Optimal Scheduling of Cooperative Tasks in a Distributed System Using an Enumerative Method

Dar-Tzen Peng and Kang G. Shin, *Fellow, IEEE*

***Abstract*— This paper considers preemptive (resume) scheduling of cooperative tasks that have been preassigned to a set of processing nodes in a distributed system, where each task is assumed to consist of several modules. During the course of their execution, these tasks communicate with each other to collectively accomplish a common goal. Such intertask communications lead to precedence constraints between the modules of different tasks. Our scheduling objective is to minimize the maximum normalized task response time, called the *system hazard.* Real-time tasks and the precedence constraints among them are expressed in a PERT/CPM form with activity on arc (AOA), called the *task graph* (TG), in which dominance relationship between simultaneously schedulable modules is derived and used to reduce the size of the set of active schedules to be searched for an optimal schedule. Lower-bound costs are estimated, which are then used to bound the search. Finally, a demonstrative example and some computational experiences are presented.**

***Index Terms*— Branch-and-bound (B&B) method, distributed real-time systems, dominance relation, lower-bound cost estimates, precedence constraints, preemptive (resume) scheduling of real-time tasks, regular performance measures, system hazard.**

I. Introduction

REAL-TIME systems are characterized by their requirement that the execution of their computational tasks must be not only logically correct but also completed in time; otherwise, catastrophe may ensue [1]. In a distributed real-time system, the tasks communicate with each other to accomplish the overall system goal. This paper addresses the "optimal" (in the sense to be defined later) preemptive scheduling of communicating real-time tasks that have already been preassigned to the processing nodes (PN's) of a distributed system. The scheduling objective used is to minimize the *system hazard,* or the maximum normalized task response time over all tasks. By properly selecting the normalization factor for each task, one can show that the system hazard subsumes many other scheduling criteria related to meeting task deadlines [2].

Let $T = \{T_i\colon i = 1, 2, \cdots, m\}$ be a set of $m \geq 2$ cooperative tasks to be executed by a set, $N = \{N_k\colon k = 1, 2, \cdots, n\}$, of $n \geq 2$ PN's. Using the method in [3], one can decompose each task T_i with release time r_i into several *modules* on the basis of intertask communications. Intertask communications introduce precedence constraints between the modules of different tasks. Each module of T_i must be executed by the PN to which T_i is assigned. There may also be a communication delay between the completion of a module and its subsequent enabling of another module on a different PN. The execution time of a module by the associated PN and each communication delay are assumed to be given. Worst-case values of these are usually used for the purpose of scheduling real-time tasks.

A PERT/CPM graph with Activity On Arc (AOA)[1] [4], called the *task graph* (TG), is used to describe communication delays and the precedence constraints between modules and the release and completion of each task. Unlike a general PERT/CPM graph—which usually has only a single pair of starting and ending nodes—the TG has multiple pairs of starting and ending nodes, each representing the release and the completion events of a task. Thus, a starting (ending) node is said to be "realized" when the corresponding task is released (completed). Note that the completion of a task T_i means the completion of all modules[2] and communication delays which precede T_i's ending node in the TG. Depending on how the TG is constructed to reflect T_i's completion, the set of these preceding modules may not contain all T_i's constituent modules. This means that even the last module(s) necessary to complete T_i may not belong to T_i. Hence, when solving the task scheduling problem (TSP), what is more important is which PN executes a given module and which ending node the module precedes in the TG, rather than which task the module belongs to. Following the convention in any PERT/CPM graph with AOA, each module or communication delay of the TG will henceforth be called an *activity.*

Fig. 1 shows an example TG consisting of four tasks T_1, T_2, T_3, and T_4. T_1 and T_2 are assigned to N_1, while T_3 and T_4 are assigned to N_2. Thus, all modules on the left-hand side (LHS) of the shaded areas in Fig. 1 are to be executed by N_1, whereas those on the right-hand side (RHS) to be executed by N_2. Each arc in the TG represents an activity and the weight associated with an arc is the execution time (magnitude) of the corresponding module (delay). In this TG, there are four starting nodes n_1, n_2, n_3, and n_{11}

Manuscript received January 22, 1990; revised October 26, 1992. This work was supported in part by the National Science Foundation under Grant DMC-8721492 and by the Office of Naval Research under Contract N00014-85-K-0122. Recommended by Simon Lam.

D.-T. Peng was with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122. He is now with the Microelectronics and Technology Center, Allied Signal Aerospace Company, Columbia, MD 21045-1998.

K. G. Shin is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9206896.

[1]The reason why the AOA-type—instead of the commonly used AON (Activity On Node) type—graph is used to describe a TG is that a node in an AOA-type TG corresponds more naturally to the readiness (completion) state of all modules immediately following (preceding) the node as can be seen below.

[2]Which may belong to T_i as well as to other tasks.

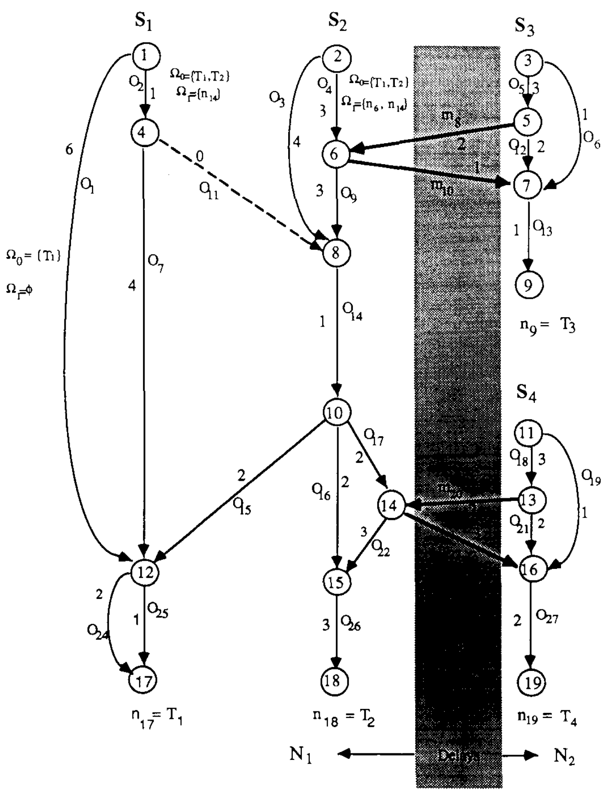


Fig. 1. An example TG.

(labeled S_1, S_2, S_3, and S_4, respectively), and four ending nodes n_{17}, n_{18}, n_9, and n_{19}, which represent the release and completion events of T_1, T_2, T_3, and T_4, respectively. A task is considered completed only after all activities preceding its ending node have been completed. All but T_4—which is released at time $t = 20$—are released at $t = 0$, meaning that n_1, n_2, n_3, and n_{11} are "realized" at $t = 0, 0, 0$, and 20, respectively. For convenience, we write $n_p = T_i$ if n_p is the ending node representing T_i's completion. Also, the completions of T_1, T_2, T_3, and T_4 represent the completions of all activities preceding n_{17}, n_{18}, n_9, and n_{19}, respectively.

Let c_i^ζ be the time when T_i is completed under a scheduling algorithm ζ. Then, the *system hazard* of a set T of tasks is defined as $\Theta^\zeta \triangleq \max_{T_i \in T} \bar{c}_i^\zeta$, where $\bar{c}_i^\zeta = (c_i^\zeta - r_i)/w_i$ is the normalized task response time, r_i the release time, and $w_i > 0$ the normalization factor of T_i. We want to find an optimal preemptive (resume) scheduling algorithm ζ^* for T such that $\Theta^{\zeta^*} = \min_\zeta \Theta^\zeta$. That is, ζ^* minimizes the maximum normalized response time over all tasks. Notice that Θ^ζ is a regular[3] performance measure.

For example, the two schedules generated by ζ_1 and ζ_2 for the four tasks in Fig. 1 are given in Fig. 2, where $w_1 = 30, w_2 = 40$, and $w_3 = w_4 = 20$. ζ_1 schedules modules[4] on each PN according to ascending order of their indices, whereas ζ_2 considers the normalization factor for each task as well as the precedence constraints that affect the scheduling of modules on other PN's. As shown in Fig. 2, we get

$$\bar{c}_1^{\zeta_1} = \frac{34}{30} \quad \bar{c}_2^{\zeta_1} = \frac{37}{40} \quad \bar{c}_3^{\zeta_1} = \frac{16}{20} \quad \bar{c}_4^{\zeta_1} = \frac{33-20}{20} = \frac{13}{20}$$

and

$$\bar{c}_1^{\zeta_2} = \frac{29}{30} \quad \bar{c}_2^{\zeta_2} = \frac{37}{40} \quad \bar{c}_3^{\zeta_2} = \frac{7}{20} \quad \bar{c}_4^{\zeta_2} = \frac{30-20}{20} = \frac{10}{20}.$$

Thus,

$$\begin{aligned}\Theta^{\zeta_1} &= \max\left\{\tfrac{34}{30}, \tfrac{37}{40}, \tfrac{16}{20}, \tfrac{13}{20}\right\} = \tfrac{34}{30} > \tfrac{29}{30} \\ &= \max\left\{\tfrac{29}{30}, \tfrac{37}{40}, \tfrac{7}{20}, \tfrac{10}{20}\right\} = \Theta^{\zeta_2}\end{aligned}$$

showing the ζ_2's superiority over ζ_1.

Even without considering the communication delays, the above scheduling problem is hard, regardless of whether or not preemptions are allowed. For example, the general job-shop scheduling problem [6]—a special case of the above scheduling problem—is already NP-hard for the problems even as simple as $T_2 \mid m_j \geq 3 \mid C_{\max}$[5] or $T_3 \mid m_j \geq 2 \mid C_{\max}$ [7], [8]. Consequently, except in case of a single PN [9], no polynomial time algorithms are likely to exist for our scheduling problem; some form of heuristic and/or enumeration is the only recourse to the problem.

The scheduling problem considered here falls in the realms of resource-constrained task scheduling [10] as well as job-shop scheduling, and thus, its solution approaches must be related to these two types of scheduling problems. Like any other NP-hard problem, approaches to these problems are concerned with how to improve the efficiency of search for an optimal or suboptimal solution by 1) using dominance properties (DP's) to reduce the size of the state space to be searched and 2) deriving a lower-bound cost as tight as possible at each stage to bound the search more efficiently. Giffler and Thompson [11] are the first to propose a systematic approach to generating the set of all *active schedules* based on which (and variations thereof) most implicit enumerative algorithms have been developed. A set A of active schedules is said to *dominate* another set $S\,(\supseteq A)$ of all schedules in the sense that inclusion of an optimal schedule in S implies the inclusion of the same schedule in A. In other words, to find optimal schedules with respect to (w.r.t) any regular measure, it suffices to consider only the set of active schedules, thus reducing the size of the state space to be searched. For example, to solve a job-shop scheduling problem, Brooks and White [12] used two bounds, called the *job bound* and the *machine bound,* to bound the search for an optimal schedule among the set of all active schedules. Schrage [13], [14] developed a similar approach to generating the set of all active schedules for preemptive and nonpreemptive cases, and proposed DP's and lower-bound costs to find optimal schedules for a network scheduling problem. An extensive survey of job-shop scheduling with branch-and-bound (B&B)

[3]A performance measure Z is said to be *regular* if 1) the scheduling objective is to minimize Z, and 2) Z increases only if one or more task completion times in the schedule increases [5].

[4]Time delays are not schedulable objects.

[5]A two-machine job-shop in which the number of operations in any job is greater than or equal to 3 and the scheduling objective is to minimize the maximum job completion time among all jobs [15].

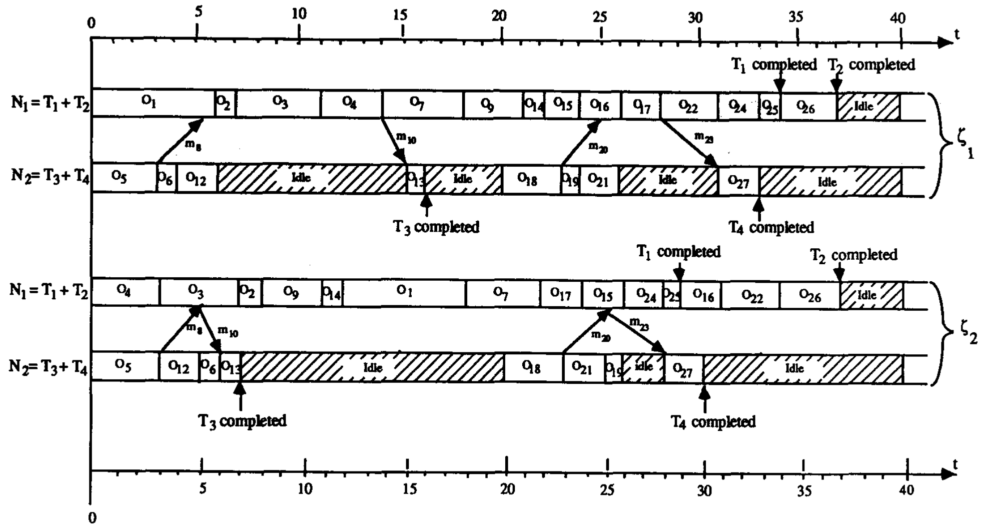


Fig. 2. Two schedules of the TG in Fig. 1.

methods can be found in [16], where job-shop scheduling was modeled by settling pairs of disjunctive arcs and a tighter bound of cost was also developed by including many other bounds as special cases. Possible extensions of the problems and variations of the solution techniques are described in [6].

Unfortunately, none of these approaches are directly applicable to our scheduling problem due mainly to the structural differences. Furthermore, while most known scheduling objectives are to minimize the makespan or to meet task deadlines, our scheduling objective is to minimize the system hazard. As stated in [2], there are distinct advantages in using the system hazard over other objectives. For example, if all tasks to be scheduled are periodic, then minimizing the makespan is not appropriate unless all task periods are the same. On the other hand, an objective simply to meet all task deadlines may not be good enough because one such schedule may still be better than another in the following sense. Suppose there are only two dependent periodic tasks T_1 and T_2 to be scheduled, where the periods of T_1 (to be scheduled on N_1) and T_2 (to be scheduled on N_2) are 10 and 20, respectively. Consider two schedules ζ_1 and ζ_2 of these two tasks in the time period [0, 20). In ζ_1, T_2 is completed at time 12, and the two invocations of T_1 are completed at times 9.9 and 14, respectively. In ζ_2 however, T_2 is completed at time 14, and the two invocations of T_1 are completed at times 6 and 16, respectively. Since all deadlines are met, both ζ_1 and ζ_2 are optimal schedules if the objective is simply to meet all task deadlines. However, it is reasonable to argue that ζ_2, which with a lower system hazard than ζ_1 ($0.7 < 0.99$), should be superior to ζ_1 because a slight increase in T_1's and/or T_2's execution time may cause T_1 to miss one of its deadlines in ζ_1, but not in ζ_2. Hence if the normalization factors are chosen to be the task periods as in the above example, then using the system hazard maximizes the system's ability to meet task deadlines and allows for better load sharing in the system. This is especially true when variations in task execution times are possible, and the variances are proportional to the task flow times.

In this paper, we develop a B&B algorithm to find an optimal schedule w.r.t. the system hazard. We do this by deriving and then using the dominance relationship between simultaneously schedulable modules. Also, lower-bound costs are derived to effectively guide the search for an optimal schedule. Although the general structure of B&B algorithms is well known, it is the derivation of good DP's and tight lower-bound costs that are essential to the success of any B&B algorithm.

A set of DP's w.r.t. all regular measures are identified in Section II. Section III discusses more DP's w.r.t. the system hazard. Using all the DP's derived in Sections II and III, we show in Section IV how the B&B algorithm is guided by a tight lower-bound cost to efficiently find an optimal schedule. A demonstrative example and some computational experiences are presented in Section V. Finally, the paper concludes with Section VI.

II. Dominance Properties for All Regular Measures

Since preemptions create an infinite number of possible schedules for a given problem, the main idea behind the DP's w.r.t. *all* regular measures is to eliminate unnecessary preemptions and reduce the processor idle times caused by precedence constraints between modules.

An unfinished module O_b in the TG is said to be *schedulable* if 1) the task containing O_b has been released, and 2) all of O_b's preceding modules and communication delays have already been completed. The DP's to be identified are based on the following observations:

OB1: Preemptions that do not improve performance must be disallowed to reduce the number of possible branches in the B&B search tree.

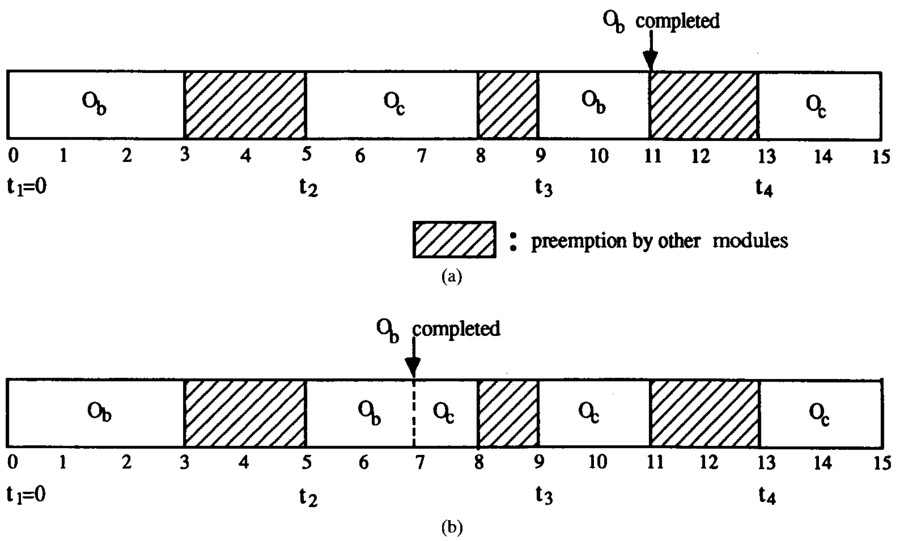


Fig. 3. Two schedules generated by ζ_1 and ζ_2. (a) Original partial schedule by ζ_1. (b) Alternative partial schedule by ζ_2.

OB2: A PN should not be left idle if there are modules schedulable on the PN.

OB3: A schedule must always be replaced by another schedule if the latter can reduce the completion time of a task without delaying the completion of any other task.

These observations are obvious since preemptions are allowed and regular measures used.

Let O_b and O_c be two modules schedulable on N_k, ζ_1 and ζ_2 be two preemptive scheduling algorithms, and Z^{ζ_1} and Z^{ζ_2} be the performances of ζ_1 and ζ_2, respectively. OB1 leads to the following theorem w.r.t. any regular measure.

Theorem 1: Given any schedule with preemptions between O_b and O_c, there always exists another schedule without such preemptions such that the new schedule achieves a cost equal to, or lower than, that of the original schedule.

Proof: Let O_b and O_c be simultaneously schedulable on N_k at $t_1 = 0$. Consider the partial schedule in Fig. 3(a) generated by a scheduling algorithm ζ_1, where the shaded areas represent preemptions by modules other than O_b and O_c. As shown in Fig. 3(a), ζ_1 allows O_c to interrupt O_b at $t_2 > t_1$. Without loss of generality, we may assume that O_b is completed before O_c. We want to show that there always exists another scheduling algorithm ζ_2 that does not allow preemptions between O_b and O_c and achieves an equal or lower cost as compared to the original algorithm ζ_1.

Since O_b is completed before O_c in the original schedule, construct the new schedule with ζ_2, which simply rearranges the execution orders of O_b and O_c such that O_b is executed to completion without O_c's interruptions after t_1 (Fig. 3(b)). (O_b could still be interrupted by modules other than O_c as shown in the shaded areas in Fig. 3(b).) It is shown that the completion time of O_b has been reduced while that of O_c can be maintained. Thus, using ζ_2 will never result in a larger task completion time than using ζ_1 because there are always more schedulable modules available to ζ_2 than ζ_1. Similar arguments also hold if O_c is assumed to complete before O_b in the original schedule. In such a case, however, ζ_2 must rearrange the execution order of O_b and O_c such that O_c is now executed to completion without O_b's preemptions. Notice that which of O_b and O_c to execute first at $t_1 = 0$ under ζ_1 for both of the above cases makes no difference. The theorem follows since $Z^{\zeta_2} \leq Z^{\zeta_1}$ by OB3. Q.E.D.

For the purpose of disallowing unnecessary preemptions, the implications of Theorem 1 are as follows. If there are r modules $M = \{O_j | j = 1, 2, \cdots, r\}$ that are simultaneously schedulable on N_k at $t_1 = 0$, then for each of these r branches we are expanding, the corresponding module shall be allowed to run to completion unless modules other than those in M become schedulable at $t > t_1$. In other words, for each branch j under expansion, no module in M is allowed to preempt the execution of the module O_j corresponding to branch j. Only those modules that become schedulable at a future time $t > t_1$ may preempt O_j. In the following, the DP's are identified based on OB2 and OB3.

While the term "precede" holds its usual meaning between two nodes or two arcs in the TG, we also want to use it between a node and an arc as follows.

Definition 1: An activity (module or communication delay) A_x is said to ***immediately precede*** a node n_p if n_p is at the "head" of A_x in the TG, written as ***head***$(A_x) = n_p$, and A_x is said to ***precede*** n_p if either A_x immediately precedes n_p, or ***head***(A_x) precedes n_p in the TG.

Therefore, n_p is said to be ***realized*** if a task whose starting event is represented by n_p has been released, and all activities that immediately precede n_p have been completed.

Since the precedence relation in the TG is ***transitive*** [17], there are usually many nodes preceded by a module, say O_x, although only one of them may be immediately preceded by O_x. Besides, some of the nodes preceded by O_x may not even be "located" in the same PN, say N_k, that executes O_x. Let $\Omega_0(O_x)$ be the set of nodes which are preceded by O_x, are

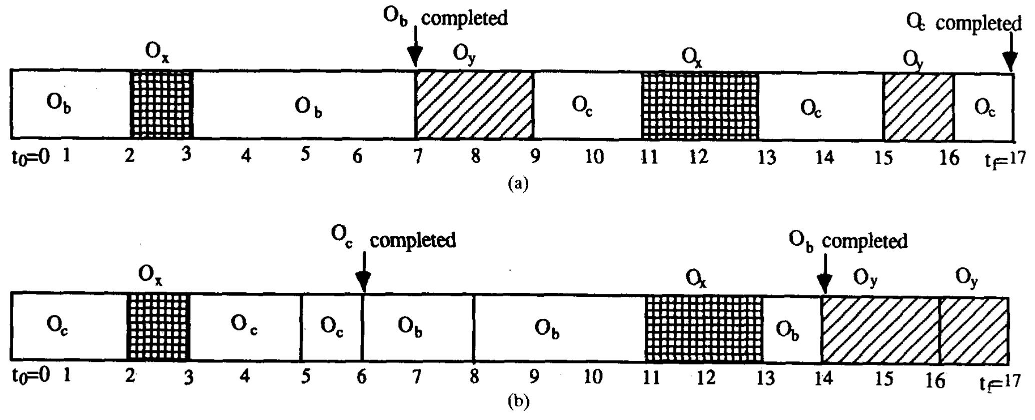


Fig. 4. Two partial schedules generated by ζ_1 and ζ_2. (a) Original partial schedule μ_1 generated by ζ_1. (b) Modified partial schedule μ_2 generated by ζ_2.

indeed located in N_k, and represent some task completions. Also, let $\Omega_1(O_x)$ be the set of PN "boundary" nodes that are preceded by O_x, are located in N_k, and are placed at the "tails" of some communication delays. For the example TG in Fig. 1, $\Omega_0(O_2) = \{n_{17}, n_{18}\} = \{T_1, T_2\}$ and $\Omega_1(O_2) = \{n_{14}\}$ since n_{14} is a boundary node of N_1, preceded by O_2 and located at the tail of communication delay m_{23}. Likewise, $\Omega_0(O_1) = \{n_{17}\} = \{T_1\}$ and $\Omega_1(O_1) = \emptyset$. Moreover, $\Omega_0(O_4) = \{T_1, T_2\}, \Omega_1(O_4) = \{n_6, n_{14}\}$, and $\Omega_0(O_5) = \{T_3, T_4\}, \Omega_1(O_5) = \{n_5\}, \Omega_0(O_6) = \{T_3\}, \Omega_1(O_6) = \emptyset, \Omega_0(O_{18}) = \{T_4\}$ and $\Omega_1(O_{18}) = \{n_{13}\}$. Note that $\Omega_0(O_2)$ does not contain T_4 since n_{19} is not located in N_1. Likewise, none of T_1 and T_2 are contained in $\Omega_0(O_5)$. $\Omega_0(O_x)$ and $\Omega_1(O_x)$ indicate which and how tasks will benefit from the completion of O_x. Specifically, $\Omega_0(O_x)$ represents the set of tasks on a PN, say N_k, whose completions must be preceded by that of O_x. On the other hand, tasks on PN's other than N_k can only benefit from the completion of O_x through the realization of nodes in $\Omega_1(O_x)$.

Based on the above observation and given any two modules O_b and O_c simultaneously schedulable on N_k, we write $O_c \Rightarrow^Z (=^Z) O_b$ if $\Omega_j(O_c) \supseteq (=) \Omega_j(O_b), j = 0, 1$. Notice that if O_b precedes O_c then $O_b \Rightarrow^Z O_c$, but the converse is not necessarily true. The relation $\Rightarrow^Z$ is transitive and $=^Z$ is an equivalence relation. Besides, $O_b =^Z O_c$ iff $O_b \Rightarrow^Z O_c$ and $O_c \Rightarrow^Z O_b$.

Definition 2: Let ζ_1 be an algorithm that schedules O_b before O_c on N_k at time t_0. It is said to be *advantageous* w.r.t. a regular measure Z for N_k to execute O_c before O_b at t_0 if for any such ζ_1, there always exists another algorithm ζ_2 which contains the ζ_1's partial schedule prior to t_0, schedules O_c before O_b at t_0, and satisfies $Z^{\zeta_2} \leq Z^{\zeta_1}$.

Based on OB2 and OB3, the following theorem associates the relation $\Rightarrow^Z$ between O_b and O_c with the N_k's scheduling decision.

Theorem 2: Given any regular measure $Z, O_c \Rightarrow^Z O_b$ implies that it is always advantageous for N_k to execute O_c before O_b, where O_b and O_c are modules simultaneously schedulable on N_k.

Proof: Suppose ζ_1 is a scheduling algorithm under which N_k executes O_b before O_c at time t_0. Consider the partial schedule μ_1 within the time interval $[t_0, t_f]$ generated by ζ_1, where t_0 is the time both O_b and O_c are schedulable, and t_f the time O_c is completed (Fig. 4(a)). In addition to O_b and O_c, there are only two other types of modules that could possibly be executed by N_k within $[t_0, t_f]$: 1) those modules that have no precedence relation with either of O_b and O_c, and 2) those modules that are preceded by O_b but has no precedence relation with O_c. Modules of type 1) can preempt O_b and/or O_c at any time within $[t_0, t_f]$, whereas those of type 2) can only be executed after O_b's completion, but may preempt O_c at any time within $[t_0, t_f]$. Let O_x and O_y denote, respectively, the representative modules of these two types.

Construct a new partial schedule μ_2 for N_k within $[t_0, t_f]$ as follows. While keeping the schedule for all O_x's unchanged, rearrange all the other modules to be executed in order of O_c, O_b, and O_y's (Fig. 4(b)). μ_2 is feasible since all precedence constraints are still met within $[t_0, t_f]$. Given $\Omega_0(O_c) \supseteq \Omega_0(O_b)$ and $\Omega_1(O_c) \supseteq \Omega_1(O_b)$, we want to show that there exists another scheduling algorithm ζ_2 that contains μ_2 as well as ζ_1's partial schedule prior to t_0, and satisfies the inequality $Z^{\zeta_2} \leq Z^{\zeta_1}$.

Let $\Delta_0 = \Omega_0(O_c) - \Omega_0(O_b), \delta_1 = \Omega_1(O_c) - \Omega_1(O_b)$, and Δ_1 be the set of tasks located in nodes other than N_k, the completion event of each of which is preceded by at least a node in δ_1. Also, let $\Delta = \Delta_0 \cup \Delta_1$ and $\overline{\Delta} = T - \Delta$, where T is the set of all tasks. That is, $\overline{\Delta}$ represents the set of tasks preceded by both, or neither, of O_b and O_c, while Δ contains the set of tasks preceded by O_c and, possibly, by both O_c and O_b. We now separately compare the possible completion times of tasks in Δ and those in $\overline{\Delta}$ when either μ_1 or μ_2 is N_k's partial schedule within $[t_0, t_f]$. Because of the way μ_2 is constructed, O_c has a smaller completion time under μ_2, but O_b and O_y's will have larger completion times as compared to the case of using μ_1. Since O_b precedes $O_y, O_b \Rightarrow^Z O_y$, and thus, $O_c \Rightarrow^Z O_y$. That is, any task whose completion event preceded by O_y and/or O_b is also preceded by O_c, meaning that the preceded task cannot be completed without completing O_y's, O_b, and O_c anyway. This implies that there exist a schedule ζ_2 containing both ζ_1's prior partial schedule and μ_2, under which the completion times of tasks in Δ could be reduced further, without increasing those in $\overline{\Delta}$. Since these arguments are valid for any time t_0, the theorem follows by OB3. Q.E.D.

For the example TG in Fig. 1, since $\Omega_0(O_4) = \Omega_0(O_2) \supseteq \Omega_0(O_1)$ and $\Omega_1(O_4) \supseteq \Omega_1(O_2) \supseteq \Omega_1(O_1), O_4 \Rightarrow^Z O_2 \Rightarrow^Z O_1$. Therefore, it is advantageous for N_1 to execute these three modules in order of O_4, O_2 and O_1 (with the possibility that some other modules such as O_3 may be executed in between them).

It is worth pointing out implications of Theorem 2. First, the partial schedules μ_1 and μ_2 in Theorem 2 were constructed while implicitly honoring Theorem 1. Second, $\Omega_2(O_c) \supseteq \Omega_2(O_b)$ does not necessarily imply that it is advantageous for N_k to execute O_c before O_b, where $\Omega_2(O_x)$ is the set of *all* tasks preceded by O_x. Third, if it is advantageous at t_0 for N_k to execute O_b before O_c and vice versa, then it makes no difference as to which of O_b and O_c is executed first at t_0. In other words, to search for an optimal schedule, it suffices at t_0 to consider only the case where either of O_b and O_c is executed first.

The DP in Theorem 2 is based only on the property expressed in the form of $\Rightarrow^Z$, which does not always exist between simultaneously schedulable modules. It is very difficult to derive any finer-grain DP's other than the above for *all* regular measures. However, if the problem is restricted to a specific regular measure, some finer DP's w.r.t. that measure can be derived. In Section III we derive such DP's w.r.t. Θ, which will then be used to simplify the search further in our B&B algorithm.

III. Dominance Properties w.r.t. System Hazard

The approach to determining which of O_b and O_c to be scheduled first w.r.t. Θ is equivalent to determining the relative "urgency" of each task in $\Omega_0(O_b) \cup \Omega_0(O_c)$. Before proceeding any further, it is necessary to introduce an optimal *single-machine* scheduling algorithm developed by Baker *et al.* [9], which we call *Algorithm A* in the rest of this paper. A set of jobs with arbitrary release times and precedence constraints is to be scheduled on a single machine so as to minimize the maximum job completion cost, where the cost associated with each job can be any monotone nondecreasing function of its completion time. As can be seen from the following steps, the computational complexity of Algorithm A is $O(N^2)$, where N is the number of jobs to be scheduled.

SA1: Modify job release times, where possible, to meet the precedence constraints among the jobs, and then arrange the jobs in nondecreasing order of their modified release times to create a set of disjoint blocks of jobs. For example, suppose jobs X, Y, and Z are released at $t = 0, 2, 15$, respectively, X precedes Y which precedes Z, and 5 units of time are required to complete each of X and Y. Then, the job Y's release time is modified to $t = 5$, Z's release time remains unchanged at $t = 15$, and two blocks of jobs {X, Y } and {Z} will then be created.

SA2: Consider a block B with block completion time $t(B)$. Let B' be the set of jobs in B that do not precede any other jobs in B. Select a job l from B' such that $f_l(t(B))$ is the minimum, where $f_l(t)$ is the nondecreasing cost function of job l if it is completed at t. This implies that l be the last job to be completed in B.

SA3: Create subblocks of jobs in the set $B - \{l\}$ by arranging the jobs in nondecreasing order of modified released times as in SA1. (If l is preempted several times before its completion, the deletion of l is equivalent to punching several holes in B.) The time interval(s) alloted to l is (are) then the difference between the interval of B and the interval(s) allotted to these subblocks.

SA4: For each subblock, repeat SA2 and SA3 until time slot(s) is (are) allotted to every job.

Our scheduling problem is much more complicated than that treated by Algorithm A, since two modules belonging to two different tasks could be assigned to two different PN's and must meet the precedence constraints between them. However, the following useful lemma can be derived from SA2.

Lemma 1: Let O_b and O_c be two modules simultaneously schedulable on N_k at time t_0 such that $\Omega_1(O_c) \supseteq \Omega_1(O_b), \Omega_0(O_b) = \{T_m\}$, and $\Omega_0(O_c) = \{T_n\}$, for some tasks $T_m, T_n \in T$. Then, it is advantageous w.r.t. Θ to execute O_c before O_b at t_0 if the following inequality (see Fig. 5) holds:

$$\frac{t - r_n}{w_n} \geq \frac{t - r_m}{w_m} \qquad \forall t \geq t_0 + R \tag{1}$$

where R is the sum of the remaining execution times of all unfinished modules on N_k that precede at least one of T_m and T_n.

Proof: Since $\Omega_1(O_c) \supseteq \Omega_1(O_b)$, the lemma follows if we can show that executing O_c before O_b is advantageous for T_m and T_n. Consider again the partial schedule μ_1 in Fig. 4(a) generated by ζ_1, and the modified partial schedule μ_2 in Fig. 4(b) generated by ζ_2, where ζ_2, in this case, differs from ζ_1 only in μ_1 and μ_2. In both μ_1 and μ_2, recall that O_x's represent those modules that can preempt O_b and O_c at any time within $[t_0, t_f]$, whereas O_y's are those modules that can be executed only after the O_b's completion but can preempt O_c at any time within $[t_0, t_f]$. To prove that $\Theta^{\zeta_2} \leq \Theta^{\zeta_1}$, four cases must be considered depending on whether or not the completions of O_y and O_c represent those of T_m and T_n, respectively.

Case 1: Neither O_y nor O_c is a *completing module* whose completion represents the completion of T_m or T_n. In this case, all task completion times are the same under ζ_1 and ζ_2, thus $\Theta^{\zeta_2} = \Theta^{\zeta_1}$.

Case 2: The completion of T_m is represented by that of O_y, but O_c is not a completing module. The normalized response time of T_m under ζ_2 is larger than that under ζ_1 while that of T_n remains unchanged. However, by (1), the normalized response time of T_n is larger than that of T_m under ζ_2 which, in turn, is larger than that of T_m under ζ_1. This implies that the maximum of these remain unchanged regardless whether ζ_2 (in place of ζ_1) is used or not. Thus, $\Theta^{\zeta_2} = \Theta^{\zeta_1}$.

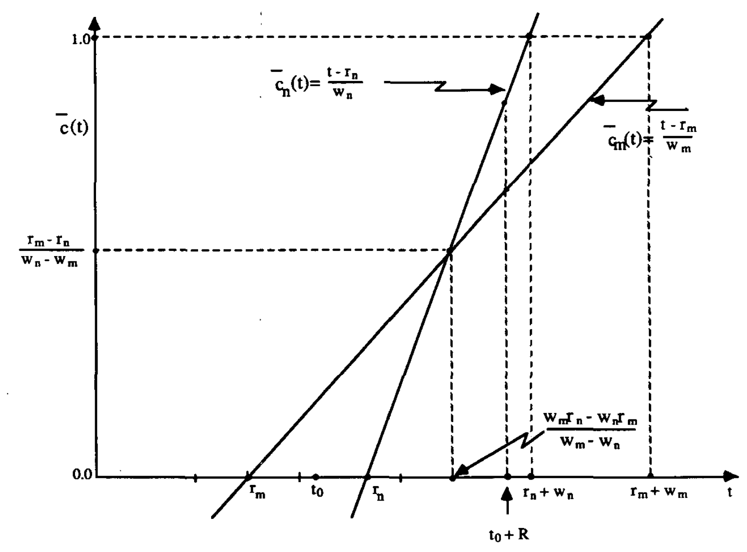


Fig. 5. An example cost structure of T_m and T_n.

Case 3: O_c is a completing module, but O_y is not. The normalized response time of T_n is thus reduced while that of T_m remains unchanged under ζ_2 when compared to ζ_1, thereby making $\Theta^{\zeta_2} \leq \Theta^{\zeta_1}$.

Case 4: Both O_y and O_c are completing modules. From (1) and SA2 of Algorithm A, it is advantageous to execute O_c before O_b (and O_y), thus $\Theta^{\zeta_2} \leq \Theta^{\zeta_1}$. Q.E.D.

Notice that in the above proof, O_y, rather than O_b, is checked; if such an O_y is nonexistent (O_b itself is a completing module of T_m), then we only need to replace O_y with O_b in the proof. Also, it is important to point out that Lemma 1 always holds no matter how many such T_m's are in $\Omega_0(O_b)$. Specifically, suppose 1) $\Omega_1(O_c) \supseteq \Omega_1(O_b)$ and 2) $\Omega_0(O_b) = \{T_{m_1}, T_{m_2}, \cdots, T_{m_q}\}$, and $\Omega_0(O_c) = \{T_n\}$, for some tasks $T_{m_1}, T_{m_2}, \cdots, T_{m_q}, T_n \in T$. Then, it is still advantageous to execute O_c before O_b at t_0 if (1) holds for each pair of T_n and $T_{m_j}, 1 \leq j \leq q$. The proof is similar to that of Lemma 1, except that we now have to consider the completion event of each of T_{m_j}. It can be seen that, in each of the four cases considered, $\Theta^{\zeta_2} \leq \Theta^{\zeta_1}$ still holds because the system hazard represents the maximum, rather than the sum, of a set of numbers. As we shall see later, this fact is essential in proving Theorem 3 below. Finally, as is shown in Fig. 5, t_0 is not restricted to the time after both T_m and T_n are released; t_0 can be any time before either T_m or T_n is completed.

Equation (1) holds if and only if the following inequality holds (see Fig. 5):

$$t_0 + R \geq (w_m r_n - w_n r_m)/(w_m - w_n) \quad \text{if } w_m > w_n$$

or

$$r_m \geq r_n \quad \text{if } w_m = w_n \tag{2}$$

where the RHS of the first inequality represents a particular time t such that $(t - r_m)/w_m = (t - r_n)/w_n$. It may be noted that (2) conforms to the various optimal policies known in scheduling theory. This fact leads to the following definition of the "relative superiority" (or urgency) w.r.t. Θ between two unfinished tasks T_m and T_n.

Definition 3: For two unfinished tasks T_m and T_n on N_k, T_n is said to be 1) *superior* to T_m w.r.t. Θ, written as T_n $\mathbf{sp}^{\Theta}$ T_m, at t_0 if (2) holds, and 2) *equal* to T_m w.r.t. Θ, written as T_n $\mathbf{eq}^{\Theta}$ T_m, if $w_m = w_n$ and $r_m = r_n$.

T_n $\mathbf{sp}^{\Theta}$ T_m is defined only when $w_n \leq w_m$ and that T_n $\mathbf{eq}^{\Theta}$ T_m iff T_n $\mathbf{sp}^{\Theta}$ T_m and T_m $\mathbf{sp}^{\Theta}$ $T_n \forall t_0$. Notice that while $\mathbf{eq}^{\Theta}$ is an equivalence relation, $\mathbf{sp}^{\Theta}$ is *not* transitive, meaning that T_n $\mathbf{sp}^{\Theta}$ T_m and T_m $\mathbf{sp}^{\Theta}$ T_p do not necessarily imply T_n $\mathbf{sp}^{\Theta}$ T_p. Furthermore, if T_n $\mathbf{sp}^{\Theta}$ T_m at t_0, then T_n $\mathbf{sp}^{\Theta}$ T_m at all $t_1 \geq t_0$. Based on the superiority relation between two tasks, we have the following definitions on their preceding modules O_b and O_c.

Definition 4: O_c is said to

1) *dominate* O_b w.r.t. Θ, written as $O_c \Rightarrow^{\Theta} O_b$, at t_0 if (a) for every $T_m \in \Omega_0(O_b)$ there exists a $T_n \in \Omega_0(O_c)$ such that T_n $\mathbf{sp}^{\Theta}$ T_m at t_0 and (b) $\Omega_1(O_c) \supseteq \Omega_1(O_b)$, and
2) be *similar* to O_b w.r.t. Θ, written as $O_c\,\mathbf{S}^{\Theta}\,O_b$, at t_0 if $O_c \Rightarrow^{\Theta} O_b$ and $O_b \Rightarrow^{\Theta} O_c$ at t_0.

Definition 4 specifies the relative urgency of a schedulable module O_x in term of those of the tasks in $\Omega_0(O_x)$ and the tasks preceded by $\Omega_1(O_x)$. It is useful to examine the properties of $\Rightarrow^{\Theta}$. First, $O_c \Rightarrow^{Z} O_b$ implies $O_c \Rightarrow^{\Theta} O_b$ at any time $t_0 \geq 0$, but the converse is not always true. In particular, it is even possible that both $O_c \Rightarrow^{\Theta} O_b$ and $\Omega_0(O_c) \subset \Omega_0(O_b)$ hold. Second, $\Rightarrow^{\Theta}$ is not transitive, and thus, neither is $\mathbf{S}^{\Theta}$. Finally, if $O_c \Rightarrow^{\Theta} O_b$ at t_0, then $O_c \Rightarrow^{\Theta} O_b$ at any time $t_1 \geq t_0$.

Based on the discussions thus far, the DP's between O_b and O_c are summarized in the following theorem.

Theorem 3: Let O_b and O_c be two modules schedulable on N_k. Then, w.r.t. Θ,

1) it is advantageous to execute O_c before O_b at t_0 if $O_c \Rightarrow^{\Theta} O_b$ at t_0, and
2) it makes no difference as to which of O_b and O_c is executed first at t_0 if $O_c \mathbf{S}^{\Theta} O_b$ at t_0.

Proof of 1): Since $O_c \Rightarrow^{\Theta} O_b$ at t_0, for every task $T_m \in \Omega_0(O_b)$ there exists a corresponding $T_n \in \Omega_0(O_c)$ such that (1) and (2) hold. From Lemma 1, executing O_c before O_b at t_0 can reduce the maximum normalized response times[6] of such T_m's and T_n's. The completion time of every task other than such T_m's and T_n's does not increase because $\Omega_1(O_c) \supseteq \Omega_1(O_b)$.

Proof of 2): From the definition of $\mathbf{S}^{\Theta}$ and the result of Part 1), the proof directly follows. Q.E.D.

From the discussion of Definition 4, there are cases where both $O_c \Rightarrow^{\Theta} O_b$ and $O_b \Rightarrow^{Z} O_c$ hold. This simply indicates that executing O_c before O_b (Theorem 3) is just as good as executing O_b before O_c (Theorem 2). Although Theorem 3 is useful, its implementation is not as straightforward as Theorem 2, because neither $\Rightarrow^{\Theta}$ nor $\mathbf{S}^{\Theta}$ is transitive. Corollaries 1 and 2 below indicate that, even though neither $\Rightarrow^{\Theta}$ nor $\mathbf{S}^{\Theta}$ is transitive, the orders of executing modules implied by these two relations are transitive.

Corollary 1: Let $S_k(t_0) = \{O_j\colon 1 \le j \le s\}$ be a subset of $s \ge 2$ modules schedulable on N_k at time t_0.

1) If $O_s \Rightarrow^{\Theta} O_{s-1}, O_{s-1} \Rightarrow^{\Theta} O_{s-2}, \cdots$, and $O_2 \Rightarrow^{\Theta} O_1$ at t_0, then it is advantageous for N_k to execute O_s before O_1 at t_0.
2) If $O_1 \Rightarrow^{\Theta} O_s$ in addition to the condition in (a), then it makes no difference as to which module in $S_k(t_0)$ is executed first.

(The proof follows directly from Theorem 3 and Definition 2, and thus, is omitted.)

Corollary 2: Let $S_k(t_0)$ be the same as in Corollary 1. If $O_s\, \mathbf{S}^{\Theta}\, O_{s-1}, O_{s-1}\, \mathbf{S}^{\Theta}\, O_{s-2}, \cdots$, and $O_2\, \mathbf{S}^{\Theta}\, O_1$ at t_0, then it makes no difference as to which module in $S_k(t_0)$ is executed first. (The proof follows directly from Part 2) of Theorem 3 and thus is omitted.)

It is worth pointing out that $O_s \Rightarrow^{\Theta} O_{s-1}, O_{s-1} \Rightarrow^{\Theta} O_{s-2}, \cdots, O_2 \Rightarrow^{\Theta} O_1$, and $O_1 \Rightarrow^{\Theta} O_s$ do not imply that $O_s\, \mathbf{S}^{\Theta}\, O_{s-1}, O_{s-1}\, \mathbf{S}^{\Theta}\, O_{s-2}, \cdots$, and $O_2\, \mathbf{S}^{\Theta}\, O_1$ at t_0; the converse does not hold either. For convenience, the set of schedulable modules for which execution order is immaterial is called an *immaterial set* (IM). As we shall see in Section IV, knowledge of an IM of size as large as possible greatly simplifies the search for an optimal schedule. An important property associated with two IM's is given in the following corollary.

Corollary 3: Let $\Pi_k^1(t_0) = \{O_1^1, O_2^1, \cdots, O_{s_1}^1\}$ and $\Pi_k^2(t_0) = \{O_1^2, O_2^2, \cdots, O_{s_2}^2\}$ be two distinct IM's of size s_1 and s_2 on N_k at t_0. If there exist $O_b \in \Pi_k^1(t_0)$ and $O_c \in \Pi_k^2(t_0)$ such that executing O_c before O_b at t_0 is advantageous, then it is advantageous to execute O_j^2 before O_i^1 at $t_0, \forall i, j$.

Proof: By assumption, it is advantageous at time t_0 to execute O_j^2 before O_c, O_c before O_b, and O_b before O_i^1. Since O_j^2 is arbitrarily chosen from $\Pi_k^2(t_0)$ and O_i^1 from $\Pi_k^1(t_0)$, the corollary follows. Q.E.D.

Because $O_c \Rightarrow^{\Theta} (\mathbf{S}^{\Theta}) O_b$ at t_0 implies the same relation at any $t_1 \ge t_0$. Corollary 3 simply says that it is advantageous to execute all modules in $\Pi_k^2(t_0)$ before any module in $\Pi_k^1(t_0)$. Corollary 3—which deals with the "uninterruptability" of an immaterial set—can be thought of as another version of Theorem 1, which deals with the uninterruptability of a single module.

It can be seen that the DP's in Theorem 3 are identified only under the condition $\Omega_1(O_c) \supseteq \Omega_1(O_b)$. Without this condition, it is very difficult to find any DP useful for our scheduling problem because of the interdependencies between scheduling decisions on different PN's. In Section IV we show how the DP's presented so far are used in the B&B algorithm to find an optimal schedule.

IV. Search for an Optimal Schedule with a B&B Algorithm

The proposed B&B algorithm is described in terms of its two phases: *branching* and *bounding*. The branching process expands an active parent vertex[7] to generate child vertices while the bounding process derives a lower-bound cost for each child vertex to guide the search [18]. In Section IV-A, we show how the DP's derived thus far are incorporated into the B&B algorithm for efficient branching. Lower-bounds of system hazard are then derived and used in Section IV-B for efficient bounding.

A. Generation of a Small Set of Schedules Using DP's

As mentioned earlier, the set of active nonpreemptive schedules contains all optimal schedules w.r.t. any regular measure. Thus, to minimize any regular measure, it is sufficient to consider only this set of active schedules. The above statement is true for nonpreemptive scheduling problems that may be as difficult as the job-shop scheduling problems. In this paper, however, we are concerned with the derivation of an optimal preemptive, rather than nonpreemptive, schedule. Therefore, the set of "active" schedules for our scheduling problem must be generated differently from that for the nonpreemptive case. Specifically, in the course of searching for an optimal schedule, we consider only those scheduling options that do not violate any of the three rules (IP1-IP3) listed shortly. This is because a schedule violating any of the three rules must be nonoptimal according to the discussions in Section III. Furthermore, since we are interested in finding only one optimal schedule, we will consider only one of the scheduling options in the same IM. Given the active set generated earlier, an optimal schedule w.r.t. Θ can then be obtained by applying the bounding process only on this set.

The three scheduling rules listed below are important implications of the DP's presented in the previous section. After

[6]Recall that Θ is the maximum, rather than the sum, of a set of scalar quantities. Hence this statement always holds no matter how many such T_m's are in $\Omega_0(O_b)$ for each T_n in $\Omega_0(O_c)$.

[7]The term "vertex" instead of the more commonly used term "node" is used to avoid possible confusion with the nodes of the TG and the nodes of a history tree to be introduced.

elaborating on these three rules, we will give a detailed account of generating a set of active schedules. (As before, $S_k(t_0)$ represents the set of all modules schedulable on N_k at time t_0.)

IP1: N_k must not be left idle at t_0 if $S_k(t_0) \neq \emptyset$.

IP2: N_k must execute $O_c \in S_k(t_0)$ before $O_b \in S_k(t_0)$ at time t_0 if (T1): $O_c \Rightarrow^Z O_b$ and $O_c \neq^Z O_b$, and/or (T2): $O_c \Rightarrow^\Theta O_b$ and, O_b and O_c do not belong to the same IM.

IP3: Once O_c has been chosen by N_k at t_0, no other module in $S_k(t_0)$ is allowed to preempt O_c before its completion. A new module that becomes schedulable at time $t_1 > t_0$ (and thus belongs to $S_k(t_1)$, not to $S_k(t_0)$) could preempt O_c; t_1 is the only time at which O_c can be preempted by this new module.

IP1 and IP3 are based on OB2 and Theorem 1, respectively, while IP2 comes from Theorems 2 and 3 and needs further elaboration on its implementation because of the aforementioned properties of $\Rightarrow^Z, =^Z, \Rightarrow^\Theta$ and $\mathbf{S}^\Theta$:

P1: $\Rightarrow^Z$ and $=^Z$ are transitive. Besides, if $O_c \Rightarrow^Z (=^Z) O_b$, then $O_c \Rightarrow^\Theta (\mathbf{S}^\Theta) O_b$ at any $t_0 \geq 0$; but the converse is *not* true.

P2: Neither $\Rightarrow^\Theta$ nor $\mathbf{S}^\Theta$ are transitive.

P3: If $O_c \Rightarrow^\Theta (\mathbf{S}^\Theta) O_b$ at t_0, then $O_c \Rightarrow^\Theta (\mathbf{S}^\Theta) O_b$ at any $t_1 \geq t_0$.

P1 suggests that T1 of IP2 be tested before T2, and that O_b be immediately excluded from consideration for scheduling if T1 holds. P2 and part (a) of Corollary 1 suggest that the test for T2 on the set of modules survived (called the *survival set*) the test for T1 be different from the test for T1. Specifically, all O_b– O_c pairs in the survival set are tested first for T2 and then any O_b asserted by T2 is excluded from consideration for scheduling. P3 implies that T2 (and T1, of course) need not be tested again until a new module becomes schedulable on N_k. Let $R_k(t_0)$ denote the set of modules which survive tests for both T1 and T2. Then, by Corollary 2 and part (b) of Corollary 1, $R_k(t_0)$ may be further partitioned into q_k (yet to be determined) IM's, $\Pi_k^j(t_0), j = 1, 2, \cdots, q_k$, in each of which changing the execution order does not affect the optimality. Thus, it is sufficient to pick an arbitrary module from each $\Pi_k^j(t_0)$ and consider whether or not to execute it next. Because of P2, each $\Pi_k^j(t_0)$ can be constructed as follows. First, using Corollary 2 and starting with an arbitrary module, a $\Pi_k^j(t_0)$ is formed by adding a new module O_b to it whenever there exists an O_c already in $\Pi_k^j(t_0)$ such that $O_c \mathbf{S}^\Theta O_b$. Second, use part 2) of Corollary 1 to merge a *cyclic* set of IM's into $\Pi_k^j(t_0)$. A set of IM's is said to be cyclic if elements from IM's of the set form a cycle of the dominance relation as described in part 2) of Corollary 1. This merger is to further reduce the number of branches generated at the vertex since the single module to be arbitrarily picked from the IM is now picked from a larger size IM after the merger.

To meet the uninterruptability requirement, a tree, called the *history tree* (HT_k), for N_k is used to keep track of execution order for the modules on N_k. As shown in Fig. 6, each HT_k has only one vertical thread of branches, in which

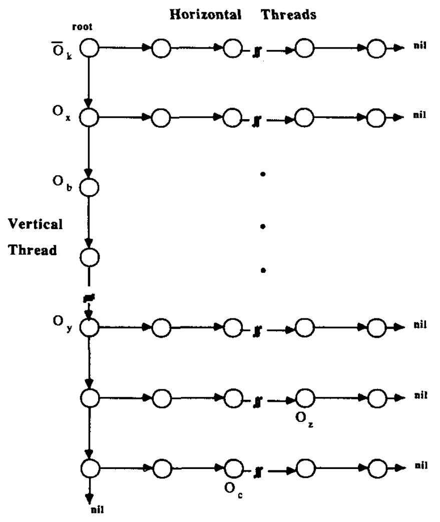


Fig. 6. The history tree of N_k.

each node represents a module that is partially completed and preempted by the one immediately above it, and the root is the module being executed by N_k. For each module on the vertical thread, a horizontal thread of branches is also constructed to record the set of modules that were not selected by N_k even after surviving both tests of T1 and T2. As we shall see later, each HT_k is constructed and updated such that a module schedulable on N_k may appear at most at one node of HT_k. HT_k's are used as F1, a tool for checking whether or not Theorem 3 and the uninterruptability requirement of IM's have been violated in any PN's prior partial schedule, and F2, a guide for each PN to select a module without violating the two conditions of F1 at least up to the time of the next scheduling decision. The violations stated in F1 are possible because more dominance relations will be established as time goes by.

In what follows, we briefly describe how HT_k's are constructed and updated, and explain how F1 and F2 are done to further simplify the search in the B&B algorithm. Let t_1 be the time when a module or communication delay is completed, or a new task is released, and thus, a new scheduling decision has to be made. Also, let $R_k(t_1) \subseteq S_k(t_1)$ be the set of all schedulable modules, each of which survived both T1 and T2 at t_1, and $\Pi_k^1(t_1), \Pi_k^2(t_1), \cdots, \Pi_k^{q_k}(t_1)$ be the q_k IM's resulting from partitioning $R_k(t_1)$. If the original HT_k is a null tree, then checking F1 is unnecessary and selecting a module from any of $\Pi_k^j(t_1)$ will violate neither of the two conditions of F1. Let γ_k denote the module selected by N_k. Thus, an HT_k is created by using γ_k as its root and including each module in $R_k(t_1) - \{\gamma_k\}$ in the horizontal branch rooted by γ_k. If the original HT_k is not null, then both F1 and F2 need to be performed. For F1, there are at least two cases, V1 and V2, to be checked:

V1: There are at least a pair of modules, say, O_b and O_c with $O_c \Rightarrow^\Theta O_b$ at t_1, such that the node representing O_b is on the vertical thread of HT_k and has the node representing O_c on one of its branches (Fig. 6).

V2: There are at least three modules, say, O_x, O_y and O_z, such that (1) both O_x and O_y are on the vertical thread of HT_k and O_y is rooted by O_x (i.e., O_y has ever been directly or indirectly preempted by O_x), and (2) O_z is rooted by O_y and belongs to the same IM as O_x (i.e., the IM containing O_x and O_z has ever been interrupted by that containing O_y).

If any of V1 and V2 is true for any PN, then the parent vertex y_0 being expanded is discarded because y_0 will never lead to an optimal schedule, or there exists at least another optimal schedule that does not include y_0 as its partial schedule. If the parent vertex y_0 is not discarded, then F2 needs to be performed on each N_k such that the selected module for N_k will not violate either of the two conditions in F1 at t_1. F2 is done by building a set of "prohibited" IM's on N_k. Specifically, a prohibited IM is the one that has ever been preempted on N_k. Thus, if a module is chosen from a prohibited IM by N_k at t_1, then the uninterruptability requirement of this IM is violated at t_1. Further, to satisfy Theorem 1, if a module is to be selected by N_k from an IM of which one module is being executed, then N_k can only continue the module that N_k has been executing. After applying F1 and F2, let γ_k denote the module selected for N_k (if such a γ_k does not exist, then N_k is kept idle until the time of next scheduling decision) and let $\overline{O}_k$ represent the module that was being executed at the time of selecting γ_k. Then, HT_k is updated according to the following rules:

Case 1: $\gamma_k \neq \overline{O}_k$ and $\overline{O}_k$ is not completed at t_1. A new node representing γ_k is created on top of $\overline{O}_k$, indicating the preemption of $\overline{O}_k$ by γ_k. For this new node, a horizontal thread is constructed to include each module in $R_k(t_1) - \{\gamma_k\}$ if it is not already in HT_k. This is to indicate that, among all modules in $R_k(t_1)$, only γ_k is selected and the thread contains the other modules which are not selected by N_k at t_1.

Case 2: $\gamma_k \neq \overline{O}_k$ and $\overline{O}_k$ is completed at t_1. If γ_k is not the module previously preempted by $\overline{O}_k$, then the node representing $\overline{O}_k$ is replaced by that representing γ_k, and each module in $R_k(t_1) - \{\gamma_k\}$ must be appended to the original horizontal thread of $\overline{O}_k$ in case it is not already in HT_k. Otherwise, $\overline{O}_k$ is simply deleted from HT_k, and each module of $R_k(t_1) - \{\gamma_k\}$ in the horizontal thread of $\overline{O}_k$ is appended to the horizontal thread of γ_k if it is not already in HT_k.

Case 3: $\gamma_k = \overline{O}_k$ and $\overline{O}_k$ is not completed at t_1. No new node is created to preempt $\overline{O}_k$. However, modules in $R_k(t_1) - \{\gamma_k\}$ must be appended to the horizontal thread of $\overline{O}_k$.

Case 4: No γ_k is selected. Delete $\overline{O}_k$, if any, from HT_k to let HT_k become null and N_k be idle. This is because $\overline{O}_k$ must be completed at t_1 and no more schedulable modules are available to N_k.

Also, to ensure that γ_k appears only once in HT_k, the γ_k already included in one of the horizontal threads must be deleted in the above rules. Except when $\overline{O}_k$ or the module is preempted by $\overline{O}_k$ in Case 2, γ_k can never be any node in the vertical thread.

Based on the above discussions of IP1-3, we can now construct the algorithm which generates a small set of active schedules using DP's. It is helpful to summarize and/or introduce the notation to be used in the algorithm as follows:

A:	Set of modules without preceding activities on the TG. A module in A becomes schedulable whenever the task containing it is released.
B:	Set of modules with preceding activities on the TG. A module in B becomes schedulable upon completion of all its preceding activities as well as the release of the task containing it.
Y:	Set of communication delays. Each communication delay must have at least one preceding module.
t_0:	The current time or the time a scheduling decision has been made.
t_1:	The earliest time since t_0 when at least a module or communication delay is completed or a task is released. That is, t_1 is the time when a new scheduling decision has to be made.
S_k:	Set of schedulable modules at t_1 on N_k including those partially completed.
L_d:	Set of ready communication delays, m_i's, at t_1. The remaining period of m_i is denoted by v_i. Since communication delays are not schedulable objects, each v_i is reduced to the passage of time.
R_k:	Subset of schedulable modules on N_k, each of which survives tests for both T1 and T2 of IP2 at t_1. Thus, $R_k \subseteq S_k$.
Π_k^j:	The jth IM on N_k at t_1. Each Π_k^j and the total number of IM's on N_k, denoted by q_k, may change with time.
y_0:	The "parent" vertex in the state space to be searched.
y_1:	The "child" vertex of y_0.
α_b:	Release time of $O_b \in A$.
β_k:	Completion time of a module which is executed by N_k.
β_d:	The earliest completion time among all communication delays in L_d.

Since preemptions are allowed, a vertex y is represented by the following information:

PS_k:	The partial schedule of N_k containing all scheduled (completed and uncompleted) modules.
HT_k:	The history tree of N_k.

Using the DP's and notation introduced thus far, we present the algorithm for generating a small set of active schedules that contains at least one optimal schedule.

```
**PROCEDURE** create_root_of_search_tree
**For** $k = 1, 2, \cdots, n$ **do**
  1. Initialize$S_k := \emptyset, R_k := \emptyset$.
  2. Set $PS_k = HT_k := \emptyset$.
**end_do**.
Set $t_1 := 0, t_0 := 0$, and $L_d := \emptyset$ to create the root vertex $y_0 = y_1$.
**end**_create_root_of_search_tree.
```

```
**MAIN PROGRAM** generate_active_schedules
S1.  Initialize $A$, $B$ and $Y$.
S2.  create_root_of_search_tree.
S3.  **While** the generated vertex $y_1$ does not represent a complete schedule **and** each $HT_k$ survives V1 and V2 of F1 **do**:
```

3a. Create S_k's (L_d) by moving all modules (communication delays) that become schedulable (ready) at t_1 from A to B (Y) to the corresponding S_k's (L_d).
3b. Perform the dominance tests T1 and T2 on each S_k to generate R_k, and partition each R_k to derive $\Pi_k^j, j = 1, 2, \cdots, q_k$.
3c. Perform F2 to choose $\Pi_k^i, i = 1, 2, \cdots, \hat{q}_k$, from that in Step 3b, where i is the new index and $\hat{q}_k \le q_k$. Let $\Gamma_k = \{O_k^1, O_k^2, \cdots, O_k^{\hat{q}_k}\}$ be the set of modules chosen by N_k.
3d. Set $y_0 := y_1$, and create a child vertex y_1 of y_0 for each PN. Let γ_k denote the module chosen by N_k in y_1. (If $\hat{q}_k = 0$ then $\gamma_k :=$ *null*, meaning that N_k is left idle in y_1.)
3e. **For** each created y_1 **do**:

3.e.1 Update $HT_k, k = 1, 2, \cdots, n$, according to the rules described above.
3.e.2 Set $t_0 := t_1$ and prepare to obtain a new t_1 as below.
3.e.3 Determine $\alpha^* = \min_{O_b \in A}\{\alpha_b\}$,

$$\beta^* = \min\{\beta_d, \min_{k=1,\cdots,n}\{\beta_k\}\},$$

and set $t_1 := \min\{\alpha^*, \beta^*\}$, the earliest time a new scheduling decision has to be made.
3.e.4 Modify PS_k by scheduling γ_k during the interval $[t_0, t_1]$ (let N_k be idle if $\gamma_k =$ *null*), $k = 1, 2, \cdots, n$.
3.e.5 Return to Step S3.

While satisfying the DP's derived earlier, the algorithm recursively generates a set of preemptive active schedules in a depth-first fashion. This set of active schedules contains at least one optimal schedule. Also, because of the uninterruptability in Theorem 1, the depth of the search tree generated is at most twice the total number of modules to be scheduled.

B. Estimation of Lower-Bound Cost

Once the rule for expanding a vertex is determined, the efficiency of search for an optimal schedule with a B&B algorithm depends solely on $\hat{\Theta}(y)$, the lower-bound cost of vertex y, and the computation needed to obtain $\hat{\Theta}(y)$. Based on Algorithm A, various $\hat{\Theta}(y)$'s can be derived depending on how the precedence constraints are relaxed and how the release time and cost function of each module (communication delay) are determined. For one extreme, we may ignore all precedence constraints between any two PN's, assume that all tasks have been released, apply Algorithm A to obtain the minimal maximum cost for each PN, and use the maximal mini-max cost among all PN's as a lower-bound cost $\hat{\Theta}_1(y)$. For the other extreme, we may consider all precedence constraints and task release times, and use the same method to derive another lower-bound cost $\hat{\Theta}_2(y)$. Obviously, $\hat{\Theta}_2(y)$ is tighter than $\hat{\Theta}_1(y)$ but requires more effort to compute. Since all other bounds in between these two extremes can be derived similarly, we shall consider only these two extreme bounds.

Consider a vertex y at $t = t_1$, let $g_i(y)$ be the actual path cost of T_i from the root to y. $g_i(y)$ can be easily computed from the partial schedule represented by y as follows. Since some tasks may have been completed before $t_1, g_i(y)$ is determined as T_i's ***normalized partial response time*** at t_1:

$$g_i(y) \triangleq \begin{cases} \overline{c}_i & \text{if } T_i \text{ is completed before } t_i \\ (t_1 - r_i)/w_i & \text{otherwise} \end{cases} \tag{3}$$

where $\overline{c}_i, r_i$, and w_i are the normalized response time, release time and normalization factor of T_i, respectively. Note that $g_i(y) < 0$ if T_i is not yet released before t_1.

To derive $\hat{\Theta}_1(y)$, precedence constraints between two PN's are ignored, and all unfinished tasks are assumed to have been released. These assumptions make it possible to schedule tasks, rather than individual modules, with Algorithm A. Specifically, in step SA2 of Algorithm A, set the cost function of unfinished T_i as

$$f_i(t) := (t - r_i)/w_i = g_i(y) + (t - t_1)/w_i \tag{4}$$

and let $t_1 + R_k(B)$ be the block completion time $t(B)$, where $R_k(B)$ is the sum of all remaining execution times of the set of unfinished modules each of which precedes at least a task in B. Suppose $\hat{\theta}_{1k}(y)$ is the maximum between 1) the mini-max cost of N_k obtained from Algorithm A, and 2) the maximum $g_i(y)$ obtained from (3) among all tasks completed before t_1 on N_k. Then $\hat{\Theta}_1(y) \triangleq \max_{N_k \in N} \hat{\theta}_{1k}(y)$ is a lower-bound cost of y, and the computational complexity is $O(n|\overline{T}|^2)$, where n is the total number of PN's. $|\overline{T}|$ the average number of tasks on each PN.

$\hat{\Theta}_2(y)$ is derived while all precedence constraints and task release times are considered. This also implies that modules, rather than tasks, are the objects to be scheduled by Algorithm A. In a simpler case, the release time s_b and cost function $f_b(t)$ of an unfinished module O_b can be determined as follows. Since O_b is not schedulable until each task that contains at least a module preceding O_b has been released, s_b is set to the maximum release time among all such tasks. Because it is the completion of a task, rather than that of a module, that accounts for the cost, $f_b(t)$ may be set as:

$$f_b(t) := \begin{cases} (t - r_i)/w_i & \text{if } head(O_b) = T_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where t is the completion time of T_i, and $head(O_b) = T_i$ means that $head(O_b)$ is the node representing the completion event of T_i. After applying Algorithm A, define $\hat{\theta}_{2k}(y)$ similarly to $\hat{\theta}_{1k}(y), k = 1, 2, \cdots, n$. Then, $\hat{\Theta}_2(y) \triangleq \max_{N_k \in N} \hat{\theta}_{2k}(y)$ is a tighter lower-bound cost than $\hat{\Theta}_1(y)$, and the computational complexity is $O(n|\overline{P}|^2)$, where $|\overline{P}|$ is the average number of modules on each PN. Note that more accurate, but more computational demanding release time and cost function for each unfinished module are possible. For example, the minimum time from a task's release to a module's release may also be considered to determine the release time of the module. Furthermore, the effect of a module's completion on the completions of tasks in the other PN's may be included in deriving a more accurate cost function of that module. These issues are partially addressed in [2].

As mentioned earlier, $\hat{\Theta}_2(y) \ge \hat{\Theta}_1(y), \forall y$, because $\hat{\Theta}_1(y)$ is a lower-bound of $\hat{\Theta}_2(y)$. There is always a trade-off between

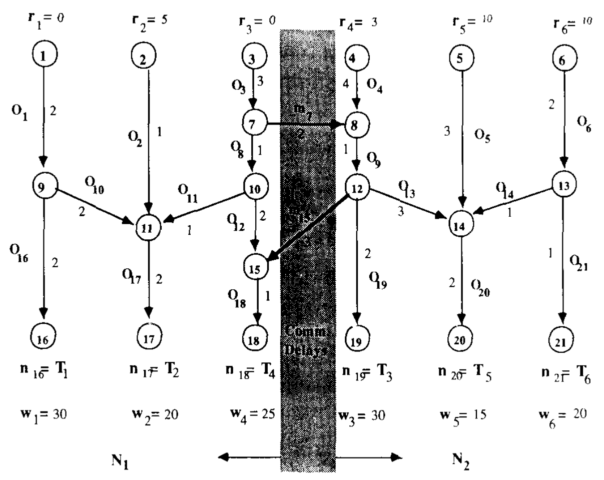


Fig. 7. The example TG.

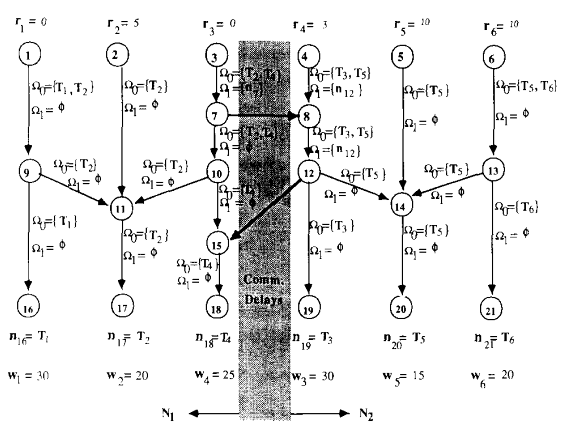


Fig. 8. The $\Omega_0(\bullet)$ and $\Omega_1(\bullet)$ sets of the TG in Fig. 7.

the accuracy and computational complexity of any lower-bound.

V. An Example and Computational Experiences

In this section, a demonstrative example and some computational experiences of the proposed B&B method are presented.

A. An Example

Consider the scheduling problem for the TG in Fig. 7, where six tasks $T_1, T_2, \cdots, T_6$ are to be executed by N_1 and N_2. All modules on the LHS of the shaded area of Fig. 7 are to be executed by N_1 and those on the RHS by N_2. Each task is released by realizing its starting node and completed by realizing its ending node. The release times and normalization factors are $r_1 = 0, r_2 = 5, r_3 = 0, r_4 = 3, r_5 = r_6 = 10, w_1 = 30, w_2 = 20, w_3 = 30, w_4 = 25, w_5 15$ and $w_6 = 20$. Notice that, while all the other tasks are released and completed on the same PN, T_3 (T_4) is released on N_1 (N_2) but completed on N_2 (N_1). Fig. 8 shows $\Omega_0(\bullet)$ and $\Omega_1(\bullet)$ of each module and Fig. 9 gives the cost function for each task. Using $\hat{\Theta}_1(y)$, we show in Fig. 10 all the vertices that have been generated in ascending order of their indices, and in Fig. 11 the corresponding optimal schedule.

Before reaching an optimal schedule—which is represented by 17 vertices from the root to vertex v_{25}—a total of only 25 vertices are generated. The first five vertices v_1–v_5 were generated because DP's exist neither between O_1 and O_3 nor between O_{16} and O_3. Since O_3 is denied and O_{16} is chosen at v_5, vertices where O_3 preempts O_{16} will never be generated by expanding v_5 except at, or after, O_{16}'s completion (Theorem 1). When expanding v_6, where O_3 and O_2 are both schedulable, only v_7 where O_3 is chosen by N_1 is generated because $O_3 \Rightarrow^Z O_2$. This branch of the tree is expanded until v_9, whose lower-bound cost becomes 0.73. Even with the same lower-bound cost 0.70, v_4, rather than v_3, is chosen for expansion because the depth-first policy is used to break a tie on lower-bound costs.

As v_{10} is expanded, both v_{11} and v_{12} have to be generated because no DP's between O_8 and O_{16} exist (since neither $T_2\,\mathrm{sp}^{\Theta}\,T_1$ nor $T_1\,\mathrm{sp}^{\Theta}\,T_1$.) From v_{12}, only v_{13}, where O_8 is selected, is generated because $O_8 \Rightarrow^Z O_2$ and $O_8 \Rightarrow^Z O_{10}$ although O_2, O_8 and O_{10} are all schedulable at $t = 7$ (i.e., t_1 of v_{12} and t_0 of v_{13}). While expanding v_{13}, four modules (O_2, O_{10}, O_{11} and O_{12}) on N_1 and two modules (O_{13} and O_{19}) on N_2 are schedulable, making a total of eight combinations. Since O_2, O_{10}, and O_{11} belong to the same IM and dominate ($\Rightarrow^{\Theta}$) O_{12}, an arbitrary module, say O_2, is chosen for N_1. On the other hand, $O_{13} \Rightarrow^{\Theta} O_{19}$, so only O_{13} is chosen by N_2, although both are schedulable. Therefore, out of these eight combinations, only one vertex v_{14} needs to be generated without sacrificing optimality, and thus, significantly simplifies the search.

Another situation to be noted is when both O_5 and O_6 become schedulable on N_2 at $t = 10$ as v_{15} is expanded. Since $O_6 \Rightarrow^Z O_5(O_{13})$ and $O_5(O_{13}) \Rightarrow^{\Theta} O_{19}$ at $t = 10$, the only vertex to be generated is for O_6 to preempt O_{13} on N_2 although a total of four modules (O_5, O_6, O_{13} and O_{19}) are schedulable on N_2. After completing O_6, O_{13} resumes its execution. We can easily see how the DP's are followed similarly by the rest of the optimal schedule.

For completeness, we show how $\hat{\Theta}_1(y)$ and $\hat{\Theta}_2(y)$ can be derived for the above example. Consider v_{14} (see Figs. 10 and 11) for instance, where $t = 9$ when T_1 is completed while neither T_5 nor T_6 is released. The normalized partial response times of tasks at $t = t_1 = 9$ are $g_1(v_{14}) = (7-0)/30 = 7/30, g_2(v_{14}) = (9-5)/20 = 4/20, g_3(v_{14}) = (9-0)/30 = 9/30, g_4(v_{14}) = (9-3)/25 = 6/25, g_5(v_{14}) = (9-10)/15 = -1/15, g_6(g_{14}) = (9-10)/20 = 1/20$. To derive $\hat{\Theta}_1(v_{14})$, precedence constraints between N_1 and N_2 are relaxed and all tasks are assumed to have been released. The block completion time $t(B)$ for unfinished tasks T_1 and T_2 on N_1 is determined to be $t_1 + R_1(B) = 9 + 8 = 17$, where $R_1(B)$ is the sum of remaining execution times of all unfinished modules on N_1, (i.e., $O_{10}, O_{11}, O_{12}, O_{17}$ and O_{18}), each of which precedes T_2 or T_4. Since $f_2(17) > f_4(17)$ (Fig. 9), by Algorithm A, T_4 should be completed last. Delete those modules that precede only T_4 and compute the block completion time for those

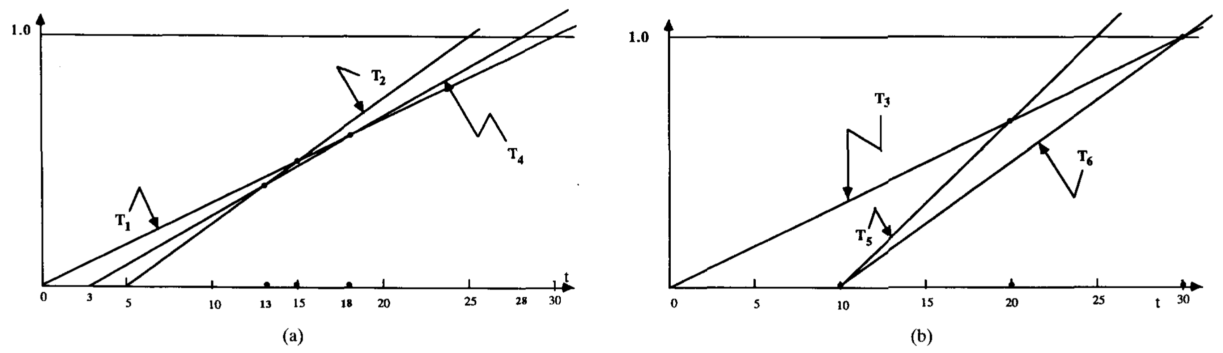


Fig. 9. Cost functions of tasks. (a) Tasks on N_1. (b) Tasks on N_2.

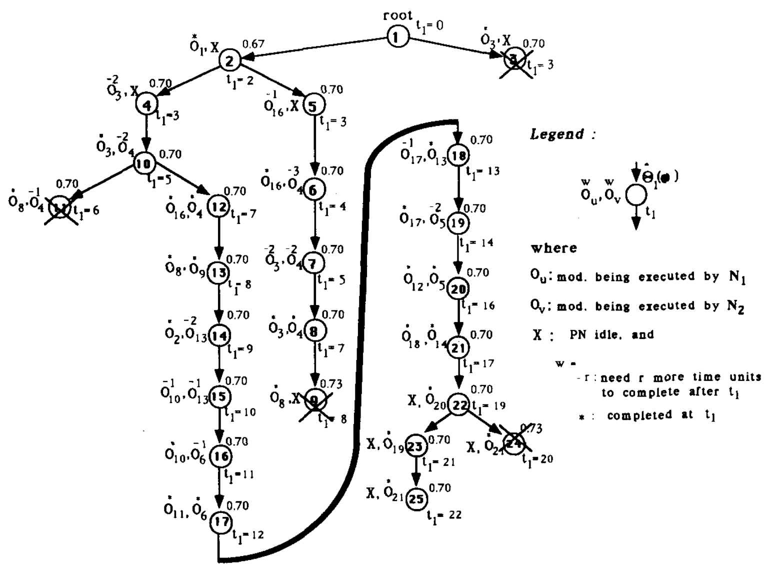

root
Legend :
where
O_u: mod. being executed by N_1
O_v: mod. being executed by N_2
X : PN idle, and
-r : need r more time units to complete after t_l
* : completed at t_l

Fig. 10. The search tree of the TG of Fig. 7.

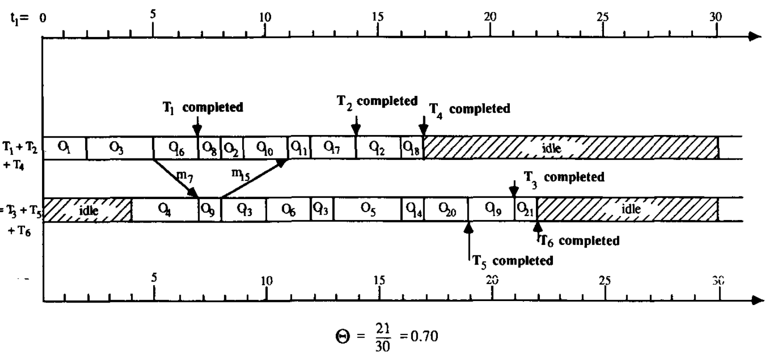


Fig. 11. The optimal schedule of Fig. 10.

that precede only T_2 to obtain $t_l + 5 = 14$. This implies that the mini-max lower-bound cost for T_2 and T_4 on N_1 be max $\{f_2(14), f_4(17)\} = \max\{g_2(v_{14}) + 5/20, g_4(v_{14}) + 8/25\} = \max\{9/20, 14/25\} = 14/25$. Thus, $\hat{\theta}_{11}(v_{14}) = \max\{14/25, 7/30\} = 14/25$.

Similarly, we proceed with tasks T_3, T_5 and T_6 on N_2. They should be completed in the order of T_5, T_3, and T_6, and with completion times 19, 21 and 22, respectively. Thus, $\hat{\theta}_{12}(v_{14}) = \max\{f_3(21), f_5(19), f_6(22)\} = \max\{21/30, 9/15, 12/20\} = 21/30$. It follows that $\hat{\Theta}_1(v_{14}) =$

$\max\{\hat{\theta}_{11}(v_{14}), \hat{\theta}_{12}(v_{14})\} = 21/30$ as shown in Fig. 10.

$\hat{\Theta}_2(v_{14})$ is derived while honoring all precedence constraints and all task release times. The release times of unfinished modules on N_1 at v_{14} are $s_{10} = s_{11} = s_{12} = 0, s_{17} = 5$ and $s_{18} = 3$, and by (5), the cost functions for these modules are 0 except O_{17} and O_{18}, whose cost functions are those of T_2 and T_4, respectively. By applying Algorithm A and comparing the result with $g_1(v_{14})$, we obtain $\hat{\theta}_{21}(v_{14}) = \max\{7/30, 14/25\} = 14/25$, which is the same as $\hat{\theta}_{11}(v_{14})$. For the unfinished modules on N_2 at $v_{14}, s_5 = s_6 = s_{14} = s_{20} = s_{21} = 10$, and $s_{13} = s_{19} = 0$. Similarly, the cost functions for O_5, O_6, O_{13} and O_{14} are 0 while those for O_{19}, O_{20} and O_{21} are those for T_3, T_5 and T_6, respectively. By applying Algorithm A, $\hat{\theta}_{22}(v_{14}) = 21/30$ is obtained, which is again equal to $\hat{\theta}_{12}(v_{14})$. Thus, $\hat{\Theta}_2(v_{14}) = \hat{\Theta}_1(v_{14}) = 21/30$. These results are expected because the unfinished modules always create the same single block in Step SA2 of Algorithm A regardless whether $\hat{\Theta}_1(y)$ or $\hat{\Theta}_2(y)$ is derived.

B. Computational Experiences

The proposed B&B algorithm, which embodies Theorems 1–3 and Corollaries 1–3, was coded in Pascal and run on a VAX-8600 computer with 4.3 BSD UNIX (UNIX is a trademark of AT&T Bell Laboratories) operating system. In order to test a wider class of sample problems, a total of 90 sets of tasks were randomly generated according to the classification of tasks: 1) the average number of modules per task is either 4, 6, or 8 and 2) the number of PN's in the system is either 2, 5, or 8.

Consider a class where the modules per task is x_1 and the number of PN's x_2. A total of 30, 20, and 10 locally numbered nodes are initially set up on each PN for $x_1 = 4, 6$ and 8, respectively. The module between nodes n_i and $n_j, i < j$, on the same PN is generated if the outcome from a random experiment using a uniform distribution is greater than the threshold $p\rho^r$, where $p \leq 1$ is the initial probability, $\rho = 0.5$ the *discount factor*, and $r = j - i - 1$ the *discount period*. That is, given p, the larger the difference $j - i$, the less likely is a module generated between n_i and n_j. Therefore, by tuning p and considering the total number of tasks to be created, we can generate task sets with the desired x_1 value. After removing each "dangling" node, tasks are created by randomly assigning their starting and ending nodes to the remaining nodes. The number of tasks created for each class is pre-determined such that the average numbers of tasks on each PN are 2, 1.5, and 1 for $x_1 = 4, 6$, and 8, respectively. For example, two tasks on the average are created for the class where $x_1 = 8$ and $x_2 = 2$. Ten tasks are created for the class where $x_1 = 4$ and $x_2 = 5$, and so on. Also, communication delays are generated for each task and randomly inserted between the task and other tasks on different PN's to establish the required precedence constraints.

For each of these 9 classes, 10 task sets were tested. Table I summarizes the (rounded) average numbers of activities, and tasks created for each class. The test results, which include the (rounded) average number of vertices generated and the CPU time (the sum of user and system times) consumed, are recorded in Table II and plotted in Fig. 12. According to our experiences, the variance of each entry in Table II grows rapidly as the number of PN's increases. For example, for $x_1 = 8$, the variance of the number of vertices generated (CPU time used) for the entry $x_2 = 2$ is about 27 (1389) times higher than that for the entry $x_2 = 8$. Therefore, the test results are very sensitive to the way in which the random

TABLE I
THE NINE CLASSES OF TG'S TESTED

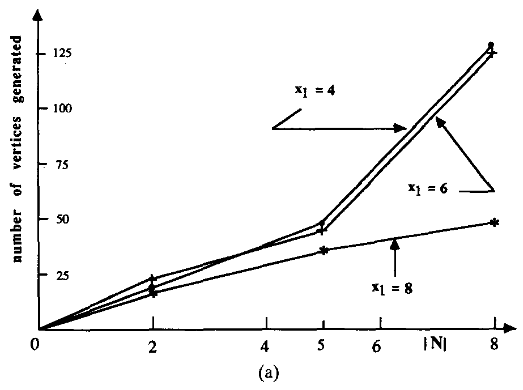
x_1	x_2: n = 2	n = 5	n = 8
4	20[a], 4[b]	55, 10	92, 16
6	20, 3	55, 7	90, 12
8	18, 2	47, 5	78, 8

[a]Number of activities
[b]Number of tasks

TABLE II
THE EST RESULTS OF THE NINE CLASSES

x_1	x_2: n = 2	n = 5	n = 8
4	19[a], 0.23[b]	48, 2.51	130, 18.23
6	20, 0.20	47, 2.24	126, 16.25
8	18, 0.11	37, 0.86	49, 3.02

[a]Number of vertices generated.
[b]CPU time used (seconds).



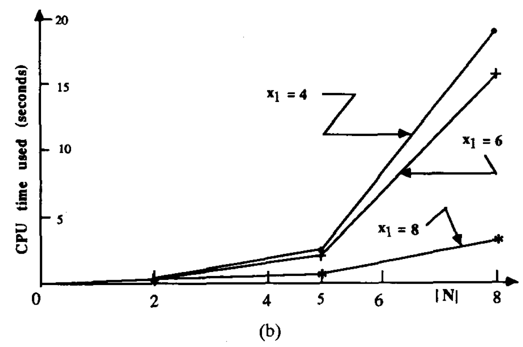


Fig. 12. Test results. (a) Number of vertices generated. (b) CPU time used (seconds).

samples are generated as well as to the set of task sets actually tested as the number of PN's increases.

VI. Conclusion

Since the task scheduling problem with precedence constraints in a distributed system is generally NP-hard, some form of heuristic is necessary to solve it. In this paper, we have presented a new approach to the scheduling problem using a B&B algorithm on the basis of

- Modeling the task set with an acyclic graph,
- Identifying the DP's w.r.t. all regular measures and the system hazard,
- Developing the vertex expansion algorithm using the DP's into which the B&B algorithm is embedded, and
- Deriving lower-bound costs for each vertex so that the B&B algorithm may be efficiently guided to find an optimal schedule.

Our computational experiences, albeit limited, have indicated that this approach is very efficient in searching for an optimal schedule. Because our approach depends on the B&B algorithm, it can be extended (with certain modifications, of course) to solve similar problems with other types of resource constraints. For example, suppose only r units of memory is available at time t_0, then this constraint can be easily imposed to the branching process: only those schedulable modules at t_0 whose individual memory requirement is equal or less than r is allowed to be considered for further branching in the proposed B&B algorithm.

References

[1] K. G. Shin, C. M. Krishna, and Y.-H. Lee, "A unified method for evaluating real-time computer controllers and its applications," *IEEE Trans. Automat. Contr.*, vol. AC-30, no. 4, pp. 357–366, Apr. 1985.

[2] D. T. Peng and K. G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems," in *Proc. IEEE 9th Int. Conf. Distributed Comput. Syst.*, Newport Beach, CA, 1989, pp. 190–198.

[3] D. T. Peng, "Modeling, assignment and scheduling of tasks in distributed real-time systems," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI, Dec. 1989.

[4] H. A. Taha, *Operations Research: An Introduction.* New York: Macmillan, 1976, pp. 357–366.

[5] K. R. Baker, *Introduction to Sequencing and Scheduling.* New York: Wiley, 1974.

[6] R. Bellman, A. O. Esogbue, and I. Nabeshima, *Mathematical Aspects of Scheduling and Applications.* Elmsford, NY: Pergamon, 1982.

[7] T. Gonzalez and S. Sahni, "Flowshop and jobshop schedules: Complexity and approximation," *Operations Res.*, vol. 26, no. 1, Jan.–Feb. 1978, pp. 36–52.

[8] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Ann. Discrete Math.*, vol. 1, pp. 343–362, 1977.

[9] K. R. Baker *et al.*, "Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints," *Operations Res.*, vol. 31, no. 2, Mar.–Apr. 1983, pp. 381–386.

[10] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Applied Math.*, vol. 5, 1983, pp. 11–24.

[11] B. Giffler and G. L. Thompson, "Algorithms for solving production scheduling problems," *Operations Res.*, vol. 8, no. 4, pp. 487–503, 1960.

[12] G. H. Brooks and C. R. White, "An algorithm for finding optimal or near optimal solutions to the production scheduling problems," *J. Indust. Eng.*, vol. 16, 1965, pp. 34–40.

[13] L. Schrage, "Solving resource-constrained network problems by implicit enumeration—Nonpreemptive case," *Operations Res.*, vol. 18, pp. 263–278, 1970.

[14] ——, "Solving resource-constrained network problems by implicit enumeration—Preemptive case," *Operations Res.*, vol. 20, pp. 668–677, 1972.

[15] E. L. Lawler *et al.*, "Recent developments in deterministic sequencing and scheduling: A survey," in *Deterministic and Stochastic Scheduling*, Dempster *et al.*, Eds. Dordrecht, The Netherlands: Reidel, 1982, pp. 35–74.

[16] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Job-shop scheduling by implicit enumeration," *Management Scie.*, vol. 24, no. 4, pp. 441–450, 1977.

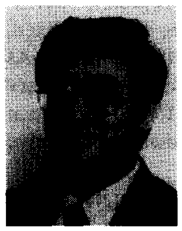
[17] J. P. Tremblay and R. Manohar, *Discrete Mathematical Structures with Applications to Computer Science.* New York: McGraw-Hill, 1975, pp. 149–162.

[18] W. H. Kohler and K. Steiglitz, "Enumerative and interactive computational approach," in *Computer and Job-Shop Scheduling Theory*, Coffman Eds. New York: Wiley, 1976, pp. 229–287.

Dar-Tzen Peng received the B.E.E. degree, the M.S. degree in management science both in Taiwan, the M.S.E. degree in computer, information and control engineering, and the Ph.D. degree in computer science and engineering in 1984 and 1990, respectively, from the University of Michigan, Ann Arbor.

From 1984 to 1989 he was a Research Assistant at the University of Michigan, working on geometric modeling and distributed real-time computing. He joined the Allied Signal Microelectronics and Technology Center in 1989 as an Member of the Technical Staff where his work consists of the research, design, and implementation of fault-tolerant distributed real-time computing systems. Currently, he is involved in the design of Real-Time Executive Module (RTEM), a software implementation of a multiprocessor architecture for fault tolerance. His research interests include fault-tolerant real-time computing, computer networks, and the automatic design of such systems.

Dr. Peng is a member of the Association for Computing Machinery and the IEEE Computer Society.



Kang G. Shin received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY in 1976 and 1978, respectively. He is currently Professor and Chair of Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, NY. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, the Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California, Berkeley, and the International Computer Science Institute, Berkely, CA. He has authored/coauthored over 240 technical papers (more than 100 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, and robotics and automation. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing. He was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of IEEE Transactions on Computers on Real-Time Systems, a Program Co-Chair for the 1992 International Conference on Parallel Processing, and served numerous technical program committees. He chaired the IEEE Technical Committee on Real-Time Systems during 1991–1993, is a Distinguished Visitor of the Computer Society of the IEEE, an Editor of IEEE Trans. on Parallel and Distributed Computing, and an Area Editor of the *International Journal of Time-Critical Computing Systems.*

In 1987, Dr. Shin received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from The University of Michigan.