Scheduling Task Systems With Resources

by

Errol Lynn Lloyd

B.S., The Pennsylvania State University
(1975)

B.S., The Pennsylvania State University
(1975)

S.M., Massachusetts Institute of Technology
(1977)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1980

Signature of Author.....
Department of Electrical Engineering and Computer Science
April 24, 1980

Certified by.....
Ronald L. Rivest
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Committee on Graduate Students

Scheduling Task Systems With Resources

by

Errol Lynn Lloyd

Submitted to the Department of Electrical Engineering and Computer Science
on April 24, 1980 in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Abstract

Minimum execution time scheduling of task systems with resources has been the subject of several papers over the past few years. The model used for much of this work assumes that the resources in the system are continuous. That is, there is one unit of each resource, and a task may require any portion of that unit during its execution. While this is a reasonable assumption for certain bin packing applications, it is intuitively unreasonable for certain other applications. In particular, the resources associated with computer systems - readers, printers, disk drives - are not "continuous" resources. We present an alternative model of task systems with resources in which the resources are discrete. That is, there are a specific number of indivisible units of each resource and a task may require only integral numbers of those units. Several results involving the worst case performance of list scheduling and critical path scheduling with respect to this model are given. A new result on critical path scheduling of task systems with continuous resources is also given. Finally, a comparison will be made between corresponding bounds for the continuous and discrete models.

Thesis Supervisor: Ronald L. Rivest
Title: Associate Professor of Computer Science and Engineering

Acknowledgements

My advisor, Ron Rivest, is to be thanked for his guidance and support thoughout my graduate career. His suggestions and questions have greatly improved both the accuracy and the presentation of this thesis. His patience in listening to initial versions of many of these results is especially appreciated. Adi Shamir and Christos Papadimitriou provided a number of suggestions and comments which helped to make this thesis a more cohesive and uniform document.

Special thanks go to two people who helped to shape my career: my father, because he first introduced me to computers when I took a programming course from him while in high school, and Donald Johnson of Penn State, who first introduced me to the notions of algorithmic complexity. They are also the finest two teachers that I know - I hope that in my teaching career, I can approach their level of excellence.

Next, I would like to thank my friends at MIT and elsewhere for their encouragement and friendship. In particular, my fellow graduate students Paul Bayer, Peter Bloniarz, Andrea LaPaugh, Bill Masek and Ron Pinter, have made MIT a good place to live and study over the past five years.

Most importantly, I would like to thank my family for their support and love: my brother, Russell, with whom I've enjoyed many years of friendly competition; my parents, I shall be forever thankful for them; my wife Isabel, with whom I've shared the past nine years at Penn State and MIT - together we look forward to whatever the future brings.

This work was partially supported by the National Science Foundation under Grant MCS78-05849.

Table of Contents

List of Figures

Chapter 1 - Task Systems

Over the past fifteen years one of the most active areas of computer science and industrial engineering research has been the scheduling of large systems. This research has been motivated both by the existence of large industrial scheduling problems and by the existence of high speed digital computers to solve those problems. Moreover, the models used to study these scheduling problems have attracted great theoretical interest, and as a result, an immense quantity of research has been done on them.

In general, a scheduling problem is of the following form: Given a set of tasks which need to be completed, produce a schedule of minimum length for completing those tasks. Often, there are a number of constraints placed upon the form that the schedule may take. For example, some tasks may need to be completed before others can be started, or there may be a limit on the number of tasks that can be "in progress" at any given time, or some tasks may require longer to complete than others. Many types of constraints are possible.

It should be apparent, even from the informal description given above, that the scheduling of systems of tasks is not trivial, and that ad-hoc methods have almost no chance of producing even near optimal schedules, much less optimal schedules. The obvious approach then is to formulate a standard set of rules (hopefully, a good set) for producing schedules. Indeed, the design and analysis of algorithms for scheduling has been the primary area of research concentration. For some classes of task systems, fast algorithms have been developed which produce optimal schedules for those systems. For other classes of task systems, it has been shown that finding algorithms which produce optimal schedules in a reasonable amount of time is unlikely. For these classes of task systems, the research has focused on producing good, polynomial time, heuristic algorithms. That is, algorithms which, in a reasonable amount of time, produce good, though not necessarily optimal, schedules. In conjunction with this, the performance of

various simple and/or fast scheduling algorithms has been analyzed, so as to provide a performance "benchmark" that more complicated algorithms can be compared to.

In this chapter we define the notions of a task system, of a schedule, and of a number of related concepts that we will use throughout this thesis. We also give a summary of the major results pertaining to the basic task system model which we describe here.

1.1 The basic task system model

A task system is a system $S = \langle T, \prec, m \rangle$ where:

1. $T = \{T_1, \ldots, T_n\}$ is a set of tasks - associated with T_i is a positive integral execution time τ_i.
2. $\prec$ is a partial order specifying precedence constraints between the tasks.
3. There are m identical processors.

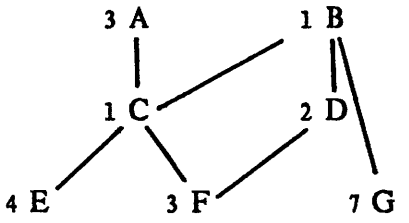
With respect to the precedence constraints, we have the following definitions: If $T_i \prec T_j$, then T_j is a successor of T_i, and T_i is a predecessor of T_j. We will represent the partial order by a directed acyclic graph (dag) with one node for each task and one arc for each relation in the partial order. We assume that there are no transitive edges in the dag. Two examples of task systems are given in Figure 1.1 -- one is a fully general task system and the other is a task system in which all of the tasks have an execution time of one.

A valid schedule for a task system S, is a mapping $\sigma: T \rightarrow (N - \{0\})$ such that:

1. For all $t \in (N - \{0\})$, $m \geq |\{T_i \in T: \sigma(T_i) \leq t \leq \sigma(T_i) + \tau_i - 1\}|$.
2. If $T_i \prec T_j$, then $\sigma(T_i) + \tau_i - 1 < \sigma(T_j)$.

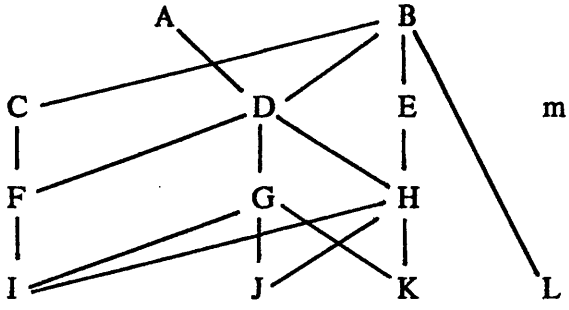
These two conditions correspond to our intuitive notion of what constitutes a schedule: that the tasks be executed on m processors subject to the precedence constraints. More specifically, the first condition ensures that at most m processors are in use at any given time. The second condition ensures that the precedence constraints are not violated. That is, if $T_i \prec T_j$, then T_i must have completed execution before T_j can begin execution.

Figure 1.1: Basic task systems



m = 3 processors

a) A task system with 3 processors and 7 tasks. The task execution times are given beside the tasks in the dag.



m = 2 processors

b) A task system with 2 processors and 13 tasks. Each task has an execution time of one.

Figure 1.2: Valid schedules

Schedule:

A	A	A	C	E	E	E	E
B	G	G	G	G	G	G	G
	D	D		F	F	F	

Time unit: 1 2 3 4 5 6 7 8

a) A valid schedule for the task system given in Figure 1.1a. Cross-hatching is used to indicate idle processors. The mapping σ is not given explicitly.

Schedule:

A	B	E	C	F	G	I	K
M		D	H	L		J	

Time unit: 1 2 3 4 5 6 7 8

b) A valid schedule for the task sytem given in Figure 1.1b.

Given a valid schedule, we define for each $i \in (N - \{0\})$, the set $B_i = \{T_j \in T: \sigma(T_j) \leq i \leq \sigma(T_j) + \tau_j - 1\}$. Also, let $\omega = \min\{i : (Nj > i)[B_j = \emptyset]\}$. The schedule has length ω, and consists of the ω time units $B_1, \ldots, B_\omega$. For each time unit B_i, if $|B_i| < m$, then B_i has $m - |B_i|$ idle processors. Intuitively, we assume that the processors are numbered from 1 to m, and that processors 1 through $|B_i|$ have tasks executing on them and that processors $|B_i| + 1$ through m are idle. Examples of valid schedules for the task systems in Figure 1.1 are given in Figure 1.2.

Finally, we note that there are a number of criterion for determining the "goodness" of a schedule. The most widely used, and in many senses the most natural, is that of minimizing the schedule length. This criterion is refered to as the minimum execution time or latest finishing time criterion. This is the measure of optimality that we use throughout this thesis.

1.2 Common submodels

The model of task systems presented above provides a starting point for virtually all theoretical scheduling research. This model has proven however, to be extremely difficult to deal with in its full generality. Moreover, many practical applications are most effectively analyzed using various submodels of the model given above. Most of the research has focused on two particular submodels of the basic task system model. These submodels are:

1. Task systems where $<$ is empty. That is, there are no precedence constraints in the system.
2. Task systems where all of the task execution times are identical. In this case we assume without loss of generality that each $\tau_i = 1$. These are unit execution time (UET) task systems.

With the exception of Chapter 7, we will deal exclusively with UET task systems in this thesis.

1.3 Scheduling algorithms

In this section we describe the three types of schedules which we will utilize.

1.3.1 List schedules

List schedules are the most basic of the schedules which we will examine. They are of particular

interest not only because of their simplicity, but also because most interesting scheduling algorithms produce schedules which form a subclass of the list schedules. Intuitively, a list schedule is formed as follows: Consider any (ordered) list L, of the tasks in T. The tasks are scheduled as follows: Whenever a processor becomes idle, the list L is instantaneously scanned from its beginning until a task T (if any) is found all of whose predecessors have completed execution. Task T is assigned to the idle processor and is removed from L.

More formally, a task T_i is ready at time l if for every T_j such that $T_j < T_i$, $\sigma(T_j) + \tau_j - 1 < l$. A list schedule is a valid schedule which is generated as follows:

1. Initially, L is an (ordered) list of the tasks in T and l is 1.
2. While L is nonempty perform this step
   a. Let $k = |\{T_i \notin L : \sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1\}|$
   b. For each of the first m - k (or less, if there aren't m - k) ready tasks T_i, on L at time l, let $\sigma(T_i) = l$ and remove T_i from L.
   c. Let $l = 1 + \min\{\sigma(T_i) + \tau_i - 1 : T_i \notin L \text{ and } \sigma(T_i) + \tau_i - 1 \geq l\}$.

Figure 1.3 shows an example of a list schedule for the UET task system given in Figure 1.1b.

List schedules are particularly attractive when dealing with UET task systems. In this case the restriction that only list schedules (and subclasses of list schedules) be considered as possible schedules for the task system causes no loss of generality. To see this, consider any schedule for a UET task system, and assume that schedule consists of time units $B_1, \ldots, B_\omega$. A schedule with length no more than ω results from the list consisting of the tasks in B_1, followed by the tasks in B_2, followed by the tasks in B_3, and so on, ending with the tasks in B_ω. Figure 1.4 shows that it is not generally true for non-UET task systems that there is always a list schedule of minimum length among all schedules for the system.

Finally, for list schedules, note that given a list L, the corresponding schedule (i.e. the mapping σ) is uniquely determined. For this reason, it is common practice when dealing with list schedules to simply

Figure 1.3: A list schedule for the task system in Figure 1.1b.

List: (M L K J I A B C D E F G H)

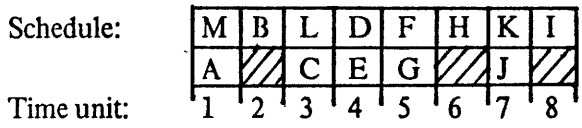
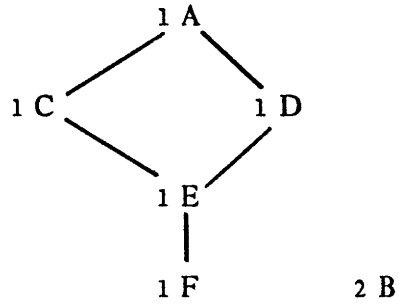
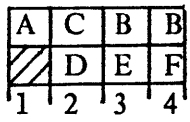


Figure 1.4: List schedules are not best for non-UET systems.



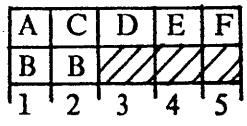
Task execution times are given beside the tasks.

An optimal schedule:



A list schedule:

List: (A B C D E F)



In fact, $\sigma(B) = 1$ in every list schedule for this system.

give the list L, along with an informal description of the underlying schedule. The mapping σ is not formally specified, because it tends to obscure, rather than illuminate, the nature of the schedule. Throughout this thesis we will follow the practice of specifying only the list and not the mapping σ.

1.3.2 Critical path schedules

Critical path schedules are one of the most widely studied subclasses of list schedules. Intuitively, these are schedules in which the tasks are ordered within the list according to their distance from a leaf of the dag which represents the precedence structure (a leaf is a node with no successors). The idea is that tasks far from the leaves should be executed first.

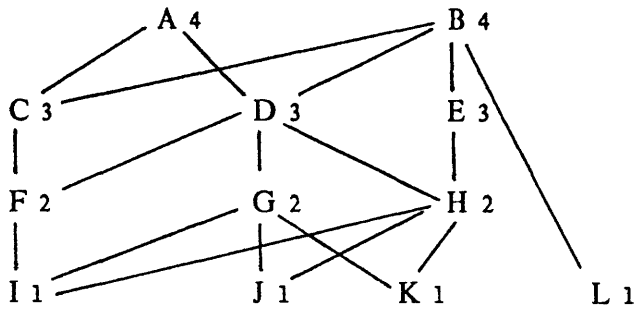
More formally, the level of a task in the precedence structure may be defined as follows: If T_i has no successors, then $level(T_i) = 1$; otherwise, $level(T_i) = 1 + \max\{level(T_j): T_i < T_j\}$. A critical path schedule is a list schedule derived from a list having the property that for any two tasks T and S, if level(T) > level(S), then T precedes S on the list. Because the list contains the tasks ordered according to their levels, these schedules are also called level schedules. An example of a critical path schedule is given in Figure 1.5.

As noted above, critical path schedules have been studied extensively. They are of substantial practical and theoretical interest for three reasons: First, the method is intuitively appealing. Second, the method is applicable to any system having precedence constraints. Third, these schedules are easy to construct - using breadth first search the list can be constructed in time linear with the number of edges in the dag representing the precedence constraints.

1.3.3 Coffman-Graham scheduling

Coffman-Graham schedules are the third class of schedules we utilize. These schedules are a subclass of critical path schedules in which the tasks of each level are ordered in a particular way. Specifically, Coffman-Graham schedules are a class of list schedules for which the list is formed according to the following rules: Each task is assigned a label as follows:

Figure 1.5: Critical path schedules



m = 2 processors

The numbers beside the tasks are the levels of the tasks.

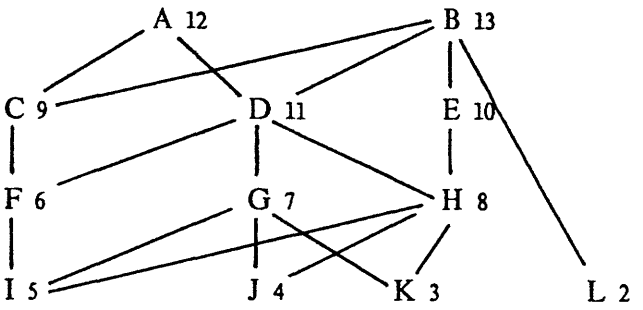
Critical path list: (A B C E D F G H I J K L M)

Schedule:

A	C	D	F	H	I	K
B	E	L	G	M	J	

Time unit: 1 2 3 4 5 6 7

Figure 1.6: Coffman-Graham schedules



m = 2 processors

The numbers beside the tasks are the Coffman-Graham labels of the tasks.

Coffman-Graham list: (B A D E C H G F I J K L M)

Schedule:

B	D	C	G	I	K	M
A	E	H	F	J	L	

Time unit: 1 2 3 4 5 6 7

1. Select a task which has no successors and assign label 1 to that task.
2. Assume that labels 1, ... , i - 1, have already been assigned. For each unlabeled task T, all of whose successors have been labeled, form a list (in decreasing order) of the labels of T's immediate successors. Assign label i to the task whose list is lexicographically the smallest.

The list used to do the scheduling contains the tasks in decreasing order of their labels. An example of a Coffman-Graham labeling and the corresponding schedule are given in Figure 1.6.

These schedules were first investigated by Coffman and Graham [CG] in conjunction with UET task systems where m=2. As we note in the next section, Coffman-Graham schedules are guaranteed to be optimal in this limited case, while list and critical path schedules are not. Since the initial work of Coffman and Graham, these schedules have been investigated by several other researchers, including Lam and Sethi, Goyal, Leung and Jaffe [LS, Go, Le, Ja]. In general, the mathematical properties of Coffman-Graham schedules make them easier to analyze than the more general case of critical path schedules. However, because Coffman-Graham schedules are a subclass of critical path schedules, certain results about Coffman-Graham schedules - in particular, lower bounds on worst case performance - can be applied to critical path schedules as well. We will make use of this relationship in Chapter 5.

1.4 A survey of major results

In the remainder of this chapter we survey the major results pertaining to the minimum execution time scheduling problem for the basic task system model and to the three types of schedules which we utilize. These results are basically of two kinds: either they are NP-completeness results, therefore implying that finding algorithms which produce optimal schedules in a reasonable amount of time is unlikely; or they are bounds on the worst case performance of list, critical path or Coffman-Graham scheduling. We first briefly review the notions of NP-completeness.

1.4.1 NP concepts

Throughout this thesis a recurrent concept is the notion of a problem being NP-complete or

NP-hard. In this section we give a brief description of these ideas. The reader is refered to the book by Garey and Johnson [GJ79] for a detailed discussion.

The set NP consists of all languages which can be recognized by a nondeterministic Turing machine in polynomial time. Similarly, the set P consists of all languages which can be recognized by a deterministic Turing machine in polynomial time. It is not known whether P is properly contained in NP. A language L_0 in NP is NP-complete is the following condition is satisfied:

> Given a deterministic algorithm of time complexity $T(n) \geq n$ for recognizing L_0, for each language L in NP, there is an effective way to find a deterministic algorithm of time complexity T(p(n)) for recognizing L, where p is a polynomial depending on L.

Clearly, if any NP-complete language is in P, then P = NP. The usual method of showing that a language L_0 is NP-complete is to show that:

1. L_0 is in NP
2. There exists an NP-complete language L, which is reducible to L_0 in deterministic polynomial time.

A language for which the second condition can be shown, but not the first is NP-hard. The recognition of such languages is at least as hard as the recognition of NP-complete languages.

Finally, we note that it is widely believed that $P \neq NP$. This belief springs from the fact that there has been an immense amount of time and energy devoted to finding a polynomial time algorithm for NP-complete problems. Moreover, it is generally acknowledged that obtaining lower bounds on time complexity are among the hardest types of results to obtain. This may help to explain why no one has been able to show $P \neq NP$, even though most researchers believe that is the case. Thus, there is strong evidence that polynomial time algorithms for obtaining solutions to NP-complete problems do not exist. This leaves us to concentrate on the performance of heuristic algorithms for these problems.

1.4.2 NP results

There are two important NP-completeness results pertaining to finding minimum length schedules for task systems.

For UET task systems with m processors, Ullman [U73, U75, U76] has shown that finding minimum length schedules is NP-complete. Lenstra and Kan [LK] have shown the same result using a different construction. A major open problem is whether this result is true for any fixed $m \geq 3$. That is, whether, for any fixed number of processors $m \geq 3$, finding minimum length schedules for UET task systems with m processors is NP-complete. As mentioned earlier, when $m = 2$, there is a polynomial time algorithm for finding minimum length schedules. Also, if the precedence constraints are restricted to a forest, then there is a polynomial time scheduling algorithm. Both of these results are given in the next section.

For task systems with unrestricted task execution times and no precedence constraints, Bruno, Coffman and Sethi [BCS] have shown that finding minimum length schedules is NP-hard even for systems with just two processors.

Finally, both Ullman [U73, U75, U76] and Lenstra and Kan [LK] have shown the following: That finding minimum execution time schedules for task systems with two processors, precedence constraints, and task execution times restricted to be either 1 or 2, is NP-complete.

1.4.3 Performance results

As evidenced by the NP-completeness results given in the previous section, for most interesting scheduling problems it is unlikely that polynomial time algorithms exist which produce optimal schedules. For this reason, most of the research attention has been on analyzing the performance of various heuristic scheduling methods. Almost all of these results involve worst case performance. That is, an upper bound is given for the ratio of the length of a schedule of a particular type (for instance, a list schedule) to the length of an optimal schedule for the same task system. In this survey we restrict our

attention to the worst case performance of list, critical path and Coffman-Graham schedules. Again we note that most useful scheduling algorithms can be formulated as algorithms which produce schedules which are a subclass of the list schedules, and that critical path and Coffman-Graham schedules have properties which make them particularly attractive, both theoretically and practically.

Many of the results which we cite are also the best possible results. This means that the result is both an upper and lower bound on the worst case ratio between the length of a schedule of the particular type and the length of an optimal schedule. That is, there exists a task system, a schedule of the particular type and an optimal schedule for that task system, such that the ratio of the schedule lengths is arbitrarily close to the upper bound.

Throughout this thesis, given a task system S, we use the following four values when citing various results:

OPT is the length of an optimal schedule for S

LIST is the maximum length of any list schedule for S

CPATH is the maximum length of any critical path schedule for S

CG is the maximum length of any Coffman-Graham schedule for S

Before actually giving any results, we note that there are two excellent references for the interested reader. Most of the major results cited in this and the previous section are given a full treatment, including proofs, in the book by Coffman [C]. Secondly, a near exhaustive listing of scheduling results for many kinds of task systems and scheduling algorithms is given in [GLLK].

The most extensive research with regard to the schedules that we are considering has been done for UET task systems. Some of the earliest work was done by Graham [G66] who showed that LIST/OPT $\leq 2 - 1/m$, and that this is the best possible result. Chen [Ch] has shown that CPATH/OPT $\leq 4/3$ if $m = 2$ and that CPATH/OPT $\leq 2 - 1/(m - 1)$ if $m \geq 3$. Each portion of this bound is the best possible. This result shows that critical path schedules have slightly better worst case behavior than do list

schedules in the general UET case. If the precedence structure is restricted to a tree, Hu [H] shows that critical path schedules are optimal.

With regard to Coffman-Graham schedules and the UET case, there are two major results. If $m = 2$, then Coffman and Graham [CG] have shown that these schedules are optimal. If $m \geq 2$, then Lam and Sethi [LS] have shown that $CG/OPT \leq 2 - 2/m$, and that this is the best possible result. An alternative method of producing optimal schedules when $m = 2$ is given by Fujii, Kasami and Ninomiya [FKN]. This method is based on maximal matchings and has not been generalized for systems with more than two processors.

With respect to task systems with no precedence constraints and arbitrary execution times, there are several interesting results pertaining to list scheduling. Graham [G66] has shown that in this instance, $LIST/OPT \leq 2 - 1/m$, and that this is the best possible result. This is exactly the same bound as was given for LIST/OPT in the UET case. In fact, Graham [G66] has shown that this same bound holds, even for task systems with <u>both</u> precedence constraints and unrestricted task execution times. Graham [G69] has also shown the following result which explicitly incorporates the task execution times: $LIST/OPT \leq 1 + (m - 1)[\max\{\tau_i : T_i \in T\}]/(\Sigma_{T_i \in T} \tau_i)$. Note that both Coffman-Graham and critical path schedules are equivalent to list schedules in this context because there are no precedence constraints. There are, however, a number of other types of schedules which have been studied for this submodel. Most of these are subclasses of list schedules in which the tasks are ordered in the list based on the task execution times. Again the reader is refered to [C] and [GLLK] for a thorough treatment.

1.5 Extensions

For many practical applications the basic task system model presented here has proven to be insufficient. For this reason, and out of theoretical curiosity, a number of extensions to the basic task system model have been investigated.

One major area of research in this regard has been the study of preemptive scheduling. In this

extension, a task may be interrupted during its execution and then continued later in the schedule. For UET systems, this produces no new results, however for systems where task execution times are not restricted, this is an interesting and powerful extension. A large number of results have been obtained on preemptive scheduling, many of them analogous to the results cited in the previous section. Most of these results may be found in [C] and [GLLK].

Other extensions to the basic model include the following: Liu and Liu [Li] and Jaffe [Ja] have investigated task systems with processors of different types - each task specifies the type of processor that it must execute on. Ibarra and Kim [IK], Kafura and Shen [KS] and Jaffe [Ja] have investigated task systems where the processors have different speeds. Lloyd has studied UET task systems where each task may require more than one processor during its execution. These results are presented in Chapter 8. Finally, a number of researchers have investigated task systems with resources. These systems are the main focus of this thesis.

Chapter 2 - Task Systems With Resources

For many practical scheduling problems the basic task system model presented in Chapter 1 is inadequate. For these problems, the performance bounds for the basic model are neither accurate nor informative. Intuitively, the basic model does not take enough of the parameters of these problems into consideration to provide good bounds. For instance, consider the following three scheduling problems:

1. A computer system has associated with it, in addition to processors, several types of resources, including memory, disk drives and printers. In general, there is a set of jobs to be executed on the system, and, depending on the circumstances, there may or may not be precedence constraints associated with these jobs. Each job has certain requirements with respect to the resources of the system. For example, a job may require 20K of memory, two disk drives and a printer. The problem is to produce a schedule for executing this set of jobs in a minimum amount of time. Clearly, for such a schedule to be valid, the demand of the jobs executing at any given time, for each resource, should not exceed the available quantity of the resource.
2. A large construction company possesses a certain amount of equipment: bulldozers, trucks, cranes, etc. In addition, the company has a number of employees. Together the equipment and the employees constitute the resources of the company. In general, there is a set of construction projects for the company to complete. Each project requires certain pieces of equipment and numbers of people. Here again, the problem is to produce a schedule for completing the projects in a minimum period of time, given the resources of the company.
3. An idealized bin packing problem is the following: Given a set of items and a set of bins, pack the items into a minimum number of bins. The items are of identical size and shape, although they may vary in other parameters - for instance, in weight and cost. The bins are identical in all respects. In addition to having a fixed size and shape, the bins have fixed capacities with respect to

the other parameters of the items. For example, there may be limits on the total weight and total cost of the items packed into any single bin. In addition, there may or may not be a limit on the total number of items that can be packed into any single bin. The problem is to pack the items into a minimum number of bins without violating the capacity constraints of the bins.

The outstanding feature of each of these problems is the presence of a resource constraint. These constraints are sufficiently powerful, that it is unreasonable to expect that using the basic task system model for analyzing the performance of scheduling alogrithms for these problems will provide useful results. The power of these constraints can, however, be captured by extending the basic task system model to include a set of resources. Each task may require some or all of the resources during its execution. Such a task system with resources can be used to effectively model each of the three problems outlined above, although for problem 2 and possibly for problem 3, there is no processor constraint. We will return to the question of processor constraints in a later section.

In the remainder of this thesis we deal exclusively with task systems with resources. Depending on the exact nature of the problem under consideration, there are two alternative formal models of task systems with resources that may be utilized. In the next two sections we examine those two models.

2.1 Task systems with continuous resources

In this section we examine task systems with continuous resources. This model has been used to obtain almost all performance bounds for the scheduling of task systems with resources to date.

2.1.1 The model

A UET task system with continuous resources is a system $S = \langle T, <, m, s \rangle$ where:

1. $T = \{T_1, \ldots, T_n\}$ is a set of tasks - associated with T_i is a positive integral execution time τ_i.
2. $<$ is a partial order specifying precedence constraints between the tasks.
3. There are m identical processors.
4. s is the number of different resources. It is assumed that $s \geq 1$, that there is exactly one unit of

each resource, and that each task may require any portion of that one unit for each resource. For each task T_i and each resource v, $R_v(T_i) \in [0, 1]$ specifies the portion of resource v required by task T_i during its execution. Because a task may require any portion of each resource (all, none, 1/2, or .000001, for instance) we say that the resources are continuous.

A valid schedule for a task system with continuous resources S, is a mapping $\sigma: T \rightarrow (N - \{0\})$ such that:

1. For all $l \in (N - \{0\})$, $m \geq |\{T_i \in T: \sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1\}|$.
2. If $T_i < T_j$, then $\sigma(T_i) + \tau_i - 1 < \sigma(T_j)$.
3. For all $l \in (N - \{0\})$, and v, $1 \leq v \leq s$, $1 \geq \Sigma R_v(T_i)$ summing over all T_i such that $\sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1$.

This definition is identical to the one for basic task systems, except for condition 3. This last condition insures that at any given time unit, the currently executing tasks do not require more than one unit of each resource.

Intuitively, a list schedule for a task system with continuous resources may be constructed as follows: Initially, let L be any (ordered) list of the tasks in T. The tasks are scheduled as follows: Whenever a processor becomes idle, the list L is instantaneously scanned from its beginning and the first task T (if any) which meets the following criteria is removed from L and assigned to the idle processor: 1. Each task T_j such that $T_j < T$, has completed execution and 2. If $[r_1, \ldots, r_s]$ represents the total resource requirements of all currently executing tasks, then for each resource v, $r_v + R_v(T) \leq 1$. This last requirement guarantees that the currently executing tasks do not require more than a total of one unit for any resource. More formally, a list schedule for a task system with continuous resources is a valid schedule which is generated as follows:

1. Initially, L is an (ordered) list of the tasks in T and l is 1.

2. While L is nonempty perform this step

   a. Let $k = |\{T_i \notin L : \sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1\}|$

   b. For each v, $1 \leq v \leq s$, let $r_v = \Sigma R_v(T_i)$ summing over all T_i such that

      $\sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1$

   c. Let L' be a list of the ready tasks on L at time l, the tasks in the same order on L' as on L

   d. While L' is nonempty and $k < m$ perform this step

      i. Let T be the first task on L'

      ii. If for each v, $1 \leq v \leq s$, $r_v + R_v(T) \leq 1$,

         then let $\sigma(T) = l$, let $k = k+1$, for each v, let $r_v = r_v + R_v(T)$, and remove T from L

      iii. Remove T from L'

   e. Let $l = 1 + \min \{\sigma(T_i) + \tau_i - 1 : T_i \notin L \text{ and } \sigma(T_i) + \tau_i - 1 \geq l\}$

An example of a task system with continuous resources and a list schedule for that system is given in Figure 2.1.

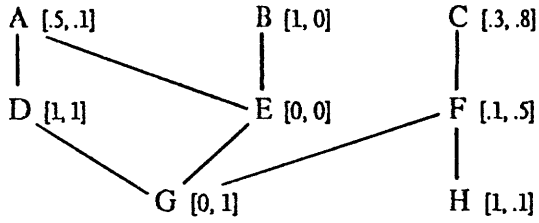
We note that critical path and Coffman-Graham schedules retain their original definitions of being particular subclasses of list schedules.

2.1.2 Shortcomings

There are two major shortcomings of the task system with continuous resources model.

First, the assumption that the resources are "continuous" is not an accurate reflection of either existing computer systems or of many industrial scheduling problems. In those instances, resources are much more "discrete" in nature than they are "continuous". For instance, computing resources such as tape drives and line printers are generally available only in small quantities and a task can require only whole units of them. Moreover, while memory may be thought of as being continuous due to its large size, it is debatable whether memory should even be viewed as a limiting resource in terms of practical

Figure 2.1: Example of a task system with continuous resources



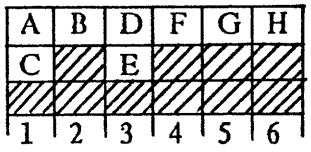
m = 3 processors

2 continuous resources

The resource requirements of the tasks are given in a vector beside the task.
Each task has an execution time of one.

List: (A B C D E F G H)

Schedule:



computation.

Second, the performance bounds that have been obtained for various heuristics with respect to the continuous resources model, depend on the number of different resources, but not on the actual number of discrete units of each resource. For systems in which the available quantities of the resources are small, the actual worst case performance of various heuristics may be much better than these bounds indicate.

2.2 Task systems with discrete resources

To try to overcome the perceived shortcomings of the task systems with continuous resources model, we consider a model of task systems with discrete resources - there is a fixed number of indivisible units of each resource which tasks may require during execution.

2.2.1 The model

A task system with discrete resources is a system $S = \langle T, <, m, s \rangle$ where:

1. $T = \{T_1, .. , T_n\}$ is a set of tasks - associated with T_i is a positive integral execution time τ_i.
2. $<$ is a partial order specifying precedence constraints between the tasks.
3. There are m identical processors.
4. s is the number of different resources. It is assumed that $s \geq 1$, that there are r_i indivisible units of resource i, and that a task may require only integral numbers of these units for each resource.

For each task T_i and each resource v, $R_v(T_i)$ specifies the number of units of resource v required by task T_i during its execution. Because a task may require only integral numbers of units of each resource, we say that the resources are discrete.

A valid schedule for a task system with discrete resources S, is a mapping $\sigma: T \rightarrow (N - \{0\})$ such that:

1. For all $l \in (N - \{0\})$, $m \geq |\{T_i \in T: \sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1\}|$.
2. If $T_i < T_j$, then $\sigma(T_i) + \tau_i - 1 < \sigma(T_j)$.
3. For all $l \in (N - \{0\})$, and v, $1 \leq v \leq s$, $r_v \geq \Sigma R_v(T_i)$ summing over all T_i such that

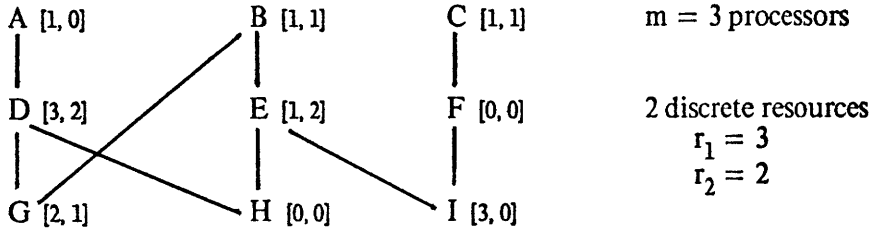
$$\sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1.$$

This definition is identical to the one for basic task systems, except for condition 3. This last condition insures that at any given time unit, the currently executing tasks do not require more than the existing number of units of each resource.

Intuitively, a list schedule for a task system with discrete resources may be constructed as follows: Initially, let L be any (ordered) list of the tasks in T. The tasks are scheduled as follows: Whenever a processor becomes idle, the list L is instantaneously scanned from its beginning and the first task T (if any) which meets the following criteria is removed from L and assigned to the idle processor: 1. Each task T_j such that $T_j < T$, has completed execution and 2. If $[r'_1, \ldots, r'_s]$ represents the total resource requirements of all currently executing tasks, then for each resource v, $r'_v + R_v(T) \leq r_v$. More formally, a list schedule for a task system with discrete resources is a valid schedule which is generated as follows:

1. Initially, L is an (ordered) list of the tasks in T and l is 1.
2. While L is nonempty perform this step
   a. Let $k = |\{T_i \notin L : \sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1\}|$
   b. For each v, $1 \leq v \leq s$, let $r'_v = \Sigma R_v(T_i)$ summing over all T_i such that
      $\sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1$
   c. Let L' be a list of the ready tasks on L at time l, the tasks in the same order on L' as on L.
   d. While L' is nonempty and $k < m$ perform this step
      i. Let T be the first task on L'
      ii. If for each v, $1 \leq v \leq s$, $r'_v + R_v(T) \leq r_v$,
         then let $\sigma(T) = l$, let $k = k+1$, for each v, let $r'_v = r'_v + R_v(T)$ and remove T from L
      iii. Remove T from L'
   e. Let $l = 1 + \min \{\sigma(T_i) + \tau_i - 1 : T_i \notin L \text{ and } \sigma(T_i) + \tau_i - 1 \geq l\}$

Figure 2.2: Example of a task system with discrete resources



Each task has an execution time of one.

List: (I H G F E D C B A)

Schedule:

C	F	D	I	G
B	E		H	
A				

Time unit: 1 2 3 4 5

An example of a task system with discrete resources and a list schedule for that system is given in Figure 2.2.

We note that critical path and Coffman-Graham schedules retain their original definitions of being particular subclasses of list schedules.

2.2.2 Discussion

We are not the first to consider task systems with discrete resources. The original formulation of task systems with resources by Garey and Graham [GG73, GG75] involved discrete resources. Moreover, an NP-completeness result of Ullman [U76] involves discrete resources. However, as far as performance bounds are concerned, almost all of the previous work has been done for systems with continuous resources. The only results pertaining to the discrete model are some limited results of Goyal [Go] and Leung [Le] involving systems with 0-1 resources. These are systems with exactly one indivisible unit of each resource. A task either requires all of a resource or none of it.

As noted earlier, the discrete resources approach is designed to overcome the perceived shortcomings of the continuous resources approach. The performance bounds for systems with discrete resources will incorporate the values $r_1, \ldots, r_s$ (these are the number of units of each resource). This means that the performance bounds will distinguish between task systems with different numbers of the same resource, unlike in the continuous resources case. They will also be able to indicate the effect on performance, if additional units of an existing resource are added to the system.

In the remainder of this chapter, we survey the NP-completeness results involving task systems with resources (discrete and continuous) and discuss the role of processors in this model.

2.3 Why study heuristics?

In our discussion of basic task systems in the previous chapter we mentioned several NP-completeness results regarding the minimum execution time scheduling of those systems. As might be expected, much the same results exist for task systems with resources. In this case however, the results

are more definitive than for basic task systems. Ullman [U76] has shown that finding minimum execution time schedules for UET task systems with discrete resources is NP-complete, even for systems with only two processors, one discrete resource with one unit (and arbitrary precedence constraints). For continuous resources, Garey and Johnson [GJ74] show that finding minimum execution time schedules is NP-complete for UET task systems with three processors, one continuous resource, and no precedence contraints. They also show [GJ74] that finding minimum execution time schedules is an NP-complete problem for UET task systems with two processors, one continuous resource and precedence constraints restricted to a forest.

From the above results we can conclude that for virtually all interesting scheduling problems for task systems with resources, it is unlikely that polynomial time alogrithms exist which produce optimal schedules. This leaves the study of heuristic algorithms for scheduling. In this thesis we examine list and critical path schedules. As noted in Chapter 1, these are the two simplest and most intuitive scheduling heuristics for UET systems. We will not be particularly concerned with Coffman-Graham scheduling, except in one instance where we use it to get a lower bound on the worst case performance of critical path scheduling. The reason for this lack of intense interest in Coffman-Graham scheduling is that, particularly when dealing with extensions of the basic task system model, experience has shown that the difference in the worst case performance of critical path and Coffman-Graham scheduling is very small relative to the worst case bound. Because this difference is so small, the analysis of the performance of both critical path schedules and Coffman-Graham schedules is of little or no practical interest.

2.4 The processors question

In both of the models of task systems with resources we study, there is a set of m processors. The role that these processors should play in this model is a serious question, both theoretically and practically. There are two distinct schools of thought on this issue.

One approach is to assume that the processors play no role in constraining the schedule. In this

case, it is assumed that the number of processors is at least as large as the number of tasks in the system (i.e. $m \geq n = |\mathcal{T}|$). This assumption means that given any time units B_i, B_j with $j > i$, and any task $T \in B_j$, the reason that T did not execute in B_i is due to either a resource constraint or a precedence constraint. It is not the case that B_i was "full", which would mean that there was "no room" for T in B_i. As far as performance bounds are concerned under this assumption, it is as if processors never appeared in the model at all. The quantity m plays no role in the bounds for task systems with no processor constraint. For certain applications, this is a reasonable assumption - for instance, applications 2 and 3 that were discussed at the beginning of this chapter: In the scheduling problem for a construction company given there, there was nothing corresponding to a processor constraint. In the bin packing problem it was noted that there may or may not be a limit on the number of items placed into any single bin (such a limit corresponds to a processor constraint). Much of the previous work on performance bounds for task systems with resources has been on systems without a processor constraint.

The second approach to the role of processors in the task system with resources model is that the processors are vital in determining worst case performance, and that many applications demand a model involving processors. Even so, it can be argued that no generality is lost by using a "no processor constraint" approach, since processors can be treated as just another resource. That is, given a performance bound for systems with no processor constraint, and a task system with s resources and a processor constraint, simply apply the bound as if the system had $s+1$ resources. However, from an intuitive viewpoint, this approach is suspect, since processors are not "just another resource". The processor resource possesses certain characteristics that are not shared by resources in general. In particular, every task requires exactly one unit of the processor resource - no more and no less. Furthermore, with respect to task systems with continuous resources, the processor resource is unique in that a task may not require just any portion of the resource, as was assumed for continuous resources in general. At least intuitively, there is no reason to believe that treating the processors as an additional

kind of resource will result in meaningful worst case bounds.

2.5 The problems to be studied

In this thesis we study minimum execution time scheduling of UET task systems with resources. We examine the following four models:

UET task systems with continuous resources and no processor constraint

UET task systems with continuous resources and a processor constraint

UET task systems with discrete resources and no processor constraint

UET task systems with discrete resources and a processor constraint

We investigate the worst case performance of list and critical path scheduling for each of these models. We also compare the bounds for the four models and try to delineate the relationships between those bounds.

Chapter 3 - List Scheduling

In this chapter we study the list scheduling of UET task systems with resources. As noted in the last chapter, list schedules are the fundamental type of schedule which we consider, and most scheduling algorithms produce classes of schedules which are subclasses of the list schedules. Moreover, no generality is lost by restricting our attention to list schedules when dealing with UET task systems, because there is always a list schedule of optimal length.

For comparison purposes, we again mention the following two results on the worst case performance of list scheduling for basic UET task systems (i.e. systems without any resources). If there is no processor constraint ($m \geq n$) then all list schedules are optimal. That is, LIST/OPT = 1. If there is a processor constraint ($m \geq 2$) then LIST/OPT $\leq 2 - 1/m$, and this is the best possible result [G66].

3.1 Continuous resources

The major work on list scheduling for UET task systems with continuous resources is by Garey, et.al. [GGJY]. They show for a system with no processor constraint ($m \geq n$), that LIST/OPT $\leq s \cdot OPT/2 + s/2 + 1$, and that systems exist for which LIST/OPT $\geq s \cdot OPT/2 + s/2 + 1 - 2s/OPT$. This upper bound can be compared to the corresponding result for UET task systems with no resources. That comparison shows that adding even a single continuous resource to a UET task system results in a tremendous degradation of the worst case behavior of list scheduling. That is, for a UET task system without resources, list schedules are always optimal, whereas the addition of a single continuous resource can result in list schedules having length quadratic in the length of an optimal schedule. This comparison confirms our earlier comments that performance bounds based on the basic model are probably not good indicators of performance for problems involving resources.

For UET task systems with continuous resources and a processor constraint, there are no tight upper bounds. There are, however, two partial results. First, is the result of Garey, et.al. [GGJY] cited above,

using s+1 resources instead of s - the extra resource accounting for the existence of the processor constraint. This yields LIST/OPT $\leq$ (s+1)·OPT/2 + s/2 + 3/2. Second, Yao [Y] has shown that LIST/OPT $\leq$ min{m, (m-1)s·OPT/(2m) + 7(m-1)s/(2m) + 1}. As mentioned above, neither of these results is best possible.

3.2 Discrete resources

In this section we state and prove worst case performance bounds for the list scheduling of UET task systems with discrete resources. The only previous work for these systems is by Goyal [Go] and Leung [Le]. Goyal investigated UET task systems with one discrete resource, where $r_1 = 1$ (there is exactly one unit of that one resource, so each task either requires all of the resource or none of it). He shows for systems with no processor constraint ($m \geq n$), that LIST/OPT ≤ 2, and for systems with processor constraints ($m \geq 2$), that LIST/OPT $\leq 3 - 2/m$. Moreover, both of these results are the best possible. Comparing these bounds to those for UET task systems without resources, we note that the addition of one unit of one discrete resource caused the worst case ratio of LIST to OPT to increase by 1 in the no processor constraint case, and by 1 - 1/m for systems with a processor constraint. Leung investigated UET task systems with discrete resources in which each $r_i = 1$, under the restriction that each task may require at most one unit of resource (i.e. for each task T, $\Sigma_{i=1}^{s} R_i(T) \leq 1$). He showed that LIST/OPT $\leq$ min{m, (2-1/m) + s(1-1/m)}, and that this is the best possible result. Our results generalize the results of Goyal and Leung.

3.2.1 Two results

We prove the following two results about the worst case performance of list scheduling for UET task systems with discrete resources:

Theorem 3.1: If $m \geq n$ (no processor constraint), then LIST/OPT $\leq 1 + r$, where $r = \Sigma_{i=1}^{s} r_i$.

Moreover, this bound is the best possible.

Theorem 3.2: If $m \geq 2$ (a processor constraint), then LIST/OPT $\leq \min\{m, (2\text{-}1/m) + r(1\text{-}1/m)\}$, where $r = \Sigma_{i=1}^{s} r_i$. Moreover, this bound is the best possible.

These results are proven in the next two sections. Before doing so, however, there are several remarks to be made about these two theorems.

First, note the surprising role played by the resources in determining the worst case bound. The relevant quantity is not the number of different resources, but rather is the sum total of all the units of all the resources in the system. The number of different resources and the distribution of the r units of resource among those different resources is no factor. This means that the worst case bound for LIST/OPT is the same for a system with 1000 units of one resource as it is for a system with one unit of each of 1000 resources. This contrasts sharply with the results for UET task systems with continuous resources, where the key parameter is s, the number of different resources.

Second, these bounds indicate that for each unit of (any) resource added to a UET task system, the worst case ratio of LIST to OPT increases by 1 in the no processor constraint case, and by 1 - 1/m in the processor constraint case. This follows, because for $r = 0$, our results are identical to those cited in the introduction to this chapter as the best possible bounds for LIST/OPT for basic UET task systems (i.e. without resources). These results provide a clear indication of the role of the resources in determining worst case behavior.

Third, unlike the situation for UET task systems with continuous resources, there is a tight upper bound for UET task systems with discrete resources and a processor constraint. For that result, we note that the bound of m holds for every $r \geq m - 1$. This indicates the point at which the processor constraint dominates the resource constraint with respect to the worst case performance of list scheduling.

3.2.2 The upper bounds

In this section and the next we prove Theorems 3.1 and 3.2 - the upper bounds in this section and the lower bounds in the next. In both sections we concentrate on the proof of Theorem 3.2 - the result for

systems with a processor constraint. We do this because those results are slightly more complicated (due to the presence of the processor constraint) than those for Theorem 3.1. At the end of each section we briefly indicate how to modify those results to obtain the results for the no processor constraint case.

Lemma 3.1: If $m \geq 2$ (a processor constraint), then LIST/OPT $\leq \min\{m, (2-1/m) + r(1-1/m)\}$, where

$$r = \Sigma_{i=1}^{s} r_i.$$

Proof

Assume that a UET task system with discrete resources is given. We prove the result by obtaining a lower bound on OPT, and an upper bound on LIST. Combining these bounds gives an upper bound for LIST/OPT.

We make use of the following notation throughout the proof: Let k be the length of a critical path in a directed acyclic graph representing the precedence constraints, and for each resource i, let $x_i = \Sigma\ R_i(T_j)$ summing over all $T_j \in T$. That is, x_i is the total demand for resource i among all of the tasks in the system.

Consider an optimal schedule for the system. Three observations can be made: First, an optimal schedule can be no shorter than k, the length of a critical path. Second, an optimal schedule can do no better than to have tasks executing on each of the processors during each time unit. Third, for each resource, an optimal schedule can do no better than to have all units of that resource utilized during each time unit. Thus, OPT $\geq \max\{k, n/m, x_1/r_1, \ldots, x_s/r_s\}$.

Now consider an arbitrary list schedule for the system. Such a schedule consists of two types of time units: Those in which all processors have tasks executing on them, and those in which at least one processor is idle. The number of time units with idle processors may be bounded above as follows: Whenever a processor is idle during a time unit, each unexecuted task, T, is prevented from executing on that processor for one of two reasons: Either a predecessor of T has not yet executed, or, for some resource j, the demand for resource j by tasks executing during that time unit, together with the

demand for resource j by task T, exceeds r_j. It is well known that there can be at most k time units in which only the first constraint prevents tasks from executing. Moreover, at each time unit where the second constraint prevents some task from executing, at least one unit of some resource must be required by some task executing in that time unit. Hence, there are at most $\Sigma_{i=1}^{s} x_i$ time units in which there is an idle processor due, in part, to the second constraint. Thus,

$LIST \leq k + \Sigma_{i=1}^{s} x_i + (n-k-\Sigma_{i=1}^{s} x_i)/m = n/m + (1-1/m)k + (1-1/m)\Sigma_{i=1}^{s} x_i$.

$\therefore$ $LIST/OPT \leq [n/m + (1-1/m)k + (1-1/m)\Sigma_{i=1}^{s} x_i] / \max\{k, n/m, x_1/r_1, \ldots, x_s/r_s\}$

$\leq (2-1/m) + (1-1/m)\Sigma_{i=1}^{s} r_i$

$= (2-1/m) + r(1-1/m)$

Finally, note that for all m, $LIST/OPT \leq m$, since

1. A list schedule cannot have a time unit in which all processors are idle unless the schedule has completed.
2. There are at most m·OPT tasks in the entire task system.

$\therefore$ $LIST/OPT \leq \min\{m, (2-1/m) + r(1-1/m)\}$ □

Lemma 3.2: If $m \geq n$ (no processor constraint), then $LIST/OPT \leq 1 + r$, where $r = \Sigma_{i=1}^{s} r_i$.

Proof

First note that since $m \geq n$, each time unit of any schedule can be treated as having at least one idle processor. Then, analogously to the proof of Lemma 3.1, we can show that

$OPT \geq \max\{k, x_1/r_1, \ldots, x_s/r_s\}$ and $LIST \leq k + \Sigma_{i=1}^{s} x_i$.

$\therefore$ $LIST/OPT \leq [k + \Sigma_{i=1}^{s} x_i] / \max\{k, x_1/r_1, \ldots, x_s/r_s\}$

$\leq 1 + \Sigma_{i=1}^{s} r_i$

$= 1 + r$. □

3.2.3 The lower bounds

In this section we prove that the upper bounds for LIST/OPT given in the previous section are the best possible upper bounds for the worst case performance of list scheduling for UET task systems with discrete resources.

Lemma 3.3: If $m \geq 2$ (a processor constraint), then LIST/OPT $\leq \min\{m, (2\text{-}1/m) + r(1\text{-}1/m)\}$, where $r = \Sigma_{i=1}^{s} r_i$, is the best possible bound.

Proof

We show that for any number of processors m, and any distribution of r units of resource, that the ratio LIST/OPT can be arbitrarily close to $\min\{m, (2\text{-}1/m) + r(1\text{-}1/m)\}$. We let $r = \Sigma_{i=1}^{s} r_i$, where r_i is the number of units of resource i. We assume that each r_i is nonzero, and that r does not exceed m - 1. Now, let z be a multiple of m and consider a task system consisting of the following tasks:

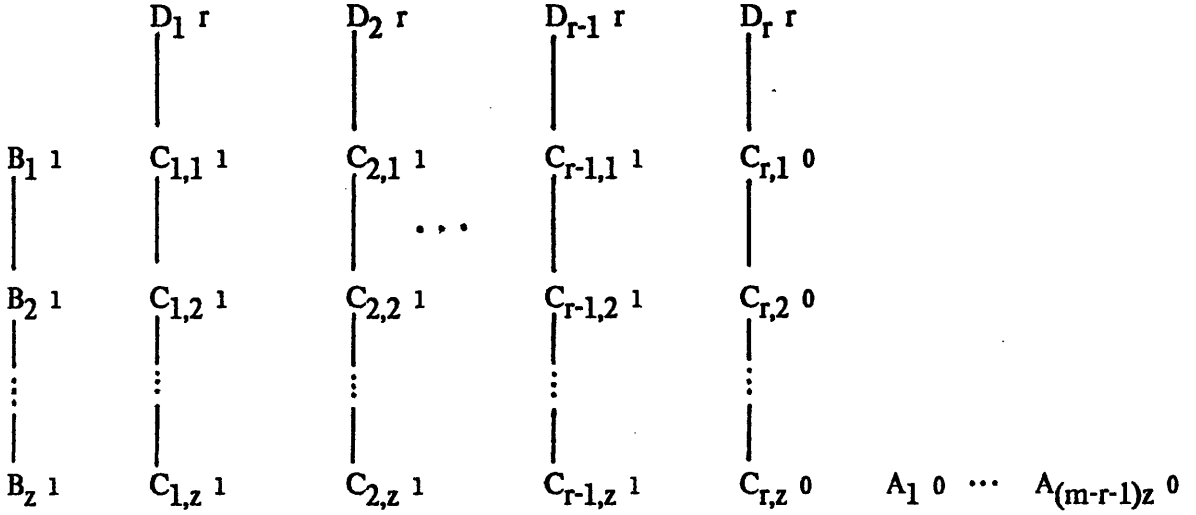
1. Tasks $A_1, \ldots, A_{(m-r-1)z}$ where each A_i requires no resources.
2. Tasks $B_1, \ldots, B_z$ where $B_i \prec B_{i+1}$ for $1 \leq i \leq z - 1$ and where each B_i requires one unit of resource s and 0 units of all other resources.
3. For each resource v, $1 \leq v \leq s$, there are tasks $D_1^v, \ldots, D_{r_v}^v$, each of which requires all the units of all resources, and tasks C_{ij}^v for $1 \leq i \leq r_v$ and $1 \leq j \leq z$, each of which requires one unit of resource v and 0 units of all other resources. The exception is that tasks $C_{r_s,1}^s, \ldots, C_{r_s,z}^s$ require no resources. Furthermore, for each v and i, $1 \leq i \leq r_v$, $D_i^v \prec C_{i1}^v \prec C_{i2}^v \prec \ldots \prec C_{iz}^v$. Such a sequence of tasks will be referred to as the D_i^v chain.

An example of such a task system for the case of $s = 1$ is shown in Figure 3.1.

An optimal schedule for this UET task system with discrete resources has length OPT = z + r. In this schedule the D-tasks execute in the first r time units, and the C-tasks, B-tasks and A-tasks execute in the next z time units. During each of these z time units, r C-tasks, one B-task and m-r-1

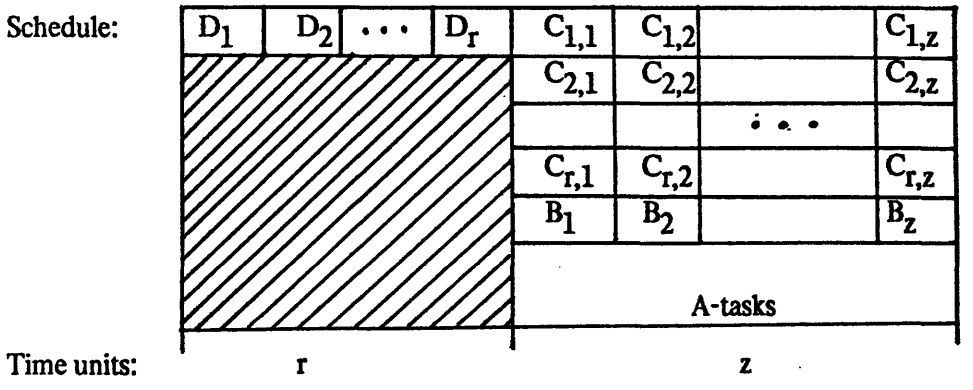
Figure 3.1: Task system used in Lemma 3.3.

Assume that s = 1, hence r = r_1. There are no processor constraints.



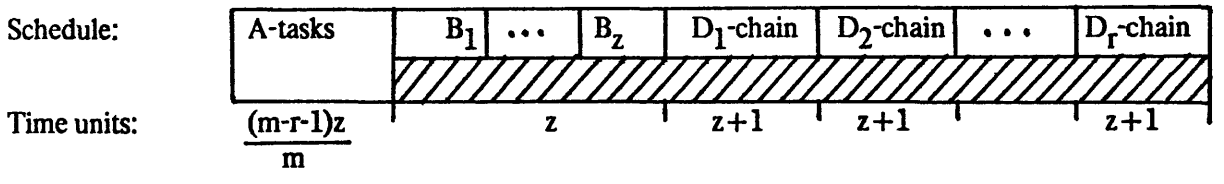
The resource requirements are given beside each task.

Figure 3.2: An optimal schedule



Length = r + z

Figure 3.3: A "bad" list schedule



Length = [2 - 1/m + r(1 -1/m)]z + r

A-tasks execute. Moreover, all units of each resource are used during each of these z time units. Figure 3.2 shows an optimal schedule for the task system given in Figure 3.1. Note that an optimal schedule can be generated from the list (D-tasks, C-tasks, B-tasks, A-tasks). Such a list schedule will be identical to the one described here except that some of the A-tasks will execute with the D-tasks instead of with the B-tasks and C-tasks.

Now consider the list (A-tasks, B-tasks, D_1^1-chain, ... , $D_{R_1}^1$-chain, D_1^2-chain, ... , $D_{R_s}^s$-chain). In this schedule, the A-tasks execute in the first (m-r-1)z/m time units. All m processors are utilized during these time units. The B-tasks execute in the next z time units. Since each D-task requires all the units of each resource, none of the D-tasks or C-tasks execute with the B-tasks. Finally, the D_i^v-chains execute, one chain at a time. The execution of each chain requires z+1 time units. Thus, this schedule has length LIST = (m-r-1)z/m + z + (z+1)r = (2-1/m + r(1-1/m))z + r. Figure 3.3 shows such a list schedule for the task system given in Figure 3.1.

$\therefore$ LIST/OPT = [(2-1/m + r(1-1/m))z + r] / (z + r)

$\therefore$ $\text{limit}_{z \to \infty}$ LIST/OPT = (2-1/m) + r(1-1/m).

Finally, if r > m - 1 then the bound of m for LIST/OPT can be approached by considering a system with the same set of tasks as if r = m - 1, with the same resource requirements as if r = m - 1. □

Lemma 3.4: If $m \geq n$ (no processor constraint), then LIST/OPT $\leq 1 + r$, where $r = \Sigma_{i=1}^{s} r_i$, is the best possible bound.

Proof

We show that for any distribution of r units of resources that the ratio LIST/OPT can be arbitrarily close to 1 + r, assuming that there is no processor constraint. We let $r = \Sigma_{i=1}^{s} r_i$, where r_i is the number of units of resource i. We assume that each r_i is nonzero. Now, let z be an arbitrary integer and consider a task system consisting of the following tasks:

1. Tasks $B_1, \ldots, B_z$ where $B_i \prec B_{i+1}$ for $1 \leq i \leq z - 1$ and where each B_i requires one unit of resource s and 0 units of all other resources.

2. For each resource v, $1 \leq v \leq s$, there are tasks $D^v_1, \ldots, D^v_{r_v}$, each of which requires all the units of all resources, and tasks C^v_{ij} for $1 \leq i \leq r_v$ and $1 \leq j \leq z$, each of which requires one unit of resource v and 0 units of all other resources. The exception is that tasks $C^s_{r_s,1}, \ldots, C^s_{r_s,z}$ require no resources. Furthermore, for each v and i, $1 \leq i \leq r_v$, $D^v_i \prec C^v_{i1} \prec C^v_{i2} \prec \ldots \prec C^v_{iz}$.

This task system is identical to the task system described in the proof of Lemma 3.3, except that there are no A-tasks. Similarly to that result, an optimal schedule for this UET task system with discrete resources can be generated from the list (D-tasks, C-tasks, B-tasks). This schedule has length OPT = z + r. Also similarly to the proof of Lemma 3.3, consider the list (B-tasks, D^1_1-chain, ..., $D^1_{R_1}$-chain, D^2_1-chain, ..., $D^s_{R_s}$-chain). The schedule generated from this list has length LIST = z + (z + 1)r = (1 + r)z + r.

$\therefore$ LIST/OPT = [(1 + r)z + r] / (z + r)

$\lim_{z \to \infty}$ LIST/OPT = 1 + r. □

Chapter 4 - Critical Path Scheduling - Continuous Resources

In this chapter we study critical path scheduling of UET task systems with continuous resources. As noted earlier, critical path schedules are a widely studied subclass of list schedules. For comparison purposes we again mention the following two results on the worst case performance of critical path scheduling for basic UET task systems (i.e. systems without any resources). If there is no processor constraint ($m \geq n$) then critical path schedules are optimal. That is CPATH/OPT = 1. If there is a processor constraint ($m \geq 2$) then CPATH/OPT $\leq 4/3$ if $m = 2$, and CPATH/OPT $\leq 2 - 1/(m-1)$ if $m \geq 3$. These are the best possible bounds [Ch].

4.1 No processor constraint

The major work to date on critical path scheduling for UET task systems with continuous resources is by Garey, et.al. [GGJY]. They show for a system with no processor constraints, that CPATH/OPT $\leq 1 + 17s/10$, and that this is the best possible result. This result can be compared to the corresponding result for UET task systems with no resources (that result is CPATH/OPT = 1). That comparison shows that for every continuous resource added to a UET task system, the worst case bound for CPATH/OPT increases by 17/10. This result can also be compared to that for list scheduling of UET task systems with continuous resources and no processor constraint. That comparison shows that the worst case behavior of critical path schedules is far better than that of list schedules for these systems - in the worst case, CPATH grows linearly with OPT, while LIST grows quadratically with OPT. This contrasts sharply with the relationship between LIST and CPATH for UET task systems without resources and no processor constraints, where both types of schedules are always optimal.

4.2 A processor constraint

For critical path scheduling of UET task systems with continuous resources and a processor constraint, there are only two limited results (aside from our work). First, Yao [Y] has shown that

CPATH/OPT $\leq$ min{ m, 2 + 2s - (2s+1)/m }. Second, the result of Garey, et.al. [GGJY] given in the previous section can be applied using s+1 resources (the extra resource accounting for the processor constraint) yielding CPATH/OPT $\leq$ 27/10 + 17s/10. In general, neither of these results is the best possible. In the remainder of this section we prove the following result about critical path scheduling of UET task systems with continuous resources:

Theorem 4.1: If m $\geq$ 2 (a processor constraint), then

$$\text{CPATH/OPT} \leq \begin{cases} m & \text{if } 2 \leq m < s+1 \\ (s+m+1)/2 & \text{if } s+1 \leq m < 2s+1 \\ (4s+m+3)/4 & \text{if } 2s+1 \leq m < 8s/3+1 \\ (14s+m+9)/10 & \text{if } 8s/3+1 \leq m < 3s+1 \\ 2+17s/10-(3s+1)/m & \text{if } 3s+1 \leq m \text{ and } m \geq 10 \\ 2+5s/3-(8s/3+1)/m & \text{if } 3s+1 \leq m \text{ and } m < 10 \end{cases}$$

Moreover, each portion of this bound is the best possible.

4.2.1 An interpretation

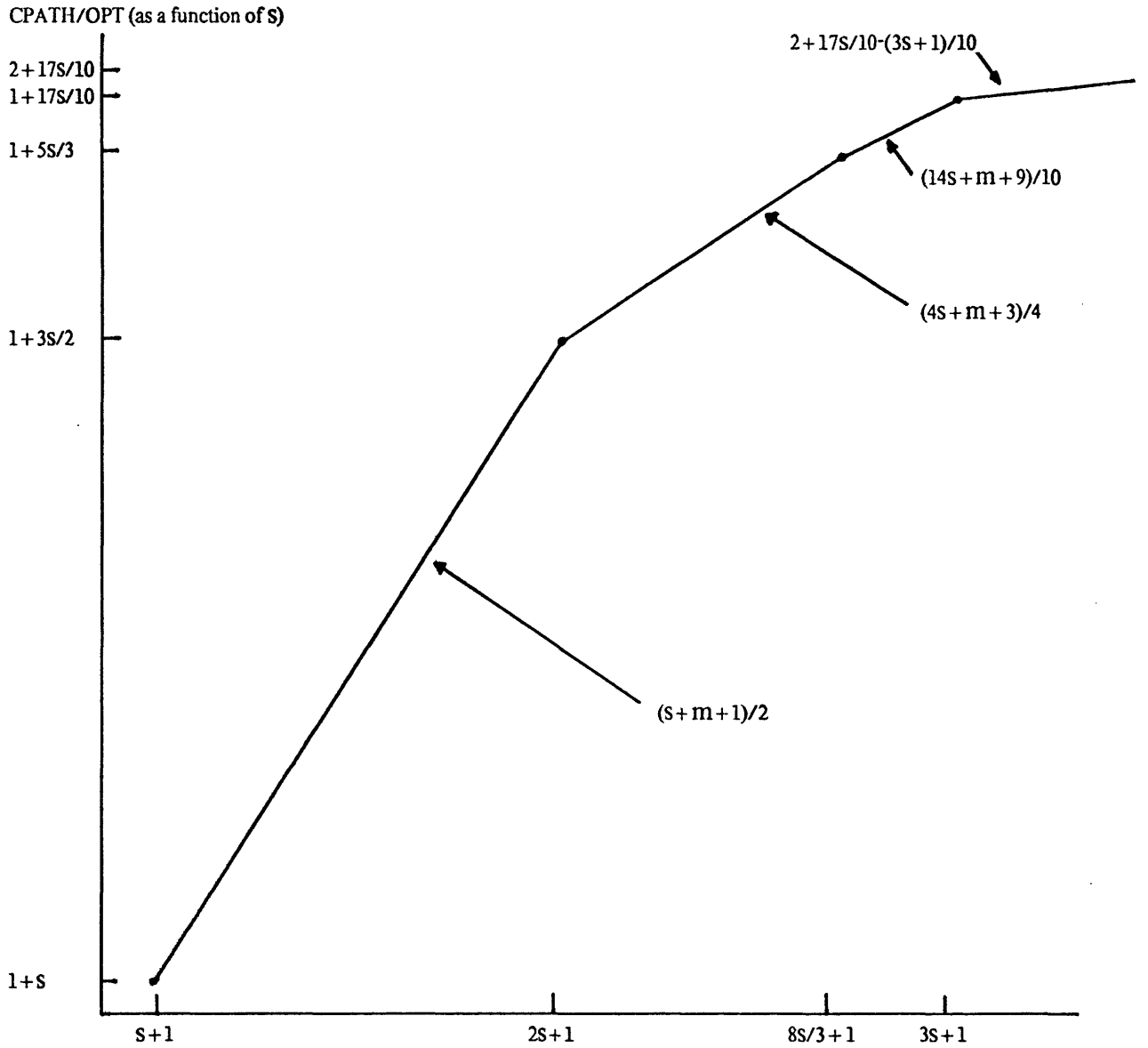
Because the bound given in Theorem 4.1 is somewhat imposing, it is useful to obtain an intuition about the nature of that bound. In this section we try to provide this intuition from the point of view of the "lower bound". That is, we discuss the principles behind the construction of task systems for which critical schedules exist which achieve various portions of the bound. We will concentrate on the middle four portions of the bound. The other two portions arise mainly from "boundary" constraints. In particular, the first portion ($2 \leq m < s+1$) is the situation where the processor constraint dominates worst case behavior. The final portion ($3s+1 \leq m < 10$) arises because s and m are both small. We ignore these two protions of the bound in the rest of this discussion.

The key to understanding the middle four portions of the bound is the following: When constructing a task system for which a "bad" critical path schedule exists, there are three kinds of

constraints to deal with: precedence constraints, processor constraints and resource constraints. Moreover, there are actually s kinds of resource constraints - one constraint for each continuous resource. A task system with a "bad" critical path schedule (presumably) exploits each of these constraints to the fullest. Now consider the bound $2 + 17s/10 - (3s+1)/m$. The various terms of that bound can be interpreted as follows: The term $17s/10$ arises from the exploitation of the resource constraints. There are $3s \cdot OPT$ tasks involved in this. A term of 1 arises from the exploitation of the precedence constraints. There are OPT tasks involved in this. Finally, the term $1 - (3s+1)/m$ arises from the exploitation of the processor constraints. All of the remaining tasks in the system are involved in this. Similar interpretations exist for the other three portions of the bound. However, in those cases, only the resource and precedence constraints are exploited and not the processor constraints. Only when $m > 3s+1$ is it "profitable" to exploit the processor constraints.

This interpretation can be seen more clearly, if we assume that s is fixed, and that m and CPATH/OPT are expressed as functions of s (Figure 4.1 shows the plot of such a function). Initially, assume that $m = s+1$ and that we have a UET task system with continuous resources S, such that a critical path schedule exists for S, with CPATH/OPT arbitrarily close to $s+1$. In S, there are OPT tasks devoted to exploiting the precedence constraints, and for each continuous resource, there are OPT tasks devoted to exploiting the constraint imposed by that resource. The processor constraints are not being exploited at all. Now consider how S is modified as m is increased, one processor at a time, from $s+1$ to $2s+1$. Each time m is increased, several tasks are added to S. The purpose of adding these tasks is to more fully exploit the resource constraints. Each time m increases by one processor, the worst case bound increases by a constant amount (namely, 1/2) due to the addition of those tasks. At $m = 2s+1$, there are OPT tasks devoted to exploiting the precedence constraints, and for each continuous resource, there are $2 \cdot OPT$ tasks devoted to exploiting the constraint imposed by that resource. Now consider the (similar) situation as m is increased, one processor at a time from $2s+1$ to $8s/3+1$. Again tasks are added to S each time m

Figure 4.1: Graph of the upper bound as a function of s



increases. Now, however, the worst case bound increases by only 1/4 each time m increases. For a third time, consider the (similar) situation as m is increased one processor at a time, from $8s/3+1$ to $3s+1$. Again tasks are added to S each time m increases. In this instance, the worst case bound increases by only 1/10 each time m increases. At $m=3s+1$, there are OPT tasks devoted to exploiting the precedence constraints, and $3s \cdot OPT$ tasks devoted to exploiting the resource constraints - for each continuous resource, there are $3 \cdot OPT$ tasks exploiting the constraint imposed by that resource. At this point, the precedence and resource constraints are fully exploited. Finally, as m is increased beyond $3s+1$, yet more tasks are added to S. These tasks exploit the processor constraint. Note however, that the bound increases only so slightly in this range, and that in fact, it converges to $2 + 17s/10$ as m approaches infinity.

4.2.2 A comparison

Although Theorem 4.1 provides (in contrast to previous results) a tight upper bound for the worst case performance of critical path scheduling of UET task systems with continuous resources and a processor constraint, there is a question of how much that result really improves over previous results. That is, consider the bounds (cited earlier) of Yao [Y] and Garey, et.al. [GGJY] as they apply to UET task systems with continuous resources and a processor constraint. Those results can be combined to yield the following composite bound:

$$CPATH/OPT \leq \min\{ m, 2 + 2s - (2s+1)/m, 27/10 + 17s/10\}$$

The question which arises, is whether this composite bound is much worse than the best possible bound (our Theorem 4.1). The answer to this question is yes. For instance, if $s > 6$ and $m = 1.8s + 2$, the composite bound indicates that $CPATH/OPT \leq 17s/10 + 27/10$. The bound that we give shows that $CPATH/OPT \leq 14s/10 + 3/2$. The difference between the two bounds is $3s/10 + 6/5$ -- a value which grows linearly with s. In percentages, the composite bound in this case is too large by over 21 percent. Table 4.1 shows both the composite bound and our best possible bound for several specific

Table 4.1

A comparison between the composite bound [Yao,GGJY] and the best possible bound

s	m	composite	best	error in composite
2	4	4	3.5	14%
	5	5	4	25%
	6	5.17	4.25	22%
	7	5.29	4.43	19%
	8	5.37	4.54	18%
	∞	6	5.4	11%
8	10	10	9.5	5%
	15	15	12	25%
	20	16.3	13.75	19%
	25	16.3	14.6	12%
	30	16.3	14.77	10%
	∞	16.3	15.6	4%
15	20	20	18	11%
	25	25	20.5	22%
	30	28.2	23	23%
	35	28.2	24.5	15%
	40	28.2	25.75	10%
	45	28.2	26.4	7%
	50	28.2	26.58	6%
	∞	28.2	27.5	3%

The above values have been rounded to two decimal places.

combinations of s and m. That table also shows the percentage error in the composite bound relative to the best possible bound for each such combination of s and m.

Note from Table 4.1 that although our results improve upon the composite result whenever $m > s+1$, the improvement is usually most significant when the number of processors is small relative to the number of continuous resources.

4.2.3 The upper bound

In the next two sections we prove Theorem 4.1. The upper bound is given in this section and the lower bound is given in the section 4.2.4.

4.2.3.1 Preliminaries

Before beginning the proof of the upper bound, we require several definitions.

With respect to the usage of the resources in the system, we have the following definitions: $R_{max}(T) = \max \{R_v(T): 1 \leq v \leq s\}$. Given task T, $R_v(T)$ is the R_v-value of T and $R_{max}(T)$ is the R_{max}-value of T. This notation is extended to a set of tasks B, with $R_v(B) = \Sigma R_v(T)$ over all $T \in B$ and $R_{max}(B) = \Sigma R_{max}(T)$ over all $T \in B$. For completeness, if B is empty, let $R_{max}(B) = 0$. Finally, a set of tasks B, is a legal set of tasks if for each resource v, $R_v(B) \leq 1$.

With respect to the precedence constraints, we remind the reader of the definition of the level of a task: If T_i has no successors then $level(T_i) = 1$; otherwise, $level(T_i) = 1 + \max\{level(T_j): T_i < T_j\}$. This notion can be extended to a set of tasks B, by letting $level(B) = \max\{level(T_i) : T_i \in B\}$.

4.2.3.2 Proof outline

Consider any critical path schedule for a task system *S*. The time units of that schedule may be divided into three sets: those time units where the final task of each level executes, those where all of the processors are utilized and those where at least one processor is idle due solely to resource constraints. Call these path, full and resource time units respectively. The proof follows by bounding the number of time units of each type. The number of path time units is bounded by the length of an optimal schedule.

The number of full time units can be bounded using the length of an optimal schedule and the number of tasks executed in resource and path time units. The number of resource time units can bounded by the use of a "weighting function".

A weighting function W, is a mapping from the interval [0, 1] to an interval [0, x], where the x depends on the particular weighting function. We extend the functional notation to tasks and let $W(T) = W(R_{max}(T))$. Moreover, if B is a set of tasks, then $W(B) = \Sigma W(T)$ over all $T \in B$. (Our use of weighting functions is motivated by, and draws upon, the work of Garey, et.al. [GGJY]). Given a particular weighting function and a set of resource time units, the average weight associated with each of those time units can be bounded below (this lower bound will be 1). Moreover, by examining an optimal schedule, the total weight associated with all tasks executing in resource time units can be bounded above. Combining these two bounds gives an upper bound on the number of resource time units. The result then follows from the upper bounds on the numbers of path, full and resource time units.

4.2.3.3 Two important properties

In this section we introduce two properties of weighting functions.

Definition: Weighting function W has Property A, if:

Given a task T' and a nonempty set of tasks B such that:

$R_{max}(T) \geq R_{max}(T')$ for each $T \in B$ and $R_{max}(T') > 1 - R_{max}(B)$,

then $W(B) \geq 1$.

Definition: Weighting function W has Property B, if:

Given a set of time units $\{B_1, \ldots, B_t\}$ with $t \geq 1$ and $Y = \bigcup_{i=1}^{t} B_i$, such that:

For every task $T \in B_i$, $1 < i \leq t$, and every j, $1 \leq j < i$, $R_{max}(T) > 1 - R_{max}(B_j)$,

then there exists a task $T^* \in Y$, such that $W(Y - \{T^*\}) \geq t-1$.

Intuitively, Property A states that given a set of n tasks in which the total resource requirements of the tasks exceeds one, then the total weight of the largest n-1 tasks is at least one. Property B will be used to

obtain a lower bound on the average weight associated with a resource time unit.

Lemma 4.1: If W is a weighting function which has Property A, then W also has Property B.

Proof

Assume that W is a weighting function which has Property A, and let $\{B_1, \ldots, B_t\}$ be a set of time units with $Y = \bigcup_{i=1}^{t} B_i$ such that for every task $T \in B_i$, $1 < i \leq t$, and every j, $1 \leq j < i$, $R_{max}(T) > 1 - R_{max}(B_j)$. We want to show that there exists a task $T^* \in Y$ such that $W(Y-\{T^*\}) \geq t-1$. Without loss of generality, assume that $W(B_i) < 1$ for each time unit B_i, $1 \leq i \leq t$. The proof is by induction on t.

If $t = 1$ the lemma is immediate, so suppose that $t \geq 2$. Consider time units B_{t-1} and B_t. Let X be any task in B_t. Then $R_{max}(X) > 1 - R_{max}(B_{t-1})$. Moreover, for any task $T \in (B_{t-1} \cup \{X\})$, $R_{max}(T) > 1 - R_{max}(B_{t-1} \cup \{X\} - \{T\})$. In particular, let Z be a task in $(B_{t-1} \cup \{X\})$ with a minimal R_{max}-value. From Property A, it follows that $W(B_{t-1} \cup \{X\} - \{Z\}) \geq 1$.

Now consider the set of time units $\{B'_1, \ldots, B'_{t-1}\}$, where $B'_i = B_i$ for $1 \leq i \leq t-2$, and $B'_{t-1} = \{Z\}$. Let $Y' = \bigcup_{i=1}^{t-1} B'_i$. By induction, there exists a $T^* \in Y'$, such that $W(Y'-\{T^*\}) \geq t-2$. Thus, $W(Y - \{T^*\}) \geq W(Y' - \{T^*\}) + W(B_{t-1} \cup \{X\} - \{Z\}) \geq t-2 + 1 = t-1$. □

4.2.3.4 The weighting functions

Three weighting functions are used in the proof of the main theorem. Three functions are used, as opposed to just one, due to varying requirements with respect to the weights assigned in various parts of that proof. Weighting function W_1 has the property that if $\alpha_1 + \alpha_2 \leq 1$, then $W_1(\alpha_1) + W_1(\alpha_2) \leq 1.5$. Moreover, values of α_1 and α_2 exist such that $W_1(\alpha_1) + W_1(\alpha_2) = 1.5$. A similar statement can be made about weighting function W_2 and the value 1.6. Weighting function W_3 has the property that if $\alpha_1 + \ldots + \alpha_n \leq 1$, then $W_3(\alpha_1) + \ldots + W_3(\alpha_n) \leq 1.7$. These properties play a critical role in establishing various segments of the upper bound.

For each of the three weighting functions which we introduce, we give two major results. First, we give an upper bound on the weight of a legal set of tasks. As a corollary to this result we give an upper

bound on the weight of any set of tasks drawn from the task system which we are considering. Both of these bounds depend upon the cardinality of the set of tasks being considered. These results will allow us to bound the total weight of the tasks executing in resource time units. Secondly, we show that the weighting function has Property B.

4.2.3.4.1 The first weighting function

Definition: $W_1(\alpha) = \begin{cases} 0 & \text{if } \alpha = 0 \\ 1/4 & \text{if } \alpha \in (0, 1/4] \\ 1/2 & \text{if } \alpha \in (1/4, 1/2] \\ 1 & \text{if } \alpha \in (1/2, 1] \end{cases}$ u.br Lemma 4.2: If B is a legal set of tasks, then $W_1(B) \leq \min\{ (|B|+s)/2, (|B|+4s)/4 \}$.

Proof

Recall that B is a legal set of tasks if for each resource v, the total usage of v by the tasks in B does not exceed one.

Part 1: Let $X = \{T \in B: R_{max}(T) > 1/2\}$ and let $x = |X|$. Since for each resource v, there is at most one $T \in B$, such that $R_v(T) > 1/2$, it must be that $x \leq s$. Moreover, if $R_{max}(T) > 1/2$ then $W_1(T) = 1$. Each task $T \in (B - X)$ has $R_{max}(T) \leq 1/2$, hence $W_1(T) \leq 1/2$. Thus, $W_1(B)$ is bounded above by $\max[x + (|B|-x)/2]$ such that $x \leq s$. This maximum occurs at $x = s$. Therefore, $W_1(B) \leq s + (|B|-s)/2 = (|B|+s)/2$.

Part 2: Let $X = \{T \in B: R_{max}(T) > 1/2\}$, let $x = |X|$, let $Y = \{T \in B: 1/4 < R_{max}(T) \leq 1/2\}$ and let $y = |Y|$. Similarly to Case 1, we deduce that $x \leq s$ and $y \leq 3s - 2x$. Moreover, if $R_{max}(T) > 1/2$ then $W_1(T) = 1$ and if $1/4 < R_{max}(T) \leq 1/2$ then $W_1(T) = 1/2$. Each task $T \in (B - X - Y)$ has $R_{max}(T) \leq 1/4$ and $W_1(T) \leq 1/4$. Thus, $W_1(B)$ is bounded above by $\max[x + y/2 + (|B|-x-y)/4]$ such that $x \leq s$ and $y \leq 3s - 2x$. This maximum occurs at $x = y = s$, so $W_1(B) \leq s + s/2 + (|B|-2s)/4 = (|B|+4s)/4$. □

Corollary 4.1: Given a set of tasks $Y \subseteq T$, then $W_1(Y) \leq \min\{ (|Y|+s\cdot OPT)/2, (|Y|+4s\cdot OPT)/4 \}$.

Proof

Let $B_1, \ldots B_{OPT}$ be the time units of an optimal schedule restricted to the tasks in Y. Then, $Y = \cup_{i=1}^{OPT} B_i$ and $W_1(Y) = \Sigma_{i=1}^{OPT} W_1(B_i)$.

Part 1: By Lemma 4.2, each $W_1(B_i) \leq (|B_i|+s)/2$. Thus, $W_1(Y) \leq \Sigma_{i=1}^{OPT} (|B_i|+s)/2 = s \cdot OPT/2 + \Sigma_{i=1}^{OPT} |B_i|/2 = (|Y|+s \cdot OPT)/2$.

Part 2: By Lemma 4.2, each $W_1(B_i) \leq (|B_i|+4s)/4$. Thus, $W_1(Y) \leq \Sigma_{i=1}^{OPT} (|B_i|+4s)/4 = (|Y|+4s \cdot OPT)/4$. □

Lemma 4.3: Weighting function W_1 has Property B.

Proof

By Lemma 4.1, it is sufficient to show that W_1 has Property A. Consider a task T' and a nonempty set of tasks B, such that $R_{max}(T) \geq R_{max}(T')$ for each $T \in B$ and $R_{max}(T') > 1 - R_{max}(B)$. We want to show that $W_1(B) \geq 1$.

If $R_{max}(T) > 1/2$ for any $T \in B$, then the lemma is immediate, so suppose $R_{max}(T) \leq 1/2$ for each $T \in B$. If $R_{max}(T') = 0$ then $R_{max}(B) \geq 1$, hence $W_1(B) \geq 1$, so suppose $R_{max}(T') > 0$.

Case 1: $R_{max}(T') \in (0, 1/4]$

Then $R_{max}(B) > 3/4$. Since for each $T \in B$, $0 < R_{max}(T) \leq 1/2$, we have that $|B| \geq 2$. Moreover, for $T \in B$, $W_2(T)$ is either 1/4 or 1/2. If $|B| \geq 4$, then the lemma is immediate. If $|B| = 3$ then at least one of the tasks has an R_{max}-value exceeding 1/4, hence it has a weight of 1/2. The other two tasks have weights of at least 1/4. Thus, $W_1(B) \geq 1$. If $|B| = 2$, then both of the tasks in B must have R_{max}-values exceeding 1/4, hence they have weights of 1/2, and $W_1(B) = 1$.

Case 2: $R_{max}(T') \in (1/4, 1/2]$

Then $R_{max}(B) > 1/2$. Hence $|B| \geq 2$, since $R_{max}(T) \leq 1/2$ for each $T \in B$. Since for each $T \in B$, $R_{max}(T) \geq R_{max}(T')$, we have: $R_{max}(T) \in (1/4, 1/2]$ and $W_1(T) = 1/2$ for $T \in B$. Thus, $W_1(B) = |B|/2 \geq 1$. □

4.2.3.4.2 The second weighting function

Definition: $W_2(\alpha) =$ 0 if $\alpha = 0$

10/100 if $\alpha \in (0, .092]$

15/100 if $\alpha \in (.092, .136]$

20/100 if $\alpha \in (.136, .182]$

25/100 if $\alpha \in (.182, .204]$

30/100 if $\alpha \in (.204, .250]$

40/100 if $\alpha \in (.250, .296]$

45/100 if $\alpha \in (.296, .318]$

50/100 if $\alpha \in (.318, .364]$

55/100 if $\alpha \in (.364, .408]$

60/100 if $\alpha \in (.408, .500]$

1 if $\alpha \in (.500, 1]$

We have the following facts which follow from the definition of W_2:

Fact 1: If $\alpha \in (.092, .500]$, then $W_2(\alpha) \leq (1.64)\alpha$.

Fact 2: If $|B| = 3$ and $R_v(B) \leq 1$, then $W_2(R_v(B)) \leq 17/10$.

Fact 3: If $|B| = 2$ and $R_v(B) \leq 1$, then $W_2(R_v(B)) \leq 16/10$.

Fact 4: If $|B| = 2$ and $R_v(B) \leq .500$, then $W_2(R_v(B)) \leq 7/10$.

The following claim is useful in proving Lemma 4.4:

Claim A: If B is a set of tasks such that $R_v(B) \leq 1$ and $|B| \geq 2$ then $W_2(R_v(B)) \leq (|B|+14)/10$.

Proof

If $|B| \leq 3$ then the claim follows from Facts 2 and 3, so, assume that $|B| \geq 4$. Define the following two sets of tasks:

$Y = \{T \in B: R_v(T) > .500\}$

$X = \{T \in B: .092 < R_v(T) \leq .500\}$

Clearly, $W_2(R_v(B)) = W_2(R_v(Y)) + W_2(R_v(X)) + W_2(R_v(B\text{-}X\text{-}Y))$. Note that if $T \in Y$, then $W_2(R_v(T)) = 1$ and if $T \in B\text{-}X\text{-}Y$ then $W_2(R_v(T)) \leq 10/100$. Thus,

$W_2(R_v(B)) \leq |Y| + W_2(R_v(X)) + (|B| - |X| - |Y|)/10$.

Case 1: $|Y| = 0$

Then, $W_2(R_v(B)) = W_2(R_v(X)) + (|B| - |X|)/10$.

If $|X| \leq 2$, then since for each $T \in X$, $W_2(R_v(T)) \leq 60/100$, we have $W_2(R_v(B)) \leq (60/100)|X| + (|B|-|X|)/10 = 5|X|/10 + |B|/10 < (|B|+14)/10$.

If $|X| > 2$, then by Fact 1, $W_2(R_v(X)) \leq 1.64$, hence $W_2(R_v(B)) \leq 1.64 + (|B| - |X|)/10 \leq 1.64 + [|B| - 3]/10 < (|B|+14)/10$.

Case 2: $|Y| = 1$

Note that $R_v(X) < .500$ and

$$W_2(R_v(B)) \leq 1 + W_2(R_v(X)) + [|B|-|X|-1]/10 \qquad (I)$$

If $|X| = 0$, then from (I), $W_2(R_v(B)) \leq 1 + (|B|-1)/10 < (|B|+14)/10$.

If $|X| = 1$, then $W_2(R_v(X)) \leq 60/100$, so from (I), $W_2(R_v(B)) \leq 1 + 60/100 + (|B|-2)/10 = (|B|+14)/10$.

If $|X| = 2$, then by Fact 4, $W_2(R_v(X)) \leq 7/10$, so from (I),

$W_2(R_v(B)) \leq 1 + 7/10 + (|B|-3)/10 = (|B|+14)/10$.

If $|X| = 3$, then let $\max_v(X) = \max\{R_v(T): T \in X\}$.

If $\max_v(X) > .318$ then the other two tasks in X have R_v-values totaling less than .182, since $R_v(X) < .500$. Then at least one of these other two tasks must have an R_v-value less than .091. But, by definition, each task in X has an R_v-value exceeding .092. Thus, $\max_v(X) \leq .318$.

If $\max_v(X) \in (.250, .318]$, then $W_2(\max_v(X)) \leq 45/100$. The other two tasks in X have R_v-values not exceeding .136 and .182 respectively, hence they have a total weight not exceeding 35/100. Thus, $W_2(R_v(X)) \leq 80/100$.

If $\max_v(X) \in (.092, .250]$, then $W_2(\max_v(X)) \leq 30/100$. The other two tasks in X have R_v-values not exceeding .204, hence they have a total weight not exceeding 50/100. Thus, $W_2(R_v(X)) \leq 80/100$.

Thus, if $|X| = 3$ then $W_2(R_v(X)) \leq 80/100$, hence $W_2(R_v(B)) \leq 1 + 80/100 + [|B|-4]/10 = (|B|+14)/10$.

If $|X| \geq 4$, then from Fact 1, $W_2(R_v(X)) \leq 1.64R_v(X) \leq .82$. Then from (I), $W_2(R_v(B)) \leq 1 + .82 + [|B|-|X|-1]/10 \leq 1.82 + [|B|-5]/10 < (|B|+14)/10$. □

Lemma 4.4: If B is a legal set of tasks, then $W_2(B) \leq (|B|+14s)/10$.

Proof

Partition the tasks in B into s sets $D_1, \ldots, D_s$, where $T \in D_v$ if and only if v is the minimum index such that $R_v(T) = R_{max}(T)$. Clearly, $W_2(B) = \Sigma_{v=1}^{s} W_2(R_v(D_v))$. Now partition the resources into sets $Z_0, \ldots, Z_n$, according to the sizes of the respective D_v sets. That is, resource v is placed into set $Z_{|D_v|}$ (Figure 4.2). Thus, $W_2(B) = \Sigma_{j=0}^{n} (\Sigma_{v \in Z_j} W_2(R_v(D_v)))$. Clearly, for each $v \in Z_0$, $W_2(R_v(D_v)) = 0$ and from the definition of W_2 it follows that for each $v \in Z_1$, $W_2(R_v(D_v)) \leq 1$. Moreover, from Claim A, it follows that for each $j \geq 2$, and each $v \in Z_j$, $W_2(R_v(D_v)) \leq (j + 14)/10$ and $\Sigma_{v \in Z_j} W_2(R_v(D_v)) = [(j+14)/10]|Z_j|$. Thus, $W_2(B) \leq |Z_1| + \Sigma_{j=2}^{n} [(j + 14)/10]|Z_j| = \Sigma_{j=1}^{n} j|Z_j|/10 + \Sigma_{j=1}^{n} 14|Z_j|/10 - |Z_1|/2$. But, the Z_i's are a partition of the resources, so $\Sigma_{j=1}^{n} |Z_j| \leq s$. Moreover, that partition is based on a partition of the tasks such that $\Sigma_{j=1}^{n} j\,|Z_j| \leq |B|$. Also, $|Z_1| \geq 0$.

$\therefore W_2(B) \leq |B|/10 + 14s/10 - 0/10 = (|B| + 14s)/10$ □

Corollary 4.2: Given a set of tasks $Y \subseteq T$, then $W_2(Y) \leq (|Y|+14s \cdot OPT)/10$.

Let $B_1, \ldots, B_{OPT}$ be the time units of an optimal schedule restricted to the tasks in Y. By Lemma 4.4, each $W_2(B_i) \leq (|B_i| + 14s)/10$. Thus, $W_2(Y) = \Sigma_{i=1}^{OPT} W_2(B_i) \leq \Sigma_{i=1}^{OPT} [|B_i| + 14s]/10 = 14s \cdot OPT/10 + \Sigma_{i=1}^{OPT} |B_i|/10 = (|Y| + 14s \cdot OPT)/10$. □

Lemma 4.5: Weighting function W_2 has Property B.

Proof

By Lemma 4.1 it is sufficient to show that W_2 has Property A. Consider a task T' and a

Figure 4.2: Partitioning the resources.

	Resource						
	1	2	3	4	5	6	7
T_1:	-	-	-	(.7)	-	-	-
T_2:	-	-	-	-	-	-	(.6)
T_3:	-	-	-	-	(.5)	.2	-
T_4:	-	-	-	-	(.1)	.1	-
T_5:	.1	(.3)	-	-	-	-	-
T_6:	-	-	-	-	(.1)	-	-
T_7:	-	-	-	(.2)	-	-	.2
T_8:	(.8)	-	-	.1	-	-	-
T_9:	-	(.6)	-	-	-	-	-
T_{10}:	-	-	(.7)	-	-	.4	-
T_{11}:	-	-	-	-	(.3)	-	-

These are the resource requirements for the tasks in a system with 11 tasks and 7 resources. A zero requirement is shown as a dash. The largest requirement of each task is circled.

$$B = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}\}$$

$D_6 = \emptyset \longrightarrow Z_0 = \{6\}$

$D_1 = \{T_8\}$
$D_3 = \{T_{10}\}$ $\longrightarrow Z_1 = \{1, 3, 7\}$
$D_7 = \{T_2\}$

$D_2 = \{T_5, T_9\}$
$D_4 = \{T_1, T_7\}$ $\longrightarrow Z_2 = \{2, 4\}$

$Z_3 = \emptyset$

$D_5 = \{T_3, T_4, T_6, T_{11}\} \longrightarrow Z_4 = \{5\}$

$Z_5 = \ldots = Z_{11} = \emptyset$

Task partition Resource partition

nonempty set of tasks B such that $R_{max}(T) \geq R_{max}(T')$ for $T \in B$, and $R_{max}(T') > 1 - R_{max}(B)$. We want to show that $W_2(B) \geq 1$.

If $|B| = 1$, the result follows immediately from the definition of W_2, so assume that $|B| \geq 2$. Let $min(B) = min\{R_{max}(T): T \in B\}$. If there is only one resource in the task system, then min(B) is the smallest resource requirement of any task in B. Given a time unit B, it is possible to compute a lower bound for $W_2(B)$ based on |B|, min(B) and $R_{max}(B)$. In particular, Table 4.2 gives various combinations of |B|, min(B) and $R_{max}(B)$, each of which implies that $W_2(B) \geq 1$. These values were verified using the MACSYMA system of the MIT Laboratory for Computer Science. The program used to do the verification is shown in Figure 4.3.

Now consider the possible values of $W_2(T')$. If $W_2(T') \geq 50/100$, then for each $T \in B$, $W_2(T) \geq 50/100$. Since $|B| \geq 2$, we have $W_2(B) \geq 1$. If $W_2(T') = 10/100$, then $0 < R_{max}(T') \leq .092$. But this implies that $R_{max}(B) > .908$ and $min(B) > 0$ hence from Table 4.2, $W_2(B) \geq 1$. If $R_{max}(T) = 0$, then $R_{max}(B) \geq 1$, hence $W_2(B) \geq 1$.

There are six remaining possibilities for $W_2(T')$: 15/100, 20/100, 25/100, 30/100, 40/100, and 45/100. Associated with each of these weights there is a range $(\alpha_1, \alpha_2]$ in which $R_{max}(T')$ must lie. Moreover, in each instance it follows that $min(B) > \alpha_1$ and that $R_{max}(B) > 1 - \alpha_2$. For each $(\alpha_1, \alpha_2]$ pair, an examination of the "relevant" entries in Table 4.2, shows that $W_2(B) \geq 1$ in all instances. A guide to the "relevant" entries of Table 4.2 is given in Table 4.3. In Table 4.3, for each of the six possible values of $W_2(T')$, we give the values α_1, α_2, the subsequent lower bounds on min(B) and $R_{max}(B)$ and the entries of Table 4.2 that need to be examined. Note that entries are not listed for each size of |B| in every case. In particular, for each $W_2(T')$ possibility, only one entry of the form (|B|, min(B), 0) is given. Such an entry implies that $W_2(B) \geq |B| W_2(min(B)) \geq 1$. Thus, for any larger |B|, we also have $W_2(B) \geq 1$.

For example, when $W_2(T') = 25/100$, $R_{max}(T') \in (.182, .204]$. Thus, $min(B) > .182$ and

Table 4.2

$\|B\|$	min(B)	$R_{max}(B)$	$\|B\|$	min(B)	$R_{max}(B)$	$\|B\|$	min(B)	$R_{max}(B)$
2	0	.750	4	0	.820	7	0	.868
2	.250	.704	4	.136	.818	7	.092	0
2	.296	.682	4	.182	0			
						8	0	.870
3	0	.818	5	0	.864	8	.092	0
3	.182	.750	5	.136	0			
3	.250	0				9	0	.872
			6	0	.866	9	.092	0
			6	.092	.862			
			6	.136	0	10	0	0

An entry (i, x, y) in this table is interpreted as follows: If B is a set of tasks such that $|B| = i$, $\min(B) > x$, and $R_{max}(B) > y$, then $W_2(B) > 1$.

Table 4.3

$W_2(T')$	(α_1, α_2)	min(B)>	$R_{max}(B)$>	Relevant Entries
15/100	(.092, .136]	.092	.864	(2, 0, .750), (3, 0, .818), (4, 0, .820), (5, 0, .864), (6, .092, .862), (7, .092, 0)
20/100	(.136, .182]	.136	.818	(2, 0, .750), (3, 0, .818), (4, .136, .818), (5, .136, 0)
25/100	(.182, .204]	.182	.796	(2, 0, .750), (3, .182, .750), (4, .182, 0)
30/100	(.204, .250]	.204	.750	(2, 0, .750), (3, .182, .750), (4, .182, 0)
40/100	(.250, .296]	.250	.704	(2, .250, .704), (3, .250, 0)
45/100	(.296, .318]	.296	.682	(2, .296, .682), (3, .250, 0)

Figure 4.3: MACSYMA program used to verify the values in Table 4.2.

The function CALC takes three inputs: B, MINB, and RMAXB, and computes the minimum total weight of the tasks in a time unit where:

B is the number of tasks in the time unit

MINB is a lower bound on the resource requirement of each task in the time unit. That is, for each task T, $R_{max}(T) > MINB$.

RMAXB is alower bound on the total resource requirement of all the tasks in the time unit. That is, $R_{max}(B) > RMAXB$.

CALC finds the minimum total weight by doing an exhaustive search of the possible values for the resource requirements of the tasks in the time unit. For convenience, weights are multiplied by 100 and resource requirements are multiplied by 1000.

Sample ouput of the program is:

```
CALC(2, 296, 682)$                    (input to MACSYMA)

2 296 682 100                         (MACSYMA output - the fourth
                                       value is the minimum weight)
```

```
CALC(B, MINB, RMAXB) :=(MINWT : 100,
                        FOR J FROM 0 THRU 9 DO
                            (IF MINB = RES_J THEN BOT : J+1),
                        HELPCALC(B, 0, 0),
                        PRINT(B, MINB, RMAXB, MINWT))

HELPCALC(COUNT, CURWT, CURRES) :=
        IF COUNT = 0
            THEN (IF CURWT < MINWT AND CURRES > RMAXB
                      THEN MINWT : CURWT),
            ELSE (IF CURWT + WTS_BOT * COUNT < MINWT
                      THEN FOR I FROM BOT THRU 10 DO
                          HELPCALC(COUNT - 1, CURWT + WTS_I, CURRRES +RES_I))
```

The values of the WTS and RES arrays are as follows:

	0	1	2	3	4	5	6	7	8	9	10
WTS	0	10	15	20	25	30	40	45	50	55	60
RES	0	92	136	182	204	250	296	318	364	408	500

$R_{max}(B) > 1 - .204 = .796$. If $|B| \geq 4$, it follows from $|B|$ and $\min(B) > .182$ that $W_2(B) \geq 4$ $W_2(\min(B)) \geq 4\ (25/100) = 1$. If $|B| < 4$, the entries (2, 0 , .750) and (3, .182, .750) in Table 4.2 indicate that $W_2(B) \geq 1$. □

4.2.3.4.3 The third weighting function

Definition: $W_3(\alpha) = \begin{cases} (6/5)\alpha & \text{if } \alpha \in [0, 1/6] \\ (9/5)\alpha - 1/10 & \text{if } \alpha \in (1/6, 1/3] \\ (6/5)\alpha + 1/10 & \text{if } \alpha \in (1/3, 1/2] \\ (6/5)\alpha + 4/10 & \text{if } \alpha \in (1/2, 1] \end{cases}$

This is the weighting function defined in Garey, et.al.[GGJY]. In that paper the following corollary and lemma about W_3 are proven.

Corollary 4.3: Given a set of tasks $Y \subseteq T$, then $W_3(Y) \leq 17s \cdot OPT/10$.

Lemma 4.6: Given $0 \leq \alpha < 1/2$, and a set of tasks $B = \{T_1, \ldots, T_n\}$ with $n \geq 2$, such that $R_{max}(T_1) \geq R_{max}(T_2) > \alpha$ and $\alpha \geq 1 - R_{max}(B)$, then $W_3(B) \geq 1$.

A straight-forward consequence of Lemma 4.6 and the definition of W_3 (used to handle $|B| = 1$ and $R_{max}(T') \geq 1/2$) is that W_3 has Property A, hence:

Lemma 4.7: Weighting function W_3 has Property B.

4.2.3.5 The main result

In this section we complete the proof of the upper bound. Assume that a UET task system with continuous resources $S = \langle T, <, m, s \rangle$ is given. Let CPATH be a set containing the time units of a critical path schedule and let OPT be a set containing the time units of an optimal schedule for this system. As usual, we also let CPATH and OPT be the lengths of these schedules when appropriate. The time units in CPATH are partitioned into the following three sets:

$P = \{B_i \in \text{CPATH}: (\forall j > i)[\text{level}(B_i) > \text{level}(B_j)]\}$

$F = \{B_i \in \text{CPATH} : |B_i| = m \text{ and } B_i \notin P\}$

$H = \{B_i \in \text{CPATH}: |B_i| < m \text{ and } B_i \notin P\}$

The time units in P are path time units, those in F are full time units, and those in H are resource time units. Clearly, CPATH = |P| + |F| + |H|.

Let Q = {T ∈ *T*: T ∈ B_i and B_i ∈ H} (i.e. Q consists of all tasks executing in resource time units of CPATH). Clearly, |P| ≤ OPT and |F| ≤ OPT - |P|/m - |Q|/m. The number of resource time units |H|, can be bounded by use of the following lemma (adapted from a lemma given by Garey, et.al.[GGJY]).

Lemma 4.8: If W is a weighting function which has Property B, then there exists a set of tasks Q' ⊆ *T* with |Q'| = |Q| such that |H| ≤ W(Q').

Proof

Assume that W is a weighting function which has Property B. Let k be the maximum level of any task in *T*. For each level *l*, 1 ≤ *l* ≤ k, there is one time unit B_l ∈ P with level(B_l) = *l*. Let T_l be any task in B_l with level(T_l) = *l*. Moreover, for each level *l*, 1 ≤ *l* ≤ k, define the following two sets:

A_l = {B_i ∈ H : level(B_i) = *l*}

L_l = {T: level(T) = *l* and (∃B_i ∈ A_l)[T ∈ B_i]} ∪ {T_l}.

Thus, A_l contains all of the resource time units where the highest level of any task executing in the time unit is *l*. Likewise, L_l contains task T_l and all level *l* tasks executing in a resource time unit where the highest level of any task executing in the time unit is *l*. Figure 4.4 shows the correspondence between L_l, T_l and A_l.

Consider any set A_l. We claim that there exists a task X_l ∈ L_l such that $W(L_l - \{X_l\}) \geq |A_l|$. If $|A_l| = 0$ then the result is immediate, so assume that $|A_l| \geq 1$. Let $B_1, \ldots, B_{|A_l|}$ be the time units in A_l. For each B_i ∈ A_l, let $B_i' = B_i \cap L_l$. There is one B_i' for each B_i, and each B_i' contains at least one task. Also, let $B'_{|A_l|+1} = \{T_l\}$. Note that $\bigcup_{i=1}^{|A_l|+1} B_i' = L_l$. Moreover, each B_i' contains only level *l* tasks.

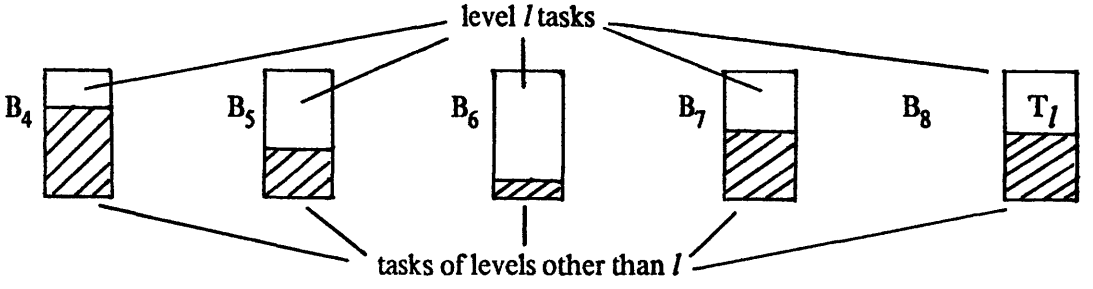
Now consider any B_j' and B_i', with j < i. Let T be any task in B_i'. When T was scheduled, all tasks with levels larger than *l* must have already been scheduled in time units prior to B_j.

Figure 4.4: An example of the sets A_l and L_l, and the task T_l

Assume that B_8 has a level of l and is a path time unit. This means that the task in B_8 of the highest level has level l, and that all tasks executing in time units after B_8 have levels less than l.

Some number of time units immediately preceding B_8 also have a level of l. Assume that these are time units B_4, B_5, B_6, and B_7. The set A_l consists of these 4 time units. The set L_l consists of all of the level l tasks which execute in these 4 time units, along with task T_l.

$A_l = \{B_4, B_5, B_6, B_7\}$ $\quad$ $\text{level}(B_i) = l$ for i = 4, 5, 6, 7, 8
B_8 is B_l in this instance



B_4, B_5, B_6 and B_7 are resource time units and B_8 is a path time unit

$L_l = \{T: \text{level}(T) = l \text{ and } T \text{ is in a time unit in } A_l\} \cup \{T_l\}$

The tasks in the non-shaded portions of B_4, B_5, B_6, B_7 and B_8 are the tasks in L_l.

Moreover, the only tasks already scheduled in time unit B_j were level l tasks. Thus, T was not scheduled to execute in B_j due solely to resource constraints imposed by the level l tasks in B_j. This means that for $T \in B_i'$, $R_{max}(T) > 1 - R_{max}(B_j')$ for all $j < i$. Thus, the B_j' s form a set of time units for which the conditions given in the definition of Property B hold. Then, since weighting function W has Property B, there exists a task $X_l \in L_l$ (since $L_l = \bigcup_{i=1}^{|A_l|+1} B_i'$) such that $W(L_l - \{X_l\}) \geq |A_l|$, and the claim is proved. □

Finally, let $Q' = (Q \cup \{T_l : 1 \leq l \leq k\}) - \{X_l : 1 \leq l \leq k\}$. Clearly, $|Q'| = |Q|$.

$\therefore |H| = \Sigma_{l=1}^{k} |A_l| \leq \Sigma_{l=1}^{k} W(L_l - \{X_l\}) \leq W(Q')$, since $\bigcup_{l=1}^{k} (L_l - \{X_l\}) \subseteq Q'$. □

From Lemma 4.8, it follows that given a particular weighting function W* which has Property B, there exists a set of tasks $Q' \subseteq T$ such that $|Q'| = |Q|$ and $|H| \leq W^*(Q')$.

Thus, $CPATH = |P| + |F| + |H| \leq |P| + (OPT - |P|/m - |Q|/m) + W^*(Q')$, and with a reordering of terms,

$$CPATH \leq OPT + |P|(1-1/m) - |Q|/m + W^*(Q'). \qquad (II)$$

There are six cases to consider based on the relative values of s and m.

Case 1: $2 \leq m < s+1$

Then $CPATH \leq m\,OPT$ since at least one task must execute during each time unit of CPATH.

Case 2: $s+1 \leq m < 2s+1$

Let W_1 be the weighting function W*. By Corollary 4.1, $W_1(Q') \leq [|Q'| + s \cdot OPT]/2 = [|Q| + s \cdot OPT]/2$. Thus from (II), $CPATH \leq OPT + |P|(1-1/m) - |Q|/m + [|Q| + s \cdot OPT]/2 = (1 + s/2) \cdot OPT + |P|(1-1/m) + |Q|[1/2 - 1/m]$. But, $1/2 - 1/m \geq 0$ and $|Q| \leq m \cdot OPT - |P|$. Hence, $CPATH \leq (1+s/2) \cdot OPT + |P|(1-1/m) + (m \cdot OPT - |P|)[1/2 - 1/m] = [(s+m)/2] \cdot OPT + |P|/2 \leq [(s+m+1)/2] \cdot OPT$, since $|P| \leq OPT$.

$\therefore CPATH/OPT \leq (s+m+1)/2$.

Case 3: $2s+1 \leq m < 8s/3 + 1$

First assume that $m \geq 4$. Let W_1 be the weighting function W*. Then by Corollary 4.1, $W_1(Q') \leq [|Q'|+4s\cdot OPT]/4$. Similarly to Case 2, we derive from (II) that $CPATH/OPT \leq (4s+m+3)/4$.

Now assume that $m < 4$. The only combination of s and m to lie in this range is $s=1$ and $m=3$. But, from Case 2 (since the assumption that $m < 2s+1$ was not used in that proof), $CPATH/OPT \leq (s+m+1)/2 = (4s+m+3)/4$ when $s=1$ and $m=3$.

Case 4: $8s/3 + 1 \leq m < 3s+1$

First assume that $m \geq 10$. Let W_2 be the weighting function W*. Then by Corollary 4.2, $W_2(Q') \leq [|Q'|+14s\cdot OPT]/10$. Similarly to Case 2 we derive from (II) that $CPATH/OPT \leq (14s+m+9)/10$.

Now assume that $m < 10$. The only combination of s and m to lie in this range is $s=3$ and $m=9$. But, from Case 3, $CPATH/OPT \leq (4s+m+3)/4 = (14s+m+9)/10$ when $s=3$ and $m=9$.

Case 5: $3s+1 \leq m$ and $m \geq 10$

First assume that $|Q| \geq 3s\cdot OPT$. Let W_3 be the weighting function W*. Then by Corollary 4.3, $W_3(Q') \leq 17s\cdot OPT/10$. Thus, from (II), $CPATH \leq OPT + |P|(1 - 1/m) - |Q|/m + 17s\cdot OPT/10$. But $-|Q| \leq -3s\cdot OPT$ and $|P| \leq OPT$, so $CPATH \leq OPT + OPT\cdot(1-1/m) - 3s\cdot OPT/m + 17s\cdot OPT/10 = OPT[2 + 17s/10 - (3s+1)/m]$.

Now assume that $|Q| < 3s\cdot OPT$. Let W_2 be the weighting function W*. Then by Corollary 4.2, $W_2(Q') \leq [|Q'|+14s\cdot OPT]/10 = [|Q|+14s\cdot OPT]/10$. Thus from (II), $CPATH \leq OPT + |P|(1-1/m) - |Q|/m + [|Q|+14s\cdot OPT]/10 = OPT\cdot[1+14s/10] + |P|(1-1/m) + |Q|[1/10 - 1/m]$. But $1/10 - 1/m \geq 0$, $|Q| < 3s\cdot OPT$ and $|P| \leq OPT$. Hence, $CPATH \leq OPT\cdot[1+14s/10] + OPT\cdot(1-1/m) + 3s\cdot OPT\cdot[1/10 -1/m] = OPT\cdot[2 + 17s/10 - (3s+1)/m]$. Thus, $CPATH/OPT \leq 2 + 17s/10 - (3s+1)/m$.

Case 6: $3s+1 \leq m$ and $m < 10$

First assume that $|Q| \geq (8s/3)\cdot OPT$. Let W_2 be the weighting function W*. Then, by Corollary 4.2,

$W_2(Q') \leq [|Q'| + 14s \cdot OPT]/10$. Similarly to Case 5, we derive from (II) that $CPATH/OPT \leq 2 + 5s/3 - (8s/3+1)/m$.

Now assume that $|Q| < (8s/3) \cdot OPT$. Let W_1 be the weighting function W*. Then by Corollary 4.1, $W_1(Q') \leq [|Q'|+4s \cdot OPT]/4$. Similarly to Case 5, we derive from (II) that $CPATH/OPT \leq 2 + 5s/3 - (8s/3 + 1)/m$.

This completes the proof of the upper bound for Theorem 4.1. □

4.2.4 The lower bound

In this section we prove that the upper bound for CPATH/OPT given in Theorem 4.1 is the best possible upper bound, completing the proof of that result.

For each possible combination of s and m, we exhibit a UET task system with continuous resources, $S = \langle T, \prec, m, s \rangle$, a critical path schedule for that system, and an optimal schedule for that system such that the ratio CPATH/OPT is arbitrarily close to the appropriate upper bound. As in the proof of the upper bound, there are six cases to consider based on the relationship between s and m. The constructions that we use in the six cases are similar, but not identical. They make use of task systems which differ primarily in the resource usages of certain tasks in the system. The overall precedence structures of these systems are the same, as are the resource usages of several of the tasks. Thus, before proving each of the lemmas, this general task system structure is introduced. The aspects of the system which are the same in all cases are specified. We indicate which parameters will be specified within the proofs of the individual lemmas. We also sketch optimal and critical path schedules for this general system. The exact nature of these schedules will, of course, depend upon the values assigned to the unspecified parameters within the proofs of the individual lemmas.

4.2.4.1 A general task system structure

Assume that $s \geq 1$ and $m \geq 2$, with $m \geq s+1$, are given (in the next section we will indicate how to handle the case of $m \leq s$). Integers x and z are to be specified later, as is ϵ, a positive constant. Consider

a task system S^* with the following tasks:

1. D_i for $1 \leq i \leq x$, such that $R_1(D_i) = \epsilon$ and $R_v(D_i) = 0$ for $v \neq 1$.
2. B_0 such that $R_1(B_0) = 1$ and $R_v(B_0) = 0$ for $v \neq 1$.
3. B_i for $1 \leq i \leq s$, such that $R_i(B_i) = 1$ and $R_v(B_i) = 0$ for $v \neq i$.
4. C_i for $1 \leq i \leq s$. These tasks require no resources.
5. A_j^i for $1 \leq i \leq s$ and $1 \leq j \leq z$. For $v \neq i$, $R_v(A_j^i) = 0$. The usage of resource i by each task A_j^i (its R_i-value) will be specified later (it will be a non-zero requirement). Tasks $A_1^i, \ldots, A_z^i$ are called A^i-tasks.

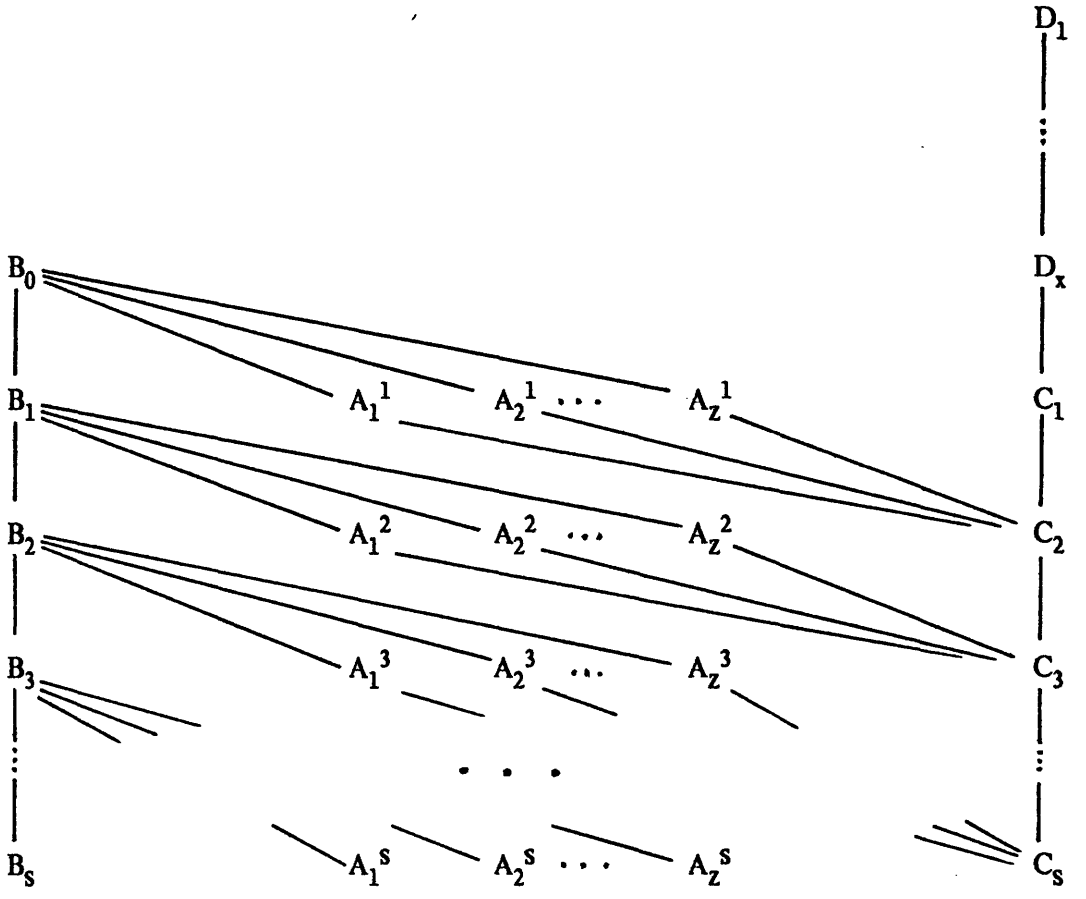
This task system has the following precedence constraints:

1. For $1 \leq i \leq x-1$, $D_i < D_{i+1}$. Moreover, $D_x < C_1$.
2. For $0 \leq i \leq s-1$, $B_i < B_{i+1}$ and $B_i < A_j^{i+1}$ for $1 \leq j \leq z$.
3. For $1 \leq i \leq s-1$ and $1 \leq j \leq z$, $A_j^i < C_{i+1}$.
4. For $1 \leq i \leq s-1$, $C_i < C_{i+1}$.

The precedence structure of this system is shown in Figure 4.5.

Assuming that the constants x, z and ϵ have been specified, consider the following schedule for S^* (Figure 4.6a): In the first $s+1$ time units execute the B-tasks. In the next x time units execute the D-tasks on processor m, and execute all of the A-tasks on the other m-1 processors. In the final s time units execute the C-tasks. Such a schedule has length $x + 2s + 1$. The assumption that the A-tasks can all be executed in time units $s+2$ through $x+s+1$ depends only on the number of A-tasks (which is sz) and on the resource requirements of the A-tasks - no precedence constraints are involved since after task B_s executes in time unit $s+1$, all of the A-tasks are available for execution. In each of the results using this general task system, the value z and the resource requirements of the A-tasks are specified so the A-tasks can indeed be executed in just x time units on m-1 processors and so the total requirement for resource 1 during each of those x time units does not exceed $1 - \epsilon$. This last condition is needed since

Figure 4.5: The general task system structure used for the lower bounds.



The non-zero resource requirements of these tasks are:

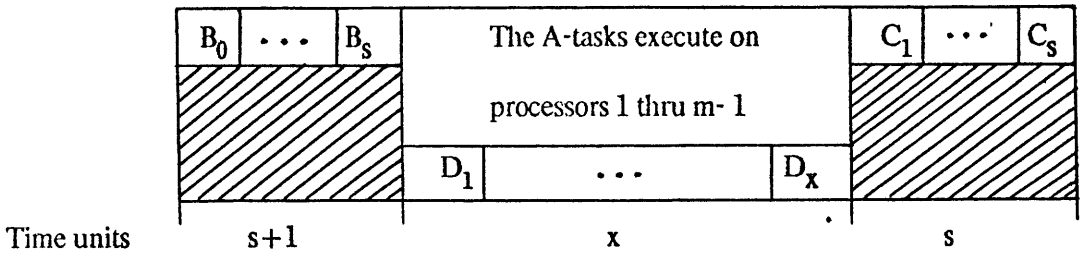
Each D-task requires ε of resource 1

B_0 requires all of resource 1

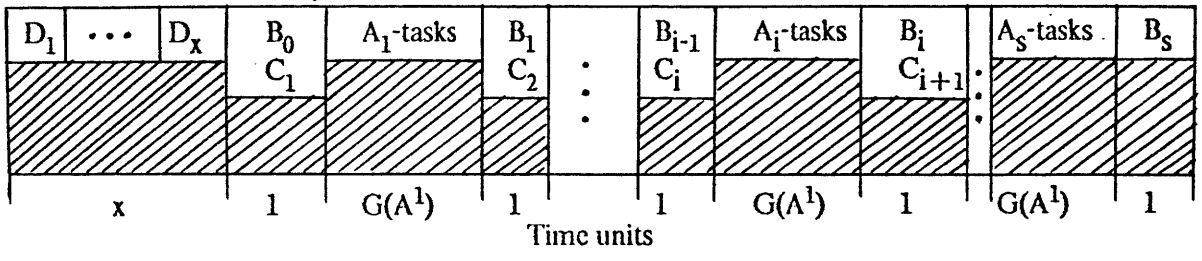
B_i requires all of resource i, $i > 0$

Each A^i-task requires a non-zero portion of resource i

Figure 4.6: Two schedules for the general task system structure



a) An optimal schedule -- length = x+2s+1



b) A critical path schedule -- length = x+s+1+s G(A^1)

each of the D-tasks requires ε of resource 1.

Now consider the critical path schedule for S^* generated from the following list: (D-tasks, B_0, C_1, A^1-tasks, B_1, C_2, A^2-tasks, ... , B_{s-1}, C_s, A^s-tasks, B_s). In this schedule, (Figure 4.6b) the D-tasks execute in the first x time units, then B_0 and C_1 execute in the next time unit, followed by the execution of the A^1-tasks. After those tasks have executed, B_1 and C_2 execute, followed by the execution of the A^2-tasks, and so on. Eventually, B_{s-1} and C_s execute, followed by the execution of the A^s-tasks. In the final time unit B_s executes. Assuming that the A^i-tasks are assigned the same resource requirements for resource i, as the A^1-tasks are assigned for resource 1 and that they are scheduled identically to the A^1-tasks, this schedule has length CPATH = $x + s + 1 + sG(A^1)$, where $G(A^1)$ is the length of the schedule for the A^1-tasks.

In the individual proofs which follow, several things are done. First, the values of x, z and ε are specified, and the remaining resource requirements for the A-tasks are given. We then show that the A-tasks can be executed on m-1 processors in x time units with the total requirement for resource 1 by the A-tasks, in each of those x time units, not exceeding $1 - \varepsilon$. This establishes that OPT $\leq x + 2s + 1$. The value of $G(A^1)$ is then derived by analyzing a particular list schedule for the A^1-tasks, establishing that CPATH $\geq x + s + 1 + sG(A^1)$. The lower bound for the worst case of CPATH/OPT is then obtained by combining the bounds for OPT and CPATH.

4.2.4.2 The Simple Cases

Lemma 4.9: If $2 \leq m < s + 1$, then CPATH/OPT can be arbitrarily close to m.

Proof

Assume that there are only m-1 resources. That is, assume s = m-1. (i.e. in the task system used to show that the upper bound of m may be approached arbitrarily closely, the tasks require only the first m-1 resources). The next lemma shows that in this case (i.e. $m \geq s+1$), that CPATH/OPT can be arbitrarily close to $(s+m+1)/2$. But, if $m = s+1$, then $(s+m+1)/2 = m$. □

Lemma 4.10: If $s + 1 \leq m < 2s + 1$ then CPATH/OPT can be arbitrarily close to $(s+m+1)/2$.

Proof

Let $c = (m-s-1)/s$. Let x be a positive integer such that $x = 0 \bmod 2s$, let $z = [1+c]x$ and let $\varepsilon < 1/12$. Now consider the task system S^* as specified in the previous section, using these values of x, z and ε. The remaining resource requirements of the A-tasks are:

For each i, $1 \leq i \leq s$,

x of the A^i-tasks have an R_i-value of $1/2 + \varepsilon$

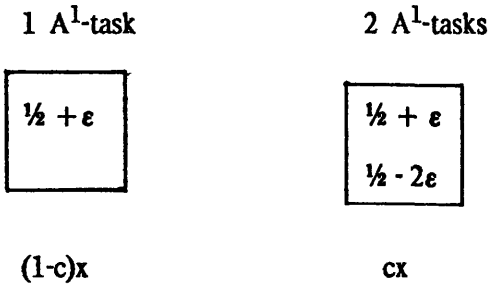
cx of the A^i-tasks have an R_i-value of $1/2 - 2\varepsilon$.

Note that for each i, we have specified resource requirements for exactly $x + cx = [1+c]x = z$ A^i-tasks. As desired, in total there are $zs = (m-1)x$ A-tasks.

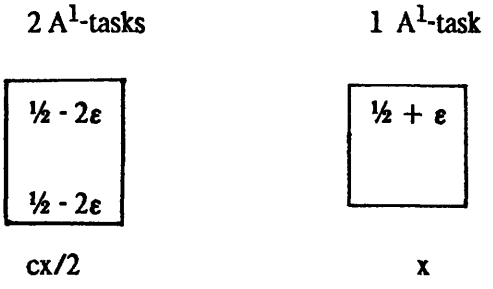
As noted in the previous section, OPT $\leq x + 2s + 1$ provided all of the A-tasks can be executed on m-1 processors in just x time units, with the total requirement for resource 1 by the A-tasks in each time unit not exceeding $1 - \varepsilon$. This can be done by executing the following tasks at each of those x time units (Figure 4.7a): For each i, $1 \leq i \leq s$, an A^i-task with an R_i-value of $1/2 + \varepsilon$ executes. This utilizes s processors at each time unit. Moreover, for $cs = m-s-1$ values of i, an A^i-task with an R_i-value of $1/2 - 2\varepsilon$ executes. Since m-1 A-tasks execute per time unit, all of the A-tasks can be executed in x time units. Note that for each i, there are (1-c)x time units in which one A^i-task executes and there are cx time units in which two A^i-tasks execute. Moreover, the total requirement for <u>each</u> resource during each time unit does not exceed $1 - \varepsilon$. Therefore, OPT $\leq x + 2s + 1$.

Also as noted in the previous section, for critical path schedules, CPATH $\geq x + s + 1 + sG(A^1)$, where $G(A^1)$ is the length of a particular list schedule (which we are about to specify) for the A^1-tasks. Consider the following schedule for the A^1-tasks (Figure 4.7b): In the first cx/2 time units two A^1-tasks with R_1-values of $1/2 - 2\varepsilon$ execute. These time units are followed by x time units in

Figure 4.7: The schedules used in Lemma 4.11.



a) An optimal schedule -- for each other resource v, A^v-tasks execute (in a similar manner) with these A^1-tasks.



b) The schedule used for $G(A^1)$ -- these tasks execute alone.

In each of the above figures, the values inside of the boxes indicate the R_1-values of the the tasks executing in those time units. The values under the boxes indicate the number of time units where tasks with those particular R_1-values execute.

which one A^1-task with an R_1-value of $1/2 + \varepsilon$ executes per time unit. Note that in each of the first $cx/2$ time units the total requirement for resource 1 is $2(1/2 - 2\varepsilon) = 1 - 4\varepsilon$. During the execution of these time units the smallest resource requirement of any unexecuted A^1-task is $1/2 - 2\varepsilon$, a value which exceeds 4ε. This means that none of the A^1-tasks which execute later in the schedule can execute in these time units. This assures that the schedule we have described here is a valid list schedule. Thus, $G(A^1) = cx/2 + x$, and $CPATH \geq x + s + 1 + s[cx/2 + x] > x(m+s+1)/2$.

$\therefore$ $CPATH/OPT \geq (x(m+s+1)/2)/(x+2s+1)$

$\lim_{x \to \infty} CPATH/OPT = (m+s+1)/2.$ □

Lemma 4.11: If $2s + 1 \leq m < 8s/3 + 1$, then CPATH/OPT can be arbitrarily close to $(4s+m+3)/4$.

Proof

Let $c = (m-2s-1)/s$. Note that $0 \leq c < 2/3$. Also, let $q = 0$ if $c \leq 1/2$ and let $q = \lceil \log [(1-c)/(2-3c)] \rceil$ if $c > 1/2$. Let x be an integer such that $x = 0 \bmod s2^q$, let $z = [2+c]x$, and let $Y = 3c/2 + (1-c)/2^{q-1}$. (The origin of Y will be explained a little later in the proof). Let $\varepsilon = \varepsilon_0 = 1/10^{q+2}$. Also, for $1 \leq k \leq q$, let $\varepsilon_k = 10\varepsilon_{k-1}$. Now consider the task system S^* using these values of x, z and ε. The remaining resource requirements of the A-tasks are:

For each i, $1 \leq i \leq s$:

1. $(1-c)x$ of the A^i-tasks have an R_i-value of $1/2 + \varepsilon_0$

   $(1-c)x$ of the A^i-tasks have an R_i-value of $1/2 - 2\varepsilon_0$.

2. For $0 \leq k \leq q-1$,

   $(1-c)x/2^k$ of the A^i-tasks have an R_i-value of $1/2 + \varepsilon_k$.

   $(1-c)x/2^k$ of the A^i-tasks have an R_i-value of $1/4 + 2\varepsilon_k$.

   $(1-c)x/2^k$ of the A^i-tasks have an R_i-value of $1/4 - 4\varepsilon_k$.

3. Yx of the A^i-tasks have an R_i-value of $1/2 + \varepsilon_q$.

   Yx of the A^i-tasks have an R_i-value of $1/4 + 2\varepsilon_q$.

Yx of the A^i-tasks have an R_i-value of $1/4 - 4\epsilon_q$.

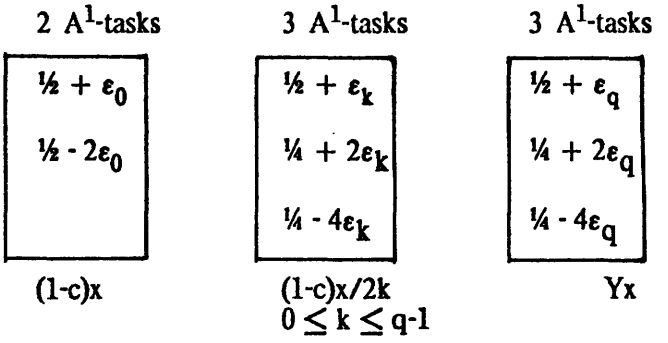
There are two cases to be considered here:

1. If $q = 0$, then no tasks are assigned resource requirements in part 2 of the above specifications. In this instance $Y = c$.
2. If $q > 0$, then some tasks are assigned resource requirements in part 2 of the specifications. Note that $Y \geq 0$, since $q < 1 + \log[(1-c)/(2-3c)]$.

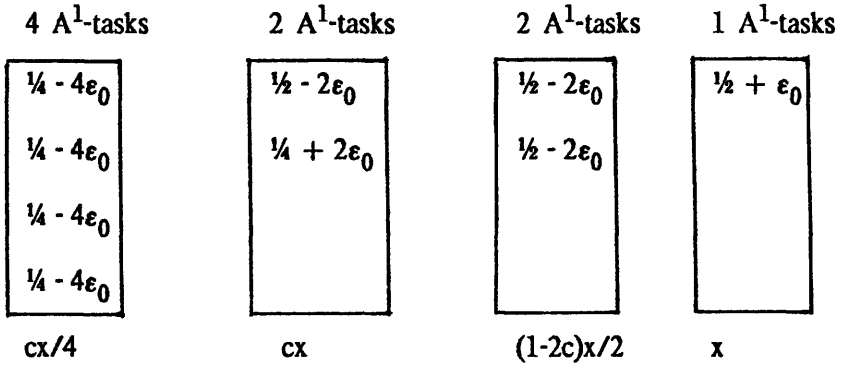
In both cases, resource requirements are specified for exactly z A^i-tasks. The constant Y was chosen so that this was the case. Intuitively, in part 2 of the specifications, we assign R_i-values to the tasks in a series of sets of tasks. The number of tasks in each set is one half the number of tasks in the preceding set. Since there are only $[2+c]x = z$ A^i-tasks, the series must be terminated at an appropriate point. In this instance, that is after q sets. The value 3Yx is the number of A^i-tasks whose R_i-value has not been specified when the series is terminated. These 3Yx tasks are the tasks assigned R_i-values in part 3 of the specifications.

As before, $OPT \leq x + 2s + 1$ provided all of the A-tasks can be executed in x time units with the total requirement for resource 1 by the A-tasks in each time unit not exceeding $1 - \epsilon$. This can be done by executing the following tasks at each of those x time units (Figure 4.8a): For each i, $1 \leq i \leq s$, either 2 or 3 A^i-tasks execute at each of the x time units. More specifically, for $(1-c)s = 3s-m+1$ values of i, two A^i-tasks execute. They have R_i-values of $1/2 + \epsilon_0$ and $1/2 - 2\epsilon_0$. For the other $cs = m-2s-1$ values of i, three A^i-tasks execute. They have R_i-values of $1/2 + \epsilon_k$, $1/4 + 2\epsilon_k$ and $1/4 - 4\epsilon_k$, for some k, $0 \leq k \leq q$. Since at each time unit $2(1-c)s + 3cs = m-1$ tasks execute, all of the A-tasks can be executed in x time units. Note that for each i, there are $(1-c)x$ time units in which two A^i-tasks execute and there are cx time units in which three A^i-tasks execute. Moreover, since $\epsilon_k \geq \epsilon_0 = \epsilon$ for $0 \leq k \leq q$, the total requirement for any resource during each time unit does not exceed $1 - \epsilon$. Thus, the A-tasks can be executed in just x time units, and $OPT \leq x + 2s + 1$.

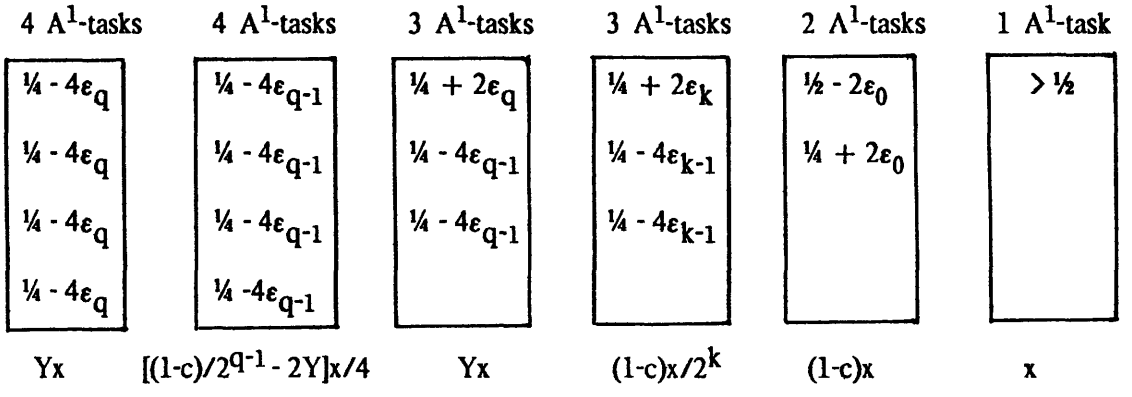
Figure 4.8: The schedules used in Lemma 4.12.



a) An optimal schedule -- for each other resource v, A^v-tasks execute (in a similar manner) with these A^1-tasks.



b) The schedule used for $G(A^1)$ when $q = 0$. These tasks execute alone.



c) The schedule used for $G(A^1)$ when $q > 0$. These tasks execute alone.

For critical path schedules, CPATH $\geq x + s + 1 + sG(A^1)$. There are two cases to consider based on the value of q (i.e. $q = 0$ and $q > 0$).

If $q = 0$, consider the following schedule for the A^1-tasks (Figure 4.8b): In the first cx/4 time units, four A^1-tasks with R_1-values of $1/4 - 4\varepsilon_0$ execute in each time unit. Next there are cx time units in which two A^1-tasks execute during each time unit. These tasks have R_i-values of $1/2 - 2\varepsilon_0$ and $1/4 + 2\varepsilon_0$. Thirdly, there are $(1-2c)x/2$ time units in which two A^1-tasks, each with an R_i-value of $1/2 - 2\varepsilon_0$, execute. Finally, there are x time units in which one A^1-task with an R_i-value of $1/2 + \varepsilon_0$ executes per time unit. Note that in each of the first cx/4 time units the total requirement for resource 1 is $4(1/4 - 4\varepsilon_0) = 1 - 16\varepsilon_0$. During the execution of these time units the smallest resource requirement of any unexecuted A^i-task is $1/4 - 4\varepsilon_0$, a value which exceeds $16\varepsilon_0$. This means that none of the A^1-tasks which execute later in the schedule can execute in these time units. Similar remarks can be made about each of the other time units in this schedule. This assures that the schedule we have described here is a valid list schedule. Thus, $G(A^1) = cx/4 + cx + (1-2c)x/2 + x = [3/2 + c/4]x$.

If $q > 0$, consider the following schedule for the A^1-tasks (Figure 4.8c): In the first Yx/4 time units four A^1-tasks with R_1-values of $1/4 - 4\varepsilon_q$ execute in each time unit. Next, there are $[(1-c)/2^{q-1} - 2Y]x/4$ time units in which four A^1-tasks with R_1-values of $1/4 - 4\varepsilon_{q-1}$ execute per time unit (since $q \geq \log[(1-c)/(2-3c)]$ this quantity is non-negative). In the next Yx time units, three A^1-tasks execute per time unit: these tasks have R_1-values of $1/4 + 2\varepsilon_q$, $1/4 - 4\varepsilon_{q-1}$, and $1/4 - 4\varepsilon_{q-1}$. Similarly, in the next $(1-c)x/2^{q-1}$ time units three A^1-tasks execute per time unit. These tasks have R_1-values of $1/4 + 2\varepsilon_{q-1}$, $1/4 - 4\varepsilon_{q-2}$, and $1/4 - 4\varepsilon_{q-2}$. Generally, for k, $q-1 \geq k \geq 1$, there are $(1-c)x/2^k$ time units with three A^1-tasks executing per time unit. These tasks have R_1-values of $1/4 + 2\varepsilon_k$, $1/4 - 4\varepsilon_{k-1}$, and $1/4 - 4\varepsilon_{k-1}$. Following these time units there are (1-c)x time units with two A^1-tasks executing per time unit: These tasks have R_1-values of $1/2 - 2\varepsilon_0$ and $1/4 + 2\varepsilon_0$. Finally, there are x time units

in which one A^1-task executes per time unit. Each of these tasks has an R_1-value exceeding 1/2. Note that in each of the first Yx/4 time units the total requirement for resource 1 is $4(1/4 - 4\varepsilon_q) = 1 - 16\varepsilon_q$. During the execution of these time units the smallest resource requirement of any unexecuted A^1-task is $1/4 - 4\varepsilon_q$, a value which exceeds $16\varepsilon_q$. This means that none of the A^1-tasks which execute later in the schedule can execute in these time units. Similar remarks can be made about each of the other time units in this schedule. This assures that the schedule we have described here is a valid list schedule. Thus, $G(A^1) = [Y/4 + ((1-c)/2^{q-1} - 2Y)/4 + Y + \Sigma_{k=1}^{q-1}(1-c)/2^k + (1-c) + 1]x = [3/2 + c/4]x$.

$\therefore$ In both cases, $G(A^1) = [3/2 + c/4]x$ and $CPATH \geq x + s + 1 + s[3/2 + c/4]x > x(4s+m+3)/4$.

$\therefore$ $CPATH/OPT \geq (x[4s+m+3]/4)/(x + 2s + 1)$

$\text{limit}_{x \to \infty} CPATH/OPT = (4s+m+3)/4.$ □

4.2.4.3 A useful set of independent tasks

In the next two lemmas, we make use of a set of tasks originally described by Johnson, et.al.[JDUGG]. We have modified this set of tasks slightly to better suit our purposes.

Given some resource (say, resource 1) and an integer y, we will describe a set of 3y - 1 independent tasks. Each task requires some non-zero portion of the resource. These tasks can be grouped into three sets of tasks: In the first set all of the tasks have R_1-values of approximately 1/6; in the second set the tasks have R_1-values of approximately 1/3; and in the third set the tasks have R_1-values exceeding 1/2. Within each set the tasks differ slightly in their resource requirements. For instance, in the first set some of the tasks have resource requirements exceeding 1/6 and some have requirements less than 1/6. There are y tasks in each of the first two sets and y - 1 tasks in the third.

More formally, assume that an integer y, with $y = 0 \bmod 10$ is given. Let δ be such that $0 < \delta << 18^{-y/10}$. Also, let $\delta_i = \delta\, 18^{y/10 - i}$ for $1 \leq i \leq y/10$. Consider the following three sets of tasks:

1. The first set contains y tasks, T^1_{ji} for $0 \leq j \leq 9$ and $1 \leq i \leq y/10$. These tasks have the following resource requirements for $1 \leq i \leq y/10$:

$$R_1(T^1_{0i}) = 1/6 + 33\delta_i$$

$$R_1(T^1_{1i}) = 1/6 - 3\delta_i$$

$$R_1(T^1_{2i}) = R_1(T^1_{3i}) = 1/6 - 7\delta_i$$

$$R_1(T^1_{4i}) = 1/6 - 13\delta_i$$

$$R_1(T^1_{5i}) = 1/6 + 9\delta_i$$

$$R_1(T^1_{6i}) = R_1(T^1_{7i}) = R_1(T^1_{8i}) = R_1(T^1_{9i}) = 1/6 - 2\delta_i$$

2. The second set contains y tasks, T^2_{ji} for $0 \leq j \leq 9$ and $1 \leq i \leq y/10$. These tasks have the following resource requirements for $1 \leq i \leq y/10$:

$$R_1(T^2_{0i}) = 1/3 + 46\delta_i$$

$$R_1(T^2_{1i}) = 1/3 - 34\delta_i$$

$$R_1(T^2_{2i}) = R_1(T^2_{3i}) = 1/3 + 6\delta_i$$

$$R_1(T^2_{4i}) = 1/3 + 12\delta_i$$

$$R_1(T^2_{5i}) = 1/3 - 10\delta_i$$

$$R_1(T^2_{6i}) = R_1(T^2_{7i}) = R_1(T^2_{8i}) = R_1(T^2_{9i}) = 1/3 + \delta_i$$

3. The third set contains y - 1 tasks, T^3_i for $1 \leq i \leq y\text{-}1$. Each task requires $1/2 + \delta$ of resource 1.

An optimal schedule for these 3y-1 tasks has length y. It consists of time units with the following tasks:

1. For $2 \leq j \leq 9$ and $1 \leq i \leq y/10$, a T^3-task and T^1_{ji} and T^2_{ji}
2. For $1 \leq i \leq y/10$, a T^3-task and T^1_{0i} and T^2_{1i}
3. For $1 \leq i < y/10$, a T^3-task and T^1_{1i} and $T^2_{0,i+1}$
4. $T^1_{1,y/10}$ and T^2_{01}

Now consider the list $(T^1_{01}, \ldots, T^1_{91}, T^1_{02}, \ldots, T^1_{92}, \ldots, T^1_{0,y/10}, \ldots, T^1_{9,y/10}, T^2_{01}, \ldots, T^2_{91}, \ldots, T^2_{0,y/10}, \ldots, T^2_{9,y/10}, T^3_1, \ldots, T^3_{y-1})$. This list results in a schedule with length 17y/10 - 1. This follows easily from

the results in [JDUGG]. We give an informal description of the schedule here. The schedule has y/5 time units in which 5 tasks from the first set execute per time unit and in which the total resource requirement in each of the time units exceeds 5/6; y/2 time units in which 2 tasks from the second set execute per time unit and in which the total resource requirement in each of the time units exceeds 2/3; and, y - 1 time units in which one task from the third set executes per time unit.

Now assume that y is fixed. Since each task in the system requires a non-zero portion of the resource, and since (in both of the schedules given above) each time unit has 5 or fewer executing tasks, there exists a $\beta_y > 0$, such that the resource requirement of every task can be reduced by β_y without changing either of the two schedules. Moreover, this implies that the total resource usage during any single time unit in these two schedules does not exceed $1 - \beta_y$.

In the next result, some A^i-tasks are assigned R_i-values in a manner similar to those assigned in previous lemmas, and some are assigned R_i-values similar to the resource requirements of the J-tasks.

4.2.4.4 The remaining cases

Lemma 4.12: If $8s/3 + 1 \leq m < 3s + 1$, then CPATH/OPT can be arbitrarily close to $(14s+m+9)/10$.

Proof

Let $c = (m-2s-1)/s$ and let $q > 0$ be an arbitrary integer. Note that $2/3 \leq c < 1$. Let $x = 20s2^{q-1}$, let $z = [2+c]x - 1$ and let $Y = 3c-2 + (1-c)/2^{q-1}$. The value Y will serve a purpose in this result similar to what it served in the previous result. Also similarly to the previous result, let $\varepsilon = \varepsilon_0 << \min\{ \beta_{Yx}, 1/10^{q+2} \}$ and for $1 \leq k \leq q$, let $\varepsilon_k = 10\varepsilon_{k-1}$. Now consider the task system S^* using these values of x, z, and ε. The remaining resource requirements of the A-tasks are as follows:

For each i, $1 \leq i \leq s$,

1. $(1-c)x$ of the A^i-tasks have an R_i-value of $1/2 + \varepsilon_0$.

   $(1-c)x$ of the A^i-tasks have an R_i-value of $1/2 - 2\varepsilon_0$.

2. For $0 \leq k \leq q-1$,

$(1\text{-}c)x/2^k$ of the A^i-tasks have an R_i-value of $1/2 + \varepsilon_k$.

$(1\text{-}c)x/2^k$ of the A^i-tasks have an R_i-value of $1/4 + 2\varepsilon_k$.

$(1\text{-}c)x/2^k$ of the A^i-tasks have an R_i-value of $1/4 - 4\varepsilon_k$.

3. $3Yx - 1$ of the A^i-tasks are assigned R_i-values equal to the R_1-values of the tasks in a set of $3Yx - 1$ J-tasks. These A^i-tasks will be called <u>type J</u> A^i-tasks.

An optimal schedule for this task system has a similar form for the execution of the A-tasks as the optimal schedules in the previous lemma. As before, OPT $\leq x + 2s + 1$ provided all of the A-tasks can be executed in x time units on m-1 processors. This can be done by executing the following tasks at each of those x time units: For $(1\text{-}c)s = 3s\text{-}m+1$ values of i, two A^i-tasks execute: these tasks have R_i-values of $1/2 + \varepsilon_0$ and $1/2 - 2\varepsilon_0$. For the <u>other</u> $cs = m\text{-}2s\text{-}1$ values of i, either:

1. Three A^i-tasks execute having R_i-values of $1/2 + \varepsilon_k$, $1/4 + 2\varepsilon_k$, and $1/4 - 4\varepsilon_k$ for some k, $0 \leq k \leq q\text{-}1$, or
2. Two or three type J tasks execute (as noted in section 4.3, three type J tasks execute in all but one of these time units).

Note that at each time unit no more than $2(1\text{-}c)s + 3cs = m\text{-}1$ tasks execute. Also, for each i, there are cx time units in which three A^i-tasks execute and there are $(1\text{-}c)x$ time units in which two A^i-tasks execute. Thus, the A-tasks can be executed in just x time units and the total requirement for any single resource during each time unit does not exceed $1 - \varepsilon$. Thus, OPT $\leq x + 2s + 1$.

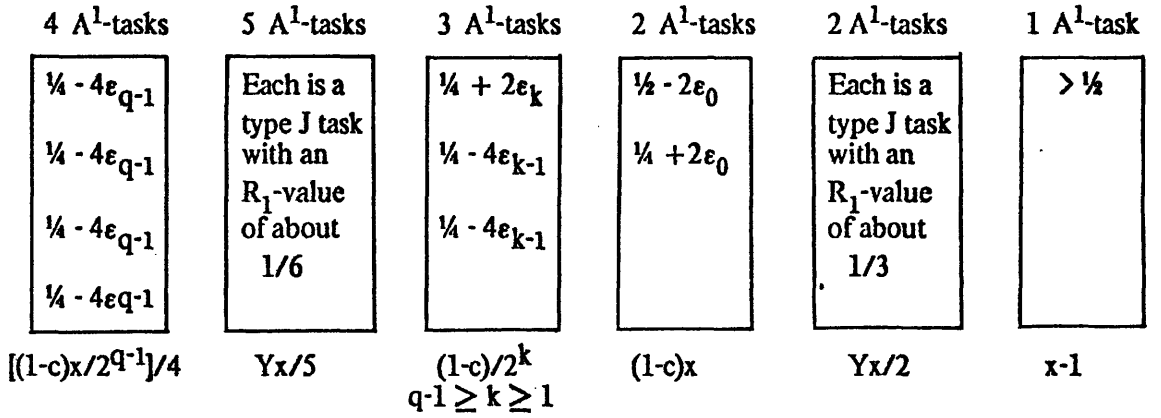
The execution of the A^1-tasks is also similar to that in the previous lemma. In that lemma (for $q > 0$), there were essentially four types of time units: those with 4, 3 , 2 or 1 tasks. Let T4, T3, T2 and T1 designate all of the time units of each type. Each of these types of time units will also occur here. In addition, in this proof, we have time units where only type J A^1-tasks execute. As indicated in our discussion in the previous section, there will be three types of time units where type J A^1-tasks execute. These time units contain 5, 2 and 1 tasks, and will be referred to as J5, J2 and J1,

respectively. The schedule used to derive $G(A^1)$ consists all of these time units in the following order: T4, J5, T3, T2, J2, J1 and T1. That is, first all of the T4 time units execute, then all of the J5 time units execute, and so on.

More formally, consider the following schedule for the A^1-tasks (Figure 4.9): In the first $[(1-c)x/2^{q-1}]/4$ time units four A^1-tasks, each with an R_1-value of $1/4 - 4\varepsilon_{q-1}$, execute in each time unit. Next, there are $Yx/5$ time units in which five type J tasks execute - as noted in the previous section, each of these tasks has an R_1-value of approximately 1/6. Next, similarly to the critical path schedule described in Lemma 4.11, for $q-1 \geq k \geq 1$, there are $(1-c)x/2^k$ time units with three tasks executing per time unit. These tasks have R_1-values of $1/4 + 2\varepsilon_k$, $1/4 - 4\varepsilon_{k-1}$, and $1/4 - 4\varepsilon_{k-1}$. Following these time units there are $(1-c)x$ time units with two A^1-tasks executing per time unit. These tasks have R_1-values of $1/2 - 2\varepsilon_0$ and $1/4 + 2\varepsilon_0$. Next, there are $Yx/2$ time units with two type J tasks executing per time unit - as noted in the previous section, these tasks have R_1-values of approximately 1/3. Finally, there are x-1 time units in which one A^1-task executes per time unit. Each of these tasks has an R_1-value exceeding 1/2. Note that in each of the first $[(1-c)/2^{q-1}]x/4$ time units the total requirement for resource 1 is $4(1/4 - 4\varepsilon_{q-1}) = 1 - 16\varepsilon_{q-1}$. During the execution of these time units the smallest resource requirement of any unexecuted A^i-task is approximately 1/6 (actually, just a little less than 1/6). But, ε_{q-1} was chosen such that $1/6 >> 16\varepsilon_{q-1}$. This means that none of the A^1-tasks which execute later in the schedule can execute in these time units. Similar remarks can be made about each of the other time units in this schedule. This assures that the schedule we have described here is a valid list schedule. Thus, $G(A^1) = ([(1-c)/2^{q-1}]/4 + Y/5 + \Sigma_{k=1}^{q-1}(1-c)/2^k + (1-c) + Y/2 + 1)x - 1 = [(16+c)/10 - (1-c)/(20\,2^{q-1}]x - 1$. Hence, $CPATH \geq x + s + 1 + sx[(16+c)/10 - (1-c)/(20\,2^{q-1})] - s$. But, $x = 20s2^{q-1}$, so $CPATH > x[s(16+c)/10 + 1] - s^2$.

$\therefore CPATH/OPT \geq (x[s(16+c)/10 + 1] - s^2)/(x+2s+1)$

Figure 4.9: The schedule used for $G(A^1)$ in Lemma 4.13.



$\text{limit}_{x \to \infty}$ CPATH/OPT $\geq (14s+m+9)/10$. □

Lemma 4.13: If $3s + 1 \leq m$ and $m \geq 10$, then CPATH/OPT can be arbitrarily close to

$$2 + 17s/10 - (3s+1)/m.$$

Proof

Let $x = 0 \bmod 10m$, let $z = 3x - 1$ and let $\varepsilon = \beta_x$. Consider the task system S^* using these values of x, z and ε. For each i, $1 \leq i \leq s$, the A^i-tasks are assigned R_i-values equal to the R_1-values of the tasks in a set of z J-tasks. In addition to the usual tasks in S^* the following tasks are added to S^*:

1. G, a task which requires no resources.
2. F_j for $1 \leq j \leq (m-3s-1)x$. These tasks require no resources.
3. E with $R_i(E) = 1$ for $1 \leq i \leq s$.

The following precedence constraints are also added to the system:

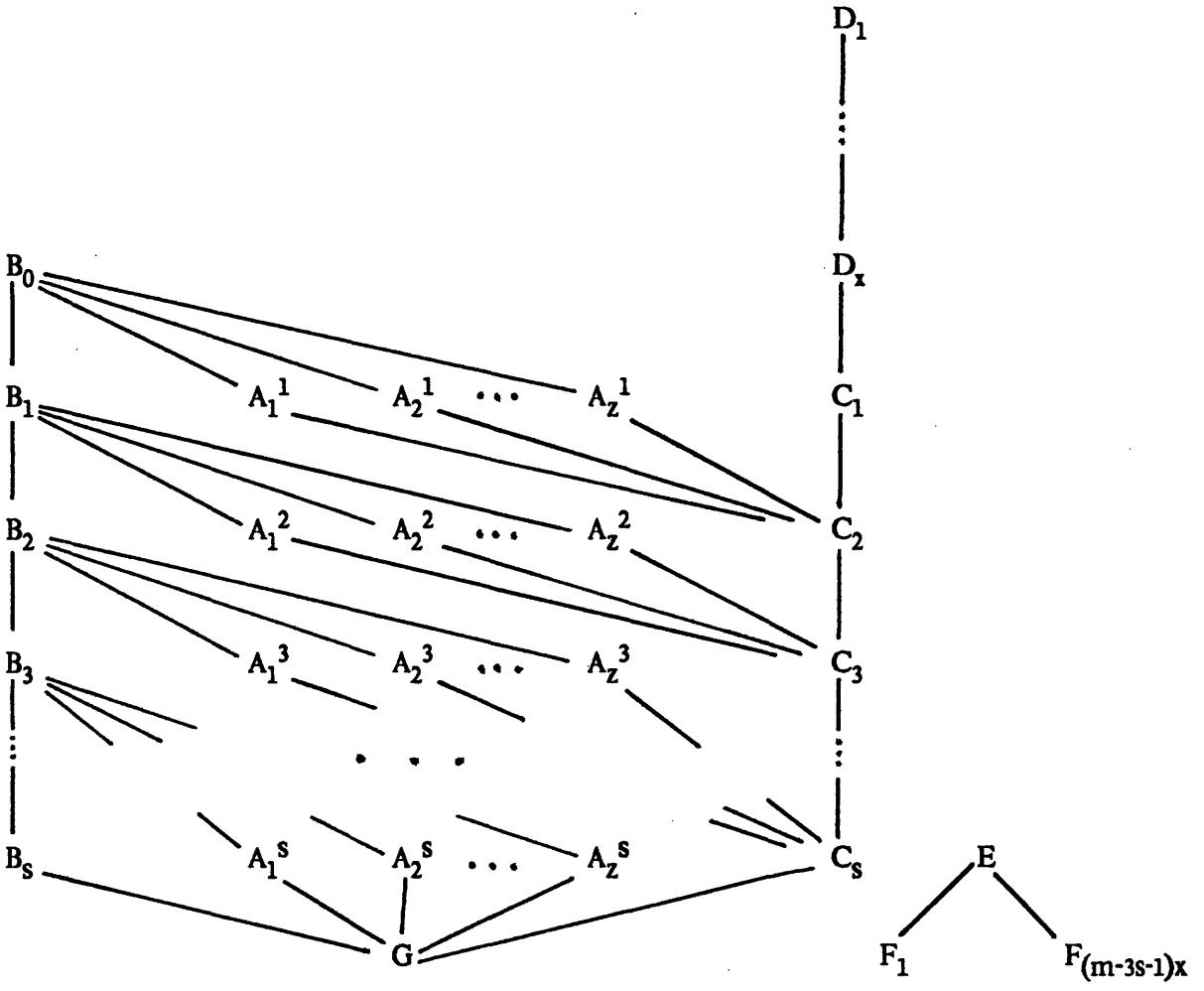
1. For $1 \leq j \leq z$, $A_j^s \prec G$.
2. $B_s \prec G$, and $C_s \prec G$.
3. For $1 \leq j \leq (m-3s-1)x$, $E \prec F_j$.

The precedence structure of this task system is shown in Figure 4.10.

An optimal schedule for this system is: In the first s+2 time units execute the B-tasks and task E. In the next x time units the A-tasks, D-tasks and F-tasks are executed (1 D-task, m-3s-1 F-tasks and no more than 3s A-tasks per time unit). For each i, there are x-1 time units where three A^i-tasks execute and there is one time unit where two A^i-tasks execute. In the final s+1 time units execute the C-tasks followed by task G. Thus OPT $\leq s + 2 + x + s + 1 = x + 2s + 3$.

Now consider the following critical path schedule: Execute the D-tasks and tasks B_0 and C_1 in the first x+1 time units. In the next 17x/10 - 1 time units execute the A^1-tasks. Then, execute B_1 and C_2, followed by the A^2-tasks in the next 17x/10 - 1 time units, and so on, until B_s executes. Then

Figure 4.10: The task system used in Lemma 4.14.



The non-zero resource requirements of these tasks are:

Each D-task requires ε of resource 1

B_0 requires all of resource 1

B_i requires all of resource i, $i > 0$

Each A^i-task requires a non-zero portion of resource i

E requires all of the resources

G, the C-tasks and the F-tasks require no resources

exccute E and G. In the final (m-3s-1)x/m time units execute the F-tasks. Thus, CPATH $\geq$ x + 1 + 17xs/10 + 1 + (m-3s-1)x/m > x[2 + 17s/10 - (3s+1)/m].

$\therefore$ CPATH/OPT $\geq$ x[2 + 17s/10 - (3s+1)/m]/(x + 2s + 3)

$\lim_{x \to \infty}$ CPATH/OPT = 2 + 17s/10 - (3s+1)/m. □

Lemma 4.14: If 3s + 1 $\leq$ m and m < 10, then CPATH/OPT can be arbitrarily close to

2 + 5s/3 - (8s/3 + 1)/m.

Proof

The task system we describe here combines various aspects of the systems used in Lemmas 4.11 and 4.13. We use the task system structure from Lemma 4.13 (i.e. with the added tasks) and we assign the A-tasks resource requirements as was done in Lemma 4.11.

More formally, assume s and m are given. Let c $\in$ (1/2, 2/3) and let q = $\lceil \log[(1-c)/(2-3c)] \rceil$. Let x be an integer such that x = 0 mod $sm2^q$, let z = [2+c]x and let Y = 3c-2 + $(1-c)/2^{q-1}$. Let ε = $\varepsilon_0 = 1/10^{q+2}$. Also, for 1 $\leq$ k $\leq$ q, let $\varepsilon_k = 10\varepsilon_{k-1}$. Consider the task system S^* using these values of x, z and ε.

For each i, 1 $\leq$ i $\leq$ s:

1. (1-c)x of the A^i-tasks have an R_i-value of 1/2 + ε_0

   (1-c)x of the A^i-tasks have an R_i-value of 1/2 - $2\varepsilon_0$.

2. For 0 $\leq$ k $\leq$ q-1,

   $(1-c)x/2^k$ of the A^i-tasks have an R_i-value of 1/2 + ε_k.

   $(1-c)x/2^k$ of the A^i-tasks have an R_i-value of 1/4 + $2\varepsilon_k$.

   $(1-c)x/2^k$ of the A^i-tasks have an R_i-value of 1/4 - $4\varepsilon_k$.

3. Yx of the A^i-tasks have an R_i-value of 1/2 + ε_q.

   Yx of the A^i-tasks have an R_i-value of 1/4 + $2\varepsilon_q$.

   Yx of the A^i-tasks have an R_i-value of 1/4 - $4\varepsilon_q$.

These are exactly the same specifications for the R_i-values of the A^i-tasks as given in Lemma 4.12.

In addition to the usual tasks in S^*, the following tasks are added to S^*:

1. G, a task which requires no resources.

2. F_j for $1 \leq j \leq (m-[2+c]s-1)x$. These tasks require no resources.

3. E with $R_i(E) = 1$ for $1 \leq i \leq s$.

The following precedence constraints are also added to the system:

1. For $1 \leq j \leq z$, $A_j^s < G$.

2. $B_s < G$, and $C_s < G$.

3. For $1 \leq j \leq (m-3s-1)x$, $E < F_j$.

An optimal schedule for this system is similar to that for the system used in the proof of the previous lemma. The B-tasks and task E are executed in the first s+2 time units. In the next x time units the A-tasks, D-tasks and F-tasks are executed. In each of those x time units, [2+c]s A-tasks, 1 D-task and (m-[2+c]-1) F-tasks execute. For each i, there are (1-c)x time units where two A^i-tasks execute and there are cx time units where three A^i-tasks execute. In the final s time units the C-tasks are executed. Thus, $OPT \leq x + 2s + 2$.

Now consider the following critical path schedule: Execute the D-tasks and tasks B_0 and C_1 in the first x+1 time units. In the next $[3/2 + c/4]x$ time units execute the A^1-tasks (this follows from the proof of Lemma 4.13, where $G(A^1) = [3/2 + c/4]x$). Then execute B_1 and C_2, followed by the A^2-tasks in the next $[3/2 + c/4]x$ time units, and so on, until B_s executes. Next execute E and G. Finally, execute the F-tasks in the final $(m-[2+c]s-1)x/m$ time units. Thus, $CPATH \geq x + 1 + ([3/2 + c/4]x + 1)s + 1 + (m-[2+c]s-1)x/m > x[2 + 3s/2 - (2s+1)/m + cs(1/4 - 1/m)]$.

$\therefore$ $CPATH/OPT \geq x[2 + 3s/2 - (2s+1)/m + cs(1/4 - 1/m)]/(x + 2s + 2)$

$\lim_{c \to 2/3} CPATH/OPT \geq x[2 + 5s/3 - (8s/3 + 1)/m]/(x + 2s + 2)$

$\lim_{x \to \infty} CPATH/OPT = 2 + 5s/3 - (8s/3 + 1)/m$ □

Chapter 5 - Critical Path Scheduling - Discrete Resources

In this chapter we study critical path scheduling of UET task systems with discrete resources - both with and without processor constraints. Unfortunately, there are no results for this problem per se. It is possible, however, to make some conclusions about this problem based on results for Coffman-Graham scheduling of UET task systems with 0-1 resources. These are UET task systems with discrete resources in which each $r_i = 1$ -- that is, there is exactly one unit of each resource, hence a task either requires all of a resource or none of it. Because Coffman-Graham schedules are a subclass of the critical path schedules, any lower bound on CG/OPT for UET task systems with 0-1 resources, is also a lower bound on CPATH/OPT for UET task systems with discrete resources. This follows because systems with 0-1 resources are a subclass of the systems with discrete resources. Although at first glance, it appears that any lower bound on CPATH/OPT obtained in this manner would be fairly weak, we will, in fact. (in section 5.2) be able to use such a lower bound to make some fairly strong statements about critical path scheduling of UET task systems with discrete resources. Before doing so, however, we present two results on Coffman-Graham scheduling of UET task systems with 0-1 resources.

5.1 Coffman-Graham scheduling of systems with 0-1 resources

Coffman-Graham scheduling of UET task systems with 0-1 resources has been studied by Goyal [Go] for the limited case of one resource. He shows that for m = 2, CG/OPT $\leq$ 3/2, and that this is the best possible result. This type of scheduling is also mentioned by Leung [Le]. He conjectures that for UET task systems with 0-1 resources, Coffman-Graham schedules provide substantially better performance than do list schedules.

For purposes of comparison, we note that the results of Chapter 3 can be applied to UET task systems with 0-1 resources giving the results LIST/OPT $\leq 1 + s$ if there is no processor constraint, and LIST/OPT $\leq \min\{m, (2-1/m) + s(1-1/m)\}$ if there is a processor constraint. Moreover, both of these

results are the best possible bounds.

In this section we prove the following two results on Coffman-Graham scheduling of UET task systems with s 0-1 resources when $s \geq 0$:

Theorem 5.1: If $m \geq n$ (no processor constraint) then $CG/OPT \leq 1 + s$. Moreover, this is the best possible result.

Theorem 5.2: If $m \geq 2$ (a processor constraint) then

$$CG/OPT \leq \begin{cases} m & \text{if } s \geq m \\ m - 1/2 & \text{if } s = m - 1 \\ (2-2/m) + s(1-1/m) & \text{if } s \leq m - 2 \end{cases}$$

Moreover, this is the best possible result.

These results show that Leung's conjecture about the relationship between Coffman-Graham scheduling and list scheduling is wrong: Coffman-Graham scheduling does not provide substantially better worst case performance than list scheduling for UET task systems with 0-1 resources. In fact, for systems with no processor constraints, Coffman-Graham scheduling has exactly the same worst case performance as list scheduling. We will prove these two theorems, and then, in section 5.2, we will discuss how these results apply to critical path scheduling of UET task systems with discrete resources.

5.1.1 The upper bounds

Lemma 5.1: If $m \geq n$ (no processor constraint), then $CG/OPT \leq 1 + s$.

Proof

This result is trivial because Coffman-Graham schedules are a subclass of list schedules and as noted above, it follows from Theorem 3.1, for UET task systems with 0-1 resources and no processor constraint that $LIST/OPT \leq 1 + s$. □

Lemma 5.2: If $m \geq 2$ (a processor constraint), then

$$CG/OPT \leq \begin{cases} m & \text{if } s \geq m \\ m - 1/2 & \text{if } s = m - 1 \\ (2-2/m) + s(1-1/m) & \text{if } s \leq m - 2 \end{cases}$$

5.1.1.1 Proof Outline

We prove the upper bound in two stages. Initially, we show that, given a Coffman-Graham schedule, some of the tasks can be placed into sets $W_0, \ldots, W_p$ (called segments) such that given tasks $T \in W_i$ and $S \in W_{i+1}$, it must be that $T <^+ S$, where $<^+$ is the transitive closure of the precedence relation. This property implies that all the tasks in segment W_i must execute before any of the tasks in W_{i+1} execute. This allows us to examine each segment individually, and obtain a worst case bound for the length of the portion of the Coffman-Graham schedule where the tasks in the segment execute, to the length of an optimal schedule for the tasks in the segment. This we do in the second stage of the proof. A portion of this proof is largely a modification (to accomodate resource tasks) of a proof by Lam and Sethi [LS]. In particular, most of the first stage of the proof and the second half of the second stage of the proof are drawn from their work.

5.1.1.2 Segments

Before beginning, we make the following assumption about how tasks are assigned to processors when using list schedules (our formal definition did not mention which tasks execute on which processors). Since we are dealing with UET task systems, this assignment is relatively simple: If $T_1, \ldots, T_x$, with $x \leq m$, are the tasks executing in a particular time unit, with $LABEL(T_1) > LABEL(T_2) > \ldots > LABEL(T_x)$, then task T_i executes on processor i. Here $LABEL(T_i)$ refers to the label assigned to T_i using the Coffman-Graham labeling algorithm. Note that in the list used to do the scheduling, T_1 appears before T_2, which appears before T_3, and so on.

Finally, a task T with $R_{max}(T) = 0$ is a non-resource task, and a task T with $R_{max}(T) > 0$ is a

resource task.

Now consider any Coffman-Graham schedule. As usual, we let CG refer to both the set of time units comprising the Coffman-Graham schedule and the length of that schedule. As noted above, we will form sets of tasks called segments. This is done in two stages. First, we form blocks of tasks, and then combine those blocks to form segments. Blocks are formed from the Coffman-Graham schedule as follows:

Definiton: Form blocks $X_q, X_{q-1}, \ldots, X_0$, for some $q > 0$, as follows:

1. U_0 is the task executed on processor one in time unit B_{CG}.
2. For $i \geq 1$, U_i is the task executed on processor one in the maximal time unit B_λ where:
   a. A non-resource task executes on processor one in B_λ.
   b. $(\forall T \neq U_i)[T \in B_\lambda \Rightarrow \text{LABEL}(T) < \text{LABEL}(U_{i-1})]$.
3. For $q \geq i \geq 1$, $X_{i-1} = \{T : \sigma(U_i) < \sigma(T) \leq \sigma(U_{i-1}) \text{ and LABEL}(T) \geq \text{LABEL}(U_{i-1})\}$

   $X_q = \{T : \sigma(T) \leq \sigma(U_q) \text{ and LABEL}(T) \geq \text{LABEL}(U_q)\}$

An example is shown in Figure 5.1. Note that not every task belongs to a block - such a task is called an extra task. The last time unit of each block either contains an extra task or it has an idle processor. Also, for block X_j, $\sigma(X_j) = \min\{\sigma(T): T \in X_j\}$. That is, $\sigma(X_j)$ is the earliest time at which a task of block X_j executes.

The following lemma about blocks is useful:

Lemma 5.3: For $q \geq i > 0$, task U_i is a predecessor of each task in block X_{i-1}.

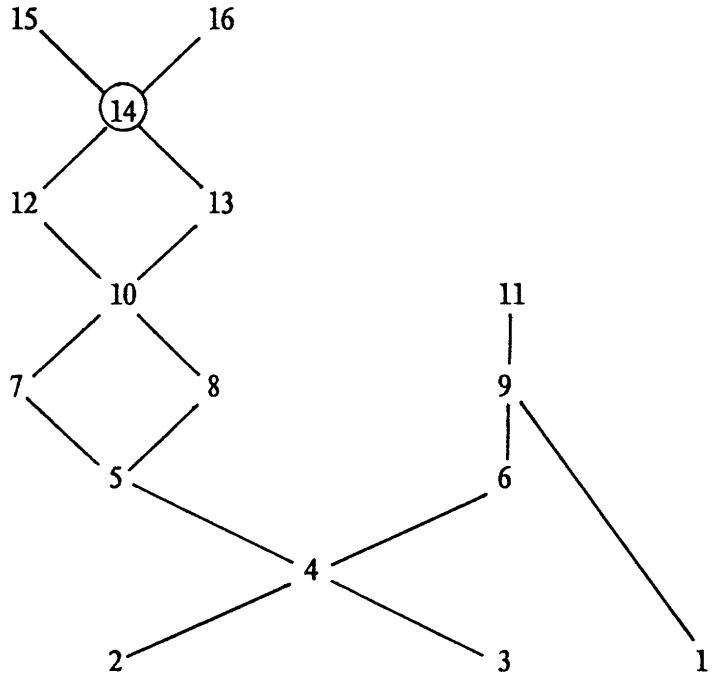
Proof

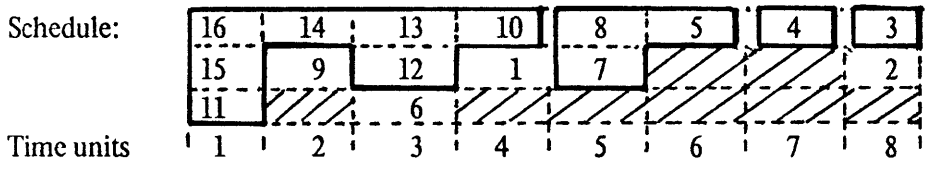
Consider any U_i and block X_{i-1}. Three things should be noted:

1. U_i is a non-resource task.
2. Each task in X_{i-1} has a label at least as large as $\text{LABEL}(U_{i-1})$.
3. Each task executed in the same time unit as U_i has a label smaller than $\text{LABEL}(U_{i-1})$.

Figure 5.1: Example of the division of a Coffman-Graham schedule into blocks.

Consider a task system with 3 processors and one 0-1 resource. The precedence structure is given below. The numbers are the Coffman-Graham labels of the tasks. These numbers will be used to refer to the tasks. Circled tasks require the resource.

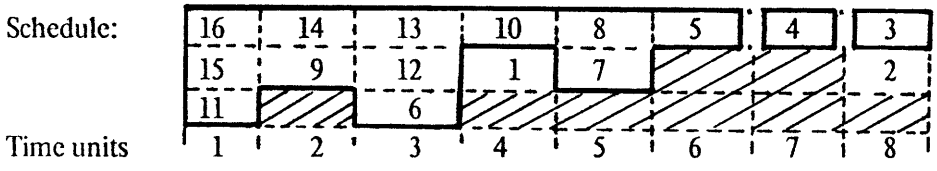




Blocks are outlined in the above schedule.

Figure 5.2: Example of the division of a Coffman-Graham schedule into segments.

The task system given in Figure 5.1 is used.



Segments are outlined in the above schedule.

Now consider any task $T \in X_{i-1}$ which has no predecessors in X_{i-1}. Why didn't T execute in the same time unit as U_i? Because LABEL(T) exceeds the label of each task executing with U_i, and U_i is a non-resource task, it follows that $U_i \prec T$. Thus, U_i is a predecessor of every task in X_{i-1}. □

Segments are composed of blocks and a few extra tasks. Specifically, segments $W_0, \ldots, W_p$, for some $p \geq 0$, are formed as follows:

1. Initially, let $W_0 = X_q$, let $i = 0$ and let $j = q - 1$.

2. While $j \geq 0$ do

   if $(\forall T \in W_i)(\forall T' \in X_j)[T \prec^+ T']$

   then W_i is complete

   let $W_{i+1} = X_j$, let $i = i + 1$, and let $j = j - 1$

   else let $G = \{E \notin W_i : \text{LABEL}(E) > \text{LABEL}(U_j)$ and $(\exists T \in W_i)[T \prec^+ E]\}$

   let $W_i = W_i \cup X_j \cup G$ and let $j = j - 1$

3. Let $p = i$, and W_p is complete.

An example showing segments is given in Figure 5.2. Intuitively, segments are formed from left to right by combining successive blocks until a block is encountered, all of whose tasks are successors of all the tasks already in the segment. At this point the segment is complete and a new segment is started. Extra tasks are added to the segment for accounting purposes which arise in the second stage of the proof. Extra tasks which are placed into a segment are called latecomers.

Lemma 5.4: For $0 \leq i < p$, if $T \in W_i$ and $T' \in W_{i+1}$, then $T \prec^+ T'$.

Proof

Consider any W_i and W_{i+1} for some i, $0 \leq i < p$. Assume that segment W_{i+1} consists of blocks $X_c, \ldots, X_{c-k}$, for some $k \geq 0$, along with some latecomers. It follows from the construction of segments, for each $T \in W_i$ and $T' \in X_c$, that $T \prec^+ T'$. If $k = 0$, it also follows that there are no latecomers in W_{i+1}, so the lemma holds. Thus, assume $k > 0$. From Lemma 5.3, for all j, $c \geq j > c-k$, task U_j

precedes each task in X_{j-1}. Then by transitivity, each task $T \in W_i$, precedes each task in $X_c \cup X_{c-1} \cup \ldots \cup X_{c-k}$. The only other tasks in W_{i+1} are latecomers. The first latecomer added to W_{i+1} is, by definition, a successor of a task in X_c. Each subsequent latecomer to W_{i+1} is a successor of either a task in some block of W_{i+1} or of a latecomer already in W_{i+1}. In either case, by transitivity, each $T \in W_i$ precedes each latecomer in W_{i+1}. □

Because of the preceding lemma, we are free to treat each segment individually with respect to obtaining an upper bound. That is, because each task in segment W_i must execute before any task in W_{i+1} can execute, we have that $OPT \geq \Sigma_{i=0}^{p} OPT_i$, where OPT is the length of an optimal schedule for the entire task system, and OPT_i is the length of an optimal schedule for a task system consisting of the tasks in W_i, (and the precedence constraints restricted to those tasks). Moreover, $CG = \Sigma_{i=0}^{p} CG_i$, where CG is the length of a Coffman-Graham schedule for the entire task system, and CG_i is the length of the portion of the Coffman-Graham schedule under consideration restricted to the tasks in W_i. The equality follows because at least one task from each time unit belongs to some segment. In the next section we show that for each i, $0 \leq i \leq p$, $CG_i/OPT_i \leq b$, where b depends on the relationship of s and m. It follows that, given a particular relationship between s and m, $CG/OPT \leq b$. Thus, in the remainder of the proof we assume that the Coffman-Graham schedule consists of a single segment W. That segment consists of blocks $X_q, \ldots, X_0$, and some number of latecomers. We let OPT be an optimal schedule for the tasks in W.

5.1.1.3 The individual bounds

In this section we complete the proof of the upper bound. As noted previously, m is a trivial upper bound on CG/OPT. This handles the case of $s \geq m$. Moreover, Goyal [Go] has shown that $CG/OPT \leq 3/2$ if $s=1$ and $m=2$, and it has been shown [CG,LS] that $CG/OPT \leq 2 - 2/m$ if $s=0$ and $m \geq 2$. Thus, we assume that $s \geq 1$ and $m \geq 3$ in the remainder of this proof.

The following lemma about segments is useful:

Lemma 5.5: If W contains blocks $X_q, \ldots, X_0$, then there are at least q latecomers in W.

Proof

We consider the procedure by which segments are formed, and show that each time the else-clause in step 2 of that procedure is executed, at least one latecomer is added to W. Since the else-clause is executed for each block added to W (except the first block), the lemma follows.

Assume that blocks $X_q, \ldots, X_{j+1}$ are already in W (along with latecomers) and that there are tasks $T \in W$ and $T' \in X_j$ such that $T \prec^+ T'$ is false. Choose T so that it has no successors in W and T' so that it has no predecessors in X_j. Let $I = \{T \in X_j: T \text{ has no predecessors in } X_j\}$. Clearly, $T' \in I$. Now consider U_{j+1}. By definition $U_{j+1} \in W$. From Lemma 5.3, U_{j+1} is a predecessor of each task in X_j. It follows from there being no transitive edges in the dag for $\prec$, that when labeling U_{j+1}, the largest $|I|$ labels of its successors are the labels of the tasks in I. Now consider task T. By definition, $\text{LABEL}(T) > \text{LABEL}(U_{j+1})$. Since T has no successors in W, and $T \prec^+ T'$ is false, it follows that there is a task $E \notin W$ such that $\text{LABEL}(E) > \text{LABEL}(U_j)$ and $\sigma(E) < \sigma(X_j)$. Intuitively, the first condition holds because LABEL(E) must exceed the label of some task in I, since $\text{LABEL}(T) > \text{LABEL}(U_{j+1})$. The second condition holds since E is not in X_j. Therefore, each time the else-clause is executed in the procedure defining segments, at least one latecomer is added to W. □

5.1.1.3.1 The case $s = m - 1$

Given a segment W, let a be the number of resource tasks in W and let d be the number of time units in the Coffman-Graham schedule having a resource task executing on processor one.

Lemma 5.6: $CG \leq (m\,OPT + a + 1)/2$

Proof

From the constructions of blocks and segments it follows that for each time unit $B \in CG$, not having a resource task executing on processor one, that one of the following holds:

1. B is the last time unit in W.
2. B is not the final time unit of any block. This means that there are at least two tasks of W which are not latecomers and execute in B.
3. B is the final time unit of block X_i, for some $i \neq 0$ (i.e. not the last block). This means that at least one latecomer was placed into W when block X_{i-1} was added to W.

Note that there are CG - d time units not having a resource task executing on processor one, and for only one of these time units can item 1 (above) hold. Thus, $d + 2[CG - d - 1] + 1 = 2\,CG - d - 1$ is a lower bound on the number of tasks in W. Since m OPT is an upper bound on the number of tasks in W, we have $m\,OPT \geq 2\,CG - d - 1$.

Clearly $d = a - k$ for some $k \geq 0$, hence, $m\,OPT \geq 2\,CG - [a - k] - 1$.

$\therefore CG \leq (m\,OPT + a + 1)/2 - k/2$

$\leq (m\,OPT + a + 1)/2$ □

Three corollaries follow directly from the proof of the above lemma:

Corollary 5.1: If a resource task executes on any processor other than processor one, then

$CG \leq (m\,OPT + a)/2$.

Corollary 5.2: If $m\,OPT \geq 2\,CG - d$, then $CG \leq (m\,OPT + a)/2$.

Corollary 5.3: If any time unit with a resource task executing on processor one, has a task $T \in W$, executing on processor two, and T is not a latecomer, then $CG \leq (m\,OPT + a)/2$.

To complete the proof for $s = m - 1$ there are three cases to consider:

Case 1: A resource task executes on a processor other than processor one.

From Corollary 5.1, it follows that $CG/OPT \leq (m\,OPT + a)/(2\,OPT)$. But $a \leq (m - 1)OPT$, since there are only m - 1 units of resource available at each time unit of OPT.

$\therefore CG/OPT \leq (m\,OPT + (m - 1)\,OPT)/(2\,OPT)$

$= m - 1/2$

Case 2: Each resource task executes on processor one and $a \leq (m - 1)\,OPT - 1$.

From Lemma 5.6, $CG/OPT \leq (m\,OPT + a + 1)/(2\,OPT)$

$$\leq (m\,OPT + (m - 1)\,OPT)/(2\,OPT)$$

$$= m - 1/2$$

Case 3: Each resource task executes on processor one and $a = (m - 1)\,OPT$.

These conditions mean that in each time unit of OPT, m - 1 tasks require a resource, and that each resource task requires exactly one unit of one resource. In particular, consider the first time unit of OPT. Since $m \geq 3$, (hence $s \geq 2$), there are at least two resource tasks executing in that time unit. Let T_1 and T_2 be two such tasks. In the Coffman-Graham schedule, T_1 and T_2 both execute on processor one. Without loss of generality, assume that T_1 executes before T_2. There are only three possible reasons why T_2 did not execute with T_1 in the Coffman-Graham schedule:

1. Due to processor constraints. That is, when T_2 was scheduled, the only reason that it was not scheduled to execute with T_1, was that the time unit where T_1 executes already contained m tasks. Let T_3 be the task which executes on processor two. It follows that $LABEL(T_1) > LABEL(T_3) > LABEL(T_2)$, and that $\sigma(T_1) \leq \sigma(T_3) < \sigma(T_2)$. From Lemma 5.3, since T_1 and T_2 have no predecessors in W, it follows that T_1 and T_2 are in block X_q. Then, from the definition of blocks, $T_3 \in X_q$, hence $T_3 \in W$. Thus, the time unit where T_1 executes has a resource task executing on processor one and a task $T_3 \in W$ on processor two. Since T_3 is not a latecomer, from Corollary 5.3, $CG \leq (m\,OPT + a)/2$. As in Case 1, $CG/OPT \leq m - 1/2$.
2. Due to precedence constraints. That is, some task $T_3 < T_2$ had not executed prior to time unit $\sigma(T_1)$ in the Coffman-Graham schedule. It follows that $LABEL(T_1) > LABEL(T_3) > LABEL(T_2)$ and that $\sigma(T_1) \leq \sigma(T_3) < \sigma(T_2)$. As above, it follows that T_3 is in W. But this is a contradiction, since T_3 must execute before T_2 in OPT and T_2 executes in the first time unit of OPT.

3. Due to resource constraints. That is, some task T_3 executes in the same time unit of the Coffman-Graham schedule as T_1 and requires the same resource as T_2. It follows that $LABEL(T_1) > LABEL(T_3) > LABEL(T_2)$ and that $\sigma(T_1) \leq \sigma(T_3) < \sigma(T_2)$. As above, it follows that T_3 is in W. But this is a contradiction since T_3 is a resource task, and it doesn't execute on processor one.

This completes the proof for the case s = m-1. □

5.1.1.3.2 The case $s \leq m - 2$

Given a segment W, the time units of the Coffman-Graham schedule can be partitioned into the following three sets:

$F = \{B \in CG: |B| = m \text{ and } (\forall T \in B)[T \in W \text{ and } T \text{ is not a latecomer}]\}$

$H = \{B \in CG: B \notin F \text{ and } (\exists T \in B)[T \in W \text{ and } T \text{ is not a latecomer and } T \text{ is a resource task}]\}$

$P = CG - F - H$

It follows that for each $B \in P$, either B has an idle processor or there is an extra task in B (this extra task may or may not be a latecomer). The time units in F are full time units, those in H are resource time units and those in P are partial time units.

Lemma 5.7: If the first time unit of CG is either a full or resource time unit, then $OPT \geq |P| + 1$.

Proof

Consider the partial time units of W and number them (left to right) from 1 to |P|. For $1 \leq i < |P|$, let V_i be the task executed on processor one in the time unit immediately following partial time unit i. Let T* be the task executed on processor one in partial time unit 1. There are two observations to be made:

1. $T^* < V_1$. To see that this is so, consider the time unit where T* executes. Since this is a partial time unit, any extra tasks in this time unit have a label smaller than $LABEL(V_1)$. Since V_1 executes after time unit $\sigma(T^*)$, for some task T executing in that time unit, $T < V_1$. Suppose T

$\neq T^*$. Since LABEL(T^*) > LABEL(T), and V_1 is the task with the highest label that either T or T^* can precede, it must be that $T^* \prec V_1$.

2. For $1 \leq j \leq |P| - 1$, every $T \in W$, such that LABEL(T) $\geq$ LABEL(V_j), precedes a task $R \in W$, such that LABEL(R) $\geq$ LABEL(V_{j+1}). To see that this is so, consider any task T with LABEL(T) $\geq$ LABEL(V_j). If $T \prec V_{j+1}$, the claim holds, so assume not. Let T' be the task executed on processor one in partial time unit $j+1$. Similarly to the previous observation, $T' \prec V_{j+1}$. It follows from LABEL(T) $\geq$ LABEL(V_j) and LABEL(V_j) $\geq$ LABEL(T'), that LABEL(T) $\geq$ LABEL(T'). Since $T' \prec V_{j+1}$ and T doesn't, T must precede some task R with LABEL(R) > LABEL(V_{j+1}). All that remains is to show that $R \in W$. If R is in some block then it is in W, so assume that R is an extra task. If $\sigma(R) < \sigma(X_0)$, then R is a latecomer to W (it is added no later than when block X_0 is added to W). If $\sigma(R) \geq \sigma(X_0)$ then $R \in X_0$, since $V_{j+1} \in X_0$ and LABEL(R) > LABEL(V_{j+1}). This is a contradiction since R is an extra task. Thus, $R \in W$.

From the above two observations, it follows that task T^* and every task $T \in W$ with LABEL(T) $\geq$ LABEL(T^*), precedes a chain of at least $|P| - 1$ tasks (with each task of that chain a member of W).

Now consider the first time unit B_1 of W. There are two cases:

Case 1: B_1 is a resource time unit.

If some task $T \in (B_1 \cap W)$ precedes task T^* then T precedes a chain of at least |P| tasks, each of which is in W, hence OPT $\geq |P| + 1$. Thus suppose that there is no such task T. Since there is either an idle processor or an extra task in B_1 (which must have a lower label than T^*), when T^* was scheduled there was still room in B_1 for it. Since T^* couldn't have been prevented from executing there due to resource constraints (T^* requires no resources), there must exist a task Q such that $Q \prec T^*$. Moreover, $Q \in W$ since $T^* \in W$ and T^* is in the first partial time unit of W (i.e. Q cannot be an extra task). Hence, Q precedes a chain of at least |P|

tasks, each of which is in W, hence OPT $\geq |P| + 1$.

Case 2: B_1 is a full time unit.

Let $A_1, \ldots, A_m$ be the tasks executing in B_1. If each A_i has a label exceeding LABEL(T*) then there are at least m+1 tasks in W, each preceding a chain of at least |P| tasks, each of which is in W. It follows that OPT $\geq$ |P|+1. Thus assume that for some A_i, LABEL(A_i) $<$ LABEL(T*). Then, identically to Case 1, there exists a task Q $\in$ W, such that Q $<$ T*, hence OPT $\geq |P| + 1$. □

Now we complete the proof of the upper bound for $s \leq m - 2$. Note that it follows from previous arguments, that there are at least $m|F| + |H| + 2|P| - 1$ tasks in W. Again there are two cases to consider based on time unit B_1 of the Coffman-Graham schedule:

Case 1: B_1 is a full or resource time unit.

First note that OPT $\geq |H|/s$, m OPT $\geq m|F| + |H| + 2|P| - 1$ and that OPT $\geq |P| + 1$ (from Lemma 5.7). Moreover, CG = $|F| + |H| + |P|$, so

$$\begin{aligned} m\,CG &= [m|F| + |H| + 2|P| - 1] + [(m-2)(|P| + 1)] + [(m-1)|H|] - m + 3 \\ &\leq m\,OPT + (m-2)\,OPT + (m-1)s\,OPT - (m-3) \\ &= [2m - 2 + s(m-1)]\,OPT - (m-3) \\ &\leq [2m - 2 + s(m-1)]\,OPT, \text{ since } m \geq 3. \end{aligned}$$

$\therefore$ CG/OPT $\leq (2 - 2/m) + s(1 - 1/m)$.

Case 2: B_1 is a partial time unit.

Since B_1 is the first time unit of the schedule, there are no latecomers in B_1. Moreover, because it is a partial time unit, there must either be an extra task or an idle processor in B_1, hence $|B_1 \cap W| \leq m - 1$. Since none of the tasks in $B_1 \cap W$ requires a resource, it follows that each task in $X_q - B_1$ has a predecessor in $B_1 \cap W$. From Lemma 5.3 and the manner in which latecomers are added to W, it follows that each task in $W - X_q$ has a predecessor in X_q. Then by transitivity, each task in

$W - B_1$ has a predecessor in $B_1 \cap W$. Now consider an optimal schedule for W. Such a schedule must have an idle processor in its first time unit, since the only tasks that can execute there are those in $B_1 \cap W$. Thus, $m\ OPT \geq m|F| + |H| + 2|P|$. From the proof of Lemma 5.7, it follows that $OPT \geq |P|$. Moreover, $OPT \geq |H|/s$. Thus,

$$m\ CG = [m|F| + |H| + 2|P|] + [(m-1)|H|] + [(m-2)|P|]$$

$$\leq m\ OPT + (m-1)s\ OPT + (m-2)\ OPT$$

$$= [2m - 2 + s(m-1)]\ OPT$$

$$\therefore\ CG/OPT \leq (2 - 2/m) + s(1 - 1/m)$$ □

This completes the proof of the upper bound. □

5.1.2 The lower bounds

In this section we prove that the upper bounds given in Theorems 5.1 and 5.2 are the best possible bounds. We concentrate on proving that the bound given in Theorem 5.2 - the processor constraint case - is the best possible result. At the end of the section we indicate how to modify that proof to show that the upper bound given in Theorem 5.1 - the no processor constraint case - is the best possible result.

Lemma 5.8: If $m \geq 2$ (a processor constraint), the upper bound given in Theorem 5.2 is the best possible result.

The task systems we will use to prove this lower bound will consist of various combinations of the following two sets of tasks (Figure 5.3):

Definition: An RES_z-structure consists of:

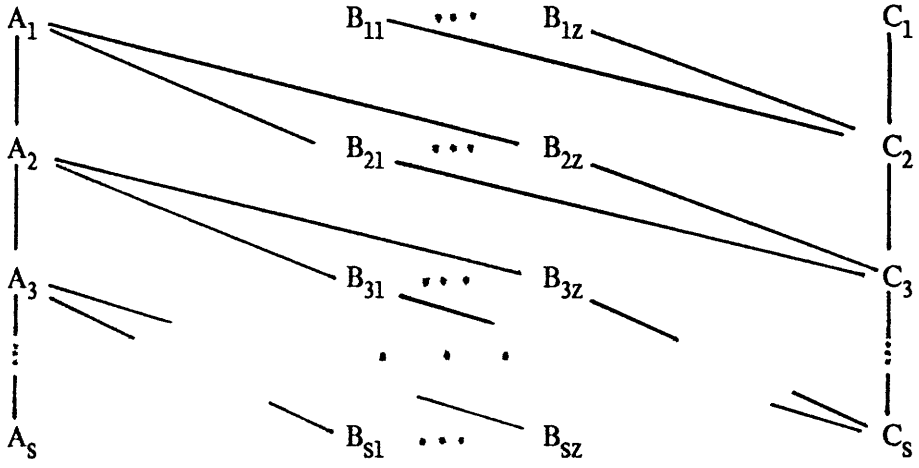
1. The following tasks:

   A_v for $1 \leq v \leq s$, where A_v requires only resource v

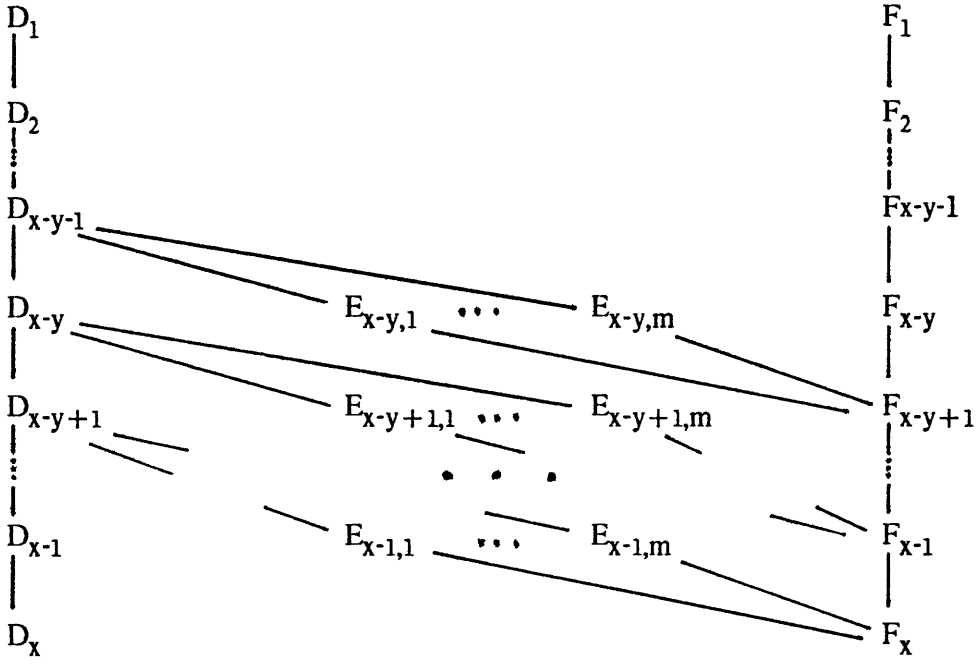
   B_{vj} for $1 \leq v \leq s$, $1 \leq j \leq z$, where B_{vj} requires only resource v

   C_v for $1 \leq v \leq s$, where C_v requires only resource v

Figure 5.3: Two useful structures



a) An RES_z-structure.



b) A $PREC_{x,y}$-structure.

2. The following precedence constraints:

$A_v \prec A_{v+1}$ and $C_v \prec C_{v+1}$ for $1 \leq v \leq s\text{-}1$

$A_v \prec B_{v+1,j}$ for $1 \leq v \leq s\text{-}1$ and $1 \leq j \leq z$

$B_{vj} \prec C_{v+1}$ for $1 \leq v \leq s\text{-}1$ and $1 \leq j \leq z$

Definition: A $PREC_{x,y}$-structure consists of:

1. The following tasks:

D_j for $1 \leq j \leq x$, where D_j requires no resources

E_{jk} for $x\text{-}y \leq j \leq x\text{-}1$, $1 \leq k \leq m$, where E_{jk} requires no resources

F_j for $1 \leq j \leq x$, where F_j requires no resources

2. The following precedence constraints:

$D_j \prec D_{j+1}$ and $F_j \prec F_{j+1}$ for $1 \leq j \leq x\text{-}1$

$D_j \prec E_{j+1,k}$ for $x\text{-}y\text{-}1 \leq j \leq x\text{-}2$ and $1 \leq k \leq m$

$E_{jk} \prec F_{j+1}$ for $x\text{-}y \leq j \leq x\text{-}1$ and $1 \leq k \leq m$

For $1 \leq v \leq s$, we will refer to tasks $B_{v1}, \ldots, B_{vz}$ as B_v-tasks, and for $x\text{-}y \leq j \leq x\text{-}1$ we will refer to tasks $E_{j1}, \ldots E_{jm}$ as E_j-tasks.

These two structures can be combined by the use of the following precedence relations:

1. $RES_z \prec PREC_{x,y}$ means that $A_s \prec D_1$

$B_{sk} \prec F_1$ for $1 \leq k \leq z$

$C_s \prec F_1$

2. $PREC_{x,y} \prec RES_z$ means that $D_x \prec A_1$

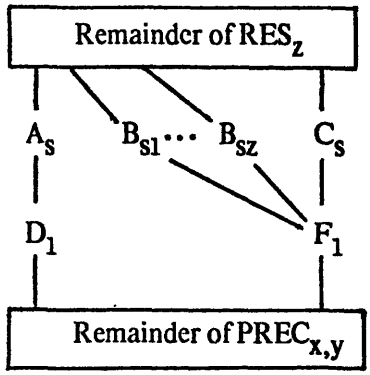
$D_x \prec B_{1k}$ for $1 \leq k \leq z$

$F_x \prec C_1$

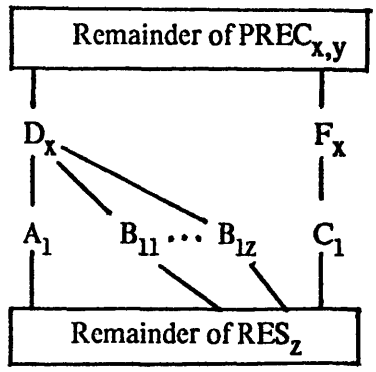
These precedence relations are shown in Figure 5.4.

Now consider possible Coffman-Graham labelings of these structures:

Figure 5.4: Precedence relations between the structures

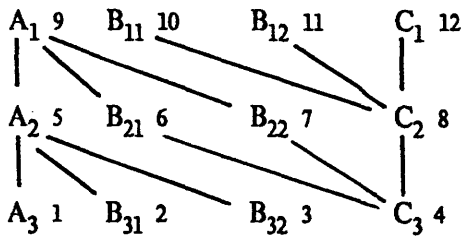


a) $RES_z \prec PREC_{x,y}$

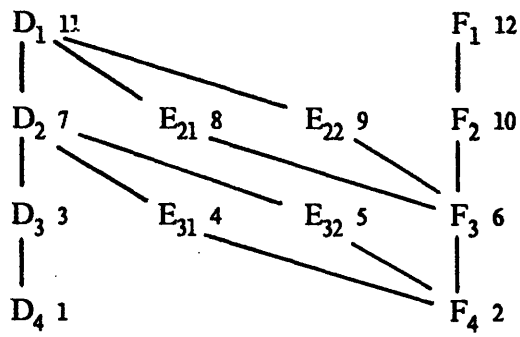


b) $PREC_{x,y} \prec REX_z$

Figure 5.5: Bad CG labelings



a) A bad CG labeling of RES_2 when s = 3.



b) A bad CG labeling of $PREC_{4,2}$ when m = 2.

Labels are given beside the tasks.

Definition:

1. A Coffman-Graham labeling of a RES_z-structure is a bad CG labeling if:

$label(B_{vk}) > label(A_v)$ for $1 \leq v \leq s$ and $1 \leq k \leq z$

$label(C_v) > label(A_v)$ for $1 \leq v \leq s$

$label(C_v) > label(B_{vk})$ for $1 \leq v \leq s$ and $1 \leq k \leq z$

2. A Coffman-Graham labeling of a $PREC_{x,y}$-structure is a bad CG labeling if:

$label(E_{jk}) > label(D_j)$ for $x-y \leq j \leq x-1$ and $1 \leq k \leq m$

$label(F_j) > label(D_j)$ for $1 \leq j \leq x$

$label(F_j) > label(E_{jk})$ for $x-y \leq j \leq x-1$ and $1 \leq k \leq m$

Figure 5.5 shows examples of bad CG labelings.

Proof of Lemma 5.8

Assume that $s \geq 1$ and $m \geq 2$ are given. Let q, x, y and z be integers to be specified later. Consider a task system S^* consisting of $q+1$ RES_z-structures: RES_z^1, ... , RES_z^{q+1}, and q $PREC_{x,y}$-structures: $PREC_{x,y}^1$, ... , $PREC_{x,y}^q$. Intuitively, we arrange these structures in a stack, alternating RES_z-structures and $PREC_{x,y}$-structures, with a RES_z-structure on the top and on the bottom of the stack (Figure 5.6). Formally, $RES_z^i \prec PREC_{x,y}^i$ for $1 \leq i \leq q$ and $PREC_{x,y}^i \prec RES_z^{i+1}$ for $1 \leq i \leq q$. Now consider a Coffman-Graham labeling of S^* in which each RES_z-structure and each $PREC_{x,y}$-structure has a bad CG labeling. To see that such a labeling exists, consider the point in the labeling process when labels have been assigned to the tasks in RES_z^{i+1}. Assume that this is a bad CG labeling. Now, $PREC_{x,y}^i$ can have a bad CG labeling only if the labeling algorithm assigns a smaller label to D_x^i than it does to F_x^i. But, this is precisely what the labeling algorithm does since RES_z^{i+1} has a bad CG labeling, hence $label(C_1^{i+1}) > label(A_1^{i+1})$ and $label(C_1^{i+1}) > label(B_{1k}^{i+1})$ for $1 \leq k \leq z$. A similar observation can be made about a bad CG labeling of RES_z^i, given that $PREC_{x,y}^i$ has already been assigned a bad CG labeling. Thus, a Coffman-Graham labeling of S^* in

Figure 5.6: The task system S*

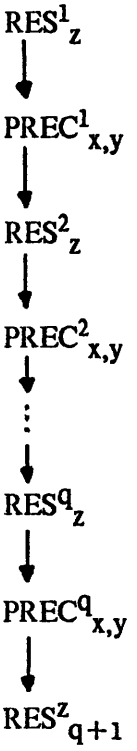
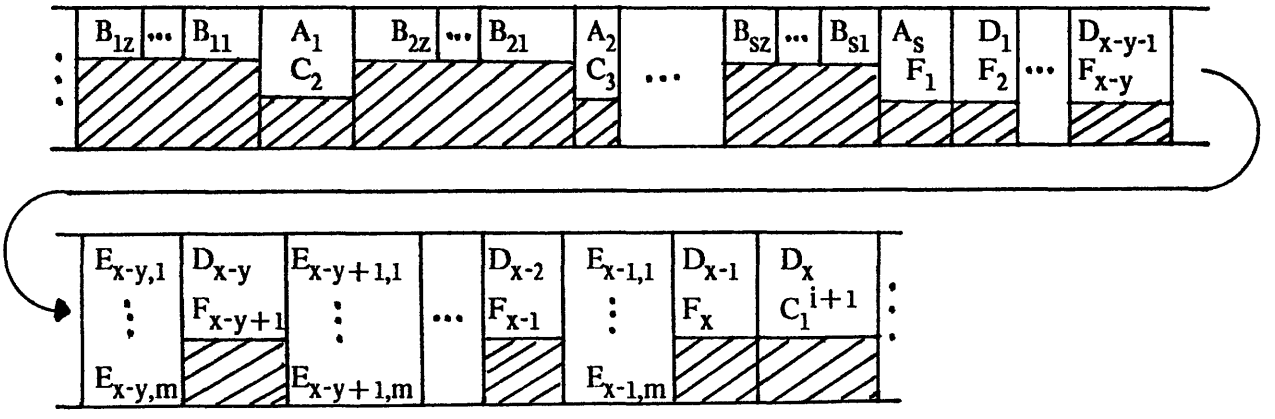


Figure 5.7: The Coffman-Graham schedule - execution of RES^i_z and PREC$^i_{x,y}$ after C^i_1 has executed. The superscript i is omitted from the tasks.



which each of the structures has a bad CG labeling does exist.

The initial portion of the list (used to schedule S^*) which is formed as a result of this labeling is: (C_1^1, B_1^1-tasks, A_1^1, C_2^1, B_2^1-tasks, A_2^1, ... , C_s^1, B_s^1-tasks, A_s^1, F_1^1, D_1^1, F_2^1, D_2^1, ... , F_{x-y-1}^1, D_{x-y-1}^1, F_{x-y}^1, E_{x-y}^1-tasks, D_{x-y}^1, ... , F_{x-1}^1, E_{x-1}^1-tasks, D_{x-1}^1, F_x^1, D_x^1, C_1^2, ...). Beginning with C_1^2 the pattern repeats for RES_z^2 and $PREC_{x,y}^2$, then for RES_z^3 and $PREC_{x,y}^3$, and so on.

The Coffman-Graham schedule produced from this list is as follows: Execute task C_1^1 in the first time unit, followed by the remainder of RES_z^1 and task F_1^1 in the next $(z+1)s$ time units (each B_{vk}^1 executes alone, since label(B_{vk}^1) > label(A_v^1) and A_v^1 precedes all of the tasks that might execute with B_{vk}^1). In the next $x+y$ time units execute the remainder of $PREC_{x,y}^1$ and task C_1^2. This consists of x time units in which two tasks execute per time unit and y time units in which m of the E^1-tasks execute per time unit. In the next $(z+1)s$ time units execute the remainder of RES_z^2 and task F_1^2. And so on. The pattern repeats (Figure 5.7) until RES_z^{q+1} executes in the final $(z+1)s$ time units. This Coffman-Graham schedule has length

$$CG = 1 + ((z+1)s + x + y)q + (z+1)s. \qquad (I)$$

Now we want to get an upper bound on the length of an optimal schedule for this system. There are three cases to consider based on the three parts of the lower bound given in the statement of Theorem 5.2.

Case 1: $s \geq m$

Without loss of generality assume $s = m$. Let z be an arbitrary integer and let $x = y = q = 0$. The task system S^* consists just of RES_z^1. From (I), $CG = 1 + (z+1)s = sz + s + 1$.

Consider the following schedule for this task system: In the first s time units execute the A^1-tasks. In the next z time units, execute all of the B^1-tasks, with s tasks executing per time unit - one task requiring each resource. Finally, execute the C^1-tasks in the last s time units. This schedule has length $z + 2s$, hence $OPT \leq z + 2s$.

$\therefore$ CG/OPT $\geq (sz + s + 1)/(z + 2s)$

$\text{limit}_{z \to \infty}$ CG/OPT $\geq s = m$.

Cases 2 and 3: $s \leq m-1$

Consider the following condition:

Condition 1: For $2 \leq i \leq q$, if the D^{i-1}-tasks, the A^i-tasks and the tasks in RES_z^{i-1} have executed, then all of the following tasks can be executed in the next z time units: the B^i-tasks, the D^i-tasks, the E^{i-1}-tasks and the F^{i-1}-tasks.

Whether or not this condition holds depends upon the relative values of s, m, x, y and z. Also, if the condition holds nonvacuously (i.e. $q \geq 2$), then the following also hold:

1. If the A^1-tasks have been executed, then the B^1-tasks and D^1-tasks can be executed in just z time units.
2. If the D^q-tasks, the A^{q+1}-tasks and the tasks in RES_z^q have been executed, then all of the following tasks can be executed in the next z time units: the B^{q+1}-tasks, the E^q-tasks and the F^q-tasks.

Lemma 5.9: If $s = m-1$ and $x \geq 2$, with $q = x$, $z = 2x$ and $y = 0$, then Condition 1 holds.

Proof

First observe that $y = 0$ means that there are no E^{i-1}-tasks for any i-1. The B^i-tasks, D^i-tasks and F^{i-1}-tasks can be executed in just z time units as follows (Figure 5.8): In time unit k, execute tasks $B^i_{1k}, \ldots, B^i_{sk}$. Since $s = m-1$, this utilizes m-1 processors in each of the z time units. The D^i-tasks and F^{i-1}-tasks execute on the unused processor: the D^i-tasks executing in the first z/2 (=x) time units and the F^{i-1}-tasks executing in the second z/2 (=x) time units. □

Lemma 5.10: If $s \leq m-2$ and x is an integer such that $x \geq 2$ and $(x-1) = 0 \bmod m$, with $z = x$, $q = x$ and $y = (m-s-2)(x-1)/m$, then Condition 1 holds.

Figure 5.8: Execution of the tasks - Lemma 5.9.

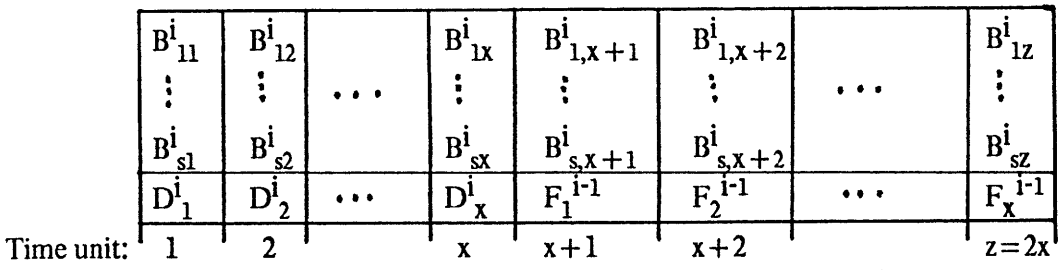


Figure 5.9: Execution of the tasks - Lemma 5.10

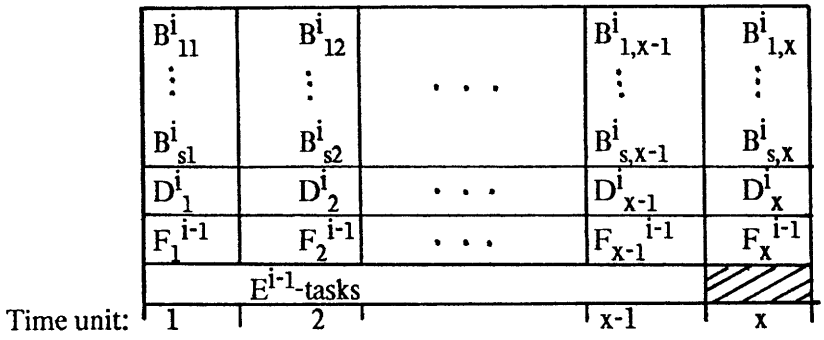


Figure 5.10: A "good" schedule.

Tasks executed	A^1-tsks	B^1-tsks D^1-tsks	...	C^{i-1}-tsks A^i-tsks	B^i-tsks D^i-tsks E^{i-1}-tsks F^{i-1}-tsks	...	C^q-tsks A^{q+1}-tsks	B^{q+1}-tsks E^q-tsks F^q-tsks	C^{q+1}-tsks
Number of time units	s	z	...	s+1	z $2 \leq i \leq q$	...	s+1	z	s

Proof

First observe that y is an integer and that there are (m-s-2)(x-1) of the E^{i-1}-tasks. The B^i-tasks, D^i-tasks, E^{i-1}-tasks and F^{i-1}-tasks, can be executed in just z time units as follows (Figure 5.9): In time unit k, execute tasks $B^i_{1k}, \ldots, B^i_{sk}$, D^i_k and F^{i-1}_k. This utilizes s+2 processors, leaving m-s-2 processors at each time unit to execute the E^{i-1}-tasks on. These tasks are executed in time units 1 thru z-1 (=x-1), with m-s-2 of the E^{i-1}-tasks executing per time unit. □

To complete the proof of the lower bound we assume that q, x, y and z are chosen such that Condition 1 holds. Consider the following schedule for the task system (Figure 5.10): In the first s time units execute the A^1-tasks. Execute the B^1-tasks and D^1-tasks in the next z time units. This is possible since Condition 1 holds. In the next s+1 time units execute the C^1-tasks and the A^2-tasks. Now execute the B^2-tasks, D^2-tasks, E^1-tasks and F^1-tasks in the next z time units. This is possible since Condition 1 holds. In the next s+1 time units execute the C^2-tasks and the A^3-tasks. Now execute the B^3-tasks, D^3-tasks, E^2-tasks and F^2-tasks in the next z time units. And so on. This pattern continues until the C^q-tasks and A^{q+1}-tasks execute. Then execute the B^{q+1}-tasks, E^q-tasks and F^q-tasks in the next z time units. Again, this is possible since Condition 1 holds. Finally, execute the C^{q+1}-tasks in the last s time units. This schedule has length $(s+z+1)q + z + 2s$. Thus, given s and m, provided q, x, z and y are specified so Condition 1 holds, we have:

$$OPT \leq (s+z+1)q + z + 2s. \qquad \text{(II)}$$

Case 2 - completion: s = m - 1

Let x be an arbitrary integer with q = x, z = 2x and y = 0. By Lemma 5.9, Condition 1 holds, and from (II), $OPT \leq (s+2x+1)x + 2x + 2s = 2x^2 + (s+3)x + 2s$. From (I), $CG = 1 + ((2x+1)s + x)x + (2x+1)s = (2s+1)x^2 + 3sx + s + 1$.

$\therefore \lim_{x \to \infty} CG/OPT = (2s + 1)/2 = s + 1/2 = (m-1) + 1/2 = m - 1/2.$

Case 3 - completion: $s \leq m - 2$

Let x be an integer such that $x > s$ and $(x-1) = 0 \bmod m$, with $z = x$, $q = x$, and $y = (m-s-2)(x-1)/m$. By Lemma 5.10, Condition 1 holds, and from (II), $OPT \leq (s+x+1)x + x + 2s = x^2 + (s+2)x + 2s$. From (I), $CG = 1 + ((x+1)s + x + (m-s-2)(x-1)/m)x + (x+1)s = ((2-2/m) + s(1-1/m))x^2 + (2s - (m-s-2)/m)x + s + 1$.

$\therefore \text{limit}_{x \to \infty} CG/OPT \geq (2 - 2/m) + s(1 - 1/m)$

This concludes the proof of Lemma 5.8, showing that the bound given in Theorem 5.2 is the best possible bound. □

Lemma 5.11: If $m \geq n$ (no processor constraint) then the upper bound given in Theorem 5.1 is the best possible upper bound.

Proof

Consider a task system S^* as described in the previous proof, with x an integer, $x \geq 2$, $z=x$, $q=x$ and $y=0$. It follows from that proof (equation I) that there exists a Coffman-Graham schedule for S^* of length

$$CG = 1 + ((x+1)s + x)x + (x+1)s = (s+1)x^2 + 2sx + s + 1.$$

From the proof of Lemma 5.10, it follows that Condition 1 holds given these values of x, z, q and y. This in turn implies that equation II given there holds, hence there exists a (optimal) schedule for S^* of length

$$OPT \leq (s+x+1)x + x + 2s = x^2 + (s+2)x + 2s.$$

$$\therefore CG/OPT \leq [(s+1)x^2 + 2sx + s + 1]/[x^2 + (s+2)x + 2s]$$

$$\text{limit}_{x \to \infty} CG/OPT = 1 + s$$ □

5.2 The implication for critical path scheduling

Now we consider the implication of the above results for critical path scheduling of UET task systems with discrete resources. Because Coffman-Graham scheduling is a subclass of critical path

scheduling and UET task systems with 0-1 resources are a subclass of UET task systems with discrete resources, we have the following two lower bound results for UET task systems with discrete resources:

Theorem 5.3: If $m \geq n$ (no processor constraint) then, in the worst case, CPATH/OPT can be arbitrarily close to $1 + s$.

Theorem 5.4: If $m \geq 2$ (a processor constraint) then, in the worst case, CPATH/OPT can be arbitrarily close to

$$\begin{cases} m & \text{if } s \geq m \\ m - 1/2 & \text{if } s = m - 1 \\ (2-2/m) + s(1-1/m) & \text{if } s \leq m - 2 \end{cases}$$

In the remainder of this section we concentrate on critical path scheduling of systems without processor constraints. Similar remarks apply for critical path scheduling of systems with processor constraints, except that they are complicated by the fact that the lower bound has three portions.

The result in Theorem 5.3 can be compared to the result of Garey, et.al. [GGJY], for critical path scheduling of UET task systems with continuous resources. That result is CPATH/OPT $\leq 1 + 17s/10$. If we let $f(s, r_1, \ldots, r_s)$ be the best possible worst case bound for critical path scheduling of UET task systems with discrete resources, we have:

$$1 + s \leq f(s, r_1, \ldots, r_s) \leq 1 + 17s/10 \qquad \text{(III)}$$

Several remarks can be made about equation III.

First, regardless of the actual values of $r_1, \ldots, r_s$, the function f is essentially a linear function in s. The values of $r_1, \ldots, r_s$ (i.e. the distribution of units of resource among the various resources) are relatively unimportant in determining the worst case bound on CPATH/OPT. This is in sharp contrast to the situation for list scheduling of UET task systems with discrete resources. In that instance, the bound was LIST/OPT $\leq 1 + r$ where $r = \Sigma_{i=1}^{s} r_i$. There, the number of different resources didn't matter at all - only the total number of units of resource of any kind in the task system.

Second, relatively little additional information about the worst case performance of critical path

scheduling for UET task systems with resources is to be gained by explicitly obtaining the function f. That is, the results on the worst case performance of critical path scheduling provided by the continuous model are going to be relatively close to those provided by the discrete model. These bounds are related by a constant - both are bounded by linear functions of s. Again this contrasts sharply with the results of Chapter 3 on list scheduling. In that chapter, we saw that the list scheduling results based on the discrete model had a much higher information content than those based on the continuous model. Here, they do not.

Chapter 6 - Overview: UET Results

6.1 Summary

In the past several chapters, we have studied list and critical path scheduling of UET task systems with resources. The formal model of task systems with resources used in most previous work involving the analysis of scheduling heuristics for these types of systems, involves continuous resources. That is, there is one unit of each resource and a task may require any portion of that one unit. We noted that there are some serious questions about the appropriateness of that model in regard to certain applications. In particular, the assumption that resources are continuous seems inappropriate for applications where the available quantities of each resource are small. To try to overcome these perceived shortcomings of the model with continuous resources, we introduced UET task systems with discrete resources. In that model, there are a specific number of units of each resource, and a task may require only integral numbers of those units. Our hope was that performance bounds based on this model with discrete resources would provide substantially more information than bounds based on the model with continuous resources. In particular, information about the affect on performance of increasing or decreasing the available units of resource in the system. Moreover, we noted that depending upon the particular application, the presence of processor constraints was or was not appropriate. Thus, we investigated the worst case performance of list and critical path scheduling for four models: those with discrete or continuous resources and with or without processor constraints. A summary of the major results now known about these problems is given in Table 6.1. Of the results given there, we note that the two results for UET task systems with continuous resources and no processor constraints are due to Garey, et.al. [GGJY], and that the rest of the results are given in this thesis.

Finally, to reiterate the remarks made in the last chapter about the relationship between the models with discrete and continuous resources, we found that our expectation that bounds based on the model

Figure 6.1: Summary of the results for UET task systems with resources

		LIST/OPT	CPATH/OPT
Continuous	No processor constraint	[GGJY] $sOPT/2+s/2+1$ "almost" best possible	[GGJY] $1+17s/10$ best possible
	Processor constraint	[GGJY] $\min\{m, (s+1)OPT/2+s/2+3/2\}$ [Yao] $\min\{m, (m-1)sOPT/(2m)+7(m-1)s/(2m)+1\}$	m if $2 \le m < s+1$ $(s+m+1)/2$ if $s+1 \le m < 2s+1$ $(4s+m+3)/4$ if $2s+1 \le m < 8s/3+1$ $(14s+m+9)/10$ if $8s/3+1 \le m < 3s+1$ $2+17s/10-(3s+1)/m$ if $3s+1 \le m$, $m \ge 10$ $2+5s/3-(8s/3+1)/m$ if $3s+1 \le m$, $m < 10$ best possible
Discrete	No processor constraint	$1+r$ best possible	$> 1+s$
	Processor constraint	$(2-1/m) + r(1-1/m)$ best possible	$\ge m$ if $s \ge m$ $\ge m-1/2$ if $s = m-1$ $> (2-2/m) + s(1-1/m)$ if $s \le m-2$

Unless otherwise noted, each of the above results is an upper bound.
Except where noted, all of these results are given in this thesis.

with discrete resources would have a much higher information content than bounds based on the model with continuous resources, was both right and wrong. For list scheduling, this was certainly the case - the results were particularly strong for the model with discrete resources and were particularly weak for the model with continuous resources. For critical path scheduling, we found that while bounds based on the model with discrete resources should have a slightly higher information content than bounds based on the model with continuous resources, the additional useful information is not nearly as great as for list scheduling. For this reason, obtaining tight bounds for critical path scheduling of UET task systems with discrete resources does not appear to be a particularly important problem.

6.2 Open Problems

There are obviously a large number of questions which remain unanswered as a result of this research. We mention only a few of the problems which we feel are the most important here.

First, is to analyze the worst case performance of other scheduling algorithms with respect to the task system model with discrete resources. In particular, the performance of the resource decreasing algorithm. This is a list scheduling algorithm in which the tasks are ordered in the list according to their R_{max}-values -- tasks with the largest R_{max}-values coming first in the list. This algorithm has been analyzed by Garey, et.al. [GGJY] for UET task systems with continuous resources and no processor constraints. For that model they show that RDEC/OPT $\leq$ 1 + 17s/10, and that task systems and resource decreasing schedules for those systems exist, such that RDEC/OPT $\geq$ 1 + 1.69s (where RDEC/OPT is the worst case ratio of the length of a resource decreasing schedule for a task system to the length of an optimal schedule for that task system). Note that this is the same upper bound as that for CPATH/OPT. An interesting question which might be answered via the model with discrete resources, is whether or not resource decreasing schedules and critical path schedules are as comparable as they appear based on the worst case performance bounds for UET task systems with continuous resources and no processor constraints.

Second, is to find algorithms which have a worst case performance bound substantially better than O(s). Consider, for instance, the scheduling of UET task systems with 0-1 resources and no processor constraints. All of the scheduling alogrithms that we have examined - list, critcal path, Coffman-Graham - as well as the resource decreasing algorithm (and simple variations of it), have a worst case performance bound of 1 + s when applied to these systems. An algorithm which had any kind of sublinear (in s) worst case performance would be a significant advance. Presumably, such an algorithm for UET task systems with 0-1 resources could be extended to provide a sublinear algorithm for more general UET task systems with resources - either continuous or discrete.

Third, is the analysis of scheduling algorithms with respect to the model with discrete resources in other contexts. For instance, in a model with no precedence constraints, but where task execution times are not restricted. In Chapter 7 we give two results on the worst case performance of list scheduling for that particular model.

Chapter 7 - Non-UET results

In this chapter we investigate list scheduling of task systems with resources where no precedence constraints exist and where task execution times are not restricted. As noted previously, this submodel is one of the two major submodels used to investigate scheduling algorithms. Also as mentioned earlier, we note that there is not always a list schedule of optimal length for such task systems. Despite that, because list schedules are intuitively simple and are easy to construct, they provide the basis for most scheduling algorithms for task systems of the type we study here. In this chapter we deal exclusively with list scheduling. For comparison purposes, we note that Graham [G66] has shown that if $m \geq 2$ (a processor constraint), then LIST/OPT $\leq 2 - 1/m$, and that this is the best possible result. We also note that if $m \geq n$ (no processor constraint), then LIST/OPT = 1.

7.1 Continuous resources

The only two significant results for list scheduling of task systems with continuous resources and no precedence constraints, are by Garey and Graham. They show [GG73, GG75] that if $m \geq n$ (no processor constraint), then LIST/OPT $\leq 1 + s$ and, [GG75], if $m \geq 2$ (a processor constraint), then LIST/OPT $\leq \min\{(m+1)/2, s+2-(2s+1)/m\}$. Moreover, they show that both of these bounds are the best possible.

7.2 Discrete resources

There are no previous results about the scheduling of task systems with discrete resources and no precedence constraints. In this section we prove the following two results about such systems:

Theorem 7.1: If $m \geq n$ and $s=1$, then LIST/OPT $\leq 2 - 1/r_1$. Moreover, this result is the best possible.

Theorem 7.2: If $m \geq n$, $s=2$, and $r_2=1$, then LIST/OPT $\leq 2 - 1/r_1$. Moreover, this result is the best possible.

7.2.1 Discussion

There are three things to be noted about these results.

First, and most obvious, is that given a system with a single type of resource, the addition of a single unit of a second type of resource has no affect on the worst case performance of list scheduling. This is somewhat surprising, and the question arises whether this is a general phenomenon. That is, can single units of a third resource, a fourth resource, and so on, be added to the system without affecting the worst case performance of list scheduling? Not surprisingly, the answer is no. Figure 7.1 shows an example of a system where the addition of a single unit of a third type of resource results in a worst case bound exceeding $2 - 1/r_1$.

Second, it is interesting to note that for the special case of $r_1 = r_2 = 1$, list schedules are optimal. As the example in Figure 7.1 shows, this phenomenon does not generalize.

Third, we can compare these results to those for task systems with continuous resources. For systems with $s = 1$, the results for continuous resources indicate that LIST/OPT ≤ 2. Our results show that LIST/OPT $\leq 2 - 1/r_1$. Obviously, for systems with a small number of units of resource, our result provides a somewhat better indication of the worst case performance of list scheduling. For systems with $s = 2$, our results show how significant the difference can be between the discrete and continuous bounds when small quantities of resources are involved. For example, if $r_1 = 2$ and $r_2 = 1$, our bound shows that LIST/OPT $\leq 3/2$. The bound based on systems with continuous resources is LIST/OPT ≤ 3. Moreover, if $r_1 = r_2 = 1$, then our bound indicates that list scheduling is optimal. Again the bound based on systems with continuous resources is LIST/OPT ≤ 3.

7.2.2 Upper bounds

In this section we prove the two upper bounds associated with Theorems 7.1 and 7.2. In the next section we show that those two bounds are the best possible upper bounds.

Note that we can prove both of the upper bounds, merely by proving the upper bound for the case

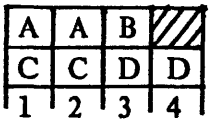
Figure 7.1: An observation

Consider a task sytem with 4 tasks and 3 resources:

Task	Execution Time	Resource Requirements
A	2	[0 1 0]
B	1	[1 0 0]
C	2	[1 0 1]
D	2	[0 1 1]

Where $r_1 = r_2 = r_3 = 1$

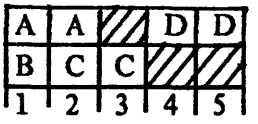
An optimal schedule:



A list schedule:

List: (A C B D)

Schedule:



LIST/OPT = $5/4 > 1 = 1 - 1/r_1$

of $s = 2$ and $r_2 = 1$ (Theorem 7.2). From such a proof it follows immediately that the same bound holds for $s = 1$ (Theorem 7.1). Similarly, if we show that the upper bound is achievable for the case of $s = 1$ (Theorem 7.1), then the bound is achievable for the case of $s = 2$ and $r_2 = 1$ (Theorem 7.2). Before proving these results, we have the following mathematical fact:

Claim 7.1: If $X \leq D$, and $B \geq AC$, with A, B, C, D, X all non-negative, then

$$(X + A)/(CX + B) \leq (D + A)/(CD + B)$$

Proof

Assume $X \leq D$ and $B \geq AC$. Then $B - AC \geq 0$, so

$$(B - AC)X \leq (B - AC)D$$

$$\Rightarrow BX + ACD \leq BD + ACX$$

$$\Rightarrow CDX + BX + ACD + AB \leq CDX + BD + ACX + AB$$

$$\Rightarrow (CD + B)(X + A) \leq (CX + B)(D + A)$$

$$\Rightarrow (X + A)/(CX + B) \leq (D + A)/(CD + B)$$ □

Lemma 7.1: If $m \geq n$, $s=2$ and $r_2=1$, then $LIST/OPT \leq 2 - 1/r_1$.

Proof

Consider any task system with two discrete resources, where $r_1 \geq 1$ and $r_2=1$. Let LIST be any list schedule for that system. Similarly to an earlier proof, for each time unit B of LIST, we let $R_i(B) = \Sigma R_i(T)$ summed over all $T \in B$, and $R_i(LIST) = \Sigma R_i(B)$ summed over all time units B in LIST. There are several cases to consider based on the resource usage in various time units of LIST.

Case 1: In each time unit B of LIST, $R_2(B) = 1$.

Since $r_2 = 1$, this means that LIST = OPT, hence $LIST/OPT = 1 < 2 - 1/r_1$.

Case 2: In each time unit B of LIST, $R_1(B) > r_1/2$.

Since $R_1(B) > r_1/2$, we have $R_1(B) \geq (r_1 + 1)/2$. Then $R_1(LIST) \geq (r_1+1)LIST/2$. But, $OPT \geq R_1(LIST)/r_1$. It follows that $OPT \geq [(r_1+1)LIST/2]/r_1$.

$\therefore$ LIST/OPT $\leq 2r_1/(r_1+1) = 2 - 2/(r_1+1) \leq 2 - 1/r_1$.

Case 3: In some time unit B of LIST, $R_1(B) \leq r_1/2$ and $R_2(B) = 0$.

Let $F = \{B \in \text{LIST}: R_1(B) \leq r_1/2 \text{ and } R_2(B) = 0\}$ and let $B^* \in F$, be a time unit such that $R_1(B^*) = \min\{R_1(B): B \in F\}$. Let $s = \max\{\sigma(T): T \in B^*\}$ and let $f = \max\{\sigma(T) + \tau_T - 1: T \in B^*\}$. That is, s is the latest starting time of any task in B^* and f is the latest finishing time of any task in B^*. Note that at least one task in B^* has an execution time at least as large as $f - s + 1$ (in particular, each task which finishs at time unit f).

Now consider any time unit B_i, $1 \leq i < s$. There is at least one task T^* in B^* which did not execute in B_i (in particular, a task starting at time unit s). Task T^* must have been prevented from executing in B_i by the resource constraints. In particular, since $R_2(T^*) = 0$, it was prevented from doing so by the constraint imposed by resource 1. Thus, $R_1(B_i) + R_1(T^*) > r_1$, hence, $R_1(B_i) + R_1(B^*) > r_1$.

Similarly, consider any time unit B_i, $f < i \leq \text{LIST}$ and any task $T \in B_i$. Task T did not execute in time unit B^* due to the constraint imposed by resource 1. Thus, $R_1(T) + R_1(B^*) > r_1$, hence, $R_1(B_i) + R_1(B^*) > r_1$.

Finally, let $d = R_1(B^*)$

$$e = \min\{R_1(B_i): 1 \leq i < s \text{ or } f < i \leq \text{LIST}\}$$

$$x = f - s + 1$$

$$y = \text{LIST} - x$$

As noted earlier, at least one task executes for at least x time units. For each of the x time units, B_i, $s \leq i \leq f$, $R_1(B_i) \geq R_1(B^*)$. Also, $y = (s - 1) + (\text{LIST} - f)$ and $\text{LIST} = x + y$. Moreover, from the arguments given above $e \geq r_1 - d + 1$. The situation is shown in Figure 7.2a.

$\therefore$ OPT $\geq \max\{x, [dx + cy]/r_1\}$

$$\geq \max\{x, [dx + (r_1 - d + 1)y]/r_1\}.$$

Figure 7.2: Resource usages in a list schedule

Schematic:

y time units	x time units
$R_1(B) \geq r_1 - d + 1$	$R_1(B) \geq d$ $(\exists T)[\tau_T \geq x]$

a) The situation in case 3.

Schematic:

y time units	x time units
$R_1(B) \geq r_1 - d + 1$	$R_1(B) \geq d$ $R_2(B) = 1$

b) The situation in case 4.

Intuitively, OPT is at least as long as the time it takes to execute any task (and some task has an execution time of at least x), and is at least as long as a schedule in which resource 1 is fully utilized at each time unit. There are two subcases to consider:

Subcase 1: $x \geq [dx + (r_1 - d + 1)y]/r_1$

It follows that $x \geq (r_1 - d + 1)y/(r_1 - d)$ and that $LIST/OPT \leq (x + y)/x = 1 + y/x$. If $d = 0$, then $y = 0$, hence $LIST/OPT = 1$, so assume that $d > 0$. Then, substituting for x,

$$LIST/OPT \leq 1 + (r_1 - d)/(r_1 - d + 1)$$
$$= 2 - 1/(r_1 - d + 1)$$
$$\leq 2 - 1/r_1 \text{ since } d > 0.$$

Subcase 2: $x < [dx + (r_1 - d + 1)y]/r_1$

It follows that $x < (r_1 - d + 1)y/(r_1 - d)$ and that $LIST/OPT \leq (x + y)/[dx/r_1 + (r_1 - d + 1)y/r_1]$. Moreover, since $d \leq r_1/2$, it follows that $(r_1 - d + 1)/r_1 > d/r_1$.

Using Claim 7.1, with $A = y$, $C = d/r_1$, $B = (r_1 - d + 1)y/r_1$, and $D = (r_1 - d + 1)y/(r_1 - d)$ we have

$$LIST/OPT \leq [(r_1 - d + 1)y/(r_1 - d) + y]/[(d/r_1)(r_1 - d + 1)y/(r_1 - d) + (r_1 - d + 1)y/r_1]$$
$$= 2 - 1/(r_1 - d + 1)$$
$$\leq 2 - 1/r_1 \text{ since } d > 0.$$

Case 4: In each time unit B of LIST, either $R_1(B) > r_1/2$ or $R_2(B) = 1$.

Let $F = \{B \in LIST: R_2(B) = 1\}$. Also, let $B^* \in F$, be a time unit such that $R_1(B^*) = \min\{R_1(B): B \in F\}$. Note that $R_1(B^*) \leq r_1/2$, since otherwise every $B \in LIST$ has $R_1(B) > r_1/2$. This was handled in case 2.

Now consider any time unit B_i preceding B^* in LIST such that $R_2(B_i) = 0$. Since $R_2(B^*) = 1$, there is at least one task T^* in B^* which does not execute in B_i. The reason that it does not execute in B_i is because of the constraint imposed by resource 1. Thus, $R_1(B_i) + R_1(T^*) > r_1$,

hence $R_1(B_i) + R_1(B^*) > r_1$.

Similarly, consider any time unit B_i following B^* such that $R_2(B_i) = 0$. There must be a task T in B_i which does not execute in B^*. This follows because $R_1(B^*) \leq r_1/2$ and $r_1/2 < R_1(B_i)$. The constraint imposed by resource 1 is the reason that T does not execute in B^*. Thus, $R_1(T) + R_1(B^*) > r_1$, so $R_1(B_i) + R_1(B^*) > r_1$.

Finally, let $d = R_1(B^*)$

$$e = \min \{R_1(B) : R_2(B) = 0\}$$

$$x = |\{B \in \text{LIST}: R_2(B) = 1\}|$$

$$y = \text{LIST} - x$$

Note that $y = |\{B: R_2(B) = 0\}|$ and that $\text{LIST} = x + y$. Moreover, by the argument given above, $e \geq r_1 - d + 1$. The situation is shown in Figure 7.2b.

$$\therefore \text{OPT} \geq \max\{x, \lceil dx + ey \rceil / r_1\}$$

$$\geq \max\{x, \lceil dx + (r_1 - d + 1)y \rceil / r_1\}$$

As in Case 3, it follows that $\text{LIST/OPT} \leq 2 - 1/r_1$. □

7.2.3 Lower bounds

In this section we show:

Lemma 7.2: If $m \geq n$ and $s = 1$, then the bound $\text{LIST/OPT} \leq 2 - 1/r_1$, is the best possible bound.

Proof

Consider a task system consisting of the following tasks:

1. A, with $\tau_A = r_1$ and $R_1(A) = 1$.
2. B_i for $1 \leq i \leq r_1(r_1 - 1)$, with $\tau_{B_i} = 1$ and $R_1(B_i) = 1$.

There are, of course, no precedence constraints. The system is shown in Figure 7.3a. Consider a schedule for this system generated from the list: $(B_1, B_2, \ldots, B_{r_1(r_1 - 1)}, A)$. Such a schedule (Figure 7.3b) consists of $r_1 - 1$ time units with r_1 B-tasks executing in each time unit, followed by the

Figure 7.3: The bound is achievable

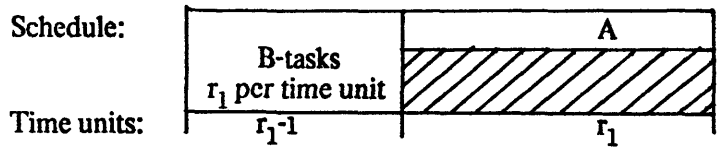
A: $\tau_A = r_1$ $\quad B_1 \cdots B_{r_1(r_1-1)}$: $\tau_{B_i} = 1$

$R_1(A) = 1$ $\quad R_1(B_i) = 1$

a) The task system with $s = 1$ and r_1 units of that resource.

A list schedule:

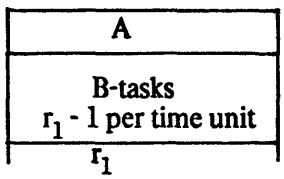
List: $(B_1\ B_2 \cdots B_{r_1(r_1-1)}\ A)$



length $= 2r_1 - 1$

b) List schedule

An optimal schedule:



length $= r_1$

c) An optimal schedule

execution of task A. This requires an additional r_1 time units. Thus, LIST = $(r_1 - 1) + r_1 = 2r_1 - 1$. Now consider a schedule for this task system generated from the list: $(A, B_1, B_2, \ldots, B_{r_1(r_1 - 1)})$. Such a schedule (Figure 7.3c) consists of r_1 time units. In each time unit, task A is executing on the first processor, and $r_1 - 1$ B-tasks are executing on the other processors. Thus, OPT = r_1.

$\therefore$ LIST/OPT = $(2r_1 - 1)/r_1 = 2 - 1/r_1$. □

Chapter 8 - Concurrent Task Systems

In this chapter we investigate an extension of the basic task system model that was discussed in Chapter 1. This extension allows tasks to require more than one processor at each step of their execution.

8.1 The model

A task system with concurrency is a system $S = \langle T, \prec, m, C \rangle$ where:

1. $T = \{T_1, \ldots, T_n\}$ is a set of tasks - associated with T_i is a positive integral execution time τ_i.
2. $\prec$ is a partial order specifying precedence constraints between the tasks.
3. There are m identical processors.
4. $C \subseteq \{1, 2, \ldots, m\}$. The elements of C are degrees of concurrency.

Associated with each task T_i, is a degree of concurrency $q_i \in C$. Intuitively, task T_i must execute for τ_i time units, and requires q_i processors for each of those time units. Task T_i is said to require $\tau_i q_i$ processor units to execute. When convenient, we let q_X represent the degree of concurrency of task X.

A valid schedule for a task system with concurrency S is a mapping $\sigma: T \rightarrow (N - \{0\})$ such that:

1. For all $l \in (N - \{0\})$, $Q_l \leq m$, where $Q_l = \Sigma\, q_i$ summing over all T_i such that $\sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1$.
2. If $T_i \prec T_j$, then $\sigma(T_i) + \tau_i - 1 < \sigma(T_j)$.

As far as performance bounds are concerned, we restrict our attention to list schedules. Intuitively, for task systems with concurrency, a list schedule is one where, if m - k processors are available, the first unexecuted task on the list, all of whose predecessors have completed and whose degree of concurrency does not exceed m - k, is executed. More formally, a task T_j is ready at time l if for every T_i such that $T_i \prec T_j$, $\sigma(T_i) + \tau_i - 1 < l$. A list schedule is a valid schedule which is generated as follows:

1. Initially, L is an (ordered) list of the tasks in T and l is 1.
2. While L is nonempty perform this step

a. Let $k = \Sigma q_i$ summed over all $T_i \notin L$ such that $\sigma(T_i) \leq l \leq \sigma(T_i) + \tau_i - 1$.

b. Let L' be a list of the ready tasks on L at time l, the tasks in the same order on L' as on L.

c. While L' is nonempty and $k < m$ perform this step

i. Let T be the first task on L'.

ii. If $q_T \leq m - k$,

then let $\sigma(T) = l$, let $k = k + q_T$ and remove T from L.

iii. Remove T from L'.

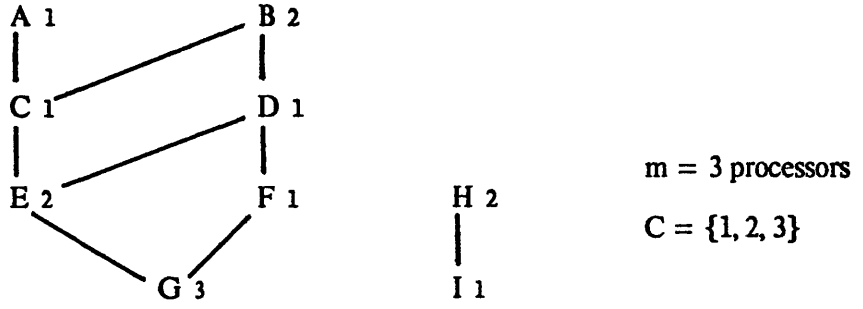
d. Let $l = 1 + \min \{\sigma(T_i) + \tau_i - 1 : T_i \notin L \text{ and } \sigma(T_i) + \tau_i - 1 \geq l\}$.

Examples of a concurrent task system and a list schedule for that system are given in Figure 8.1.

A task system with concurrency in which all tasks have the same execution time (which is assumed to be one) is a concurrent UET task system. All of our results are about concurrent UET task systems. As with the basic UET task system model, no generality is lost by restricting our attention to list schedules when dealing with concurrent UET task systems, since there is always a list schedule which is an optimal schedule.

The task systems with concurrency model arises from several sources. A situation where one processor is to monitor another processor on a particular set of jobs is an example of a task explicitly requiring more than one processor. Moreover, with the current interest in parallel processing, the development of algorithms which require several processors to be simultaneously devoted to a single task seems inevitable. Apart from computer applications, task systems with concurrency model certain practical situations more precisely than standard task systems. For example, a construction company may want to allocate its supply of men to complete some system of jobs. They know the number of men and the number of hours required to complete each job and are interested in completing the system of jobs as soon as possible. This problem is naturally modeled as a scheduling problem for a task system with concurrency.

Figure 8.1: An example of a task system with concurrency



The degree of concurrency of each task is given beside the task. Each task has an execution time of one.

A list schedule:

List: (H I G F E D C B A)

Schedule:

H	I	D	F	G
H	B	B	E	G
A	B		E	G

Time unit: 1 2 3 4 5

As is the case with several other extensions of the standard model a task system with concurrency can be viewed as a restricted type of task system with resources. That is, given a task system with concurrency S, consider a task system with one discrete resource and no processor constraint. Furthermore, suppose there are m (the number of processors in S) units of that resource available and each task requires α units of the resource where $\alpha \in C$. This restricted type of task system with discrete resources is equivalent to a task system with concurrency. In as much as this relationship exists, our results can be viewed as results for this restricted type of task system with resources. However, we feel that the approach through the resource model is an unnatural one for the problems we have described and that the task systems with concurrency approach is more instructive. We know of no results about task systems with concurrency other than those presented here.

8.2 The complexity of concurrent UET scheduling

In this section we give two NP-completeness results involving concurrent UET task systems. In subsequent sections other aspects of the problem are examined, based on the probable non-existence of polynomial time algorithms for finding optimal schedules for such systems.

8.2.1 Arbitrary concurrency, no precedence constraints

Consider the following decision problem:

CONCURRENCY: Given a deadline d', and a concurrent UET task system in which m is arbitrary, $<$ is empty (i.e. there are no precedence constraints) and $C = \{1, \ldots, m\}$, does there exist a schedule for the system with length not exceeding d'?

CONCURRENCY is stated as a decision problem, rather than as an optimization problem, so that it is easily seen to be in NP. Note that any degree of concurrency up to the number of processors is allowed.

Theorem 8.1: CONCURRENCY is NP-complete.

Proof

Garey and Johnson [GJ79] have noted that the problem of scheduling task systems with arbitrary

execution times and no precedence constraints to meet a deadline d on p processors is NP-complete. That problem reduces to CONCURRENCY by exchanging each execution time for an equal degree of concurrency and letting $d' = p$ and $m = d$. □

8.2.2 Bounded concurrency, arbitrary precedence constraints

Consider the following decision problem:

12CONCURRENCY: Given a deadline d', and a concurrent UET task system in which there are 3 processors, $<'$ is arbitrary and $C = \{1,2\}$, does there exist a schedule for the system with length not exceeding d'?

It has been shown by Ullman [U75] that the following problem is NP-complete:

NOIDLE: Given a deadline d, such that $n = dm$ and a UET task system $\langle T, <, m\rangle$ in which m and $<$ are arbitrary, and $T = \{T_1, \ldots, T_n\}$, does there exist a schedule for the system with length not exceeding d?

Intuitively, NOIDLE asks if the specified task system can be scheduled so that no idle time exists in the schedule. The remainder of this section is devoted to showing that 12CONCURRENCY is NP-complete. The reduction given here is an adaptation of a construction developed by Ullman [U76].

Theorem 8.2: 12CONCURRENCY is NP-complete.

Proof

Let a UET task system $S = \langle T, <, m\rangle$ and a deadline d, such that $n = dm$, be an instance of NOIDLE. Consider the following instance of 12CONCURRENCY:

1. Let $d' = 2md$, and let $S' = \langle T', <', 3, \{1,2\}\rangle$.
2. For each task $T_i \in T$, there are two tasks T_i and $T_i^!$ in T'. Each has an execution time of one. Let $q_i = 2$, $q_i^! = 1$, and $T_i^! <' T_i$. Moreover, if the relation $V < T_i$ exists in S, then the relation $V <' T_i^!$ is in S'. Call tasks T_i and $T_i^!$ regular tasks.
3. There are 2md tasks X_i, for $1 \leq i \leq 2md$. For each i, $1 \leq i \leq 2md - 1$, the precedence constraint

$X_i <' X_{i+1}$ is in S'. Furthermore, if $0 \leq (i - 1 \bmod 2m) \leq m - 1$ then $q_{X_i} = 2$, otherwise $q_{X_i} = 1$. Call each X_i a contour task.

Note that a schedule for S' meeting the deadline d', can have no idle time, since the schedule for S meeting deadline d is to have no idle time.

Claim: If a schedule of length d exists for S, then a schedule of length d' exists for S'.

Proof

Consider a schedule of length d for S. Consider any time unit l in that schedule, and let $T_{l_1}, \ldots, T_{l_m}$ be the tasks executed in that time unit. Then, in the schedule for S', in time unit $2m(l - 1)+i$ execute tasks $X_{2m(l - 1)+i}$ and T'_{l_i}, and in time unit $2m(l - 1)+m+i$ execute tasks $X_{2m(l - 1)+m+i}$ and T_{l_i} for $1 \leq i \leq m$. This produces a schedule for S' in which no idle time exists. All that remains is to verify that no precedence constraints are violated. Clearly none of the constraints between the contour tasks are violated and none of the constraints of the form $T' <' T$ are violated. Consider any constraint of the form $V <' T'$. This means that $V < T$ in S, so V is executed before T in the schedule for S. Then in our constructed schedule for S', V executes before both T' and T. Hence, none of the precedence constraints is violated and a valid schedule of length d' exists for S'. □

Claim: If a schedule of length d' exists for S', then a schedule of length d exists for S.

Proof

Consider a schedule of length d' for S'. Since $d' = 2dm$, contour task X_i must execute in time unit i of the schedule. The regular tasks must then execute in the processor units not being used by the contour tasks. These remaining processor units have a very particular distribution. The first m time units of the schedule each has one processor unit available for regular tasks, the second m time units each has two processor units available for regular tasks, the third m time units each has one processor unit available for regular tasks, and so on. The pattern of m time units

with one processor unit available and then m time units with two processor units available repeats itself d times. We will call the ith set of m time units band i. This pattern and the no idle time observation combine to force the "primed" regular tasks to execute only during time units when one processor unit is available, and the "unprimed" regular tasks to execute only during time units when two processor units are available. This is shown in Figure 8.2.

Therefore, the schedule for S is as follows: In time unit l of the schedule, execute the tasks corresponding to the m (unprimed) regular tasks executed in band $2l$ of the schedule for S'. This schedule clearly meets the deadline of d and since each task in T corresponds to an unprimed task in T', each task in T is executed at some time unit of the schedule. All that remains is to verify that the precedence constraints are not violated. Consider any precedence relation V $<$ T in S. The relations V $<'$ T' and T' $<'$ T are in S'. Suppose V and T were executed in the same band in the schedule for S'. Then T' would also be executed in that band. But primed regular tasks must be executed in bands with only one processor unit available per time unit. Contradiction. Thus, in the schedule for S', V is executed in some band before the band that T is executed in, hence V is executed before T in the schedule for S. Therefore, a valid schedule exists for S. □

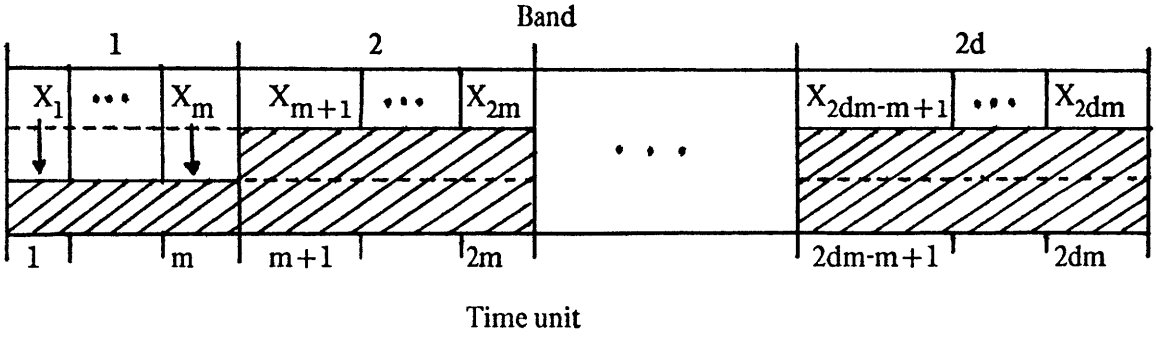
Finally, we note that 12CONCURRENCY is obviously in NP, hence it is NP-complete. □

We conclude this section by noting that by using a straight-forward modification of the contour tasks, it can be shown that 12CONCURRENCY is NP-complete for any fixed number of processors m $\geq$ 3.

8.3 Worst case bounds

In this section we show that for concurrent UET task systems, the ratio of the length of an arbitrary list schedule for the system to the length of an optimal schedule is bounded above by (2m-r)/(m-r+1), where r is the maximum degree of concurrency. As noted earlier, when r = 1 these systems become basic UET task systems. In this instance, our bound becomes 2 - 1/m, which is the corresponding bound for basic systems as given by Graham [G66]. In this section we also show that concurrent UET task

Figure 8.2: The schedule produced by the contour tasks and deadline 2dm.



Regular tasks must execute in the cross-hatched time units -- primed tasks in odd numbered bands and unprimed tasks in even numbered bands.

systems exist for which the ratio of the length of a list schedule for this system to the length of an optimal schedule is $\lfloor(2m-r)/(m-r+1)\rfloor$.

8.3.1 An upper bound

Theorem 8.3: Let $S = \langle T, \prec, m, C\rangle$ be a concurrent UET task system where r is the maximum degree of concurrency in C. Then LIST/OPT $\leq (2m-r)/(m-r+1)$.

Proof

Let OPT be the length of an optimal schedule for S and let LIST be the length of an arbitrary list schedule for S. First we give a lower bound on the length of an optimal schedule. Let h be the length of a critical path in the dag for $\prec$, and let $\alpha = \Sigma\, q_i$ such that $T_i \in T$. This is the total number of processor units required for the actual execution of tasks in T. An optimal schedule must be at least as long as the length of a critical path for the system and must be at least as long as a schedule with no idle time for a task system requiring α processor units. Thus, OPT $\geq \max(h, \alpha/m)$.

Next we give an upper bound on the length of an arbitrary list schedule. Consider any time unit l of the schedule which has more than r - 1 idle processors. Because there are at least r idle processors in that time unit, all unexecuted tasks must be successors of the tasks executing in that time unit. Let k be the highest level which has a task executing in time unit l. Since a task is only a predecessor of tasks at lower levels then the task's own level, time unit l must be the last time unit during which tasks at level k are executed. Therefore, there are at most h time units in which more than r - 1 processors are idle. At all other time units at least m-r+1 processors must be executing tasks. Hence, LIST $\leq$ $h+(\alpha-h)/(m-r+1)$.

$\therefore$ LIST/OPT $\leq [h+(\alpha-h)/(m-r+1)]/\max(h,\alpha/m)$, which, by a simple case analysis, reduces to

LIST/OPT $\leq (2m-r)/(m-r+1)$. □

8.3.2 A lower bound

The remainder of this section is devoted to showing that concurrent UET task systems exist for

which there are list schedules such that the ratio of the length of the schedule to the length of an optimal schedule asymptotically approaches $\lfloor(2m-r)/(m-r+1)\rfloor$. While this is not exactly the bound derived above, the difference is less than one.

Assume that m, the number of processors, and r, the maximum degree of concurrency are given. Let n be any positive integer. The following three sets of tasks will be used to construct the desired task systems:

An A-structure consists of: Tasks A_{ij} for $1 \le i \le m-r+1$ and $1 \le j \le n$, where $q_{A_{ij}} = 1$.

$$A_{ij} < A_{i,j+1} \text{ for } 1 \le j \le n-1 \text{ and } 1 \le i \le m-r+1,$$

A B-structure consists of: An A-structure.

$$\text{Tasks } B_i, \text{ for } 1 \le i \le \lfloor m/r \rfloor, \text{ with } q_{B_i} = r.$$

$$B_i < A_{j1} \text{ for } 1 \le i \le \lfloor m/r \rfloor \text{ and } 1 \le j \le m-r+1.$$

A C-structure consists of: Tasks C_i, for $1 \le i \le \lfloor m/r \rfloor$, with $q_{C_i} = r$.

$$\text{Tasks } D_j \text{ for } 1 \le j \le n, \text{ with } q_{D_j} = 1.$$

$$C_i < D_1 \text{ for } 1 \le i \le \lfloor m/r \rfloor.$$

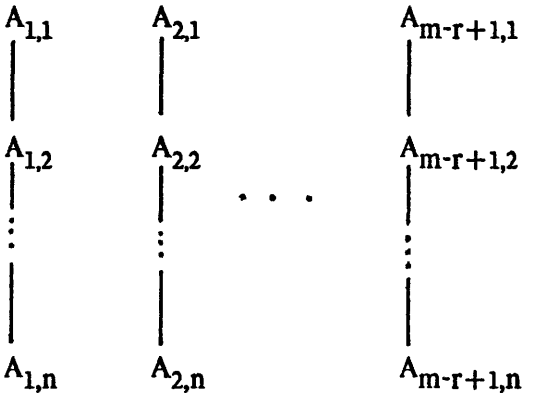
These three structures are shown in Figure 8.3.

Next we give the specifications for a task system for which a list schedule with the desired length relative to an optimal schedule exists. We let $b = \lfloor m/(m-r+1) \rfloor$. There are two cases to consider.

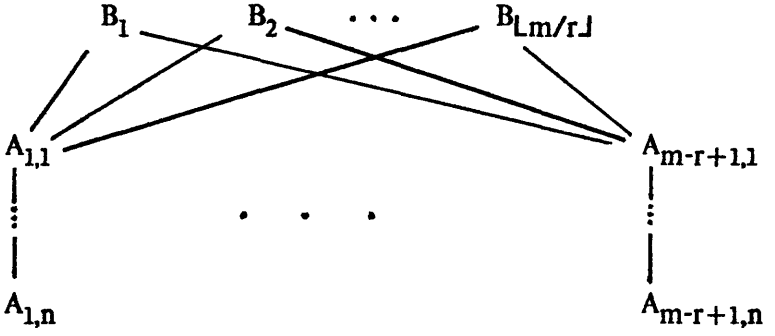
Case 1: $m/(m-r+1)$ is an integer, hence $b = m/(m-r+1)$.

Consider the following task system $S = \langle T, <, m, C \rangle$, where r is the maximum degree of concurrency in C. T and $<$ consist of the tasks and associated precedence constraints from one A-structure and b-1 B-structures. This system is shown in Figure 8.4a. The system consists of (b-1) $\lfloor m/r \rfloor$ independent tasks each with concurrency r, and $n(m-r+1)b = nm$ tasks each with concurrency 1. Note that these tasks with concurrency 1 form m independent chains of n tasks each, and that an optimal schedule requires at least n time units after the last task with concurrency r is

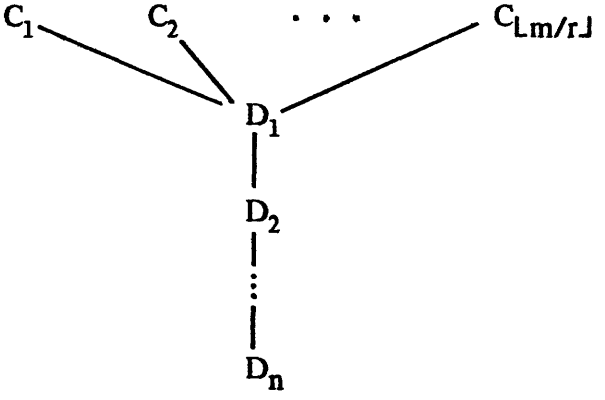
Figure 8.3: Sets of tasks used to construct task systems



a) An A-structure - all of these tasks have concurrency 1

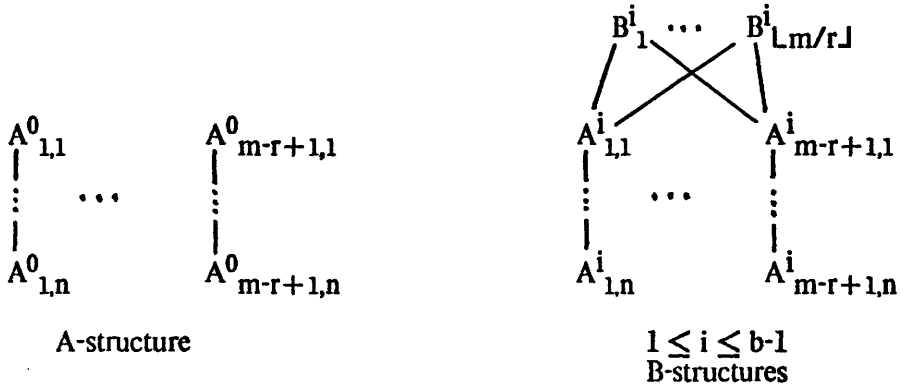


b) A B-structure - the B-tasks have concurrency r, and the A-tasks have concurrency 1

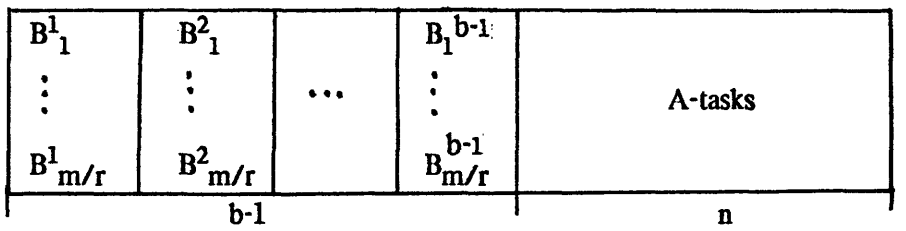


c) A C-structure - the C-tasks have concurrency r, and the D-tasks have concurrency 1

Figure 8.4: A task system and two schedules - case 1

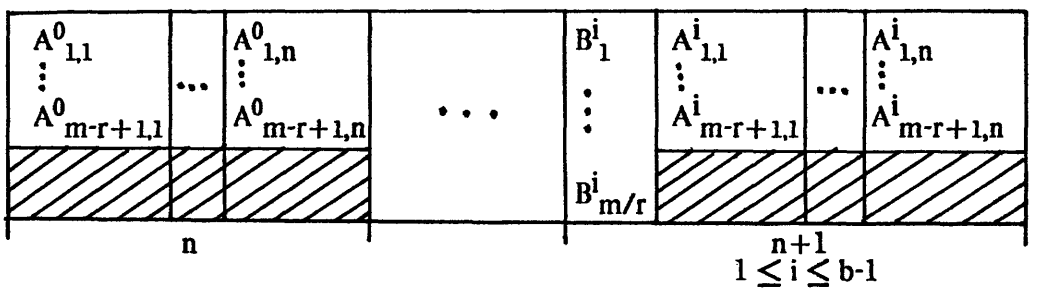


a) The task system



To simplify the figure it is assumed that m/r is an integer. The m chains, each with n tasks with concurrency 1, execute in the final n time units.

b) An optimal schedule



To simplify the figure it is assumed that m/r is an integer.

c) A "bad" schedule

executed.

The following is an optimal schedule: In the first b-1 time units execute all of the tasks with concurrency r by executing $\lfloor m/r \rfloor$ tasks with concurrency r at each time unit (and allowing any processor units not used by those tasks to be used to execute any available tasks with concurrency 1). Complete the schedule by executing the remaining tasks with concurrency 1 in the final n time units. The schedule is shown in Figure 8.4b. An optimal schedule thus has length b+n-1. Call this value OPT.

Now consider the following schedule. In the first n time units execute the tasks in the A-structure. Then execute the tasks with concurrency r from one of the B-structures, followed by the tasks in the A-structure associated with that B-structure. This requires n+1 time units. Continue by executing the other B-structures, one at a time in the same manner, until all tasks are executed. The schedule is shown in Figure 8.4c. The length of the schedule is $n+(b-1)(n+1) = bn+b-1$. Call this value LIST.

$\therefore$ LIST/OPT $= (bn+b-1)/(n+b-1)$ and $\lim_{n \to \infty}$ LIST/OPT $= b$. Furthermore,

$b = m/(m-r+1)$ which is an integer. Thus, $b = m/(m-r+1)+\lfloor (m-r)/(m-r+1) \rfloor =$

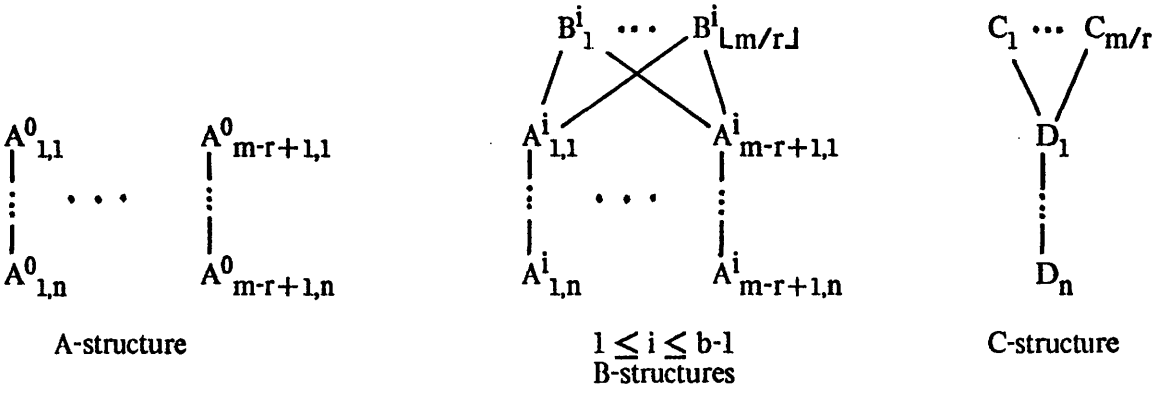
$\lfloor (m+(m-r))/(m-r+1) \rfloor = \lfloor (2m-r)/(m-r+1) \rfloor$.

$\therefore \lim_{n \to \infty}$ LIST/OPT $= \lfloor (2m-r)/(m-r+1) \rfloor$.

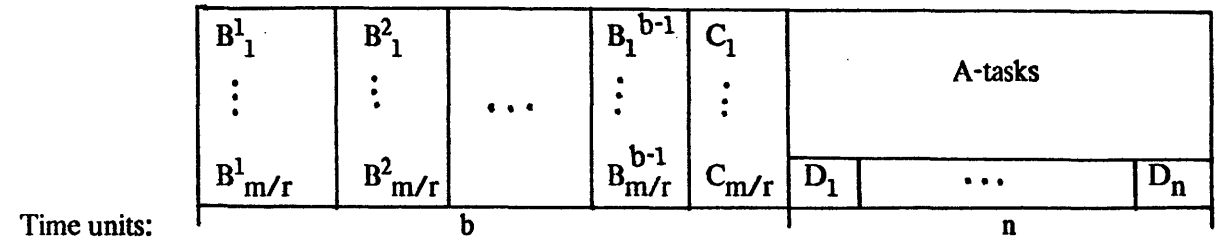
Case 2: $m/(m-r+1)$ is not an integer.

Consider the following task system $S = \langle T, \prec, m, C \rangle$, where r is the maximum degree of concurrency in C. T and $\prec$ consist of the tasks and associated constraints from one A-structure, b - 1 B-structures and one C-structure. This is shown in Figure 8.5a. Similarly to Case 1, an optimal schedule first executes the tasks with concurrency r and then completes the execution of the tasks with concurrency 1. This is shown in Figure 8.5b. An optimal schedule has length OPT $= b+n$. Also, there is a list schedule which first executes the tasks in the A-structure, then executes the tasks in each of the

Figure 8.5: A task system and two schedules - case 2

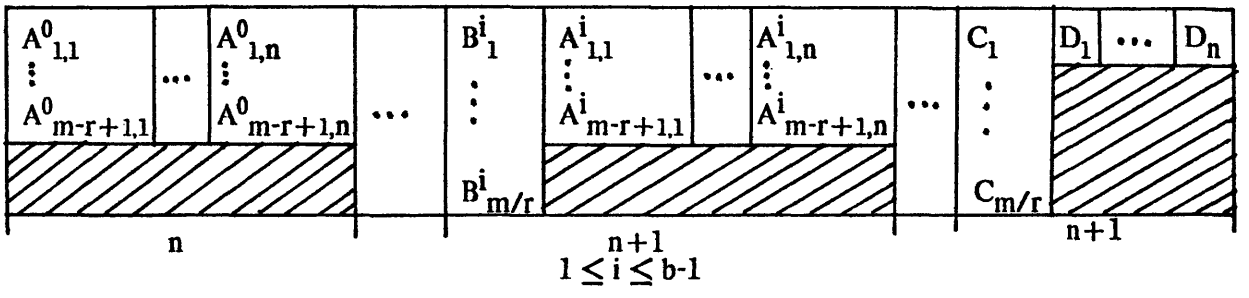


a) The task system



To simplify the figure it is assumed that m/r is an integer. The m chains, each with n tasks with concurrency 1, execute in the final n time units. Depending on the relative values of m and r, some idle time may exist in the final n time units.

b) An optimal schedule



To simplify the figure it is assumed that m/r is an integer.

c) A "bad" schedule

B-structures, and finally executes the tasks in the C-structure. This schedule is shown in Figure 8.5c. It has length LIST = $n+b(n+1) = n(b+1)+b$.

$\therefore$ LIST/OPT = $(n(b+1)+b)/(b+n)$ and $\lim_{n \to \infty}$ LIST/OPT = $b+1$. But $m/(m-r+1)$ is not an integer. Thus, $b+1 = \lfloor m/(m-r+1) \rfloor + 1 = \lfloor (m-1)/(m-r+1) \rfloor + 1 = \lfloor ((m-1)+(m-r+1))/(m-r+1) \rfloor = \lfloor (2m-r)/(m-r+1) \rfloor$.

$\therefore$ $\lim_{n \to \infty}$ LIST/OPT = $\lfloor (2m-r)/(m-r+1) \rfloor$. □

8.4 A restricted problem

We examine concurrent UET task systems in which $C = \{1,2\}$. As shown earlier, for any fixed number of processors exceeding 2, the scheduling of such systems is NP-complete. In this section we give a polynomial time algorithm which produces optimal schedules on two processors. This algorithm is a modification of the algorithm given by Coffman and Graham [CG] which produces optimal schedules for basic UET task systems on two processors.

Assume that $S = \langle T, <, m, \{1,2\} \rangle$ is a concurrent UET task system. The algorithm is as follows:

1. Add all transitive edges to the dag representing $<$.
2. Remove all tasks with concurrency two from this system along with any precedence constraints directly involving them. This yields a basic UET task system (i.e. without concurrency) $S' = \langle T', <', m \rangle$. Call this the underlying system.
3. Remove all transitive edges from the dag representing $<'$.
4. Use the Coffman-Graham algorithm to produce a list which can be used to schedule S'.
5. Append (in any order) the tasks with concurrency two to the front of the list. This new list can be used to schedule S.

Essentially, the tasks with concurrency two are removed from the original system, a schedule is found for the underlying system and then each task with concurrency two is fit into that schedule as soon as all of its predecessors have been executed.

Theorem 8.4: The algorithm given above produces optimal schedules for concurrent UET task systems (in which each task has concurrency 1 or 2) on two processors.

Proof

Suppose the schedule produced by this algorithm is not optimal. Let OPT be an optimal schedule. Because there are only two processors, if a task with concurrency two is executed at some time unit, then no other task can be executed at that time unit. This means that the tasks with concurrency two can be removed from OPT, and the schedule compressed to get a schedule for the underlying system. Two things should be noted about this schedule for the underlying system:

1. It is a valid schedule, since $V <' T$ in S' if and only if there exists a (possibly empty) sequence of tasks $P_1, \ldots, P_k$, such that $V < P_1 < \ldots < P_k < T$ in S,
2. It is necessarily shorter than the schedule produced for the underlying system in step 4 of the algorithm.

But an optimal schedule for the underlying system results from the list which was produced in step 4, hence a contradiction. □

References

[BCS] Bruno, J., Coffman, E.G. Jr., and Sethi, R., "Scheduling independent tasks to reduce mean finishing time," CACM 17(1974), 382-387.

[Ch] Chen, N.F., "An analysis of scheduling algorithms in multiprocessor computing systems", Technical Report UIUCDCS-R-75-724, Department of Computer Science, University of Illinois at Urbana-Champaign.

[C] Coffman, E.G., editor. Computer and job-shop scheduling theory. Wiley (1976), 299 pp.

[CG] Coffman, E.G. and Graham, R.L., "Optimal scheduling for two-processor systems", Acta Informatica 1(1972), 200-213.

[FKN] Fujii, M., Kasami, T. and Ninomiya, K., "Optimal sequencing of two equivalent processors," SIAM Journal of Applied Mathematics, 17(1969), 784-789; Erratum 20(1971), 141.

[GG73] Garey, M.R. and Graham, R.L., "Bounds on scheduling with limited resources," Operating Systems Review, 7(1973), 104-111.

[GG75] Garey, M.R. and Graham, R.L., "Bounds for multiprocessing scheduling with resource constraints," SIAM Journal on Computing, 4(1975), 187-200.

[GGJY] Garey, M.R., Graham, R.L., Johnson, D.S., and Yao, A.C., "Resource constrained scheduling as generalized bin packing," Journal of Combinatorial Theory, Series A. 21(1976), 257-298.

[GJ74] Garey, M.R. and Johnson, D.S., "Complexity results for multiprocessor scheduling under resource constraints." Proceedings of the 8th Annual Princeton Conference on Information Sciences and Systems (1974) 168-172.

[GJ79] Garey, M.R. and Johnson, D.S., Computers and intractibility - A guide to the theory of NP-Completeness, W.H. Freeman and Company (1979), 338pp.

[Go] Goyal, D.K., "Scheduling equal execution time tasks under unit resource restriction," CS-76-038, Computer Science Department, Washington State University.

[G66] Graham, R.L., "Bounds for certain multiprocessing anomalies," Bell System Technical Journal, 45(1966), 1563-1581.

[G69] Graham, R.L., "Bounds on multiprocessing timing anomalies," SIAM Journal of Applied Mathematics, 17(1969), 416-429.

[G72] Graham, R.L., "Bounds on multiprocessing anomalies and related packing algorithms," Proceedings AFIPS Conference, 40(1972), 205-217.

[GLLK] Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G., "Optimization and approximation in deterministic sequencing and scheduling: A survey", BW 82/77, Mathematisch Centrum, Amsterdam, The Netherlands.

[H] Hu, T.C., "Parallel sequencing and assembly line problems," Operations Research 9(1961), 841-848.

[IK] Ibarra, O.H. and Kim, C.E., "On two-processor scheduling of one- or two-unit time tasks with precedence constraints," Journal of Cybernetics, 5(1975), 87-109.

[Ja] Jaffe, J., "Parallel computation: Synchronization, scheduling and schemes," PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1979.

[JDUGG] Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R. and Graham, R.L., "Worst-case Performance Bounds for Simple One-dimensional Packing Algorithms," SIAM Journal of Computing 3(1974), 299-325.

[KS] Kafura, D.G. and Shen, V.Y., "Task scheduling on a multiprocessor system with independent memories," SIAM Journal of Computing, 6(1977), 167-187.

[LS] Lam, S. and Sethi, R., "Worst case analysis of two scheduling algorithms," SIAM Journal of Computing, 6(1977), 518-536.

[LK] Lenstra, J.K. and Rinnooy Kan, A.H.G., "Complexity of scheduling under precedence constraints," Operations Research, 26(1978), 22-35.

[Le] Leung, J. Y-T., "Bounds on list scheduling of UET tasks with restricted resource constraints," Information Processing Letters, 9(1979), 167-170.

[Li] Lui, J.W.S. and Lui, C.L., "Bounds on scheduling algorithms for heterogeneous computer systems," Technical Report UIUCDCS-R-74-632, Department of Computer Science, University of Illinois at Urbana-Champaign, 1974.

[U73] Ullman, J.D., "Polynomial complete scheduling problems," Operating Systems Review, 7(1973), 96-101.

[U75] Ullman, J.D., "NP-complete scheduling problems," Journal of Computer and Systems Sciences, 10(1975) 384-393.

[U76] Ullman, J.D., "Complexity of Sequencing Problems," in Computer and Job-Shop Scheduling Theory, E.G. Coffman, editor, Wiley (1976), 139-164.

[Y] Yao, A.C., "Scheduling unit-time tasks with limited resources," Proceedings of the Sagamore Computer Conference, Springer-Verlag, (1974) 17-36.

Biographical Note

The author was born in Baltimore, Maryland on December 20, 1953 and was raised in Reisterstown, Maryland. He attended Franklin Senior High School where he was valedictorian of his graduating class and received a National Merit Letter of Commendation. He also earned varsity letters in indoor and outdoor track while at Franklin. His other major activity in his high school years was his involvement with Boy Scouts, where he earned the rank of Eagle Scout and received the God and Country Award.

The author attended college at Penn State and majored in both Computer Science and Mathematics. While at Penn State he received several awards for academic achievement and graduated with a grade point average of 3.98 out of 4.00. He was actively involved with academic governance both on the university and departmental levels - he was the student senator from the College of Science to the University Faculty Senate, and was the chairman of the Computer Science Undergraduate Advising Committee.

The author began graduate school at MIT in September 1975. In June 1977 he received his S.M. degree in Computer Science. While at MIT, in addition to his research and classes, he taught recitation sections of courses 6.030 and 6.031. He was also active in intramural athletics - squash, softball, badminton, bowling and basketball. In addition, he enjoys skiing and golf.

While at Penn State, the author met Isabel B. Knowlton, whom he married on August 10, 1975. She will be receiving her PhD in June, 1980 from MIT in Ceramic Science. The author has accepted an assistant professorship at the University of Pittsburgh beginning September 1980 and his wife a position at the Westinghouse Research and Development Center in Pittsburgh.