ON FINDING APPROXIMATE SOLUTIONS TO SOME PROBLEMS

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

TEOFILO FRANCISCO GONZALEZ-ARCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

August, 1975

DEDICATION

To my parents

Mr. Teofilo F. Gonzalez Jr.

And

Mrs. Honoria A. de Gonzalez

To my brother and sisters

Jorge, Graciela and Hilda

ACKNOWLEDGMENTS

I wish to thank my advisor, Professor Sartaj Sahni, for guiding me during the course of this research. Most of the results were obtained by joint research carried out by the two of us. The results in Chapter IV are a combined effort with Professor Oscar H. Ibarra, for which, I want to thank him. I wish to thank Professor William R. Franta for introducing us to the problems of Chapter III.

This work was partially supported by Consejo Nacional de Ciencia y Tecnologia (Mexico).

ABSTRACT

Efficient approximation algorithms for statistical tests, graph partition and job sequencing are obtained. These polynomial time bounded algorithms guarantee approximate solutions that are within a certain percentage of the true optimal solution value. The specific problems studied are the k-MaxCut; Kolmogorov-Smirnov and Lilliefors tests; and scheduling on: uniform processor systems, open shops, flow shops and job shops. For preemptive scheduling disciplines we show that the flow shop and job shop problems are P-Complete. For other problems it is shown that finding a good approximation algorithm is as hard as finding a good algorithm for the optimal solution, i.e. the approximation problem is also P-Complete. Some problems with this property are: the travelling salesperson, cycle covers, 0-1 Integer Programming, multicommodity network flows, quadratic assignment, general partition, k-MinCluster and the generalized assignment. Efficient exact algorithms are obtained for the Kolmogorov-Smirnov and Lilliefors tests ; preemptive open shop scheduling and nonpreemptive scheduling on 2 processor open shops.

BIOGRAPHICAL SKETCH

The author was born on January 26, 1948 in Monterrey (Mexico). He graduated from the Instituto Tecnologico y de Estudios Superiores de Monterrey A.C. with a B.S. in Computer Science. He joined the graduate school at the University of Minnesota in September 1972.

CONTENTS

CHAPTER I

INTRODUCTION

In this thesis we study several problems for which there are no known polynomial time algorithms. These problems fall into the class of problems known as P-Complete ([8],[21] and [35]), which we now define:

Definition 1.1 A problem L will be said to be P-Complete iff the following holds: L can be solved in polynomial deterministic time iff the class of nondeterministic polynomial time languages is the same as deterministic polynomial time languages (i.e. P = NP). ■

Our notion of P-Complete corresponds to the one used by Sahni [35]. This can easily be seen equivalent to that of Cook [8]. Knuth [23] suggests the terminology NP-Complete. However, his notion of "completeness" is that of Karp [21]. Since the equivalence or non - equivalence of the two notions is not known, we will use the term NP-Complete for problems that can be shown complete with Karp's definition and P-Complete for those which require the definition of Sahni [35]. The reader unfamiliar with P-Complete problems is referred to [21] and [35]. All problems that are NP-Complete (i.e. complete under Karp's definitions) are also

P-Complete ([8] and [35]). The reverse is unknown. At present it is not known whether P = NP. None of the P-Complete problems has a known polynomial time algo - rithm. All P-Complete problems have the property that if one is solvable in polynomial time, then all other problems in this class will also have a polynomial time bounded algorithm. The best known algorithms for these problems require an exponential amount of time, with respect to the size of the problem being solved. Since it is conjectured that $P \neq NP$, it is unlikely that any P-Complete problem has a polynomial solution.

We shall make use of the operator "α" as in $P_1 \; \alpha \; P_2$ to mean problem P_1 reduces to problem P_2. Informally, this will mean that if P_2 can be solved in polynomial time then so will P_1. By P = NP we shall denote the question: Is the class of nondeterministic polynomial time languages the same as the class of deterministic polynomial time languages.

Many of these problems have practical significance and for some we might be interested in algorithms that will produce good approximate solutions quickly. Several authors ([13],[15],[17],[19],[24],[33],[34] and [36]) have shown that even though finding the optimal solution to a problem is very expensive, we can construct very fast algorithms that obtain solutions which are guaranteed to be within a certain percentage of the true optimal solution value. Such algorithms will be termed ε-approximate algorithms.

Definition 1.2 An algorithm will be said to be an ε-approximate algorithm for a problem P iff $\left|\frac{f^* - \hat{f}}{f^*}\right| \leq \varepsilon$ and either i) P is a maximization problem and

$0 < \varepsilon < 1$

or ii) P is a minimization problem and $\varepsilon > 0$

where f* denotes the optimal solution value (assumed > 0) and $\hat{f}$ is the approximate solution value obtained. ∎ For a more detailed definition see [36].

In Chapters II, IV, V and VI we present ε-approximate algorithms for several P-Complete problems. These problems include graph partition and scheduling on: uniform processor systems, open shops, flow shops and job shops.

After looking at these results and at the results of other people in the area ([13],[15],[17],[19],[24],[33],[34] and [36]), we could easily conjecture that every naturally occuring P-Complete problem has a polynomial time bounded ε-approximate algorithm. In Chapter VII we present a strong argument against such a conjecture. We show that for several natural P-Complete problems their approximation problem is also P-Complete. Thus for these problems the approximation problem is as hard as the exact, in the sense that a polynomial time bounded algorithm for the former imply a polynomial time bounded algorithm for the latter. Some of the approximation problems that are P-Complete are the travelling salesperson, cycle covers, 0/1 integer programming, multicommodity network flows, quadratic assignment, general partition, k-MinCluster and the generalized assignment problem.

Several nonpreemptive scheduling problems have been shown to be P-Complete. Some of these can be solved in polynomial time when we allow preemptive schedules. The problems in Chapter IV and V become polynomial solvable when we allow preemptive schedules. However, Ullman [37] has shown that the general preemptive problem with precedence constrains is also P-Complete. In Chapter VII we show that the flow shop and job shop preemptive scheduling is also P-Complete. Even though preemptive scheduling is as hard as nonpreemptive, the solutions given by the former are much better than the ones given by the latter. In Chapter VI we present bounds between the ratio of nonpreemptive and preemptive optimal solutions.

Approximate solutions are not restricted to problems which are hard to solve exactly. In Chapter III we present approximate solutions to problems that belong in P, i.e. problems that can be solved in polynomial time. Here, we look at the Kolmogorov-Smirnov and

Lilliefors statistical tests. In the case of these problems, we want to compute a value K_{max}, and compare it against a given critical value, in order to accept or reject a hypothesis. This critical value is itself known only to some accuracy. So, there is no need to compute K_{max} any more accurately than the accuracy of the critical value. Even though the exact value of K_{max} can be determined in O(n) time, we show how to obtain an approximate value in less time and using less space than the exact algorithm. The O(n) exact algorithm presented in Chapter III is itself an improvement over the best previously known algorithm (which had a complexity of O(n log n)).

CHAPTER II

GRAPH PARTITION

2.1 Introduction

In this Chapter, we look at a problem that arises in information retrieval [20]. This problem is that of obtaining an optimal set of k or n/k clusters given n documents. When the optimization criteria is to maxi - mize the dissimilarity among the clusters, it is shown that ε-approximate solutions may be obtained in O(n) time for $\varepsilon \geq 1/k$ or $\varepsilon \geq k/n$ respectively. Thus, for the n/k cluster problem the solution values are guaranteed to be very close to the optimal for "large" n. When the optimization criteria is that of minimizing the dissimilarity among documents in the same cluster, the approximation problem becomes P-Complete, as we shall see in Chapter VII. Note that this change in optimization criteria does not change the optimal solutions but does change the complexity of obtaining approximate solutions.

The set of n documents is represented by a weighted undirected complete graph G. The vertices are label 1 thru n with vertex i corresponding to document i and the weight of the edge (i,j), w(i,j) is a measure of the documents i, j. The objective is to partition the set of n documents into k disjoint clusters (groups) such that the total dissimilarity among clusters (i.e. Σ w(i,j) for i,j in different clusters) is maximized. Sometimes, we may be interested in obtaining n/k clusters for some constant integer k. We first show that the clustering problem with these two optimization constrains is P-Complete. Then we present the approx - imation algorithm.

The following known NP-Complete problems (see Karp [21]) shall be used in the reductions:

i) Partition: Given s integers $(c_1, c_2, \ldots, c_s)$ is there a subset $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$

ii) Cut: Given an undirected graph G(N,A), weighting function $w : A \to Z$, positive integer W, is there a set $S \subseteq N$ such that $\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w\{u,v\} \geq W$

iii) Sum of Subsets: Given n+1 positive integers $(r_1, r_2, \ldots, r_n, m)$, is there a subset of the r_i's that sums to m.

Before proceeding with the completeness proofs we present below abstract formulations of the cluster problems (k-MaxCut and n/k-MaxCut) together with some generalizations of the Partition problem.

a) k-Partition: Given n integers $r_1, r_2, \ldots, r_n$ and an integer $k \geq 2$, are there disjoint subsets $I_1, I_2, \ldots, I_k$ such that $\bigcup_{i=1}^{k} I_i = 1,2,\ldots,n$ and

$$\sum_{i \varepsilon I_1} r_i = \sum_{i \varepsilon I_\ell} r_i, \quad 2 \leq \ell \leq k$$

(The Partition problem i) above is then just the 2-Partition problem.)

b) k-Cut: Given an undirected graph $G = (N,A)$, integer $k \geq 2$, weighting function $w : A \rightarrow Z$, positive integer W, are there disjoint sets $S_1, S_2, \ldots, S_k$ such that $\bigcup_{i=1}^{k} S_i = N$ and $\sum_{\substack{\{u,v\} \varepsilon A \\ u \varepsilon S_i \\ v \varepsilon S_j \\ i \neq j}} w(u,v) \geq W$.

(The Cut problem ii) above is just the 2-Cut problem.)

b') k-MaxCut[1]: Find disjoint sets, S_1 $1 \leq i \leq k$,

[1]The k-MaxCut problem is also a generalization of the 'grouping of ordering data' problem studied in [1]. [1] restricts the set S_i to be sequential, i.e., if $i,j \varepsilon S_\ell$ and $i < j$ then $i+1, i+2, \ldots, j-1 \varepsilon S_\ell$. [1] presents an $O(kn^2)$ dynamic programming algorithm for this.

such that $\bigcup_{i=1}^{k} S_i = N$ and $\sum_{\substack{\{u,v\}\varepsilon A \\ u\varepsilon S_i \\ v\varepsilon S_j \\ i\neq j}} w\{u,v\}$

is maximized.

c) $\lceil n/k \rceil$-Partition: same as a) except that the number of disjoint subsets is now $\lceil n/k \rceil$, $k \geq 2$.

d) $\lceil n/k \rceil$-Cut: same as b) except the number of disjoint subsets is now $\lceil n/k \rceil$, $k \geq 2$.

d') $\lceil n/k \rceil$-MaxCut: same as b') with k replaced by $\lceil n/k \rceil$.

2.2 Completeness Proofs and Approximations

In this section we first prove (Lemma 2.2.1) the completeness of the problems a) to d). Then we will present algorithm MAXCUT which generates approximate solutions.

Lemma 2.2.1

(I) The following problems are NP-COMPLETE

a) k-Partition

b) k-Cut

c) $\lceil n/k \rceil$-Partition

d) $\lceil n/k \rceil$-Cut

(II) k-MaxCut and $\lceil n/k \rceil$-MaxCut are P-Complete.

Proof

We have to show that i) if P = NP then a)-d) can be solved in polynomial time and ii) if a)-d) can be solved in polynomial time then the class of P-Complete problems is polynomial solvable (this can be shown by reducing any known P-Complete problem to a)-d)).

i) is trivial, so we shall only show ii).

ii) Partition α k-Partition. For any Partition problem $(c_1, c_2, \ldots, c_s)$ define a k-Partition problem $(r_1, r_2, \ldots, r_{s+k-2})$ where

$$r_i = \begin{cases} c_i & 1 \leq i \leq s \\ p & s+1 \leq i \leq s+k-2 \end{cases} \quad \text{and } p = \Sigma c_i/2$$

(we may assume that Σc_i is even as otherwise the partition problem clearly has no solution). Now, $\Sigma r_i = kp$ and the k-Partition problem has a solution iff the corresponding Partition problem has one.

k-Partition α k-Cut. If the given k-Partition problem is $(r_1, r_2, \ldots, r_n)$ define the corresponding k-MaxCut problem to be $G = (N,A)$ with $N = (1,2,\ldots,n)$,

$$A = \{\{i,j\} | i \epsilon N, \; j \epsilon N, \; i \neq j\}$$

$$w(\{i,j\}) = r_i r_j$$

and $$W = \frac{(k-1)}{2k} (\Sigma r_i)^2$$

(Note, we may again assume k divides Σr_i.) Clearly, there is a k-Cut $\geq$ W iff $(r_1, r_2, \ldots, r_n)$ has a k-partition.

Partition α $\lceil n/k \rceil$-Partition. We prove this only for $k = 2$. From the partition problem $(c_1, c_2, \ldots, c_n)$, $s > 3$, construct the following $\lceil n/2 \rceil$-partition problem:

$$r_i = c_i \qquad 1 \le i \le s$$

$$r_i = p \qquad s+1 \le i \le n$$

$$n = 2(s-2)$$

$$p = \Sigma c_i/2 \qquad \text{(if } \Sigma c_i \text{ is odd then there is no partition)}$$

Clearly, the partition problem has a solution iff the $\lceil n/2 \rceil$-Partition problem has one.

$\lceil n/k \rceil$-Partition α $\lceil n/k \rceil$-MaxCut. The proof for this is similar to that for k-Partition α k-Cut, II follows from I and the techniques of [23]. ■

We note that the proofs used in Lemma 2.2.1 are minor extensions of the ones used in Karp [21]. The (n-k)-Partition and (n-k)-MaxCut problems are polynomial. We next present an approximation algorithm for the k-MaxCut and $\lceil n/k \rceil$-MaxCut problems. Consider the algorithm MAXCUT below: (Intuitively, this algorithm begins by placing one vertex of G into each of the ℓ sets S_i $1 \le i \le \ell$; the remaining $n-\ell$ vertices are examined one at a time. Examination of a vertex, j, involves determining the set S_i $1 \le i \le \ell$ for which

$\sum_{m\varepsilon S_i} w\{m,j\}$ is minimal. Vertex j is then inserted/assigned to this set.) A similar algorithm for this problem appears in [20].

Algorithm MAXCUT (ℓ,G)

// ℓ...number of disjoint sets, S_i, into which the vertices, N = (1,2,...,n), of the graph G(N,A) are to be partitioned, SOL...the value of the vertex partitioning obtained, w{i,j}...weight of the edge {i,j}. SET(i) ... the set to which vertex i has been assigned (SET(i) = 0 for all vertices not yet assigned to a set) WT(i) ... used to compute $\sum_{m\varepsilon S_i} w\{m,j\}$, $1 \le i \le \ell$.

This algorithm assumes that the graph G(N,A) is presented as n lists $v_1, v_2, \ldots, v_n$. Each list v_i contain all the edges, {i,j}εA, that are adjacent to vertex i. No assumption is made on the order in which these edges appear in the list. //

Step 1 // Initialize // If ℓ >n then do

$$SOL \leftarrow \sum_{\{i,j\}\varepsilon A} w\{i,j\}$$

$S_i \leftarrow \{i\}$ $\quad 1 \le i \le n$

$S_i \leftarrow \{\emptyset\}$ $\quad n+1 \le i \le \ell$

Stop

end;

otherwise $\quad S_i \leftarrow i \quad 1 \le i \le \ell$

$WT(i) \leftarrow 0 \quad 1 \leq i \leq \ell,$

$SET(i) \leftarrow i \quad 1 \leq i \leq \ell$

$SET(i) \leftarrow 0 \quad \ell+1 \leq i \leq n$

$$SOL \leftarrow \sum_{\substack{\{i,j\} \varepsilon A \\ 1 \leq i < j \leq \ell}} w\{i,j\}$$

$j \leftarrow \ell + 1$

<u>Step 2</u> // process edge list of vertex j //

<u>for</u> each edge {j,m} on the edge list of vertex j <u>do</u>

<u>if</u> SET(m) $\neq$ 0 <u>then</u> WT(SET(m)) $\leftarrow$ WT(SET(m)) + w{j,m} ;

<u>end</u>

$d_j \leftarrow$ degree of vertex j = # of edges adjacent to Vertex j

<u>Step 3</u> // find the set for which $\sum_{m \varepsilon S_i} w\{j,m\}$ is minimal//

look at WT(a) $1 \leq a \leq \min\{d_j+1, \ell\}$ and determine i such that WT(i) is minimal in this range. (Note that if $d_j + 1 \leq \ell$ then at least one of WT(a) $1 \leq a \leq d_j+1$ must be 0 and minimal. For $d_j+1 \geq \ell$ all WT(a) are looked at and the minimal found.)

<u>Step 4</u> // assign vertex j to set S_i //

SET(j) $\leftarrow$ i

<u>Step 5</u> // update SOL and reset WT //

<u>for</u> each edge {j,m}εA for which SET(m) $\neq$ 0 <u>do</u>

<u>if</u> SET(m) $\neq$ i <u>then</u> SOL $\leftarrow$ SOL + w{j,m}

WT(SET(m)) $\leftarrow$ 0 ; <u>end</u>

<u>Step 6</u> // next vertex // $j \leftarrow j + 1$,

<u>if</u> $j \leq n$ <u>then</u> <u>go</u> <u>to</u> STEP 2

<u>otherwise</u> terminate algorithm

end MAXCUT

<u>Lemma 2.2.2</u>

The time complexity of algorithm MAXCUT is $O(\ell + n + e)$ on a random access machine (n is the number of vertices, e the number of edges and ℓ the number of groups into which the vertices are to be partitioned).

<u>Proof</u>

Step.	Time Per Execution	Total Time
1	$O(n + e + \ell)$	$O(n + e + \ell)$
2	$O(d_j)$	$O(e)$
3	$O(d_j + 1)$	$O(e + n)$
4	$O(1)$	$O(n)$
5	$O(d_j)$	$O(e)$
6	$O(1)$	$O(n)$

Hence , the total time = $O(n + e + \ell)$

<u>Lemma 2.2.3</u>

Algorithm MAXCUT is a 1/k - approximate algorithm for the k-MaxCut problem.

<u>Proof</u>

If $n \leq k$ then MAXCUT generates the optimal solution value.

Define the internal weight of the set S_i to be

$\sum_{\substack{u \neq v \\ u,v \in S_i}} w\{u,v\}$. Then the total internal weight

$(TIW) = \sum_{i=1}^{k}$ internal weight (S_i). The external weight

$$(EW) = \sum_{\substack{u,v \\ u \in S_i \\ v \in S_j \\ i \neq j}} w\{u,v\} .$$

In <u>Step 4</u> when vertex j is assigned to set i either

$WT(i) = 0$ (corresponding to $d_j < \ell$) or

$WT(i) \leq \sum_{1 \leq m \leq k} WT(m)/k$. i.e. if the total internal weight

increases by WT(i) then the external weight increases by at least (k-1)WT(i). Consequently, at termination, TIW < EW/(k-1) (note that SOL = EW). But, the optimal value of the solution $\leq$ TIW + EW. Let F* be the optimal. EW = SOL is the approximation obtained by MAXCUT. The worst case occurs when TIW approaches EW/(k-1). Hence

$$\left|\frac{F^* - SOL}{F^*}\right| < 1/k.$$

From Lemma 2.2.3 it follows that algorithm MAXCUT is a k/n -approximate algorithm for the n/k-MaxCut problem. While approximately optimal clusters may be found in linear time using the maximization criteria, one of the results in Chapter VII is that finding approximately optimal clusters under the minimization criteria is P-Complete. This is the approximation problem is as hard as the exact, in the sense that a polynomial time bounded algorithm for the former implies a polynomial time bounded algorithm for the latter.

CHAPTER III

STATISTICAL TESTS

3.1 Kolmogorov-Smirnov and Lilliefors Tests

The Kolmogorov-Smirnov and Lilliefors tests allow us to evaluate the hypothesis that a collected data set, i.e. a random sample $X_1, X_2, \ldots, X_n$, was drawn from a specified continuous distribution function F(X). For both tests, a determination is made of the numeric difference between the specified distribution function F(X), and the sample distribution function S(X) as defined by equation 3.1.1.

$$S(X) = \{(\text{number of } X_i\text{'s} \le X)/n\} \quad (3.1.1)$$

If the sample, $X_1, X_2, \ldots, X_n$, has been sorted into nondecreasing order so that $X_1 \le X_2 \le \ldots \le X_n$, then the Kolmogorov-Smirnov statistics K^+_{max} (maximum positive) K^-_{max} (maximum negative) and K_{max} (maximum absolute) deviations are computed by formulas 3.1.2.

$$K^+_{max} = \sqrt{n} \max_{1 \le j \le n} \left\{ \frac{j}{n} - F(X_j) \right\}$$

$$K^-_{max} = \sqrt{n} \max_{1 \le j \le n} \left\{ F(X_j) - \frac{j-1}{n} \right\} \quad (3.1.2)$$

$$K_{max} = \max \{ K^+_{max} , K^-_{max} \}$$

The distribution functions of K^+_{max}, K^-_{max}, K_{max} are known and tabulated. We accept the null hypothesis that the sample was indeed drawn from the distribution F(X) if the statistics computed do not exceed the critical values tabulated for the level of significance selected. For certain F(X), (see [25],[26]) tabulated values of the test statistic distributions are available for the case where the actual parameters of F(X) have been replaced by estimates computed from the sample. The test also has application for certain spectral tests, see for example, [9, p. 197].

Previous algorithms [22, 27 and 32] for computing these test statistics are essentially identical to algorithm K below:

<u>Algorithm K(K^+_{max}, K^-_{max}, K_{max})</u>

// Knuth's algorithm for Kolmogorov-Smirnov test statistics [22 pp.44] //

<u>Step 1</u> obtain the n observations $X_1, X_2, \ldots, X_n$

<u>Step 2</u> sort them so that $X_1 \leq X_2 \leq \ldots \leq X_n$

<u>Step 3</u> Compute K^+_{max}, K^-_{max} and K_{max} using equation 3.1.2.

<u>end</u> K ∎

Since, step 2 sorts the observations, it requires O(n log n) time. The remainder of the algorithm takes O(n) time (assuming F(X) may be computed in a constant amount of time O(1)). Hence, the total time required is O(n log n). The algorithm we present in section 3.2 computes the test statistics K^+_{max}, K^-_{max} and K_{max} without explicity sorting the X_i's. This algorithm has a time complexity of O(n). The tabulated acceptance/rejection values of these statistics are usually accurate only to three or four decimal places. Hence, there seems little point in computing these statistics to greater preci - sion than the tabulated values. With this in mind, we present in section 3.3 an approximation algorithm which guarantees a certain closeness to the exact values of K^+_{max}, K^-_{max} and K_{max}. This approximate algorithm require less storage space than the exact algorithm and so should be useful when n is large. The computing time is still O(n). Empirical tests, in section 3.4, show that the approximation algorithm is actually slightly faster than the exact algorithm. The desired closeness of the approximate and exact solutions can be fixed through an algorithm parameter.

Both the exact and approximate algorithms apply equally well to the Lilliefors test [6] which is very similar to the Kolmogorov-Smirnov Test. In this test, instead of using the raw observations, X_i, the observations are first normalized as in equation (3.1.3) and then these normalized observations are used in (3.1.2) to obtain the test statistics. If the Z_i's are the normalized values of X_i then

$$Z_i = \frac{X_i - \overline{X}}{s} \qquad 1 \le i \le n$$

$$\text{where } \overline{X} = \sum_1^n X_i/n \qquad (3.1.3)$$

$$\text{and } s = \sqrt{\left(\sum_1^n (X_i - \overline{X})^2/(n-1)\right)}$$

Since the normalization can clearly be done in O(n) time and the rest of the computation is the same as in the Kolmogorov-Smirnov test, it follows that our algorithms can also be used to obtain the Lilliefors statistics in O(n) time.

3.2 Exact Solutions

Our algorithm to compute the values of K^+_{max}, K^-_{max} and K_{max} for the Kolmogorov-Smirnov test proceeds by dividing the range of the cumulative distribution function F(X) into n+1 intervals. The point y, $0 \le y \le 1$ lies in the interval $\lceil y * n \rceil$. For each of the n samples or points X_i $1 \le i \le n$, the value of $F(X_i)$ is computed. For each of the n+1 intervals for F(X), the number of sample points for which F(X) is in that interval is recorded, together with the minimum and maximum values of F(X) achieved in that interval. Theorem 3.2.1 shows that this information is sufficient to enable an accurate determination of the values of K^+_{max}, K^-_{max} and K_{max}. We first formally present the algorithm. Lemmas 3.2.1 and 3.2.2 analyze the time and space complexity of this algorithm.

Algorithm KS$(n, K^+_{max}, K^-_{max}, K_{max})$

// This algorithm inputs n sample points and performs the Kolmogorov-Smirnov test against the cumulative distribution function F(X). The outputs of the algorithm are: K^+_{max}... the K^+ maximum deviate

K^-_{max}... the K^- maximum deviate

K_{max}... the absolute maximum deviate

3 vectors of size n+1 each are made use of:

NUM_i... number of samples in bin i

MAX_i... maximum sample value in bin i

MIN_i... minimum sample value in bin i

$\Big\}\; 0 \le i \le n$

//

Step 1 // Initialize //

```
for i ← 0 to n do
    XMIN_i ← 1
    XMAX_i ← 0
    NUM_i ← 0
end
```

Step 2 // input observations and put into bins //

```
for i ← 1 to n do
    input X
    f ← F(X)
    j ← ⌈f*n⌉   // compute bin for X //
    NUM_j ← NUM_j + 1
    if MAX_j < f then [MAX_j ← f]
```

```
                if MIN_j > f then [ MIN_j ← f ]
            end
Step 3  // process each bin finding maximum positive
           and negative deviates //
            j ← 0; DP ← 0; DN ← 0;
            for i ← 0 to n do
            if NUM_i > 0 then [ z ← MIN_i - j/n
                                if z > DN then [ DN ← Z]
                                j ← j + NUM_i
                                z ← j/n - MAX_i ,
                                if z > DP then [ DP ← z ]
                                ]
            end
Step 4  // Compute K+max, K-max and Kmax //
            K+max ← √n * DP
            K-max ← √n * DN
            Kmax ← max { K+max, K-max }
            return
end KS ■
```

We now prove that the above algorithm does in fact give the correct results.

<u>Theorem 3.2.1</u> Algorithm KS gives the correct values for K^+_{max}, K^-_{max} and K_{max}.

<u>Proof</u> We prove this only for the case where all the

sample points X_i are distinct. The extension to the general case is fairly straight-forward. The proof is in two parts. First, we show that it is sufficient to consider only the smallest and largest samples in each bin and then that algorithm KS determines accurately the index of these sample points in case all samples were sorted into nondecreasing order.

(i) Since F(X) is a cumulative distribution function, it must be monotone increasing in X i.e. $x > y$ iff $F(x) > F(y)$. Hence, it is immaterial whether for each bin we retain the largest and smallest sample points or the largest and smallest values for F( ). Let X be the smallest sample point, Z the largest and Y any sample point in bin i. Then $X \leq Y \leq Z$ and $F(X) \leq F(Y) \leq F(Z)$. Let j, k, ℓ be the number of sample points $\leq$ X, Y and Z respectively. Then $j \leq k \leq \ell$.

By definition $K^+(\hat{x}_j) = j/n - F(\hat{x}_j)$. If $k \neq \ell$ then:

$$K^+(Y) = k/n - F(Y)$$

$$\leq \frac{\ell}{n} - \frac{1}{n} - (F(Z) - 1/n)$$

$$= \ell/n - F(Z) = K^+(Z)$$

also, if $k \neq j$,

$$K^-(Y) = F(Y) - \frac{(k-1)}{n}$$

$$\leq F(X) + \frac{1}{n} - \frac{(j-1)}{n} - \frac{1}{n}$$

$$= F(X) - \frac{(j-1)}{n} = K^-(X)$$

Hence, for any bin it is sufficient to consider only the maximum and minimum in that bin.

(ii) Again, since F(X) is monotone increasing, all sample points in bin ℓ are less than all sample points in bin $\ell + 1$, $1 \leq \ell < n$. Therefore if X is the smallest and Z the largest sample points in bin ℓ then the number of sample points < X is $\sum_{0 \leq i < \ell} NUM_i$ and the number $\geq$ Z is $\sum_{0 \leq i \leq \ell} NUM_i$.

(i) and (ii) show that the correct values for K^+_{max} and K^-_{max} are obtained. By definition of K_{max}, the correct K_{max} is also obtained.

Lemma 3.2.1 The time complexity of algorithm KS is O(n).

Proof

Step.	Time.
1	O(n)
2	O(n)
3	O(n)
4	O(1)

Hence the total time = O(n)

Lemma 3.2.2 Algorithm KS requires 3n + c amount of space where c is a constant.

Proof The vector NUM, MAX and MIN each are of size n + 1. A fixed amount of additional space for simple variables such as i and j is also required. Hence the total space requirements are 3n + c.

3.3 Approximations

In this section we present an algorithm to deter - mine approximatetely, the values of K^+_{max}, K^-_{max} and K_{max}. This algorithm is slightly faster than the algorithm of section 3.2 and requires at most 1/3 the space required by that algorithm. This algorithm is very similar to algorithm KS. It has only m+1 $\leq$ n+1 bins and does not keep track of the values of MAX_i and MIN_i. Instead the approximation $MAX_i \simeq MIN_i \simeq (i - .5)/m$ is used. Before obtaining bounds on the algorithm we state it formally to point out the differences from algorithm KS.

Algorithm APPROX_KS (n, $\hat{K}^+_{max}$, $\hat{K}^-_{max}$, $\hat{K}_{max}$, m)

// Find approximations to K^+_{max}, K^-_{max}, K_{max} using only m+1 bins. Variables have same meanings as in algorithm KS //

Step 1 // initialize bins //

```
        for i ← 0 to m do
            NUM_i ← 0
        end
```

Step 2 // input and count number of sample points in each bin //

```
          for i ←1 to n do
              input X
              f ← F(X); j ← ⌈f*m⌉ ; //compute bin for
                                          X //
              NUM_j ← NUM_j + 1
          end
Step 3  // process each bin finding approximate values
for maximum positive and negative deviates from F(X) //
          DP ← 0; DN ← 0;
          if NUM_0 > 0 then [ DP ← NUM_0/n ]
          j ← NUM_0
          for i ← 1 to m do
          if NUM_i > 0 then [ z  ← (i-.5)/m - j/n
                            if z ← DN then [DN ← z]
                            j ← j + NUM_i
                            z ← j/n - (i - .5)/m
                            if z ← DP then [DP ← z]
                            ]
          end
Step 4  // Compute K̂+max, K̂-max, K̂max //
          K̂+max ← √n * DP
          K̂-max ← √n * DN
          K̂max ← max { K̂+max, K̂-max }
          return
end APPROX_KS ■
```

Theorem 3.3.1 The following relations hold between the approximate values $\hat{K}^+_{max}$, $\hat{K}^-_{max}$ and $\hat{K}_{max}$ as given by algorithm APPROX_KS and the exact values K^+_{max}, K^-_{max} and K_{max} given by algorithm KS:

(i) $|\hat{K}^+_{max} - K^+_{max}| \leq \sqrt{n}\ /(2m)$

(ii) $|\hat{K}^-_{max} - K^-_{max}| \leq \sqrt{n}\ /(2m)$

and (iii) $|\hat{K}_{max} - K_{max}| \leq \sqrt{n}\ /(2m)$

Proof (i) follows from the observation that for any bin, i, if $\ell = \sum_{0\leq j\leq i} NUM_i$; if X is a sample point such that $\lceil F(X) * m \rceil = i$ and if there are k sample points $\leq$ X then $K^+(X) - \hat{K}^+(\text{bin } i)$

$$\doteq \sqrt{n}\ (k/n - F(X)) - \sqrt{n}\ (\ell/n - (i-.5)/m)$$

$$= \sqrt{n}\ ((k-\ell)/n + (i-.5)/m - F(X))$$

$$\leq \sqrt{n}\ /(2m).$$

The proofs for (ii) and (iii) are similar. ∎

Lemma 3.3.1 The computing time for algorithm APPROX_KS is O(n) and the space required is n+c for m $\leq$ n and c a constant.

Proof Follows the pattern of the proofs for Lemmas 3.2.1 and 3.2.2. ∎

3.4 Empirical Results

In order to determine the relative performance of our algorithms on practical sample sizes, we programmed algorithms KS, APPROX_KS and algorithm K in FORTRAN and ran several tests on the Cyber 74. The sorting method used for algorithm K was heapsort. Three distribution functions: normal, exponential and uniform were tried so as to reflect the differences in the computing times for F(X). Table 3.4.1 presents the results obtained for various sample sizes. The times are the mean computing times over several experiments. As can be seen from this table, algorithm K required from about 2 to 3 times the time required by our algorithms. This difference will, of course, become larger for larger sample sizes. Algorithm APPROX_KS took roughly the same time as algorithm KS but used considerably less storage. The observed difference between the exact and approximate values of the test statistics was about half the theoretical maximum of Theorem 3.3.1.

We have presented linear time algorithms for the Kolmogorov-Smirnov and Lilliefors tests. While these algorithms are faster than those of [6], [22], [27] and [32], one should note that this speed up is obtained by avoiding a sort of the sample. If the sample is already known to be sorted or has to be sorted for some other reason, then the values of K^+_{max}, K^-_{max} and K_{max} can be computed more efficiently by a direct application of (3.1.2). Thus, we recommend the use of algorithm KS when the sample is not sorted to begin with, nor has to be sorted for other perpose. Algorithm APPROX_KS is recommended in cases where n is large, storage small and the acceptance/rejection values of K^+_{max}, K^-_{max} and K_{max} are themselves known only approximately (i.e. only a few digits of significance is desired). The value of m to use can be determined using Theorem 3.3.1.

distribution	sample size n	number of experiments	COMPUTING TIME										K/KS
			K		KS		APPROX KS						
							M = n		M = n/4		M = n/16		
			mean	std.	mean	std.	mean	std.	mean	std.	mean	std.	
exponential	100	50	17.8	1.0	9.0	0.9	7.9	0.8	6.8	0.8	6.4	0.5	1.98
	500	30	107	2.7	44.3	1.8	35.7	2.0	33.0	1.5	31.2	1.2	2.42
	1000	20	231	2.0	88.0	2.0	78.0	2.0	66.0	2.2	63.0	2.0	2.63
	5000	5	1342	8.9	439	5.9	387	4.9	329	6.2	308	2.4	3.06
uniform	100	50	15.9	0.8	7.2	0.7	5.8	0.7	4.6	0.7	4.4	0.8	2.21
	500	30	97.9	2.5	35.2	1.6	29.1	1.7	23.4	1.1	21.5	1.1	2.78
	1000	20	213	4.6	71.8	2.9	59.9	2.6	46.9	1.8	43.9	2.2	2.97
	5000	5	1260	5.0	352	8.0	294	4.4	234	4.0	213	4.6	3.58
normal	100	50	25.9	1.3	17	1.0	16	0.9	15.2	0.8	14.8	0.9	1.52
	500	30	147	3.7	84	2.9	80	2.9	75	2.3	72.0	2.5	1.75
	1000	20	308	10.3	168	5.7	158	4.9	149	5.5	145	4.8	1.83
	5000	5	1760	31.0	838	20.0	801	4.5	738	4.6	731	6.9	2.10

Times in milliseconds

Table 3.4.1 MEAN EXECUTION TIMES FOR THE KOLMOGOROV-SMIRNOV TEST STATISTICS.

CHAPTER VI

UNIFORM PROCESSOR SYSTEMS

4.1 Introduction

A uniform processor system [15] is one in which the processors P_1, $P_2, \ldots,$ P_m have relative speeds s_1, $s_2, \ldots, s_m$ respectively. It is assumed that the speeds have been normalized such that $s_1 = 1$ and $s_i \geq 1$ for $2 \leq i \leq m$. The problem of scheduling n independent tasks (J_1, $J_2, \ldots, J_n$) with execution times (t_1, $t_2, \ldots,$ t_n) on m uniform processors to obtain a schedule with the optimal (least) finish time is known to be NP-Complete [3,15]. Hence, it appears unlikely that there is any polynomial time bounded algorithm to generate such schedules. For preemptive scheduling, however, optimal finish time algorithms can be obtained in polynomial time [16, 30]. Horowitz and Sahni [15] showed that for any m, polynomial time algorithms exist to obtain schedules with a finish time arbitrarily close to the optimal finish time. The complexity of these algorithms was, however, exponential in m. The purpose of this Chapter is to study the finish time properties of LPT schedules with respect to the optimal finish time.

Definition 4.1.1

An LPT (Largest Processing Time) schedule is a schedule obtained by assigning tasks to processors in order of nonincreasing processing times. When a task is being considered for assignment to a processor, it is assigned to that processor on which its finishing time will be earliest. Ties are broken by assigning the task to the processor with least index. ■

One may easily verify that for identical processor systems, this definition is equivalent to that of [4], p. 100. Graham [13] studied LPT schedules for the special case of identical processors, i.e., $s_i = 1$, $1 \leq i \leq m$. If $\hat{f}$ is the finish time of the LPT schedule and f^* the optimal finish time, then Graham's result is that $\hat{f}/f^* \leq \frac{4}{3} - \frac{1}{3m}$ and that this bound is the best possible bound. In section 4.2 we extend his work to the general case of uniform processors. While the bound we obtain is best possible for $m = 2$, it appears that it is not so for $m > 2$. In view of this, we turn our attention to another special case of uniform processors i.e., $s_i = 1$, $1 \leq i < m$ and $s_m = s \geq 1$. This case has previously been studied by J.W.S. Liu and C.L. Liu [29]. Using a priority assignment according to lengths of tasks, they show that $f/f^* \leq \frac{2(m-1+s)}{s+2}$ for $s \leq 2$ and $f/f^* \leq \frac{m-1+s}{2}$ for $s \geq 2$, where f is the finish time of the priority schedule.

Similar bounds for list schedules are also obtain ed by them. We are able to show that for $m \geq 3$ $\hat{f}/f^* \leq 3/2 - 1/(2m)$ and that this bound is the best possible for $m = 3$. For $m > 3$ we conjecture that $\hat{f}/f^* \leq 4/3$.

Before presenting our results we develop the necessary notation and basic results. If S is the set of tasks being scheduled, then it will sometimes be necessary to distinguish between finish times of different sets of tasks. To do this S will appear as a superscript along with $\hat{f}$ or f^* as in $\hat{f}^S$ and f^{*S}. If the number of processors is important, then this number will appear as a subscript as in $\hat{f}_m$, f^{*S}_m etc. We shall refer to the sets of tasks (jobs) by their task execution time. Thus we speak of a set, S, of tasks $(t_1 \geq t_2 \geq \ldots \geq t_n)$ meaning the execution time of task J_i is t_i and $t_i \geq t_{i+1}$, $1 \leq i < n$. The m processors $P_1, P_2, \ldots, P_m$ are assumed ordered such that $s_1 = 1$ and $1 \leq s_i \leq s_{i+1}$, $2 \leq i < m$. The following result from [4, p. 102] is made use of:

<u>Lemma 4.1.1</u> If for any m $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ is the smallest set of tasks for which $\hat{f}/f^* > k$ then t_n determines the finish time $\hat{f}$ (i.e. task n has the latest completition time).

<u>Proof</u> Appears in [4] p. 102. ■

4.2 Basic Results

In this section, we prove two important lemmas that are used throughout the Chapter (Lemmas 4.2.2 and 4.2.3). We also derive the bound 2m/(m+1) for the ratio $\hat{f}/f^*$ for the general m-processor system. Examples are shown for which $\hat{f}/f^*$ approaches 3/2 as $m \to \infty$.

We begin with the following lemma, informally, it states that if either the LPT or optimal schedule of an (m+1)-processor system has an idle processor, then the ratio $\hat{f}/f^*$ for this schedule is no worse that $\hat{f}/f^*$ For m processors.

Lemma 4.2.1 For $m \geq 1$, let $g(m, s_2,\ldots,s_m)$ be such that $\hat{f}_m/f^*_m \leq g(m, s_2,\ldots,s_m)$. Consider any (m+1)-processor system with job set $S = (t_1 \geq t_2 \geq\ldots\geq t_n)$ and processor speeds $1 = s_1 \leq s_2 \leq\ldots\leq s_{m+1}$. If a processor is idle in either the LPT or optimal schedule of S, then $\hat{f}^S_{m+1}/f^{*S}_{m+1} \leq g(m, s_3/s_2,\ldots, s_{m+1}/s_2)$.

Proof Suppose in the LPT schedule of S a processor P_i is idle. Then it must be the case that in the optimal schedule, P_i is also idle. Otherwise, $\hat{f}^S_{m+1} \leq t_n/s_i$, $f^{*S}_{m+1} \geq t_n/s_i$ and $\hat{f}^S_{m+1}/f^{*S}_{m+1} = 1$. So we need only consider the case when P_i is idle in the optimal schedule. If P_i is idle then clearly P_1 is also idle or can be made idle without increasing f^* by scheduling the jobs on P_1 onto P_i. Consider the m-processor system with job set S and processor speeds $1 = s_2/s_2 \leq s_3/s_2 \leq \ldots \leq s_{m+1}/s_2$. Then by assumption, for this system, $\hat{f}_m^S/f_m^{*S} \leq g(m, s_3/s_2, \ldots, s_{m+1}/s_2)$. Moreover, $\hat{f}_{m+1}^S \leq s_2 \hat{f}_m^S$ and $f_{m+1}^{*S} = s_2 f_m^{*S}$. It follows that $\hat{f}_{m+1}^S/f_{m+1}^{*S} \leq g(m, s_3/s_2, \ldots, s_{m+1}/s_2)$. ■

The next lemma gives an estimate of $\hat{f}/f^*$ for the case when $\hat{f}$ is determined by the job with the smallest execution time.

Lemma 4.2.2 Consider an m-processor system with job set $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ and speeds $s_1, s_2, \ldots, s_m$. If in the LPT schedule of S, the finish time $\hat{f}$ is determined by t_n, (i.e., if task n has the latest completition time) then $\hat{f}/f^* \leq 1 + \frac{(m-1)t_n}{Qf^*}$, where $Q = \Sigma s_i$.

Proof Let the LPT schedule be as shown in Fig. 4.2.1 , where P_k determines the finish time. Each T_i is the sum (possibly 0) of execution times of jobs scheduled on P_i prior to t_n's assignment, $T_1 + \ldots + T_m = t_1 + \ldots t_{n-1}$.

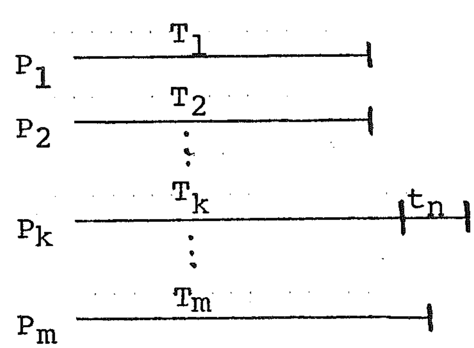

Fig. 4.2.1

Since task n determines the finish time,

$\hat{f} = (T_k + t_n)/s$ and $\frac{T_i + t_n}{s_i} \geq \hat{f}$ for $i \neq k$. Hence ,

$\hat{f}s_i - T_i \leq t_n$ and so $\hat{f} \sum_{i \neq k} s_i - \sum_{i \neq k} T_i \leq (m-1)t_n$.

This, together with $\hat{f}s_k = T_k + t_n$ yields

$$\begin{aligned} \hat{f}Q &\leq T_i + mt_n \\ &= t_i + (m-1)t_n. \end{aligned}$$

Since $f^* \geq t_i/Q$, we get $\hat{f}/f^* \leq 1 + \frac{(m-1)t_n}{f^*Q}$ ∎

Using Lemmas 4.2.1 and 4.2.2, we can now derive a bound for the m processor system.

<u>Theorem 4.2.1</u> For an m-processor system, $\hat{f}/f^* \leq \frac{2m}{m-1}$.

<u>Proof</u> For $m = 1$, the theorem obviously holds. Now suppose the theorem holds for $1,2,\ldots,m-1$ processors but fails for m-processors. Let $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ be the smallest set of jobs which gives a bound $\hat{f}_m/f_m^* > \frac{2m}{m+1}$. Then by Lemma 4.1.1, t_n determines the finish time. There are two cases to consider. Both lead to a contradiction.

<u>Case 1</u> $n \geq m + 1$. Then by Lemma 4.2.2,

$$\begin{aligned} f_m/f_m^* &\leq 1 + \frac{(m-1)t_n}{Qf^*} \\ &\leq 1 + \frac{(m-1)t_n}{Q(\frac{\Sigma t_i}{Q})} \end{aligned}$$

$$\leq 1 + \frac{(m-1)t_n}{nt_n} = 1 + \frac{(m-1)}{n} \leq 1 + \frac{m-1}{m+1} = \frac{2m}{m+1} \quad ,$$

a contradiction.

Case 2 $n \leq m$. Then in the optimal schedule, either each processor has exactly one job or a processor is idle. In the first case, $\hat{f}_m / f^*_m = 1$, since no processor can be idle in the LPT schedule (see proof of Lemma 4.2.1). For the second case $\hat{f}^S_m / f^{*S}_m \leq \hat{f}^S_{m-1} / f^{*S}_{m-1}$ $\leq \frac{2(m-1)}{m} \leq \frac{2m}{m+1}$ by Lemma 4.2.1. Either case leads to a contadiction. ■

Corollary 4.2.1 For an m-processor system, $\hat{f}/f^* < 2$.

The bound of Theorem 4.2.1 is probably not a tight bound. However, we can show that there are examples approaching the bound 1.5 as $m \to \infty$. ■

Theorem 2.2 For every $m \geq 2$, there is an example of an m-processor system and a set of jobs S for which $\hat{f}^S / f^{*S} = c$, where c is a positive root of the equation $2s^m - s^{m-1} - \ldots - s - 2 = 0$.

Proof The example we shall construct has job set $S = (t_1 \geq t_2 \geq \ldots \geq t_m \geq t_{m+1})$ (where m is the number of processors) and processor speeds $1 = s_1 \leq \ldots \leq s_m$. The t_i's and s_i's will satisfy the following properties (see Fig. 4.2.2):

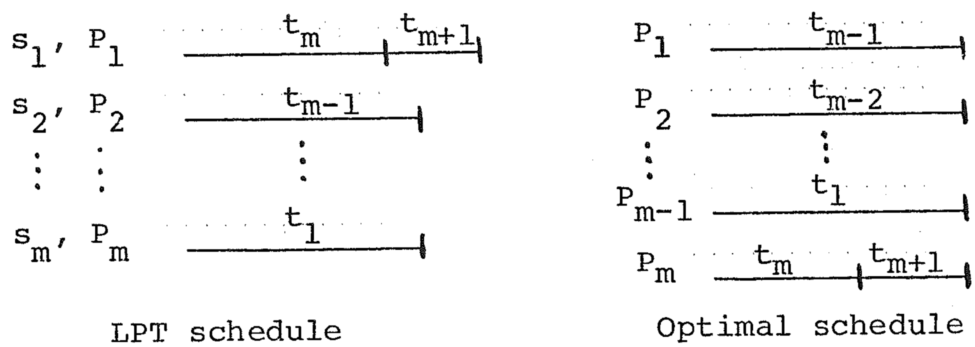

Fig. 4.2.2

(4.2.1) $\hat{f} = t_m + t_{m+1}$ and $f^* = \dfrac{t_m + t_{m+1}}{s_m}$

(4.2.2) $t_m = t_{m+1} = t$,

(4.2.3) $t_m + t_{m+1} = 2t = \dfrac{t_i + t}{s_{m-i+1}}$ for $1 \leq i \leq m-1$,

(4.2.4) $\dfrac{t_m + t_{m+1}}{s_m} = \dfrac{2t}{s_m} = \dfrac{t_i}{s_{m-i}}$ for $1 \leq i \leq m-1$.

Then $\hat{f}/f^* = \dfrac{2t}{\frac{2t}{s_m}} = s_m$. From properties (4.2.1) - (4.2.4) we can derive the equation for s_m . From (4.2.3) we get:

(4.2.5) $t_i = 2ts_{m-i+1} - t = t(2s_{m-i+1} - 1)$.

(From 4.2.4) we have

(4.2.6) $s_m t_i = 2ts_{m-i}$

(4.2.5) and (4.2.6) yields

(4.2.7) $s_{m-i+1} = \dfrac{2s_{m-i} + s_m}{2s_m}$ for $1 \leq i \leq m-1$.

Using (4.2.7) repeatedly for $i=1,2,...,m-1$ we get:

$$s_m = \frac{2s_{m-1} + s_m}{2s_m}$$

$$= \frac{2\left(\frac{2s_{m-2} + s_m}{2s_m}\right) + s_m}{2s_m}$$

$$= \frac{2s_{m-2} + s_m + s_m^2}{2s_m^2}$$

$$= \frac{2\left(\frac{2s_{m-3} + s_m}{2s_m}\right) + s_m + s_m^2}{2s_m^2}$$

$$= \frac{2s_{m-3} + s_m + s_m^2 + s_m^3}{2s_m^3}$$

$$\vdots$$

$$= \frac{2s_1 + s_m + s_m^2 + \ldots + s_m^{m-1}}{2s_m^{m-1}}$$

Hence

$$s_m = \frac{2 + s_m + s_m^2 + \ldots + s_m^{m-1}}{2s_m^{m-1}} \quad \text{(since } s_1 = 1\text{)}$$

or

$$(4.2.8) \quad 2s_m^m - s_m^{m-1} - s_m^{m-2} - \ldots - s_m - 2 = 0 .$$

The polynomial on the left hand side of (4.2.8) has one sign change and so from Descartes rule it also has one positive real root. The root must clearly be > 1 as otherwise the right hand side is < 0.

Let c be a positive root of equation (4.2.8). We can construct an example of an m-processor system with $\hat{f}/f^* = c$ by setting $s_m = c$ and computing $s_2, \ldots, s_{m-1}$ in terms of c using (4.2.7). (Of course, $s_1 = 1$.) Then by letting $t_m = t_{m+1} = t$, we can determine the