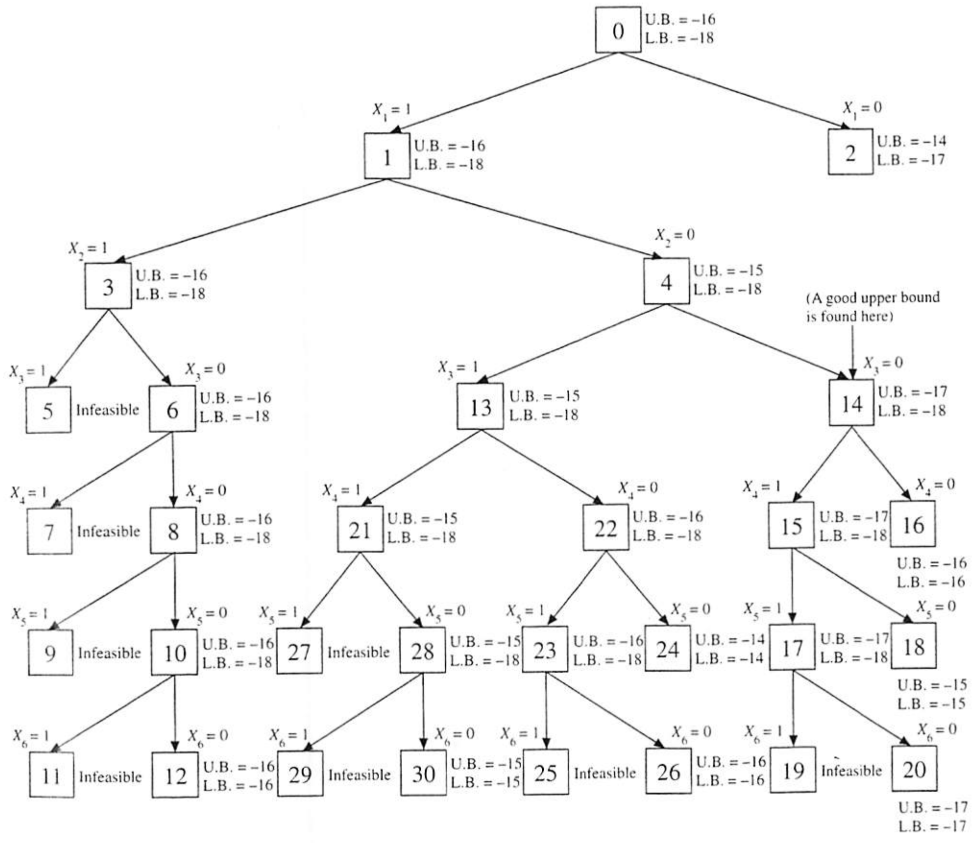
FIGURE 5–28 A 0/1 knapsack problem solved by the branch-and-bound strategy.



In the tree shown in Figure 5–28,

(1) Node 2 is terminated because its lower bound is equal to the upper bound of node 14.

(2) All other nodes are terminated because the local lower bound is equal to the local upper bound.

5-9 A JOB SCHEDULING PROBLEM SOLVED BY THE BRANCH-AND-BOUND APPROACH

While it is easy to explain the basic principles of the branch-and-bound strategy, it is by no means easy to use this strategy effectively. Clever branching and bounding rules still need to be devised. In this section, we shall show the importance of having clever bounding rules.

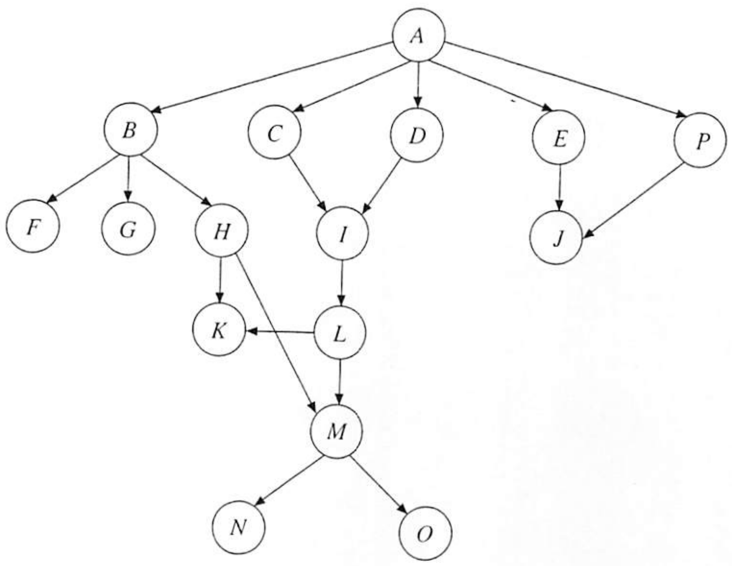
Our problem is a job scheduling problem based on the following assumptions:

(1) All the processors are identical and any job can be executed on any processor.
(2) There is a partial ordering of jobs. A job cannot be executed if one of its ancestor jobs, if it exists, has not been executed yet.
(3) Whenever a processor is idle and a job is available, this processor must start working on the job.
(4) Each job takes equal time for execution.
(5) A time profile is given which specifies the number of processors that can be used simultaneously in each time slot.

The object of the scheduling is to minimize the maximum completion time, which is the time slot in which the last job is finished.

In Figure 5–29, a partial ordering is given. As can be seen, job *I* must wait for jobs *C* and *D* and job *H* must wait for job *B*. At the very beginning, only job *A* can be executed immediately.

FIGURE 5–29 A partial ordering of a job scheduling problem.



Let us consider the time profile shown in Table 5–11.

TABLE 5–11 A time profile.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
3	2	2	2	4	5	3	2	3

This time profile indicates that at time $t = 1$, only three processors can be used and at $t = 5$, four processors can be active.

For the partial ordering in Figure 5–29 and the above time profile, there are two possible solutions:

Solution 1:

T_1	T_2	T_3	T_4	T_5	T_6	
A	B	C	H	M	J	
*	D	I	L	E	K	Time = 6
*				F	N	
				P	O	
					G	

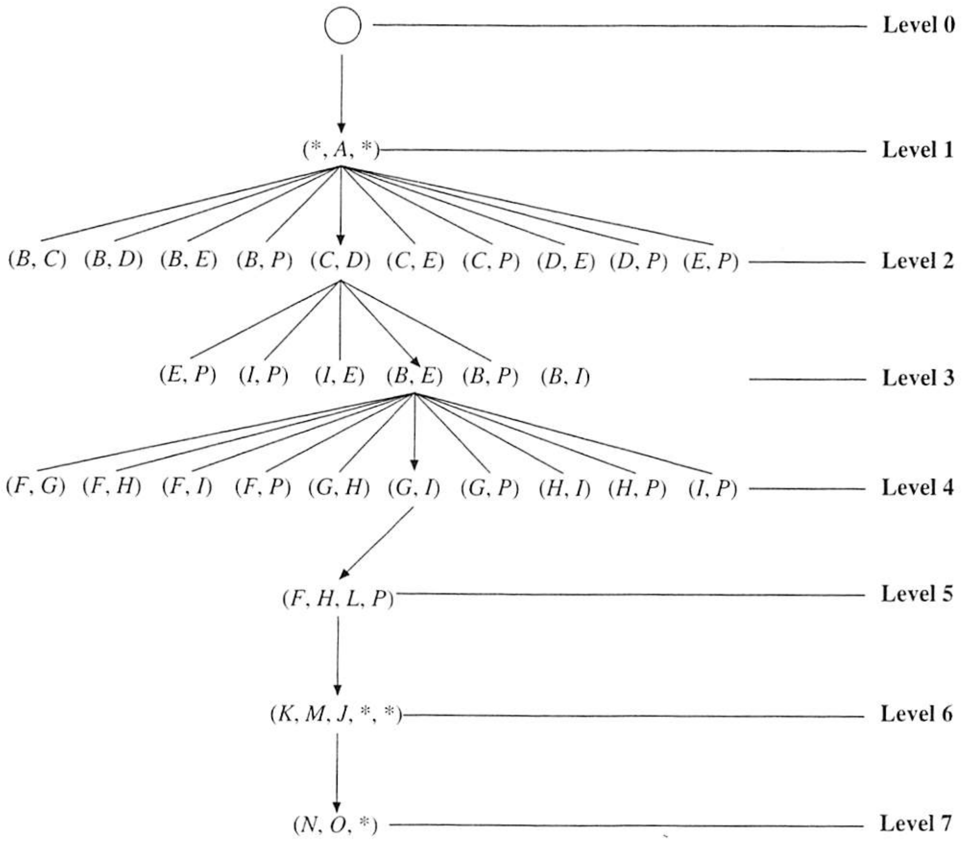
Solution 2:

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
A	B	D	F	H	L	M	N	Time = 8
*	C	E	G	I	J	K	O	
*				P	*	*		
				*	*			
					*			

Obviously, Solution 1 is better than Solution 2. Our job scheduling is defined as follows: We are given a partial ordering graph and a time profile; find a solution with the minimal time steps to finish all the jobs. This problem is called equal execution-time job scheduling problem with precedence constraint and time profile. It was proved to be an NP-hard problem.

We shall first show that this job scheduling problem can be solved by tree searching techniques. Suppose that our job partial ordering is that shown in Figure 5–29 and the time profile is that in Table 5–11. Then a partial solution tree is shown in Figure 5–30.

FIGURE 5–30 Part of a solution tree.


Level 0
(*, A, *)
Level 1
(B, C) (B, D) (B, E) (B, P) (C, D) (C, E) (C, P) (D, E) (D, P) (E, P)
Level 2
(E, P) (I, P) (I, E) (B, E) (B, P) (B, I)
Level 3
(F, G) (F, H) (F, I) (F, P) (G, H) (G, I) (G, P) (H, I) (H, P) (I, P)
Level 4
(F, H, L, P)
Level 5
(K, M, J, *, *)
Level 6
(N, O, *)
Level 7

* indicates an idle processor

The top of the tree is an empty node. From the time profile, we know that the maximum number of jobs that can be executed is three. However, since job A is the only job which can be executed at the very beginning, level 1 of the solution tree consists of only one node. After job A is executed, from the time profile, we know that two jobs may be executed now. Level 2 of the solution tree in Figure 5–30 shows all the possible combinations. Limited by space, we only show the descendants of nodes as a subset of nodes of the solution tree.

In the above paragraph, we showed how our job scheduling problem with time profile can be solved by the tree searching strategy. In the following sections, we shall show that in order to use the branch-and-bound strategy, we need to invent good branching and bounding rules, so that we avoid searching the entire solution tree.

The four rules below can be used by our branch-and-bound method. We shall not prove the validity of these rules. Instead, we shall only informally describe them.

RULE 1: Common successors effect

This rule can be informally explained by considering Figure 5–30 again. In this case, the root of the solution tree will be the node consisting of only one job, namely *A*. This node will have many immediate descendants and two of them are (C, E) and (D, P). As shown in Figure 5–29, jobs *C* and *D* share the same immediate descendant, namely job *I*. Similarly, *E* and *P* share the same descendant, namely job *J*. In this case, Rule 1 stipulates that only one node, either (C, E) or (D, P) needs to be expanded in the solution tree, because the length of any optimal solution headed by node (C, E) will be the same as that headed by node (D, P).

Why can we make this conclusion? Consider any feasible solution emanating from (C, E). Somewhere in the feasible solution, there are jobs *D* and *P*. Since *C* and *D* share the same descendant, namely job *I*, we can exchange *C* and *D* without changing the feasibility of the solution. By similar reasoning, we can also exchange *P* and *E*. Thus, for any feasible solution headed with (C, E), we can transform it into another feasible solution headed with (D, P), without changing the length of the solution. Thus, Rule 1 is valid.

RULE 2: Internal node first strategy

The internal node of the job precedence graph shall be processed earlier than the leaf node.

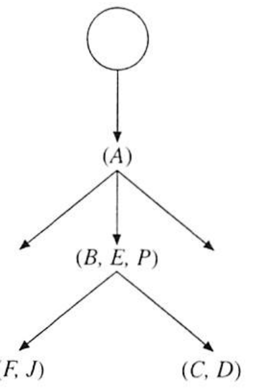
This rule can be informally explained by considering the job precedence graph in Figure 5–29 and the time profile in Table 5–12.

TABLE 5–12 A time profile.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
1	3	2	3	2	2	3	2	3

We show a part of the solution tree in Figure 5–31 for this case. After expanding node (B, E, P) of the solution tree, we have many immediate descendants for node (B, E, P) and two of them are (C, D) and (F, J) as illustrated in Figure 5–31.

FIGURE 5–31 A partial solution tree.



Since jobs C and D are internal nodes in the job precedence graph where F and J are leaf nodes, Rule 2 suggests that we should traverse node (C, D) and terminate node (F, J) for reasons similar to those used in explaining Rule 1.

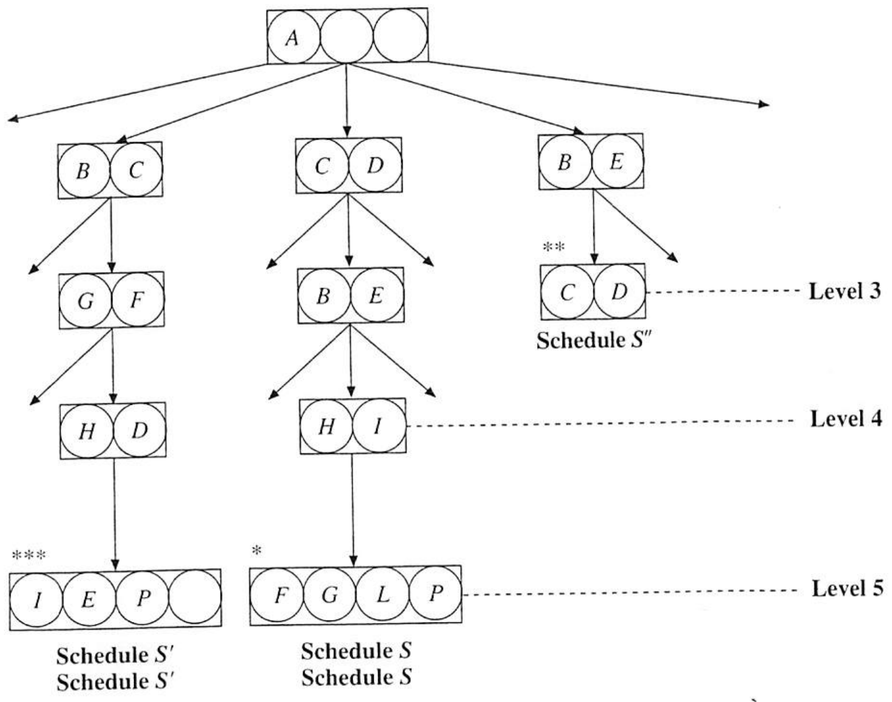
Rule 2 suggests that the candidate set should be split into two subsets, one for the active internal nodes and one for the active leaf nodes of the job precedence graph. The former has a higher priority to be processed. Since this set has a smaller size, it reduces the number of possible choices. The set of active leaf nodes will be chosen only when the size of the set of active internal nodes is smaller than the number of active processors. Since leaf nodes have no successors, it makes no difference how they are selected. Therefore, we may arbitrarily choose any group of them. As illustrated in Figure 5–30, after traversing node (B, E) at level 3, we have five jobs in the current candidate set (F, G, H, I, P) and there are two processors active. So we generate the total ten possible combinations. But, if we only consider the internal nodes, then we have only three nodes generated. They are (H, I), (H, P) and (I, P). The other seven nodes will never be generated.

RULE 3: Maximizing the number of processed jobs

Let $P(S, i)$ be a set of already processed jobs in the partially developed or completely developed schedule S, from time slot 1 to time slot i. Rule 3 can be stated as follows:

A schedule S will not be an optimal solution if P(S, i) is contained in P(S′, i) for some other schedule S′. Rule 3 is obviously correct. Figure 5–32 shows how Rule 3 can be used to terminate some nodes of the solution tree. For Figure 5–32, we claim that the schedules from node "***" and from "**" cannot be better than the schedule from "*". This is due to the fact $P(S'', 3) = P(S, 3)$ and $P(S', 5)$ is contained in $P(S, 5)$.

FIGURE 5–32 Processed jobs effect.



$\begin{cases} P(S'', 3) = (A, B, C, D, E) \\ P(S, 3) = (A, B, C, D, E) \end{cases}$

$P(S'', 3) = P(S, 3)$

$\begin{cases} P(S', 5) = (A, B, C, D, E, F, G, H, I, P) \\ P(S, 5) = (A, B, C, D, E, F, G, H, I, P, L) \end{cases}$

$P(S', 5)$ is contained by $P(S, 5)$

RULE 4: Accumulated idle processors strategy

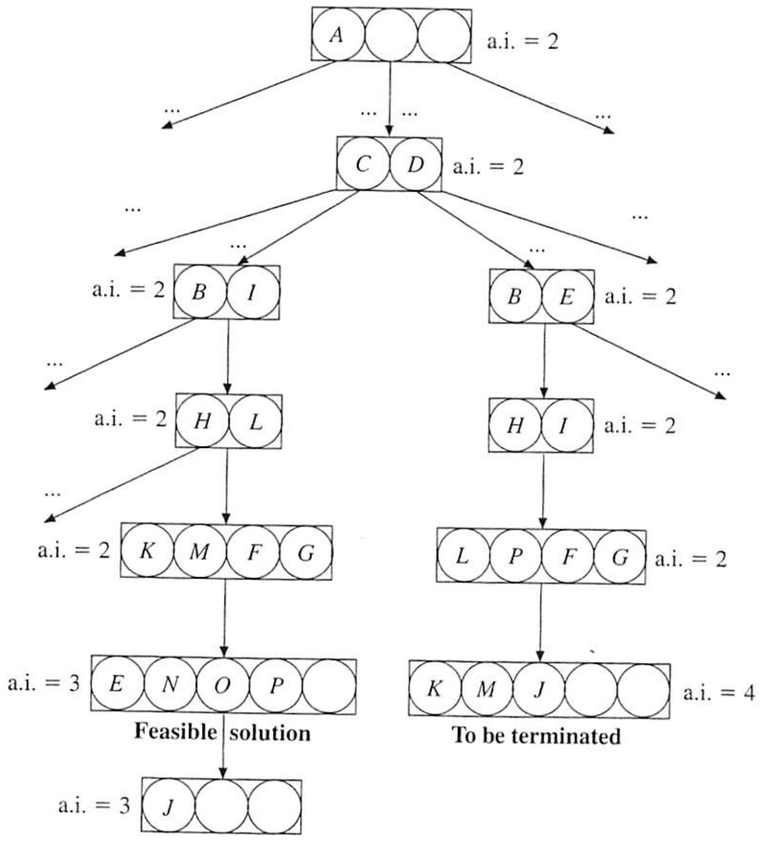
Rule 4 can be stated as follows:

Any partially developed schedule with the number of accumulated idle processors greater than that of a feasible schedule cannot be better than this feasible solution and can therefore be terminated.

Rule 4 shows that if we have already found a feasible solution, then we can use the total number of idle processors to terminate other nodes of the solution tree.

Consider Figure 5–33. Node (K, M, J) of the solution tree has the number of accumulated idle processors equal to 4 which is greater than that of the current feasible solution. So node $(K, M, J, *, *)$ is bounded by using Rule 4.

FIGURE 5–33 Accumulated idle processors effect.



5–10 *A** ALGORITHM

The *A** algorithm is a good tree searching strategy very much favored by artificial intelligence researchers. It is quite regretful that it is often ignored by algorithm researchers.

Let us first discuss the main philosophy behind the *A** algorithm. The best way to do this is to compare it with the branch-and-bound strategy. Note that in