Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems

Kang G. Shin and Dar-Tzen Peng[1]

TR-88-005

October 21, 1988

[1]Authors are with the International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, California 94704-1105, on leave from the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122.

STATIC ALLOCATION OF PERIODIC TASKS WITH PRECEDENCE CONSTRAINTS IN DISTRIBUTED REAL-TIME SYSTEMS

Dar-Tzen Peng and Kang G. Shin

ABSTRACT

Using two branch-and-bound (B&B) algorithms, we propose an optimal solution to the problem of allocating (or *assigning* with the subsequent *scheduling* considered) periodic tasks to a set of heterogeneous processing nodes (PNs) of a distributed real-time system. The solution is optimal in the sense of minimizing the maximum normalized task response time, called the *system hazard*, subject to precedence constraints among the tasks to be allocated.

First, the task system is described as a *task graph* (TG), which represents computation and communication modules as well as the precedence constraints among them. Second, the exact system hazard of a complete assignment is determined so that an optimal (rather than suboptimal) assignment can be derived. This exact cost is obtained by optimally scheduling the modules assigned to each PN with a B&B algorithm guided by the dominance relationship between simultaneously schedulable modules. Thirdly, to reduce the amount of computation needed for an optimal assignment, we derive a lower-bound system hazard that is obtainable with a polynomial time algorithm. This lower-bound cost, together with the exact cost of a complete assignment, is used to efficiently guide the search for an optimal assignment. Finally, examples are provided to demonstrate the concept, utility and power of our approach.

Index Terms - Branch-and-bound (B&B) algorithm, computation and communication modules, dominance properties (DPs), inter-module communication (IMC), inter-processor communication (IPC), task invocation and release times, multi-project scheduling, lower-bound cost, system hazard.

---

[1]Authors are with the International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, California 94704-1105, on leave from the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122

1. INTRODUCTION

The workload in a real-time system consists of periodic and aperiodic tasks. Periodic tasks are the "base load" and invoked at fixed time intervals while aperiodic tasks are the "transient load", arriving randomly in response to environmental stimuli.

Since both task response times and system reliability can be improved by using multiple CPUs and memories, distributed computing systems are attractive candidates for implementing real-time systems. (The task response time is the time interval between a task invocation and its completion.) However, one of the most important design issues is the distribution of workload over the processing nodes (PNs) in the system to achieve the desired performance. In this paper, we deal exclusively with the *static allocation* of periodic tasks to the PNs in a distributed real-time system. The allocation is static because it remains unchanged during the entire mission lifetime as long as PNs are fault-free. By 'allocation' we mean both *assignment* and *scheduling* of tasks. Specifically, the performance of our task allocation is that of task assignment determined by simultaneously considering the subsequent optimal scheduling of the assigned tasks. This is in sharp contrast to conventional methods which deal with either assignment or scheduling of tasks, but not both. Note that the performance or cost of any assignment strongly depends on the scheduling of the assigned tasks. The allocation of aperiodic tasks is usually treated as a *dynamic load sharing* problem and beyond the scope of this paper.

Three features that distinguish our task assignment[2] problem from others are:

F1. Tasks communicate with one another to accomplish a common system goal. These communications impose precedence constraints during the course of concurrent execution of communicating tasks ([PeS87]).

F2. The tasks to be assigned are executed periodically at fixed time intervals during the mission lifetime.

[2]Since the performance of our task allocation is determined by scheduling the tasks assigned, the terms "allocation" and "assignment" are used interchangeably for the rest of the paper.

F3. Tasks are usually time-critical, meaning that each task execution is associated with a hard deadline. If the execution of a task is not completed before its deadline, catastrophic outcomes might ensue, e.g., a robot may collide with another robot or even with an operator.

F1 and F2 describe the structure of the task system, while F3 can be used to specify the criterion function for the task allocation problem. Because of F2, only the task invocations in a *planning cycle* (to be defined in Section 2) L need to be considered for the allocation of tasks since the behavior of task invocations within L repeats itself for the whole mission lifetime. The criterion function to be minimized for our allocation problem is the *system hazard* Θ, or the maximum normalized task response time, where the maximum runs over all invocations of all periodic tasks of the system within L, and the normalized response time of a task is the ratio of the task response time to its invocation period. This criterion is chosen because it subsumes other criterion functions related to avoidance of a task missing its deadline. Moreover, an allocation with a lower Θ will result in a lower probability of each task missing its deadline should the task execution time be a random variable rather than a fixed expected value. As we shall see in Section 2, our task allocation model is quite different from others in the following two aspects:

- Our model has a finer granularity in describing the precedence constraints between tasks. This captures the fact that in many cases tasks are communicating with each other during the course of their execution.
- All invocations of tasks in a planning cycle, rather than only one execution of each task, are considered.

Task assignment and scheduling problems are studied extensively in both fields of Operations Research and Computer Science [Bak74, Fre82, WoS74, CHL80]. For a set of independent periodic tasks, Dhall and Liu [DhL78] and their colleagues developed various assignment algorithms based on an optimal scheduling algorithm called *rate monotonic priority*

assignment [LiL73] or *intelligent fixed priority algorithm* [Ser72]. However, if precedence constraints exist among tasks like our case, a general approach to nonperiodic task assignment problems must be taken, and the set of 'tasks' to be assigned includes all task invocations within a planning cycle. (Of course, all invocations of the same task must be assigned to the same PN.) Depending on the assumptions and criterion functions used, nonperiodic task assignment problems are formulated in different ways. However, most prominent methods for task assignment in distributed systems are concerned with minimizing the sum of two scalar quantities: task processing costs on all assigned processors and interprocessor communications (IPC) costs [CHL80, Sto77, Efe82, DaF84, ShT85, ChL87, Vir84, PrK84]. As was reviewed in [CHL80], these methods are based on graph theoretic [Sto77, Sto78, StB78, ChA83, DaF84], integer programming [Chu69, CHL80, MLT82], or heuristic [Gon77, ElH80, Efe82, ChA83, Kan84, ShT85, ChL87, Vir84, PrK84] solutions. Real-time constraints are difficult to impose when the graph theoretic approach is used. Integer programming methods, on the other hand, allow for constraints that all of the tasks assigned to a processor must be completed within a given time. However, these constraints do not account for task queueing and precedence constraints between tasks.

Few results have been reported on the task assignment with precedence constraints, because most of such problems are NP-hard [LeK78, Cof76, GaJ79, LLK82]. This fact calls for the development of enumerative optimization methods or approximate algorithms using heuristics [KoS76]. For example, in [KaN84] an enumeration tree of task scheduling is generated and searched using a heuristic algorithm called the CP/MISF (Critical Path/Most Immediate Successors First) and an optimal/approximate algorithm called the DF/IHS (Depth-First/Implicit Heuristic Search) to obtain an approximate minimum schedule length (i.e., makespan) for a set of tasks. Chu and Lan [ChL87] chose to minimize the maximum processor workload for the assignment of tasks in a distributed real-time system. Workload was defined as the sum of IPC and accumulated execution time on each processor. A wait-time-ratio between two assignments

was defined in terms of task queueing delays. Precedence relations were used, in conjunction with the wait-time-ratios, to arrive at two heuristic rules for task assignment.

To our best knowledge, there are no results reported in the literature on the task allocation problem dealing with all of the foregoing three features of distributed real-time systems. This is probably because the problems associated with designing and analyzing such real-time systems are very hard. For example, even for a given assignment, it is shown in [GoS78, LRB77, LLK82] that most job-shop scheduling problems, which are special cases of our scheduling problem after task assignment, are already NP-hard. Thus, heuristic or enumeration algorithms must be sought.

Two branch-and-bound (B&B) algorithms are proposed to solve our task allocation problem: one, called the B&BA algorithm, for the optimal assignment of tasks, and the other, called the B&BS algorithm, for the optimal scheduling of assigned tasks. The exact cost of a complete assignment determined by the B&BS algorithm is used by the B&BA algorithm to derive an optimal assignment. For non-terminal vertices[3] (i.e., partial assignments) of the B&BA algorithm, however, lower-bound (rather than exact) costs are estimated by a polynomial-time algorithm to ease the ensuing computational difficulty. The B&BS algorithm uses the dominance relationship between simultaneously schedulable modules to simplify the search for an optimal schedule of modules assigned to each PN.

The rest of this paper is organized as follows. The description of the task system and problem formulation are the subject of Section 2. The B&BS algorithm that determines the exact cost for a terminal vertex of the B&BA algorithm is presented in Section 3. Section 4 presents a polynomial-time algorithm to evaluate a "good" lower-bound cost for a non-terminal vertex of the B&BA algorithm. This lower-bound cost is used, together with the exact cost obtained in Section 3, to derive an optimal assignment of periodic tasks. An example is presented in Section 5 to demonstrate the power and utility of our task allocation method. Finally, the paper

[3]We use the term "vertex", instead of the more frequently used "node", to avoid confusion with PNs and the nodes of a task

concludes with Section 6.

2. TASK SYSTEM DESCRIPTION AND PROBLEM FORMULATION

It is desirable to model critical real-time systems (e.g., nuclear reactor control) in great details such that the true nature of the system under study is accurately represented. For example, inter-task communications depend on the assignment of the communicating tasks. Therefore, we shall model the task system under consideration to include modules associated with pure computations as well as those with communications.

Let $T = \{ T_i , 1 \le i \le |T| \}$ be the set of $|T| \ge 2$ periodic tasks to be allocated among the set of $|N| \ge 2$ processing nodes (PNs), $N = \{ N_k , 1 \le k \le |N| \}$, of the system, where $|A|$ is the cardinality of the set A. It is assumed that any two tasks residing in different PNs can communicate with each other by using the usual primitives SEND-RECEIVE-REPLY and QUERY-RESPONSE. If $T_i \in T$ issues a SEND to $T_j \in T$, T_i remains blocked (thus establishing a precedence relation) until a REPLY from T_j is received. If T_j executes a RECEIVE before the requested message arrives, it also remains blocked. A task may QUERY another task for information, which replies by executing a RESPONSE. Unlike SEND-RECEIVE-REPLY, the task being queried does not get blocked regardless whether the QUERY has arrived or not.

Each $T_i \in T$ with period p_i consists of one or more *computation modules* M_{ia}'s for pure computation, and *communication modules* $X_{ij,i}$, $X_{ji,i}$, $j \ne i$, $1 \le j \le |T|$, where $X_{ij,i}$ and $X_{ji,i}$ are modules on the T_i's side associated with the communication from T_i to T_j and communication from T_j to T_i, respectively. Note that if T_i communicates with T_j (by, say, sending a message to T_j) more than once during each execution, then different $X_{ij,i}$ and $X_{ji,i}$ need to be introduced for each such communication. Communication modules represent routines for assembling packets and packetizing messages. It is worth pointing out that, while M_{ia}'s, $X_{ij,i}$ and $X_{ji,i}$ are to be executed by N_k to which T_i is assigned, they do not all have to be completed

graph (to be described).

in order to complete T_i. For example, the completion of T_i does not require the completion of $X_{ij,i}$ associated with a RESPONSE from T_i to T_j meaning that N_k can, for some reason, postpone the response to T_j's query until T_i is completed.

Let $e_{ia} > 0$ be the *nominal computation* (measured in number of basic steps) of M_{ia} and $q_k^i \geq 0$ be the processing power (measured in number of basic steps per second) of N_k for T_i. Then, the execution time (measured in seconds) of M_{ia} on N_k is described as $e_{ia}^k = e_{ia}/q_k^i$. The quantity e_{ia} is just a measure of the computation needed for M_{ia} without considering the processing ability of a PN that executes it, while q_k^i is the description of such an ability of N_k for T_i. Unlike e_{ia}'s, the nominal computation $\chi_{ij,e} \geq 0$ of $X_{ij,e}$, $e = i, j$, depends on the assignment of T_i and T_j. If the two communicating tasks T_i and T_j are assigned to the same PN, then $\chi_{ij,e}$ is small since such communication can be achieved via accessing shared memory. Otherwise, a larger $\chi_{ij,e}$ will be needed for packetizing (assembling) a message (packets) from T_i to T_j into packets (a message) for transmission (processing). Let $\chi_{ij,e}(0)$ and $\chi_{ij,e}(1)$, $e = i, j$, denote, respectively, the value of $\chi_{ij,e}$ when T_i and T_j are, and are not, assigned to the same PN. Similarly to M_{ia}'s, we can obtain the execution time of $X_{ij,i}$ and $X_{ji,i}$ on N_k as $\chi_{ij,i}^k(s) = \chi_{ij,i}(s)/q_k^i$ and $\chi_{ji,i}^k(s) = \chi_{ji,i}(s)/q_k^i$, $s = 0, 1$. Unlike e_{ia}, $\chi_{ij,i}$ ($\chi_{ji,i}$) may not necessarily be the same for all invocations of T_i. As mentioned earlier, even though $X_{ij,e}$'s are on the T_e's side, where $e = i$ or j, their completions are not always required to complete T_e. Also, it is worth mentioning that the division of T_i into M_{ia}'s, $X_{ij,i}$ and $X_{ji,i}$ is essential to the B&BA algorithm in Section 4.

Let $\bar{\chi}_{ij} > 0$ be the *nominal communication delay* (measured in seconds per distance unit) from T_i to T_j and $d_{hk} \geq 0$ be the communication distance (measured in distance units) from N_h to N_k, $1 \leq h, k \leq |N|$. Then, the actual communication delay $\bar{\chi}_{ij}^{hk}$ from T_i to T_j, which are assigned to N_h and N_k, respectively, is described as $\bar{\chi}_{ij}^{hk} = \bar{\chi}_{ij} d_{hk}$. The quantity $\bar{\chi}_{ij}$ is usually a function of the "size" of message to be transferred, and d_{hk} a function of the actual distance (e.g., the

number of hops) between N_h and N_k. Therefore, if $h = k$, we may set $d_{hk} := 0$ to ignore[4] $\bar{\chi}_{ij}^{hk}$. On the other hand, we may set $d_{hk} := \infty$ if there is no path between N_h and N_k. Following the usual practice, $d_{hk} = d_{kh}$ is assumed in our problem formulation. Besides, we assume *deterministic* communications in the system, i.e., for any communication from T_i to T_j, it is always possible to identify which invocations of T_i and T_j this communication is associated with.

Define a *planning cycle* L as a time period from the time when all tasks are simultaneously invoked to that of their next simultaneous invocations. The behavior of the task system within L repeats itself for the whole mission lifetime. Without loss of generality, we may simply choose $L = [0, l)$, where l, the length of L, is the least common multiple (LCM) of $\{ p_i, T_i \in T \}$. Consider all invocations of periodic tasks in a planning cycle L. T_i is invoked l/p_i times within L; denote the v-th invocation of T_i by T_{iv}, $1 \le v \le l/p_i$, with its deadline being the v+1-th invocation time of T_i. To describe the nominal computations and communication delays of all modules within L and the precedence constraints among them, an acyclic directed task graph (TG) with Activity On Arc (AOA) [Tah76] is used, where an arc represents a module and a node is an event representing the completion of some module(s). Following common practice, the nodes in TG are numbered in such a way that the number assigned to the event at the tail of an arc is always smaller than that assigned to the event at its head, and the number assigned to an arc is always smaller than those assigned to the arcs it precedes. The weight of an arc represents the nominal computation or communication delay of the corresponding module in the TG. As a node n_p in TG can also represent the completion of a certain invocation T_{iv}, we write $n_p = T_{iv}$ should this happen. Fig. 1 shows an example TG consisting of three tasks T_1, T_2 and T_3 with periods 40, 40 and 20, respectively, where e_{ia}, $\chi_{ij,\varepsilon}$, and $\bar{\chi}_{ij}$ are also given. Within the planning cycle [0, 40), except for T_3 which is invoked twice, both T_1 and T_2 are invoked only once. Therefore, in this TG, $n_{19} = T_{11}$, $n_{20} = T_{21}$, $n_9 = T_{31}$ and $n_{21} = T_{32}$. The various communications involved in the TG can be explained as follows. T_1 packetizes ($X_{12,1}$) a QUERY message to T_2 for

[4]Only the communication delay is ignored, but the execution times of the corresponding communication modules are not.

information, T_2 assembles ($X_{12,2}$) the message after it arrives, and then T_2 packetizes ($X_{21,2}$) the RESPONSE message which contains the queried information and sends it back to T_1. After the RESPONSE message arrives, T_1 assembles ($X_{21,1}$) the message to get the queried information. On the other hand, T_3 and T_2 exchange messages using SEND-RECEIVE-REPLY for both information and synchronization. T_3 packetizes ($X_{32,3}$) a SEND message to T_2, and T_2 executes a RECEIVE and packetizes ($X_{23,2}$) the corresponding REPLY message. The REPLY message, which may also contain information for T_3, is used by T_2 to unblock T_3 and, thus, can only be sent out after the original SEND message from T_3 has been received. As T_2 may proceed to assemble ($X_{32,2}$) the SEND message only after receiving it, T_3 may also assemble ($X_{23,3}$) the REPLY message only after it arrives at T_3. Notice that $\chi_{23,3}$ will have different values for T_{31} and T_{32}. Also, the completion of $X_{21,2}$, a RESPONSE from T_2 to T_1, is required for the completion of T_1, rather than T_2, meaning that T_2 may choose to finish its own computation first before responding to T_1's query.

Let $B_k^\delta \subseteq T$ be the subset of tasks assigned to N_k by an algorithm δ such that $B_k^\delta \cap B_h^\delta = \varnothing \ \forall h \neq k$ and $\bigcup_{N_k \in N} B_k^\delta = T$, and let C_{iv}^δ and r_{iv} be the completion and invocation time instants of T_{iv}, respectively. Define the *normalized task response time* $\overline{C}_{iv}^\delta$ of T_{iv} as follows.

$$\overline{C}_{iv}^\delta \triangleq \frac{C_{iv}^\delta - r_{iv}}{p_i}. \tag{1}$$

Then, the *node hazard* of N_k, θ_k^δ, and the *system hazard*, Θ^δ, are defined as:

$$\theta_k^\delta \triangleq \max_{T_i \in B_k^\delta} \left[\max_{1 \le v \le l/p_i} \overline{C}_{iv}^\delta \right], \text{ and}$$
$$\Theta^\delta \triangleq \max_{N_k \in N} \theta_k^\delta. \tag{2}$$

In other words, Θ^δ is the maximum $\overline{C}_{iv}^\delta$ over all invocations of all tasks in T which are distributed

over the PNs of the system. We want to find an optimal task assignment algorithm δ^* that achieves $\Theta^{\delta^*} = \min_{\delta} \Theta^{\delta}$. Since δ^* minimizes the maximum $\bar{C}_{iv}^{\delta}$, it allows for better system load sharing (rather than balancing) and a smaller probability of each task missing its deadline. Notice that $\bar{C}_{iv}^{\delta}$ (thus θ_k^{δ} and Θ^{δ}) depends not only on δ, but also on how the tasks assigned under δ are actually scheduled on each PN. An optimal preemptive (resume) scheduling algorithm ζ^* needs to be derived such that the Θ^{δ} obtained is the smallest value of $\bar{C}_{iv}^{\delta}$ for each δ.

For example, suppose the TG in Fig. 1 is to be assigned to two PNs, N_1 and N_2, with $q_1^i = q_2^i = 1$, $i = 1, 2, 3$, and $d_{12} = d_{21} = 1$. Then, since the two PNs are identical, only four different assignment algorithms need to be considered: δ_0, δ_1, δ_2 and δ_3, where δ_0 assigns all three tasks to a single PN, while δ_i assigns T_i to a PN and the other two tasks to the other PN. By using an optimal scheduling algorithm ζ^* under each of these assignments, we obtain $\Theta^{\delta_i} = \max\{\theta_1^{\delta_i}, \theta_2^{\delta_i}\}$ as follows.

$$\Theta^{\delta_0} = \max\left\{\frac{31}{40}, 0\right\} = \frac{31}{40}, \qquad \Theta^{\delta_1} = \max\left\{\frac{39}{40}, \frac{29}{40}\right\} = \frac{39}{40},$$

$$\Theta^{\delta_2} = \max\left\{\frac{34}{40}, \frac{42}{40}\right\} = \frac{42}{40}, \qquad \Theta^{\delta_3} = \max\left\{\frac{10}{20}, \frac{32}{40}\right\} = \frac{32}{40}.$$

Therefore, $\delta^* = \delta_0$, for Θ^{δ_0} is the smallest among these four system hazards. This counter-intuitive result comes from the fact that communication modules and delays are the predominant part of the example task system. Thus, the algorithm which assigns all tasks to a single PN is superior to the others because it minimizes the actual communication delays. The TG with execution times and actual communication delays under δ_3 is given in Fig. 2, and the associated optimal schedule in Fig. 3.

Our task allocation problem is general enough to capture the three aforementioned features of distributed real-time systems. For example, $X_{ij,\epsilon}$ may be defined for any (T_i, T_j) pair and the precedence constraints imposed by $X_{ij,\epsilon}$ can always be embedded into the TG. Moreover, the

criterion function, Θ^{δ}, is directly related to F3.

3. EXACT COST OF A TERMINAL VERTEX IN B&BA ALGORITHM

The exact cost of a terminal vertex in the B&BA algorithm corresponds to the system hazard Θ (the superscript δ is dropped for notational simplicity) of a complete task assignment. The lower-bound cost $\hat{\Theta}(x)$ of a non-terminal vertex x, however, represents the lower-bound estimate of $\Theta^{*}(x)$, the minimal Θ obtainable among those complete assignments each of which has x as its partial assignment. Therefore, $\hat{\Theta}(x)$ is usually the sum of two costs: a) the actual path cost from the root to x (i.e., the part of Θ contributed only by the tasks already assigned within x), and b) a lower-bound estimate of the minimum path cost from x to a terminal vertex (i.e., a lower-bound estimate of the part of Θ to be contributed by those tasks not yet assigned within x). The exact cost of a complete assignment is essential for the B&BA algorithm to derive an optimal assignment because it offers an upper-bound cost against which other lower-bound or exact costs may compare to decide whether or not the corresponding vertex should be pruned during the search process [KoS76].

Given a complete assignment, the problem of finding an optimal schedule that achieves Θ is a *multi-project scheduling* problem. Unfortunately, the multi-project scheduling problem is NP-hard because it contains a job-shop scheduling problem, an NP-hard problem [GoS78, LRB77], as a special cases [BEN82]. To obtain Θ, an optimal multi-project scheduling algorithm, called *Algorithm B* [PeS88], will be used. In other words, the aforementioned B&BS algorithm is the algorithm resulting from the application of Algorithm B to our scheduling problem for given complete assignments. For completeness, we will briefly describe Algorithm B and its use for our scheduling problem in the remainder of this section. (A complete account of Algorithm B is given in [PeS88].) Since Algorithm B will be described in the context of multi-project scheduling, it is informative to make the correspondence of terms between this algorithm and the scheduling problem for a given assignment (Table 1).

Table 1. The Correspondence of Terms

Multi-Project Scheduling	→	Complete-Assignment Scheduling
project J_j	→	task invocation T_{iv}
machine M_k	→	processing node (PN) N_k
multi-project graph (MPG)	→	task graph (TG)
operation	→	computation or communication module
time delay	→	communication delay
release time r_j of J_j	→	invocation time r_{iv} of T_{iv}
completion time C_j of J_j	→	completion time C_{iv} of T_{iv}
normalization factor w_j of J_j	→	invocation period p_i of T_i
normalized project flowtime $\bar{C}_j$ of J_j	→	nomalized task response time $\bar{C}_{iv}$ of T_{iv}
machine hazard θ	→	node hazard θ
system hazard Θ	→	system hazard Θ

** The term 'job' is reserved for Algorithm A of [BLL83] (see Section 3.2)

Algorithm B deals with the optimal preemptive (resume) multi-project scheduling problems subject to (i) precedence constraints of a general form as in any AOA network, and (ii) time delays present between *operations*. Each project J_j is to be executed by a pre-specified machine, and each machine M_k can execute an operation of a project at any one time. We want to derive a schedule that minimizes the system hazard, Θ, the maximum (over all projects) of the *normalized project flowtime* $\bar{C}_j \triangleq (C_j - r_j) / w_j$, where C_j is the completion time, r_j the release time and w_j the normalization factor of J_j.

Using an AOA type of network, the multi-project is modeled as an acyclic directed graph, called a *multi-project graph* (MPG), in which an arc represents an operation or a time delay and a node represents an event of some activity's completion. This MPG is used to identify, among others, the *dominance properties* (DPs) which will be used to reduce the number of branches

generated to search for an optimal schedule.

3.1. Dominance Properties in Multi-Project Scheduling

The DPs to be identified are based on the following observations:

OB1. Preemptions which do not improve performance must be disallowed to reduce the number of possible branches generated in the B&B algorithm.

OB2. It is undesirable to leave a machine idle when there are schedulable operations on the machine since preemptions are allowed.

OB3. It is always advantageous to use a scheduling algorithm, if any, which can reduce the completion time of a project without increasing others' if a *regular* (performance) measure is used. (A performance measure Z is said to be *regular* if the scheduling objective is to minimize Z, and Z increases only by increasing at least one project completion time in the schedule [Bak74].)

The DPs necessary for the multi-project scheduling are summarized below in theorem (and corollary) form without proofs, where O_b and O_c are two simultaneously schedulable operations (time delays are not subject to scheduling) on M_k (see [PeS88] for their formal treatment).

Theorem 1: For any regular measure, the decision for M_k as to which of O_b and O_c should be executed first is always time-invariant.

For the purpose of eliminating unnecessary preemptions, Theorem 1 simply states that if O_b should be executed before O_c (or O_b should preempt O_c) at t_0, then it will never be advantageous to let O_c preempt O_b at any future time $t > t_0$. The other necessary DPs are based on two 2-tuple of sets $\Omega(O_b) = (\Omega_0(O_b), \Omega_1(O_b))$ and $\Omega(O_c) = (\Omega_0(O_c), \Omega_1(O_c))$ associated with O_b and O_c, respectively. $\Omega_0(O_b)$ ($\Omega_0(O_c)$) is the set of nodes in MPG on M_k, each of which represents an event of project completion, and is preceded by O_b (O_c). $\Omega_1(O_b)$ ($\Omega_1(O_c)$), on the other hand, is the set of nodes in MPG on M_k, each of which is located at the ''tail'' of some time

delay, and preceded by O_b (O_c). An example showing how $\Omega_0(\bullet)$ and $\Omega_1(\bullet)$ are obtained for a operation is given in Fig. 4, where four projects, J_1, J_2, J_3 and J_4, are to be executed by two machines, M_1 and M_2, and all operations at the LHS and RHS of time delays are to be executed by M_1 and M_2, respectively.

Theorem 2: If $\Omega_0(O_b) \subseteq \Omega_0(O_c)$ and $\Omega_1(O_b) \subseteq \Omega_1(O_c)$, then it is advantageous for M_k to execute O_c before O_b w.r.t. any regular measure at any time t_0.

Theorem 2 is obvious since completing O_c before O_b will result in more schedulable operations, thus making it possible to reduce the completion times of some projects without increasing those of the others. Let the notation $O_c \Rightarrow^Z O_b$ represent that both conditions $\Omega_0(O_b) \subseteq \Omega_0(O_c)$ and $\Omega_1(O_b) \subseteq \Omega_1(O_c)$ hold.

The DPs w.r.t. Θ are used to determine the relative "superiority" (or urgency) and "equality" between any two projects in $\Omega_0(O_b)$ and $\Omega_0(O_c)$. Let J_m and J_n be two projects to be executed on M_k with release times and normalization factors r_m, w_m, r_n and w_n, respectively. J_n is said to be *superior* to J_m w.r.t. Θ, written as $J_n \ \mathrm{sp}^{\Theta} J_m$, at time t_0 if the following inequality holds:

$$(t - r_n)/w_n \geq (t - r_m)/w_m, \quad \forall\, t \geq t_0 + R^k_{t_0}(J_m, J_n), \tag{3}$$

where $R^k_{t_0}(J_m, J_n)$ is the minimum time M_k needs to complete both J_m and J_n since t_0. That is, $R^k_{t_0}(J_m, J_n)$ is the sum of the remaining execution times of all operations each of which is to be executed by M_k and precedes at least one of J_m and J_n. On the other hand, J_n is said to be *equal* to J_m w.r.t. Θ, written as $J_n \ \mathrm{eq}^{\Theta} J_m$, if $w_m = w_n$ and $r_m = r_n$. Thus, $J_n \ \mathrm{eq}^{\Theta} J_m$ iff $J_n \ \mathrm{sp}^{\Theta} J_m$ and $J_m \ \mathrm{sp}^{\Theta} J_n \ \forall\, t_0$.

The notion of superiority and equality can be easily seen in the case of a single machine multi-project scheduling problem. If $J_n \ \mathrm{sp}^{\Theta} J_m$ at any time t_0, then it is advantageous to complete J_n before J_m, and if $J_n \ \mathrm{eq}^{\Theta} J_m$, then the order of completing J_m and J_n makes no

difference. Notice that Eq. (3) holds iff the following inequality holds:

$$t_0 + R^k_{t_0}(J_m, J_n) \geq \frac{w_m r_n - w_n r_m}{w_m - w_n}, \quad \text{if } w_m > w_n, \text{ or}$$
$$r_m \geq r_n, \quad \text{if } w_m = w_n, \tag{4}$$

where the RHS of the first inequality represents a particular time t such that $(t - r_m)/w_m = (t - r_n)/w_n$. Based on the notion of superiority between two projects, O_c is said to *sub-dominate* O_b, w.r.t. Θ, written as $O_c =>^{\Theta}_0 O_b$, at t_0 if for every J_m in $\Omega_0(O_b)$ there exists a J_n in $\Omega_0(O_c)$ such that J_n sp^{Θ} J_m at t_0, and O_c is said to be *sub-similar* to O_b, w.r.t. Θ, written as O_c S^{Θ}_0 O_b, at t_0 if $O_c =>^{\Theta}_0 O_b$ and $O_b =>^{\Theta}_0 O_c$ at t_0. Accordingly, O_c is said to *dominate* O_b w.r.t. Θ at t_0, written as $O_c =>^{\Theta} O_b$, if (a) $O_c =>^{\Theta}_0 O_b$ at t_0, and (b) $\Omega_1(O_c) \supseteq \Omega_1(O_b)$, and that O_c is said to be *similar* to O_b w.r.t. Θ at t_0, written as O_c S^{Θ} O_b, if O_c S^{Θ}_0 O_b at t_0, and $\Omega_1(O_c) = \Omega_1(O_b)$. The following theorem captures the relative urgency of one operation over the other in terms of the dominance relation between them.

Theorem 3: Executing O_c before O_b at t_0 is advantageous if $O_c =>^{\Theta} O_b$ at t_0, but it makes no difference as to which of O_b and O_c is executed first at t_0 if O_c S^{Θ} O_b at t_0.

Since neither $=>^{\Theta}$ nor S^{Θ} is transitive, the following three corollaries of Theorem 3 are essential to the efficient implementation of the underlying B&B algorithm.

Corollary 1: Let $S_k(t_0) = \{ O_j , 1 \leq j \leq s \}$ be a subset of schedulable operations on M_k at t_0, where $s = |S_k(t_0)| \geq 2$.

(a) If $O_s =>^{\Theta} O_{s-1}$, $O_{s-1} =>^{\Theta} O_{s-2}$, $\cdots$, and $O_2 =>^{\Theta} O_1$ at t_0, then it is advantageous for M_k to execute O_s before O_1 at t_0.

(b) If $O_1 =>^{\Theta} O_s$ in addition to the condition in (a), then it makes no difference as to which operation of $S_k(t_0)$ is executed first at t_0.

Corollary 2: Let $S_k(t_0)$ be the same as in Corollary 1. If O_s S^{Θ} O_{s-1}, O_{s-1} S^{Θ} O_{s-2}, $\cdots$, and O_2 S^{Θ} O_1 at t_0, then it makes no difference as to which operation in $S_k(t_0)$ is executed first at t_0.

Within $S_k(t_0)$, $O_s\ S^{\Theta}\ O_{s-1}$, $O_{s-1}\ S^{\Theta}\ O_{s-2}$, $\cdots$, and $O_2\ S^{\Theta}\ O_1$ at t_0 do not necessarily imply that $O_s \Rightarrow^{\Theta} O_{s-1}$, $O_{s-1} \Rightarrow^{\Theta} O_{s-2}$, $\cdots$, $O_2 \Rightarrow^{\Theta} O_1$, and $O_1 \Rightarrow^{\Theta} O_s$ at t_0; the converse is not true either. For convenience, the set of schedulable operations for which execution order is immaterial is called an *immaterial set.*

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 immaterial sets of M_k at t_0. If there exists $O_b \in \Pi_k^1(t_0)$ 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, $\forall\, i, j$.

Corollaries 1 and 2 are useful because they show that, even though neither $\Rightarrow^{\Theta}$ nor S^{Θ} is transitive, the execution orders of operations implied by these two relations are transitive. Moreover, since $O_c \Rightarrow^{\Theta} (S^{\Theta})\ O_b$ at t_0 implies those at $t_1 \geq t_0$, Corollary 3 simply states that it is advantageous to execute all operations in $\Pi_k^2(t_0)$ before any operation in $\Pi_k^1(t_0)$. Corollary 3 -- which deals with the "uninterruptability" of an immaterial set -- can be thought of as a different version of Theorem 1, which deals with the uninterruptability of a single operation.

The DPs described in Theorems 1-3 and Corollaries 1-3 are used to generate a subset of active schedules which contains at least one optimal schedule as follows. Theorems 2 and 3 suggest that it is never advantageous for M_k to execute O_b at t_0 if there exists another schedulable operation O_c such that (T1): $O_c \Rightarrow^{Z} O_b$ and O_c does not equal O_b, and/or (T2): $O_c \Rightarrow^{\Theta} O_b$ and (a) O_c is not similar to O_b or (b) O_c and O_b are not a part of either dominance cycle described in Corollary 2 or Part (b) of Corollary 1. Moreover, T1 should be tested before T2 because $\Rightarrow^{Z}$ implies $\Rightarrow^{\Theta}$ $\forall\, t_0$. (Note that $\Rightarrow^{\Theta}$ does not always imply $\Rightarrow^{Z}$, though.) The DPs described in Theorem 1 and Corollaries 1-3 suggest that the B&B algorithm should keep track of the operations (immaterial sets) on a stack L_k (I_k) for each M_k that have been partially executed.[5] The operation (immaterial set) on the top of L_k (I_k) is the one being executed by M_k.

[5]By 'an immaterial set has been partially executed', we mean that at least one of its elements has been executed and there exist

An operation (immaterial set) is pushed into L_k (I_k) if the operation (the immaterial set containing the operation) is to preempt the operation (immaterial set), if any, on the top of L_k (I_k). Similarly, an operation (immaterial set) is popped off L_k (I_k) if it is completed, and the operation (immaterial set) to be executed next is chosen in a LIFO fashion from the stack provided no preemptions occur. Therefore, the depth of the search tree of the B&B algorithm is at most twice the total number of operations to be scheduled. A complete account of this algorithm is given in [PeS88].

3.2. Lower-Bound Costs of a Non-Terminal Vertex

For a given non-terminal vertex (i.e., a partial schedule) y, we shall derive two lower-bound costs $\hat{\Theta}_1(y)$ with computational complexity $O(|M|(\bar{P}\log\bar{P}+\log|M|))$, and $\hat{\Theta}_2(y)$ with complexity $O(|M|(\bar{P}^2+\log|M|))$, of $\Theta^*(y)$, where $|M|$ is the number of machines in the system and $\bar{P}$ the average number of uncompleted operations on each machine. Both $\hat{\Theta}_1(y)$ and $\hat{\Theta}_2(y)$ are derived by ignoring the precedence constraints between machines, and $\hat{\Theta}_1(y)$ is a lower-bound of $\hat{\Theta}_2(y)$. Thus, $\hat{\Theta}_2(y)$ is a better estimate of $\Theta^*(y)$ than $\hat{\Theta}_1(y)$, but requires more computation to derive.

For a non-terminal vertex y at t_1, let $g_j(y)$ be the actual path cost of J_j from the root to y, and $h_j(y)$ be the cost of J_j from y to a terminal vertex (i.e., a complete schedule) provided that J_j is executed last on its corresponding machine, say M_k. Then, $g_j(y)$ can be computed from the partial schedule y as the normalized "partial" flowtime of J_j at t_1:

$$g_j(y)=\begin{cases}\bar{C}_j & \text{if } J_j \text{ is completed before } t_1,\\ (t_1-r_j)/w_j & \text{otherwise,}\end{cases} \tag{5}$$

where $\bar{C}_j$, r_j and w_j are the normalized flowtime, release time and normalization factor of J_j, respectively. Note that $g_j(y)<0$ if J_j is not yet released at t_1. Let $R^k_{t_1}(*)$ be the sum of the remaining execution times of all unfinished operations at t_1 on M_k which precede at least one

at least an element that has not yet been completed.

project on M_k. Then,

$$h_j(y) = \begin{cases} 0 & \text{if } J_j \text{ is completed before } t_1, \\ R^k_{t_1}(*)/w_j & \text{otherwise,} \end{cases} \tag{6}$$

and $g_j(y) + h_j(y)$ is the total cost of J_j provided that J_j is executed last. Thus, $\hat{\Theta}_1(y) \triangleq \max_{M_k} \{ \min_{J_j \in B_k} \{ g_j(y) + h_j(y) \} \}$ is a lower-bound cost, where B_k is the set of projects on M_k.

$\hat{\Theta}_2(y)$ is obtained by computing the actual cost (i.e., Θ) for each M_k by applying the optimal scheduling algorithm, called *Algorithm A*, of [BLL83] on the set of unfinished operations. For completeness, Algorithm A of complexity $O(N^2)$, where N is the number of jobs[6] to be scheduled, is briefly described below. 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 task completion cost.

SA1. Modify job release times, where possible, to meet the precedence constraints between 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, if jobs X, Y and Z are released at $t = 0, 2, 15$, respectively, X precedes Y which in turn precedes Z, and 5 units of time are required to complete each of X and Y, then job Y's release time is modified to $t = 5$, Z's is kept at $t = 15$ and two blocks of jobs {X,Y} and {Z} will be created.

SA2. Consider a block B with block completion time $t(B)$. Let B' be the set of jobs in B which do not precede any other jobs in B. Select a job l such that $f_l(t(B))$ is the minimum among all jobs in B', 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.

[6] As we shall see; the "job" in Algorithm A corresponds to the "operation" of our Algorithm B.

SA3. Create subblocks of jobs in the set $B - \{ l \}$ by arranging the jobs in nondecreasing order of modified release 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) allotted to l is 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.

Let $\eta_k^*(y)$ be the mini-max normalized project flowtime resulting from the application of Algorithm A on the set of unfinished operations on M_k, and $\eta^*(y) = \max_{M_k} \eta_k^*(y)$. Then, $\hat{\Theta}_2(y)$ can be obtained as $\hat{\Theta}_2(y) = \max \{ g(y), \eta^*(y) \}$, where $g(y) = \max_{J_i} g_i(y)$. As mentioned earlier, $\hat{\Theta}_2(y) \geq \hat{\Theta}_1(y)$, $\forall y$, for $\hat{\Theta}_1(y)$ is a lower-bound of $\hat{\Theta}_2(y)$.

As was shown by the computational experience in [PeS88], Algorithm B turned out to be quite efficient. It can be directly used, via Table 1, to derive the system hazard Θ of a complete task assignment. This optimal scheduling algorithm for the complete assignment of tasks is what we have called the B&BS algorithm. One minor difference between Algorithm B and B&BS is that, while it is possible for M_k to execute any project, N_k must execute all invocations of its assigned tasks within a planning cycle.

4. LOWER-BOUND COST OF A NON-TERMINAL VERTEX IN B&BA ALGORITHM

The B&BS algorithm presented in the last section is necessary for the B&BA algorithm to find an optimal assignment. But, using the B&BS algorithm also for all the non-terminal vertices (i.e., partial assignments) of the B&BA algorithm is too costly and unnecessary. This is because the B&BA algorithm is guaranteed to find an optimum assignment as long as the cost estimate for each non-terminal (terminal) vertex is a lower-bound (true value) of the optimal cost for this vertex (see, e.g., [KoS76]). A looser, but inexpensive, lower-bound cost may be more attractive than a tighter, expensive one. In this section, we present such a lower-bound cost which is

obtainable in polynomial time.

The lower-bound cost $\hat{\Theta}(x)$ of a non-terminal vertex x is a lower-bound estimate of $\Theta^*(x)$, the minimum system hazard among the set of all complete assignments each with x as its partial assignment. The non-terminal vertex x can be represented as a subset of $\{(T_i, N_k), T_i \in T, N_k \in N\}$, where each pair (T_i, N_k) represents the assignment of T_i to N_k.

The following two steps are necessary to derive $\hat{\Theta}(x)$ for each x.

- Construct a precedence graph $TG_k(x)$, $k = 1, 2, \cdots, |N|$, from the original task graph TG.
- Apply an optimal scheduling algorithm ζ on a simplified version of $TG_k(x)$ to obtain the lower-bound cost $\hat{\Theta}_k(x)$. The desired $\hat{\Theta}(x)$ is then chosen as the maximum of $\hat{\Theta}_k(x)$ over all k.

These steps are detailed in the following two subsections.

4.1. Obtaining $TG_k(x)$

Each $TG_k(x)$ is composed of two subgraphs, $TG1(x)$ and $TG2_k(x)$: $TG1(x)$ represents those tasks already assigned within x and is common for all k, whereas $TG2_k(x)$ represents the load imposed on N_k by those tasks that have not yet been assigned within x.

For each x, let $B_k(x)$ be the set of tasks already assigned to N_k, $B(x) = \bigcup_k B_k(x)$, and $\bar{B}(x) = T - B(x)$, the set of tasks not yet assigned within x. $TG1(x)$ can be obtained from TG by the following steps.

Step 1. $\forall a$, set $e_{ia}^k := e_{ia} / q_k^i$ if $T_i \in B_k(x)$, and $e_{ia}^k := 0$ if $T_i \in \bar{B}(x)$. This represents the computation load on N_k contributed by T_i under the partial assignment x.

Step 2. Case 1: $T_i \in B_k(x)$ and $T_j \in B_h(x)$, $h \neq k$.

Set $\chi_{ij,i}^k(1) := \chi_{ij,i}(1) / q_k^i$ and $\chi_{ji,i}^k(1) := \chi_{ji,i}(1) / q_k^i$ to represent the communication load between T_i and T_j on N_k if T_j is assigned to N_h.

Case 2: $T_i \in B_k(x)$ and either $T_j \in B_k(x)$ or $T_j \in \bar{B}(x)$.

Set $\chi^k_{ij,i}(0) := \chi_{ij,i}(0) / q^i_k$ and $\chi^k_{ji,i}(0) := \chi_{ji,i}(0) / q^i_k$ to represent the communication load between T_i and T_j on N_k if T_j is also assigned to N_k or not yet assigned at all.

Case 3: $T_i \in \bar{B}(x)$.

Set $\chi^k_{ij,i}(s) := \chi^k_{ji,i}(s) := 0, s = 0, 1$.

Step 3. Set $\bar{\chi}^{kh}_{ij} := \bar{\chi}_{ij} d_{hk}$ if $T_i \in B_k(x)$ and $T_j \in B_h(x)$. Otherwise, set $\bar{\chi}^{kh}_{ij} := 0$ to represent the case where at least one of T_i and T_j is not yet assigned within x.

Step 4. Simplify and restructure the resulting task graph to obtain $TG1(x)$ by deleting and/or adding *dummy* modules to preserve the precedence and timing constraints. (A *dummy* module is a module with zero computation time or communication delay.)

The resulting $TG1(x)$ is the partial task system corresponding to those tasks already assigned within x, while ignoring those tasks not yet assigned. For example, consider the TG shown in Fig 1, where the three tasks T_1, T_2 and T_3 are to be assigned to N_1 and N_2 with $d_{12} = d_{21} = 1$. Assume $q^1_1 = q^2_1 = q^3_1 = 2$ and $q^1_2 = q^2_2 = q^3_2 = 1$. Let $x = \{(T_1, N_1), (T_2, N_2)\}$, i.e., T_1 is assigned to N_1, T_2 to N_2 and $\bar{B}(x) = \{T_3\}$. The resulting $TG1(x)$ is shown in Fig. 5 with all related execution times and actual communication delays properly indicated, but all modules associated with T_3, an unassigned task, are ignored. While all the nodes associated with T_3 are merged and deleted in the restructured TG, n_{12} and n_{15} cannot be merged because the earliest time n_{15} can be realized is 20 (T_{32}'s invocation time), but n_{12} may be realized before or after t=20. It is interesting to see $TG1(x)$ in Fig. 6, where $x = \{(T_1, N_1), (T_3, N_2)\}$, and a dummy module between n_4 and n_{16}, and another between n_5 and n_{17} are added to preserve the precedence constraints between modules of T_1 and T_3 as T_2 is ignored.

$TG2_k(x)$ represents the total minimum load on N_k imposed by the tasks in $\bar{B}(x)$, and thus, can be expressed as $TG2_k(x) = \{ \psi^k_{iv}(x), v = 1, 2, \ldots, l/p_i, T_i \in \bar{B}(x) \}$, where $\psi^k_{iv}(x)$ represents

the minimum load imposed on N_k by T_{iv}, the v-th invocation of $T_i \in \bar{B}(x)$, and l is the length of a planning cycle. $\psi_{iv}^k(x)$ is derived by considering whether T_i will be assigned to N_k or not. For each $T_i \in \bar{B}(x)$, let $MIL_{iv}^k(x)$ ($\overline{MIL}_{iv}^k(x)$) denote the *Minimum Imposed Load* on N_k by T_{iv} when T_i is (is not) to be assigned to N_k. $MIL_{iv}^k(x)$ consists of three parts: 1) T_{iv}'s computation modules, 2) communication modules $X_{ij,i}$'s and $X_{ji,i}$'s between T_{iv} and T_j's invocations that have already been assigned to N_h, $h \neq k$, and 3) all the other communication modules to those tasks that either have been assigned to N_k or not yet assigned. Because of the way $TG1(x)$ is constructed (Step 2), $\overline{MIL}_{iv}^k(x)$ consists of only one part: the sum of $(\chi_{ji,j}^k(1) - \chi_{ji,j}^k(0))$'s and $(\chi_{ij,j}^k(1) - \chi_{ij,j}^k(0))$'s for each communication between $T_j \in B_k(x)$ and T_{iv}. In other words, $\overline{MIL}_{iv}^k(x)$ is an extra load on N_k to facilitate the IPC between N_k and N_h, $k \neq h$, to which T_{iv} is assigned. Since each module of $MIL_{iv}^k(x)$ ($\overline{MIL}_{iv}^k(x)$) usually has a different release time and deadline, and is necessary for the completion of different tasks, it is very difficult to determine the actual minimum load imposed by T_{iv} without further simplification of $MIL_{iv}^k(x)$ and $\overline{MIL}_{iv}^k(x)$. We have taken the following simplifying steps:

SP1. Since only those modules which are required for T_{iv}'s completion are of interest to us, those modules unnecessary for T_{iv}'s completion are excluded from $MIL_{iv}^k(x)$ and $\overline{MIL}_{iv}^k(x)$.

SP2. $MIL_{iv}^k(x)$ and $\overline{MIL}_{iv}^k(x)$ of SP1 are then treated as two independent single modules with the following execution times $E_{iv}^k(x)$ and $\bar{E}_{iv}^k(x)$, respectively:

$$E_{iv}^k(x) = \sum_a e_{ia}^k + \sum_{\substack{T_j \in B_k(x), \\ h \neq k}} \left[\chi_{ij,i}^k(1) + \chi_{ji,i}^k(1) \right] + \sum_{\substack{T_j \in B_k(x) \text{ or} \\ T_j \in \bar{B}(x)}} \left[\chi_{ij,i}^k(0) + \chi_{ji,i}^k(0) \right], \tag{7}$$

$$\bar{E}_{iv}^k(x) = \sum_{T_j \in B_k(x)} \left\{ \left[\chi_{ji,j}^k(1) - \chi_{ji,j}^k(0) \right] + \left[\chi_{ij,j}^k(1) - \chi_{ij,j}^k(0) \right] \right\}. \tag{8}$$

Note that the three terms of the RHS of Eq. (7) correspond to the three parts of $MIL_{iv}^k(x)$ described above.

It can be seen that if $MIL_{iv}^{k}(x)$ ($\overline{MIL}_{iv}^{k}(x)$) of SP1, $\forall i, v$ and k, together with $TG\,1(x)$ generates a lower-bound of $f^{*}x$, then so does $MIL_{iv}^{k}(x)$ ($\overline{MIL}_{iv}^{k}(x)$) of SP2 together with $TG\,1(x)$, because the single-module version of $MIL_{iv}^{k}(x)$ ($\overline{MIL}_{iv}^{k}(x)$) has less precedence and timing constraints than its counterpart of SP1. Even though each of $MIL_{iv}^{k}(x)$ and $\overline{MIL}_{iv}^{k}(x)$ of SP2 is needed for T_{iv}'s completion, they are not necessarily released at the same time. This is because $\overline{MIL}_{iv}^{k}(x)$ is part of the T_{iv}'s communication partner T_j, whose period is p_j, whereas $MIL_{iv}^{k}(x)$ is part of T_{iv}, whose period is p_i. Therefore, it is still difficult to determine the minimum imposed load by simply comparing $E_{iv}^{k}(x)$ and $\bar{E}_{iv}^{k}(x)$. Based on these $MIL_{iv}^{k}(x)$ and $\overline{MIL}_{iv}^{k}(x)$, SP3 below remedies this difficulty.

SP3. The minimum load $\psi_{iv}^{k}(x)$ imposed on N_k by T_{iv} is constructed with a) lumped execution time $\lambda_{iv}^{k}(x) = \min\{\, E_{iv}^{k}(x), \bar{E}_{iv}^{k}(x)\,\}$, b) release time $\sigma_{iv}^{k}(x)$ = minimum of the release times of $MIL_{iv}^{k}(x)$ and $\overline{MIL}_{iv}^{k}(x)$, and c) cost function = $(t - r_{iv}) / p_i$.

The above $\psi_{iv}^{k}(x)$ is indeed the minimum load imposed on N_k by T_{iv} in the sense that $\psi_{iv}^{k}(x)$'s together with $TG\,1(x)$ generate a lower-bound of $\Theta^{*}(x)$ regardless whether T_i is assigned to N_k or to a different PN. With $\psi_{iv}^{k}(x)$ determined for T_{iv}, the minimum load imposed by T_i on N_k is constructed as $\Psi_i^{k}(x) \triangleq \bigcup_{v} \psi_{iv}^{k}(x)$, and $TG\,2_k(x)$, the minimum load imposed by all the tasks in $\bar{B}(x)$, is calculated as $TG\,2_k(x) = \bigcup_{T_i \in \bar{B}(x)} \Psi_i^{k}(x)$. Finally, calculate $TG_k(x) \triangleq TG\,1(x) \cup TG\,2_k(x)$.

4.2. Lower-Bound Cost $\hat{\Theta}(x)$

For each k, $\hat{\Theta}_k(x)$, the lower-bound node hazard of N_k, (and thus, $\hat{\Theta}(x) \triangleq \max_{N_k \in N} \hat{\Theta}_k(x)$) can be obtained by applying an optimal scheduling algorithm on the partial task graph $TG_k(x) = TG\,1(x) \cup TG\,2_k(x)$. Since the problem of deriving such an $\hat{\Theta}_k(x)$ is NP-hard, some form of approximation is called for. Some of precedence and/or timing constraints on $TG_k(x)$

need to be relaxed to derive an approximate polynomial-time algorithm. Several candidate $\hat{\Theta}_k(x)$'s can be obtained depending on the extent to which these constraints are relaxed. For example, on $TG\,1(x)$, we may ignore both timing and precedence constraints between PNs and derive an $\hat{\Theta}_k(x)$ as was done in deriving $\hat{\Theta}_1(y)$ or $\hat{\Theta}_2(y)$ in Section 3. Even though $\hat{\Theta}_2(y)$ is a better bound than $\hat{\Theta}_1(y)$, both of them are obtained without considering the completion of any other task on different PNs. In what follows, a third method for deriving an $\hat{\Theta}_k(x)$ will be described by considering both task completions on N_k as well as on other PNs. Thus, the resulting $\hat{\Theta}_k(x)$ should be the best among the three.

Since Algorithm A is to be used for N_k, the main theme in deriving a better $\hat{\Theta}_k(x)$ relies solely on how to embed the effects of those tasks of $TG\,1(x)$ located in PNs other than N_k into the release times and cost functions of those modules of $TG\,1(x)$ located in N_k. The release time of any module m_b of $TG\,1(x)$ (the release times and the cost functions of $TG\,2_k(x)$ were already determined in the last subsection) is relatively easy to obtain because it is the latest task invocation time among those having at least a module preceding m_b. For example, M_{24} (and also the dummy $X_{23,2}$) of Fig. 5 has a release time of 0 since T_{11}, T_{21} and T_{31} are all invoked at time 0 (see also Fig. 1). However, the release time of $X_{32,2}$ is 20 because T_{32} is invoked at time 20.

To derive the cost function for each module of $TG\,1(x)$ located in N_k, the notion of *outgoing communication point* (OCP) needs to be introduced. An OCP of N_k is a node on $TG\,1(x)$ which is located at the tail of a delay module. A module m_b in N_k is called an *OCP module* if $head(m_b)$ is an OCP, i.e., a module whose completion may enable some modules in PNs other than N_k. See Fig. 5 for an example. Since n_4 is an OCP of N_1, $X_{12,1}$ is an OCP module. Consider an OCP and a node $n_q = v_y$ on $TG\,1(x)$, i.e., n_q is the node representing the completion of certain task invocation v_y. Assume that v_x has been assigned to a PN other than N_k, where the OCP is located. If the OCP precedes v_x (i.e., n_q), then the *critical path* (it may contain some modules in N_k) of length β_x from the OCP to v_x represents the minimum time to

complete v_x after the realization of the OCP. Otherwise, the completion time of v_x is independent of the realization time of the OCP.

Let $V(m_b)$ denote the set of all invocations preceded by the OCP $head(m_b)$. Also, let v_y be a task invocation in N_k, and $f_y(t)$ be the cost function, r_y the invocation time and p_y the period of v_y. The cost function $f_b(t)$ of m_b can now be obtained using the following rules.

R1. If m_b is not a (invocation) completing module and m_b is not an OCP module then $f_b(t) := 0$.

R2. If m_b is not a completing module and m_b is an OCP module **then** $f_b(t) := \max_{v_n \in V(m_b)} g_n(t)$, where $g_n(t) = (t + \beta_n - r_n) / p_n$, and β_n is the length of the critical path from $head(m_b)$ to v_n.

R3. If $head(m_b) = v_y$ and m_b is not an OCP module **then** $f_b(t) := f_y(t) = (t - r_y) / p_y$, the cost function of v_y.

R4. If $head(m_b) = v_y$ and m_b is an OCP module **then** $f_b(t) := \max \{ f_y(t), \max_{v_n \in V(m_b)} g_n(t) \}$, where $g_n(t)$ as is in $R2$.

Once the release time and cost function of each module on $TG_k(x)$ located in N_k are determined, Algorithm A can be applied to obtain $\hat{\Theta}_k(x)$, and thus, $\hat{\Theta}(x) = \max_{N_k \in N} \hat{\Theta}_k(x)$. Notice that the optimal schedule obtained above is only for N_k on this simplied version of $TG_k(x)$. It is not necessarily optimal for another PN, say N_h, on that of $TG_h(x)$, rather the resulting $\hat{\Theta}(x)$ is a lower-bound of $\Theta^*(x)$ because of the cost function selection rules of the module considered. Also, the computational complexity of deriving $\hat{\Theta}(x)$ is $O(|N|(Q^2 + \log|N|))$, where $|N|$ is the number of PNs and Q the *total* number of modules in the system.

5. AN EXAMPLE

Consider an example of allocating the three tasks T_1, T_2 and T_3 to two PNs, N_1 and N_2, in Fig. 1. Within the planning cycle [0, 40), T_1 and T_2, both with period 40, are invoked only once while T_3, with period 20, is invoked twice. Suppose $q_1^1 = 2$, $q_1^2 = 1$, $q_1^3 = 1$, $q_2^1 = 1$, $q_2^2 = 2$, $q_2^3 = 1$, and $d_{12} = d_{21} = 1$.

Fig. 7 shows all the vertices x's, numbered in the order of times of their generation. The assignment and $\hat{\Theta}(x)$ associated with each vertex x are also indicated in Fig. 7. Our B&B algorithm is shown to be quite efficient in this example because only two (out of a total of 8) terminal vertices are generated before an optimal assignment is found. Specifically, vertex 6, with $\hat{\Theta}(6) = \frac{39}{40}$, is first eliminated as soon as vertex 8 with the exact cost (i.e., system hazard Θ) $\frac{28}{40} \le \hat{\Theta}(6)$ is generated. Then, all active vertices 4, 5, and 8 are eliminated after vertex 9 is generated since the value of Θ of this complete assignment is 23.5 / 40 is the smallest of all these vertices. Thus, vertex 9, which assigns all three tasks to N_2, is an optimal assignment, and its optimal schedule derived by Algorithm A is shown in Fig. 8. One reason for this counter-intuitive result was already given in Section 2; communication modules and delays are the major part of the TG, and thus, assigning all tasks to a single PN to minimize the communication and delays is superior to the others. There is another reason why all tasks are not assigned to N_1. Since T_{32} is invoked at t=20, the length of a path from n_{15} to n_{20} (Fig. 1) is critical to the performance of the assignment. If all tasks are assigned to N_1 which has a smaller processing power for T_2 than N_2 does, then this path will be longer than the optimal solution.

To see how $\hat{\Theta}(x)$ is obtained for a non-terminal vertex x, $\hat{\Theta}(5)$ is computed as follows. Since $B_1(5)=\{T_1\}$, $B_2(5)=\{T_2\}$, $\bar{B}(5)=\{T_3\}$, and $q_1^1 = q_2^2 = 2$, $e_{1a}^1 := e_{1a}/2$, $e_{2a}^2 := e_{2a}/2$, $\forall a$, $\chi_{12,1}^1(1) := \chi_{12,1}(1) / 2$, $\chi_{21,1}^1(1) := \chi_{21,1}(1) / 2$, $\chi_{12,2}^2(1) := \chi_{12,2}(1) / 2$, $\chi_{21,2}^2(1) := \chi_{21,2}(1) / 2$, $\chi_{23,2}^2(0) := \chi_{23,2}(0) / 2$, $\chi_{32,2}^2(0) := \chi_{32,2}(0) / 2$ to accommodate the loads imposed on N_1 and N_2 by

the tasks in $B_1(5)$ and $B_2(5)$, respectively. The resulting $TG\,1(5)$ is shown in Fig. 9, ignoring T_3 and the communication delays between T_3 and T_2. (Note the differences between this figure and Fig. 5.)

To derive $\hat{\Theta}_1(5)$, we need to obtain the minimum load $TG\,2_1(5)$ imposed only by T_3 on N_1 since $\bar{B}(5) = \{T_3\}$. Thus, $TG\,2_1(5) = \{\Psi_3^1(5)\} = \{\psi_{31}^1(5), \psi_{32}^1(5)\}$ since T_3 is invoked twice in one planning cycle. However, a further examination on the original TG shows that there is no extra load imposed on N_1 by T_3, if T_3 is not assigned to N_1, because T_3 does not communicate with T_1. It follows that, though assigning T_3 to N_1 will impose some load on N_1, we need not consider T_3 to derive $\hat{\Theta}_1(5)$. That is, $\hat{\Theta}_1(5)$ can be obtained by scheduling only $TG_1(5) = TG\,1(5)$. The release time and cost function of each module in T_1 are determined as follows. Since no modules of T_1 are preceded by T_{32}, each of them has a release time of $t = 0$. To find the cost functions, rules R1-R4 of the last section are followed. Specifically, from R3 the cost functions of M_{13} and $X_{21,1}$ (Fig. 9) is $t/40$, which is identical to that of T_1. From R1, the cost functions of M_{11} and M_{12} are 0. Finally, the cost function of $X_{12,1}$ is determined by considering the critical paths of lengths β_1 and β_2 from n_4, which is an OCP, to n_{19} and n_{20}, respectively. That is, from R2, the cost function of $X_{12,1}$ is $\max\{g_1(t), g_2(t)\} = g_1(t)$, where $g_1(t) = (t + \beta_1 - 0)/40 = (t + 25.5)/40$ and $g_2(t) = (t + \beta_2 - 0)/40 = (t + 13)/40$. As a result of applying Algorithm A, $\hat{\Theta}_1(5) = \dfrac{27.5}{40}$.

To derive $\hat{\Theta}_2(5)$, we need to obtain the minimum load $TG\,2_2(5) = \{\Psi_3^2(5)\} = \{\psi_{31}^2(5), \psi_{32}^2(5)\}$ imposed by T_3 on N_2 when T_3 is and is not assigned to N_2. If T_3 is assigned to N_2, from Eq. (7), $E_{31}^2(5) = 4$ ($E_{32}^2(5) = 5$) since all modules of T_3 contributing to the summation are required for T_{31}'s (T_{32}'s) completion. On the other hand, if T_3 is not assigned to N_2, the extra loads imposed on N_2 (Eq. (8)) are $\bar{E}_{31}^2(5) = (2-1)/2 = 0.5$ and $\bar{E}_{32}^2(5) = (2-0)/2 = 1$ since the first (second) $X_{32,2}$ is not required for T_{31}'s (T_{32}'s) completion. Following SP3 in the last section, we derive the $\psi_{31}^2(5)$'s execution time as $\lambda_{31}^2(5) = \min\{4, 0.5\} = 0.5$, release time as

$\sigma^2_{31}(5) = 0$, and cost function as $(t - 0)/20$. $\psi^2_{32}(5)$ can be derived similarly to $\psi^2_{31}(5)$ except for $\sigma^2_{32}(5)$. Specifically, $\lambda^2_{32}(5)$ = min {5, 1} = 1, cost function = $(t - 20) / 20$, and $\sigma^2_{32}(5) = 0$ since $X_{23,2}$ is executable before T_{32} is invoked. In order to use Algorithm A for deriving $\hat{\Theta}_2(5)$, the release time and cost function of each module in N_2 on TG 1(5) also need to be determined. As a result of scheduling the simplified version of $TG_2(5)$ (the union of $\Psi^2_3(5)$ and the simplified version of TG 1(5)), we get $\hat{\Theta}_2(5) = \frac{20}{40}$. Therefore, $\hat{\Theta}(5) = \max\{\hat{\Theta}_1(5), \hat{\Theta}_2(5)\} = \hat{\Theta}_1(5) = \frac{27.5}{40}$.

6. CONCLUSION

Task allocation (assignment with the subsequent scheduling considered) is one of the most important design issues in distributed real-time systems. However, this problem is generally known to be NP-hard even without considering the precedence constraints among tasks. In this paper, we have addressed the problem of allocating a set of periodic tasks with precedence constraints among the set of processing nodes of a distributed real-time computing system. We solved this problem by using two B&B algorithms. Evaluation of the performance of a given assignment is equivalent to optimally scheduling assigned tasks and was handled by a B&B algorithm guided with the dominance properties between simultaneously schedulable modules.

Although more computational experience with the proposed algorithm needs to be gained, we believe that both the dominance properties and the lower-bound costs present in this paper can ease the computational difficulty significantly. This fact has been justified in [PeS88] and by the demonstrative example in this paper to some extent.

Because of the enumeration nature of the proposed B&B algorithms, it is also possible to extend the proposed approach to deal with task allocation problems with other resource (e.g. memory and communication bandwidth) constraints. To ease computation requirements further, a tighter and easily obtainable lower-bound cost estimate needs to be derived for each non-terminal vertex in the search tree. Alternatively, an approximate algorithm which efficiently finds

a suboptimal assignment with known performance should be practically valuable. This is a matter of our future inquiry.

REFERENCES

[Bak74] K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley & Sons, 1974.

[BEN82] R. Bellman, A. O. Esogbue and I. Nabeshima, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, 1982, pp. 281-321.

[BLL83] K. R. Baker, *et al.*, "Preemptive Scheduling of A Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints", *Operations Research*, Vol. 31, No. 2, Mar-Apr. 1983, pp. 381-386.

[ChA83] T. C. K. Chow and J. A. Abraham, "Load Redistribution Under Failure in Distributed Systems", *IEEE Trans. on Computers*, Vol C-32, No. 9, Sep. 1983, pp. 799-808.

[CHL80] W. W. Chu, *et al.*, "Task Allocation in Distributed Data Processing", *IEEE Computer*, Vol. 13, Nov. 1980, pp. 57-69.

[ChL87] W. W. Chu and L. M. Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems", *IEEE Trans. on Computers*, Vol. C-36, No. 6, Jun. 1987, pp. 667-679.

[Chu69] W. W. Chu, "Optimal File Allocation in a Multiple Computing System", *IEEE Trans. on Computers*, Vol. C-18, Oct. 1969, pp. 885-889.

[Cof76] E. G. Coffman, *Computer and Job-Shop Scheduling Theory*, New York, Wiley and Sons, 1976.

[DaF84] B. Dasarathy and M. Feridun, "Task Allocation Problems in the Synthesis of Distributed Real-Time Systems", *Proc. 5th IEEE Real-Time System Symposium*, Dec. 1984, pp. 135-144.

[DhL78] S. K. Dhall and C. L. Liu, "On a Real-Time Scheduling Problem", *Opns Res*, Vol. 26, No. 1, 1978, pp. 127-140.

[Efe82] K. Efe, "Heristic Models of Task Assignment Scheduling Distributed Systems", *IEEE Computer*, Vol. 15, No. 6, Jun. 1982, pp. 50-56.

[ElH80] O. I. El-Dessouki and W. H. Huan, "Distributed Enumeration on Network Computers", *IEEE Trans. on Computers*, Vol. C-29, Sep. 1980, pp. 818-825.

[Fre82] S. French, *Sequencing and Scheduling*, Halsted Press, 1982.

[GaJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, N.Y., 1979.

[Gon77] M. J. Gonzalez, "Deterministic Processor Scheduling", *ACM Computing Surveys*, Vol. 9, No. 3, Sep. 1977, pp. 173-204.

[GoS78] T. Gonzalez and S. Sahni, "Flowshop and Jobshop Schedules : Complexity and Approximation", *Operations Research*, Vol. 26, No. 1, Jan-Feb. 1978, pp. 36-52.

[KaN84] H. Kasahara and S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing", *IEEE Trans. on Computers*, Vol. C-33, No. 11, Nov. 1984, pp. 1023-1029.

[KoS76] W. H. Kohler and K. Steiglitz, "Enumerative and Iterative Computational Approach" in *Computer and Job-Shop Scheduling Theory*, Coffman *eds.*, Wiley and Sons, 1976, pp. 229-287.

[LeK78] J. K. Lenstra and A. H. G. R. Kan, "Complexity of Scheduling under Precedence Constraints", *Operations Research*, Vol. 26, No. 1, Jan-Feb. 1978, pp. 23-35.

[LiL73] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *J. of ACM*, Vol. 20, No. 1, 1973, pp. 46-61.

[LLK82] E. L. Lawler, *et al.*, "Recent Developments in Deterministic Sequencing and Scheduling: A Survey", in *Deterministic and Stochastic Scheduling*, Dempster, *et al.* (eds), Reidel, Dordrecht, The Netherlands, 1982, pp. 35-74.

[LRB77] J. K. Lenstra, A. H. G. Rinnooy Kan and P. Brucker, "Complexity of Machine Scheduling Problems", *Ann. Discrete Math.*, Vol. 1, 1977, pp. 343-362.

[MLT82] P. Y. R. Ma, *et al.* "A Task Allocation Model for Distributed Computing Systems", *IEEE Trans. on Computers*, Vol. C-31, No. 1, Jan. 1982, pp. 41-47.

[PeS87] D. Peng and K. G. Shin, "Modeling of Concurrent Task Execution in a Distributed System for Real-Time Control", *IEEE Trans. on Computers*, Vol. C-36, No. 4, Apr. 1987, pp. 500-516.

[PeS88] D. Peng and K. G. Shin, "Multiproject Scheduling Using an Enumeration Method", submitted for publication.

[PrK84] C. C. Price and S. Krishnaprasad, "Software Allocation Models for Distributed Computing Systems", *Proc. IEEE, 4th Int'l Conf. on Distributed Computing Systems*, May 1984, pp. 40-48.

[Ser72] O. Serlin, "Scheduling of Time Critical Processes", *Proc. of AFIPS 1972 Spring Joint Computer Conf.*, AFIPS Press, Montvale, N. J., 1972, pp. 925-932.

[ShT85] C. C. Shen and W. H. Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion", *IEEE Trans. on Computers*, Vol. C-34, No. 3, Mar. 1985, pp. 197-203.

[StB78] H. S. Stone and S. H. Bokhari, "Control of Distributed Processes", *IEEE Computer*, Vol. 11, Jul. 1978, pp. 97-106.

[Sto77] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithm", *IEEE Trans. on Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.

[Sto78] H. S. Stone, "Critical Load Factors in Distributed Computer Systems", *IEEE Trans. on Software Engineering*, Vol. SE-4, No. 5, May 1978, pp. 254-258.

[Tah76] H. A. Taha, *Operations Research: An Introduction*, Macmillan Publishing Co. NY, 1976, pp. 357-366.

[Vir84] M. L. Virginia, "Heuristic Algorithms for Task Assignment in Distributed Systems", *Proc. IEEE, 4th Int'l Conf. on Distributed Computing Systems*, May 1984, pp. 30-39.

[WoS74] R. E. D. Woolsey and H. S. Swanson, *Operations Research for Immediate Applications: A Quick and Dirty Manual*, Harper and Row, 1974.

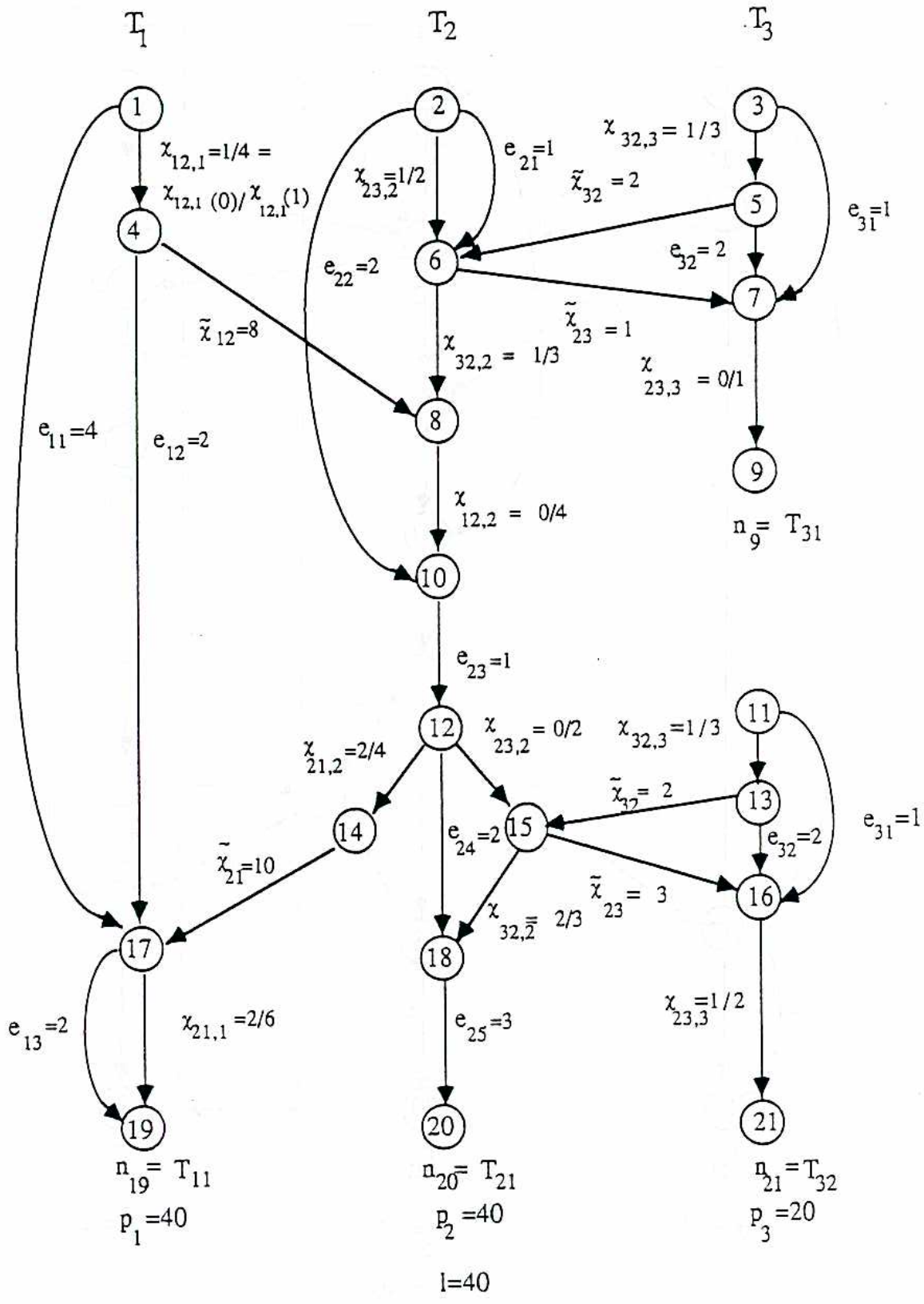


Figure 1. An Example TG

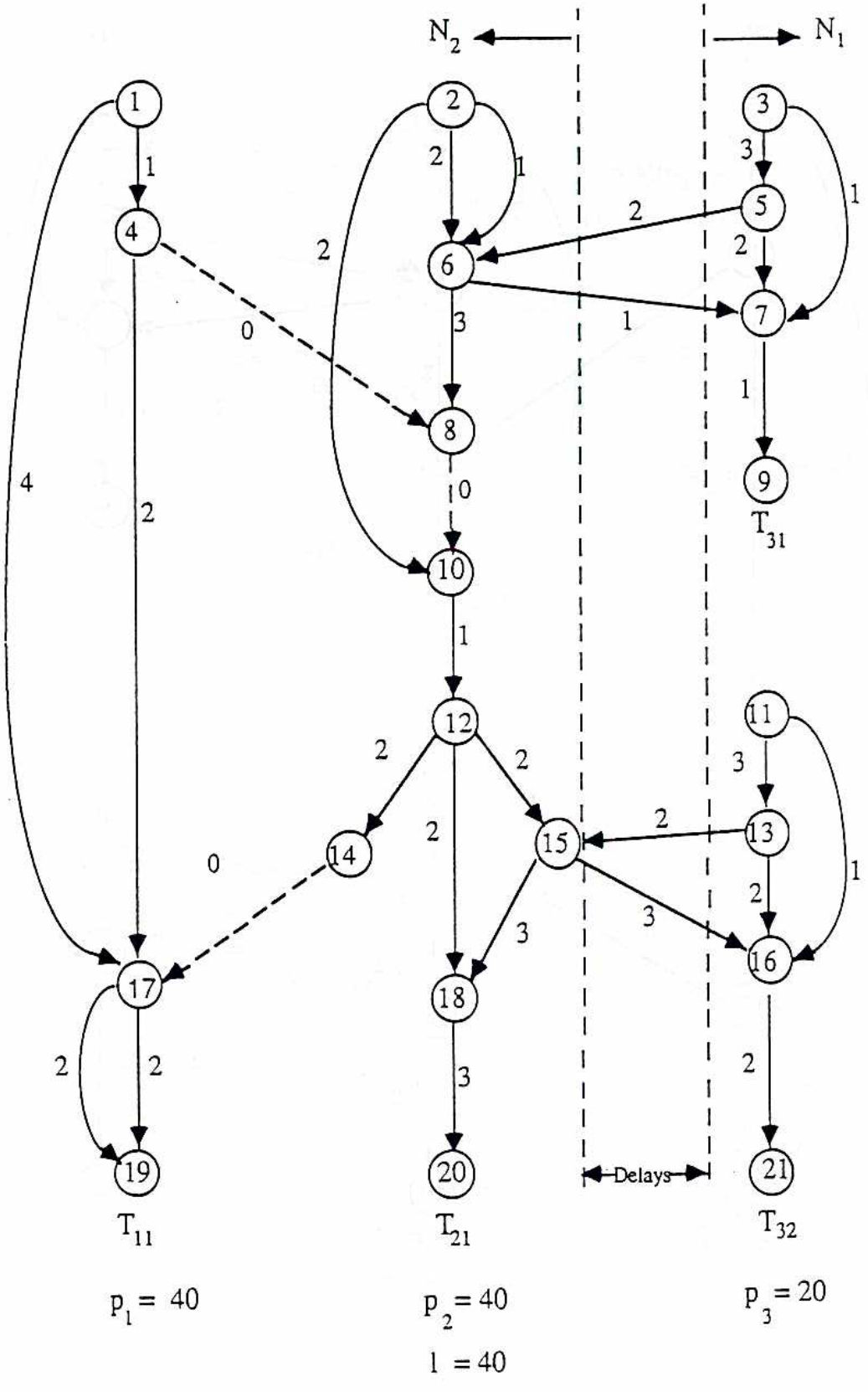
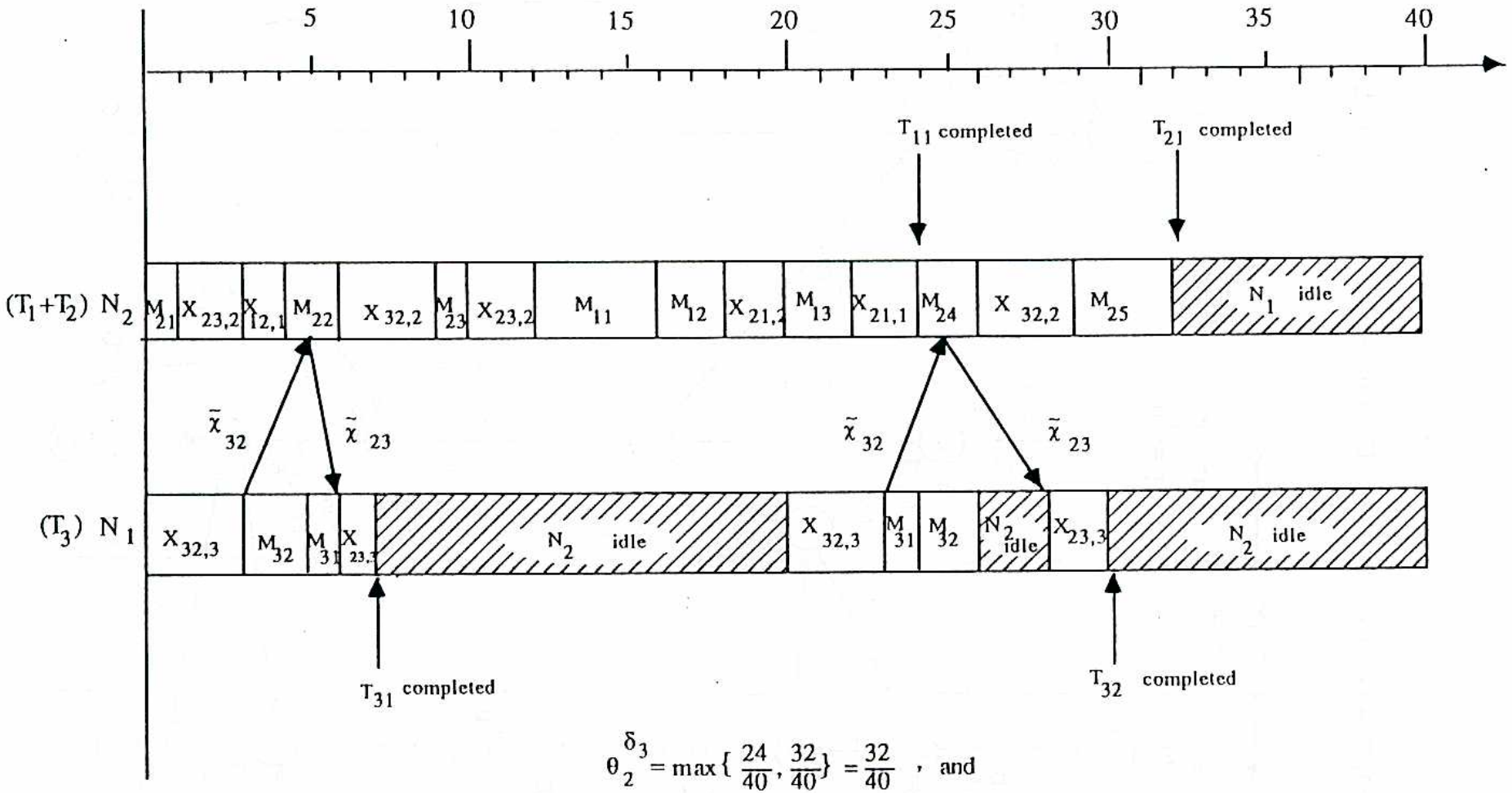


Figure 2. The TG for δ_3



$$\theta_2^{\delta_3} = \max\{\frac{24}{40}, \frac{32}{40}\} = \frac{32}{40} \quad , \text{ and}$$

$$\theta_1^{\delta_3} = \frac{10}{20}$$

$$\Theta^{\delta_3} = \max\{\frac{10}{20}, \frac{32}{40}\} = \frac{32}{40} \quad .$$

Figure 3. An Optimal Schedule for the Assignment by δ_3

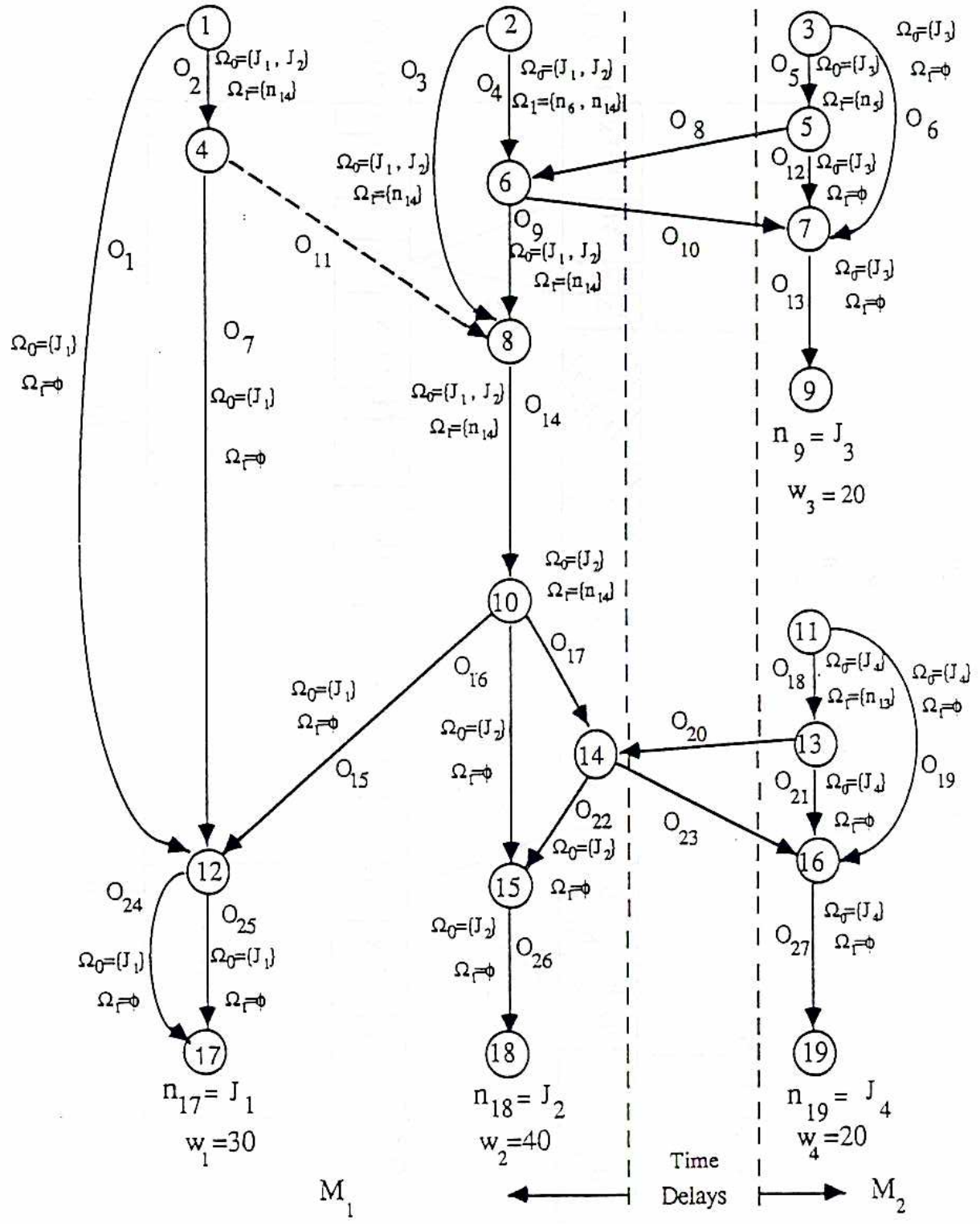


Figure 4. Ω_0 (•) and Ω_1 (•) of An Operation

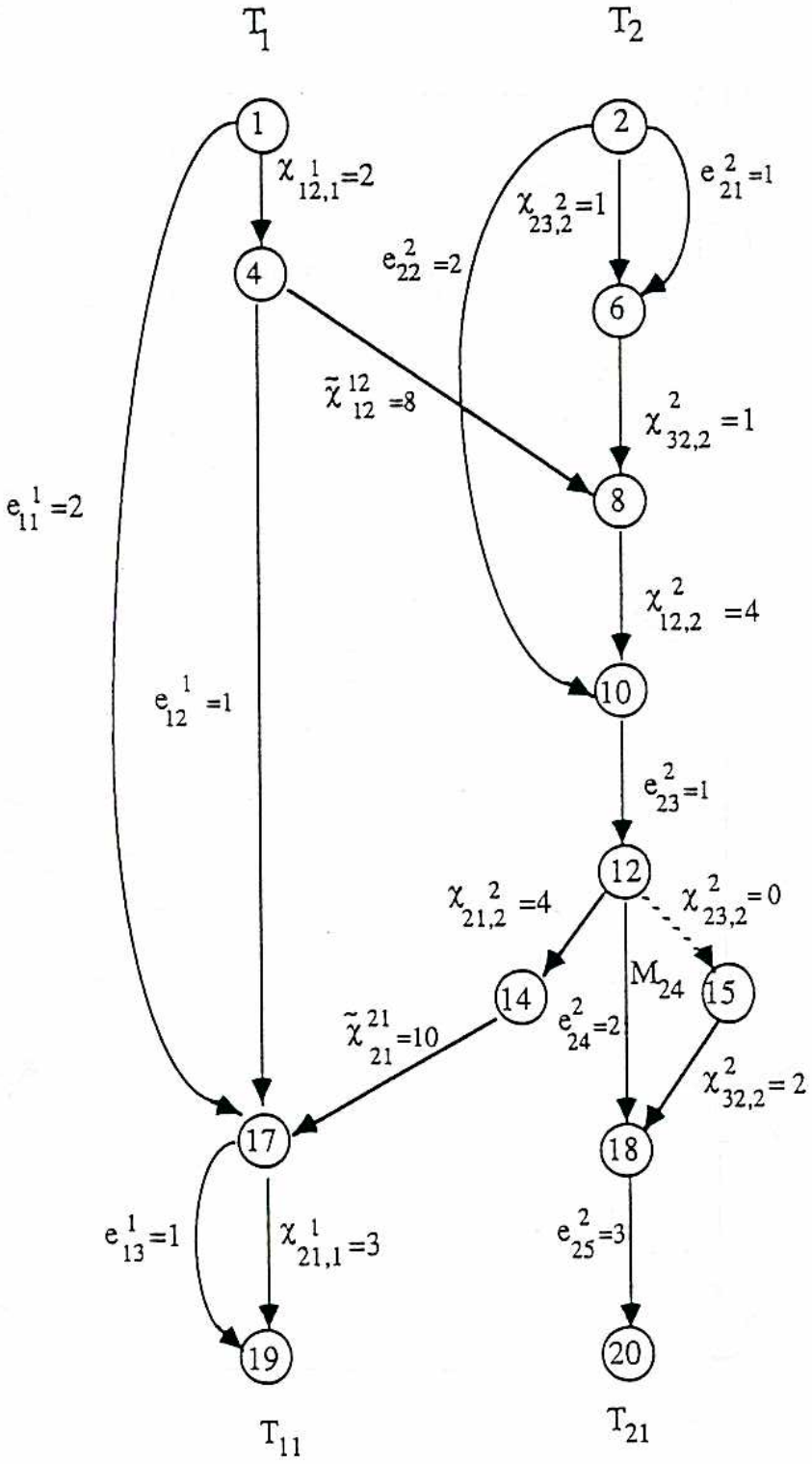


Figure 5. TG1(x) for $x=\{(T_1, N_1), (T_2, N_2)\}$

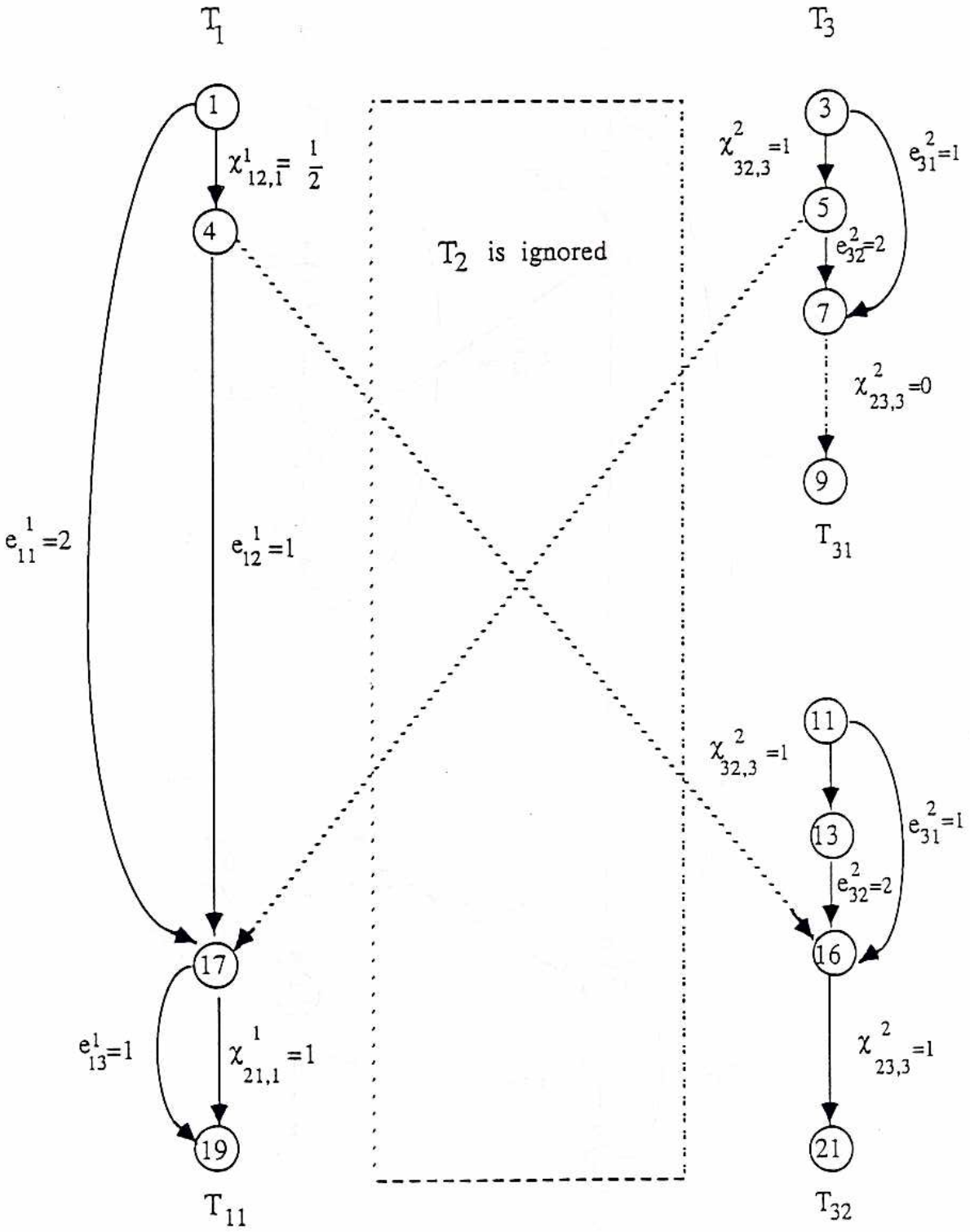


Figure 6. TG1(x) for $x = \{(T_1, N_1), (T_3, N_2)\}$

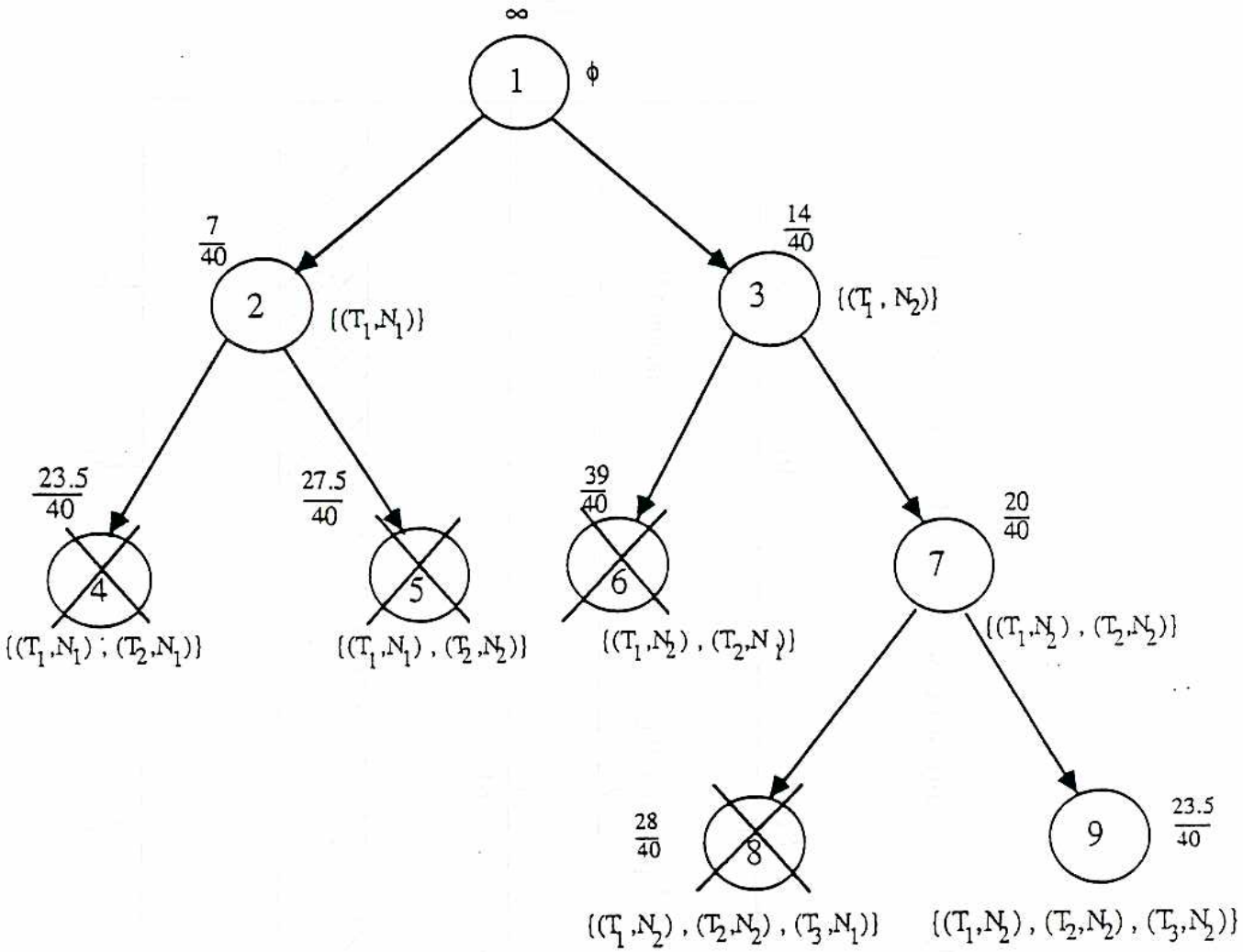


Figure 7. Search Tree Generated for the Numeric Example

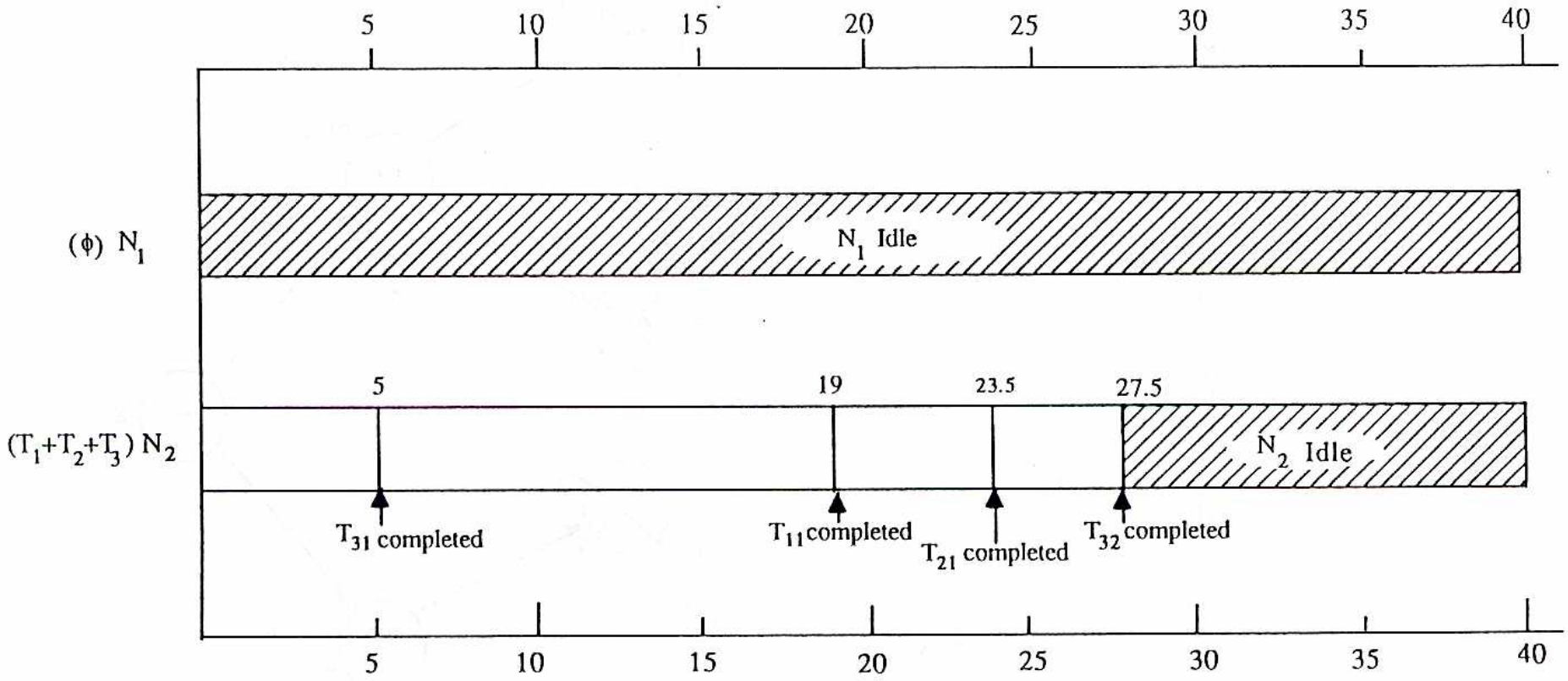


Figure 8. An Optimal Schedule for $x = \{(T_1, N_2), (T_2, N_2), (T_3, N_2)\}$

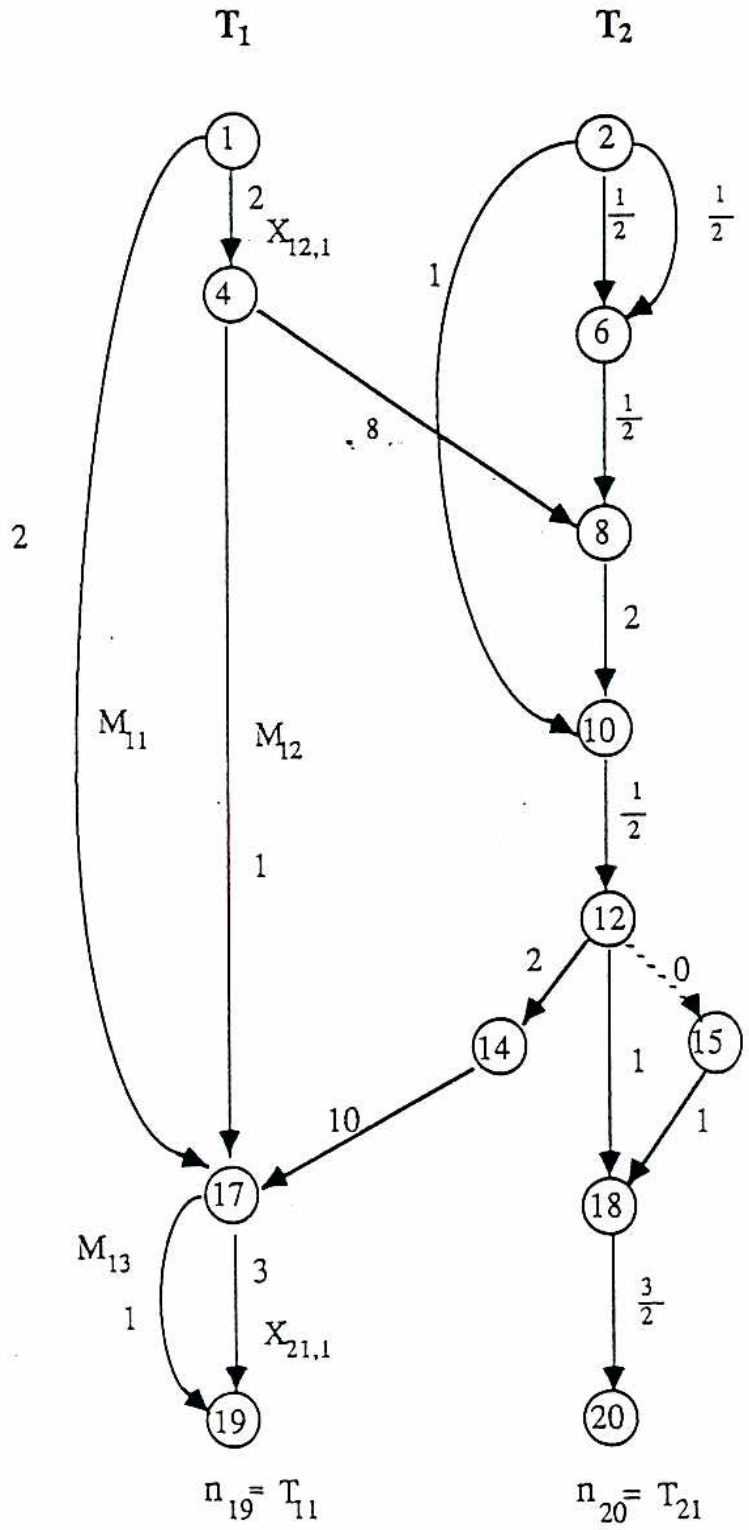


Figure 9. TG1(x) for x ={T_1 , N_1), (T_2, N_2)}