European Journal of Operational Research 29 (1987) 262–273
North-Holland

Project scheduling with resource constraints: A branch and bound approach

Nicos CHRISTOFIDES
Imperial College, London, United Kingdom

R. ALVAREZ-VALDES
Universidad de Valencia, Valencia, Spain

J.M. TAMARIT
Universidad de Valencia, Valencia, Spain

Abstract: This paper describes a branch and bound algorithm for project scheduling with resource constraints. The 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. Four lower bounds are examined. The first is a simple lower bound based on longest path computations. The second and third bounds are derived from a relaxed integer programming formulation of the problem. The second bound is based on the Linear Programming relaxation with the addition of cutting planes, and the third bound is based on a Lagrangean relaxation of the formulation. This last relaxation involves a problem which is a generalization of the longest path computation and for which an efficient, though not polynomial, algorithm is given. The fourth bound is based on the disjunctive arcs used to model the problem as a graph.

We report computational results on the performance of each bound on randomly generated problems involving up to 25 activities and 3 resources.

Keywords: Scheduling, integer programming, Lagrange multipliers

1. Introduction

Scheduling a project under resource constraints consists of determining a set of starting times for the activities of the project in such a way that the precedence constraints between them and the resource constraints, limiting the total amount of available resources at each time of the schedule, are satisfied and the total completion time is minimized. If there are no resource constraints, problems of realistic size can be solved by critical path and network flow techniques. The introduction of these constraints changes the problem into a very difficult one and more elaborate mathematical programming techniques become necessary.

Pritsker et al. (1969) produce a 0–1 integer programming formulation of the problem involving three types of variables. The number of variables increases very rapidly with problem size, and their formulation can only be used on very small

We are very grateful to the referees for their useful comments and suggestions and to Joel P. Stinson for providing us with their algorithm.

Received June 1985; revised April 1986

problems. Davis and Heidorn (1971) propose a branch and bound algorithm based upon techniques devised by Gutjahr and Nemhauser (1964) for solving the Assembly Line Balancing Problem. Their algorithm produces a family of feasible sets, sets of tasks that could have been processed at a given time. The number of such sets grows very rapidly and again only small size problems can be handled. Another 0–1 formulation is proposed by Patterson and Roth (1976). Their formulation allows the use of an implicit enumeration algorithm in a special way that exploits the structure of the problem, leading to a reduction in computer memory and time requirements. This formulation is also used by Talbot and Patterson (1978), introducing the idea of network cuts, mechanisms that identify early in the enumeration process the partial schedules that cannot lead to an optimal solution. The use of these network cuts reduces solution times further. Stinson et al. (1978) develop a branch and bound algorithm with a similar formulation and new bounds, based upon precedence and resource constraints. The use of a four element decision vector at each node allows a significant reduction in the search tree and in the corresponding solution times.

There has been just one attempt at using Lagrangean Relaxation in order to obtain bounds for a branch and bound algorithm, made by Fisher (1973). He relaxes the resource constraints using Lagrangean multipliers. The relaxed problem is very easy to solve, but the problem of determining multipliers that produce a good solution is not. The algorithm obtains good bounds but computational results are not given.

A completely different approach for solving this problem was proposed by Balas (1969), using the concept of a disjunctive graph for the job shop scheduling problem. If there is just one machine of each type, two activities, i and j, that need the same machine cannot be processed simultaneously. In order to avoid this possibility a disjunctive pair of arcs $\{(i, j), (j, i)\}$ is added to the original graph, creating the disjunctive graph. Taking a feasible selection S which includes exactly one arc from each pair, together with the set H of fixed arcs, the longest path in the resulting graph is a feasible solution for the original problem. The problem is then one of selecting that S which produces a graph whose longest path is minimum. Balas (1970) generalizes this idea for the project scheduling problem. Since, in general, the amount of available resources is not equal to one, two activities needing the same resource are not necessarily incompatible. A selection S does not need to include one arc of each pair. There are many more possible selections and the feasibility of solutions is not assured. Some stability conditions are introduced in order to get a feasible solution, but the implementation of an algorithm is not simple and computational results are not given. Gorenstein (1972) designs an algorithm following the ideas of Balas. In order to compute the generalized stability coefficient that guarantees the feasibility of the solution, he devises a procedure based on the maximum flow in a bipartite graph. This procedure requires a large computational effort that limits the use of the algorithm. Bennington and McGinnis (1974) point out that Gorenstein's procedure is not efficient because the stability conditions of Balas are too restrictive.

The work reported in this paper is organized as follows: in Section 2 we give the formulation of the problem, defining its elements and underlying hypothesis. In Section 3 we describe the branch and bound algorithm, based on the idea of using disjunctive arcs for resolving conflicts. A first lower bound LB0, based on the longest path from each activity to the end of the project, is included and initial computational results are given. The remainder of the paper is devoted to determining tighter bounds. In Section 4 we give a 0–1 formulation and two relaxations. First we try the linear relaxation and we produce several types of cutting planes that improve the resulting bound LB1. Then we try the Lagrangean relaxation of the resource constraints. The relaxed problem can be seen as a generalization of a longest path computation including costs on the activities and we propose a simple and efficient branch and bound algorithm to solve it and obtain the lower bound LB2. Some computational results of the performance of these two bounds are given. In Section 5 we implement another lower bound LB3 working directly on the disjunctive graph of precedence constraints, creating a special kind of disjunctive arc to partly represent resource constraints. In Section 6, computational results are obtained using the algorithm described in Section 3 with lower bounds LB0 and LB3. Project scheduling problems are solved with up to 25 activities and 3 resources. The conclusions appear in Section 7.

2. Problem formulation

The problem of project scheduling with resource constraints can be formulated as follows:

Min t_n,

subject to

$$t_j - t_i \geq d_i, \quad (i, j) \in H,$$

$$\sum_{S(t)} r_{ik} \leq b_k, \quad \begin{matrix} t = 1, \ldots, T, \\ k = 1, \ldots, K, \end{matrix}$$

where

t_i = starting time of activity i, $i = 1, \ldots, n$.
H = set of pairs of activities with precedence constraints.
d_i = processing time (duration) of activity i.
r_{ik} = amount of resource k required by activity i.
$S(t)$ = set of activities in process at time t.
b_k = total availability of resource k.

We assume that the duration d_i of every activity i is known and independent of the moment in which the activity is processed. The resource requirements r_{ik} are also known constants over the entire interval in which the activity is processed and independent of the moment of the process. The level of resource availability b_k is a known constant throughout the schedule. Finally, the setup times are negligible or are included in the processing times. Usually activities 1 and n are fictitious activities that indicate the beginning and the end of the project.

3. The branch and bound algorithm

3.1. Description of the algorithm

The algorithm is based upon branch and bound during which a feasible schedule is built, trying at each node to put in process all the unscheduled activities satisfying the precedence constraints. The only moments in time to be considered are those in which one or more activities finish. At every one of these moments we define as candidates the activities not in the partial schedule whose predecessors have finished. These activities can start at that time if there are no problems with the resource constraints. The candidates are ordered by increasing $L(i)$, the length of the longest path from activity i to the end of the project. A candidate goes into the schedule and starts its process at that time if its resource requirements do not exceed the available resources left by the activities already in process.

If a candidate cannot start, there is a conflict that produces a new branching. The branches describe ways of resolving it, that is, decisions about which activities are to be delayed. One way is to delay the activity i that causes the conflict. In order to determine the other ways of solution we introduce the idea of an alternative. A set A of activities in process is an alternative to a candidate i if i could start if the activities of A were not in process. The delay of the activities of A allows the process of i and the conflict is solved. The delay of a candidate or an alternative is introduced by adding arcs that force some activities to wait until the completion of some others. The best way of delaying a candidate is to choose as the initial vertices of the added arcs either the candidate or some other activity in process not belonging to A.

A first lower bound LB0 is introduced. If we add the arc (i, j) to delay j, a lower bound for the total completion time of the project will be: LB0 $= t_i + d_i + L(j)$, that is, the finishing time of i plus the length of the logest path from j to n. This bound, based only upon procedence constraints, is useful for problems in which the resource constraints are not very tight. Its main advantage is its ease of computation.

We follow a depth-first strategy in the tree search. We go to backtrack when a schedule is completed or a branch is fathomed by the lower bound. At the backtrack we remove the added arcs corresponding to the last alternative studied and add new arcs for the next one at the same level. If there is no alternative left at that level, we go up to the previous one. When we get to level zero, the process is finished.

There is a special case to be considered. If at a certain moment the set of activities in process is empty and a candidate cannot be processed simultaneously with any other unscheduled activity, this 'non-sharing' candidate can be put in process without considering the alternatives. All other ways of resolving the conflict that this candidate necessarily originates cannot lead to a better solution.

The procedure is summarized as follows:

Step 1 (Start). Let T be un upper bound on the total completion time. For every activity i, calculate $L(i)$. Set $p=0$ (level of branching). Put initial activity 1 in process: $t_1=0$; $Q=\{1\}$ (partial schedule); $S=\{1\}$ (activities in process).

Step 2 Set $m=\min\{t_i+d_i;\ i\in S\}$. For all $j\in S$ such that $t_j+d_j=m$, $S=S-\{j\}$. $C=\emptyset$ (set of candidates); $N=\emptyset$ (activities which cause conflicts).

Step 3 (Construction of C). For each $i\in Q$, if for all $(j,i)\in H$, $t_j+d_j\leqslant m$, $C=C\cup\{i\}$. If $C=0$, go to Step 2. Otherwise, order C by decreasing $L(i)$. If $S=0$, go to Step 11. Otherwise,

Step 4 (Test of candidates). For each $i\in C$, if $\Sigma_{j\in S}r_{jk}+r_{ik}>b_k$, for some k, $N=N\cup\{i\}$. Otherwise, $Q=Q\cup\{i\}$; $S=S\cup\{i\}$; $t_i=m$. If the last scheduled activity is n, the schedule is completed: Set $T=\min\{T,\ t_n+d_n\}$ and go to Step 10.

Step 5. If $N=0$, go to Step 2. Otherwise,

Step 6 (Construction of $\mathscr{A}(i)$, set of alternatives to i). Select $i\in N$: $N=N-\{i\}$. Form the set

$$\mathscr{A}(i)=\Big\{A\subset S/\sum_{j\in S}r_{jk}-\sum_{h\in A}r_{hk}+r_{ik}\leqslant b_k,\ \text{for all } k\Big\}.$$

Step 7 (Identification of the best way of delaying the candidate and every alternative). For the candidate i, find a^* such that $t_{a^*}+d_{a^*}=\min\{t_a+d_a,\ a\in A\in\mathscr{A}(i)\}$. For each $A\in\mathscr{A}(i)$, find $a^*\in A$, such that $t_{a^*}+d_{a^*}=\min(t_a+d_a,\ a\in A^*\in\mathscr{A}(i)-\{A\})$. If $t_{a^*}+d_{a^*}<m+d_i$, keep $b(A)=a^*$ to delay A in the backtrack. If $m+d_i\leqslant t_{a^*}+d_{a^*}$, keep $b(A)=i$.

Step 8 (Branching). Set $p=p+1$; $H^*(p)=\{(a^*,i)\}$; $B(p)=\mathscr{A}(i)$ (set of branches at level p).

Step 9 (Lower bound). Set $\mathrm{LB0}=\max\{t_j+d_j+L(i),\ (j,i)\in H'(p)\}$. If $\mathrm{LB0}\geqslant T$, go to Step 10. Otherwise, go to step 5.

Step 10 (Backtrack). $H=H-H'(p)$; $H'(p)=0$, If $B(p)=0$, set $p=p-1$. If $p=0$, STOP. Otherwise, select $A\in B(p)$. $B(p)=B(p)-\{A\}$; $H'(p)=\{(b(A),\ j)\ \ j\in A\}$. $H=H\cup H'(p)$; $Q=Q-A$; $S=S-A$. Go to Step 9.

Step 11 (Non-sharing candidate). For each $i\in C$, if, for all $j\in Q$, $r_{jk}+r_{ik}>b_k$ for some k: $Q=Q\cup\{i\}$; $S=S\cup\{i\}$; $t_i=m$, and go to Step 2. Otherwise, go to Step 4.

3.2. Computational results

We have tried to generate problems similar to the test problems used in the literature. A feature that appears to be very important is the strength of the resource constraints. A measure for it is the ratio between the total resource requirements (sum of the resource requirements of each activity by its processing time) and the total available resources (sum of resource availabilities by the length of the longest path in the original graph). That is,

$$r=\Big(\sum_i\sum_k r_{ik}d_i\Big)\Big/\Big(L\sum_k b_k\Big),$$

where L is the length of the longest path. If this ratio varies between 0.5 and 0.99 we consider the problem loosely constrained. Problems with ratios between 1.0 and 1.5 are tightly constrained. This ratio is mentioned by Davis and Patterson (1975), who generate problems with ratios between 0.53 and 1.50. We have randomly generated problems 1–20 with ratios 0.5–0.9 and problems 21–40 with ratios 1.0–1.5. All of them have 25 activities and 3 resources. Durations are in the range 1–9, and resource requirements in the range 0–6. The resource availabilities are 6. In the graph of precedence constraints the ratio between arcs and vertices varies between 1 and 3.

Computational results using the algorithm described in Section 3.1, with lower bound LB0 are shown in Table 1. The solution times correspond to a UNIVAC 1100 computer.

4. An integer programming formulation and its relaxations

The computational results reported above indicate the limitations of the branch and bound algorithm with lower bound LB0 when problems are tightly constrained. In this Section we present an integer programming formulation and two relaxations: a linear relaxation with the introduction of cutting planes and a Lagrangean relaxation.

Table 1
Computational results of branch and bound with LB0

Problem number	Depth of tree	Arcs ades to first solution	Number of nodes	Nodes fathomed	First solution value	Optimal solution value	CPU time (secs.)
1	8	10	48	26	31	31	0.19
2	13	14	276	201	25	24	0.30
3	5	5	496	302	30	30	0.67
4	13	15	1172	785	26	25	1.07
5	8	9	3349	2146	29	29	3.42
6	6	7	29	16	49	49	0.16
7	0	0	1	0	77	77	0.14
8	9	12	3430	1989	37	36	5.74
9	8	10	460	304	38	37	0.84
10	6	8	556	343	26	26	1.14
11	7	7	58	36	36	36	0.19
12	11	11	513	389	24	23	0.51
13	18	21	8054	4897	39	35	10.21
14	11	11	210	141	36	35	0.37
15	4	4	5	0	33	33	0.18
16	9	9	267	184	26	25	0.44
17	8	9	931	525	38	36	1.58
18	8	8	179	106	39	38	0.34
19	8	11	422	200	54	54	1.06
20	9	11	199	143	25	22	0.36
21	21	33	62	8	74	71	0.33
22	31	56					60.00 [a]
23	28	29	10724	4521	57	55	24.50
24	25	45					60.00 [a]
25	19	37					60.00 [a]
26	19	25	328	169	71	71	0.69
27	8	8	9	0	81	81	0.16
28	20	32	217	94	54	54	0.60
29	18	33	74	8	63	63	0.32
30	33	49					60.00 [a]
31	21	31					60.00 [a]
32	24	43	2322	834	66	66	6.33
33	23	41	1399	657	64	60	3.40
34	17	29	119	24	77	74	0.54
35	17	29	2867	1469	70	70	5.71
36	23	41	663	269	68	68	2.19
37	25	47	4998	2294	71	69	14.13
38	17	23	189	29	72	72	0.52
39	17	21	3780	976	73	72	10.06
40	23	27					60.00 [a]

[a] Indicates that the problem could not be solved within the limit of 60 CPU seconds.

4.1. An integer programming formulation of the problem

Let

$$y_{it} = \begin{cases} 1 & \text{if activity } i \text{ starts at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

The problem can be formulated as follows:

$$\text{Minimize} \quad \sum_t t y_{nt}, \tag{4.1}$$

subject to

$$\sum_t y_{it} = 1, \quad i = 1, \ldots, n, \tag{4.2}$$

$$\sum_t t(y_{jt} - y_{it}) \geqslant d_i, \quad (i, j) \in H, \tag{4.3}$$

$$\sum_i r_{ik} \left(\sum_{t-d_i+1}^{t} y_{im} \right) \leqslant b_k, \quad k = 1, \ldots, K, \quad t = 1, \ldots, T, \tag{4.4}$$

$$y_{it} \in \{0, 1\}. \tag{4.5}$$

Constraints (4.2) indicate that every activity must start once. Constraints (4.3) are the precedence constraints and (4.4) the resource constraints.

4.2. Linear programming relaxation

If we relax the integrality requirements (4.5) the problem becomes a LP problem and it can be somewhat strengthened adding constraints which are redundant with respect to the original integer problem but not with respect to the LP relaxation. For every activity i: $y_{it} = 0$ if $t < m(i)$, where $m(i)$ is the early start time of activity i, obtained by calculating the longest path in the graph of precedence constraints. Moreover, $y_{it} = 0$ if $t > M(i)$, where $M(i)$ is the difference between an upper bound T on the completion time of the project and the longest path from i to the end.

We have solved some examples using the LP code FMPS of UNIVAC. Table 2 summarizes the results on a set of 20 test problems. Problems 1–10 are a subset of the loosely constrained problems used in Table 1 with 25 activities. Problems 11–20 are tightly constrained problems with up to 15 activities. We have not used problems of Table 1 with tight resource constraints because the corresponding LP problems were too expensive to solve with the available LP code. Bound LB1 has not been included in the branch and bound algorithm and the results of Table 2 serve only as a reference for the quality of a LP-based bound.

Column (1) of Table 2 gives the optimal solution values; column (2) the lengths of the longest path and column (3) the bounds obtained by the linear relaxation.

4.3. Cutting planes

In order to improve the quality of the LP bounds we have devised a collection of cutting planes. We start from the LP solution and we identify which integer constraints are violated. We introduce new constraints or modify the existing ones to avoid these violations and solve the linear problem again, obtaining a new optimal solution. Proceeding in that way we have used four types of cutting planes.

(a) We introduce a term for activity n in the resource constraints, with $r_{nk} = b_k$ and $d_n = T$, in order to prevent other activities being processed simultaneously with n (or even after it). The results obtained with this new constraint,

$$\sum_i r_{ik}\left(\sum_{t-d_i+1}^{t} y_{im}\right) + r_{nk}\sum_{1}^{T} y_{nm} \leqslant b_k, \quad \forall t, \forall k.$$

appear in column (4) of Table 2.

Table 2
Computational results of linear and Lagrangean relaxations

Problem number	Optimal value	PERT value	Linear relaxation	Linear +1	Linear +1, 2	Linear +1, 2, 3	Linear +1, 2, 3, 4	Lagrangean relaxation
1	31	29	29.0	29.0	29.0	29.0	29.0	29.21
2	30	30	30.0	30.0	30.0	30.0	30.0	30.0
3	25	23	23.33	24.0	24.0	24.23	24.23	24.23
4	49	46	46.0	46.0	46.0	46.20	46.20	46.0
5	37	27	29.22	29.62	30.37	31.75	32.33	30.10
6	37	33	33.0	33.0	33.0	33.38	34.50	33.32
7	26	25	25.0	25.0	25.0	25.0	25.0	25.0
8	35	30	30.56	30.97	31.63	33.49	33.49	31.87
9	38	33	33.81	33.81	33.81	34.30	34.81	34.32
10	54	40	42.47	42.47	42.47	44.42	45.10	52.66
11	18	9	11.69	12.29	13.30	13.83	16.22	11.63
12	22	9	12.16	13.99	16.20	17.25	19.50	12.46
13	23	14	14.62	15.36	16.42	16.58	21.79	15.74
14	26	15	17.83	17.84	18.26	20.15	26.0	18.34
15	21	14	15.0	15.24	15.39	17.98	20.68	15.73
16	20	11	13.82	14.25	14.92	16.21	18.94	14.08
17	22	13	14.15	16.06	16.87	17.25	20.71	15.06
18	21	13	15.0	16.03	16.9	18.22	19.67	15.60
19	18	10	10.96	11.64	12.22	12.50	18.0	11.64
20	20	11	12.84	14.62	15.14	16.0	20.0	13.43

(b) In the solutions corresponding to column (4) of Table 2, the processing of activity n is scattered over time because this activity appears in the objective function with a linear coefficient in t. We try to get integrality by making y_{nt} more 'expensive' for large values of t by introducing in the objective function the cost coefficient 10^t instead of t. The objective function will be

$$\sum_t 10^t y_{nt}.$$

The influence of this change in the objective function, which produces a problem equivalent to the original one, is shown in column (5) of Table 2.

(c) We reinforce the precedence constraints with a new constraint:

$$\sum_{m=t}^{T} y_{im} + \sum_{s=1}^{t+d_i-1} y_{js} \leqslant 1, \quad t = 1, \ldots, T;$$

$$(i, j) \in H.$$

The results obtained when we add these constraints appear in column (6) of Table 2.

(d) Non-integrality makes resource constraints very loose. For instance, if $r_{ik} = 4$ and $r_{jk} = 3$, with $b_k = 6$, activities i and j cannot be processed simultaneously. But the linear programme can make $y_{it} = 0.85$ and $y_{jt} = 0.85$. To avoid that, we construct sets B of mutually incompatible activities. We can then write

$$\sum_{i \in B} y_{it} \leqslant 1, \quad t = 1, \ldots, T.$$

The effect of this constraint, which appears in column (7) of Table 2, is clearly different for the two sets of problems.

4.4. Lagrangean relaxation

Relaxing the resource constraints of the integer formulation with multipliers $\lambda_{tk} \geqslant 0$, we obtain the relaxed problem:

$$\text{Minimize} \quad \sum_t t y_{nt} + \sum_i \sum_t \alpha_{it} y_{it} - \sum_k \sum_t \lambda_{tk} b_k,$$

subject to

$$\sum_t y_{it} = 1, \quad i = 1, \ldots, n,$$

$$\sum_t t(y_{jt} - y_{it}) \geqslant d_i, \quad (i, j) \in H,$$

$$y_{it} \in \{0, 1\},$$

where

$$\alpha_{it} = \sum_k r_{ik} \left(\sum_{m=t}^{t-d_i+1} \lambda_{mk} \right), \quad i = 1, \ldots, n, \quad t = 1, \ldots, T.$$

This relaxed problem can be seen as a generalization of a longest path computation in which we have to minimize not only the completion time but the sum of the 'costs' α_{it} of starting activity i at time t. We propose an algorithm to solve this problem.

Given an upper bound T on the completion time of the project and $m(n)$, the longest path in the graph of precedence constraints, the basic idea is to solve a subproblem for each value of the slack $h = T - m(n)$. Let z_0 be the value of the best current solution. Initially $z_0 = m(n) + \sum_i \alpha_{im(i)}$, with each activity starting at its early start time. For a given slack h, the objective function of the subproblem is $m(n) + h + \sum_i \alpha_{it}$. Let

$$\alpha^*_{is(i)} = \min\{\alpha_{is} / m(i) \leqslant s \leqslant m(n) + h - L(i)\}.$$

If $m(n) + h + \sum_i \alpha^*_{is(i)} \geqslant z_0$, the subproblem does not need to be solved. Otherwise, we solve it following a branch and bound procedure. An upper bound is z_0. At node zero, a lower bound, obtained by relaxing the precedence constraints, is $\mathrm{LW}_1 = m(n) + h + \sum_i \alpha^*_{is(i)}$. Clearly, $\mathrm{LW}_1 < z_0$. We check the feasibility of the solution given by the times corresponding to $s(i)$. If the solution is feasible, it is the optimal solution. If it is not, some precedence constraints are violated. We construct a better lower bound starting from LW_1 and augmenting the α_{it} of some activities. If we substitute $\alpha^*_{is(i)} + \alpha^*_{js(j)}$ of a violated arc (i, j) by $\min\{\alpha_{is} + \alpha_{jt} / t - s \geqslant d_i\}$ the new value of the sum is still a lower bound, because in the optimal solution this precedence constraint must be satisfied. We can repeat this substitution for other violated arcs in such a way that an activity is not reassigned twice. We denote the resulting bound LW_2. The reassignment may produce a feasible solution that would be optimal. If the solution is still not feasible and $\mathrm{LW}_2 < z_0$, we go to the next level of branching fixing the starting time of some activity at a certain time. At each node in the search tree we calculate similar bounds LW_1 and LW_2 and fathom the branch if we find a feasible solution or $\mathrm{LW}_2 \geqslant z_0$.

The algorithm can be summarized as follows:

Step 1 (Initialization). Set $H = T - m(n)$ (total slack); $h = -1$ (value of the slack defining the subproblem); $z_0 = m(n) + \sum_i \alpha_{im(i)}$.
Step 2 (Next subproblem). Set $h = h + 1$; $p = 0$ (level of branching); $F = \emptyset$ (set of fixed activities). If $h > H$, go to Step 8.
(Solution of the subproblem)
Step 3 (Computation of LW_1). For each activity i, $A(i) = \{t/m(i) \leqslant t \leqslant m(n) + h - L(i)\}$; $\alpha^*_{is(i)} = \min\{\alpha_{is}/s \in A(i)\}$. Compute

$$LW_1 = m(n) + h + \sum_{i \notin F} \alpha^*_{is(i)} + \sum_{i \in F} \alpha_{it(i)}.$$

If $LW_1 \geqslant z_0$, go to Step 7.
Step 4 (Check of the feasibility of the solution). Set $Q = 0$ (set of violated arcs). For all $(i, j) \in H$ such that $s(j) - s(i) < d_i$, $Q = Q \cup \{(i, j)\}$. If $Q = 0$ (solution feasible), set $z_0 = LW_1$ and go to Step 7.
Step 5 (Computation of LW_2). Set $LW_2 = LW_1$. Select $S \subset Q$ such that an activity does not appear twice. For all $(i, j) \in S$,

$$LW_2 = LW_2 - \alpha^*_{is(i)} - \alpha^*_{js(j)} + \min\{\alpha_{is} + \alpha_{jt}/t - s \geqslant d_i\}.$$

If $LW_2 \geqslant z_0$, go to Step 7. Otherwise, check the feasibility of the reassigned solution. If it is feasible, set $z_0 = LW_2$ and go to Step 7. Otherwise,
Step 6 (Branching). Set $p = p + 1$. Fix the starting time of an activity i at $t(i)$; $F = F \cup \{i\}$; $f(p) = 1$ (status of the level); $o(p) = i$ (activity involved in the branching). Go to Step 3.
Step 7 (Backtrack). If $p = 0$, go to Step 2. Otherwise, if $f(p) = 1$, set $f(p) = 0$; $i = o(p)$; $F = F - \{i\}$; $A(i) = A(i) - \{t(i)\}$, and go to Step 3. If $f(p) = 0$, $i = o(p)$; $A(i) = A(i) \cup \{t(i)\}$; $p = p - 1$, and repeat the step.
Step 8 (End). The optimal solution value is z_0.

The algorithm described above efficiently solves the relaxed problem and we can use it to obtain lower bound LB_2. The bounds obtained by the Lagrangean relaxation appear in column (8) of Table 2. We have used Subgradient Optimization (Held et al., 1974) to get the multipliers at each iteration and every problem includes 50 iterations.

5. An alternative method of obtaining lower bounds

5.1. Description of the bound

From the results of the previous Section it is clear that we need a method of relaxing the resource constraints which does not produce a 'too relaxed' problem. The method that we propose in this Section is based on the idea of disjunctive arcs.

If, in the integer programming formulation of Section 4.1, we relax the resource constraints (4.4), the problem is a simple longest path computation and its solution will be a lower bound. Now, let us assumed that two activities, a and b, cannot be processed simultaneously because of their resource requirements. Hence, in every feasible solution activities a and b will be one after another, but we do not know their relative order in the optimal solution. We create two subproblems:

Problem A

$$\text{Min} \quad \sum_t t y_{nt},$$
$$\text{s.t.} \quad (4.2),\ (4.3),\ (4.5)$$
$$\sum_t t(y_{at} - y_{bt}) \geqslant d_b;$$

Problem B

$$\text{Min} \quad \sum_t t y_{nt},$$
$$\text{s.t.} \quad (4.2),\ (4.3),\ (4.5),$$
$$\sum_t t(y_{bt} - y_{at}) \geqslant d_a.$$

We relax the resource constraints and introduce the disjunctive pair $\{(a, b), (b, a)\}$. The arc (b, a) creates problem A, forcing a to come after b, and the arc (a, b) the problem B, forcing b to come after a. One of them is a lower bound for the optimal solution. If both have solutions greater than or equal to the best current solution, the partial schedule that we are building cannot lead to a better solution and the node is fathomed.

We can generalize this idea: Let us assume that we are in a certain search tree node and that we have a partial schedule. For the remaining activities the problem is that of (4.1), subject to (4.2), (4.3), (4.4) and (4.5). We relax constraints (4.4) and add constraints of type (4.3) corresponding to all the possible combinations of disjunctive arcs

created to separate pairs of activities which cannot be processed simultaneously. We solve all these problems. If the minimum of all solution times is greater than or equal to the best current solution, the node is fathomed. The added arcs may represent all the limitations of the resource constraints or may not, either because there are sets of activities that cannot be processed together but can be processed two at a time, or because we do not include all the disjunctive arcs that can be found.

If we use n disjunctive arcs, we have to make 2^n longest path computations. Though the problems are very easy to solve, we have to get a balance between the speed of computation and the quality of the bound. Based on our computational experience, we use a maximum of 8 arcs, that is, we have to solve at most 256 problems. We have developed a procedure to reduce the graph in which the logest path has to be calculated and to avoid the explicit computation of every longest path.

In every subproblem we have a graph whose vertices are the unscheduled activities and whose arcs are the precedence constraints between them. Before calculating the longest path we can reduce the graph using the following observations: If one activity cannot be processed simultaneously with any other in the graph, we can remove it from the graph and add its duration to the solution of the problem in the reduced graph. If one activity i can be processed only with j and $d_i \geq d_j$, we can remove them from the graph and add d_i to the solution of the problem in the reduced graph. If $d_i < d_j$, we can remove i from the graph and make $d_j = d_j - d_i$, adding d_i to the solution of the problem in the redued graph. These observations are very useful in tightly constrainted problems in which they lead to significant increases in the solution value of the subproblems. Sometimes they reduce the graph completely and the computation of the longest path is not necessary, as can be seen

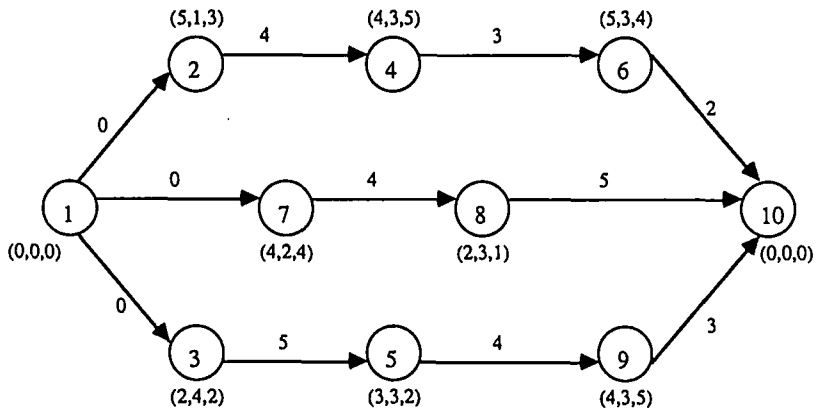

Figure 1. Original graph of Example 1

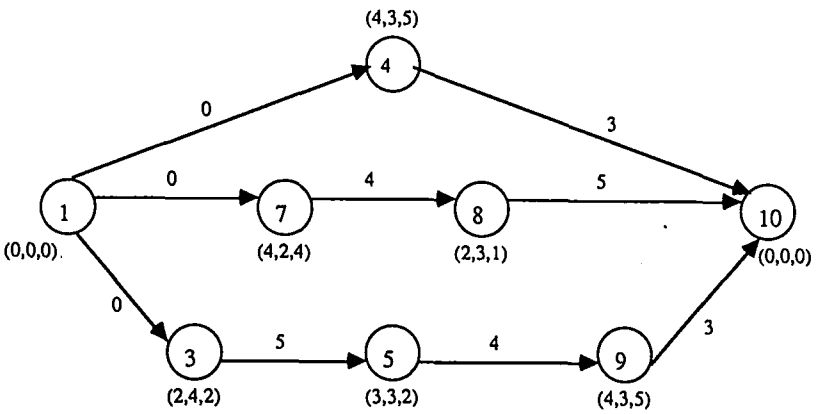

Figure 2. Graph of Example 1 after first reduction

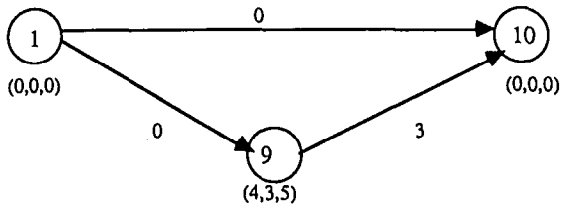

Figure 3. Graph of Example 1 after further reductions

Table 3
Reductions in the number of longest paths to be computed

Problem	Number of problems without reduction	Number of problems with reduction
1	1288	30
6	256	10
11	40	14
13	3638	149
17	1988	119
18	290	58
25	6114	430

Table 4
Computational results of the algorithm with LB0 and LB3

Problem number	First feasible solution	Optimal solution	Number of nodes	Nodes fathomed by LB0	Nodes fathomed by LB3	CPU time (seconds)	Stinson time (seconds)
1	31	31	48	28	0	0.18	1.69
2	25	24	432	318	0	0.56	2.88
3	30	30	582	358	0	1.39	1.13
4	26	25	1806	1227	0	2.46	3.62
5	29	29	4827	3124	0	9.57	1.14
6	49	49	32	18	0	0.16	1.23
7	77	77	1	0	0	0.13	0.98
8	36	34	35	22	0	0.18	1.36
9	38	37	389	297	0	1.38	2.65
10	26	26	618	388	0	2.43	1.82
11	36	36	85	54	0	0.26	1.30
12	24	23	667	512	0	1.00	3.78
13	39	35	121	20	56	11.72	2.49
14	36	35	281	191	0	0.69	2.37
15	33	33	4	0	0	0.14	1.42
16	26	25	266	184	0	0.65	2.02
17	38	35	1025	587	0	2.36	4.49
18	39	38	176	104	0	0.47	1.90
19	54	54	793	413	0	2.53	3.48
20	24	21	350	257	0	0.81	2.82
21	74	71	12	0	5	0.84	7.63
22	63	60	110	0	54	7.81	47.50
23	57	55	86	2	39	5.64	12.44
24	57	54	70	0	33	5.13	23.02
25	58	57	104	0	51	22.44	43.20
26	71	71	146	29	44	9.15	4.00
27	81	81	1	0	0	0.20	1.06
28	54	54	22	3	8	0.66	3.03
29	63	63	8	0	4	0.91	6.29
30	54	50	134	16	48	8.73	8.34
31	66	62	102	1	46	10.82	16.41
32	66	65	60	2	27	3.87	21.80
33	64	60	76	4	33	2.70	20.90
34	77	74	16	0	6	0.87	6.54
35	70	70	30	0	15	1.43	3.86
36	68	68	28	2	12	0.99	7.63
37	71	69	66	0	31	4.40	18.67
38	72	72	10	2	3	0.99	9.27
39	73	72	35	0	18	7.86	10.58
40	62	58	104	2	48	17.62	63.53

in Example 1. We keep the durations of the activities removed from the graph in D.

Example 1. Let us assume we have the graph of Figure 1. The numbers beside the nodes are the resource requirements r_{ik}, and the processing times appear on the arcs. The resource availabilities, b_k, are equal to 6, $k = 1, 2, 3$. Activities 2 and 6 cannot be processed with any others. We remove them from the graph and $D = d_2 + d_6 = 6$. The reduced graph appears in Figure 2. Similarly, we can eliminate activities 3 and 7, adding d_7 to D. We can eliminate activity 4, which can be processed only with 8, adding d_4 to D and making $d_8 = d_8 - d_4$. Finally, we can eliminate activities 5 and 8 adding d_5 to D. The final value of D is 18 and the reduced graph is shown in Figure 3. The solution of the problem is $D + d_9 = 21$, which is much greater than the length of the longest path in the original graph, which was 12.

Usually we do not need to study explicitly all the possible combinations of added arcs. Let us suppose that we have two disjunctive arcs $\{(a, b), (b, a)\}$ and $\{(c, d), (d, c)\}$. We introduce first arcs (a, b) and (c, d). We calculate the longest path and the resulting bound is greater than the best current solution. We examine the longest path and, for example, (a, b) is in it, but (c, d) is not. We do not need to study the problem with added arcs (a, b) and (d, c), because the new graph includes the former logest path. Hence, the length of the longest path in the new graph will be at least equal to the length of the previous one, and the resulting bound will be greater than the best current solution. The benefits of the reduction in the number of problems to be explicitly solved can be seen in Table 3.

5.2. Computational results

We have used the test problems described in Section 3.2. Table 4 shows the computational results obtained for these test problems using the branch and bound algorithm described in Section 3.1 with lower bounds LB0 and LB3 (the bound described in Section 5.1). The solution times correspond to a UNIVAC 1100 computer.

Note that the value of the first feasible solution is on average within 2.7% of the optimal solution value.

6. Conclusions

We have developed a branch and bound algorithm and embedded in it four lower bounds to reduce the tree search. The first lower bound LBØ is based only on the precedence constraints. It is very easy to compute and it can be used jointly with any other bound. It has been considered as a part of the branch and bound algorithm. The second bound LB1 uses the linear relaxation of an integer programming formulation of the problem, improved by the introduction of cutting planes. The computational results of Section 4.3 are quite promising, but the lack of an efficient and flexible LP code has not allowed us to use it in the branch and bound procedure. The third bound LB2 is based on the Lagrangean relaxation. Though the relaxed problem can be solved very efficiently by the algorithm described in Section 4.4, the computational results that we have obtained seem to indicate that the lagrangean relaxation is not a useful technique for this problem. We tried to use bound LB2 in the branch and bound algorithm together with LB3. When LB3 failed and its distance to the best current solution was small we used LB2, but LB2 rarely fathomed a branch and the solution times were worse than those obtained using only LB3.

The fourth lower bound LB3 is based on the idea of using disjunctive arcs to partly describe the resource constraints involving the unscheduled activities at a certain node. With the refinements mentioned in Section 5.1 the bound performs quite well, specially for problems with tight resource constraints, for which LB0 was almost useless.

The last column of Table 4 shows the solution times obtained by using the branch and bound algorithm of Stinson et al. (1978). For problems 1–20, with loose resource constraints, the mean solution times using our algorithm is 1.95 seconds, and using theirs 2.23 seconds. Times are quite similar because bounds are similar and the differences on each problem may be due to the different branching strategies. For problems 21–40, with tight resource constraints, the mean solution time using our algorithm is 5.65 seconds and using theirs 16.78 seconds. For these problems, bound LB3 performs quite well and seems to be tighter than those proposed by Stinson et al. (1978).

We can conclude by saying that for problems

with loose resource constraints bounds based on longest path computations seem to be good enough. For problems with tight resource constraints, more elaborated bounds, like those proposed here, are needed. More work needs to be done to accommodate more flexible resource structures and to extend the results obtained so far to problem of realistic size.

References

Balas, E. (1969), "Machine sequencing via disjunctive graphs: An implicit enumeration algorithm", *Operations Research* 17, 941–957.

Balas, E. (1970), "Project scheduling with resource constraints", in: E.M.L. Beale (ed.), *Applications of Mathematical Programming Techniques*, American Elsevier, New York.

Bennington, G.E., and McGinnis, L.F. (1974), "An improved feasibility test for Gorenstein's algorithm", *Operations Research* 22, 1117–1119.

Davis, E.W., and Heidorn, G.E. (1971), "An algorithm for optimal project scheduling under multiple constraints", *Management Science* 17, B803–816.

Davis, E.W., and Patterson, J.H. (1975), "A comparison of heuristics and optimum solution in resource-constrained project scheduling", *Management Science* 21, 944–955.

Fisher, M.L. (1973), "Optimal solution of scheduling problems using lagrange multipliers". Part I: *Operations Research* 21, 1114–1127. Part II: in: S.E. Elmaghraby (ed.), *Symposium on the Theory of Scheduling and its Applications*, Springer, Berlin, 294–318.

Functional Mathematical Programming System (FMPS), Programmer Reference UP 8198 Rep. 1, Sperry Univac.

Gorenstein, S. (1972) "An algorithm for project (job) scheduling with resource constraints", *Operations Research* 20, 835–850.

Gutjhar, A.L., and Nemhauser, G.L. (1964), "An algorithm for the Line-Balancing Problem", *Management Science*.

Held, M., Wolfe, P., and Crowder, H.P. (1974), "Validation of the Subgradient Optimization", *Mathematical Programming* 6, 62–88.

Patterson, J.H. and Roth, G.W. (1976), "Scheduling a project under multiple resource constraints: A 0–1 programming approach", *AIEE Transactions* 8, 449–455.

Pritsker, A.A., Waters, L.J., and Wolfe, P.M. (1969), "Multiproject scheduling with limited resources: A 0–1 approach", *Management Science* 16, 93–108.

Stinson, J.P., Davis, E.W., and Khumawala, B.H. (1978), "Multiple resource-constrained scheduling using branch and bound", *AIEE Transactions* 10, 252–259.

Talbot, F.B., and Patterson, J.H. (1978) "An efficient integer programming algorithm for solving resource constrained scheduling problems", *Management Science* 24, 1163–1174.