In order to resolve a resource conflict, it is sufficient to consider only minimal delaying alternatives.*

PROOF. See Appendix 1.

Each delaying alternative D_q will be evaluated by computing a *critical sequence lower bound* L_q as defined by Stinson et al. (1978). This bound, which simultaneously considers both precedence and resource constraints, is computed as follows. Consider the delaying alternative D_q for which we wish to determine the critical sequence lower bound. Based on the added precedence relations G_q, we determine the corresponding partial schedule PS' (i.e., $PS_m + E_m - D_q$) and perform the precedence-based critical path computations for the network. Consider a critical path (ties are broken arbitrarily) with length z. All unscheduled activities $i \notin PS'$ which are not on this path are inserted into a set NC. The computation of their early start times es_i and late finish times lf_i is based solely upon precedence constraints. If there exists an unscheduled activity $i \in NC$ of duration d_i and if sufficient resources of one or more types are only available to allocate to activity i continuously for $e_i \leq d_i$ time periods somewhere in the interval from es_i to lf_i, then the project completion will be delayed by a minimum of $d_i - e_i$ time periods. As shown by Stinson et al. (1978) the critical sequence lower bound can then be computed as

$$L_q = \max_{i \in NC} (z + d_i - e_i).$$

Branching continues from the node corresponding with the delaying alternative D_q^* with the smallest L_q^* (ties are broken arbitrarily).

Contrary to the view that tree pruning by schedule dominance is not possible with depth-first procedures (Stinson et al. 1978 and Moder, Phillips, and Davis 1983), two *dominance rules* are used to prune the search tree. The first rule applies whenever it can be shown that an activity can be left-shifted and the resulting partial schedule is both precedence and resource feasible. This *left-shift dominance rule* is conceptually identical to the one used by Schrage (1970), Herroelen (1972a, b) and Stinson et al. (1978). It is implemented here as follows. Define the set $DS = \{j \in D_q^* | f_j < m + d_j\}$ as the set of activities which were started earlier than m but have to be delayed when we select the delaying alternative D_q^*. If DS is not empty, the left-shift dominance rule is invoked by using the following selection structure. If the precedence relationships which were added at previous levels of the search tree forced activity i to become eligible at time m, if the current decision was to start that activity at time m and if delaying activity set DS would allow activity i to be left-shifted without causing a resource conflict, then the corresponding partial schedule is dominated.

The second dominance rule is based on the concept of a cutset. At every time instant m a *cutset* C_m is defined as the set of all unscheduled activities for which all predecessor activities belong to the partial schedule PS_m.

THEOREM 4. *Consider a cutset C_m at time m which contains the same activities as a cutset C_k, which was previously saved during the search of* ***another*** *path in the search tree. If time k was not greater than time m and if all activities in progress at time k did not finish later than the maximum of m and the finish time of the corresponding activities in PS_m, then the current partial schedule PS_m is dominated.*

PROOF. See Appendix 1.

Backtracking occurs when a schedule is completed or a branch is to be fathomed by the lower bound calculation and/or dominance rules. At the backtrack the added arc(s) corresponding to the last delaying set element $D_q^* \in D(p)$ is (are) removed and new arcs are added for the next one at the same level. If there is no delaying alternative D_q left unexplored at this level, we backtrack to the previous one. When level zero is reached in the search tree, the search process is completed and the optimal solution has been found and has been verified.

It should be clear from the description so far that the branch-and-bound procedure is based on a search process of the *depth-first search type*. As such it is to be considered as a major extension of the depth-first search process proposed by Christofides et al. (1987). We share with them the time incrementing scheme by making partial schedule decisions at time instants m corresponding to the completion time of one or more project activities. Similar to theirs our procedure will only branch in order to resolve a resource conflict. However, the generation of the branches and the selection of the node from which to continue the search are quite different. They sort the set of eligible activities in decreasing order of the remaining critical path length and consider each activity in that order. The activity is either put in progress or, if it cannot be scheduled within the resource constraints, it is entered into a conflict set. For such a conflict activity i the set of delaying alternatives $A(i)$ consists of the possible minimal combinations of activities, the delay of which would allow the conflict activity to be scheduled. They determine how to delay conflict activity i using the earliest finishing activity in the set $A(i)$ as a predecessor. This leads to a corresponding new node in the solutions tree. The remaining branching alternatives are generated by taking for each alternative A in the set $A(i)$ precisely that activity a^* which has the smallest finishing time and which does not belong to alternative A. They do not evaluate these branching alternatives, but before proceeding with the next conflict activity they attempt to fathom the new node by a precedence-based bound.

The search procedure described in this paper is also depth-first. The set of eligible activities, however, is not sorted. Instead we construct the delaying set $D(p)$ which contains *all* the combinations of activities (eligible or in progress) which satisfy the following: (i) their delay releases enough resources to resolve the resource conflict, and (ii) the combinations are minimal, i.e., a combination may not contain other combinations as a subset. The generation of this set of minimal delaying alternatives must be exhaustive in order to guarantee optimality. The branching *alternatives are evaluated* in our approach by computing a critical sequence lower bound. In this sense, our depth-first procedure works harder initially in order to obtain an improved feasible starting schedule. In combination with the two dominance rules used, this allows us to fathom a relatively large number of nodes during backtracking.

2.2. *The Branch-and-Bound Algorithm*

The detailed algorithmic steps of the branch-and-bound algorithm are described below. For simplicity of notation the subscripts m for the sets PS, S, E and C are omitted. The *save* operation performed in Step 2 below should be distinguished from the *store* operation performed in Step 5. The *save* operation saves cutset information needed in order to apply the cutset dominance rule. The *store* operation saves information which is restored during backtracking.

Step 1. Let $T = 9999$ be an upper bound on the project duration. Set the level of the branch-and-bound tree $p = 0$. Initialize $m = 0$. For every activity i compute the remaining critical path length $RCPL_i$. Initialize the activity completion times $f_i = 9999$. Schedule the dummy start activity, i.e., set $f_1 = 0$, update the partial schedule $PS = \{1\}$ and the set of activities in progress $S = \{1\}$. Compute the lower bound as $LB(0) = RCPL_1$. Update the cutset: $C = \{x \mid x$ has activity 1 as a single predecessor$\}$ and set the early start times of the cutset activities s_x equal to zero.

Step 2. Compute the next decision point m as the earliest completion time of all activities in progress: $m = \min\{f_i, i \in S\}$. For all activities $j \in S$ for which $f_j = m$ update the set of activities in progress: $S = S - \{j\}$.

If the last scheduled activity is the dummy ending activity n, the schedule is completed. Update the schedule length $T = f_n$. If T is equal to $LB(0)$, then stop (with the optimal solution), else go to Step 7 (backtrack).

Check if the current cutset C is dominated by a previously saved cutset. If so, go to Step 7 (backtrack), else *save* the current cutset C, *save* the set S of activities in progress together with their finishing times f_j and *save* the current decision point m.

Construct the set of eligible activities: $E = \varnothing$ and for each activity $i \in C$ with $s_i = m$ update the eligible set $E = E \cup \{i\}$. If there are no eligible activities (i.e., if $E = \varnothing$), go to Step 2. If there are still some activities in progress, go to Step 4; else continue with Step 3.

Step 3. For each eligible activity $i \in E$ count the number of unscheduled activities j (not necessarily elements of E) which can be processed simultaneously with activity i without violating the precedence and resource constraints. If none can be ongoing concurrently with the eligible activity i, then put the eligible activity in progress: $PS = PS \cup \{i\}$, $S = \{i\}$, $f_i = m + d_i$ and update the cutset: $C = C - \{i\} + \{x \mid x$ is an immediate successor of i with all its predecessors in $PS\}$. For all cutset activities $x \in C$ update the early start times $s_x = f_i$.

If the eligible activity i can be scheduled concurrently with only one other activity j which is eligible and does not have a larger duration than activity i, put both activities i and j in progress and update the cutset: $C = C - \{i, j\} + \{x \mid x$ is an immediate successor of i or j with all its predecessors in the partial schedule$\}$. For all $x \in C$ update the early start times $s_x = f_i$.

If an eligible activity was scheduled during this step, go to Step 2, else go to Step 4.

Step 4. *Temporarily* put all eligible activities in progress: $PS = PS \cup E$, $S = S \cup E$,

set $f_i = m + d_i$ for all $i \in E$. Update the cutset: $C = C - E + \{x | x$ is an immediate successor of $i \in E$ with all its predecessors in $PS\}$ and set the early start times of the cutset activities as follows: $s_x = \max\{f_a | (a, x) \in H\}$.

For each resource type k check if $\sum_{i \in S} r_{ik} \leq b_k$. If there is at least one resource type k for which the sum of the resource requirements of all activities in progress exceeds the resource availability, we have a resource conflict: go to Step 5, else go to Step 2.

Step 5. Update the branch level in the search tree: $p = p + 1$. Determine for each resource type k how many units have to be freed to resolve the resource conflict, i.e., for each k set $c_k = \sum_{i \in S} r_{ik} - b_k$. Define the delaying set $D(p) = \{D_q \subset S | \sum_{i \in D_q} r_{ik} \geq c_k$ for all k and D_q does not contain other $D_v \in D(p)$ as a subset$\}$.

For every $D_q \in D(p)$ determine the earliest finishing task j, that is in progress and that is not delayed (i.e., $j \in (S - D_q)$). Define the corresponding set of tuples $G_q = \{(j, i)\}$ for all $i \in D_q$. Compute for each delaying alternative D_q the critical sequence bound L_q.

Select the $D_q^* \in D(p)$ with the smallest L_q^* (ties are broken arbitrarily). Update the delaying set: $D(p) = D(p) - D_q^*$. Set $LB(p) = \max\{LB(p-1), L_q^*\}$. If $LB(p) \geq T$, go to Step 7. Otherwise *store* the activity completion times, the partial schedule, the set of activities in progress, the cutset activities with their early start times and the decision point m.

Step 6. Branch into a new node. Define DS as the set of all activities that started earlier than m but must be delayed: $DS = \{i \in D_q^* | f_i < m + d_i\}$. Add the extra precedence relations at this level: $H = H \cup G_q^*$. Update: $PS = PS - D_q^*$, $S = S - D_q^*$. For all $i \in D_q^*$ set f_i equal to 9999. Update the cutset: $C = C + D_q^* - \{r | x \in D_q^*$ and $(x, r) \in H\}$. For $i \in D_q^*$ set the early start times: $s_i = f_j | (j, i) \in G_q^*$.

If DS is not empty, invoke the left-shift dominance rule as follows. If the precedence relations which were added at previous levels of the search tree forced an activity i to become eligible at time m, if the current decision was to start that activity at time m and if delaying activity set DS would allow this task i to be left-shifted without causing a resource conflict, then this schedule is dominated: go to Step 7 (backtrack); else, go to Step 2.

Step 7. If the branching level $p = 0$, then STOP. Else, delete the extra precedence relations which were added at this branching level: $H = H - G_q^*$. If $D(p) = \varnothing$ set $p = p - 1$ and repeat Step 7.

Select the $D_q^* \in D(p)$ with the smallest L_q^*. Update the delaying set: $D(p) = D(p) - D_q^*$. Compute the lower bound $LB(p) = \max\{LB(p-1), L_q^*\}$. If $LB(p) \geq T$, decrease the branching level: $p = p - 1$ and repeat Step 7.

Restore the activity completion times, the partial schedule, the set of activities in progress, the cutset activities, the early start times of the cutset activities and the decision point m. Go to Step 6.

2.3. *Numerical Example*

Consider the activity-on-the-node network given in Figure 1. The numbers above each node denote the fixed activity durations. The numbers below each node denote the daily resource requirement for a single resource type. The resource has a constant availability of 6 units per day.

Initializing the branch-and-bound procedure, we set $T = 9999$ and find $RCPL_1 = 10$ as the critical path length. Initialize the level of the branch-and-bound tree and the decision point: $p = 0$, $m = 0$. Initialize all activity finish times $f_i = 9999$. Schedule the dummy activity 1: $f_1 = 0$. The partial schedule is updated as $PS = \{1\}$ and the set of activities in progress is $S = \{1\}$. The lower bound is computed as $LB(0) = 10$. The cutset is computed as $C = \{2, 3, 4, 5\}$ with $s_2 = s_3 = s_4 = s_5 = 0$. This corresponds to *node* 1 at level 0 of the search tree in Figure 2.

Proceeding with Step 2, the earliest completion time of all activities in progress is $m = 0$, the next decision point. Update $S = \varnothing$. Since the current cutset $C = \{2, 3, 4, 5\}$

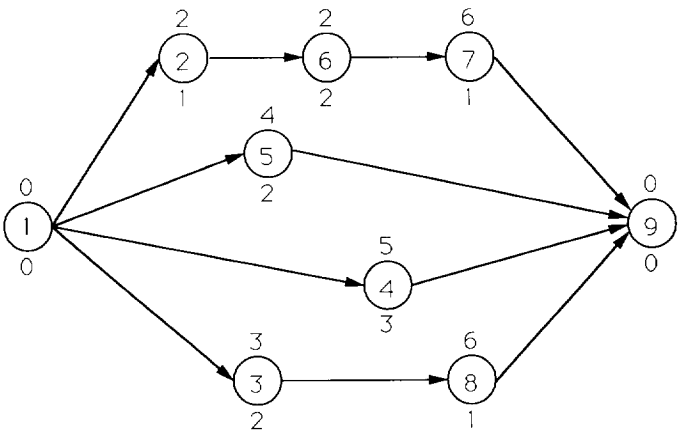


FIGURE 1. Precedence Diagram for the Problem Example.

is not dominated, it is saved. The set of eligible activities is computed as $E = \{2, 3, 4, 5\}$.

It is found in Step 3 that there is no activity in E that must be scheduled by itself (Theorem 1), nor is there an eligible activity that can be scheduled concurrently with

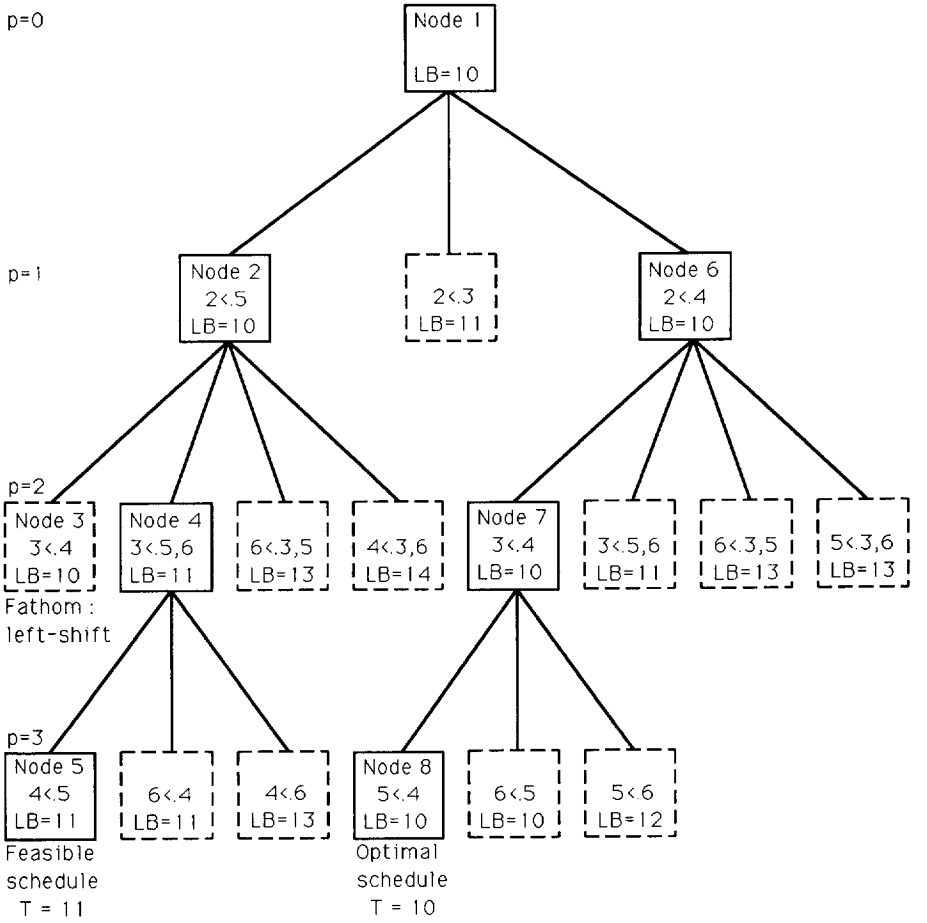


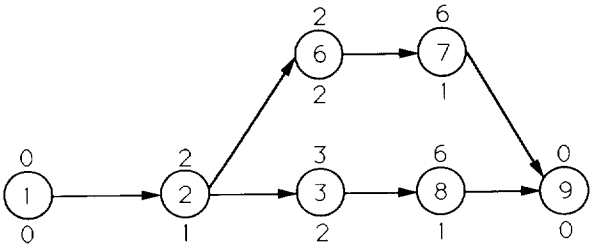
FIGURE 2. Branch-and-Bound Search Tree.

only one other activity that is both eligible and shorter in duration (Theorem 2). We proceed to Step 4. The eligible activities are *temporarily* put in progress: $PS = \{1, 2, 3, 4, 5\}$, $S = \{2, 3, 4, 5\}$, $f_2 = 2, f_3 = 3, f_4 = 5$ and $f_5 = 4$. Update the cutset $C = \{6, 8\}$, $s_6 = 2$, $s_8 = 3$.

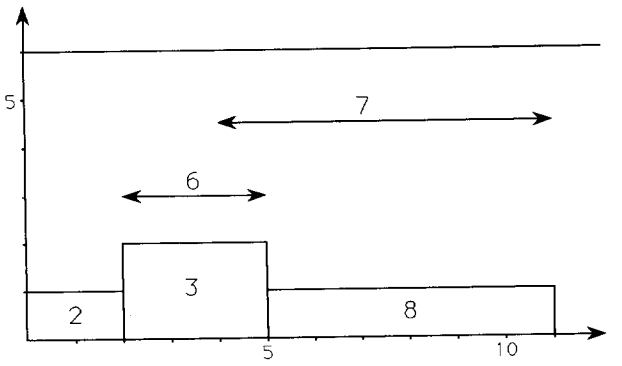
The resource requirement for type 1, summed over all activities in progress, is 8 units. A *resource conflict* occurs. Proceed with Step 5. Set $p = 1$ (level of the search tree in Figure 2) and compute the number of resource units to be released in order to resolve the resource conflict: $c_1 = 8 - 6 = 2$. The delaying set is $D(1) = \{\{3\}, \{4\}, \{5\}\}$. The delaying set is minimal. Alternative $\{2, 3\}$, for example, would be a valid delaying alternative, but is not included since its subset $\{3\}$ already belongs to the delaying set. For delaying alternative $D_1 = \{3\}$, activity 2 is found to be the earliest finishing activity that is not delayed. Set $G_1 = \{(2, 3)\}$ and compute the critical sequence bound L_1 as follows. Given the added precedence relation (2, 3), the corresponding partial schedule would be derived as $PS' = \{1, 2, 4, 5\}$. The critical path is the path 1-2-3-8-9 with a length of $z = 11$ time units. This yields the critical sequence 2-3-8 as indicated in Figure 3. Now consider the unscheduled activity 6 with $es_6 = 2$ and $lf_6 = 5$ and a daily resource requirement of 2 units. This activity can be scheduled continuously for a period $e_6 = d_6 = 2$ time units in the interval from 2 to 5. In a similar fashion the other unscheduled activity 7 with $es_7 = 4$ and $lf_7 = 11$ and a daily resource requirement of 1 unit can be scheduled continuously for a period of $e_7 = d_7 = 6$ time units. Hence, the critical sequence lower bound is computed for this delaying alternative as

$$L_1 = \max\{(z + d_6 - e_6), (z + d_7 - e_7)\} = \max\{(11 + 2 - 2), (11 + 6 - 6)\} = 11.$$

In a similar fashion we find for $D_2 = \{4\}$ the tuple $G_2 = \{(2, 4)\}$ with $L_2 = 10$. For delaying alternative $D_3 = \{5\}$, we find $G_3 = \{(2, 5)\}$ with $L_3 = 10$.



(a) Project Subnetwork.



(b) Unscheduled Activities 6 and 7 Can Be Scheduled.

FIGURE 3. Illustration of the Critical Sequence Bound.

Select $D_3^* = \{5\}$ with the smallest lower bound estimate $L_3^* = 10$ (arbitrary tie-break). Update the delaying set $D(1) = \{\{3\}, \{4\}\}$. Compute the lower bound as $LB(1) = \max\{LB(0), L_3^*\} = \max(10, 10) = 10$. Store the partial schedule $PS = \{1, 2, 3, 4, 5\}$, the set of activities in progress $S = \{2, 3, 4, 5\}$, the activity completion times $f_2 = 2, f_3 = 3, f_4 = 5$ and $f_5 = 4$, the cutset $C = \{6, 8\}$, the early start times of the cutset activities $s_6 = 2$, $s_8 = 3$ and the decision point $m = 0$.

Proceed with Step 6 and branch into *node* 2 of the search tree. The extra precedence relation (2, 5) is added. Update $PS = \{1, 2, 3, 4\}$, $S = \{2, 3, 4\}, f_5 = 9999$. Update the cutset: $C = \{5, 6, 8\}$ with $s_5 = 2$, $s_6 = 2$, $s_8 = 3$. As $DS = \varnothing$ the left-shift dominance rule does not apply and the procedure continues with Step 2.

The earliest completion time of all activities in progress is $m = 2$, the next decision point. Update the set of activities in progress $S = \{3, 4\}$. The current cutset is $C = \{5, 6, 8\}$. Since there is no identical previously saved cutset, the cutset dominance rule does not apply. Save the current cutset $C = \{5, 6, 8\}$. Save the set of activities in progress $S = \{3, 4\}$, the finish times $f_3 = 3, f_4 = 5$ and the decision point $m = 2$. The set of eligible activities is updated as $E = \{5, 6\}$. Activities 3 and 4 are still in progress, so we continue with Step 4.

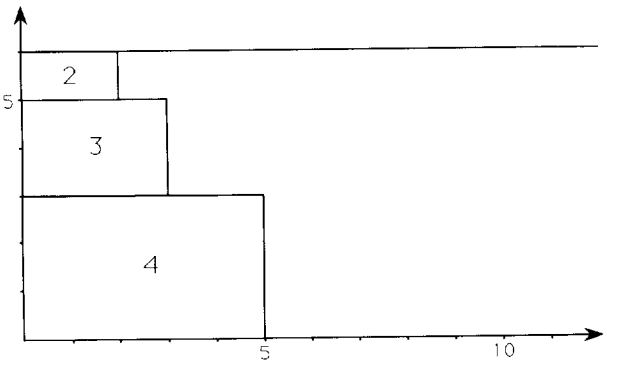
The eligible activities are *temporarily* put in progress: $S = \{3, 4, 5, 6\}, f_5 = 6, f_6 = 4$, $PS = \{1, 2, 3, 4, 5, 6\}$. Update the cutset: $C = \{7, 8\}$ and set the early start times for the cutset activities: $s_7 = 4$, $s_8 = 3$. The resource requirement for the single-resource type, summed over all activities in progress, is 9 units. A resource conflict occurs. Proceed with Step 5.

Set $p = 2$ (level of the search tree in Figure 2) and compute the number of resource units to be released in order to resolve the conflict: $c_1 = 9 - 6 = 3$. The delaying set is $D(2) = \{\{4\}, \{3, 5\}, \{3, 6\}, \{5, 6\}\}$. For delaying alternative $D_1 = \{4\}$ we find $G_1 = \{(3, 4)\}$ with $L_1 = 10$. For the second delaying alternative $D_2 = \{3, 5\}$, we find activity 6 as predecessor, yielding $G_2 = \{(6, 3), (6, 5)\}$ with $L_2 = 13$. In a similar fashion we find $G_3 = \{(4, 3), (4, 6)\}$ with $L_3 = 14$ and $G_4 = \{(3, 5), (3, 6)\}$ with $L_4 = 11$.

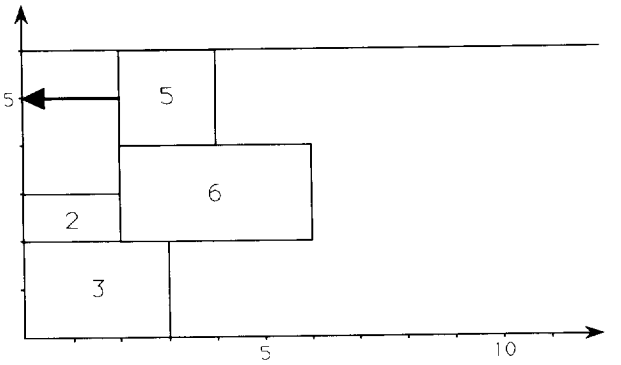
Select $D_1^* = \{4\}$ with the smallest lower bound estimate $L_1^* = 10$. Update the delaying set $D(2) = \{\{3, 5\}, \{3, 6\}, \{5, 6\}\}$. Compute the lower bound as $LB(2) = \max(10, 10) = 10 < T$. Store $S = \{3, 4, 5, 6\}, f_3 = 3, f_4 = 5, f_5 = 6, f_6 = 4$, $PS = \{1, 2, 3, 4, 5, 6\}$, $C = \{7, 8\}$, $s_7 = 4$, $s_8 = 3$ and the decision point $m = 2$. Proceed with Step 6 and branch into *node* 3 of the search tree. Update $PS = \{1, 2, 3, 5, 6\}$, $S = \{3, 5, 6\}$. Add the extra precedence relation (3, 4), $f_4 = 9999$. Update the cutset: $C = \{4, 7, 8\}$ with $s_4 = 3$, $s_7 = 4$ and $s_8 = 3$. Since $DS = \{4\}$ we try to invoke the *left-shift dominance rule*. Activity 5 was previously delayed by the extra precedence relation (2, 5) and became eligible again at time instant $m = 2$. The current decision is to start activity 5 at time 2. As indicated in Figure 4, the current decision to delay activity 4 which was in progress in the parent partial schedule, allows activity 5 to be left-shifted and start at time 0. *Node* 3 is *dominated* and we continue with Step 7 of the procedure (*backtrack*). The precedence relation (3, 4) is deleted. We select $D_4^* = \{5, 6\}$ from the delaying set $D(2) = \{\{3, 5\}, \{3, 6\}, \{5, 6\}\}$ because it has the smallest $L_4^* = 11$. Update the delaying set $D(2) = \{\{3, 5\}, \{3, 6\}\}$ and compute the lower bound as $LB(2) = \max(10, 11) = 11 < T$. Restore: $S = \{3, 4, 5, 6\}, f_3 = 3, f_4 = 5, f_5 = 6, f_6 = 4$, $PS = \{1, 2, 3, 4, 5, 6\}$, $C = \{7, 8\}$, $s_7 = 4$, $s_8 = 3$ and the decision point $m = 2$. Go to Step 6.

Branch into *node* 4 of the search tree. Update the partial schedule $PS = \{1, 2, 3, 4\}$ and the set of activities in progress $S = \{3, 4\}$. Add the extra precedence relations (3, 5) and (3, 6): $f_5 = f_6 = 9999$. Update the cutset: $C = \{5, 6, 8\}$ with $s_5 = 3$, $s_6 = 3$ and $s_8 = 3$. Since DS is empty, the left-shift dominance rule does not apply. Continue with Step 2.

Set $m = 3$ corresponding to the completion of activity 3. Update $S = \{4\}$. The current cutset $C = \{5, 6, 8\}$ is identical to the one already saved earlier at time $m = 2$. However, the dominance rule does not apply because this previously saved cutset was generated



(a) Partial Schedule $PS = \{1, 2, 3, 4\}$ for Node 2 of the Search Tree.



(b) Dominated Partial Schedule $PS = \{1, 2, 3, 5, 6\}$.

FIGURE 4. Illustration of the Left-Shift Dominance Rule.

at the parent node 2 on the *same path* of the search tree. The reason why both cutsets are identical is that the eligible activities at time $m = 2$ were delayed immediately. Save the cutset $C = \{5, 6, 8\}$, the set of activities in progress $S = \{4\}$ with $f_4 = 5$ and the decision point $m = 3$. The eligible set is now $E = \{5, 6, 8\}$. *Temporarily* put all eligible activities in progress: $PS = \{1, 2, 3, 4, 5, 6, 8\}$, $S = \{4, 5, 6, 8\}$ with $f_4 = 5, f_5 = 7$, $f_6 = 5$ and $f_8 = 9$. Update the cutset: $C = \{7\}$ with $s_7 = 5$. A resource conflict occurs again. Proceeding with Step 5, set $p = 3$ and generate $D(3) = \{\{4\}, \{5\}, \{6\}\}$. We find $G_1 = \{(6, 4)\}$ with $L_1 = 11$; $G_2 = \{(4, 5)\}$ with $L_2 = 11$ and $G_3 = \{(4, 6)\}$ with $L_3 = 13$. We select delaying alternative D_2^* with $L_2^* = 11$ and compute the lower bound as $LB(3) = \max(11, 11) = 11$. Store the set $S = \{4, 5, 6, 8\}$ with $f_4 = 5, f_5 = 7$, $f_6 = 5$ and $f_8 = 9$, the partial schedule $PS = \{1, 2, 3, 4, 5, 6, 8\}$, the cutset $C = \{7\}$ with $s_7 = 5$ and the decision point $m = 3$.

Branch into *node* 5 of the search tree. Update $PS = \{1, 2, 3, 4, 6, 8\}$, $S = \{4, 6, 8\}$. Add the extra precedence relation $(4, 5)$: $f_5 = 9999$. Update the cutset: $C = \{5, 7\}$ with $s_5 = 5$ and $s_7 = 5$. As $DS = \varnothing$ the left-shift dominance rule does not apply and we proceed with Step 2.

The next decision point now occurs at time $m = 5$, marked by the completion of activities 4 and 6. Update $S = \{8\}$. The current cutset is not dominated. Save the cutset $C = \{5, 7\}$ with $s_5 = 5$ and $s_7 = 5$ and the activities in progress $S = \{8\}$ with $f_8 = 9$. Activities 5 and 7 are eligible. Activities 5 and 7 are *temporarily* put into progress: $PS = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $S = \{5, 7, 8\}$ with $f_5 = 9, f_7 = 11$ and $f_8 = 9$. Update $C = \{9\}$ with $s_9 = 11$. No resource conflict occurs. The next decision point occurs at

time instant $m = 9$, marked by the completion of activities 5 and 8: $S = \{7\}$. The current cutset is not dominated: save $C = \{9\}$ with $s_9 = 11$, $S = \{7\}$ with $f_7 = 11$ and $m = 9$. No activities become eligible. The next decision point occurs at time $m = 11$ when activity 7 is completed: $S = \emptyset$. The current cutset is not dominated as there is only one identical cutset, which was obtained on the same path. Save $C = \{9\}$, $S = \emptyset$ and $m = 11$. $E = \{9\}$. Put activity 9 into progress: $PS = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $f_9 = 11$, $C = \emptyset$. We find a complete *feasible solution with* $T = 11$.

Continuing with Step 7, the precedence relation (4, 5) is deleted, and we select delaying alternative $D_1^* = \{4\}$ with $L_1^* = 11$. The corresponding lower bound $LB(3) = \max(11, 11) = 11 = T$. Fathom the node (the fathomed node corresponding with the precedence relation (6, 4) is indicated in dotted lines in Figure 2). Since we know for sure that the remaining delaying alternative D_3 cannot have a better lower bound estimate, the branching level is set to $p = 2$ (level 2 of the search tree) and we repeat Step 7.

The precedence relations (3, 5) and (3, 6) which were added in node 4 are deleted. Select $D_2^* = \{3, 5\}$ with $L_2^* = 13$. Compute the lower bound $LB(2) = \max(10, 13) = 13 > T$. Fathom the node. The remaining delaying alternatives cannot have a better lower bound estimate: set $p = 1$ and backtrack to level 1 of the search tree. Delete the precedence relation (2, 5) and select $D_2^* = \{4\}$ with $L_2^* = 10$. Compute the lower bound: $LB(1) = \max(10, 10) = 10 < T$. Restore and proceed with Step 6.

The procedure will further explore the search tree as indicated in Figure 2 reaching an optimal solution with schedule length of 10 time units and $f_1 = 0, f_2 = 2, f_3 = 3, f_4 = 9, f_5 = 4, f_6 = 4, f_7 = 10, f_8 = 9$ and $f_9 = 10$.

3. Computational Results

The branch-and-bound procedure has been programmed in Turbo C Version 2.0 for a personal computer IBM PS/2 (or compatibles) running under the DOS operating system. Implementing it on mainframe computers poses no problems. The 110 test problems assembled by Patterson (1984) were used to validate the procedure. These problems represent an accumulation of all multi-resource problems existing in the literature that are readily available and for which optimal solutions are known. The number of activities included in these test problems varies between 7 and 50 with the number of resource types required per activity varying between one and three. The majority of the projects (103) consist of activities which require the full complement of three different resource types for their performance. Computational results obtained by Patterson (1984) indicate that the only procedure capable of solving all 110 test problems within a 5-minute CPU time limit per problem (on an Amdahl 470/V8 mainframe) is the branch-and-bound procedure of Stinson et al. (1978) which requires an average of 0.82 seconds. We translated Stinson's FORTRAN code into Turbo C in order to compare our solution procedures.

The results obtained with both Stinson et al.'s and our procedure on a personal computer IBM PS/2 Model 70 A21 with a 25 MHz processor and coprocessor are given in Table 1. Using the number of activities as the basic attribute, the set of 110 test problems has been categorized into five subsets: 8 projects with less than 22 activities, 46 projects with 22 activities, 43 projects with 27 activities, 10 projects with 51 activities and 3 remaining projects with 23, 35, and 35 activities, respectively. For each problem category we list the mean and standard deviation of the CPU time in seconds (excluding input and output) needed by both procedures to find the optimal solution. Our branch-and-bound scheme has been tested in three different modes: without the left-shift and the cutset dominance rule (mode 1), with the left-shift dominance rule alone (mode 2), and with both left-shift and cutset dominance rule (mode 3).

As can be seen from this table, our branch-and-bound procedure in its basic mode (i.e., without left-shift and cutset dominance rule) was able to solve all 110 test problems

within an average CPU time of 2.548 seconds per problem with a standard deviation of 8.181 seconds. Introducing the left-shift dominance rule reduces the average CPU time to 1.345 seconds per problem with a standard deviation of 4.122 seconds. The full-fledged procedure with the left-shift dominance rule and the cutset dominance rule incorporated needs an average CPU time of only 0.215 seconds per problem with a standard deviation of only 0.314 seconds. This reduction in computation time clearly demonstrates the ability of the dominance rules to fathom inferior partial solutions. Stinson et al.'s code solved all 110 test problems in an average CPU time of 2.494 seconds with a standard deviation of 3.762 seconds. This indicates that on this problem set our procedure is on the average 11.6 times faster than Stinson et al.'s. Without the use of our cutset dominance rule (mode 2), our procedure already outperforms Stinson et al.'s by a factor close to 2.

Especially encouraging are the results obtained on the 51 activity problems for which our procedure outperforms Stinson et al.'s by a factor of 17. For the ten hardest problems for which our procedure in its third mode needed more than 0.5 seconds to reach the optimal solution, the impact of introducing the two dominance rules is most striking.

The very small CPU times needed by our procedure for obtaining a feasible solution which, upon backtracking, proves to be optimal, is a strong indication of the potential heuristic quality of the algorithm on those problems where it would not be able to run to completion within imposed CPU time limits.

TABLE 1

Computational Results Obtained by the Branch-and-Bound Procedure

Problem Type	Number of Problems	Demeulemeester & Herroelen Mode 1[b]	Demeulemeester & Herroelen Mode 2[c]	Demeulemeester & Herroelen Mode 3[d]	Stinson
<22[a]	8	0.008[e] 0.021[f]	0.014 0.026	0.022 0.031	0.074 0.040
22	46	1.104 1.904	0.644 0.929	0.145 0.166	1.240 0.918
27	43	4.809 12.545	2.421 6.332	0.325 0.427	3.944 5.155
51	10	2.234 3.878	1.385 2.056	0.263 0.282	4.459 4.207
23, 35, 35	3	0.097 0.064	0.093 0.058	0.057 0.055	1.743 1.653
Hardest[g]	10	19.278 21.088	9.122 11.182	0.988 0.486	9.458 8.009
Average	110	2.548 8.181	1.345 4.122	0.215 0.314	2.494 3.762

[a] Number of activities.
[b] Without left-shift and cutset dominance rule.
[c] With left-shift dominance rule alone.
[d] With left-shift and cutset dominance rule.
[e] Average CPU time in seconds.
[f] Standard deviation of the CPU time in seconds.
[g] Problems for which mode 3 needed a CPU time in excess of 0.5 seconds.

4. Conclusions

In this paper a new depth-first branch-and-bound algorithm was described for scheduling a project to minimize its total duration subject to technological precedence constraints and resource constraints. Computational experience gained with the algorithm on a personal computer was extremely promising. The 110 Patterson test problems could be solved optimally within an average CPU time of 0.215 seconds and a standard deviation of 0.314 seconds on an IBM PS/2 Model 70 A21 with a 25 MHz processor and coprocessor. Feasible solutions which are close to the optimum can be obtained in a very small amount of CPU time. As such it is on the average almost twelve times faster than the best-first procedure developed by Stinson et al. (1978), previously reported (Patterson 1984) to be the most effective and efficient on this problem set.

The small average computational requirements and the very small variance in computation time from problem to problem are largely due to the harmonious combination of the new depth-first search strategy and the strong bounding arguments and dominance rules. The principle of branching in order to resolve resource conflicts based on the exhaustive generation of minimal delaying alternatives which are evaluated using the critical sequence lower bound seems to be very effective in generating very good feasible starting solutions. In combination with the strong left-shift and cutset dominance rules it allows the branch-and-bound procedure to fathom a relatively large number of nodes in virtually no time during backtracking.[1]

[1] The authors are gratefully indebted to Professor James Patterson, Indiana University, for his critical comments on an earlier draft. Without his cooperation and assistance in providing us with the source data for the 110 test problems, this paper would not have been possible. The authors would also like to thank the Associate Editor and the anonymous referees for their help in improving the statement of the theorems and the rigour of their proofs.

Appendix 1

PROOF OF THEOREM 1. Let schedule G be an optimal continuation of partial schedule PS_m. Let time m be such that no activity is in progress and activity i satisfies the hypotheses of the theorem. If i begins at time m or no such m and i exist, we are done.

Otherwise, let activity i begin at time $m' > m$ and complete at $m'' = m' + d_i$. By hypothesis, no other activity is scheduled with activity i between m' and m''. Let J be the set of activities begun and completed in the interval $]m, m']$. Construct schedule G' by interchanging activity i with the set J so that i begins at time m and J finishes at time m''. By hypothesis, activity i was eligible at time m so that no predecessor of i belongs to J. Hence, G' is feasible. Since the completion time of G' is equal to that of G, schedule G' is also optimal. Hence, result.

PROOF OF THEOREM 2. Let schedule G be an optimal continuation of partial schedule PS_m. Let time m be such that no activity is in progress and activities i and j satisfy the hypotheses of the theorem. If i and j begin at time m or no such m, i and j exist, we are done.

Otherwise, let activity i begin at time $m' \geq m$ and complete at time $m'' = m' + d_i$. Then we have to consider the following two possibilities for the completion times of activity j:

(a) Let activity j finish at time $f_j \leq m'$. By hypothesis, no other activity is scheduled with activity i between m' and m''. Let J be the set of activities begun and completed in the interval $]m, m']$ with $j \in J$. Construct G' by interchanging activity i with the set J so that i begins at time m and J finishes at time m''. By hypothesis, activity i was eligible at time m so that no predecessor of i belongs to J. Hence, G' is feasible. Since the completion time of G' is precisely the same as that of G, schedule G' is also optimal. Construct G'' from G' by shifting activity j to start with activity i at time m. G'' is feasible since no resource constraint is violated (by hypothesis, j can be scheduled concurrently with i) and no precedence relation is violated (by hypothesis, j was eligible at time m). Since the completion time of G'' is equal to that of G', schedule G'' is also optimal.

(b) Let activity j finish at time $f_j > m'$. Construct G' by shifting activity j to start with activity i at time m'. G' is feasible since no resource constraint is violated (by hypothesis, j can be scheduled concurrently with i) and no precedence relation is violated (by hypothesis, all predecessors of j are completed before time m and no successor of j can start before time m''). Since the schedule length of G' equals that of G (only the completion time of activity j has changed, but it does not finish later than activity i), schedule G' is optimal. By hypothesis, in G' no other activity than i and j is scheduled between m' and m''. Let J be the set of activities begun and completed in the interval $]m, m']$ (the set J will be empty if activity i started at time m). Construct schedule

G'' by interchanging activities i and j with the set J such that activities i and j start at time m and J finishes at time m''. By hypothesis, activities i and j were eligible at time m so that no predecessor of i or j belongs to J. Hence, G'' is feasible. Since the schedule length of G'' equals that of G', schedule G'' is also optimal.

PROOF OF THEOREM 3. Let $(i_0, \ldots, i_p, \ldots, i_k)$ be the chain of nodes of the solutions tree that is leading to an optimal schedule G. With each such node i_p at level p of the solutions tree corresponds a delaying alternative D_p, a set of added precedence relations $G_p = \{(j, i)\}$ for all $i \in D_p$ and a set of precedence relations H_p. For node i_0 the set H_0 equals H, the original set of precedence constraints. For all other nodes i_p the set H_p is obtained by adding the extra precedence relations G_p to H_{p-1}. Let f_j^* denote the finish time of activity j in the optimal schedule G.

Let node i_q be the first node in the chain for which the delaying alternative D_q is nonminimal. If no such i_q exists, we are done.

Otherwise, a set of added precedence relations $G_q = \{(y, i)\}$ is associated with D_q as well as finish times f_j for all scheduled activities and early start times s_j for all unscheduled activities. For all $i \in D_q$: $s_i = f_y$ and for all other unscheduled activities the early start time is equal to the maximum of the earliest possible finish times of its predecessor activities.

By definition of a minimal delaying alternative, there exists at least one activity x that can be removed from D_q: $D'_q = D_q - \{x\}$ such that D'_q is a valid delaying alternative, occurring at node i'_q at level q of the search tree. Let f'_x denote the finish time of activity x if it is not delayed. Now two possibilities have to be considered:

(a) In case $f_y \le f'_x$, $G'_q = \{(y, i)\}$ for all $i \in D'_q$. As $D'_q = D_q - \{x\}$, G'_q is a subset of G_q and hence H'_q is a subset of H_q. As the schedule G is feasible to H_q, G is also feasible to H'_q and therefore can be found by continuing the branching process from i'_q. Hence, node i_q can be fathomed.

(b) If $f_y > f'_x$, we have $G'_q = \{(x, i)\}$ for all $i \in D'_q$. The early start times are updated: $s'_i = f'_x < f_y$ for all $i \in D'_q$ and the early start times of all other unscheduled activities j are dependent upon the earliest finish times of their predecessors with $s'_j \le s_j$ as no predecessor has a larger earliest finish time under H'_q than under H_q. At the next decision point $m = f'_x$ we have: $F'_m = F_m + \{x\}$ and $U'_m = U_m - \{x\}$. The following observations can be made: for all $i \in F_m$: $f'_i = f_i$, for all $j \in U'_m$: $s'_j \le s_j$ and f'_x is smaller than any feasible finish time for activity x under H_q. Now we can construct a new schedule G' with finish times g_i as follows: for all $i \in F'_m$: $g_i = f'_i$ and for all $j \in U'_m$: $g_j = f_j^*$. G' is feasible to H'_q and can therefore be found by continuing the branching process from i'_q. Since G' has the same schedule length as G, it is also optimal. Hence, node i_q can be fathomed.

Now we have proven that an optimal solution can be found if branching takes place from a delaying alternative that has one activity less than the first nonminimal delaying alternative, encountered at node i_q of the solutions tree. The process of removing an activity from this nonminimal delaying alternative can be continued until a minimal delaying alternative D''_q is obtained at node i''_q. If branching continues from i''_q, a new optimal schedule can be found along the chain $(i_0, \ldots, i_{q-1}, i''_q, i''_{q+1}, \ldots, i''_{k''})$. Along this chain all delaying alternatives at nodes $i_1, \ldots, i''_q$ are minimal by construction, while the delaying alternatives at nodes $i''_{q+1}, \ldots, i''_{k''}$ could be nonminimal. The procedure of removing activities from a nonminimal delaying alternative until a minimal delaying alternative is obtained can again be performed at that level where the first nonminimal delaying alternative occurred along the new chain. This action can then be repeated until all delaying alternatives are minimal. Hence, it is proven that optimal schedules can be obtained by considering only minimal delaying alternatives.

PROOF OF THEOREM 4. Let G be an optimal schedule with finish times f_i^* generated by continuing a partial schedule PS_m for which the cutset C_m satisfies the hypotheses of the theorem. Then there exists a cutset C_k which was previously generated on a different path in the search tree and which corresponds with a partial schedule PS_k. Moreover, $k \le m$ and for all $j \in PS_k$ we have $f_j^k \le \max\{m, f_j^m\}$ where f_j^k denotes the finish time of activity j in PS_k and f_j^m denotes the finish time of activity j in PS_m. As G is generated by continuing PS_m it holds for all activities $j \in PS_m$ that $f_j^m \le f_j^*$.

Let A be the set of all activities in the project and let J denote the set of activities j that, in the optimal schedule G, started later than at time instant m (i.e., for which $f_j^* - d_j > m$). Construct a new schedule G' as follows: for all activities $j \in J$ set $f'_j = f_j^*$ and for all activities $i \in (A - J)$ set $f'_i = f_i^k$. G' is a feasible schedule: no precedence relation is violated as, by hypothesis, for all predecessors i of activities $j \in C_m$ it holds that either $f'_i = f_i^k \le \max\{m, f_i^m\} \le \max\{m, f_i^*\}$ or $f'_i = f_i^*$ (if $i \in J$). In addition, no resource constraint is violated as PS_k and G were feasible and because for all activities $j \in PS_k$, it holds that $f_j^k \le \max\{m, f_j^m\} \le \max\{m, f_j^*\}$ while all activities $i \in J$ are only scheduled in G' after both m and the largest f_j^* of any of its predecessors j. G' has the same schedule length as G and thus G', which could be obtained by branching from the node at which PS_k was constructed, is also optimal. Therefore PS_m is dominated.

References

ALVAREZ-VALDES, R. AND J. M. TAMARIT, "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis. Part 1," Chapter 5 In *Advances in Project Scheduling*, Slowinski, R. and J. Weglarz (Eds.), Elsevier Science Publishers B.V., Amsterdam, 1989.

BALAS, E., "Project Scheduling with Resource Constraints," In *Applications of Mathematical Programming Techniques*, E. M. L. Beale (Ed.), American Elsevier, New York, 1970.

BELL, C. E. AND K. PARK, "Solving Resource-Constrained Project Scheduling Problems by A^* Search," *Naval Res. Logist. Quart.*, 37, 1 (1990), 61–84.

BELLMAN, R., A. O. ESOGBUE AND I. NABESHIMA, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, Oxford, 1982.

BLAZEWICZ, J., J. K. LENSTRA AND A. H. G. RINNOOY KAN, "Scheduling Projects to Resource Constraints: Classification and Complexity," *Discrete Appl. Math.*, 5 (1983), 11–24.

CHRISTOFIDES, N., R. ALVAREZ-VALDES AND J. M. TAMARIT, "Project Scheduling with Resource Constraints: A Branch and Bound Approach," *European J. Oper. Res.*, 29 (1987), 262–273.

COOPER, D. F., "Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation," *Management Sci.*, 22, 11 (1976), 1186–1194.

DAVIS, E. W., "Resource Allocation in Project Network Models—A Survey," *J. Industrial Engineering*, 17, 4 (1966), 177–188.

———, "Project Scheduling Under Resource Constraints: Historical Review and Categorisation of Procedures," *AIIE Trans.*, 5, 4 (1973), 297–313.

——— AND G. E. HEIDORN, "An Algorithm for Optimal Project Scheduling under Multiple Resource Constraints," *Management Sci.*, 27, 12 (1971), B803–816.

——— AND J. PATTERSON, "A Comparison of Heuristic and Optimal Solutions in Resource-Constrained Project Scheduling," *Management Sci.*, 21, 8 (1975), 944–955.

ELMAGHRABY, S. E. AND W. S. HERROELEN, "A Cost Minimization Problem with Resource-Duration Interaction," Research Report, Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Belgium, 1977.

FISHER, M., "Optimal Solution of Scheduling Problems Using Lagrange Multipliers. Part I," *Oper. Res.*, 21, 5 (1973), 1114–1127.

FRENCH, S., *Sequencing and Scheduling—An Introduction to the Mathematics of the Job-Shop*, Halsted Press, New York, 1982.

GORENSTEIN, S., "An Algorithm for Project Sequencing with Resource Constraints," *Oper. Res.*, 20, 4 (1972), 835–850.

HERROELEN, W. S., "Resource-Constrained Project Scheduling—The State of the Art," *Oper. Res. Quart.*, 23 (1972a), 261–275.

———, *Heuristische Programmatie—Methodologische benadering en praktische toepassing op een hulpmiddelentoewijzingsprobleem*, Aurelia Books, Leuven, 1972b.

JOHNSON, T. J. R., "An Algorithm for the Resource-Constrained Project Scheduling Problem," unpublished Ph.D. Dissertation, MIT, 1967.

KURTULUS, I. AND E. W. DAVIS, "Multiproject Scheduling: Categorisation of Heuristic Rules Performance," *Management Sci.*, 28, 2 (1982), 161–172.

——— AND S. C. NARULA, "Multi-Project Scheduling: Analysis of Project Performance," *IIE Trans.*, 17, 1 (1985), 58–66.

MODER, J. J., C. R. PHILLIPS AND E. W. DAVIS, *Project Management with CPM, PERT and Precedence Diagramming*, Van Nostrand Reinhold Company, New York, 1983.

PATTERSON, J., "A Comparison of Exact Procedures for Solving the Multiple Constrained Resource Project Scheduling Problem," *Management Sci.*, 30, 7 (1984), 854–867.

——— AND W. D. HUBER, "Horizon-Varying Zero-One Approach to Project Scheduling," *Management Sci.*, 20, 6 (1974), 990–998.

——— AND G. ROTH, "Scheduling a Project under Multiple Resource Constraints: A Zero-One Programming Approach," *IIE Trans.*, 8, 4 (1976), 449–455.

PRITSKER, A. A. B., L. J. WATTERS AND P. M. WOLFE, "Multi-Project Scheduling with Limited Resources: A Zero-One Programming Approach," *Management Sci.*, 16, 1 (1969), 93–108.

SCHRAGE, L., "Solving Resource-Constrained Network Problems by Implicit Enumeration—Nonpreemptive Case," *Oper. Res.*, 18, 2 (1970), 225–235.

STINSON, J. P., E. W. DAVIS AND B. M. KHUMAWALA, "Multiple Resource-Constrained Scheduling Using Branch and Bound," *AIIE Trans.*, 10, 3 (1978), 252–259.

TALBOT, B. AND J. H. PATTERSON, "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," *Management Sci.*, 24, 11 (1978), 1163–1174.