

INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.
2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.
3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.
4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.
5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms

300 North Zeeb Road
Ann Arbor, Michigan 48106

75-25,355

VILLANUEVA, Julio Estuardo, 1947-
SOME EXECUTION-TIME PROPERTIES OF OPTIMAL
SCHEDULES IN MULTIPROCESSOR SYSTEMS.

Iowa State University, Ph.D., 1975
Computer Science

Xerox University Microfilms, Ann Arbor, Michigan 48106

Some execution-time properties
of optimal schedules in
multiprocessor systems

by

Julio Estuardo Villanueva

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1975

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. DESCRIPTION OF THE MODEL AND SURVEY OF LITERATURE	6
III. STABILITY PROPERTIES OF OPTIMAL SCHEDULES	24
A. B-Schedules	24
1. Increase in the number of processors	25
2. Relaxation of the partial order	26
3. Reduction of task execution times	28
4. Combined effects	35
5. Stability of the B-schedule for variants of tree-structured graphs	36
B. A-Schedules	37
1. Increasing the number of processors	38
2. Relaxation of the partial order	42
3. Reduction of task execution times	48
4. Stability of the optimal preemptive schedule for two-processor systems	48
C. C-Schedules for Tree-Structured Graphs	50
1. Increasing the number of processors	51
2. Relaxation of the partial order	51
3. Reduction of task execution times	52
4. Combined effects	52
5. Stability of the C-schedule for variants of tree-structured graphs	53
D. Subset Assignment Schedules	54
1. Increasing the number of processors	57

2. Relaxation of the partial order	59
3. Reduction of task execution times	64
4. Combined effects	64
IV. STABILITY PROPERTIES OF OPTIMAL NONPREEMPTIVE SCHEDULES UNDER NON OPTIMAL CONDITIONS	67
V. LOWER BOUNDS ON THE RATIO CT'_{PS}/CT'_{BS}	76
A. Unequally Weighted Tasks	78
B. Equally Weighted Tasks	83
VI. BOUNDS ON THE RATIO CT'_{SSAS}/CT'_{BS}	90
VII. CONCLUSIONS	111
A. Contributions	111
B. Future Work	113
VIII. BIBLIOGRAPHY	115
IX. ACKNOWLEDGMENTS	117

LIST OF FIGURES

	<u>Page</u>
Figure 1. Example of a graph and a Gantt chart	8
Figure 2. Examples of the basic, preemptive and general scheduling disciplines	10
Figure 3. Example of B-algorithm	12
Figure 4. Example of A-algorithm	14
Figure 5. Example of C-algorithm	17
Figure 6. Illustration of the subset assignment algorithm	19
Figure 7. Example which exhibits all types of anomalies	21
Figure 8. Example of relaxation of the partial order	26
Figure 9. B-schedule for a tree with k leaves and reduced execution times	29
Figure 10. Case where $N_p \subseteq Q(V_p + 1)$	31
Figure 11. Case where $N_p \not\subseteq Q(V_p + 1)$	34
Figure 12. Effect of increasing the number of processors in an A-schedule	39
Figure 13. Partition of S' in Figure 12 using Algorithm 1	41
Figure 14. Relaxation of partial order by removing arcs (4, 2) and (3, 1) from G	43
Figure 15. Partition of S' using Algorithm 2	45
Figure 16. Anomaly in A-schedule caused by decrease in execution times	48
Figure 17. Illustration of the subset assignment schedule	56
Figure 18. Typical effect of increasing the number of processors in a subset sequence	60
Figure 19. Example of an effective move	62
Figure 20. Typical effect of relaxing the partial order in a subset sequence	65

Figure 21.	A system with equally weighted tasks that degrades with the reduction of execution time of one task	69
Figure 22.	A system with unequally weighted tasks that degrades with the reduction of execution time of one task	70
Figure 23.	A system with unequally weighted tasks that degrades with increasing k (B-schedule)	72
Figure 24.	A system with equally weighted tasks that degrades with increasing k (B-schedule)	73
Figure 25.	A system that degrades when the partial order is relaxed	74
Figure 26.	Illustration of the notation used in Chapter V	79
Figure 27.	Lower bound for a task system with unequally weighted tasks	82
Figure 28.	Example which achieves the lower bound on CT'_{PS}/CT'_{BS} when $\epsilon_i = \epsilon$	86
Figure 29.	Lower bounds for a task system with equally weighted tasks	88
Figure 30.	Forms used to construct any subset assignment	92
Figure 31.	Example which achieves the upper bound on CT'_{SSAS}/CT'_{BS} when $k > 2$ and $\epsilon_i = \epsilon$	107
Figure 32.	Example which achieves the upper bound on CT'_{SSAS}/CT'_{BS} when $k = 2$ and $\epsilon_i = \epsilon$	109

LIST OF TABLES

	Page
Table 1. Lower bounds on the ratio CT'_{PS} / CT'_{BS} for any graph G with equally weighted tasks	87
Table 2. Lower bounds on the ratio CT'_{SASS} / CT'_{BS} for any graph G with equally weighted tasks	104
Table 3. Upper bounds on the ratio CT'_{SASS} / CT'_{BS} for any graph G with equally weighted tasks on a k ($k > 2$) processor system	108
Table 4. Upper bounds on the ratio CT'_{PS} / CT'_{BS} for any graph G with equally weighted tasks on a two-processor system	110

I. INTRODUCTION

The technological constraints on the speed of computation of electronic computers have led researchers to seek alternate ways to decrease the execution time of programs. The minimization of computation times is important and even necessary in certain application areas. In some process control applications, the external world places hard constraints on the length of the interval, real-time, during which the computations must be performed. Failure to complete the computation may lead to catastrophe or at least to questionable or useless results. The length of the real-time interval, of course, varies with the application: chemical and nuclear experiments, moon landings, missile tracking, aircraft control, weather forecasting, moving a character out of a hardware buffer, etc. In process control applications, execution of programs are generally repeated. This allows for measurement of execution times and scheduling of component program parts in a multiprocessor environment.

The trend in decreasing computer costs increases the importance of minimum turnaround time for general computer users. While high costs of computing have, in the past, forced users to share equipment and hence suffer individual delay due to attempts to optimize overall utilization and throughput, future trends in some applications seem to be toward less sharing of processor and memory and a premium will be placed on turnaround time. Once again, multiprocessing in various forms, both through distributed arithmetic units and parallel instructions streams within a single program, will reduce turnaround time.

Here the automatic detection of parallelism and measurement of task times provided from an analysis of program structure would be extremely helpful. In general, multiprocessing appears to be useful in a wide spectrum of current and future application areas.

We will be concerned with the scheduling of parallel instruction streams in a real-time multiprocessor environment in an attempt to decrease the total computation time. In order to specify bounds on machine performance, we must study the stability of such schedules. By stability, we mean that the time required for execution will not increase if we relax certain constraints on the set of tasks to be executed in parallel. It is natural to avoid imposing additional stabilizing constraints if at all possible, since additional constraints also limit the available parallelism.

Preemptive scheduling is appealing because, in theory, it provides the minimum length schedule. Thus, we are also interested in the stability of preemptive schedules. Because processor switching times might not be negligible, we wish to compare the schedule length of preemptive schedules and nonpreemptive schedules, taking into consideration the switching time costs. Because of the lack of non-enumerative algorithms to produce optimal schedules, we want to study a simple low-cost scheduling technique that produces a near-optimal preemptive schedule. Our concern will be with the effect of switching costs on such schedules. The author knows of very little work done in scheduling systems where switching times are considered nonnegligible. Results in this area will help make decisions as to what levels of computing are applicable to multiprocessing and under what circumstances

preemptive scheduling is advantageous. Finally, we mention that the results reported in this work provide a general feeling for the effects of preemption costs in general systems where task systems cannot be estimated in advance.

In Chapter II, we describe the model that will be used to study scheduling problems and we survey the literature that is relevant to our interests. In Chapter III, we will study the stability of schedules produced by the four most important nonenumerative algorithms that produce an optimal schedule length when certain conditions are satisfied. In particular, we need assurance that smaller actual execution times, or increasing the number of processors, or eliminating some precedence relations does not increase the resulting schedule lengths. This chapter will report and prove results when all conditions for optimality are satisfied. Most of the results will show that we do not need to add Manacher's conditions (15) in order to preserve the stability of the schedules produced by these algorithms. However, the practical use of some algorithms requires additional constraints that also limit the available parallelism. In Chapter IV, we study the stability properties of the schedules produced by the nonpreemptive algorithms when the conditions necessary for optimality are violated. The usefulness of these algorithms under nonoptimal conditions has been established by simulation by Manacher (16), Ramamoorthy, et al. (20), Chandy and Dickson (3), and Adam, et al. (1). Most of the results reported in this chapter will show that these schedules are unstable under these conditions. In order to have stability, we have to add Manacher's conditions which are stated on pages 20 and 23.

Two of the four optimal algorithms give basic schedules or non-preemptive schedules in which it is assumed that once a processor is assigned to a task it must work continuously on this task until it has been completed. The other two algorithms reflect the results of recent studies in preemptive scheduling in which run-to-completion constraint has been relaxed. In particular, with the preemptive scheduling discipline it is possible to interrupt any processor at any time and reassign it to a different task. In this case we envision the scheduling of concurrent processors at the program or procedure level where preemption is possible. Chapters V and VI are concerned with preemption costs.

In Chapter V, we consider the effect of processor switching on the actual running time if scheduling uses the known optimal algorithms based on zero switching time. In real-life, processor switching time might not be negligible, due to other factors such as the need for a memory swap whenever a processor is switched. We will give some bounds for the ratio between the optimal preemptive schedule and the optimal nonpreemptive schedule for a graph with unequally and equally weighted tasks in which the cost of switching and preempting tasks is not zero. In Chapter VI, we will give some bounds for the ratio between a near-optimal preemptive schedule and the optimal nonpreemptive schedule for a graph with equally weighted tasks in which the cost of switching and preempting tasks is not zero. Some of the results given in Chapter VI will show that in real-life we can no longer say that the preemptive scheduling discipline is strictly more powerful than the nonpreemptive scheduling discipline, and we will have bounds on the effects of these

switching costs on a two-processor system. Finally, in Chapter VII, we summarize the results of this work and present some ideas for future study.

II. DESCRIPTION OF THE MODEL AND SURVEY OF LITERATURE

In order to study the problems mentioned in the introduction and to present some results obtained in this area by previous workers, we shall require a number of definitions.

First, let us assume we are given a set $\{P_1, P_2, \dots, P_k\}$ of identical and independent processing units, and a set $\zeta = \{T_1, T_2, \dots, T_n\}$ of tasks to be executed in the computing system. Since the processors are identical, a task can be executed on any one of the processors. A weight τ_i is associated with each task T_i and denotes the execution time of the corresponding task. We also have a partial ordering \leq on ζ . The ordering $T_i \leq T_j$ means that the execution of T_j cannot begin until the execution of T_i has been completed. T_i is called a predecessor of T_j , and T_j a successor of T_i . If there exists no task T_ℓ such that $T_i \leq T_\ell \leq T_j$ then T_i will be called an immediate predecessor of T_j , and T_j will be called an immediate successor of T_i . A task with no successor is a terminal task, and one with no predecessor is an initial task. If task T_i is neither a successor nor predecessor of T_j , then T_i and T_j are independent and therefore they may be concurrent. A task is said to be "ready" at some point in time if all of its predecessors have completed their executions. We also associate with each set of tasks a priority list \mathcal{L} .

Formally, a task system is specified by the ordered quadruple $C = (\zeta, \tau, \leq, \mathcal{L})$. Frequently, a task system $C = (\zeta, \tau, \leq, \mathcal{L})$ is represented by a precedence graph G which has the members of ζ as its vertices and there is a directed edge from T_i to T_j if, and only

if, T_j is the immediate successor of T_i . Each node in G is described by the name of the task and its associated execution time. More precisely, our computation graphs are acyclic, weighted, directed graphs, satisfying the connectivity constraint just mentioned. Let $T_{i_1}, T_{i_2}, \dots, T_{i_s}$ be the tasks in some given path in G . Then the path length is $\sum_{j=1}^s \tau_{i_j}$, and the level of a task T_i in G is the length of a longest path from T_i to a terminal task of G .

The schedule for $C = (\mathcal{C}, \tau, \prec, \mathcal{L})$ is uniquely determined by the following rules:

- 1) Whenever a processor becomes free for assignment, \mathcal{L} is scanned from left to right for the first unexecuted ready task and assigned that task.
- 2) In case of a tie among processors, they are scheduled in ascending order of their subscripts.

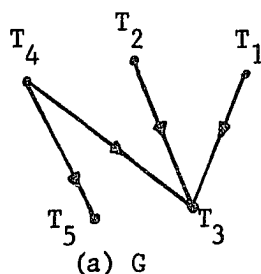
An accepted way of specifying a schedule is to use a Gantt chart (4), which consists of a time axis for each processor, with intervals marked off and labeled with the name of the task being processed. We use the symbol ϕ to represent an idle period. In Figure 1 we show a simple graph, G , and a corresponding schedule for $k = 2$ processors.

There are two interpretations on task execution times that increase the usefulness of the results when these times are not known exactly. First, the execution times may be interpreted as maximum processing times. In this case, the schedule length is the maximum time to complete the graph. Second, the execution times may be regarded as expected values of the run times considered as random variables. With this interpretation, the length of the schedule produced is an estimate of the mean

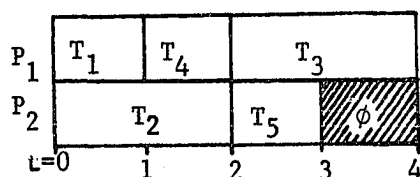
$$\mathcal{L} = (T_1, T_2, T_3, T_4, T_5)$$

$$\tau_1 = \tau_4 = \tau_5 = 1$$

$$\tau_2 = \tau_3 = 2$$



S:



(b) Gantt chart

Figure 1. Example of a graph and a Gantt chart

length of the computation over many runs. Much of the motivation for studying worst-case behavior is to derive an upper bound on the typical or expected length of the schedule.

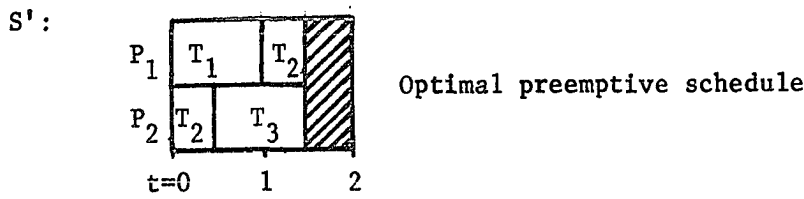
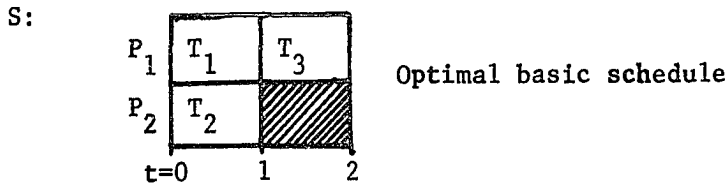
In a Basic Schedule, BS, also called a nonpreemptive schedule, it is assumed that once a processor is assigned to a task it must work continuously on this task until it is completed, while in a Preemptive Schedule, PS, the run-to-completion constraint is relaxed. Next we define a General Schedule, GS, which is not realizable in practice, but will be useful in our analysis. Suppose that the k processors in a system comprise a certain amount of computing capability rather than being discrete units. Assume further that this computing capability can be assigned to tasks in any amount up to the equivalent of one processor. If we assign α , $0 < \alpha \leq 1$ computing capability to a task,

then we assume that the computation time of the task is increased by a factor of $1/\alpha$. We allow the amount of computing capability assigned to a task to change before the task is completed, including the case where the task is not worked on at all for some interval. In Figure 2(a) we show the optimal BS and PS for a very simple graph G , and in Figure 2(b) we give a GS for G' .

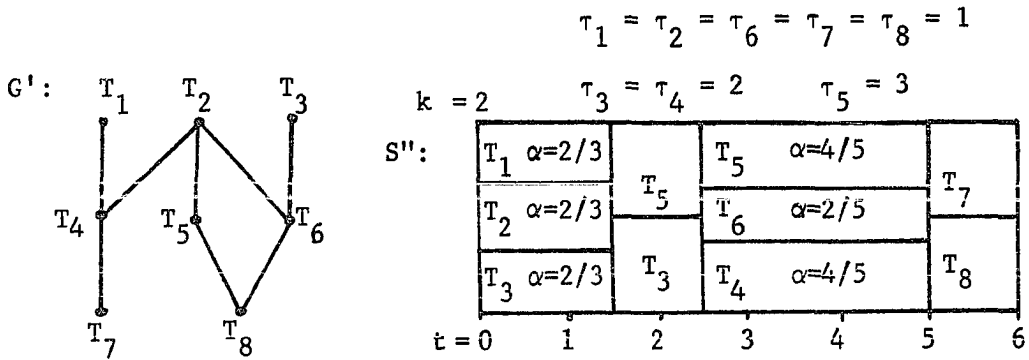
It is argued by Ullman (22) that probably no polynomial bounded algorithm exists for computing an optimal schedule for the general case. A polynomial bounded algorithm is one in which the number of steps are bounded by a polynomial in the number of nodes n . At this moment, nonenumerative techniques for finding an optimal schedule for general graphs in a k -processor system do not exist. In fact, there are only several known polynomial bounded algorithms that yield optimal schedules for certain restricted classes of systems. These algorithms and their constraints are explained in succeeding paragraphs.

For Basic Scheduling, clearly, the problem of finding an optimal BS for any given graph is effectively solvable by enumeration. However, Hu (11) gives a practical algorithm $O(n)$ for an important special case of graphs with equally weighted tasks in which the precedence relations define a tree. He assumes that an arbitrary number of processors, k , are available. A tree is defined as an acyclic directed graph in which each node, except for the root, has exactly one immediate successor.

The algorithm, also known as the B-Algorithm (5), simply follows the rule: Whenever a processor becomes free, assign it to a task, if any, all of whose predecessors have completed execution, and which is



(a) Optimal BS and PS for G



(b) General scheduling

Figure 2. Examples of the basic, preemptive and general scheduling disciplines

at the highest level of those tasks not yet assigned. If there is a tie among several tasks, then a task is selected arbitrarily. Hu showed that the given B-algorithm is optimal, and the schedule-length is given by

$$t(S) = \max_{0 \leq j \leq L} \left\{ j + \left\lceil \frac{|Q(j+1)|}{k} \right\rceil \right\}$$

where $|Q(j+1)|$ represents the number of tasks at level $j+1$ or greater and L is the length of the longest path in the graph. This bound comes from the fact that for any j , it is not possible to compute all of the tasks whose level number is greater than j in less than $\left(\frac{\text{number of such tasks}}{k}\right)$ units of time, and at least j units of time are needed to complete the remainder of the graph. When the B-algorithm is applied to a tree graph, G , with unequally weighted tasks, Kaufman (13) found that the schedule length obtained is at most p units longer than the optimal preemptive schedule for G , where: $p = \max\{\tau_i\} \forall_i \text{ in } G$. In Figure 3, the operation of the B-algorithm for $k = 3$ is illustrated. Note that in this case the tie mentioned in the description of this algorithm occurs during the first two steps. Note also that the B-schedule is also optimal for forests of trees and for p -restricted precedence graphs (2) with unit tasks. A p tree-restricted precedence graph G consists of p tree-structured subgraphs G_1, G_2, \dots, G_p such that each terminal task of G_{i-1} is a predecessor of each initial task of G_i for $i = 2, \dots, p$.

For the case of $k = 2$ processors, Bauer (2) has discovered an algorithm that will give an optimal schedule for tree-structured graphs

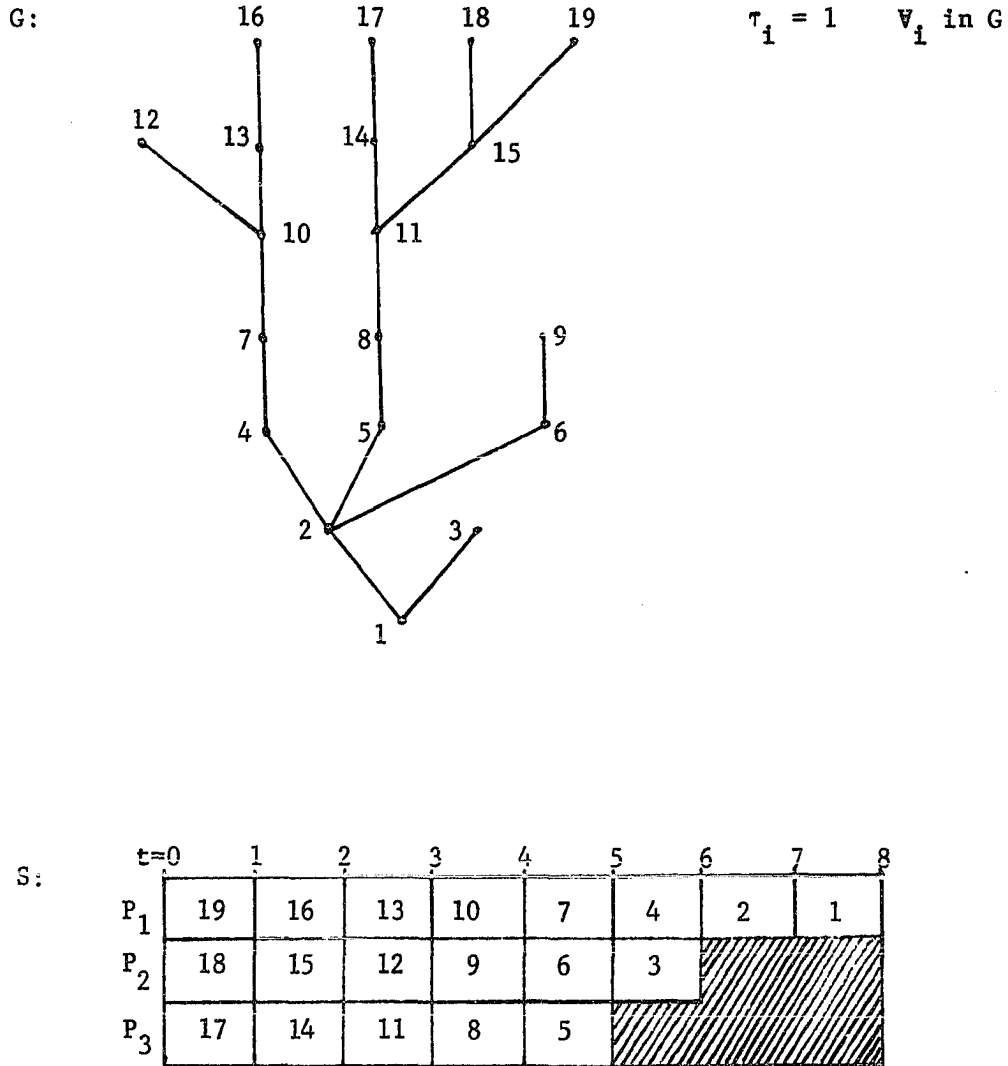


Figure 3. Example of B-algorithm

and p tree-restricted graphs, in which the nodes have weights of one or two units.

Coffman and Graham (6) present an $O(n^2)$ algorithm that finds an optimal basic schedule for general graphs on two processors when the

tasks have unit execution time. The scheduling algorithm is based on the following labeling algorithm: Let $N = (n_1, \dots, n_t)$ and $N' = (n'_1, \dots, n'_{t'})$ denote two decreasing sequences of positive integers. Define $N < N'$ if either (a) for some i , $1 \leq i \leq t$, we have $n_j = n'_j$ for $1 \leq j \leq i - 1$ and $n_i < n'_i$ or (b) $t < t'$ and $n_j = n'_j$ for $1 \leq j \leq t$. Let n denote the number of tasks in G . The labeling algorithm assigns to each task T an integer $\alpha(T) \in \{1, 2, \dots, n\}$. The mapping α is defined recursively as follows: Let $S(T)$ denote the set of immediate successors of T .

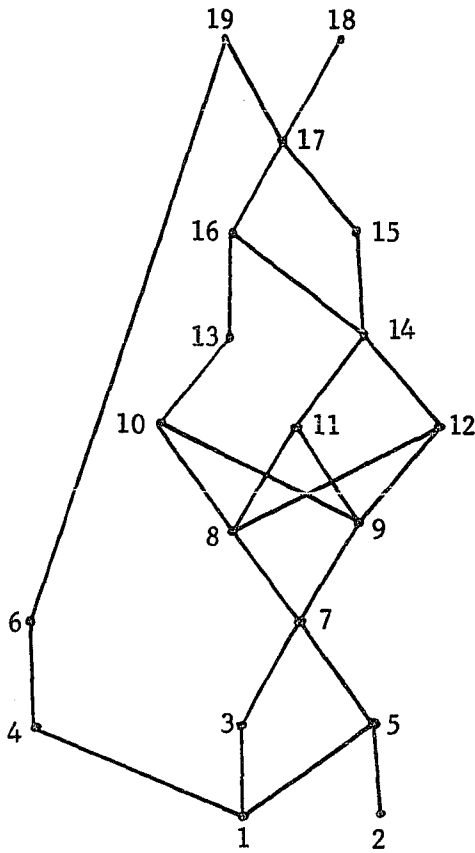
S1 - Any arbitrary task T_0 with $S(T_0) = \emptyset$ is chosen and $\alpha(T_0)$ is defined to be 1.

S2 - Suppose for some $k \leq n$ that the integers $1, 2, \dots, k - 1$ have been assigned. For each task T whose successors have all been labeled, let $N(T)$ denote the decreasing sequence of integers formed from the set $\{\alpha(T') \mid T' \in S(T)\}$. At least one of these tasks T^* must satisfy $N(T^*) \leq N(T)$ for all such tasks T , and let $\alpha(T^*) = k$.

S3 - Repeat S2 until all tasks of G have been assigned some integer.

The scheduling algorithm, also known as the A-Algorithm (5), is defined as follows: Whenever a processor becomes free, assign that task all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned. An example illustrating the A-algorithm is provided in Figure 4. This algorithm is not optimal when the number of processors is greater than two or when the tasks are of unequal execution time.

G:



$$\mathcal{L} = (T_{19}, T_{18}, \dots, T_1)$$

$$\tau_i = 1 \quad \forall_i \text{ in } G$$

S:

	t=0	1	2	3	4	5	6	7	8	9	10
P ₁	19	17	16	14	12	10	9	7	5	2	
P ₂	18	6	15	13	11	4	8		3	1	

Figure 4. Example of A-algorithm

Algorithms for finding the optimal-schedule length in the PS discipline are as rare as results for the BS discipline. An important fundamental result in preemptive scheduling is presented by McNaughton (16): For a set of independent tasks with weights $\{\tau_1, \tau_2, \dots, \tau_n\}$ and k available homogeneous processors, the optimal PS has length

$$\max \left\{ \max_{1 \leq i \leq n} \{\tau_i\}, \frac{\sum_{i=1}^n \tau_i}{k} \right\}.$$

It is clear that this computation time cannot be improved upon, since the schedule must be at least as long as the largest task and cannot be more efficient than to keep all the processors continuously busy.

Muntz (17) gives an algorithm for constructing an optimal GS for any number of processors when the computation graph is a rooted tree or forests of trees, and the node weights are mutually commensurable, i.e., there exists a real number, w , such that all node weights are integer multiples of w .

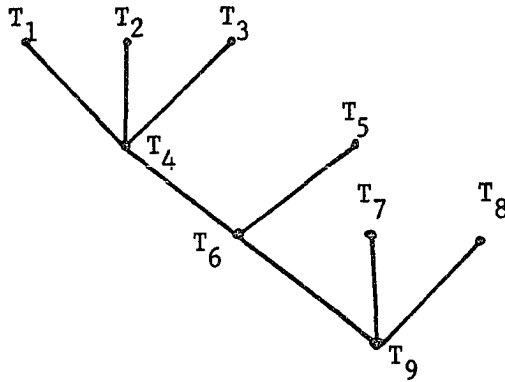
The algorithm, also known as the C-Algorithm (5), is the following: Assign one processor each to the tasks at the highest level. If there is a tie among b tasks (because they are at the same level) for the last a ($a < b$) processors, then assign a/b of a processor to each of these b tasks. Whenever either of the two events described below occurs, reassign the processors to the uncomputed portion of the graph G according to this rule. These are:

Event 1: A task is completed.

Event 2: We reach a point where, if we continue the present assignment, we would be executing some tasks at a lower level at a faster rate than other tasks at a higher level. Note that the C-algorithm, as it is defined, produces a general schedule, but there is a direct conversion from the optimal general schedule to the optimal preemptive schedule. Muntz (17) showed that the PS and GS are equally effective, because for all graphs the optimal schedules according to both disciplines have the same length. Therefore, the C-algorithm gives an optimal preemptive schedule for a tree-structured graph G and we write $CT_{GS}(G, k) = CT_{PS}(G, k) = CT_C(G, k)$ where $CT_{\alpha}(G, k)$ represents the minimum computation time of G with k processors using the α -scheduling discipline. An example illustrating the C-algorithm is provided in Figure 5. Define G_w to be the precedence graph derived from G by replacing each task T_i by n_i mutually commensurable task, $T_{i,1}, \dots, T_{i,n_i}$ each with execution time w .

Muntz and Coffman (18) give an algorithm for finding the optimal preemptive schedule for general graphs in a two-processor system based on the notion of a subset sequence. A subset sequence for a graph G , with equally weighted nodes, is a sequence of nonempty disjoint subsets of nodes S_1, S_2, \dots, S_l such that 1) if n is a node of G , then $n \in S_i$ for some i , and 2) if n, m are nodes of G , with $n \in S_i, m \in S_j$ and $n < m$, then $i < j$. A PS for G constructed using the schedule for S_1 followed immediately by the schedule for S_2 , etc., is called a subset assignment. They showed that for a 2-processor system, an optimal subset assignment for any graph G , with equally weighted nodes, is an optimal PS for G .

G:



$$\tau_1 = \tau_2 = \tau_3 = 7 \frac{1}{2}$$

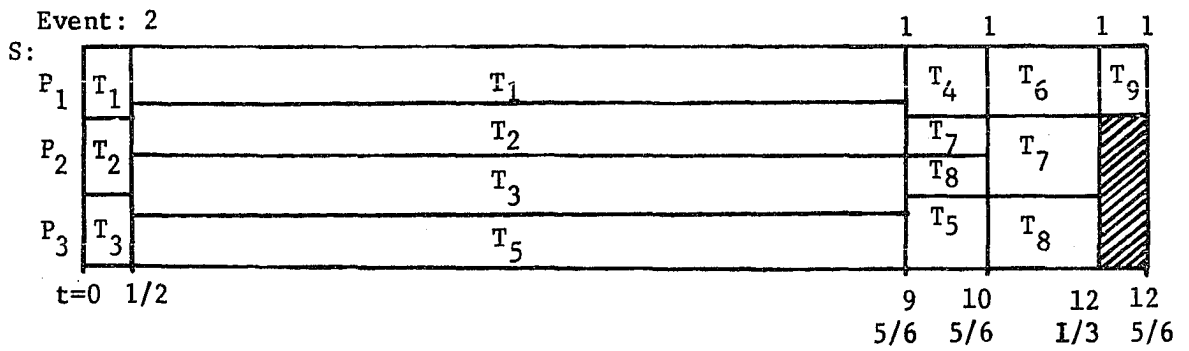
$$\tau_4 = 1$$

$$\tau_5 = 8$$

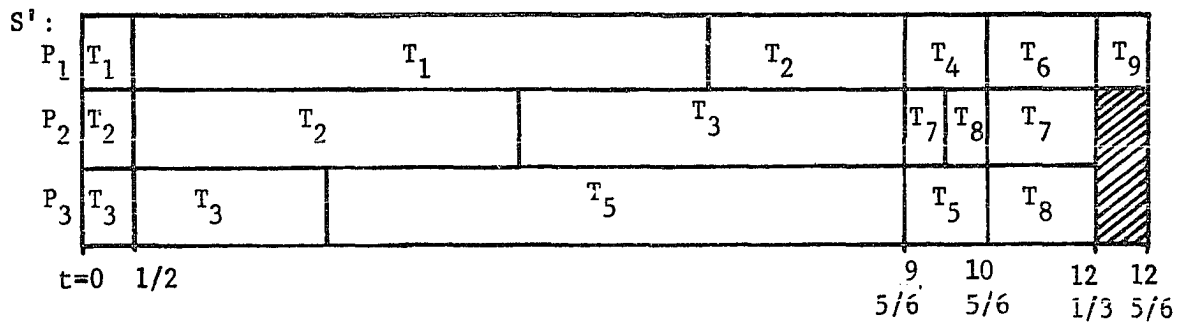
$$\tau_6 = 1 \frac{1}{2}$$

$$\tau_7 = \tau_8 = 2$$

$$\tau_9 = \frac{1}{2}$$



(a) Optimal GS obtained using C-algorithm



(b) Optimal PS obtained from the optimal GS

Figure 5. Example of C-algorithm

The algorithm for finding the optimal subset assignment for two processors is as follows: Let G be a graph with mutually commensurable

node weights and let G_w be as defined above. Let L be the length of a longest path in G .

S1 - Set index $j = 1$.

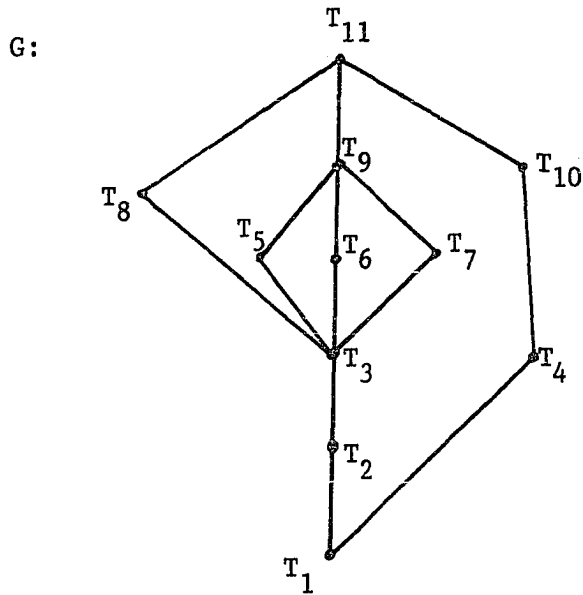
S2 - Let Ω_j be the set of all nodes which have not yet been assigned to subsets and are at level $L - j + 1$. Assign all nodes in Ω_j to S_j . If the number of nodes in Ω_j is one then go to S4.

S3 - If $j = L$ then stop, otherwise, set $j = j + 1$ and go to S2.

S4 - Let $\alpha = \{q_i\}$ be the set of all nodes which have not been assigned to subsets but all of whose predecessors are contained in $S_1 \cup S_2 \cup \dots \cup S_{j-1}$. If $\alpha = \emptyset$ then go to S3. If $\alpha \neq \emptyset$ then assign q_u to S_j , where q_u is such that $\text{level of } q_u = \max_{q_i \in \alpha} \{\text{level of } q_i\}$, and go to S3.

An informal statement of this algorithm is: Nodes are assigned to subsets level by level, with higher level nodes assigned to subsets first. The only exception to this rule is when it would result in a subset containing only one node while there is at least one other node, at a lower level, which can be assigned to this subset without violating any precedence relations. In this case a second node is selected which is at the highest possible level. As an example we show in Figure 6, a graph G and a subset sequence constructed by the algorithm. In spite of its appealing simplicity, it is unfortunate that this algorithm is not optimal for more than two processors.

In dealing with preemptive schedules, the question arises as to how many preemptions are necessary to form an optimal preemptive schedule on a general system (assuming preemption costs are zero). In general, we need not consider preemptions at intervals more frequent than one every $w/k!$ units for any k processor system. More precisely, it is



Optimal subset sequence for G: $\{T_{11}\}$, $\{T_9, T_8\}$, $\{T_5, T_6, T_7\}$, $\{T_3, T_{10}\}$,
 $\{T_2, T_4\}$, $\{T_1\}$

Figure 6. Illustration of the subset assignment algorithm

conjectured by Muntz (17) that any preemptive schedule, for a graph G is at least as long as the optimal BS for $G_{w/k!}$, i.e., $CT_{PS}(G, k) = CT_{BS}(G_{w/k!}, k)$. Incidentally, this is equivalent to the conjecture that an optimal subset assignment for $G_{w/(k-1)!}$ is an optimal PS for G . These conjectures remain unproved.

Later, Coffman and Graham (6) showed that in the limit, Algorithm A applied to $G_{w/r}$ as $r \rightarrow \infty$, converges to the C-schedule because at each point in time both give highest priority to those remaining tasks at the highest level in the graph requiring processing. For that reason, we know that the C-algorithm is optimal for any number of processors

when the computational graph is a tree-structured graph, and for general graphs on a two-processor system.

A second important run-time consideration deals with the stability of schedules. Graham (8, 9, 10) considers four types of changes, each of which may produce anomalous increases in schedule length:

- 1) changing the order of tasks in the priority list;
- 2) removing some of the precedence relations;
- 3) increasing the number of processors; and
- 4) reducing the execution time of some tasks.

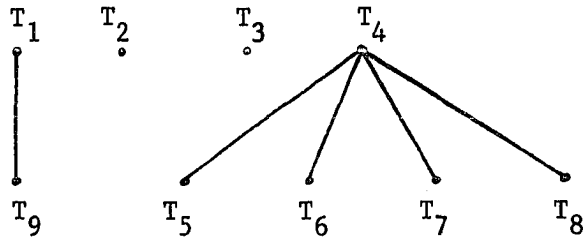
Cases 1), 3) and 4) were first discussed by Richards (21). A simple example which exhibits all four conditions has been presented by Graham (10) as shown in Figures 7(a) through 7(e). Graham proves a theorem which states that given two schedules, S and S' , where S' is related to S by the application of any combination of the four changes given above, the schedule lengths $t(S)$ and $t(S')$ are related to one another by the bound

$$\frac{t(S)}{t(S')} \leq 1 + \frac{k - 1}{k'}$$

where k, k' are the number of processors in S and S' , respectively. Graham's theorem also states that this bound is the best possible in the sense that it cannot be replaced by a smaller number. It reduces to $2 - \frac{1}{k}$ when $k = k'$. Since Graham's bound gives $\frac{t(S)}{t(S')} \leq 2$, we have an absolute bound on "anomalous" behavior.

Rules for stabilizing schedules to prevent such anomalies have been established by Manacher (15). These rules introduce more precedence relations and change the priority list to achieve stability. In

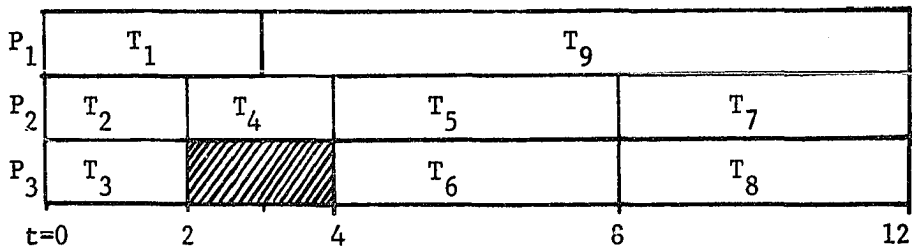
G:



$$\mathcal{L} = (T_1, T_2, \dots, T_9)$$

$$\tau = (3, 2, 2, 2, 4, 4, 4, 4, 9)$$

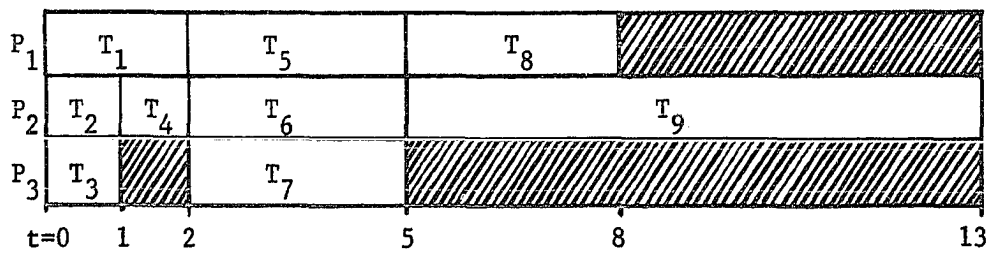
S:



(a) Expected schedule

Run-time schedule S_1 when task execution of each task is decreased by one, i.e.,

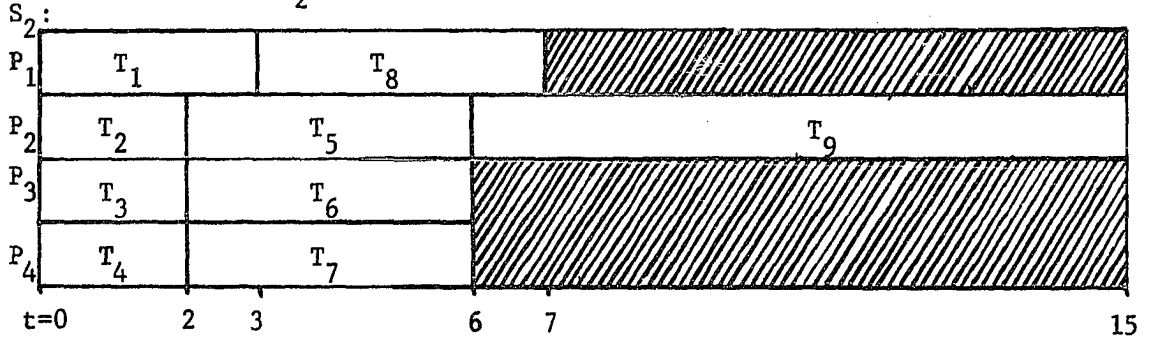
$$\tau' = (2, 1, 1, 1, 3, 3, 3, 3, 8)$$

 S_1 :

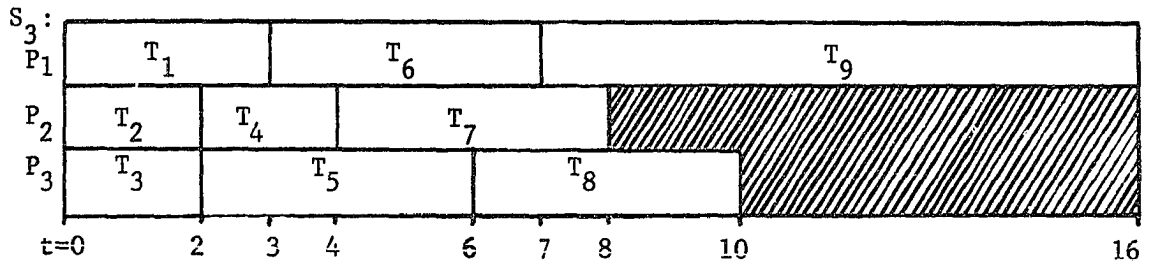
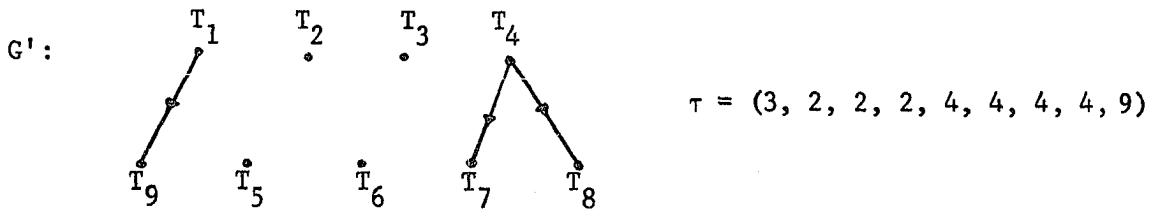
(b) Decreasing execution times

Figure 7. Example which exhibits all types of anomalies

Run-time schedule S_2 , when the number of processors is increased to 4:



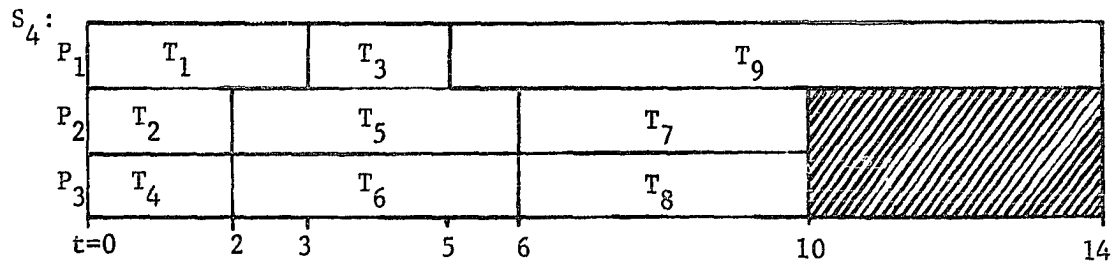
(c) Increasing the number of processors



(d) Relaxing some precedence constraints

Run-time schedule S_4 , when the priority list is changed:

$$\mathcal{L}' = (T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$$



(e) Changing the priority

Figure 7. Continued

later chapters, where we demonstrate instability, Manacher's constraints would have to be added to guarantee stability.

Recently, Kaufman (12) showed that systems with equally weighted tasks are not anomaly free.

A natural comparison is between the optimal preemptive and optimal nonpreemptive, basic, schedules. Liu (14) shows that for a given set of tasks specified by execution times and a partial order,

$$CT_{PS} \geq \frac{k+1}{2k} CT_{BS}$$

for a k -processor system. Moreover, this bound is the best possible.

In the results stated above, the cost of preemption is considered to be zero or too negligible to affect the schedule length. In real-systems, preemptions may be significant. In these cases, the usefulness of the preemptive models is lost by neglecting preemption costs.

III. STABILITY PROPERTIES OF OPTIMAL SCHEDULES

In this chapter, we study the stability properties of optimal nonpreemptive schedules and preemptive schedules. These schedules are found to be free, in almost all the cases, of three anomalies commonly found in multiprocessor systems. Stability allows us to use these schedules to compute an upper bound on the schedule length under the most constraining conditions. At run-time, these conditions can then be relaxed without danger of an increase in the schedule length.

This chapter is divided in four sections. In Section A, we study the stability properties of the B-schedule. Section B studies the stability of the A-schedules. The last two sections are dedicated to preemptive schedules. In Section C we study the stability of the C-schedules for tree-structured graphs, and finally, in Section D we study the stability properties of the Subset Assignment Schedules.

A. B-Schedules

In this section, we study the stability properties of the schedules produced by the B-algorithm. This algorithm produces optimal nonpreemptive schedules for systems in which tasks have equal execution times and the partial order is defined by a tree-structured precedence graph. We begin with the optimal B-schedule S for the case where $\tau_i = 1$ for all i and investigate the stability of this schedule by relaxing the constraints on τ_i and the precedence relations, \prec , and increasing the number of processors to form a schedule S' . Since a change in the priority list can obviously result in increased schedule length, we

hold \mathcal{L} fixed as defined for the optimal B-schedule S . Thus, we show that the optimal B-schedule S is inherently free from three common anomalies and additional constraints are not necessary (15).

1. Increase in the number of processors

Let G denote a tree-structured precedence graph which defines the partial order \leq . Let $Q(j)$ denote the set of tasks at levels greater than or equal to j and L be the maximum length path in G , and $|Q(j)|$ denote the cardinality of $Q(j)$. Hu (11) showed that the length of the optimal schedule on k processors is

$$t(S) = \max_{0 \leq j \leq L} \left\{ j + \left\lceil \frac{|Q(j+1)|}{k} \right\rceil \right\}$$

where $\lceil X \rceil$ denotes the ceiling of X . The following theorem shows the B-schedule is free of the anomaly caused by increasing the number of processors.

Theorem 1. Given a tree-structured precedence graph G with equally weighted tasks, the B-schedule is stable if the number of processors is increased.

Proof: Let $t(S)$ and $t(S')$ denote the B-schedule length for G on k and k' processors, respectively, where $k' > k$. Define

$$t_j = j + \left\lceil \frac{|Q(j+1)|}{k} \right\rceil \text{ and } t'_j = j + \left\lceil \frac{|Q(j+1)|}{k'} \right\rceil.$$

Then $t(S) = \max_{0 \leq j \leq L} (t_j)$ and $t(S') = \max_{0 \leq j \leq L} (t'_j)$. Since $k' > k$ implies $t_j \geq t'_j$

for all j , it immediately follows that $t(S) \geq t(S')$.

2. Relaxation of the partial order

In order to insure that $\leq' \subseteq \leq$ and the new precedence graph G' (which defines \leq') is a tree or forest, we define an "allowable relaxation" as the removal of an arc (T_i, T_j) from G , i.e., T_j is the "immediate" successor of T_i . Furthermore, this implies

(i) the addition of an arc (T_i, T_k) if T_k is the immediate successor of T_j in G .

(ii) the addition of no arcs if T_j is the root task in G .

In the first case, the resulting graph G' is a tree while in the second case G'' is a forest. Figure 8 shows an example of both cases. G' is formed by removing (T_3, T_5) from G and G'' is the result of removing (T_5, T_6) from G .

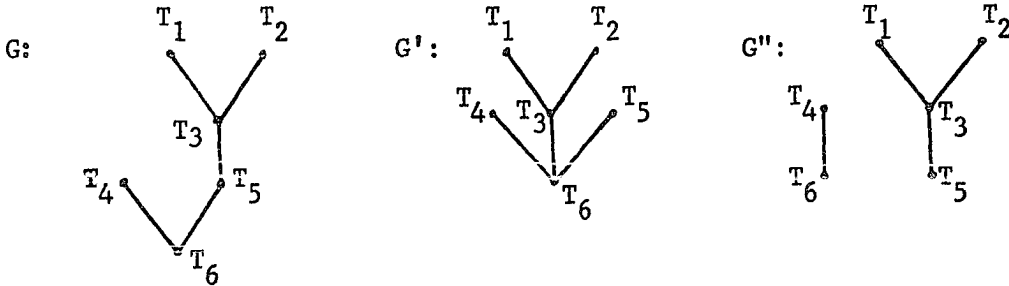


Figure 8. Examples of relaxation of the partial order

The following theorem shows that removal of a single arc from G will not cause an anomaly in the B-schedule.

Theorem 2. Given a tree-structured precedence graph G with equally weighted tasks, the B-schedule is stable under an allowable relaxation of G .

Proof: Suppose we remove arc (T_s, T_r) from G where T_s is at level $\ell + 1$ and T_r is at level ℓ . In the subtree with root T_s , the

level of each task is reduced by one and T_r and T_s have the same immediate successor in G' . Let n_i denote the number of nodes in the subtree with root T_s that are at level i in G . Clearly, $n_i = 0$ for $1 \leq i \leq \ell$, $n_{\ell+1} = 1$, $n_i \geq 0$ for $\ell + 1 < i \leq L$. Let $t(S)$ and $t(S')$ denote the B-schedule lengths for G and G' , respectively. Since G is a tree, we have as before

$$t(S) = \max_{0 \leq j \leq L} (t_j) \quad \text{where} \quad t_j = j + \left\lceil \frac{|Q(j+1)|}{k} \right\rceil$$

If G' is a tree, then we have

$$t(S') = \max_{0 \leq j \leq L} (t'_j)$$

where

$$t'_j = j + \left\lceil \frac{|Q(j+1)| - n_{j+1}}{k} \right\rceil \leq j + \left\lceil \frac{|Q(j+1)|}{k} \right\rceil = t_j$$

The result $t(S') \leq t(S)$ is immediate. If G' is a forest, we may add to G' a dummy root task T_0 with execution time zero and define it to be at level zero in G' . Since the level of all other tasks in G' remains unchanged, the above analysis may be repeated and, once again, $t(S') \leq t(S)$.

Through successive applications of Theorem 2, we have:

Corollary. Given a tree-structured precedence graph G with equally weighted tasks, the B-schedule is stable if any number of arcs are removed from G .

3. Reduction of task execution times

In this section, we assume that the actual execution time of the tasks are given by $\tau'_i \leq 1$ for $i = 1, \dots, n$. We also assume that the τ'_i form a mutually commensurable set of values with greatest common divisor δ . Denote the highest level of a task in G by L . For this case, we consider the level of any task T_i to be the level it had in the graph G where $\tau_i = 1$, for all T_i in G . Define;

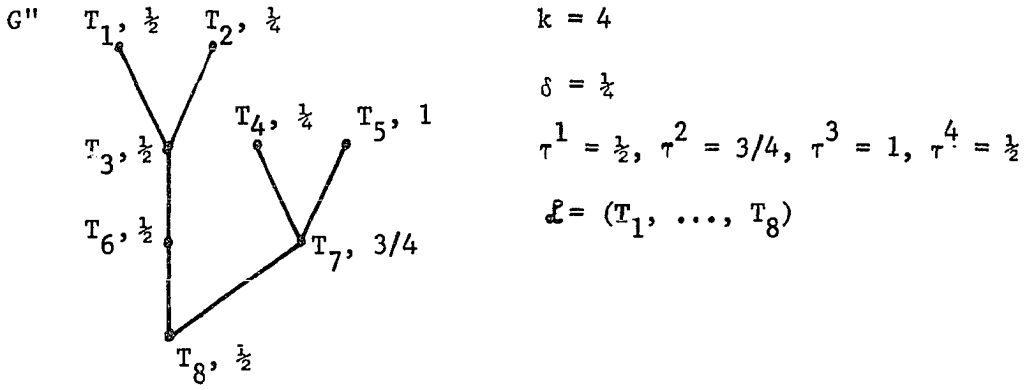
$$\tau^{\ell} = \max \{ \tau'_i : \tau'_i \text{ is the actual execution time of } T_i \text{ and } T_i \text{ is at level } \ell \text{ in } G \}$$

The analysis which follows is similar to that presented in (5). The reader may find Figure 9 helpful in following the next lemma.

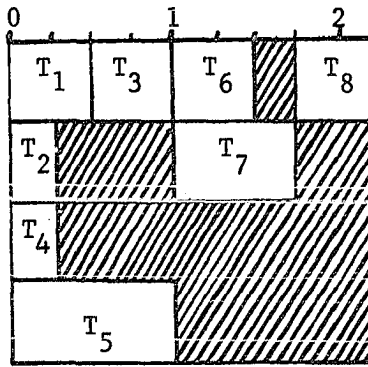
Lemma 1. Let G be a tree with no more than k leaves (initially available tasks). Assume unequal execution times τ'_i as described above. Then the B-schedule S' for G on k processors has length

$$t(S') \leq \sum_{i=1}^L \tau^i$$

Proof: Define R_i , $0 \leq i \leq \frac{t(S')}{\delta} - 1$, to be the set of tasks executed in the time interval $(i\delta, (i+1)\delta)$. Since G is a tree, the sequence $|R_0|, |R_1|, \dots, \left| \frac{R_t(S')}{\delta} - 1 \right|$ is monotonically decreasing where $|R_i|$ denotes the number of tasks in R_i . This observation follows from the fact that the execution of an available task can cause at most one task to become available for execution in the next time interval. Since $|R_0| \leq k$, we have $|R_i| \leq k$, $0 \leq i < \frac{t(S')}{\delta} - 1$. Therefore, $R_{i+1} \subseteq R_i$, or at least one task in R_{i+1} is at a lower level than its predecessor in R_i , $0 \leq i < \frac{t(S')}{\delta} - 1$. At time τ^L , all tasks at level L in



S' :



$$\tau(S') = 2\frac{1}{4}$$

$$R_0 = \{T_1, T_2, T_4, T_5\} \quad R_4 = R_5 = \{T_6, T_7\}$$

$$R_1 = \{T_1, T_5\} \quad R_6 = \{T_7\}$$

$$R_2 = R_3 = \{T_3, T_5\} \quad R_7 = R_8 = \{T_8\}$$

Figure 9. B-schedule for a tree with k leaves and reduced execution times

G are completed and in general, at time $\tau^L + \tau^{L-1} + \dots + \tau^i$, all tasks whose level is not less than i in G are completed. Hence

$$t(S') \leq \sum_{i=1}^L \tau^i.$$

Because $\tau^L \leq 1$, this Lemma also says that $t(S') \leq L$.

Lemma 2. Let G be a tree in which the tasks have unequal execution times $\tau_i' \leq 1$. If S' is a B-schedule for G on k processors, then

$$t(S') \leq \max_{0 \leq j \leq L} \left\{ j + \left\lceil \frac{|Q(j+1)|}{k} \right\rceil \right\}$$

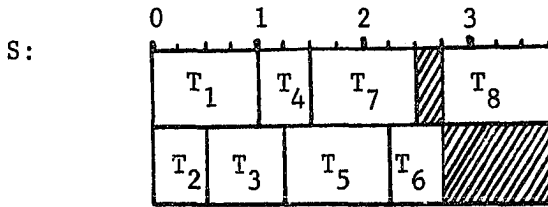
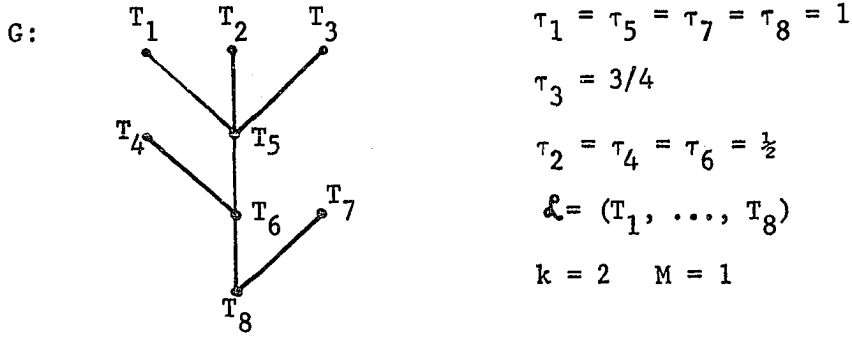
Proof: Since the result is trivially true for $L = 1$, we may assume $L > 1$. Let τ^i be as defined above and let $M = \max_{1 \leq i \leq L} (\tau^i)$. Define the following sequence of points in the schedule.

$$t_0 = 0$$

$$t_i = t_{i-1} + \min \{ M, \text{time required to complete execution of } k \text{ more tasks} \}$$

Let N_i be the set of tasks which complete execution at or before t_i and $|N_i|$ be the number of tasks in N_i . Define p to be the least value such that a time t_p some processor is idle or becomes idle. If $p = 0$, then Lemma 1 applies and we are done. Therefore, we assume $p \geq 1$. Note that no processor can be idle in the interval $(0, t_{p-1})$, and N_i is always an integral multiple of k for all i such that $1 \leq i \leq p$.

Define V_j , $0 \leq j < t(S')$ as the maximum level occupied by tasks in the tree G which have not completed execution at time t_{j+1} . In other words, V_j is the smallest level such that $Q(V_j + 1) \leq N_{j+1}$. Figure 10



$t_1 = 1$	$N_1 = \{T_1, T_2\}$	$V_0 = 4$	$p = 3$
$t_2 = 1\frac{1}{2}$	$N_2 = \{T_1, \dots, T_4\}$	$V_1 = 3$	$Q(V_1 + 1) = \{T_1, T_2, T_3\}$
$t_3 = 2\frac{1}{2}$	$N_3 = \{T_1, \dots, T_5, T_7\}$	$V_2 = 2$	$Q(V_2 + 1) = \{T_1, \dots, T_5\}$
$t_4 = 3\frac{1}{2}$	$N_4 = \{T_1, \dots, T_7\}$	$V_3 = 1$	$Q(V_3 + 1) = \{T_1, \dots, T_7\}$

Figure 10. Case where $N_p \subseteq Q(V_p + 1)$

illustrates these definitions. From the definition of p and Lemma 1, the tasks remaining to be executed after t_{p+1} can be completed in no more than $\sum_{i=1}^{V_p} \tau^i \leq V_p$ time units. Therefore

$$t(S') \leq V_p + t_{p+1} \leq V_p + M(p+1) \leq V_p + (p+1)$$

At this point we have: $|N_p| = kp$, $|N_{p+1}| > kp$ and $kp < |N_{p+1}| = |N_p| + |N_{p+1} - N_p| \leq kp + k$. So

$$p < \frac{|N_{p+1}|}{k} \leq p+1$$

$$\left\lceil \frac{|N_{p+1}|}{k} \right\rceil = p+1.$$

Therefore:

$$t(S') \leq V_p + \left\lceil \frac{|N_{p+1}|}{k} \right\rceil$$

In order to express the upper bound on $t(S')$ in terms of $Q(V_p + 1)$ we consider two cases.

Case 1. $N_p \subseteq Q(V_p + 1)$. Figure 10 illustrates this case. We have:

$$|N_{p+1} - N_p| \leq k, Q(V_p + 1) \subseteq N_{p+1} \text{ and } N_p \subseteq Q(V_p + 1).$$

If $N_{p+1} = Q(V_p + 1)$, the result follows. Suppose $Q(V_p + 1) \subset N_{p+1}$. Then

$$k \geq |N_{p+1} - N_p| = |(N_{p+1} - Q(V_p + 1)) \cup (Q(V_p + 1) - N_p)|. \text{ Since}$$

$Q(V_p + 1) \subset N_{p+1}$, we write

$$k > |N_{p+1} - Q(V_p + 1)|$$

$$\text{So } \frac{|Q(V_p + 1)|}{k} < \frac{|N_{p+1}|}{k} < \frac{|Q(V_p + 1)| + k}{k} = \frac{|Q(V_p + 1)|}{k} + 1$$

$$\text{and } \left\lceil \frac{|N_{p+1}|}{k} \right\rceil = \left\lceil \frac{|Q(V_p + 1)|}{k} \right\rceil. \text{ Therefore, } t(S') \leq V_p + \left\lceil \frac{|Q(V_p + 1)|}{k} \right\rceil$$

and the result of the lemma follows.

Case 2. $N_p \not\leq Q(V_p + 1)$. Figure 11 illustrates this case. Here N_p contains at least one task at a level lower than $V_p + 1$. We consider two subcases.

(a) Suppose there exist no integer i , $1 \leq i \leq p$ such that

$N_i \leq Q(V_i + 1)$. Then, for $0 \leq i < p$, the number of tasks executed at the highest level in (t_i, t_{i+1}) is less than k . This, in turn, implies that the maximum level of tasks which have not completed after p intervals is no greater than $L - p$. Using the definition of p and Lemma 1, we have

$$t(S') \leq p + (L - p) = L$$

and the result of the lemma follows.

(b) Suppose there exists an integer i , $1 \leq i < p$ such that

$N_i \leq Q(V_i + 1)$ and let s denote the largest such integer.

By repeating previous analysis,

$$t(S') \leq V_p + (t_p - t_s) + \left\lceil \frac{|Q(V_s + 1)|}{k} \right\rceil$$

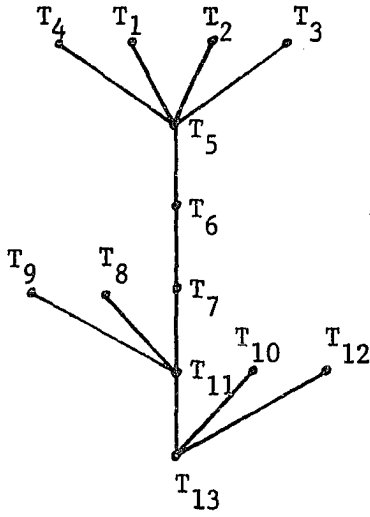
But $N_i \not\leq Q(V_i + 1)$ for $s < i \leq p$ means that the highest level of task remaining to be executed is reduced by $V_s - V_p$ in the time interval (t_s, t_p) and hence $V_p + (t_p - t_s) = V_s$. Therefore

$$t(S') \leq V_s + \left\lceil \frac{|Q(V_s + 1)|}{k} \right\rceil$$

and the results of the lemma follows.

Combining the result of Lemma 2 with Hu's result, we can now conclude that the optimal B-schedule is free from anomalies caused by reduction in execution times.

G:



$$\tau_1 = \tau_7 = \tau_{11} = \frac{1}{2}$$

$$\tau_2 = \tau_4 = \tau_8 = \tau_9 = \tau_{10} = \tau_{12} = \frac{3}{4}$$

$$\tau_3 = \tau_5 = \tau_6 = \tau_{13} = 1$$

$$\mathcal{L} = (T_1, \dots, T_{13})$$

$$k = 3$$

S:

0	1	2	3	4	5	
T ₁	T ₄	T ₅	T ₆	T ₇	T ₁₁	T ₁₃
T ₂	T ₈	T ₁₀				
T ₃	T ₉	T ₁₂				

$$t_0 = 0$$

$$p = 3 \quad N_1 \subset Q(V_1 + 1)$$

$$t_1 = 1 \quad N_1 = \{1, 2, 3\}$$

$$V_0 = 6 \quad N_2 \not\subset Q(V_2 + 1)$$

$$t_2 = 1 \frac{3}{4} \quad N_2 = \{1, 2, 3, 4, 8, 9\}$$

$$V_1 = 5 \quad Q(V_1 + 1) = \{T_1, \dots, T_4\}$$

$$t_3 = 2 \frac{1}{2} \quad N_3 = \{1, 2, 3, 4, 5, 8, 9, 10, 12\}$$

$$V_2 = 4 \quad Q(V_2 + 1) = \{T_1, \dots, T_5\}$$

$$t_4 = 3 \frac{1}{2} \quad N_4 = \{1, 2, 3, 4, 5, 6, 8, 9, 10, 12\}$$

$$V_3 = 3 \quad Q(V_3 + 1) = \{T_1, \dots, T_6\}$$

$$t_5 = 4 \frac{1}{2} \quad N_5 = \{1, \dots, 12\}$$

$$V_4 = 1$$

Figure 11. Case where $N_p \not\subseteq Q(V_p + 1)$

Theorem 3. Given a tree-structured precedence graph G , let S denote the optimal B-schedule if the tasks have equal execution times ($\tau_i = 1, \forall i$) and S' denote the B-schedule if the tasks have unequal execution times ($\tau'_i \leq 1, \forall i$). Then $t(S') \leq t(S)$.

4. Combined effects

So far, in Section A, we have established the weak stability inherent in the optimal B-schedule S if we relax a single parameter. It immediately follows that conditions on all parameters may be collectively relaxed according to a specified order without increasing the schedule length.

Theorem 4. Given a tree-structured precedence graph for a system of tasks with equal execution times, the B-schedule is stable under any collective increase in the number of processors and allowable relaxation of the partial order followed by a decrease in the execution times of some tasks.

Proof: Suppose we increase the number of processors to k' forming the schedule S' . By Theorem 1, $t(S') \leq t(S)$. Next, perform the allowable relaxations to form \leq' and the resulting schedule S'' . Since S' is an optimal B-schedule for k' processors and tasks with equal execution times, we know by the Corollary to Theorem 2 that $t(S'') \leq t(S')$. Finally, perform the reductions in execution times to form the schedule S''' . Since S'' is an optimal B-schedule for k' processors, partial order \leq' defined by a forest-structured precedence graph, and tasks with equal execution times, we know by Theorem 3 that $t(S''') \leq t(S'')$. Therefore, $t(S''') \leq t(S)$. The proof is the same if we relax the partial order

first and increase the number of processors second.

The theorem is somewhat restrictive in the sense that at run time, we would have to recalculate the priority list \mathcal{L} after the "allowable relaxation" of G and before the decrease in the execution times. In the next chapter, we conjecture a more powerful statement which asserts the stability of the B-schedule under any allowable collective change in the original parameters without a run-time recalculation of the priority list \mathcal{L} .

5. Stability of the B-schedule for variants of tree-structured graphs

Here we consider several tasking systems in which the precedence graph has a tree-like structure.

We define an allowable relaxation in a p tree-restricted precedence graph G as the removal of an arc from any subgraph G_i , $1 \leq i \leq p$, such that the new precedence graph of G_i' is a tree or forest. The following theorems show that for any p tree-restricted precedence graph G , the B-schedule is free of the three anomalies commonly found in multi-processor systems.

Theorem 5. Given a p tree-restricted precedence graph G with equally weighted tasks, the B-schedule is stable if the number of processors is increased, or under an allowable relaxation of G or if the execution time of some tasks are reduced.

Proof: Here we only give the proof when the number of processors is increased because the proofs for the other two cases are similar. By Theorem 1, the B-schedule is stable for each subgraph G_i of G if

the number of processors is increased. Also by the definition of a p tree-restricted precedence graph, all tasks in G_{i-1} must be completed before any task in G_i may begin execution. Since the schedule length is the sum of the schedules for the G_i , the result follows.

Theorem 6. Given a p tree-restricted precedence graph G with equally weighted tasks, the B-schedule is stable under any collective increase in the number of processors and allowable relaxation of the partial order followed by a decrease in the execution times of some tasks.

We omit the proof of this theorem because it is similar to the proof of Theorem 5.

In conclusion, we point out that the results of Section A also hold for forests. They also apply to reverse forests and reverse p tree-restricted precedence graphs if the reverse of the original graph is considered and the resulting schedule is reversed.

B. A-Schedules

In this section, we study the stability properties of the schedules produced by the A-algorithm. This algorithm produces optimal schedules for two-processor systems in which tasks have equal execution times and are partially ordered by a general precedence graph. These schedules are found to be free of two of three anomalies commonly found in multi-processing systems. Apart from some extensions, much of the work reported in this section originated in unpublished literature.¹

¹M. L. Liu and A. E. Oldehoeft, Department of Computer Science, Iowa State University, Private communication, 1974.

We begin with the optimal A-schedule S for the case where $k = 2$ and $\tau_i = 1$ for all i , and investigate the stability of this schedule by decreasing some of the τ_i , relaxing \leq , and increasing the number of processors to form the new schedule S' . Since a change in priority list can obviously result in increased schedule length, we hold ℓ fixed as defined for the optimal A-schedule S .

We begin with the following lemma which is valid for any task set with equal execution times and an arbitrary number of processors. Recall that $\alpha(T)$ stands for the label assigned to task T by the labeling algorithm used by the A-algorithm.

Lemma 3. Let $t(T)$ denote the time at which task T begins execution in an A-schedule. If T is executed on processor P_1 , then for any task T' such that $t(T) \leq t(T')$, we have $\alpha(T) > \alpha(T')$.

The proof is omitted here since it is identical to that given in (6) for $k = 2$ processors.

1. Increasing the number of processors

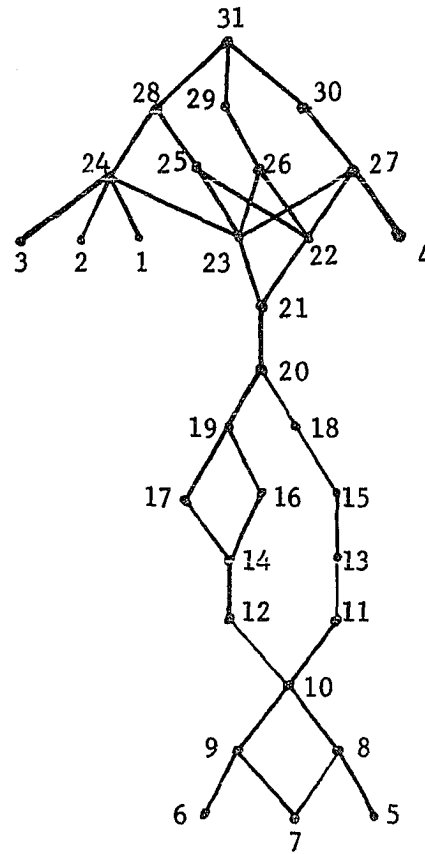
Given a task set with equal execution times, let S and S' denote the corresponding A-schedule on $k = 2$ and $k' > 2$ processors, respectively. Figure 12 illustrates the typical effect of increasing the number of processors. We now describe Algorithm 1 which will partition the schedule S' into disjoint sets X_i consisting of $2n_i - 1$ tasks such that

$$(1) \quad t(S') = \sum_i n_i$$




$$(2) \quad \text{each task in } X_{i+1} \text{ is a predecessor of each task in } X_i.$$

The result of the partition will allow us to conclude that the A-schedule is inherently free of the anomaly caused by increasing the

G:














S:

P ₁	31	30	28	26	24	23	21	20	19	17	15	13	11	10	9	7	5
P ₂		29	27	25	22	4	3	2	18	16	14	12	1		8	6	
t=0	5					10					15						

A-schedule on G with 2 processors, $t(S) = 17$

S':

P ₁	31	30	27	24	23	21	20	19	17	14	12	10	9	7
P ₂		29	26	22	3	1		18	16	13	11		8	6
P ₃		28	25	4	2				15					5
t=0	5					10								

A-schedule for G when the number of processors is increased, $t(S') = 14$

Figure 12. Effect of increasing the number of processors in an A-schedule

number of processors. The trace of applying Algorithm 1 to S' in Figure 12 is given in Figure 13.

Algorithm 1

Let $\alpha(\phi) = 0$; let V_0 denote the last task executed by P_1 , let W_0 denote the (possibly empty) task executed by P_2 with $t(W_0) = t(V_0)$.

begin

$\ell \leftarrow 0$; $X_0 \leftarrow V_0$; $t \leftarrow t(S') - 2$;

while $t \geq 0$ do

begin let T_i denote the task executed by P_i , $1 \leq i \leq k'$ at time t

if $\alpha(T_2) < \alpha(V_\ell)$

then begin $\ell \leftarrow \ell + 1$; $V_\ell \leftarrow T_1$; $W_\ell \leftarrow T_2$; $X_\ell \leftarrow \{T_1\}$ end

else $X_\ell \leftarrow X_\ell \cup T_1 \cup T_2$

while $\{\ell \neq 0$ and there exists a task $\bar{T} \notin X_j$ ($0 \leq j \leq \ell$)

such that $t(\bar{T}) \geq t$ and $\alpha(\bar{T}) > \alpha(V_{\ell-1})\}$ do

begin $X_{\ell-1} \leftarrow X_{\ell-1} \cup X_\ell \cup \bar{T}$; $\ell \leftarrow \ell - 1$ end

$t \leftarrow t - 1$

end

end

If we assume the partition of S' to define the sets X_j for $0 \leq j \leq \ell$ at the termination of Algorithm 1, the following lemmas are direct results of the algorithm.

Lemma 4. For $0 \leq j \leq \ell$, the cardinality of each set X_j is odd, i.e. $|X_j| = 2n_j - 1$.

Lemma 5. The length of the schedule for k' processors is

$$t(S') = \sum_{j=0}^{\ell} n_j.$$

	x_2		x_1				x_0							
P_1	31	30	27	24	23	21	20	19	17	14	12	10	9	7
P_2		29	26	22	3	1		18	16	13	11		8	6
P_3		28	25	4	2			15						5

Trace:

ℓ	Partitioning activity	t
0	$v_0 \leftarrow 7, x_0 \leftarrow \{7\}$	12
0	$x_0 \leftarrow \{7, 8, 9\}$	$12 \rightarrow 11$
$0 \rightarrow 1$	$v_1 \leftarrow 10, w_1 \leftarrow \emptyset, x_1 \leftarrow \{10\}$	$11 \rightarrow 10$
1	$x_1 \leftarrow \{10, 11, 12\}$	$10 \rightarrow 9$
1	$x_1 \leftarrow \{10-14\}$	$9 \rightarrow 8$
$1 \rightarrow 0$	$x_1 \leftarrow \{10-14, 16, 17\}$	$8 \rightarrow 7$
	$x_0 \leftarrow x_0 \cup x_1 \cup \{15\} = \{7-17\}$	
0	$x_0 \leftarrow \{7-19\}$	$7 \rightarrow 6$
$0 \rightarrow 1$	$v_1 \leftarrow 20, w_1 \leftarrow \emptyset, x_1 \leftarrow \{20\}$	$6 \rightarrow 5$
$1 \rightarrow 2$	$v_2 \leftarrow 21, w_2 \leftarrow \emptyset, x_2 \leftarrow \{21\}$	$5 \rightarrow 4$
$2 \rightarrow 3$	$v_3 \leftarrow 23, w_3 \leftarrow \emptyset, x_3 \leftarrow \{23\}$	$4 \rightarrow 3$
$3 \rightarrow 4$	$v_4 \leftarrow 24, w_4 \leftarrow \emptyset, x_4 \leftarrow \{24\}$	$3 \rightarrow 2$
$4 \rightarrow 3 \rightarrow 2$	$x_4 \leftarrow \{24, 26, 27\}$	$2 \rightarrow 1$
	$x_3 \leftarrow x_3 \cup x_4 \cup \{25\} = \{23-27\}$	
	$x_2 \leftarrow x_2 \cup x_3 \cup \{22\} = \{21-27\}$	
2	$x_2 \leftarrow \{21-27, 29, 30\}$	
$2 \rightarrow 1$	$x_1 \leftarrow x_1 \cup x_2 \cup \{28\} = \{20-30\}$	$1 \rightarrow 0$
$1 \rightarrow 2$	$v_2 \leftarrow 31, w_2 \leftarrow \emptyset, x_2 = \{31\}$	$0 \rightarrow -1$

Figure 13. Partition of S' in Figure 12 using Algorithm 1

Lemma 6. For each task $T \in X_j$, the labels are such that $\alpha(T) \geq \alpha(V_j) > \alpha(W_{j+1})$.

Lemma 7. Any task $\bar{T} \notin \bigcup_{j=0}^{\ell} X_j$ and executed at time $t(\bar{T}) = t(T)$ for some $T \in X_j$ ($j \geq 1$) has the property $\alpha(\bar{T}) < \alpha(V_{j-1})$.

The following theorem shows that the optimal A-schedule for two processors is free of the anomaly caused by increasing the number of processors.

Theorem 7. Given a precedence graph G with equally weighted nodes, the A-schedule for two processors is stable if the number of processors is increased to $k' > 2$.

Proof: For $0 < j \leq \ell$, if $T \in X_j$ and $T' \in X_{j-1}$, then it follows that $T' \prec T$. The proof of this fact makes use of Lemmas 3, 6 and 7. It is omitted here since it is essentially identical to the argument in (6) as part of the proof that the A-algorithm is optimal for the two-processor systems. Consequently, every task in X_{j+1} , $0 \leq j < \ell$, must be completed before any task in X_j can start in schedule S (for the two processor system) as well as in schedule S' (for the $k' > 2$ processor system). Using Lemmas 4 and 5, we have

$$t(S) \geq \sum_{j=0}^{\ell} n_j = t(S')$$

2. Relaxation of the partial order

A simple relaxation of the partial order \prec is defined as the removal of the precedence of an immediate predecessor-successor pair (T_i, T_j) while preserving the transitivity implied by the presence of the pair. More precisely, a simple relaxation of $T_i \prec T_j$ is defined by an operation on the precedence graph G in which

- (1) the arc (T_i, T_j) is removed from G
- (2) for each arc (T_k, T_i) in G , an arc (T_k, T_j) is added if necessary to insure $T_k < T_j$.
- (3) for each arc (T_j, T_q) in G , an arc (T_i, T_q) is added if necessary to insure $T_i < T_q$.

The resulting precedence graph G' defines a new partial order $<'$ which is contained in $<$. By a relaxation, we will mean the effect of any sequence of simple relaxations. Figure 14 illustrates the typical effect of a relaxation on an A-schedule for a two-processor system.

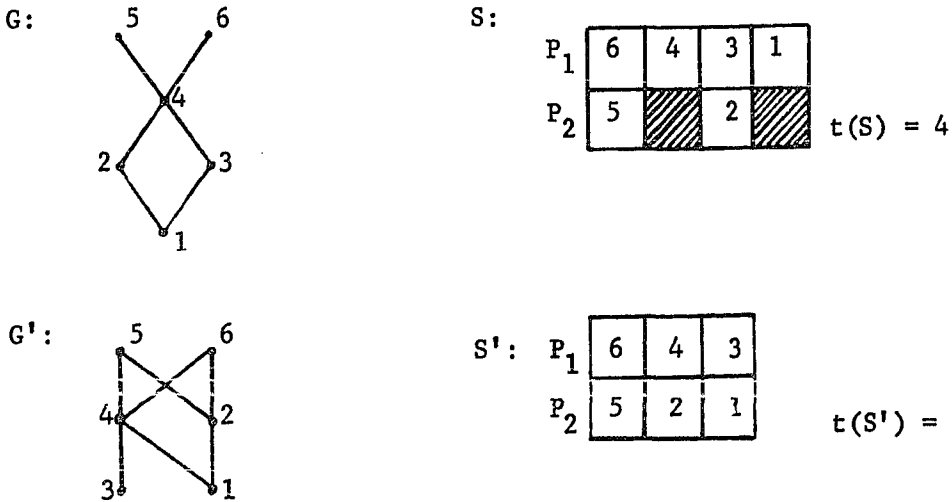


Figure 14. Relaxation of partial order by removing arcs $(4, 2)$ and $(3, 1)$ from G

As before, we let S denote the two-processor A-schedule for an equally weighted task system. Let S' denote the list schedule for the task system with the relaxed partial order using the original priority list ℓ . In a method similar to the previous subsection, we define an algorithm to partition S' . The result of the partition will allow us to conclude that the A-schedule is inherently free of the

anomaly caused by relaxing the partial order. A trace of applying this algorithm is given in Figure 15.

Algorithm 2

Let $\alpha(\phi) = 0$; let V_0 denote the last task executed by P_1 , let W_0 denote the (possibly empty) task executed by P_2 with $t(W_0) = t(V_0)$.

begin

$\ell \leftarrow 0$; $X_0 \leftarrow V_0$; $t \leftarrow (S') - 2$;

while $t \geq 0$ do

begin let T_i denote the task executed by P_i , $i = 1, 2$ at time t

if $\alpha(T_2) < \alpha(V_\ell)$

then begin $\ell \leftarrow \ell + 1$; $V_\ell \leftarrow T_1$; $W_\ell \leftarrow T_2$; $X_\ell \leftarrow \{T_1\}$ end

else $X_\ell \leftarrow X_\ell \cup T_1 \cup T_2$

while $\{\ell \neq 0$ and there exists $\bar{T} \notin X_j$, $0 \leq j \leq \ell$, such that

$\alpha(\bar{T}) > \alpha(V_{\ell-1})$, $t(\bar{T}) < t(V_\ell)$ and $\bar{T} \in S(T)$ in G

for some $T \in X_\ell\}$ do

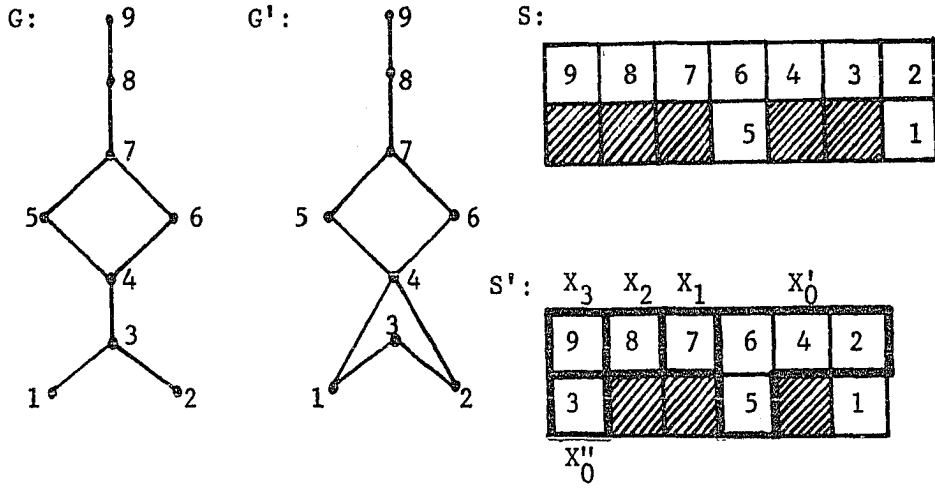
begin $X_{\ell-1} \leftarrow X_{\ell-1} \cup X_\ell \cup \bar{T}$; $\ell \leftarrow \ell - 1$ end

$t \leftarrow t - 1$

end

end

As direct results of the partitioning process, we see that Lemmas 4, 5, and 6 are also valid for the partition of S' produced by Algorithm 2. These and the following lemma are used in Theorem 8 to show that a relaxation of the partial order will not cause an anomaly in the two-processor A-schedule.



(G' is constructed from G by removing $T_i \triangleleft T_3$ for $4 \leq i \leq 9$.)

Trace:

\underline{l}	Partitioning Activity	\underline{t}
0	$V_0 \leftarrow 2, W_0 \leftarrow 1, X_0 \leftarrow \{2\}$	4
$0 \rightarrow 1 \rightarrow 0$	$V_1 \leftarrow 4, W_1 \leftarrow \emptyset, X_1 \leftarrow \{4\}$ $X_0 \leftarrow X_0 \cup X_1 \cup \{3\} = \{2, 3, 4\}$	$4 \rightarrow 3$
0	$X_0 \leftarrow \{2-5\}$	$3 \rightarrow 2$
$0 \rightarrow 1$	$V_1 \leftarrow 7, W_1 \leftarrow \emptyset, X_1 \leftarrow \{7\}$	$2 \rightarrow 1$
$1 \rightarrow 2$	$V_2 \leftarrow 8, W_2 \leftarrow \emptyset, X_2 \leftarrow \{8\}$	$1 \rightarrow 0$
$2 \rightarrow 3$	$V_3 \leftarrow 9, W_3 \leftarrow 3, X_3 \leftarrow \{9\}$	$0 \rightarrow -1$

Figure 15. Partition of S' using Algorithm 2

Lemma 8. For any $T \in X_j$, $j = 1, \dots, \ell$, there does not exist $\bar{T} \in S(T)$ in G (original precedence graph) with the properties $t(\bar{T}) < t(V_j) - n_j + 1$, $\bar{T} \notin X_j$, and $\alpha(\bar{T}) > \alpha(V_{j-1})$.

Proof: The innermost while loop in Algorithm 2 eliminates the possibility of such a \bar{T} .

Theorem 8. Given a general precedence graph G with equally weighted tasks, the two-processor A-schedule is stable under any relaxation on G .

Proof: The heart of the proof is to show that for $T' \in X_{j+1}$, we must have $T' \leq T$ for all $T \in X_j$. To facilitate the discussion, let $S(T)$ and $S'(T)$ denote the immediate successors of T in G and G' , respectively. We first establish that $V_{j+1} \leq T$ (according to G) for all $T \in X_j$, $0 \leq j < \ell$. There are two cases to consider.

Case 1. Suppose $t(V_j) - n_j + 1 \leq t(T) \leq t(V_j)$ for some $0 \leq j < \ell$ in S' . From Lemma 6, $\alpha(T) > \alpha(V_{j+1})$ so that T was considered for execution on P_2 at time $t(V_{j+1})$. Since it was not executed at that time, some predecessor T' of T had not yet completed execution, i.e. $t(T') \geq t(V_{j+1})$. But $T' \leq T$ implies $T' \leq T$ so that $\alpha(T') > \alpha(T)$ and $t(T') < t(T)$. If $t(T') = t(V_{j+1}) = t(V_j) - n_j$, then $T' = V_{j+1}$ and we are done. Otherwise, we can follow the discussion in (6) to establish by induction that $V_{j+1} \leq T$.

Case 2. Suppose $t(T) < t(V_{j+1})$ in S' . By the design of Algorithm 2, $T \in S(T')$ for some $T' \in X_j$ and $t(V_j) - n_j + 1 \leq t(T') \leq t(V_j)$. By Case 1, $V_{j+1} \leq T'$ and by transitivity $V_{j+1} \leq T$.

Thus, for all $T \in X_j$, we have established that $V_{j+1} \leq T$. Let I_j denote the set of tasks in X_j which have no predecessors in X_j . Since

$V_{j+1} < T$ for all $T \in X_j$, it follows that $S(V_{j+1}) \cap X_j = I_j$. We next establish that $T' \in X_{j+1}$ and $T \in X_j$ implies $T' < T$ (according to G) for $0 \leq j < l$.

Consider first the case where T' has no successors in X_{j+1} . There are two cases.

Case a. Suppose T' has a successor \bar{T} such that $t(\bar{T}) \geq t(V_{j+1}) - n_{j+1} + 1$. In this case either $\bar{T} \in X_{j-i}$ for some $i \geq 0$ or \bar{T} is of no interest of us.

Case b. Suppose T' has a successor \bar{T} such that $t(\bar{T}) < t(V_{j+1}) - n_{j+1} + 1$. By Lemma 8, we note that $\alpha(\bar{T}) < \alpha(V_j)$ since $\bar{T} \notin X_{j+1}$. Then $\bar{T} \notin X_i$ for $i > j + 1$ since otherwise by Lemma 6, $\alpha(\bar{T}) > \alpha(V_i)$. But from the first part of the proof, $\alpha(V_i) > \alpha(V_j)$ which is a contradiction.

In any case we have established that if $\bar{T} \in S(T')$, then $\bar{T} \in X_{j-i}$ for some $i \geq 0$ or \bar{T} is of no interest to us. We now have $S(T') \cap X_{j+1} = \emptyset$ by assumption, $S(V_{j+1}) \cap X_j = I_j$ by the first part of proof and $N(T') \geq N(V_{j+1})$ since $\alpha(T') \geq \alpha(V_{j+1})$. In a manner identical to (6), we can show $S(T') \cap X_j = I_j$. Furthermore no \bar{T} is in X_{j-i} for $i > 0$ since then for some $T'' \in I_j$, we have $T' < T'' < \bar{T}$ contradicting $\bar{T} \in S(T')$.

We have established that $T' \in X_{j+1}$ with no successor in X_{j+1} and $T \in X_j$ implies $T' < T$. Consider next the case where T' has a successor in X_{j+1} . By transitivity, it follows that $T' < T$ for all $T \in X_j$.

By Lemma 4, X_j consists of $2n_j - 1$ tasks. Since $T' \in X_{j+1}$ and $T \in X_j$ implies $T' < T$, we have

$$t(S) \geq \sum_{j=0}^{\ell} n_j = t(S')$$

3. Reduction of task execution times

Here we illustrate the instability of the A-schedule if some of the task execution times are decreased. Figure 16 presents a specific example of this type of anomaly. In order to insure stability under these conditions, additional precedence constraints (15) would have to be placed on the task system. This new system, with additional precedence constraints, is free from all three anomalies.

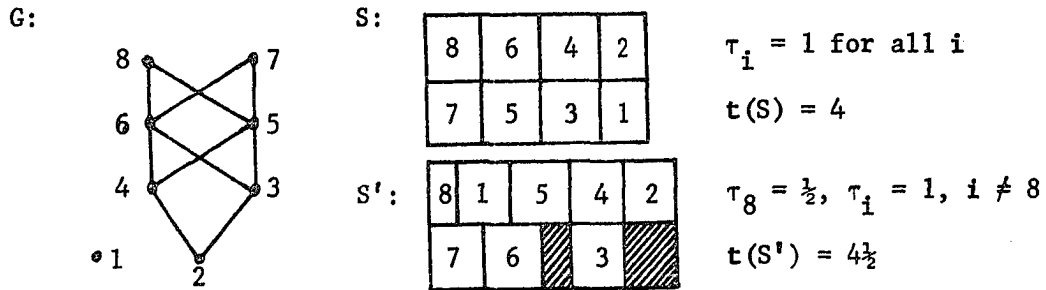


Figure 16. Anomaly in A-schedule caused by decrease in execution times

4. Stability of the optimal preemptive schedule for two-processor systems

The stability properties of the A-schedule, established in previous discussion, can be used to derive stability properties of optimal preemptive schedules for two-processor systems. Assume that the execution times, $\tau = (\tau_1, \dots, \tau_n)$ are mutually commensurable, i.e., there exists a real number w such that $\tau_i = n_i w$ for some integer n_i . Recall G_w to be the precedence graph derived from G by replacing each task T_i by n_i mutually commensurable tasks $T_{i,1}, \dots, T_{i,n_i}$ each with execution time w . Define $\{G_{w/m}\}$, $m \geq 1$ to be a sequence of precedence graphs

derived from G_w by replacing each task with m mutually commensurable tasks, each with weight w/m . Let S_m denote the A-schedule for $G_{w/m}$ on a two-processor system; S'_m denote the A-schedule for $G_{w/m}$ for $k' > 2$ processors; and S''_m denote the A-schedule for $G''_{w/m}$ on a two-processor system where G'' is derived from G by a relaxation of the partial order and $G''_{w/m}$ is derived from G'' in the manner described above.

From the previous discussion we know that $t(S'_m) \leq t(S_m)$ and $t(S''_m) \leq t(S_m)$. If we take the limit as m approaches infinity, then S_m , S'_m and S''_m converge to general schedules. These general schedules are equivalent, respectively, to preemptive schedules S , S' , and S'' produced by the so-called C-algorithm (5). Furthermore S is an optimal preemptive schedule. Taking limits on the inequalities above, we have $t(S') \leq t(S)$ and $t(S'') \leq t(S)$ resulting in the following theorem.

Theorem 9. Given a general precedence graph G with unequally weighted tasks, the two-processor schedule produced by the C-algorithm is stable if the number of processors is increased to $k' > 2$ or the partial order is relaxed.

Note that we have stability under the combined effects of first relaxing the partial order and then increasing the number of processors. This follows from the fact that S'' is an optimal preemptive schedule for the two-processor system and precedence graph G'' .

Since the A-schedule is not stable when some execution times are decreased, we can infer nothing about the corresponding stability of the optimal preemptive schedule. However, if additional precedence constraints

(15) are placed on the task system, our new system is free from all three anomalies.

C. C-Schedules for Tree-Structured Graphs

In this section, we study the stability properties of the schedules produced by the C-algorithm when it is applied to tree-structured graphs. These schedules are found to be free from three anomalies commonly found in multiprocessor systems. The study of the stability properties of the C-schedules on a two-processor system is given in two different sections, Section B and Section D, following two different approaches. Recall from Chapter II that the C-algorithm produces an optimal preemptive schedule for a tree-structured graph, i.e., $CT_{PS}(G, k) = CT_C(G, k)$, and that the execution times are assumed to be mutually commensurable.

In order to prove the optimality of the C-algorithm when it is applied to tree-structured graphs, Muntz and Coffman (19) show that if we form $G_{w/r}$, the graph obtained from G by splitting each mutually commensurable task in r subtasks, then $CT_{BS}(G_{w/r}, k) \rightarrow CT_{PS}(G, k)$ as $r \rightarrow \infty$. They also show that there exists an integer v such that:

$$CT_{BS}(G_{w/mv}, k) = CT_C(G, k) \text{ for } m = 1, 2, \dots$$

In other words, the optimal limit can be achieved in a finite number of steps.

We investigate the stability of this schedule by decreasing some of τ_i , relaxing \leq , and increasing the number of processors to form a new schedule.

1. Increasing the number of processors

We know that $CT_{PS}(G, k) = CT_C(G, k)$ and $CT_{BS}(G_{w/mv}, k) = CT_C(G, k)$.

It was shown in Theorem 1 that the optimal nonpreemptive schedule for tree-structured graphs and equally weighted tasks is stable when the number of processors is increased, i.e., $CT_{BS}(G_{w/mv}, k') \leq CT_{BS}(G_{w/mv}, k)$.

Also, it is known that the preemptive discipline is more powerful than the nonpreemptive discipline, i.e., $CT_{PS}(G, k) \leq CT_{BS}(G, k)$.

Combining all the relations given above, we find $CT_C(G, k') \leq CT_C(G, k)$.

Therefore, we have the following theorem.

Theorem 10. Given a tree-structured precedence graph G , the C -schedule is stable if the number of processors is increased.

2. Relaxation of the partial order

In this part, we investigate the stability when the partial order is relaxed. In order to insure that $<' \subseteq <$ and the new precedence graph G' , defined by $<'$, is a tree or forest, we use the same definition of an "allowable relaxation" given in Section III.A.

We know that $CT_{PS}(G, k) = CT_C(G, k)$ and $CT_{BS}(G_{w/mv}, k) = CT_C(G, k)$.

The corollary of Theorem 2 shows that the optimal nonpreemptive schedule for tree-structured graphs and equally weighted tasks is stable under

any allowable relaxation in G , i.e. $CT_{BS}(G'_{w/mv}, k) \leq CT_{BS}(G_{w/mv}, k)$.

Also, it is known that $CT_{PS}(G', k) \leq CT_{BS}(G'_{w/mv}, k)$. Then, combining all the relations given above, we find that $CT_C(G', k) \leq CT_C(G, k)$.

Therefore, we have the following theorem.

Theorem 11. Given a tree-structured graph G , the C -schedule is stable under any allowable relaxation of G .

3. Reduction of task execution times

We know that $CT_{PS}(G, k) = CT_C(G, k)$ and $CT_{BS}(G_{w/mv}, k) = CT_C(G, k)$. By Theorem 3, the optimal nonpreemptive schedule for equally weighted tasks and tree-structured graphs is stable when the execution time of one or more tasks is reduced, i.e., $t_{BS}(G'_{w/mv}, k) \leq CT_{BS}(G_{w/mv}, k)$ where $t_{BS}(G'_{w/mv}, k)$ is the completion time for the new graph G' . Also, it is known that an optimal preemptive schedule is no longer than a non-preemptive schedule, i.e., $CT_{PS}(G'_{w/mv}, k) \leq t_{BS}(G'_{w/mv}, k)$, and that $CT_{PS}(G', k) = CT_{PS}(G'_{w/mv}, k)$. Combining all the relations given above, we obtain $CT_C(G', k) \leq CT_C(G, k)$. Therefore, we have the following theorem.

Theorem 12. Given a tree-structured precedence graph G , the C-schedule is stable when the execution time of one or more tasks is reduced.

4. Combined effects

Thus far, we have established the weak stability inherent in the optimal C-schedule S if we relax a single parameter. It immediately follows that conditions on all parameters may be collectively relaxed without increasing the schedule length.

Theorem 13. Given a tree-structured precedence graph G , the C-schedule is stable under any collective increase in the number of processors, allowable relaxations of the partial order and a decrease in the execution time of some tasks.

Proof: Suppose we increase the number of processors to k' , forming the schedule S' . By Theorem 10, $t(S') \leq t(S)$. Next, perform the

allowable relaxations to form $\langle \cdot \rangle'$ and the resulting schedule S'' . By Theorem 11, $t(S'') \leq t(S')$. Finally, perform the reductions in execution time to form the schedule S''' . By Theorem 12, $t(S''') \leq t(S'')$. Therefore $t(S''') \leq t(S)$. The proof for the other five different cases is similar.

5. Stability of the C-schedule for variants of tree-structured graphs

Here, we consider several tasking systems in which the precedence graph has a tree-like structure. We also use the same definition of an allowable relaxation in a p tree-restricted precedence graph given in Section III.A.5. The following theorems show that for any p tree-restricted precedence graph, the C-schedule is free of the three anomalies commonly found in multiprocessor systems.

Theorem 14. Given a p tree-restricted precedence graph, the C-schedule is stable if the number of processors is increased, or under any allowable relaxation of G , or if the execution time of some tasks are reduced.

Proof: Here we only give the proof when the number of processors is increased because the proofs for the other two cases are similar. By Theorem 10, the C-schedule is stable for each subgraph G_i of G if the number of processors is increased. Also, by the definition of a p tree-restricted precedence graph, all tasks in G_{i-1} must be completed before any task in G_i may begin execution. Since the schedule length is the sum of the schedules for the G_i , the result follows.

Theorem 15. Given a p tree-restricted precedence graph G , the C-schedule is stable under any collective increase in the number of

processors, allowable relaxations of the partial order and a decrease in the execution times of some tasks.

We omit the proof of this theorem because it is similar to the proof of Theorem 14.

In conclusion, we point out that the results of Section C also hold for forests. They also apply to reverse forests and reverse p tree-restricted precedence graphs if the reverse of the original graph is considered and the resulting schedule is reversed.

D. Subset Assignment Schedules

In this section, we study the stability properties of preemptive schedules for k -processor systems in which the tasks are partially ordered by a general precedence graph. These schedules are found to be free from three anomalies that commonly occur in multiprocessor systems. Our interest in these schedules is motivated by the fact that they are optimal under certain conditions and in general are expected to be a good heuristic.

Because the subset assignment schedules assume unit tasks, we start with a general graph G with mutually commensurable task times. We then compute the graph G_w and form a schedule using a special subset sequence for G_w . The algorithm for constructing a subset sequence for G_w , to be called Subset Assignment Algorithm, SAA, is defined in the following manner. Nodes are assigned to subsets level by level, except when there are $u < k$ nodes at the highest level. In the latter case, as many as possible of the $k - u$ nodes necessary to "fill" the

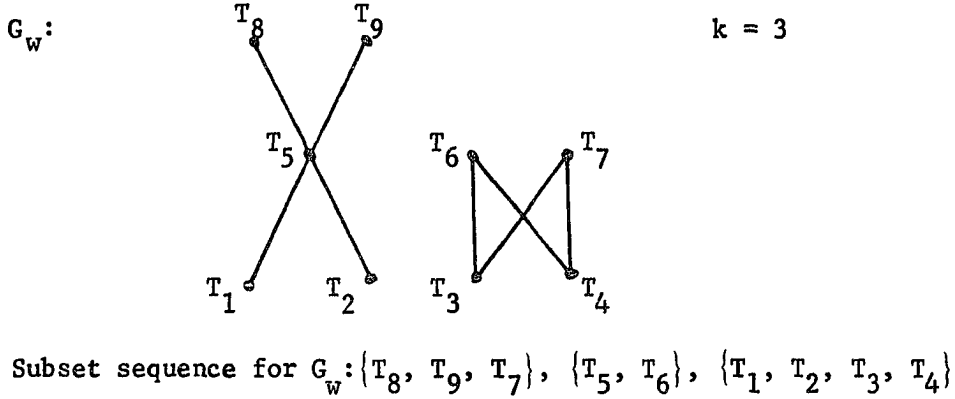
subset are chosen from the executable nodes at lower levels. Of course, if there are more than $k - u$ nodes to choose from, those at the higher levels are selected first. Note that the SAA is a generalization of the algorithm given by Muntz and Coffman (18) that constructs a subset sequence for G_w which corresponds to an optimal subset assignment for G_w with $k = 2$. Also, an optimal subset assignment for G_w is an optimal preemptive schedule for G when $k = 2$ (18). The SAA might not produce an optimal subset assignment for arbitrary graphs when $k > 2$. In the future, we will use SAS to denote a subset assignment schedule produced by the SAA. An example illustrating the SAS is given in Figure 17.

From the definition of the SAS, we see that each subset S_i , for $1 \leq i \leq \frac{L}{w}$, consists of mutually independent nodes. Therefore, using McNaughton's result (16), the minimum length schedule for S_i requires

$$t_i = \max \left\{ \frac{|S_i|}{k}, 1 \right\} \times w$$

units, where $|S_i|$ is the number of tasks in S_i . Moreover, the SAS will have a schedule length $t(S) = \sum_{i=1}^{L/w} t_i$. We will show that the SAS is free from three anomalies that commonly occur in multiprocessing systems, considering two different cases: 1) using the original subset sequence of G for the new schedule S' , and 2) recalculating the subset sequence for S' .

Suppose we do not recompute the subset sequence for a k' ($k' > k$) processor system. In other words, we use the original subset sequence for G_w as defined for the k -processor system. It is clear that increasing the number of processors will not increase the schedule length. If we do not recompute the subset sequence when the partial order is



S:

P_1	T_8	T_5	T_1	T_2
P_2	T_9	T_6	T_2	T_3
P_3	T_7		T_3	T_4
$t=0$	1	2	3	

$t(S) = 3 \frac{1}{3}$

Subset assignment schedule for G_w

Figure 17. Illustration of the subset assignment schedule

relaxed, we will be using the original subset sequence from G_w for the new graph G'_w , and, as a result, we obtain exactly the same schedule.

If we do not recompute the subset sequence when the execution time of some tasks is decreased, we will be using the subset sequence with unequal weights. In this case, we only have to study the behavior of

a particular subset when the execution of one or more tasks is reduced.

First, let $t(S) = \sum_{j=1}^{L/w} t_j$ be the completion time for the graph G_w and $t(S') = \sum_{j=1}^{L/w} t'_j$ be the completion time for G'_w when $\tau'_i \leq w$ for all i .

Using McNaughton's result (16) for independent tasks, we have

$$t'_j = \max \left\{ \max_{1 \leq i \leq |S_j|} \{\tau'_i\}, \frac{1}{k} \sum_{i=1}^{|S_j|} \tau'_i \right\} \leq t_j .$$

Thus, $t(S') \leq t(S)$ and we have the following theorem.

Theorem 16. The Subset Assignment Schedule, SAS, is stable under any collective increase in the number of processors, allowable relaxations of the partial order and a decrease in the execution times of some tasks if we hold the subset sequence for the graph G fixed.

Because the subset assignment and the C-Algorithm both produce optimal preemptive schedules for a two-processor system, Theorem 9 could be stated as a corollary of Theorem 16.

From now on, we will study the stability of the Subset Assignment Schedules, SAS, in the more interesting case when we recalculate the subset sequence for the new schedule S' .

1. Increasing the number of processors

If we increase the number of processors from k to k' , we wish to show that the resulting schedule length does not increase. Using the SAA, we form the sequence of subsets $S_1, S_2, \dots, S_{L/w}$ for a k -processor system. We then consider the sequence $\{S_i\}$ using k' -processors and we study the schedule length effect of moving nodes between subsets until we arrive at the subset sequence for k' -processors produced by the SAA. Since a finite number of moves are needed to form the subset sequence

for k' -processors, it will suffice to show that a single move will not result in an increase in the schedule length. Increasing the number of processors allows the possibility of moving nodes from a lower-level subset to a higher-level subset in a manner described by the following lemma.

Lemma 9. Let S_j and S_i be two subsets obtained by the SAA for G_w in a k -processor system. Then, any move of a node from S_j to S_i , $i < j$, required by the SAA for a k' -processor system will not result in an increase in the schedule length.

Proof: Define S'_i and S'_j as the subsets obtained from S_i and S_j after moving a node from S_j to S_i . Therefore $|S'_i| = |S_i| + 1$ and $|S'_j| = |S_j| - 1$. Let t_i , t_j , t'_i and t'_j be the time needed to execute S_i , S_j , S'_i and S'_j , respectively. We need not consider cases when $|S_j| = 1$ since such a node cannot be moved due to precedence constraints. We also need not consider the cases when $|S_i| \geq k'$ because the movement of a node from a lower-level subset S_j to a higher-level subset S_i is not allowed by the SAA when $|S_i| \geq k'$. There are four possible cases when a node can be moved from S_j to S_i , $i < j$.

Case 1. Suppose $2 \leq |S_j| \leq k'$ and $|S_i| \leq k$. Then clearly $t'_i = t_i$ and $t'_j = t_j$ since $k < k'$.

Case 2. Suppose $|S_j| > k'$ and $|S_i| \leq k$. Then clearly $t'_i = t_i$ and $t'_j < t_j$ since $k < k'$.

Case 3. Suppose $2 \leq |S_j| \leq k'$ and $k < |S_i| < k'$. Then clearly $t'_i = t_i$ and $t'_j = t_j$.

Case 4. Suppose $|S_j| > k'$ and $k < |S_i| < k'$. Then clearly $t'_i = t_i$ and $t'_j < t_j$.

In each of the above cases, it follows that $t'_i + t'_j \leq t_i + t_j$. Also $t'_\ell = t_\ell$ for $\ell \neq i, j$. Therefore $\sum_{\ell=1}^{L/w} t'_\ell \leq \sum_{\ell=1}^{L/w} t_\ell$.

Figure 18 shows the typical effect of increasing the number of processors.

Let S and S'' denote the SAS for G on a k -processor system and on a k' -processor system, respectively; and S' denote the schedule for G in a k' -processor system using the subset sequence produced by the SAA for a k -processor system.

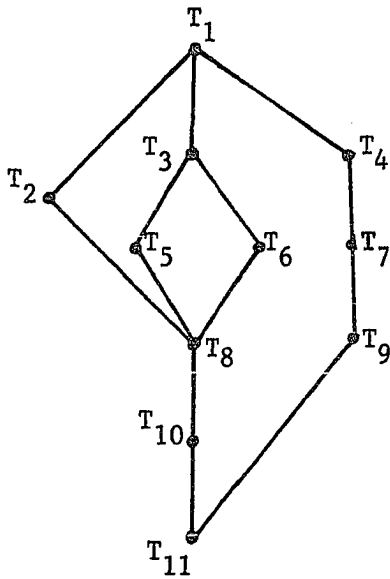
From previous discussion, we know that $t(S') \leq t(S)$ when we do not recalculate the subset sequence. Note that when we recalculate the subset sequence based on k' -processors, if the number of nodes in any subset S_i is less than k' , it is possible to "fill" S_i with ready nodes from lower levels. From Lemma 9 we know that by moving a node to S_i from a lower level we cannot increase the schedule length. Then by successive applications of Lemma 9 the resulting schedule S'' is such that $t(S'') \leq t(S')$. Therefore $t(S'') \leq t(S)$. This result gives us the following theorem.

Theorem 17. The Subset Assignment Schedule, SAS, is stable when the number of processors is increased from k to k' and the subset sequence is recomputed.

2. Relaxation of the partial order

In this part, we investigate the stability when the partial order is relaxed. Here we use the same definition of "simple relaxation" given in Section III.B. Using the SAA, we form the sequence of subsets based on a given partial order \leq . We wish to study the schedule-length

G:



$$S_1 = \{T_1\}$$

$$S_2 = \{T_2, T_3\}$$

$$S_3 = \{T_4, T_5, T_6\}$$

$$S_4 = \{T_7, T_8\}$$

$$S_5 = \{T_9, T_{10}\}$$

$$S_6 = \{T_{11}\}$$

$$t(S) = 6\frac{1}{2}$$

a) Subset sequence for G on 2 processors

$$S'_1 = \{T_1\}$$

$$S'_4 = \{T_8, T_9\}$$

$$S'_2 = \{T_2, T_3, T_4\}$$

$$S'_5 = \{T_{10}\}$$

$$S'_3 = \{T_5, T_6, T_7\}$$

$$S'_6 = \{T_{11}\}$$

$$t(S') = 6$$

b) Subset sequence for G on 3 processors

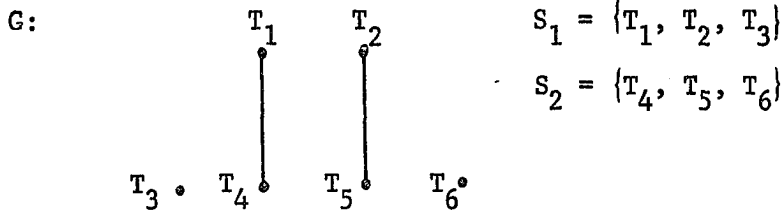
Figure 18. Typical effect of increasing the number of processors in a subset sequence

effect of moving nodes among these subsets to arrive at the subset sequence for the relaxed partial order \leq' produced by the SAA. It is, of course, assumed that \leq' is derived from \leq through a finite sequence of simple relaxations. A single simple relaxation may result in the movement of more than one node. Considering only the end result, it is possible to start with the subset sequence based on \leq produced by the SAA and arrive at the subset sequence based on \leq' produced by the SAA through a sequence of "effective moves." An "effective move" of a node between subsets S_i and S_j is defined as the move of a node n from S_i to S_j and the possible, if any, move of a node from the subset $S_j \cup \{n\}$ to the subset $S_i - \{n\}$ allowed by the SAA and caused by the first move. An effective move consists of more than one single move only in the case where $|S_i| \leq k$ and $i < j$. Any movement of a node out of S_i will be later offset by a move back into S_i . An example illustrating an effective move is provided in Figure 19.

The following lemma studies the effect on the schedule length caused by an effective move.

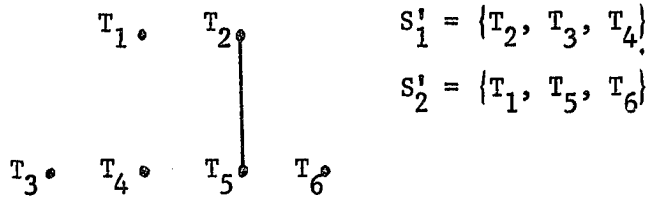
Lemma 10. Let S_j and S_i be two subsets obtained by the SAA for G_w in a k -processor system. Then, any effective move of a node between S_i and S_j required by the SAA for the new partial order will not result in an increase in the schedule length.

Proof: Define S_i' and S_j' as the subsets obtained from S_i and S_j after an "effective move" of a node from S_j to S_i . Therefore $|S_i'| = |S_i| + 1$ and $|S_j'| = |S_j| - 1$ if the effective move consists of one single move, or $|S_i'| = |S_i|$ and $|S_j'| = |S_j|$ otherwise. Let t_i , t_j , t_i' and t_j' be the time needed to execute the subsets S_i , S_j , S_i' and S_j' ,



a) Subset sequence for G on 3 processors

G': (obtained from G by removing (T_1, T_4))



b) Subset sequence for G' on 3 processors

(Note that the subset sequence for G' has been obtained from the subset sequence for G after only one effective move. The move of T_1 from S_1 to S_2 is offset by the move of T_4 from S_2 to S_1 .)

Figure 19. Example of an effective move

respectively.

We first study the effect of moving a node from S_j to S_i , $i < j$, due to a simple relaxation of the partial order. Here we do not consider the case when $2 \leq |S_j|$ and $|S_i| \geq k$ because the movement of a node from a lower-level subset S_j to a higher-level subset S_i when

$|S_i| \geq k$ is not allowed by the SAA. It is important to mention that if all the nodes in S_i and S_j are mutually independent after the simple relaxation, then we form a new subset $S_i' = S_i \cup S_j$. There are four possible cases when a node can be moved from S_j to S_i , $i < j$:

Case 1. Suppose $|S_j| > k$ and $|S_i| < k$. Then clearly $t_j' < t_j$ and $t_i' = t_i$.

Case 2. Suppose $2 \leq |S_j| \leq k$ and $|S_i| < k$. Then clearly $t_j' = t_j$ and $t_i' = t_i$.

Case 3. Suppose $|S_j| = 1$ and $|S_i| \geq k$. Then $t_j = w$, $t_i = |S_i|w/k$, $t_j' = 0$ and $t_i' = (|S_i| + 1)w/k$, therefore $t_j' + t_i' < t_j + t_i$.

Case 4. Suppose $|S_j| = 1$ and $|S_i| < k$. Then clearly $t_j' < t_j$ and $t_i' = t_i$.

Now, we study the different cases that can happen for an "effective move" of a node from S_j to S_i , $j < i$. There are three different cases:

Case 1. Suppose $|S_j| > k$ and $|S_i| \geq k$. Then $t_j = |S_j|w/k$, $t_i = |S_i|w/k$, $t_j' = (|S_j| - 1)w/k$ and $t_i' = (|S_i| + 1)w/k$, therefore $t_j' + t_i' = t_j + t_i$.

Case 2. Suppose $|S_j| > k$ and $|S_i| < k$. Then clearly $t_j' < t_j$ and $t_i' = t_i$.

Case 3. Suppose $|S_j| \leq k$. In this case, after a node n is moved from S_j to S_i , we move a node from the lower-level subset $S_i \cup \{n\}$ to "fill" the higher-level subset $S_j - \{n\}$, leaving the cardinality of both subsets unchanged. Therefore $t_i' + t_j' = t_i + t_j$.

In each of the above cases, it follows that $t_i' + t_j' \leq t_i + t_j$. Also $t_\ell' = t_\ell$ for $\ell \neq i, j$. Therefore $\sum_{\ell=1}^{L/w} t_\ell' \leq \sum_{\ell=1}^{L/w} t_\ell$.

If we recalculate the subset sequence, then by successive applications of Lemma 10, the resulting schedule S' is such that $t(S') \leq t(S)$. Therefore, we have the following theorem.

Theorem 18. The Subset Assignment Schedule, SAS, is stable under any relaxation of the partial order if we recalculate the subset sequence.

Figure 20 shows a typical effect of relaxing the partial order and recalculating the subset sequence.

3. Reduction of task execution times

It is conjectured that the Subset Assignment Schedule, SAS, is stable when the execution of one or more tasks is reduced and the subset sequence is recalculated.

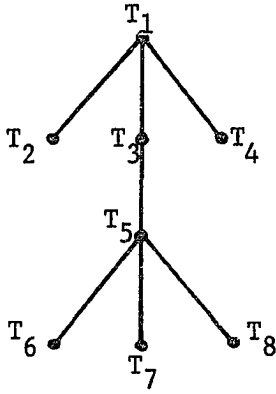
4. Combined effects

So far in this section we have established the weak stability inherent in the Subset Assignment Schedule when we increase the number of processors, or relax the precedence order, if the subset sequence is recalculated. It immediately follows that conditions on k and \leq may be collectively relaxed without increasing the schedule length.

Theorem 19. The Subset Assignment Schedule, SAS, is stable under any collective increase in the number of processors and relaxations of the partial order.

Proof: Suppose we increase the number of processors to k' forming the schedule S' . By Theorem 17, $t(S') \leq t(S)$. Next, perform the

G:



$$S_1 = \{T_1\}$$

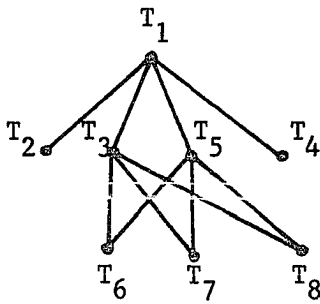
$$S_2 = \{T_2, T_3, T_4\}$$

$$S_3 = \{T_5\}$$

$$S_4 = \{T_6, T_7, T_8\}$$

$$t(S) = 4$$

a) Subset sequence for G on 3 processors

G': (obtained from G by removing (T_3, T_5))

$$S'_1 = \{T_1\}$$

$$S'_2 = \{T_2, T_3, T_5\}$$

$$S'_3 = \{T_4, T_6, T_7, T_8\}$$

$$t(S') = 3 \frac{1}{3}$$

b) Subset sequence for G' on 3 processors

Note that to obtain the subset sequence for G' from the subset sequence from G we move T_5 from S_3 to S_2 ; T_6, T_7, T_8 from S_4 to S_3 and T_4 from S_2 to S_3 .

Figure 20. Typical effect of relaxing the partial order in a subset sequence

relaxations to form \leq' and the resulting schedule S'' . By Theorem 18, $t(S'') \leq t(S')$. Therefore $t(S'') \leq t(S)$. The proof is similar for the other possible case.

We conjecture that Theorem 19 also holds if we allow additional decrease in the execution time of some tasks.

IV. STABILITY PROPERTIES OF OPTIMAL NONPREEMPTIVE SCHEDULES UNDER NONOPTIMAL CONDITIONS

In this chapter we study the stability of the schedules obtained from the A-algorithm and B-algorithm when the conditions for optimality are violated. We will apply the A-algorithm and B-algorithm to general graphs with unequally weighted tasks on a k -processor system, or to general graphs with equally weighted tasks on a k ($k > 2$) processor system. This study was motivated by the results of previous work made by Manacher (15), Ramamoorthy, et al. (20), Chandy and Dickson (3), and in particular by the simulation made by Adam, et al. (1), who found that the algorithms that assign priority depending on the task's level are near-optimal, i.e., in 90% of the cases these algorithms produced a schedule that is within 5% of the optimal execution time.

Before we report the results of this chapter, we need to generalize the labeling algorithm used to obtain the priority list, \mathcal{L} , for the A-algorithm. The new labeling algorithm is the following: Let $N = (n_1, \dots, n_t)$ and $N' = (n'_1, \dots, n'_{t'})$ denote two decreasing sequences of positive integers. Define $N < N'$ if either (a) for some i , $1 \leq i \leq t$, we have $n_j = n'_j$ for $1 \leq j \leq i - 1$ and $n_i < n'_i$, or (b) $t < t'$ and $n_j = n'_j$ for $1 \leq j \leq t$. Let n denote the number of tasks in G . The labeling algorithm assigns to each task T an integer label $\alpha(T) \in \{1, 2, \dots, n\}$. The mapping α is defined recursively as follows. Let $S(T)$ denote the set of immediate successors of T and L_i denote the set of tasks at level i in G where i is a real number in the range $0 \leq i \leq L$.

S1. Arbitrarily assign a label $\alpha(T)$ to each task $T \in L_1$ starting from 1, where the subscript i is minimal.

S2. Select the set of tasks L_j with minimal subscript j from those sets which have not been assigned labels.

S3. Suppose that $(k - 1)$ tasks have been assigned a label. For each task T in L_j , let $N(T)$ denote the decreasing sequence of integers formed by ordering the set $\{\alpha(T') \mid T' \in S(T)\}$. At least one of these tasks T^* must satisfy $N(T^*) \leq N(T)$ for all such tasks T . Choose one such T^* and define $\alpha(T^*)$ to be k .

S4. We repeat the assignment in S3 until all tasks in L_j have been assigned some integer.

S5. Repeat steps S2, S3 and S4 until all tasks are labeled.

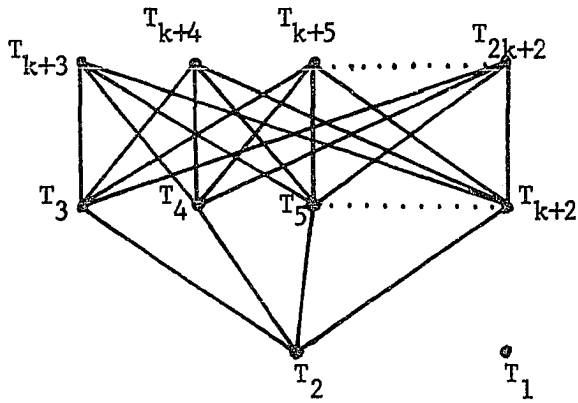
All the examples presented in this chapter show that the A-schedules and B-schedules are unstable when the conditions for optimality are violated. In order to have stability, we have to add Manacher's conditions (15).

Example 1. The example in Figure 21 shows an anomaly due to a decrease in the execution time when the A-algorithm or the B-algorithm is applied to general graphs with equally weighted tasks on a k -processor system.

In this example the schedule length is increased by $1/2$ unit if $\tau_{2k+2} = 1/2$ instead of 1 unit.

Example 2. Figure 22 shows an example of an anomaly due to a decrease in the execution time when the A-algorithm or the B-algorithm is applied to general graphs with unequally weighted tasks on a

G:



$$\tau_i = 1, \forall i$$

S:

	t=0	1	2	3
P ₁	T _{2k+2}	T _{k+2}	T ₂	
P ₂	T _{2k+1}	T _{k+1}	T ₁	
P ₃	T _{2k}	T _k		
...				
P _k	T _{k+3}	T ₃		

$$t(S) = 3$$

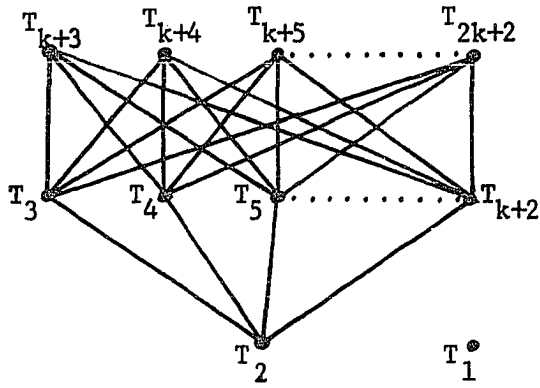
Now if $\tau_{2k+2} = \frac{1}{2}$

	t=0	1	2	3
P ₁	$\frac{1}{2}T_{2k+2}$	T ₁	T ₃	T ₂
P ₂	T _{2k+1}	T _{k+2}		
P ₃	T _{2k}	T _{k+1}		
...				
P _k	T _{k+3}	T ₄		

$$t(S') = 3\frac{1}{2}$$

Figure 21. A system with equally weighted tasks that degrades with the reduction of execution time of one task

G:



$$\tau_2 = 4$$

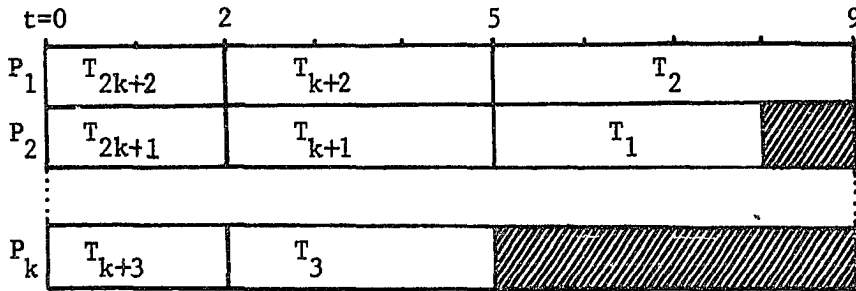
$$\tau_i = 3$$

for $i = 1$ and $3 \leq i \leq k+2$.

$$\tau_i = 2$$

for $k+3 \leq i \leq 2k+2$.

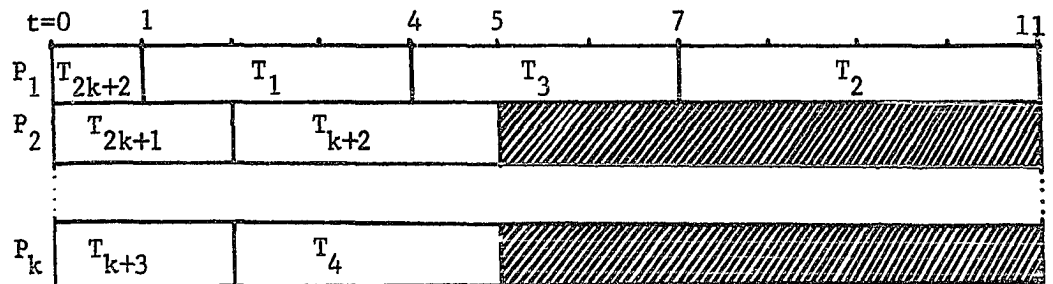
S:



$$t(S) = 9$$

Now if $\tau_{2k+2} = 1$

S':



$$t(S') = 11$$

Figure 22. A system with unequally weighted tasks that degrades with the reduction of execution time of one task

k-processor system. We see that reducing τ_{2k+2} by 1 unit, the schedule length is increased by 2 units.

An anomaly due to increasing the number of available processors is the subject of the next two examples. In these examples we present the anomaly when k is increased from 7 to 8.

Example 3. Figure 23 shows an example of an anomaly due to increasing the number of available processors when the B-algorithm is applied to a general graph with unequally weighted tasks.

Example 4. Figure 24 shows an example (12, 13) of an anomaly due to increasing the number of available processors when the B-algorithm is applied to a general graph with equally weighted tasks.

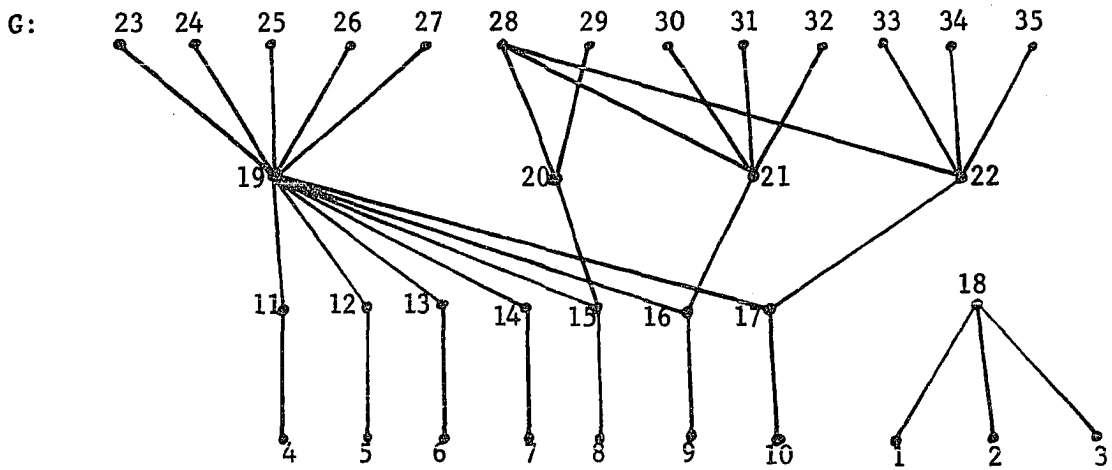
In the next example we present an anomaly due to relaxing the partial order, \leq .

Example 5. The example in Figure 25 shows an anomaly due to relaxing some precedence relations when the A-algorithm or the B-algorithm is applied to general graphs with unequally weighted tasks.

We conjecture the following:

1) The B-schedule for a tree-structured graph with unequally weighted tasks and arbitrary number of processors is stable under any collective increase in the number of processors, allowable relaxations of the partial order and a decrease in the execution time of some tasks.

2) The A-schedule for a general graph with equally weighted tasks and arbitrary number of processors is stable under any collective increase in the number of processors and relaxations of the partial order.



$$\tau_i = 1, v_i (i \neq 18)$$

$$\tau_{18} = 1\frac{1}{2}$$

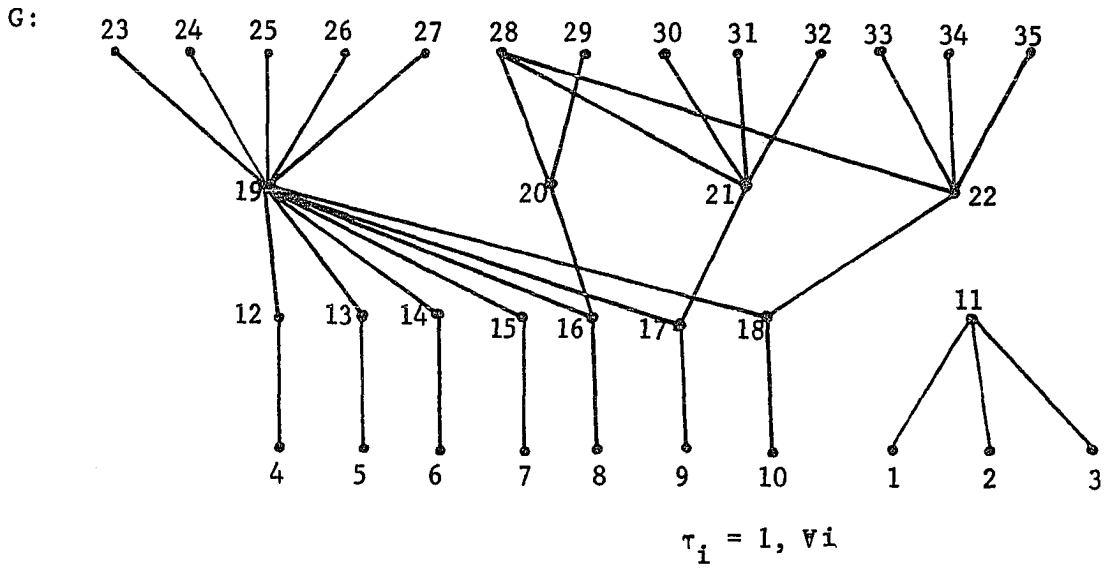
S:

	t=0	1	2	3	4	5	5½
P ₁	35	28	22	17	10		
P ₂	34	27	21	16	9		
P ₃	33	26	20	15	8		
P ₄	32	25	19	14	7		
P ₅	31	24		3	13	6	
P ₆	30	23		2	12	5	
P ₇	29	18		1	11	4	

S':

	t=0	1	2	3	4	5	6
P ₁	35	27	19	17	10	1	
P ₂	34	26	18		3	2	
P ₃	33	25		16	9		
P ₄	32	24		15	8		
P ₅	31	23		14	7		
P ₆	30	22		13	6		
P ₇	29	21		12	5		
P ₈	28	20		11	4		

Figure 23. A system with unequally weighted tasks that degrades with increasing k (B-schedule)



S:

	t=0	1	2	3	4	5
P ₁	35	28	22	18	10	
P ₂	34	27	21	17	9	
P ₃	33	26	20	16	8	
P ₄	32	25	19	15	7	
P ₅	31	24	3	14	6	
P ₆	30	23	2	13	5	
P ₇	29	11	1	12	4	

$t(S) = 5$

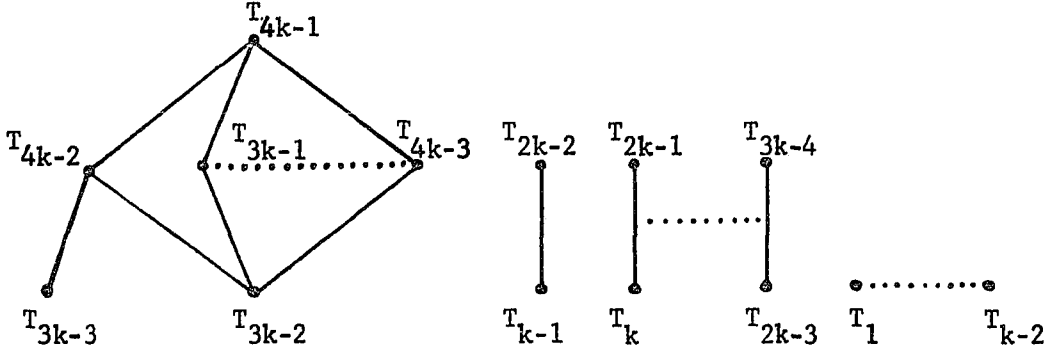
S':

	t=0	1	2	3	4	5	6
P ₁	35	27	19	18	10	1	
P ₂	34	26	11	17	9		
P ₃	33	25		16	8		
P ₄	32	24		15	7		
P ₅	31	23		14	6		
P ₆	30	22		13	5		
P ₇	29	21		12	4		
P ₈	28	20		3	2		

$t(S') = 6$

Figure 24. A system with equally weighted tasks that degrades with increasing k (B-schedule)

G:



$$\tau_i = 1 \text{ for } 1 \leq i \leq 3k-4$$

$$\text{and } 3k-1 \leq i \leq 4k-2.$$

$$\tau_{3k-2} = \tau_{4k-1} = 3$$

$$\tau_{3k-3} = 2\frac{1}{2}$$

S:

	t=0	1	2	3	4	5	6	7	
P ₁	T _{4k-1}			T _{4k-2}	T _{3k-2}				t(S) = 7
P ₂	T _{3k-4}	T _{2k-3}	T _{k-2}	T _{4k-3}	T _{3k-3}				
P ₃	T _{3k-5}	T _{2k-4}	T _{k-3}	T _{4k-4}					
⋮									
P _{k-1}	T _{2k-1}	T _k	T ₁	T _{3k}					
P _k	T _{2k-2}	T _{k-1}		T _{3k-1}					

If we relaxed the precedence relations $T_{4k-1} < T_{3k-3}$ and $T_{4k-2} < T_{3k-3}$,

we have S':

S' :	$t=0$	1	2	3	4	5	6	7			
P_1	T_{4k-1}			T_{4k-2}	T_1				$t(S') = 7\frac{1}{2}$		
P_2	T_{3k-3}			T_2	T_{3k-1}					T_{3k-2}	
P_3	T_{3k-4}	T_{2k-2}	T_k	T_{4k-3}							
\vdots											
P_{k-1}	T_{2k}	T_{k+2}	T_4	T_{3k+1}							
P_k	T_{2k-1}	T_{k+1}	T_3	T_{3k}							

Figure 25. A system that degrades when the partial order is relaxed

Assuming the conjectures are true, the B-schedule provides an upper bound on the schedule length for task systems with tree-structured graphs. For general graphs, we have obtained a negative result because the A-schedule is not stable when the execution time of some tasks is decreased. We can only say that the A-schedule is stable if we first allow any collective increase in the number of processors and relaxation of the partial order, and then, we add Manacher's precedence constraints and use the projective task list, PTL, in order to prevent the third type of anomaly. Recall that adding Manacher's precedence constraints and using the PTL we cannot have anomalies due to the increase in the number of processors and reduction of the execution time of some tasks (15).

V. LOWER BOUNDS ON THE RATIO CT'_{PS}/CT'_{BS}

In this chapter, we will give some bounds on the ratio between the optimal preemptive schedule and the optimal basic schedule when we take in consideration the cost of switching and the cost of preemption. Because we can construct optimal schedules with excessively many preemptions (thereby driving the preemption cost unfairly high), we will confine our attention to those optimal preemptive schedules which have a minimum number of preemptions. The first part of this chapter deals with general precedence graphs G , $k \geq 2$ processors and unequally weighted tasks, while the second part deals with equally weighted tasks. We assume that switching times are positive and constant. While this is a simplification of the real case (7), it does provide a first order approximation to the effect of processor switching on the schedule length.

In order to obtain the bounds we will introduce some notation. Let $\epsilon_1, \epsilon_2, \epsilon_3$ and ϵ_4 be the cost of starting a task for the first time, stopping a task permanently, starting a task after preemption and stopping a task temporarily, respectively. Let CT_{BS} and CT_{PS} be the length of the optimal basic and preemptive schedule, when we consider $\epsilon_j = 0$ for all j . Let CT'_{BS} and CT'_{PS} be the length of the optimal basic and preemptive schedule, when we consider $\epsilon_j \neq 0$ for any j .

It is important to note that the scheduling is done with the assumption that $\epsilon_j = 0$ for all j . Using this schedule, we investigate the execution time effects of $\epsilon_j \neq 0$, for any j . If we do not proceed

in this way, then the weight of the tasks including the cost of preemption, becomes schedule dependent and this class of problems is more difficult to solve.

The following definitions apply to a schedule with zero preemption and switching costs.

Define:

$N_B^{P_i}$ - the number of tasks in the optimal basic schedule that are performed by processor P_i .

$N_1^{P_i}$ - the number of tasks in the optimal PS that are started for first time in P_i .

$N_2^{P_i}$ - the number of tasks in the optimal PS that are stopped permanently in P_i .

$N_3^{P_i}$ - the number of tasks in the optimal PS that are started after preemption in P_i .

$N_4^{P_i}$ - the number of tasks in the optimal PS that are stopped temporarily in P_i .

$|G|$ - the number of tasks in the graph G .

$N_B = \max_i (N_B^{P_i})$ where P_i is any processor that finishes last.

$N_B^* = \max_i (N_B^{P_i})$ for all i .

$N_j = \min_i (N_j^{P_i})$ where P_i is any processor that finishes last and $1 \leq j \leq 4$.

$\{N_j + N_{j+2}\} = \min_i (N_j^{P_i} + N_{j+2}^{P_i})$ where P_i is any processor that finishes last and $j = 1, 2$.

$N_j^* = \max_i (N_j^{P_i})$ for $1 \leq j \leq 4$ and all i .

We can write

$$CT_{PS} + \sum_{i=1}^4 N_i^* \epsilon_i \geq CT'_{PS} \geq CT_{PS} + \sum_{i=1}^4 N_i \epsilon_i$$

and

$$CT_{BS} + N_B^*(\epsilon_1 + \epsilon_2) \geq CT'_{BS} \geq CT_{BS} + N_B(\epsilon_1 + \epsilon_2)$$

The notation used in this chapter is illustrated in Figure 26.

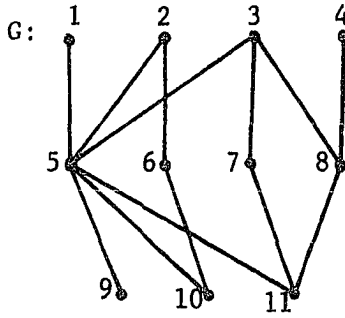
In this chapter, we will find only lower bounds on the ratio CT'_{PS}/CT'_{BS} for any task system. Based on experimentation and apparent claims in the literature (3), it appears possible to place an upper bound on the number of preemptions in any processor for an optimal schedule which has a minimum total number of preemptions. This knowledge would allow us to derive important upper bounds on the ratio CT'_{PS}/CT'_{BS} . This derivation is not presented here because the author was not able to verify the upper bound on the number of preemptions. However, using a different approach, in the next chapter we will find upper and tighter lower bounds on the ratio CT'_{PS}/CT'_{BS} for equally weighted tasks on a two-processor system.

A. Unequally Weighted Tasks

In this section we will find a lower bound on the ratio CT'_{PS}/CT'_{BS} for any general graph G with unequally weighted tasks on $k \geq 2$ processors. For our purpose, the following bounds are needed:

$$\left. \begin{matrix} N_1 \\ N_2 \end{matrix} \right\} \geq 1 \quad (1)$$

$$\left. \begin{matrix} N_3 \\ N_4 \end{matrix} \right\} \geq 0 \quad (2)$$



$$\tau_1 = 3/2$$

$$\tau_2 = \tau_3 = \tau_5 = \tau_8 = \tau_{10} = \tau_{11} = 1$$

$$\tau_4 = \tau_6 = 2$$

$$\tau_7 = 3$$

$$\tau_9 = 7/2$$

	t=0	1	2	3	4	5	6
S:	P ₁	1	5	7	11		
	P ₂	2	4	9			
	P ₃	3	6	8	10		

$$N_B^{P_1} = N_B^{P_3} = 4 \quad N_B^{P_2} = 3 \quad N_B = 4 \quad N_B^* = 4$$

$$CT'_{BS} = 6.5 + 4(\epsilon_1 + \epsilon_2)$$

a) Optimal basic schedule for G

	t=0	1	2	3	4	5	6
S':	P ₁	2	4	6	8	10	
	P ₂	4	3	7	6	11	
	P ₃	1	5	9			

$$\begin{array}{lll}
 N_1^{P_1} = 4 & N_1^{P_2} = 4 & N_1^{P_3} = 3 \\
 N_2^{P_1} = 4 & N_2^{P_2} = 4 & N_2^{P_3} = 3 \\
 N_3^{P_1} = 1 & N_3^{P_2} = 1 & N_3^{P_3} = 0 \\
 N_4^{P_1} = 1 & N_4^{P_2} = 1 & N_4^{P_3} = 0
 \end{array}
 \quad
 \begin{array}{ll}
 N_1 = 3 & N_1^* = 4 \\
 N_2 = 3 & N_2^* = 4 \\
 N_3 = 0 & N_3^* = 1 \\
 N_4 = 0 & N_4^* = 1
 \end{array}$$

$$\{N_1 + N_3\} = 3$$

$$\{N_2 + N_4\} = 3$$

$$CT'_{PS} = 6 + 4(\epsilon_1 + \epsilon_2) + \epsilon_3 + \epsilon_4$$

b) Optimal preemptive schedule for G

Figure 26. Illustration of the notation used in Chapter V

Relations 1 and 2 come from the fact that the lower bound for the number of tasks executed by any processor that finishes last is one, and the fact that we might not need to preempt any task in order to obtain the optimal schedule.

Define G' as the normalized graph obtained from G by considering $\tau'_i = \tau_i / \min\{\tau_i\}$, for all i in G . Therefore our unit of time in G' is $\min\{\tau_i\}$. We also assume that the ϵ_i 's are expressed in the same normalized unit of time. In this section, we only deal with normalized graphs and in order to facilitate the notation, in the future, we will use G and τ_i to represent the normalized graph and the normalized execution time of the tasks. For the normalized graph G , the following relations are valid: $CT_{BS} \geq N_B^* \geq N_B \geq 1$ and $\Sigma \tau_i \geq CT_{BS} \geq 1$.

The following theorem gives a lower bound on the ratio CT'_{PS}/CT'_{BS} for any normalized graph G with unequally weighted tasks on a k -processor system.

Theorem 20. For any normalized graph G with unequally weighted tasks on a k -processor system, we have

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(\Sigma \tau_i + 2\epsilon_1 + 2\epsilon_2) + \Sigma \tau_i}{2k(1 + \epsilon_1 + \epsilon_2)\Sigma \tau_i}.$$

Proof: We know that

$$CT'_{PS} \geq CT_{PS} + N_1\epsilon_1 + N_2\epsilon_2 + N_3\epsilon_3 + N_4\epsilon_4$$

and

$$CT_{BS} + N_B^*(\epsilon_1 + \epsilon_2) \geq CT'_{BS}.$$

It is also known from (14) that $CT_{PS} \geq \gamma CT_{BS}$ where $\gamma = (k + 1)/(2k)$.

Inserting the first two inequalities in the last one we obtain

$$CT'_{PS} - N_1\epsilon_1 - N_2\epsilon_2 - N_3\epsilon_3 - N_4\epsilon_4 \geq \gamma CT'_{BS} - \gamma N_B^* \epsilon_1 - \gamma N_B^* \epsilon_2$$

or

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \gamma + \frac{(N_1 - \gamma N_B^*)\epsilon_1}{CT'_{BS}} + \frac{(N_2 - \gamma N_B^*)\epsilon_2}{CT'_{BS}} + \frac{N_3\epsilon_3}{CT'_{BS}} + \frac{N_4\epsilon_4}{CT'_{BS}}.$$

Using Relations 1 and 2 and considering that $CT_{BS} + N_B^*(\epsilon_1 + \epsilon_2) \geq CT'_{BS}$,

we have

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \gamma + \frac{(1 - \gamma N_B^*)(\epsilon_1 + \epsilon_2)}{CT_{BS} + N_B^*(\epsilon_1 + \epsilon_2)}$$

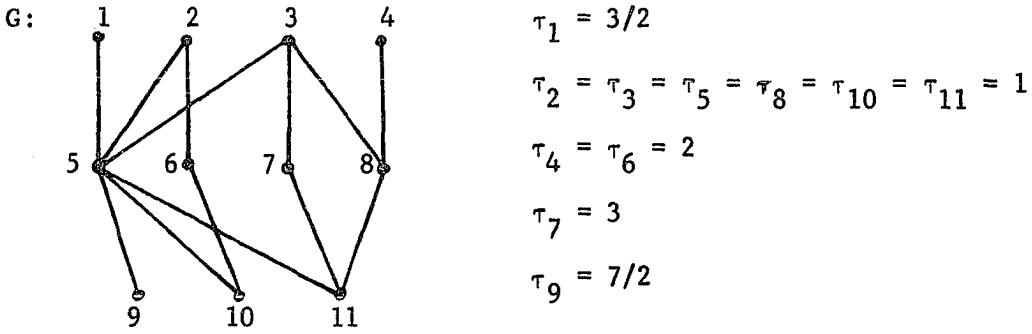
It is easy to check that the expression at the right-hand side of this inequality is monotonically decreasing with CT_{BS} and N_B^* . Therefore, the minimum is obtained when both CT_{BS} and N_B^* achieve their maximum value. Then

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \gamma + \frac{(1 - \gamma \Sigma \tau_i)(\epsilon_1 + \epsilon_2)}{(1 + \epsilon_1 + \epsilon_2)\Sigma \tau_i}$$

or

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(\Sigma \tau_i + 2\epsilon_1 + 2\epsilon_2) + \Sigma \tau_i}{2k(1 + \epsilon_1 + \epsilon_2)\Sigma \tau_i}.$$

Figure 27 shows an example of a task system with unequally weighted tasks where we calculate the lower bound and the actual ratio for a given set of values of ϵ_i . Note that the new schedule may have idle time necessary for a task awaiting the completion of the preemption or switching time of any of its predecessors. In addition, we assume deliberate insertion of idle time to keep the same scheduling of tasks used when we considered the preemptive costs negligible. In the schedule S' given in Figure 27, we insert idle time after T_1 finishes



S:

P ₁	ϵ_1	1	ϵ_2	ϵ_1	5	ϵ_2	ϵ_1	7	ϵ_2	ϵ_1	11	ϵ_2
P ₂	ϵ_1	2	ϵ_2	ϵ_1	4	ϵ_2	ϵ_1	9	ϵ_2			
P ₃	ϵ_1	3	ϵ_2	ϵ_1	6	ϵ_2	ϵ_1	8	ϵ_2			

$$CT'_{BS} = 6.5 + 4(\epsilon_1 + \epsilon_2)$$

a) Basic schedule with $\epsilon_1 \neq 0$

S' :

P ₁	ϵ_1	2	ϵ_2	ϵ_3	4	ϵ_2	ϵ_1	6	ϵ_4	ϵ_1	8	ϵ_2	ϵ_1	10	ϵ_2
P ₂	ϵ_1	4	ϵ_4	ϵ_1	3	ϵ_2	ϵ_1	7	ϵ_2	ϵ_3	6	ϵ_2	ϵ_1	11	ϵ_2
P ₃	ϵ_1	1	ϵ_2		ϵ_1	5	ϵ_2	ϵ_1	9	ϵ_2					

$$CT'_{PS} = 6 + 4(\epsilon_1 + \epsilon_2) + \epsilon_3 + \epsilon_4$$

b) Preemptive schedule with $\epsilon_1 \neq 0$

If $\epsilon_1 = .02$, $\epsilon_2 = .1$, $\epsilon_3 = .01$ and $\epsilon_4 = .05$, we have:

$$\text{Actual ratio: } CT'_{PS}/CT'_{BS} = 6.54/6.98 = 0.937$$

$$\text{Lower bound: } 0.601$$

Figure 27. Lower bound for a task system with unequally weighted tasks

and we do not assign the ready task T_6 to P_3 in order not to change the original schedule S' given in Figure 26.

B. Equally Weighted Tasks

In this section, we will find lower bounds on the ratio CT'_{PS}/CT'_{BS} for any general graph G with equally weighted tasks on $k \geq 2$ processors. We are interested in these systems because most of the literature deals with unit-task systems. For our purposes, the following bounds are needed:

$$\left. \begin{matrix} N_1 \\ N_2 \end{matrix} \right\} \geq 1 \quad (1)$$

$$\left. \begin{matrix} N_3 \\ N_4 \end{matrix} \right\} \geq 0 \quad (2)$$

$$\left\{ \begin{matrix} N_1 + N_3 \\ N_2 + N_4 \end{matrix} \right\} \geq N_B \quad (3)$$

In this case, we do not need to normalize G because all the tasks have the same execution time. Without loss of generality, consider $\tau_i = 1$ for all i in G . Also, the following relations are valid for any basic schedule on unit-task systems

$$CT_{BS} = N_B, \quad |G| \geq CT_{BS} \geq 1$$

and

$$CT'_{BS} = CT_{BS} + N_B(\epsilon_1 + \epsilon_2) = N_B(1 + \epsilon_1 + \epsilon_2).$$

Relation 3 comes from the fact that on unit-task systems in order to improve any basic schedule we can preempt some tasks and as a result

the number of times that tasks are started or stopped in P_1 will increase.

The following theorem gives a lower bound on the ratio CT'_{PS}/CT'_{BS} for the case of unequal switching and preemption costs.

Theorem 21. For any graph G with equally weighted tasks on a k -processor system,

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(CT_{BS} + 2\epsilon_1 + 2\epsilon_2) + CT_{BS}}{2kCT_{BS}(1 + \epsilon_1 + \epsilon_2)}$$

Proof: For equally weighted tasking systems, we have

$$CT'_{PS} \geq CT_{PS} + N_1\epsilon_1 + N_2\epsilon_2 + N_3\epsilon_3 + N_4\epsilon_4$$

and

$$CT'_{BS} = CT_{BS} + N_B(\epsilon_1 + \epsilon_2) = CT_{BS}(1 + \epsilon_1 + \epsilon_2).$$

It is also known from (14) that $CT_{PS} \geq \gamma CT_{BS}$ where $\gamma = (k + 1)/(2k)$.

Inserting the first two inequalities in the last one, we obtain

$$CT'_{PS} - N_1\epsilon_1 - N_2\epsilon_2 - N_3\epsilon_3 - N_4\epsilon_4 \geq \gamma CT'_{BS} - \gamma N_B\epsilon_1 - \gamma N_B\epsilon_2 \quad (4)$$

or

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \gamma + \frac{(N_1 - \gamma N_B)\epsilon_1}{CT'_{BS}} + \frac{(N_2 - \gamma N_B)\epsilon_2}{CT'_{BS}} + \frac{N_3\epsilon_3}{CT'_{BS}} + \frac{N_4\epsilon_4}{CT'_{BS}}.$$

Using Relations 1 and 2 and $CT'_{BS} = CT_{BS}(1 + \epsilon_1 + \epsilon_2)$ in the last inequality, we have

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \gamma + \frac{(1 - \gamma CT_{BS})(\epsilon_1 + \epsilon_2)}{CT_{BS}(1 + \epsilon_1 + \epsilon_2)}$$

But $\gamma = (k + 1)/(2k)$, therefore

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(CT_{BS} + 2\epsilon_1 + 2\epsilon_2) + CT_{BS}}{2k CT_{BS} (1 + \epsilon_1 + \epsilon_2)}.$$

Corollary: For any graph G with equally weighted tasks on a k -processor system,

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(|G| + 2\epsilon_1 + 2\epsilon_2) + |G|}{2k|G|(1 + \epsilon_1 + \epsilon_2)}$$

The proof is immediate if we note that the expression at the right-hand side of the inequality given in Theorem 21 is monotonically decreasing with CT_{BS} . Therefore, the minimum will be achieved when CT_{BS} is equal to $|G|$. Replacing CT_{BS} by $|G|$ we obtain the final result.

The next theorem gives a lower bound on the ratio CT'_{PS}/CT'_{BS} when all the ϵ_i are equal. We are interested in this case because the bound is easier to calculate.

Theorem 22. For any graph G with equally weighted tasks on a k -processor system with $\epsilon_i = \epsilon$ for all i ,

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(1 + 4\epsilon) + 1}{2k(1 + 2\epsilon)}$$

Moreover, this is the best possible bound.

Proof: In this case the Inequality 4 can be written as

$$CT'_{PS} \geq \gamma CT'_{BS} + (N_1 + N_2 + N_3 + N_4 - 2\gamma N_B)\epsilon.$$

Using Relation 3 and $CT'_{BS} = N_B(1 + 2\epsilon)$, we have

$$CT'_{PS} \geq \gamma CT'_{BS} + 2(1 - \gamma)N_B\epsilon$$

or

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \gamma + \frac{2(1-\gamma)\epsilon}{1+2\epsilon} = \frac{k+1}{2k} + \frac{2(k-1)\epsilon}{2k(1+2\epsilon)}.$$

Therefore,

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{k(1+4\epsilon) + 1}{2k(1+2\epsilon)}.$$

We can show that this is in fact the best possible bound by an example. Figure 28 shows a graph for which the best nonpreemptive schedule, S , and preemptive schedule, S' , have length $2 + 4\epsilon$ and $(k+1)/k + 4\epsilon$ respectively.

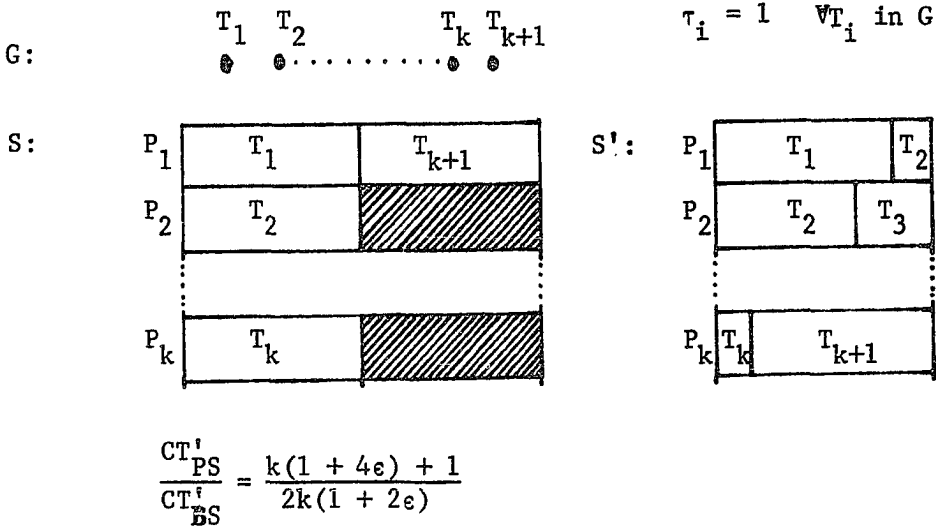


Figure 28. Example which achieves the lower bound on CT'_{PS}/CT'_{BS} when $\epsilon_i = \epsilon$.

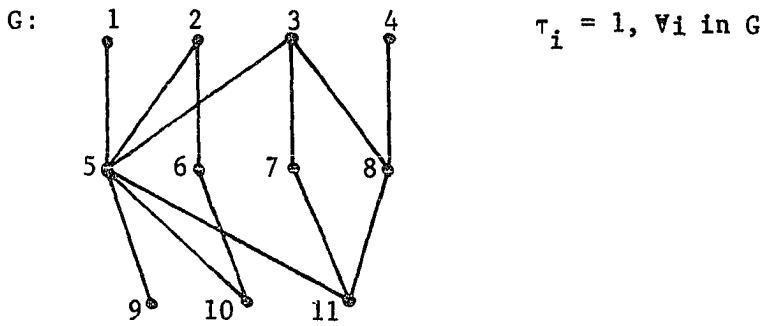
Table 1 gives the lower bounds on the ratio CT'_{PS}/CT'_{BS} for any graph G with equally weighted tasks on a k -processor system for other possible relations among the ϵ_i 's. These bounds, derived in (23), are tighter than the bound given in Theorem 21. If we replace CT_{BS} by $|G|$ in the bounds given in Table 1, we obtain new and easily calculated, but looser, bounds.

Table 1. Lower bounds on the ratio CT_{PS}'/CT_{BS}' for any graph G with equally weighted tasks

Case	Lower bound
$\epsilon_1 \geq \epsilon_3$ $\epsilon_2 \geq \epsilon_4$	$\frac{CT_{BS}'(k(1 + 2\epsilon_3 + 2\epsilon_4) + 1) + 2k(\epsilon_1 + \epsilon_2 - \epsilon_3 - \epsilon_4)}{2kCT_{BS}'(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_1 \geq \epsilon_3$ $\epsilon_4 \geq \epsilon_2$	$\frac{CT_{BS}'(k(1 + 2\epsilon_2 + 2\epsilon_3) + 1) + 2k(\epsilon_1 - \epsilon_3)}{2kCT_{BS}'(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_2 \geq \epsilon_4$	$\frac{CT_{BS}'(k(1 + 2\epsilon_1 + 2\epsilon_4) + 1) + 2k(\epsilon_2 - \epsilon_4)}{2kCT_{BS}'(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_4 \geq \epsilon_2$	$\frac{k(1 + 2\epsilon_1 + 2\epsilon_2) + 1}{2k(1 + \epsilon_1 + \epsilon_2)}$

In Figure 29, we give an example of a task system with equally weighted tasks where we calculate the lower bounds and the actual ratio on CT_{PS}'/CT_{BS}' for a given set of values of the ϵ_i .

While we have produced lower bounds for both unequally weighted and equally weighted task systems, it does not seem feasible to establish experimentally the behavior of the ratio CT_{PS}'/CT_{BS}' due to the lack of polynomial bounded algorithms to produce the optimal schedules. However, the bounds provided in this chapter allow us to calculate easily a best value of the optimal preemptive schedule length relative to the optimal basic schedule length when switching and preemption costs are positive. Given an estimate of switching and preemption



t=0	1	2	3	4
S': P ₁	1	4	7	10
P ₂	2	5	8	11
P ₃	3	6	9	

a) Optimal basic schedule with $\epsilon_i = 0$

t=0	1	2	3	
S': P ₁	1	2	5	6
P ₂	2	3	6	7
P ₃	3	4	7	8

b) Optimal preemptive schedule with $\epsilon_i = 0$

S'': P ₁	ϵ_1	1	ϵ_2	ϵ_1	4	ϵ_2	ϵ_1	7	ϵ_2	ϵ_1	10	ϵ_2
P ₂	ϵ_1	2	ϵ_2	ϵ_1	5	ϵ_2	ϵ_1	8	ϵ_2	ϵ_1	11	ϵ_2
P ₃	ϵ_1	3	ϵ_2	ϵ_1	6	ϵ_2	ϵ_1	9	ϵ_2			

$$CT_{BS}^* = 4(1 + \epsilon_1 + \epsilon_2)$$

c) Basic schedule with $\epsilon_i \neq 0$

Figure 29. Lower bounds for a task system with equally weighted tasks

$S''' :$

P_1	ϵ_1	1	ϵ_2	ϵ_3	2	ϵ_2	ϵ_1	5	ϵ_2	ϵ_3	6	ϵ_2	ϵ_1	9	ϵ_2
P_2	ϵ_1	2	ϵ_4	ϵ_3	3	ϵ_2	ϵ_1	6	ϵ_4	ϵ_3	7	ϵ_2	ϵ_1	10	ϵ_2
P_3	ϵ_1	3	ϵ_4	ϵ_1	4	ϵ_2	ϵ_1	7	ϵ_4	ϵ_1	8	ϵ_2	ϵ_1	11	ϵ_2

$$CT'_{PS} = \frac{11}{3} + \max\{3\epsilon_1 + 5\epsilon_2 + 2\epsilon_3, 3(\epsilon_1 + \epsilon_2) + 2(\epsilon_3 + \epsilon_4), 5\epsilon_1 + 3\epsilon_2 + 2\epsilon_4\}$$

d) Preemptive schedule with $\epsilon_1 \neq 0$

If $\epsilon_1 = .02$, $\epsilon_2 = .1$, $\epsilon_3 = .01$ and $\epsilon_4 = .05$, we have:

$$\text{Actual ratio: } CT'_{PS}/CT'_{BS} = 4.247/4.48 = 0.948$$

Lower bound:

$$\text{General formula: } 0.622$$

$$\text{Formula for } \epsilon_1 \geq \epsilon_3 \text{ and } \epsilon_2 \geq \epsilon_4: 0.662$$

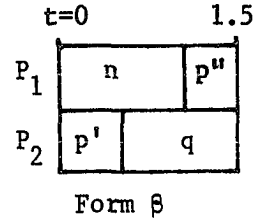
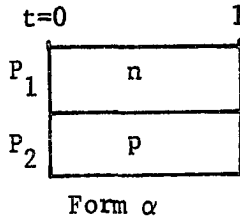
Figure 29. Continued

costs, we now have some basis for deciding whether or not to use a preemptive discipline.

VI. BOUNDS ON THE RATIO CT'_{SSAS}/CT'_{BS}

In this chapter, we will study in depth the subset assignment concept. In particular, we wish to find ratios between the shortest subset assignment schedule, SSAS, and the optimal basic schedule for general graphs with equally weighted nodes on a k -processor system when the cost of switching and the cost of preemption are not negligible. It is important to the reader to continually bear in mind that unit tasks are assumed throughout this chapter. We define a SSAS as the shortest possible preemptive schedule over all possible subset sequences. Our study is motivated by the probability that SAS, described in Chapter III, is a good heuristic for the SSAS, just as level algorithms are good heuristics for optimal nonpreemptive schedules (1). In fact the SSAS could be considered a heuristic for the optimal preemptive schedule. Rather than determining how good the heuristic is, we study only the effects of preemption costs. As a by-product, we will find sharper lower bounds on the ratio CT'_{PS}/CT'_{BS} for $k = 2$ than was provided by the analysis in Chapter V. Before we describe our model, we will give some lemmas and a theorem that are necessary for our study.

Lemma 11. Let G be an arbitrary graph all of whose nodes have unit weight. Then any subset assignment for G using two processors can be transformed into a new assignment which is no longer than the first and is constructed from forms of type α or β , where n , p , and q are nodes of G .



The proof of this lemma was given in (17).

Lemma 12. Let G be an arbitrary graph all of whose nodes have unit weight. Then any subset assignment for G using k -processors can be transformed into a new assignment which is no longer than the first and is constructed from the three forms shown in Figure 30, where m , n , o , p , q , r , s and t are nodes of G , $1 \leq u_1 < k$ and $1 < u_2 < k$.

Proof: We know that in a subset assignment each subset of the subset sequence is scheduled in an optimal way and independently of the other subsets. Depending on the number of nodes in a subset S_i , $|S_i|$, we have three different cases:

Case 1. $k \geq |S_i| \geq 1$. In this case, it is clear that $t_{\min}(S_i) = 1$ must be a lower bound since no schedule can terminate in less time than it takes to execute any task. Therefore, the optimal preemptive schedule for S_i will be an assignment of Form α .

Case 2. $2k > |S_i| > k$. In this case, by McNaughton's result (16), we know that $t_{\min}(S_i) = |S_i|/k$, since a schedule cannot be more efficient than to keep all the processors busy. Therefore, the optimal preemptive schedule for S_i will be an assignment of Form β or γ , depending on the number of nodes in S_i .

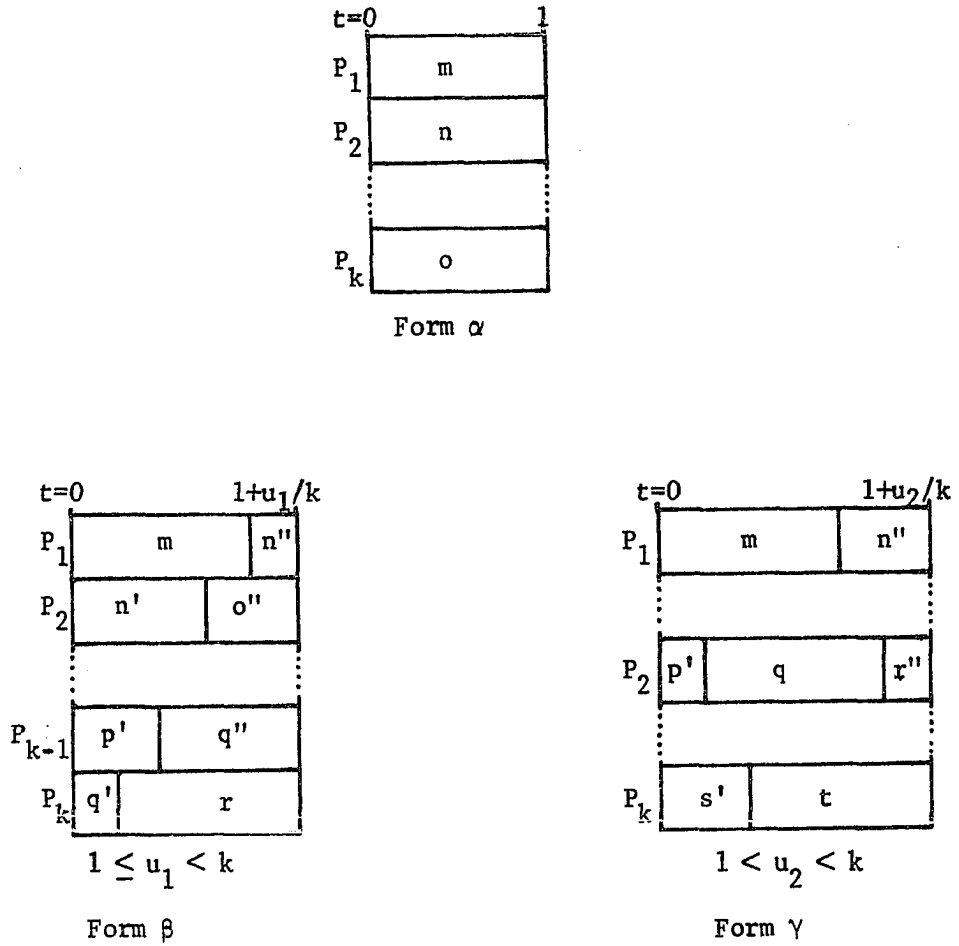


Figure 30. Forms used to construct any subset assignment

Case 3. $|S_i| \geq 2k$. In this case, we know that $t_{\min}(S_i) = |S_i|/k$, since a schedule cannot be more efficient than to keep all the processors busy. Now, we split S_i in two subsets S_i' and S_i'' such that

$$|S'_i| = \left\{ \left\lfloor \frac{|S_i|}{k} \right\rfloor - 1 + j \right\} k$$

where $j = \begin{cases} 1 & \text{if } |S_i|/k \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}$

$$|S''_i| = |S_i| - |S'_i|.$$

Because $|S'_i|$ is a multiple of the number of processors available, k , we will have assignments of Form α for the optimal preemptive schedule of S'_i . It is also easy to check that the number of nodes in S''_i is either zero or $k < |S''_i| < 2k$. When $|S''_i| = 0$ we do not care, because we do not have tasks to assign but when $k < |S''_i| < 2k$ we know, by Case 2, that the optimal preemptive schedule for S''_i will be an assignment of Form β or γ depending on the number of nodes in S''_i .

If $k < |S_i| < 2k$, we can determine if the assignment of S_i is of Form β or of Form γ by evaluating the expression

$$l = \frac{|S_i| - k}{\text{g.c.d.}(|S_i|, k)} - 1.$$

If l is equal to zero, then the schedule of S_i will be of Form β , otherwise it will be of Form γ .

In order to obtain the bounds, we make use of the following notation. Let ϵ_j ($1 \leq j \leq 4$), CT'_{PS} , CT_{PS} , CT'_{BS} and CT_{BS} have the same definition given in Chapter V. Let CT_{SSAS} be the length of the shortest possible preemptive schedule over all possible subset sequences, when we consider $\epsilon_j = 0$ for all j . Let CT'_{SSAS} be the length of the shortest possible preemptive schedule over all possible subset sequences, when we consider $\epsilon_j \neq 0$ for any j .

It is important to recall that the scheduling is done with the assumption that $\epsilon_j = 0$, for all j . Using this schedule, we investigate the execution time effects of $\epsilon_j \neq 0$ for any j . If we do not proceed in this way, then the weight of the tasks including the cost of preemption, becomes schedule dependent and this class of problems is more difficult to solve.

The following definitions apply to a schedule with zero preemption and switching cost.

Define:

N_B — the number of tasks in the optimal basic schedule that are performed by processor P_1 . (Also note that $CT_{BS} = N_B$ because we are dealing with unit-task times.)

$N_1^{P_i}$ — the number of tasks in the SSAS that are started for the first time in P_i .

$N_2^{P_i}$ — the number of tasks in the SSAS that are stopped permanently in P_i .

$N_3^{P_i}$ — the number of tasks in the SSAS that are started after preemption in P_i .

$N_4^{P_i}$ — the number of tasks in the SSAS that are stopped temporarily in P_i .

$N_j = \min_i (N_j^{P_i})$ where P_i is any processor that finishes last and $1 \leq j \leq 4$.

$\{N_j + N_{j+2}\} = \min_i (N_j^{P_i} + N_{j+2}^{P_i})$ where P_i is any processor that finishes last and $j = 1, 2$.

$N_j^* = \max_i (N_j^{P_i})$ for $1 \leq j \leq 4$ and all i .

$\{N_j^* + N_{j+2}^*\} = \max_i (N_j^{P_i} + N_{j+2}^{P_i})$ for $j = 1, 2$ and all i .

$N_j^{P_i}$ - the number of ϵ_j 's in P_i if the SSAS consists of one assignment of form ℓ , for $\ell = \alpha, \beta, \gamma$.

Let U_j^ℓ and L_j^ℓ be an upper bound and a lower bound on the number of ϵ_j 's for any P_i when the SSAS, for the entire schedule, is only composed of assignments of form ℓ , for $\ell = \alpha, \beta, \gamma$.

Let $\{U_j + U_{j+2}\}^\ell$ and $\{L_j + L_{j+2}\}^\ell$ be an upper bound and a lower bound on the number of ϵ_j and ϵ_{j+2} , $j = 1, 2$, for any P_i when the SSAS, for the entire schedule, is only composed of assignments of form ℓ , for $\ell = \alpha, \beta, \gamma$.

We can then write

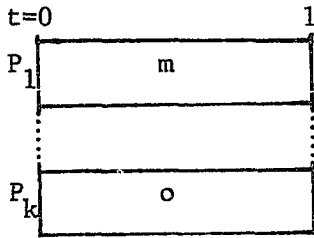
$$CT'_{BS} = CT_{BS} + N_B \epsilon_1 + N_B \epsilon_2 = N_B (1 + \epsilon_1 + \epsilon_2)$$

and

$$CT_{SSAS} + \sum_{i=1}^4 N_i^* \epsilon_i \geq CT'_{SSAS} \geq CT_{SSAS} + \sum_{i=1}^4 N_i \epsilon_i .$$

Our next step is to see how the different forms of assignments given before are influenced by considering $\epsilon_j \neq 0$, for any j .

Form α



$$CT'_{PS} = CT'_{BS} = 1 + \epsilon_1 + \epsilon_2$$

Form β

	$t=0$	$1+u_1/k$	$t(P_1) = \frac{k+u_1}{k} + \epsilon_1 + 2\epsilon_2 + \epsilon_3$
P_1	m		n''
P_2	n'		o''
\vdots	\vdots		\vdots
P_{k-1}	p'	q''	
P_k	q'	r	

$$\left. \begin{array}{l} t(P_i) = \frac{k+u_1}{k} + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 \\ \text{where: } 1 < i < k \\ 1 \leq u_1 < k \end{array} \right\}$$

$$t(P_k) = \frac{k+u_1}{k} + 2\epsilon_1 + \epsilon_2 + \epsilon_4$$

$$CT_{PS}' = \frac{k+u_1}{k} + \max \{ \epsilon_1 + 2\epsilon_2 + \epsilon_3, \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4, 2\epsilon_1 + \epsilon_2 + \epsilon_4 \}$$

Form γ

	$t=0$	$1+u_2/k$	
P_1	m		n''
\vdots	\vdots		\vdots
P_ℓ	p'	q	r''
\vdots	\vdots		\vdots
P_k	s'	t	

$$CT_{PS}' = \frac{k+u_2}{k} + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4$$

where: $1 < u_2 < k$

Before, we proceed in our analysis, we give a lemma that is going to help us to find the bounds for all N_i and N_i^* .

Lemma 13. Suppose the SSAS is only composed of assignments of Form β or only of Form γ . Then a bound on the minimum number of assignments of Form β or of Form γ in the entire schedule is equal to $N_B/2$. Also, a bound on the maximum number of assignments of Form β and of Form γ in the entire schedule is $kN_B/(k+1)$ and $kN_B/(k+2)$, respectively.

Proof: Here we only give the proof when the entire schedule is only composed of assignments of Form β because the proof for the other case is similar.

First, we will show that $N_B/2$ is a lower bound on the minimum number of assignments of Form β in the entire schedule. Consider that all the assignments of Form β have $k + \mu_1$ nodes, where μ_1 is constant for all of them and $1 \leq \mu_1 < k$. Then a lower bound on the minimum number of assignments of Form β is given by $kCT_{SSAS}/(k + \mu_1)$. But, in this case $CT_{SSAS} \geq \frac{k + \mu_1}{2k} CT_{BS}$. Therefore a lower bound on the minimum number of assignments of Form β , each with $k + \mu_1$ nodes, is equal to $CT_{BS}/2$ or $N_B/2$ because $CT_{BS} = N_B$ in an equally weighted task system. From this it follows that if μ_1 is not constant, then a lower bound on the minimum number of assignments in a schedule which only has assignments of Form β is

$$\min_{\mu_1} \left\{ \begin{array}{l} \text{bound on the minimum number of assign-} \\ \text{ments in a schedule which only have as-} \\ \text{signments of Form } \beta \text{ each with } k + \mu_1 \\ \text{nodes} \end{array} \right\} = \min_{\mu_1} \left\{ \frac{N_B}{2} \right\} = \frac{N_B}{2}.$$

Now, we will find an upper bound on the maximum number of assignments of Form β in the entire schedule. If all the assignments of Form β have $k + \mu_1$ nodes, $1 \leq \mu_1 < k$, then an upper bound on the maximum number of assignments is given by $\frac{kCT_{BS}}{k + \mu_1}$ or by $\frac{kN_B}{k + \mu_1}$ because $CT_{BS} = N_B$. This expression achieves its maximum value when μ_1 has its minimum value. Therefore, an upper bound on the maximum number of assignments of Form β , in the entire schedule, is $kN_B/(k + 1)$. The upper bound on the maximum number of assignments of Form γ in the entire schedule is found in a similar form. If all the assignments of Form γ have $k + \mu_2$ nodes,

$1 < u_2 < k$, then an upper bound on the maximum number of assignments is given by $kN_B/(k + u_2)$. This expression achieves its maximum value when u_2 has its minimum value. Therefore, an upper bound on the maximum number of assignments of Form γ in the entire schedule is $kN_B/(k + 2)$.

Now, we proceed to analyze the three possible cases.

SSAS with only assignments of Form α . In this case, it is clear that $CT_{SSAS} = CT_{BS}$ and therefore

$$U_1^\alpha = U_2^\alpha = L_1^\alpha = L_2^\alpha = N_B$$

and

$$U_3^\alpha = U_4^\alpha = L_3^\alpha = L_4^\alpha = 0.$$

SSAS with only assignments of Form β . We can observe that in one assignment of Form β , we have

$$\max_i(\beta_{N_1}^{P_i}) = \max_i(\beta_{N_2}^{P_i}) = 2$$

$$\max_i(\beta_{N_3}^{P_i}) = \max_i(\beta_{N_4}^{P_i}) = 1$$

$$(\beta_{N_1}^{P_i} + \beta_{N_3}^{P_i}) = (\beta_{N_2}^{P_i} + \beta_{N_4}^{P_i}) = 2$$

$$\min_i(\beta_{N_1}^{P_i}) = \min_i(\beta_{N_2}^{P_i}) = 1$$

$$\min_i(\beta_{N_3}^{P_i}) = \min_i(\beta_{N_4}^{P_i}) = 0.$$

Then, we will obtain the U_j^β using the relation

$$U_j^\beta = \max_i(\beta_{N_j}^{P_i}) \times (\text{bound on the maximum number of assignments of Form } \beta \text{ in } CT_{SSAS}).$$

By Lemma 13, we can write

$$U_j^\beta = \max_i(\beta_{N_j}^{P_i}) \times \frac{kN_B}{k + 1}.$$

Therefore:

$$U_1^\beta = U_2^\beta = \frac{2k}{k+1} N_B$$

$$U_3^\beta = U_4^\beta = \frac{k}{k+1} N_B$$

$$\{U_1 + U_3\}^\beta = \{U_2 + U_4\}^\beta = \frac{2k}{k+1} N_B .$$

Also, L_j^β is given by the expression

$$L_j^\beta = \min_i (\beta_{N_j}^{P_i}) \times (\text{bound on the minimum number of assignments of Form } \beta \text{ in CT}_{SSAS}).$$

By Lemma 13, we can write

$$L_j^\beta = \min_i (\beta_{N_j}^{P_i}) \times \frac{N_B}{2} .$$

Therefore:

$$L_1^\beta = L_2^\beta = N_B/2$$

$$L_3^\beta = L_4^\beta = 0$$

$$\{L_1 + L_3\}^\beta = \{L_2 + L_4\}^\beta = N_B .$$

SSAS with only assignments of Form γ . We can observe that in

one assignment of Form γ , we have

$$\max_i (\gamma_{N_1}^{P_i}) = \max_i (\gamma_{N_2}^{P_i}) = 2$$

$$\max_i (\gamma_{N_3}^{P_i}) = \max_i (\gamma_{N_4}^{P_i}) = 1$$

$$\max_i (\gamma_{N_1}^{P_i} + \gamma_{N_3}^{P_i}) = \max_i (\gamma_{N_2}^{P_i} + \gamma_{N_4}^{P_i}) = 3$$

$$\min_i (\gamma_{N_1}^{P_i}) = \min_i (\gamma_{N_2}^{P_i}) = 1$$

$$\min_i (\gamma_{N_3}^{P_i}) = \min_i (\gamma_{N_4}^{P_i}) = 0$$

$$\min_i (\gamma_{N_1}^{P_i} + \gamma_{N_3}^{P_i}) = \min_i (\gamma_{N_2}^{P_i} + \gamma_{N_4}^{P_i}) = 2.$$

Then, we will obtain the U_j^α using the relation

$$U_j^\alpha = \max_i (\gamma_{N_j}^{P_i}) \times (\text{bound on the maximum number of assignments of Form } \gamma \text{ in } CT_{SSAS}).$$

By Lemma 13, we can write

$$U_j^\alpha = \max_i (\gamma_{N_j}^{P_i}) \times \frac{kN_B}{k+2}.$$

Therefore:

$$U_1^\gamma = U_2^\gamma = \frac{2k}{k+2} N_B$$

$$U_3^\gamma = U_4^\gamma = \frac{k}{k+2} N_B$$

$$\{U_1 + U_3\}^\gamma = \{U_2 + U_4\}^\gamma = \frac{3k}{k+2} N_B.$$

Also, L_j^γ is given by the expression

$$L_j^\gamma = \min_i (\gamma_{N_j}^{P_i}) \times \frac{N_B}{2}.$$

Therefore:

$$L_1^\gamma = L_2^\gamma = \frac{N_B}{2}.$$

$$L_3^\gamma = L_4^\gamma = 0$$

$$\{L_1 + L_3\}^\gamma = \{L_2 + L_4\}^\gamma = N_B.$$

We note that the general SSAS consists of a mixture of α , β , and γ type assignments. In order to find the upper and lower bounds for the N_i^* and N_i , we need to state the following theorem which applies to any SSAS.

Theorem 23. In any SSAS,

$$\max_\ell (U_j^\ell) \geq N_j^* \geq N_j \geq \min_\ell (L_j^\ell) \quad \text{for } j = 1, 2, 3, 4 \text{ and } \ell = \alpha, \beta, \gamma$$

and

$$\max_{\ell} (U_j + U_{j+2})^{\ell} \geq \{N_j^* + N_{j+2}^*\} \geq \{N_j + N_{j+2}\} \geq \min_{\ell} (L_j + L_{j+2})^{\ell}$$

for $j = 1, 2$
and $\ell = \alpha, \beta, \gamma$.

The proof of this theorem is omitted because it is immediate.

Applying Theorem 23 we obtain the following bounds:

$$\frac{3k}{k+2} N_B \geq \begin{cases} \{N_1^* + N_3^*\} \\ \{N_2^* + N_4^*\} \end{cases} \quad (5) \quad \text{for } k > 2.$$

$$\frac{2k}{k+1} N_B \geq \begin{cases} \{N_1^* + N_3^*\} \\ \{N_2^* + N_4^*\} \end{cases} \quad (6) \quad \text{for } k = 2.$$

$$\frac{2k}{k+1} N_B \geq \begin{cases} N_1^* \\ N_2^* \end{cases} \quad (7)$$

$$\frac{k}{k+1} N_B \geq \begin{cases} N_3^* \\ N_4^* \end{cases} \quad (8)$$

$$\left\{ \begin{array}{l} \{N_1 + N_3\} \\ \{N_2 + N_4\} \end{array} \right\} \geq N_B \quad (9)$$

$$\left\{ \begin{array}{l} N_1 \\ N_2 \end{array} \right\} \geq \frac{N_B}{2} \quad (10)$$

$$\left\{ \begin{array}{l} N_3 \\ N_4 \end{array} \right\} \geq 0. \quad (11)$$

Note that the Formula 5 is valid for $k > 2$, while the Formula 6 is valid only for $k = 2$. The reason for these two bounds on $\{N_i^* + N_{i+2}^*\}$, $i = 1, 2$, is that, in a two-processor system, we cannot have assignments

of Form γ . The other bounds are valid for both cases, because the upper and lower bounds come from the case in which we have assignments only of Form β , and by Lemmas 11 and 12 we know that this type of assignment is possible for any value of k ($k \geq 2$).

The following theorem gives a lower bound on the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks on a k -processor system.

Theorem 24. For any graph G with equally weighted tasks on a k -processor system, we have

$$\frac{CT'_{SSAS}}{CT'_{BS}} \geq \frac{k(1 + \epsilon_1 + \epsilon_2) + 1}{2k(1 + \epsilon_1 + \epsilon_2)}$$

Proof: We know that

$$CT'_{SSAS} \geq CT_{SSAS} + N_1\epsilon_1 + N_2\epsilon_2 + N_3\epsilon_3 + N_4\epsilon_4$$

and

$$CT'_{BS} = CT_{BS} + N_B(\epsilon_1 + \epsilon_2).$$

It is also known from (14) $CT_{SSAS} \geq \gamma CT_{BS}$ where $\gamma = (k + 1)/(2k)$.

Inserting the first two inequalities in the last one, we obtain

$$CT'_{SSAS} - N_1\epsilon_1 - N_2\epsilon_2 - N_3\epsilon_3 - N_4\epsilon_4 \geq \gamma CT'_{BS} - \gamma N_B(\epsilon_1 + \epsilon_2) \quad (12)$$

or

$$\frac{CT'_{SSAS}}{CT'_{BS}} \geq \gamma + \frac{(N_1 - \gamma N_B)\epsilon_1}{CT'_{BS}} + \frac{(N_2 - \gamma N_B)\epsilon_2}{CT'_{BS}} + \frac{N_3\epsilon_3}{CT'_{BS}} + \frac{N_4\epsilon_4}{CT'_{BS}}.$$

Using Relations 10 and 11 and $CT'_{BS} = N_B(1 + \epsilon_1 + \epsilon_2)$, we have

$$\frac{CT'_{SSAS}}{CT'_{BS}} \geq \frac{k + 1}{2k} - \frac{\epsilon_1 + \epsilon_2}{2k(1 + \epsilon_1 + \epsilon_2)},$$

and the result of the theorem follows.

The next theorem gives a lower bound on the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks on a k -processor system when all the ϵ_i are equal, i.e., $\epsilon_i = \epsilon$, for all i .

Theorem 25. For any graph G with equally weighted tasks on a k -processor system and $\epsilon_i = \epsilon$ for all i , we have

$$\frac{CT'_{SSAS}}{CT'_{BS}} \geq \frac{k(1 + 4\epsilon) + 1}{2k(1 + 2\epsilon)}$$

Moreover, this is the best possible bound.

Proof: In this case the Inequality 12 can be rewritten as

$$CT'_{SSAS} \geq \gamma CT'_{BS} + (N_1 + N_2 + N_3 + N_4 - 2\gamma N_B)\epsilon.$$

Using Relation 9 and $CT'_{BS} = N_B(1 + 2\epsilon)$, we have

$$CT'_{SSAS} \geq \gamma CT'_{BS} + 2(1 - \gamma)N_B\epsilon$$

or

$$\frac{CT'_{SSAS}}{CT'_{BS}} \geq \frac{k + 1}{2k} + \frac{2(k - 1)\epsilon}{2k(1 + 2\epsilon)},$$

and the result of this theorem follows. We can show that this is, in fact, the best possible bound by an example. Figure 28, given in Chapter V, shows a graph for which $CT'_{SSAS} = (k + 1)/k + 4\epsilon$ and $CT'_{BS} = 2 + 4\epsilon$.

Table 2 gives the lower bounds of the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks on a k -processor system for other possible relations among the ϵ_i 's. These bounds, derived in (23), are tighter than the bound given in Theorem 24.

Because $CT_{SSAS} = CT_{SAS} = CT_{PS}$ (18) for a two-processor system, we can use Table 2 with $k = 2$ to obtain tighter bounds than those given

Table 2. Lower bounds on the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks

Case	Lower bound
$\epsilon_1 \geq \epsilon_3$ $\epsilon_2 \geq \epsilon_4$	$\frac{k(1 + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4) + 1}{2k(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_1 \geq \epsilon_3$ $\epsilon_4 \geq \epsilon_2$	$\frac{k(1 + \epsilon_1 + 2\epsilon_2 + \epsilon_3) + 1}{2k(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_2 \geq \epsilon_4$	$\frac{k(1 + 2\epsilon_1 + \epsilon_2 + \epsilon_4) + 1}{2k(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_4 \geq \epsilon_2$	$\frac{k(1 + 2\epsilon_1 + 2\epsilon_2) + 1}{2k(1 + \epsilon_1 + \epsilon_2)}$

in Theorem 21 and in Table 1. In particular, for $k = 2$, Theorems 24 and 25 yield

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{3 + 2(\epsilon_1 + \epsilon_2)}{4(1 + \epsilon_1 + \epsilon_2)} \quad \text{for unequal } \epsilon_i \text{'s, and}$$

$$\frac{CT'_{PS}}{CT'_{BS}} \geq \frac{3 + 4\epsilon}{4(1 + 2\epsilon)} \quad \text{for } \epsilon_i = \epsilon, \forall_i.$$

The next theorem gives the corresponding upper bound on the ratio CT'_{SSAS}/CT'_{BS} .

Theorem 26. For any graph G with equally weighted tasks on a k -processor system,

$$\frac{k(1 + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4) + 1}{(k + 1)(1 + \epsilon_1 + \epsilon_2)} \geq \frac{CT'_{SSAS}}{CT'_{BS}}$$

Proof: We know that

$$CT_{SSAS} + N_1^* \epsilon_1 + N_2^* \epsilon_2 + N_3^* \epsilon_3 + N_4^* \epsilon_4 \geq CT'_{SSAS}$$

and

$$CT'_{BS} = CT_{BS} + N_B(\epsilon_1 + \epsilon_2).$$

In general

$$CT_{BS} \geq CT_{SSAS}.$$

Inserting the first two equations in the last one and collecting terms, we obtain

$$CT'_{BS} + (N_1^* - N_B)\epsilon_1 + (N_2^* - N_B)\epsilon_2 + N_3^*\epsilon_3 + N_4^*\epsilon_4 \geq CT'_{SSAS}. \quad (13)$$

Dividing both sides by CT'_{BS}

$$1 + \frac{(N_1^* - N_B)\epsilon_1}{CT'_{BS}} + \frac{(N_2^* - N_B)\epsilon_2}{CT'_{BS}} + \frac{N_3^*\epsilon_3}{CT'_{BS}} + \frac{N_4^*\epsilon_4}{CT'_{BS}} \geq \frac{CT'_{SSAS}}{CT'_{BS}}.$$

Using Relations 7 and 8 and $CT'_{BS} = N_B(1 + \epsilon_1 + \epsilon_2)$ in the last inequality, we have

$$1 + \frac{(k-1)(\epsilon_1 + \epsilon_2)}{(k+1)(1 + \epsilon_1 + \epsilon_2)} + \frac{k(\epsilon_3 + \epsilon_4)}{(k+1)(1 + \epsilon_1 + \epsilon_2)} \geq \frac{CT'_{SSAS}}{CT'_{BS}}.$$

Therefore,

$$\frac{k(1 + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4) + 1}{(k+1)(1 + \epsilon_1 + \epsilon_2)} \geq \frac{CT'_{SSAS}}{CT'_{BS}}.$$

In the following theorem we give an upper bound on the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks on a k ($k > 2$) processor system when all the ϵ_i are equal.

Theorem 27. For any graph G with equally weighted tasks on a k ($k > 2$) processor system where $\epsilon_i = \epsilon$ for all i ,

$$\frac{k(1 + 6\epsilon) + 2}{(k+2)(1 + 2\epsilon)} \geq \frac{CT'_{SSAS}}{CT'_{BS}}.$$

Proof: When $\epsilon_i = \epsilon$ for all i , Inequality 13 can be rewritten as

$$1 + \frac{(N_1^* + N_2^* + N_3^* + N_4^* - 2N_B)\epsilon}{CT'_{BS}} \geq \frac{CT'_{SSAS}}{CT'_{BS}}.$$

Using Relation 5 and the fact that $CT'_{BS} = N_B(1 + 2\epsilon)$, we have

$$1 + \frac{4(k-1)\epsilon}{(k+2)(1+2\epsilon)} \geq \frac{CT'_{SSAS}}{CT'_{BS}},$$

and the result of the theorem follows.

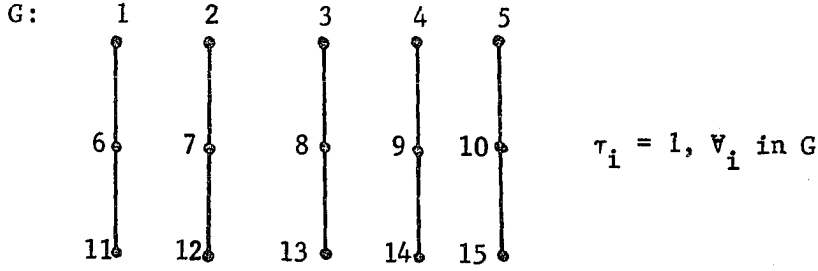
We can show that this is, in fact, the best possible bound for certain values of k by an example. Figure 31 shows a graph for which the shortest subset assignment schedule, S' , and the optimal nonpre-emptive schedule, S , have lengths $5 + 18\epsilon$ and $5 + 10\epsilon$, respectively, for the case $k = 3$. Therefore, $CT'_{SSAS}/CT'_{BS} = (5 + 18\epsilon)/(5 + 10\epsilon)$. Note that these calculations vary with the nonuniqueness of the SSAS for $k > 2$. By interchanging the roles of processors 1, 2 and 3 for the second and third subset, we obtain $CT'_{SSAS} = 5 + 14\epsilon$.

Table 3 gives the upper bounds on the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks on a k ($k > 2$) processor system for other possible relations among the ϵ_i 's. These bounds, derived in (23), are tighter than the bound given in Theorem 26.

In the following theorem we give an upper bound on the ratio CT'_{SSAS}/CT'_{BS} for any graph G with equally weighted tasks on a two-processor system when all the ϵ_i are equal.

Theorem 28. For any graph G with equally weighted tasks on a two-processor system where $\epsilon_i = \epsilon$ for all i ,

$$\frac{3 + 8\epsilon}{3 + 6\epsilon} \geq \frac{CT'_{SSAS}}{CT'_{BS}}$$



	t=0	1	2	3	4	5	
S: P ₁	1	4	7	10	13		
P ₂	2	5	8	11	14		CT' _{BS} = 5 + 10e
P ₃	3	6	9	12	15		

	t=0	5/3		10/3		5		
P ₁	1	2	6	7	11	12		
P ₂	2	3	4	7	8	9	12	13
P ₃	4	5	9	10	14	15		

CT'_{SSAS} = 5 + 18e

$$\therefore \frac{CT'_{SSAS}}{CT'_{BS}} = \frac{5 + 18e}{5 + 10e}$$

Figure 31. Example which achieves the upper bound on CT'_{SSAS}/CT'_{BS} when $k > 2$ and $\epsilon_i = \epsilon$.

Moreover, this represents the best possible bound.

Proof: When $\epsilon_i = \epsilon$ for all i , Inequality 13 can be rewritten as

$$1 + \frac{(N_1^* + N_2^* + N_3^* + N_4^* - 2N_B)\epsilon}{CT'_{BS}} \geq \frac{CT'_{SSAS}}{CT'_{BS}}.$$

Table 3. Upper bounds on the ratio CT_{SSAS}'/CT_{BS}' for any graph G with equally weighted tasks on a k ($k > 2$) processor system

Case	Upper bound
$\epsilon_1 \geq \epsilon_3$ $\epsilon_2 \geq \epsilon_4$	$\frac{k^2(1 + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4) + k(3 + 4\epsilon_1 + 4\epsilon_2 - \epsilon_3 - \epsilon_4) + 2}{(k+1)(k+2)(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_1 \geq \epsilon_3$ $\epsilon_4 \geq \epsilon_2$	$\frac{k^2(1 + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4) + k(3 + 4\epsilon_1 + \epsilon_2 - \epsilon_3 + 2\epsilon_4) + 2}{(k+1)(k+2)(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_2 \geq \epsilon_4$	$\frac{k^2(1 + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4) + k(3 + \epsilon_1 + 4\epsilon_2 + 2\epsilon_3 - \epsilon_4) + 2}{(k+1)(k+2)(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_4 \geq \epsilon_2$	$\frac{k^2(1 + 2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4) + k(3 + \epsilon_1 + \epsilon_2 + 2\epsilon_3 + 2\epsilon_4) + 2}{(k+1)(k+2)(1 + \epsilon_1 + \epsilon_2)}$

Using Relation 6 and the fact that $CT_{BS}' = N_B(1 + 2\epsilon)$, we have

$$1 + \frac{2\epsilon}{3 + 6\epsilon} = \frac{3 + 8\epsilon}{3 + 6\epsilon} \geq \frac{CT_{SSAS}'}{CT_{BS}'}.$$

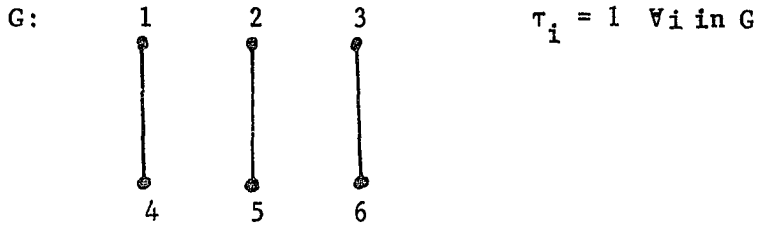
We can show that this is, in fact, the best bound by an example.

Figure 32 shows a graph for which the CT_{SSAS}' and CT_{BS}' have lengths $3 + 8\epsilon$ and $3 + 6\epsilon$, respectively.

Corollary. For any graph G with equally weighted tasks on a two-processor system where $\epsilon_i = \epsilon$ for all i ,

$$\frac{3 + 8\epsilon}{3 + 6\epsilon} \geq \frac{CT_{PS}'}{CT_{BS}'}.$$

Moreover, this represents the best bound.



S:

P_1	1	3	5
P_2	2	4	6

$CT'_{BS} = 3 + 6\epsilon$

S':

P_1	1	2	4	5
P_2	2	3	5	6

$CT'_{SSAS} = 3 + 8\epsilon$

$$\therefore \frac{CT'_{SSAS}}{CT'_{BS}} = \frac{3 + 8\epsilon}{3 + 6\epsilon}$$

Figure 32. Example which achieves the upper bound on CT'_{SSAS}/CT'_{BS} when $k = 2$ and $\epsilon_i = \epsilon$.

Because $CT_{SSAS} = CT_{SAS} = CT_{PS}$ in a two-processor system, the proof is immediate.

Table 4 gives the upper bounds on the ratio CT'_{PS}/CT'_{BS} for any graph G with equally weighted tasks on a two-processor system for other possible relations among the ϵ_i 's. These bounds are derived in (23).

Table 4. Upper bounds on the ratio CT_{PS}'/CT_{BS}' for any graph G with equally weighted tasks on a two-processor system

Case	Upper bound
General formula	$\frac{3 + 2(2\epsilon_1 + 2\epsilon_2 + \epsilon_3 + \epsilon_4)}{3(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_1 \geq \epsilon_3$ $\epsilon_2 \geq \epsilon_4$	$\frac{3 + 4(\epsilon_1 + \epsilon_2)}{3(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_1 \geq \epsilon_3$ $\epsilon_4 \geq \epsilon_2$	$\frac{3 + 2(2\epsilon_1 + \epsilon_2 + \epsilon_4)}{3(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_2 \geq \epsilon_4$	$\frac{3 + 2(\epsilon_1 + 2\epsilon_2 + \epsilon_3)}{3(1 + \epsilon_1 + \epsilon_2)}$
$\epsilon_3 \geq \epsilon_1$ $\epsilon_4 \geq \epsilon_2$	$\frac{3 + 2(\epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4)}{3(1 + \epsilon_1 + \epsilon_2)}$

VII. CONCLUSIONS

A. Contributions

We have presented a complete study of the stability of the schedules produced by the three most important polynomial bounded algorithms that produce an optimal schedule length when certain conditions are satisfied. The most interesting results are:

1) We have proved that the schedules produced by the optimal nonpreemptive scheduling algorithm for equally weighted tasks and tree-structured graphs are free of all three anomalies commonly found in multiprocessing systems. Because they are stable, we can bound the schedule length of a tree-structured graph with unequally weighted tasks. This bound is obtained assuming equal execution time τ^* for all tasks and k processors, where $\tau^* = \max_i(\tau_i)$. Then we can be assured that any schedule based on $k' > k$, unequal execution times $\tau'_i \leq \tau^*$, and any allowable relaxation of \leq will have a schedule length shorter than the given bound.

2) We have proved that the schedules produced by the optimal nonpreemptive scheduling algorithm for equally weighted tasks for two-processor systems are free of two of the three anomalies. Additional constraints would have to be placed on the task system to prevent the third type of anomaly. Here we can ensure only that increasing the number of available processors or eliminating some precedence relation does not increase the resulting schedule length. However, the new system with additional constraints is free from all three anomalies.

3) We have proved that the schedules produced by the optimal preemptive scheduling algorithm for general graphs on a two-processor system and for tree-structured graphs are free of all three anomalies. Because they are stable, we can bound the schedule length based only on the maximum execution time of each task, the minimum number of processors and the maximally constrained graph.

The subset assignment concept has been studied in depth. The most important results are:

1) We have proved that the SAS for general graphs with unequally weighted tasks on a k-processor system is free of three anomalies when the subset sequence is held fixed, and free of two anomalies when the subset sequence is recomputed. We conjecture that it is free of the third type of anomaly. Here we can ensure that increasing the number of processors, or eliminating some precedence relation, does not increase the resulting schedule length.

2) We have bounded the performance of SSAS in comparison with the optimal basic schedule, when we deal with general graphs with equally weighted tasks on a k-processor system and we consider the costs of preemption and switching positive. The SSAS is of interest because the SAS is probably a good heuristic for preemptive schedules (just as level algorithms are good heuristics for optimal nonpreemptive schedules) and because the SSAS is shorter than SAS.

Other major results of this research are the bounds given for the ratio between the optimal preemptive schedule to the optimal non-preemptive schedule for unequally and equally weighted task systems, when we take in consideration the possible costs of switching and

preempting tasks. These bounds show that in reality we can no longer say that the optimal preemptive scheduling discipline is strictly more powerful than the optimal nonpreemptive scheduling discipline. These bounds also suggest that, in some cases, reduction of schedule length will not be significant. Thus, it may not be worth the effort to implement an optimal preemptive schedule, due to the great deal of work demanded by the scheduling algorithm.

B. Future Work

We have noted the scarcity of results in the field of static scheduling. Some of the unsolved problems connected with this study are:

- 1) Completion of the study of the stability of B-schedules under nonoptimal conditions;
- 2) Completion of the study of the stability of A-schedules under nonoptimal conditions;
- 3) Determination of upper bounds of the ratio CT_{PS}^i/CT_{BS}^i for general graphs on k ($k > 2$) processor systems, and for unequally weighted tasks on a two-processor system;
- 4) Simulation studies of the behavior of the ratio CT_{PS}/CT_{BS} , taking in consideration cost of preemption and cost of switching;
- 5) A refinement of the method of comparing scheduling disciplines when the costs of preemption and switching are not negligible;
- 6) Resolution of the conjecture that the SAS is anomaly-free when the execution of one or more tasks is reduced and the subset sequence is recalculated.

The continued search for simple algorithms and analysis of their performance is worthy of further study in the overall investigation of the usefulness of multiprocessing.

VIII. BIBLIOGRAPHY

1. Adam, T. L., Chandy, K. M., and Dickson, J. R. A comparison of list schedules for parallel processing systems. Unpublished paper. Computer Science Department, University of Texas, Austin, Texas, 1974.
2. Bauer, H. Subproblems of the $m \times n$ sequencing problem. Unpublished Ph.D. dissertation. Department of Computer Science, Stanford University, 1972.
3. Chandy, K. M., and Dickson, J. R. Scheduling unidentical processors in a stochastic environment. COMCON Proceedings 72 (1972): 171-174.
4. Clark, N. The Gantt chart. London: Sir Isaac Pitman & Sons, Ltd., 1952.
5. Coffman, E. G., and Denning, P. J. Operating systems theory. Englewood Cliffs, N.J.: Prentice Hall Publishing Company, 1973.
6. Coffman, E. G., and Graham, R. L. Optimal scheduling for two-processor systems. Acta Informatica 1, No. 3 (1972): 200-213.
7. Gonzalez, M. J., and Ramamoorthy, C. V. Parallel task execution in a decentralized system. IEEE Transactions on Computers C-21 (1972): 1310-1322.
8. Graham, R. L. Bounds for certain multiprocessing anomalies. Bell System Tech. Journal 45 (1966): 1563-1581.
9. Graham, R. L. Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17 (1969): 416-429.
10. Graham, R. L. Bounds on multiprocessing anomalies and related packing algorithms. SJCC Conference Proceedings 40 (1972): 205-217.
11. Hu, T. C. Parallel sequencing and assembly line problems. Operations Research 9, No. 6 (1961): 841-848.
12. Kaufman, M. T. Anomalies in scheduling unit-time tasks. Unpublished Technical Report No. 34. Digital Systems Laboratory, Stanford University, 1972.
13. Kaufman, M. T. An almost optimal algorithm for the assembly line scheduling problem. IEEE Transactions on Computers C-23 (1974): 1169-1174.

14. Liu, C. L. Optimal scheduling on multiprocessor computing systems. IEEE Switching Automata Theory Symposium Proceedings 13 (1972): 155-160.
15. Manacher, G. K. Production and stabilization of real-time tasks schedules. Journal of the ACM 14, No. 3 (1967): 439-465.
16. McNaughton, R. Scheduling with deadlines and loss functions. Management Science 6, No. 1 (1959): 1-12.
17. Muntz, R. R. Scheduling of computations on multiprocessor systems: The preemptive assignment discipline. Unpublished Ph.D. dissertation. Electrical Engineering Department, Princeton University, 1969.
18. Muntz, R. R., and Coffman, E. G. Optimal preemptive scheduling on two-processor systems. IEEE Transactions on Computers C-18 (1969): 1014-1020.
19. Muntz, R. R., and Coffman, E. G. Preemptive scheduling of real-time tasks on multiprocessor systems. Journal of the ACM 17, No. 2 (1970): 324-328.
20. Ramamoorthy, C. V., Chandy, K. M., and Gonzalez, M. J. Optimal scheduling strategies in a multiprocessor system. IEEE Transactions on Computers C-21 (1972): 137-146.
21. Richards, P. Timing properties of multiprocessor systems. Technical paper Rep. No. TD-B60-27. Tech. Operations, Inc., Burlington, Massachusetts, 1960.
22. Ullman, J. D. Polynomial completeness of the equal execution time scheduling problem. Unpublished paper. Computer Science Report TR-115. Electrical Engineering Department, Princeton University, 1972.
23. Villanueva, J. E. Effects of preemptive costs in optimal schedules. Unpublished paper. Computer Science Department, Iowa State University, 1975.

IX. ACKNOWLEDGMENTS

I wish to express my sincerest gratitude to Dr. Robert M. Stewart for his confidence, advice, and the support of my program when it was needed.

My appreciation goes to Dr. Arthur E. Oldehoeft for suggesting the research topic, for the many hours of assistance, for his advice, extensive critique and encouragement during the preparation of this thesis.

Also a special thanks to Mr. Alan Sweet for his comments, and to all my friends who made my work at Ames pleasurable and rewarding.

I want to express my appreciation to Dr. Fred L. Mann and Dr. Samuel M. Fahr for their aid in the attainment of sponsorship from Peru-Iowa Mission for the first three years of my graduate studies at Iowa State University. My sincere thanks are offered also to Dr. John F. Timmons.

Finally, I will thank Dr. José H. Portillo for his confidence, and my parents and brothers for their encouragement during my stay in Ames.