

SCHEDULING PARALLEL PROCESSORS

A Thesis

Presented to

**The Faculty of the Division
of Graduate Studies and Research**

by

Joseph D. Marsh

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy


in the School of Industrial and Systems Engineering

Georgia Institute of Technology

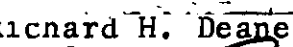
June, 1973

SCHEDULING PARALLEL PROCESSORS

Approved:



Douglas C. Montgomery, Chairman



Richard H. Deane



Fred E. Williams

Date Approved by Chairman 4/17/73

ACKNOWLEDGMENTS

Dr. Douglas C. Montgomery directed my work on the parallel processor scheduling problem. His guidance, scholarly attitude and sincere interest have been important catalysts in the conduction of this research.

Dr. Richard H. Deane and Dr. F. E. Williams contributed unselfishly and conscientiously to this investigation, providing direction and guidance which could only have resulted from many extra hours of work for them.

Dr. R. Gary Parker reviewed the manuscript and provided many helpful suggestions and comments. Special thanks are due to him.

Dr. Lynwood A. Johnson participated in the final oral examination. In addition, his lectures in scheduling theory and operations research techniques for production planning were an enlightening part of my doctoral coursework and were major factors in my decision to undertake the scheduling research which resulted in this dissertation.

The help provided by Mrs. Sharon Butler in preparing the manuscript is greatly appreciated.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS.	ii
LIST OF TABLES	v
LIST OF ILLUSTRATIONS.	vii
SUMMARY.	ix
Chapter	
I. INTRODUCTION.	1
Problem Description	
Programming Approaches	
Purpose and Objectives	
Scope and Limitations	
II. LITERATURE SURVEY	25
Results for Scheduling Parallel Processors	
Single Processor Results	
Results from Conceptually Similar Problems	
III. SCHEDULING JOBS WITH ALL INFINITE DUE DATES ON IDENTICAL PARALLEL PROCESSORS.	31
Formulation as an Augmented Traveling Salesman Problem	
Branch and Bound Algorithm Development	
Branch and Bound Algorithm	
Illustrative Problems	
IV. SCHEDULING JOBS WITH ALL INFINITE DUE DATES ON DISTINCT PARALLEL PROCESSORS	55
Extensions of Identical Processor	
Algorithm Components	
Branch and Bound Algorithm	
Illustrative Problems	
V. SCHEDULING JOBS WITH SOME FINITE DUE DATES. . .	77
Development of Feasibility Conditions	
Modification of Solution Procedure	
Illustrative Problems	

Chapter	Page
VI. COMPUTATIONAL EXPERIENCE.	97
Results for Distinct Processor Problems	
Results for Problems with Due Dates	
Results for Identical Processor Problems	
VII. HEURISTIC PROCEDURES FOR SCHEDULING PARALLEL PROCESSORS.	108
The Heuristic Procedures	
Computational Experience	
VIII. CONCLUSIONS AND RECOMMENDATIONS	125
Conclusions	
Recommendations	
Appendix	
A. FORTRAN V CODE FOR OPTIMAL, MAXIMUM REGRET AND MAXIMUM REGRET WITH LOOK AHEAD SCHEDULING .	130
B. COMPUTING TIMES FOR EXPERIMENTS WITH THE EXACT ALGORITHM	149
C. FORTRAN V CODE FOR RANDOM SCHEDULING.	155
D. FORTRAN V CODE FOR SHORTEST CHANGEOVER NEXT OR MINIMUM TIME SUBSEQUENCE SCHEDULING.	157
BIBLIOGRAPHY	162
VITA	166

LIST OF TABLES

Table		Page
1	Least Squares Estimates of a_N and b_N	100
2	Computing Times for (a) Problems with Discrete Uniform [0,20] Changeover Times, (b) Problems Where $N'=N$ Compared to (c) Mean Computing Times for Previous Problems	102
3	Computing Times for Identical Processor Problems Where (a) $N'=N$, (b) N^* is to be Determined and (c) Changeover Times are Discrete Uniform [0,20]	107
4	Comparison of Heuristic Solutions to Optimal Solutions of Distinct Processor Problems Where N^* is to be Determined and Where $c_{ijn} \sim U [0,10]$	115
5	Comparison of Heuristic Solutions to Optimal Solutions of Identical Processor Problems Where N^* is to be Determined and Where $c_{ijn} \sim U [0,10]$	116
6	Comparison of Heuristic Solutions to Estimated Total Changeover Time Distribution for Selected Large Problems	122
7	Heuristic Solutions to Selected Large Problems, Expressed in Deviations Below Estimated Mean Total Changeover Time.	123
8	Computing Times for Distinct Processor Problems Where $N=2$	150
9	Computing Times for Distinct Processor Problems Where $N=3$	151
10	Computing Times for Distinct Processor Problems Where $N=4$	152
11	Computing Times for Distinct Processor Problems Where $N=2$ and Where Due Dates are Moderately Constraining	153

Table		Page
12	Computing Times for Distinct Processor Problems Where $N=2$ and Where Due Dates are Highly Constraining.	154

LIST OF ILLUSTRATIONS

Figure		Page
1	Tree Representation of the Solution to the Identical Processor Example Where N' is Specified.	51
2	Tree Representation of the Solution to the Identical Processor Example Where N^* is to be Determined.	54
3	Tree Representation of the Solution to the Distinct Processor Example Where N' is Specified.	72
4	Tree Representation of the Solution to the Distinct Processor Example Where N^* is to be Determined.	76
5	Typical Relationships Between Time Data When Processing on a Job Begins After the Previous Job's Due Date	82
6	Typical Relationships Between Time Data When Processing on a Job Must Begin Before the Previous Job's Due Date.	82
7	Tree Representation of the Solution to the Distinct Processor Example with Moderately Constraining Due Dates	89
8	Partial Schedule $\{(7,5), (5,4), (4,2), (2,3)\}$ of S_2	91
9	Tree Representation of the Solution to the Distinct Processor Example with Restrictive Due Dates.	93
10	Partial Schedule $\{(5,2), (2,1)\}$ of S_2	96
11	Relationships Between Time Data for S_2 Illustrating Infeasibility	96

Figure		Page
12	Average Computing Time and Least Squares Lines for Distinct Processor Problems.	99
13	Average Computing Time for Distinct Processor Problems Where $N=2$ for (a) Infinite Due Dates, (b) Moderately Constraining Due Dates and (c) Highly Constraining Due Dates.	105
14	Computing Time for Alternative Scheduling Procedures for Distinct Processor Problems Where $N=3$, N^* is to be Determined, and $c_{ijn} \sim U[0,10]$	118
15	Distributions of Sampled Total Changeover Times.	120
16	Computing Times for Heuristic Solutions to Selected Large Problems.	124

SUMMARY

This investigation treats the problem of scheduling M batch-type jobs which have sequence-dependent changeover times but which are otherwise independent on N parallel processors. In general, it is assumed that the sequence-dependent changeover times are not identical for each processor, each job is available at some arbitrary time zero and deadlines or due dates may be imposed on the jobs. Each job is to be processed by exactly one of the N available parallel processors. The criterion is the minimization of total changeover time subject to the constraint that all due dates must be met. In the absence of job due dates, the criterion is to minimize total changeover time.

The solution to the parallel processor problem involves partitioning the M jobs into N or fewer distinct subsets while simultaneously determining the processing sequence within each subset. Two possible assumptions are admitted with respect to the number of partitions (processors) depending on whether this number is specified or is a decision variable.

Four programming approaches are investigated and only combinatorial programming and heuristic programming are found to be computationally feasible for problems of realistic size. It is shown that the special case where the processors are identical and where all job due dates are infinite can

be formulated as a traveling salesman problem. However, this approach fails to extend to any more generalized cases. A branch and bound algorithm which can be extended is developed for the identical processor problem where all job due dates are infinite and where the number of processors to be activated can either be specified or can be a decision variable. The algorithm is subsequently extended to admit distinct processors and jobs with due dates.

Computational experience was concentrated on distinct processor problems where N^* is to be determined and where changeover times are discrete uniform $[0, 10]$. The average computing time t_{MN} in minutes for this class of problems increases exponentially with M and N and is adequately described by

$$\hat{t}_{MN} = e^{-9.7752} (1.7480)^M (2.4600)^N.$$

Additional computational experiments included distinct processor problems under two classes of due dates, distinct processor problems under certain alternative assumptions and some identical processor problems. The computational results suggest that many moderately-sized problems are computationally infeasible.

In view of this, several heuristic procedures are developed to solve the parallel processor scheduling problem. The heuristic procedures were evaluated by comparing their

solutions for small problems to the optimal solutions found by the optimal algorithm. Larger problems were also solved heuristically and these solutions were evaluated by approximating the distributions of total changeover time for selected large problems and making comparisons based on these distributions.

Some ideas for extending the above results to include certain alternative criteria and assumptions are given.

CHAPTER I

INTRODUCTION

Scheduling research to date has been directed toward the solution of many somewhat distinct problems related to the order of processing jobs on machines. There exists a viable theory of scheduling which allows the determination of optimal schedules under various alternative criteria, constraints and assumptions. For example, optimal scheduling algorithms are known which minimize either mean flow time, maximum job lateness or total machine setup costs under certain constraints and assumptions [1].

However, there exists a well-known lack of practical application of these research results [2, 3]. One possible reason for this is that existing scheduling algorithms solve problems that are rarely found in industrial scheduling environments [2]. Another possible reason is that existing algorithms are too difficult and/or too expensive to apply in practice [3].

At least two recent surveys of companies throughout the United States support these observations [4, 5]. The survey findings provided the following suggested reasons for the lack of application of scheduling research to existing scheduling problems. First, the number of jobs scheduled at

any time usually exceeds the capability of most scheduling algorithms. Only 19% of the survey responses indicated problem sizes with no more than 10 jobs and 10 machines. Therefore, many scheduling algorithms, while perhaps approaching the correct problem, are incapable of handling the prevalent problem size.

Second, almost every company surveyed had both primary and secondary criteria while most scheduling criteria attempt to optimize a single measure of effectiveness. Most companies considered the meeting of due dates to be the most important criterion. The most common secondary objective was to minimize changeover times for which 48% of the companies said were sequence-dependent for more than half of their operations.

Third, at least 81% of the respondents indicated that several machines are available to perform similar work with about 60% of the replies indicating that the machines were of a different type. Most existing scheduling research results relates to single machine scheduling. (See [1] for example.)

The general purpose of the research reported herein is to develop solution procedures for several cases of the scheduling problem involving parallel processors. The criterion is the minimization of total changeover time subject to the constraint that all due dates must be met.

Problem Description

Specifically, the class of scheduling problems treated in this investigation involves scheduling the M batch-type jobs in job set M on the N parallel processors in machine set N . The M jobs have sequence-dependent changeover times, but are otherwise independent. This parallel processor problem is the generalization of the problem of sequencing a set of jobs with sequence-dependent setup times on a single machine. In the absence of due dates the single processor problem is the archetypal combinatorial optimization problem best known as the traveling salesman problem. The parallel processor problem under investigation could best be described as a multi-salesman traveling salesman problem where the due dates impose latest arrival time constraints.

Assumptions and Definitions

Scheduling is used synonymously with sequencing since the problem is one of partitioning the jobs into N (or fewer) distinct subsets while (simultaneously) determining the optimal processing sequence within each subset and insuring that the due date of each job is met. The intermediate or long-range smoothing of workforce and production levels is therefore not a consideration in the scheduling procedures developed in this study.

Independent jobs mean that there exist no precedence or technological constraints which require some jobs to be completed before others begin. Precedence constraints would

exist, for example, when the jobs are actually elements of an overall process where some elements are prerequisites for others.

Sequence-dependent changeover (or setup) times refers to the case where the time required for the changeover (i,j) from job i to job j depends both on i and j . In addition, changeover time dependence on processor n is assumed. The time required for changeover (i,j) on processor n is denoted c_{ijn} and is assumed to be given in an array $\underline{C} = \{c_{ijn}\}$. Each c_{ijn} is assumed to be deterministic. The special case where $c_{ijn} = c_{ijw}$ for all $i,j \in \underline{M}$ and all processors $n,w \in \underline{N}$ is called the identical processor problem. Since the changeover times for identical processors depend only on i and j , they can be given in a single changeover time matrix $\underline{C} = \{c_{ij}\}$. The general case which exists if some $c_{ijn} \neq c_{iju}$ is called the distinct processor problem.

Each job is assumed to be available at some arbitrary time zero and is to be processed exactly once by one of the N available processors. This implies that $c_{iin} = \infty$ and that no preemption is allowed.

Associated with each job j is a deadline or due date d_j measured from arbitrary time zero. The special case where no job has a due date is indicated by letting $d_j = \infty$, $\forall j \in \underline{M}$. Due dates are said to exist if any $d_j < \infty$. When the d_j are so restrictive that all possible schedules involve late jobs, there is said to exist no feasible schedule.

The processing time required for job j on processor n is denoted p_{jn} .

For a given processor, there is typically some initial (possibly idle) state and a cost is involved when changing from the initial state to the state required for processing the first job. Similarly, there is usually some final (possibly idle) processor state required after completing the schedule, and reaching this final state involves a cost which depends on the last job in the schedule. If M jobs are to be scheduled on N parallel processors, the work required in going from the initial state of processor n to the first job on processor n is defined as initial job $M + n$. Similarly, final job $M + N + n$ is defined as the work required in going from the last job on processor n 's schedule to the final state.

Notationally the original jobs are numbered $1, 2, \dots, M$; the initial jobs are numbered $M + 1, M + 2, \dots, M + N$; and the final jobs are numbered $M + N + 1, M + N + 2, \dots, M + 2N$. The time $c_{M+n,j,n}$ required to bring processor n from its initial state to the state required for processing any job j is assumed to be given and it is assumed that $c_{M+n,M+N+n,n} = 0$. It is noted that $c_{M+n,j,n} = c_{M+u,j,u}$ for any job $j \in \underline{M}$ for any processors n and u if the processors are identical. The analogous assumption is made for the time $c_{j,M+N+n,n}$ required in going from any job j to the final state on processor n .

A schedule \tilde{S}_n of M' real jobs on the n th processor may be represented by the vector of $M' + 1$ ordered pairs

$$\tilde{S}_n = [(M + n, i_{1,n}), (i_{1,n}, i_{2,n}), \dots, (i_{M',n}, M + N + n)],$$

where $i_{j,n}$ denotes the $(j + 1)$ st job in processor n 's schedule. A schedule $\tilde{S}_n = [(M + n, M + N + n)]$ indicates that processor n is not activated. A schedule \tilde{S}_n is admissible if (a) initial job $M + n$ is first, (b) final job $M + N + n$ is last, (c) all other jobs precede exactly one other job, and (d) each job is processed exactly once. A feasible schedule \tilde{S}_n is an admissible schedule in which all due dates are met.

A parallel processor schedule is a listing of single processor schedules

$$\tilde{S} = [\tilde{S}_1, \tilde{S}_2, \dots, \tilde{S}_N]$$

which includes each real job exactly once.

There are N (possibly distinct) available processors. The most general assumption is that the scheduling algorithm is to determine the optimal number of processors, $N^* \leq N$. When the number of processors to be used in the final solution is specified in advance, for example by management policy, that number will be denoted by N' . Obviously $N' \leq N$.

An exact algorithm is defined to be a solution

procedure which generates a schedule in a finite number of steps which optimizes some measure of scheduling performance. A heuristic algorithm is a solution method which generates a schedule in a finite number of steps which is in general suboptimal, although hopefully near-optimal.

Problem Statement

The problem involves finding, over all feasible schedules S , that schedule S^* for the M jobs on a (possibly given) subset of processors $\tilde{N} \subseteq \underline{N}$ which minimizes

$$\sum_{n \in \tilde{N}} \sum_{(i,j) \in S_n} c_{ijn}$$

subject to

$$\sum_{k=1}^r \left\{ p_{i_{k,n},n} + c_{(i_{k-1,n}), (i_{k,n}), n} \right\} \leq d_{i_{r,n}};$$

$$r = 1, \dots, R_n \quad \forall n \in \tilde{N},$$

where R_n is the number of real jobs scheduled on processor n and where $i_{0,n} = M + n$.

If all $d_j = \infty$, then the constraint is automatically satisfied and the problem is a traveling salesman problem with N available salesmen.

Size of the Solution Space

The number of possible solutions is frequently an indicator of the degree of difficulty encountered in solving combinatorial problems. Suppose that M jobs are to be sequenced on a single processor, say n , so that an admissible schedule is

$$S_n = \{(M + n, i_1), (i_1, i_2), \dots, (i_{M-1}, i_M), (i_M, M + N + n)\}.$$

There are exactly $M!$ such admissible single processor sequences since the M original jobs can be ordered in $M!$ ways between the initial and final jobs. (Note that the initial and final jobs do not affect the number of admissible sequences.) If each of the M jobs is to be sequenced on exactly one of N available processors, the number of admissible solutions depends on whether the number of processors to be activated is specified and whether the processors are identical or distinct.

N' Given. Suppose the M jobs are to be sequenced on N available processors in such a way that exactly N' processors are activated. First, assume that the processors are distinct. Then single processor schedules can be permuted to yield different schedules. For example, for $M=3$, $N'=2$

$$S = \{S_1; S_2\} = \{(4,1), (1,6); (5,2), (2,3), (3,7)\}$$

and

$$S' = \{S_1; S_2\} = \{(4,2), (2,3), (3,6); (5,1), (1,7)\}$$

are different schedules.

Taking this into consideration, consider a given permutation i_1, i_2, \dots, i_M of the M jobs. The initial and final jobs on the N processors cannot affect the number of admissible solutions. To partition the given permutation into N' subsets is the same as selecting $N' - 1$ spaces of the $M - 1$ spaces between jobs in the given permutation. This can be done in $\binom{M-1}{N'-1}$ ways. Since this partitioning results in a unique parallel processor schedule for each permutation of the M jobs, there are $\binom{M-1}{N'-1} M!$ admissible solutions, given the N' processors to be activated. There are $\binom{N}{N'}$ ways of selecting the processors, so that the total number of admissible solutions is

$$\binom{N}{N'} \binom{M-1}{N'-1} M!.$$

Note that this always exceeds the total number of single processor schedules by a factor of $\binom{N}{N'} \binom{M-1}{N'-1}$.

If the processors are identical, any N' processors are the same as any other N' processors. In addition, there are exactly $N'!$ permutations of single processor schedules.

Therefore, there are exactly

$$\binom{M}{N' - 1} \frac{M!}{N'!}$$

admissible identical processor schedules which activate exactly N' processors. This number can either be less than, greater than, or equal to $M!$ depending on whether

$$\binom{M}{N' - 1} \frac{1}{N'!} \begin{matrix} < \\ > \end{matrix} 1.$$

N^* To Be Determined. When the number of processors to be activated is unspecified, all but one processor may have zero jobs scheduled. Assume that the processors are distinct. Under these assumptions, a unique parallel processor schedule can be constructed by taking a given permutation i_1, i_2, \dots, i_M and placing a slash before i_1 , a slash after i_M and $N - 1$ slashes in the now $M + 1$ spaces between any combination of slashes and jobs. The number of ways of partitioning a given permutation in this manner is $\binom{N + M - 1}{M}$, the number of ways of selecting M places out of $N + M - 1$ places. Since this can be done for each permutation, there are $\binom{N + M - 1}{M} M!$ admissible distinct processor schedules. This number is always larger than $M!$

Consider the case where the processors are identical and N^* is to be determined. Permutations of single processor schedules do not yield different parallel processor

schedules. There are $N!$ permutations of an n -tuple. But some permutations result in the same parallel processor schedule. Hence there are at most

$$\binom{N + M - 1}{M} \frac{M!}{N!}$$

different identical processor schedules when N^* is to be determined.

The number of feasible schedules in the solution space is a subset of the number of admissible solutions. This number depends on the nature of the due dates.

Programming Approaches

There are at least four programming approaches to solving the problem. These approaches are described as follows.

Integer Programming

An integer programming formulation exists for the most general parallel machine scheduling problem where there exists at least one finite due date. Alternative formulations are possible for either the case where the number of processors in the final solution is not known in advance (N^* is determined by the solution procedure) or the case where the number of processors $N' \leq N$ is specified.

Consider first the case where N^* is unknown and where each processor is identical. Define activity A_i , $i = 1, \dots$,

I , as an N -dimensional column vector of 0,1 constants a_{ij} where

$$a_{ij} = \begin{cases} 1 & \text{if job } j \text{ is to be included in schedule } i \\ 0 & \text{otherwise} \end{cases}$$

An activity may be thought of as a potential schedule for one of the N available processors which satisfies all feasibility (due date) constraints. The index i ranges over the set $\{1, 2, \dots, I\}$ of all feasible single processor schedules.

Let the cost of activity i be C_i^* . That is, C_i^* is the total cost of the changeovers required by the jobs in feasible single processor schedule i . However, this cost is sequence-dependent and therefore C_i^* is the optimal solution to a traveling salesman problem with constraints on due dates and initial-final jobs. An exact algorithm exists for the determination of C_i^* [6].

The integer programming problem is to minimize

$$\sum_{i=1}^I C_i^* x_i \quad (I-1)$$

subject to

$$\sum_{i=1}^I x_i a_{ij} = 1 \quad j = 1, \dots, M \quad (I-2)$$

$$\sum_{i=1}^I x_i \leq N \quad (I-3)$$

$$x_i = 0, 1. \quad (I-4)$$

where M = total number of jobs to be scheduled.

The first M constraints (I-2) insure that each job will be in exactly one single processor schedule. Constraint (I-3) insures that $N^* \leq N$. Note that N^* is the number of activities in the optimal basis.

Unfortunately, for even small values of M , the total number of activities, I , can become very large. A precise estimate of I is impossible because its magnitude depends on the number of activities which are infeasible because of due dates. An upper bound on I is $\binom{M}{1} + \binom{M}{2} + \dots + \binom{M}{M} = 2^M - 1$ which is reached when no due dates are constraining with respect to feasible activities. For example, if $M = 50$ then $2^M - 1 > 10^6$ and even if only a small subset of this maximum number of single processor schedules are due date feasible the resulting integer program is computationally infeasible [7].

It is possible to slightly improve the computational aspects of the integer programming approach. If constraint (I-3) is ignored, the resulting integer program is actually the well-known set partitioning problem (a special set covering problem where all constraints are equalities) which has

been treated by enumerative algorithms [8, 9], dynamic programming [10] and combinatorial programming [11]. It appears that partitioning problems of up to several hundred integer variables may be solved in less than 15 minutes using the algorithms in [8, 9, 11]. The solution scheme is as follows.

Step 1. Generate the set of all $2^M - 1$ subsets of the M jobs. Eliminate those subsets which would be due date infeasible regardless of the order of processing. At this point I subsets remain.

Step 2. Solve I single processor sequencing problems to determine the C_i^* . If the jobs have all infinite due dates the problems are classical traveling salesman problems. If at least one due date is finite the problems are modified traveling salesman problems.

Step 3. Solve the set partitioning problem which results by ignoring constraint (I-3) in the integer program. If the number of activities N^* in the optimal basis is less than or equal to N , an optimal feasible solution has been determined. Otherwise, go to Step 4.

Step 4. Find the minimum number of processors \tilde{N} which will yield a feasible solution. Solve $N - \tilde{N} + 1$ integer programs identical in objective function (I-1) and constraints (I-2) but changing constraint (I-3) to

$$\sum_{i=1}^I x_i \leq \hat{N}. \quad (I-5)$$

The $N - \tilde{N} + 1$ programs would be solved with $\hat{N} = \tilde{N}, \tilde{N} + 1, \dots, N$. The solution with the lowest value for the objective function is optimal.

This scheme slightly extends the computational power of the integer programming approach. However, each step in the scheme can become computationally infeasible with moderate values of M . The simplest step, Step 1, involves a significant number of numerical operations since each of the $2^M - 1$ subsets must be checked against due date constraints. The feasibility of Step 2 depends on the efficiency of algorithms of the traveling salesman type. Perhaps the best algorithm for jobs with all infinite due dates is the Little, et al. [12] procedure which has solved a 40-salesman (job) asymmetric problem in less than nine minutes (IBM 7090). If at least one job has a finite due date the only existing algorithm for Step 2 is that of Pierce and Hatfield [6] which has solved problems of 20 jobs in less than eight minutes (IBM 7094). The feasibility of Step 4 is obviously limited by the present state of the art in integer programming algorithms [7].

The above integer programming formulation considers the case where N^* is unknown. If the number of processors to be used is specified in advance to be $N' \leq N$, an analogous formulation results with original constraint (I-3) being replaced by

$$\sum_{i=1}^I x_i = N' \quad (I-6)$$

to insure that exactly N' activities (processors) will be in the final solution.

If the processors are not identical, the formulation is essentially the same except that for each subset of the $2^M - 1$ possible subsets a distinct activity must be defined. That is, an upper bound on I is now $N (2^M - 1)$.

Dynamic Programming

It is possible to use dynamic programming as the solution technique in the development of an exact algorithm for at least some cases of the parallel processor scheduling problem. The procedure given below is basically a variation of an algorithm of Held and Karp [13] which was developed to minimize other cost criteria in scheduling parallel processors. The case of identical processors and unknown number of final processors $N^* \leq N$ will be considered.

Consider first the problem of optimally scheduling m_n real jobs with all infinite due dates on a single processor n . This is basically finding an optimal sequence which begins at job $M + n$, executes $i_{1,n}, \dots, i_{m_n,n}$ and ends at job $M + N + n$. Let s be a subset of $\{1, \dots, m_n\}$ and let $f_n(s, g)$ be the minimum cost of a subsequence on processor n which begins with dummy initial job $M + n$ and terminates with job g , $g \neq M + N + 1$. This implies that the subsequence

terminates on some real job. Also let $n(s)$ denote the number of jobs in s and let $s - g$ denote the set which results from deleting job g from s , $g \in s$.

Then for

$$n(s)=1, \quad f_n(\{g\},g) = c_{M+n,g,n} \quad \text{for any } g \quad (I-7)$$

$$n(s)>1, \quad f_n(s,g) = \min_{r \in s-g} [f_n(s-g,r) + c_{rgn}]. \quad (I-8)$$

Equation (I-8) follows from the following considerations. Suppose that in executing the jobs in s , job r immediately precedes job g . Then, assuming that the other jobs are optimally sequenced, the cost incurred is $f_n(s-g,r) + c_{rgn}$. Taking the minimum over all choices of r yields (I-8).

If K_n denotes the minimum cost of a complete schedule which begins with job $M + n$, executes m_n jobs and ends with job $M + N + n$, then

$$K_n = \min_{g \in \{1, \dots, m_n\}} [f_n\{1, \dots, m_n\}, g) + c_{g, M+N+n, n}] \quad (I-9)$$

A schedule

$$S_n = [(M+n, i_{1,n}), (i_{1,n}, i_{2,n}), \dots, (i_{m_n, n}, M+n+n)]$$

is optimum if and only if

$$K_n = f(\{i_{1,n}, \dots, i_{m_n,n}\}, i_{m_n,n}) + c(i_{m_n,n}, (M+N+n), n) \quad (I-10)$$

and for $2 \leq p \leq m_n - 1$

$$f(\{i_{1,n}, \dots, i_{p,n}, i_{p+1,n}\}, i_{p+1,n}) = f(\{i_{1,n}, \dots, i_{p,n}\}, i_{p,n}) + c(i_{p,n}, (i_{p+1,n}), n) \quad (I-11)$$

The optimal single processor schedule is computed as follows. The quantities $f_n(s, g)$ are computed recursively from (I-7) and (I-8). K_n is computed from (I-9). Then (I-10) and (I-11) are used to compute the optimal schedule where $i_{m_n,n}$ is determined first, and the $i_{m_n-1,n}, \dots, i_{1,n}$ successively.

The following parallel processor solution procedure uses the single processor formulation.

Step 1. Generate the set of all 2^{M-1} subsets of the M jobs. Call the resulting subsets $S^{(1)}, S^{(2)}, \dots, S^{(I)}$.

Step 2. Recursively compute $f_n(S^{(i)}, g)$ using (I-6) and (I-8) and compute K_n for each subset $S^{(i)}$ from equation (I-9).

Step 3. Solve a set-partitioning problem to assign the jobs to processors in such a way as to minimize

$$\sum_{n=1}^N K_n$$

Call this minimum cost partition $T^{(1)}, \dots, T^{(v)}$.

Step 4. For each $T^{(i)}$ compute the optimum schedule using (I-10) and (I-11).

The dynamic programming formulation has a structure which is similar to the integer programming approach. Step 1 essentially enumerates all potential schedules, eliminating those which are a priori infeasible. Step 2 computes costs on these schedules. Step 3 determines which schedules should appear in the final solution and Step 4 determines the overall optimal schedule.

The dynamic programming formulation therefore has many of the same computational disadvantages as the integer programming formulation. Additionally, the computer storage requirements inherent in any recursive procedure become excessive for reasonably-sized problems.

It is also possible, at least conceptually, to consider the case of finite due dates by introducing a method in Step 1 to eliminate those schedules which are obviously due date infeasible and by introducing due date constraints in Step 2 and Step 4. However, the modifications add a significant number of additional numerical operations and add greatly increased computer storage requirements to an already computationally infeasible solution procedure.

Branch and Bound Methods

One of the most successful exact solution procedures for solving combinatorial problems similar to scheduling

parallel processors is the branch and bound method. Given the problem of minimizing an objective function $f(x)$ subject to $x \in F$, the branch and bound procedure partitions the feasible region F into finer and finer subsets while computing a lower bound for each subset on the value which $f(x)$ may obtain.

Branch and bound schemes are frequently enhanced by imbedding the basic optimization problem in a larger, less restrictive problem by introducing a non-empty superset T , $F \subset T$ along with a bounded extension g of the objective function f with the requirements that (a) $g(x) = f(x)$ whenever $x \in F$ and (b) there exists an $x \in T$ such that $g(x) = f(x^*)$, where x^* is the optimal solution to the original problem.

A branch and bound algorithm must include a procedure to identify infeasible solutions, a partitioning (branching) scheme, bounding rules, and a recursive operation for forming new collections of subsets, excluding those elements which are known to be either infeasible or suboptimal.

More specifically, if Y denotes the set of all subsets of T , if \underline{T} denotes the set of all collections of subsets \underline{t} of \underline{T} and if the union of all subsets in any collection \underline{t} is denoted by $U(\underline{t})$, then these requirements may be stated as follows.

Feasibility Test. The algorithm must specify a collection \underline{t}_0 with the following properties

- (i) The elements of \underline{t}_0 contain only infeasible

solutions. That is $\cup(t_0) \subset T-F$;

- (ii) All singleton infeasible subsets $\{x\}$ are included in t_0 . That is, if $x \in T-F$, then $\{x\} \in t_0$.

One procedure for satisfying the requirement of a feasibility test is to specify a procedure to identify all singleton infeasible subsets $\{x\}$. Computation efficiency is enhanced by identifying larger infeasible subsets.

Partitioning Scheme. A partitioning, or branching, scheme is a function $p: \underline{T} \rightarrow \underline{T}$ such that

- (i) $\cup [p(t)] = (t)$;
- (ii) $T_i' \in p(t)$ only if $T_i' \subset T_i \in t$; and
- (iii) $p(t) = t$ if and only if all $T_i \in t$ are singleton subsets

Conditions (i) and (ii) state that the partitioning scheme cannot add any elements to the partitioned subsets. Condition (iii) states that the partitioning scheme must divide at least one divisible subset into proper subsets.

Bounding Rules. The algorithm must specify a lower bound on the value of $g(x)$ for any subset T_i .

The lower bounding rule is a function $b: \underline{T} \rightarrow \underline{R}$

- (i) $g(x) \geq b(T_i)$ for all $x \in T_i \subset T$;
- (ii) $b(T_i') \geq b(T_i)$ if $T_i' \subset T_i \subset T$; and
- (iii) $b(\{x\}) = g(x)$

Condition (ii) states that deleting points from subsets does not lead to lower upper bounds. Condition (iii) states that the lower bound on the cost of any solution in a

singleton subset is in fact the cost of that solution.

Recursive Operation. The branch and bound recursive operation is a function B such that if $p(\underline{t}') = \underline{t}$, then $B(\underline{t}') = \underline{t} - \underline{t}^-$, where \underline{t}^- is that subcollection of \underline{t} whose elements are known to be either dominated or infeasible.

Branch and bound algorithms have been developed to optimally schedule a moderate number of jobs with some finite due dates on a single processor [6]. The development of branch and bound algorithms to optimally schedule parallel processors is one objective of the present research.

Heuristic Programming

Approximate, or heuristic, procedures are useful in many scheduling environments. When the feasible set contains a number of solutions with insignificant differences in the objective function, then exact procedures frequently become inefficient in this near-optimal region compared to the ultimate payoff of strict optimality. Also, known exact procedures require a number of iterations which grows approximately exponentially with increasing size for combinatorial problems.

Heuristic algorithms can be classified as being either (1) exact algorithms which have been modified so that an optimum can no longer be guaranteed, or even expected and (2) approximate algorithms which do not depend on any exact algorithm.

A consideration of both classes of heuristic algorithms

with respect to parallel processor scheduling and the development of and experimentation with some heuristic algorithms to handle realistically-sized parallel processor problems is the basis of some of the present research.

Purpose and Objectives

The overall purpose of this study is to develop computationally feasible algorithms for scheduling parallel processors for a number of cases. The following specific objectives are delineated:

1. To develop exact algorithms for scheduling parallel processors under alternative assumptions, each of which, when carried to completion, guarantees an optimal solution if one exists.
2. To evaluate these exact algorithms with respect to computational limitations.
3. To develop efficient heuristic algorithms which will provide good quality solutions to the class of problems for which exact procedures are inefficient.
4. To evaluate the heuristic algorithms with respect to computational efficiency and quality of solution.

Scope and Limitations

The scope of this research is that of the short-term scheduling function relating to parallel processors where the single criterion is the minimization of total changeover time. Limitations include the consideration of

only the static case, where all jobs and their scheduling parameters are known at some arbitrary time zero. No stochastic elements are admitted in the scheduling parameters.

CHAPTER II

LITERATURE SURVEY

Results for Scheduling Parallel Processors

There is a growing literature on the problem of scheduling parallel processors. Unfortunately, most of it treats special cases whose restrictive assumptions include sequence-independent setup times in order to make the problem tractable.

An early paper by Hu [13] considered the case of dependent jobs where all setup times are zero (and therefore sequence-independent) and where all processing times are equal. These assumptions allowed the development of network algorithms to either minimize the number of processors required to complete all the jobs by a given time or to minimize the completion time of all jobs given a prescribed number of machines.

McNaughton [14] treats the problem of scheduling independent jobs with sequence independent setup times on parallel processors in order to minimize the sum of linear losses $f_i(t_i)$ which job i accrues if it exceeds its deadline by t_i time units. Important results in McNaughton's paper are theorems on lot-splitting. He shows that for the objective of minimizing total loss as defined above with a single

processor, an optimal solution exists in which no task is split. He notes that for the case of parallel processors which are not identical in capacity that the general optimal solution will contain split jobs. However, he shows that under the assumption that all deadlines are zero with losses linear in time there always exists a non-split schedule with total loss no greater than any given split schedule. His results include an algorithm for scheduling parallel processors of unequal capacities but no algorithm could be developed for the case of identical processors.

Noting that McNaughton's [14] paper yielded no identical processor scheduling algorithm, Eastman, Even, and Isaacs [15] computed upper and lower bounds on the cost of the optimal solution under the same assumptions.

Lawler [16] treated the problem of scheduling independent jobs with sequence-independent setup times on identical processors to minimize total deferral cost which is the sum of nonlinear job deferral costs $g_i(c_i)$ which is assumed to be monotonically nondecreasing with the time of completion c_i of job i . Lawler shows that the transportation method of linear programming can be used to schedule identical parallel processors when the processing times for the jobs are equal. For the case of unequal processing times an approximate solution is given based on a scheme where each job i is divided into a number a_i of "unit time" jobs.

Under the same assumptions, including sequence-

independent setup times. Root [17] developed an algorithm to optimally schedule identical parallel processors when all jobs have a common deadline d and have identical loss functions $f_i(c_i - d) = \max [0, b(c_i - d)]$, $b > 0$, where c_i is the completion time for job i .

Rothkopf [18, 19] treats the problem of scheduling identical parallel processors under the same assumptions of sequence-independent setup times but allowing job waiting (loss) functions $g_i(c_i)$ to be any function monotonic and non-decreasing in c_i , the completion time for the i th job. A dynamic programming algorithm is formulated to minimize the sum of discounted waiting costs, use costs B_{ij} if job i is processed by processor j , and costs $G_0(t_1, t_2, \dots, t_k)$ associated with a schedule in which processor j completes the jobs assigned to it at time t_j , where G_0 is a monotonic nondecreasing function of each of its arguments. McNaughton's [14] multiprocessor splitting theorem is extended to the case where jobs have waiting cost functions of the form $h_i e^{-rt}$, $r > 0$.

All of the studies referenced make the restrictive assumption of sequence-independent setup times. When this assumption is relaxed, the approach of minimizing the total loss (waiting, deferral) from all jobs becomes invalid. The approach of minimizing total loss also implicitly assumes that all job requirements remain the same whether or not the deadline is met.

An equally restrictive approach with respect to parallel processors is the variation of the classical economic lot size formula to schedule jobs with sequence-independent setup times and deadlines. This method was first proposed by Cox and Jessop [20] and recently has been revived by Elmaghraby [21, 22].

The above examination of the literature shows that the problem of sequence-dependent setup times has been neglected. The reason for this is that the sequence-dependent assumption introduces complexities which seem to prohibit the discovery of a straight-forward solution such as sequencing on some job parameter such as processing time or due date. The parallel processor results to date therefore are at present unextendable to the sequence-dependent case.

This conclusion suggests the alternate approaches to the literature of investigating single processor results which show promise of being significant in developing parallel processor algorithms and examining problems conceptually similar to parallel processor scheduling problems. These alternate approaches are discussed below.

Single Processor Results

Pierce and Hatfield [6] have developed a branch and bound algorithm for scheduling jobs on a single processor which provides an exact solution. The computational feasibility of their single processor algorithm is limited, the

largest problem solved being one of scheduling 30 jobs on a single processor. They assert that the single processor algorithm is computationally feasible up to 20 jobs. However, suggestions for extending the already limited branch and bound approach to the case of parallel processors were made. Many of the concepts developed in the present research evolved from suggestions in [6].

Results from Conceptually Similar Problems

A well-known problem which is similar to the problem of scheduling parallel processors is the delivery or routing problem. Basically, the delivery problem is concerned with the transportation of products from one set of locations to another set of locations under vehicle capacity and other constraints which govern the nature of the routings.

There are several assumptions about the delivery problem which affect the structure of the problem. However, two assumptions characterize those formulations important to parallel scheduling. One is the assumption that the route is not fixed and that the total distance traveled is sequence-dependent. The other is the assumption of several vehicles to satisfy known demands of customers at various locations.

Given the above assumptions, the traditional criterion in the delivery problem is to minimize the distance traveled. When several trucks are involved, the analogy with the parallel processor problem is clear.

The principal methods proposed for the solution of the delivery problem have been simulation [23, 24, 25, 26], integer programming [27], dynamic programming [28], and heuristic programming [29, 30, 31]. The only formulations significantly different from those discussed in Chapter I are the heuristic programming approaches. These formulations also attempt to provide solutions to problems of realistic size. Nearly all heuristic programs for the delivery problem are tour building schemes which sequentially build a delivery route for a truck based on "penalties" or costs which might occur if a particular link were not incorporated into the route. The predominant cost measure is based on the symmetrical distance assumption which is highly untenable in the parallel processor problem. Furthermore, the heuristic schemes incorporate the symmetry assumption in such a fundamental way that extension to the asymmetric case is impossible.

Therefore, very few results in the literature are applicable directly to the problem of scheduling parallel processors. However, many of the ideas and suggestions in the literature are important in the development of new procedures.

CHAPTER III

SCHEDULING JOBS WITH ALL INFINITE DUE DATES
ON IDENTICAL PARALLEL PROCESSORS

This chapter treats the special case where the processors are identical ($c_{ijn} = c_{iju}$ for any i, j, n, u) and where all due dates are infinite.

There are two cases which arise in scheduling parallel processors.

Case 1. Scheduling M jobs on exactly N' of N available processors.

Case 2. Scheduling M jobs on an optimal number N^* of N available processors where $N^* \leq N$.

The assumption underlying Case 1 is that each of N processors must have at least one job on its schedule. This situation is frequently an operating policy, e.g., to meet labor agreements. The more general assumption underlying Case 2 is that the number of processors to be activated in the schedule is a decision variable.

As shown below, either case of the identical processor problem can be formulated as an augmented traveling salesman problem. However, this formulation cannot be extended to the general case. An alternative branch and bound solution procedure which can be extended is developed.

Formulation as an Augmented Traveling Salesman Problem

Either Case 1 or Case 2 of the identical processor problem under consideration can be solved as an augmented traveling salesman problem. This approach is inapplicable if any of the present assumptions are relaxed. However, the approach can be modified so that it is extendable to the generalized problem. The augmented traveling salesman formulation for each case is given as an introduction to the more generalized approach.

Case 1 Formulation

Consider an $M + 1$ city traveling salesman problem where the distances between cities i and j are given by the generally asymmetric distance matrix $\underline{D} = \{d_{ij}\}$, where $d_{ii} = \infty$. Let the cities $1, \dots, M + 1$ be numbered in such a way that $M + 1$ is the home city. The optimal single salesman solution is that tour

$$t = [(i_1, i_2), (i_2, i_3), \dots, (i_M, i_{M+1}), (i_{M+1}, i_1)]$$

which includes each city exactly once and for which the total distance $z(t) = \sum_{(i,j) \in t} d_{ij}$ traveled is a minimum.

Consider the case where N' salesmen are available at the home city $M + 1$, where each salesman must visit at least one city and where each city is to be visited exactly once. Augment the original problem given by \underline{D} by adding $N' - 1$ artificial cities so that the new problem has $M + N'$ cities.

Let $\hat{\tilde{D}} = \{\hat{\tilde{d}}_{ij}\}$ describe the augmented problem where

$$\hat{\tilde{D}} = \begin{array}{c} \begin{array}{c} 1 \\ 2 \\ \vdots \\ M+1 \\ M+2 \\ \vdots \\ M+N' \end{array} \left[\begin{array}{cc} \begin{array}{c} 1 \quad 2 \quad \dots \quad M+1 \end{array} & \begin{array}{c} M+2 \quad \dots \quad M+N' \end{array} \\ \hline \begin{array}{c} \tilde{D} \end{array} & \begin{array}{c} \tilde{B} \end{array} \\ \hline \begin{array}{c} \tilde{A} \end{array} & \begin{array}{c} \tilde{F} \end{array} \end{array} \right] \end{array}, \quad (\text{III-1})$$

and where the $(N' - 1) \times (M + 1)$ submatrix \tilde{A} is $N' - 1$ rows identical to row $M + 1$ of \tilde{D} , the $(M + 1) \times (N' - 1)$ submatrix \tilde{B} is $N' - 1$ columns identical to column $M + 1$ of \tilde{D} and \tilde{F} is an $(N' - 1) \times (N' - 1)$ submatrix with infinite elements.

The key result to follow is that the optimal N' salesman solution is imbedded in the optimal solution to the augmented problem. The ordered pairs (i, j) in any tour

$$t' = [(i_1, i_2), (i_2, i_3), \dots, (i_{M+N'}, i_1)]$$

include one element from each row and column of $\hat{\tilde{D}}$. Tours with finite total distance exist if $M \geq N'$, a condition which is assumed.

Theorem 3.1.¹ For every tour t' with finite total distance to the $M + N'$ city problem given by \hat{D} , there exists an N' salesman solution with equal total distance and the converse holds.

Proof. Given any tour

$$t' = [(i_1, i_2), (i_2, i_3), \dots, (i_{M+N'}, i_1)]$$

to the augmented problem, the total distance traveled will be infinite if any link (i, j) is included such that $M + 1 \leq i \leq M + N'$ and $M + 1 \leq j \leq M + N'$. Therefore, tours with finite total distance will not include any links between any combination of the original home city $M + 1$ and the artificial cities $M + 2, \dots, M + N'$. If the artificial cities in t' are identified and replaced by the home city $M + 1$, the result is a sequence of N' subtours (an N' -tour) since city $M + 1$ appears $N' + 1$ times. Furthermore, the N' -tour has the same cost as t' since the artificial cities are identical to city $M + 1$.

The converse follows by beginning with any N' salesman solution. This consists of N' subtours, each involving the home city $M + 1$. In $N' - 1$ of the subtours, city $M + 1$ can be replaced by one of the $N' - 1$ artificials, yielding

¹Slightly different versions of both Theorem 3.1 and Theorem 3.2 have been simultaneously and independently proved by Hong [33] using a graph theoretic approach.

a single tour of equal cost to the $M + N'$ city problem.

This result can be used to solve the present case of the identical processor problem if it is implicitly understood that rows $M + 1, \dots, M + N'$ represent initial jobs and columns $M + 1, \dots, M + N'$ represent final jobs. The change-over time matrix \tilde{C} is augmented with N' artificial jobs to obtain

$$\hat{\tilde{C}} = \{\hat{\tilde{c}}_{ij}\} = \begin{matrix} & \begin{matrix} 1 & 2 & \dots & M & M+1 & \dots & M+N' \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ \cdot \\ \cdot \\ \cdot \\ M \\ M+1 \\ \cdot \\ \cdot \\ \cdot \\ M+N' \end{matrix} & \left[\begin{array}{cccccc} & & & & \vdots & \\ & & & & \vdots & \\ & & & & \vdots & \\ & & \tilde{C} & & \vdots & \tilde{B} \\ & & & & \vdots & \\ & & & & \vdots & \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ & & \tilde{A} & & \vdots & \tilde{F} \\ & & & & \vdots & \end{array} \right] \end{matrix}, \quad (\text{III-2})$$

where \tilde{A} is an $(N' \times M)$ submatrix of identical rows, \tilde{B} is an $(N' \times M)$ submatrix of identical columns and $\tilde{F} = \infty$.

The optimal single processor schedule

$$s^* = [(i_1, i_2), (i_2, i_3), \dots, (i_{M+N'}, i_1)]$$

to the $M + N'$ job problem is the solution to a traveling salesman problem where s^* is presented in such a way that $i_1 = M + 1$. Let i_{k_j} be the j th artificial job in s^* , e.g., $i_{k_1} = i_1 = M + 1$. The optimal parallel processor schedule \tilde{S}^* can be constructed from s^* by letting

$$\tilde{S}_j = [(i_{k_j}, i_{k_{j+1}}), \dots, (i_{k_{j+1}-1}, i_{k_{j+1}})] \quad (\text{III-3})$$

where any changeover $(i_{k_{j-1}}, i_{k_j})$ to an artificial job indicates a changeover to final job $i_{k_j} + N'$ (by Theorem 3.1).

A minor complication arises because the j th artificial job i_{k_j} is an initial job, say $M + n$, and $i_{k_{j+1}} + N'$ represents a final job $M + N' + u$, where in general $n \neq u$.

According to the original problem assumptions, such a single processor schedule is inadmissible since it implies that a schedule starts on processor n and ends on processor u . For the present special case, the complication is only notational since all initial jobs are identical and all final jobs are identical. The complication becomes formidable, however, if any assumption is relaxed.

Case 2 Formulation

An analogous augmented traveling salesman formulation exists for the case where determining the optimal number $N^* \leq N$ is part of the identical processor problem. Let \hat{D} be defined as in equation (III-1) except that $N' = N$ and $\tilde{F} = \underline{0} = \{0\}$, an $(N - 1) \times (N - 1)$ matrix with zero entries.

The following theorem shows that the optimal multi-salesman tour (and therefore the optimal number of salesmen) is imbedded in the single salesman solution of \hat{D} .

Theorem 3.2. For every tour t' with finite total distance to the $M + N$ city problem given by \hat{D} , there exists an $\tilde{N} \leq N$ salesman solution with equal total distance and the converse holds.

Proof. Tours with finite total distance to the $M + N$ city problem may now contain links (i, j) between artificial cities $M + 2 \leq i \leq M + N, M + 2 \leq j \leq M + N$. Given any tour t' to the augmented problem, the artificial cities $M + 2, \dots, M + N$ may be replaced by the home city $M + 1$ without changing the total distance traveled. Therefore, exactly N subtours exist with the same cost as t' . However, some of the subtours may be degenerate subtours $(M + 1, M + 1)$ between the home city, each indicating an idle salesman. The number of salesmen utilized, \tilde{N} , equals the number of nondegenerate subtours and $\tilde{N} \geq 1$ since $d_{M+1, M+1} = \infty$ in the augmented problem.

The converse follows from the fact that an \tilde{N} -tour includes city $M + 1$ exactly $\tilde{N} + 1$ times. Then the $\tilde{N} - 1$ artificial cities may replace all but two of the home cities in such a way as to yield an equal cost single tour to the augmented problem.

This result can similarly be used to solve the identical processor problem where the optimal number of processors

$N^* \leq N$ is to be determined. An augmented single processor problem is solved, allowing a maximum of $N - 1$ changeovers between artificial jobs. The optimal parallel processor schedule is constructed from the optimal schedule to the augmented problem according to equation (III-3).

Both theorems 3.1 and 3.2 fail to hold if either the assumption of identical processors or infinite due dates is relaxed. A solution procedure which can be extended to more general cases must generate subtours which begin with initial job $M + n$ and end with the corresponding final job $M + N + n$. An alternate solution procedure for the identical processor problem, which can be extended is developed in the next section.

Branch and Bound Algorithm Development

The approach underlying the branch and bound algorithm to follow is that of imbedding the problem of scheduling M jobs on identical processors in the larger augmented single processor scheduling problem suggested by Theorem 3.1 and Theorem 3.2. Specifically, let F be the set of all feasible solutions to a given M job, parallel processor problem of Case 1 (given N') or Case 2 (find N^*). Let T be the set of all solutions to the corresponding M job augmented single processor problem given by equation (III-2).

The branch and bound algorithm must include a partitioning scheme, procedures to identify the elements in $T-F$

and rules by which a lower bound may be computed on the cost of any schedule in each subset isolated by the partitioning scheme. An iterative logic must be developed to drive the algorithm and consists of the recursive operations by which new collections of subsets are formed and by which subsets whose elements are either dominated or known to be infeasible are eliminated.

Partitioning Scheme

A partitioning, or branching, scheme must partition a given collection of subsets \underline{t} in such a way that any partitioning results in a new collection of subsets whose elements are collectively identical to the elements in \underline{t} and such that at least one divisible subset in \underline{t} is divided into proper subsets. The partitioning scheme used in the algorithm to follow always partitions a collection \underline{t} consisting of a single divisible subset T_i into two mutually exclusive subsets T_i' and T_i'' .

Specifically, let $T_i \subseteq T$ be a subset of solutions s to the augmented problem such that $T_i \neq \emptyset$. If $A = \bigcap_{s \in T_i} s$ is the set of all changeovers common to all schedules s in T_i , let $(p,q) \in \{\bigcup_{s \in T_i} s - A\}$. Then the partition is defined as

$$T_i' = \{s | s \in T_i; (p,q) \in s\}, \quad (\text{III-4})$$

$$T_i'' = \{s | s \in T_i; (p,q) \notin s\}. \quad (\text{III-5})$$

The partition defined by equations (III-4) and (III-5) is valid since it does not add any elements to the partitioned subsets and since the subsets are proper subsets.

Theorem 3.3. Given T_i' and T_i'' as defined by (III-4) and (III-5), then

$$T_i' \cup T_i'' = T_i,$$

$$T_i' \subset T_i \text{ and } T_i'' \subset T_i,$$

$$T_i' \neq T_i'' \neq T_i.$$

Proof. For any $s \in T_i$, either $(p,q) \in s$ or $(p,q) \notin s$. If $(p,q) \in s$, $s \in T_i'$ and $s \notin T_i''$. If $(p,q) \notin s$, $s \in T_i''$ and $s \notin T_i'$. Therefore $T_i' \cup T_i'' = T_i$. Also $T_i' \cap T_i'' = \emptyset$ by definition. Therefore, $T_i' \subset T_i$, $T_i'' \subset T_i$ and $T_i' \neq T_i'' \neq T_i$.

The complete partitioning scheme must specify some procedure for selecting the changeover (p,q) . A very powerful selection procedure is to choose that element (p,q) which, if not selected, would be likely to yield suboptimal solutions in T_i' . The motivation for this selection rule is the alternate cost concept [32].

Theorem 3.4. If a single processor schedule s does not include the changeover (i,j) then the cost $z(s)$ of s is bounded as follows

$$z(s) \geq \min_{u \neq j} \hat{c}_{iu} + \min_{v \neq i} \hat{c}_{vj} = \theta_{ij}. \quad (\text{III-6})$$

Selecting the changeover (p,q) such that $\theta_{pq} = \max \theta_{ij}$ for all $(i,j) \notin s$, $s \in T_i$, has been very powerful in single-salesman problems [6, 12].

Since $F \subseteq T$, then $z(\underline{s})$ is also bounded by θ_{ij} .

Corollary 3.1. If a parallel processor schedule \underline{S} is constructed from s where $(p,q) \notin s$, then

$$z(\underline{S}) \geq \theta_{ij} = \min_{u \neq j} \hat{c}_{iu} + \min_{v \neq i} \hat{c}_{vj} \quad (\text{III-7})$$

Therefore, selecting the changeover (p,q) by maximum alternate cost should be effective in the partitioning defined by (III-4) and (III-5).

Feasibility Tests

The algorithm must at least implicitly specify a collection of subsets \underline{t}_0 containing only and all infeasible parallel processor schedules. One way of accomplishing this is to construct the corresponding parallel processor schedule \underline{S} for any augmented problem solution s and then test \underline{S} against the definition of feasibility.

This procedure is inefficient since it only operates on a single solution \underline{S} . A more efficient procedure would be to find entire subsets of infeasible solutions. If (p,q) is selected and T_i' is partitioned according to (III-4), several

conditions necessary for feasibility for any $\underline{S} \in T_i'$ are as follows.

Condition (i). If i is the beginning job and j is the ending job in the partial sequence $(i,k), \dots, (p,q), \dots, (h,j)$ containing (p,q) , any $\underline{S} \in T_i'$ is feasible only if $(j,i) \notin \underline{S}$. Otherwise there are either cycles between real jobs or changeovers from final jobs. To indicate infeasibility of any \underline{S} such that $(j,i) \in \underline{S}$, let $\hat{c}_{ji} = \infty$.

Condition (ii). If (p,q) is imbedded in a sequence $[(M + n, k), \dots, (p,q), \dots, (h,j)]$, any $\underline{S} \in T_i'$ is feasible only if $(j, M + N + u) \notin \underline{S}$, where $u \neq n$. Otherwise, a schedule begins and ends on different processors. Let $\hat{c}_{j, M+N+u} = \infty$ for all $u \neq n$ to indicate infeasibility.

Condition (iii). If (p,q) is imbedded in a sequence $[(i,k), \dots, (p,q), \dots, (j, M + N + n)]$, any $\underline{S} \in T_i'$ is feasible only if $(M + u, i) \notin \underline{S}$, where $u \neq n$. Such schedules are infeasible for the same reasons given for Condition (ii). Set $\hat{c}_{M+u, i} = \infty$ for all $u \neq n$ to indicate infeasibility.

Condition (iv). If for any $\underline{S} \in T_i'$ sequence $[(M + n, k), \dots, (h,i)] \in \underline{S}$ and sequence $[(j,g), \dots, (f, M + N + u)] \in \underline{S}$, and $u \neq n$, then any \underline{S} such that $(i,j) \in \underline{S}$ is infeasible. Such schedules also begin and end on different processors. Therefore let $\hat{c}_{ij} = \infty$.

Lower Bounds

The key prerequisite in the development of any branch and bound algorithm is the computation of an efficient

lower bound on the cost of any solution in each partitioned subset. Since Theorem 3.1 and Theorem 3.2 allow the set of all feasible solutions F to the parallel processor problem to be enveloped by the set of all solutions T to an augmented single processor problem, then two well known theorems relating to single-salesman traveling salesman problems are helpful in establishing lower bounds.

Theorem 3.5 [32]. If $\hat{C} = \{\hat{c}_{ij}\}$ describes a single salesman problem, if k_p and k_q are real numbers associated with an entry c_{pq} such that

$$c'_{pj} = c_{pj} - k_p; (j=1, \dots, r; j \neq q)$$

$$c'_{iq} = c_{iq} - k_q; (i=1, \dots, r; i \neq p)$$

$$c'_{pq} = c_{pq} - k_p - k_q$$

$$c'_{rs} = c_{rs} \quad (r \neq p; s \neq q)$$

and if

$$z(s) = \sum_{(i,j) \in s} c_{ij}$$

$$z'(s) = \sum_{(i,j) \in s} c'_{ij}$$

then

$$z'(s) = z(s) - k_p - k_q.$$

Subtracting the smallest element in a row (column) of a matrix from each element in the row (column) is defined as row (column) reduction. A fully reduced matrix is one in which there exists at least one zero in each row and in each column. The above theorem suggests reducing a matrix as much as possible while maintaining nonnegativity and using the sum of the reducing constants to bound the cost of any constructed parallel processor schedule. The following corollary makes the scheme clear.

Corollary 3.2. If

$$\hat{c}'_{ij} = \hat{c}_{ij} - \min_v \hat{c}_{iv} - \min_u [\hat{c}_{uj} - \min_v \hat{c}_{uv}], \quad (\text{III-8})$$

then

$$\begin{aligned} \sum_{(i,j) \in \tilde{S}} \hat{c}_{ij} &= \sum_{(i,j) \in \tilde{S}} \hat{c}'_{ij} + \sum_i \min_v \hat{c}_{iv} \\ &\quad + \sum_j \min_u [\hat{c}_{uj} - \min_v \hat{c}_{uv}], \end{aligned}$$

so that

$$h = \sum_i \min_v \hat{c}_{iv} + \sum_j \min_u [\hat{c}_{uj} - \min_v \hat{c}_{uv}] \quad (\text{III-9})$$

is a lower bound on $\sum_{(i,j) \in \tilde{S}} \hat{c}_{ij}$.

The quantity h in (III-9) is a lower bound $b(F)$ on the total changeover time required by any feasible parallel processor schedule. Also, h is the sum of the constants used in subtracting the maximum constants from each row and each column.

Corollary 3.2 can be used recursively to compute lower bounds on any subset T_i' partitioned from T_i by equation (III-4). Since $(p,q) \in s \in T_i'$, jobs p and q can be joined to form a composite job, say r , so that a scheduling problem with one less job is represented by T_i' . Job r incurs the same changeover times as going to p and from q . In addition, certain changeovers known to be infeasible from feasibility conditions (i) - (iv) can be assigned an infinite cost in the new scheduling problem. If the matrix describing the new problem is reduced as suggested by (III-9), then h is the time in excess of the lower bound $b(T_i)$ required by selecting (p,q) . That is,

$$b(T_i') = b(T_i) + h. \quad (\text{III-10})$$

The other theorem helpful in establishing lower bounds is Theorem 3.4 from which a lower bound on the total changeover time required for any $\tilde{S} \in T_i$ is

$$b(T_i'') = b(T_i) + \theta_{pq}. \quad (\text{III-11})$$

Recursive Procedure

The recursive procedure selected for the algorithm is one which minimizes data storage requirements, which is frequently the limiting factor in problem size capability. The operations used to drive the algorithm to optimality are as follows.

Operation (i). For any partitioning, select the smallest subset T_i which is neither known to be infeasible or dominated. If no such T_i exists, terminate the algorithm.

Operation (ii). Partition T_i into T_i' and T_i'' using equations (III-4) and (III-5).

Operation (iii). Compute a lower bound on the cost of any solution in T_i' and T_i'' using equations (III-10) and (III-11).

Branch and Bound Algorithm

Based on the above considerations, the branch and bound algorithm for the identical processor scheduling problem is as follows.

Step 1. Construct \hat{C} according to equation (III-2). If N^* is to be determined, $N' = N$ and \tilde{F} is an $(N \times N)$ matrix with $(N - 1)$ diagonal elements equal to zero and all other elements equal to infinity. Let z_0 be the total changeover time for the best schedule available at any step of the

algorithm.

Step 2. Let $T_i = T$, the set of all parallel processor schedules. Reduce \hat{C} by equation (III-8). Compute h from equation (III-9). The lower bound $b(T_i) = h$.

Step 3. Compute θ_{ij} for each changeover (i,j) by equation (III-6). Let $\theta_{pq} = \max_{i,j} \theta_{ij}$.

Step 4. Partition T_i into T_i' and T_i'' according to (III-4) and (III-5). Compute $b(T_i'')$ from equation (III-11).

Step 5. Develop the matrix describing the problem given by the subset T_i'' by letting $\hat{c}_{pq} = \infty$.

Step 6. Develop the matrix describing the problem given by T_i' by (a) letting $\hat{c}_{pj} = \infty$, $1 \leq j \leq N'$ and $\hat{c}_{iq} = \infty$, $1 \leq i \leq N'$, and (b) finding those $\hat{c}_{ij} = \infty$ for those (i,j) found infeasible by applying feasibility conditions (i) - (iv).

Step 7. Reduce \hat{C} describing T_i' according to (III-8). Compute $b(T_i')$ from (III-10). If the reduced matrix has exactly two rows and two columns whose elements are not all infinite, $T_i' = \{S\}$, a single schedule with cost $b(T_i')$. The two remaining changeovers required to complete S are those changeovers which have zero times in each finite row and column. If $T_i' = S$ and if $b(T_i') < z_0$, let $z_0 = b(T_i')$.

Step 8. If $T_i' \neq \{S\}$ or if $b(T_i') \geq z_0$, backtrack to the smallest subset, say T_i , for which $b(T_i) < z_0$ and proceed according to Step 3; if no such T_i exists, the current best schedule is optimal and the algorithm is terminated.

Otherwise, go to Step 3, letting $T_i = T_i'$.

Illustrative Problems

An example problem is solved below for each of the two possible assumptions which could be made with respect to the number of processors to be used in the final schedule.

Identical Processor Example Where N' is Specified

Consider an $M = 5$ job, $N = 2$ available identical processor problem with changeover times

$$C = \begin{bmatrix} \infty & 8 & 4 & 7 & 2 \\ 2 & \infty & 7 & 2 & 4 \\ 2 & 5 & \infty & 6 & 3 \\ 4 & 4 & 3 & \infty & 5 \\ 1 & 2 & 3 & 1 & \infty \end{bmatrix}.$$

Let the changeover time from an initial job to any job j be u_j where $\underline{u} = \{u_j\} = [6, 5, 5, 3, 5]$ and let the changeover time to final job j be v_j where $\underline{v}' = \{v_j\} = [4, 5, 5, 1, 6]$.

Assume that exactly $N' = 2$ of the $N = 2$ available processors are to be activated in the final schedule. According to Step 1, an augmented matrix

$$\hat{\tilde{C}} = \begin{bmatrix} \infty & 8 & 4 & 7 & 2 & 4 & 4 \\ 2 & \infty & 7 & 2 & 4 & 5 & 5 \\ 2 & 5 & \infty & 6 & 3 & 5 & 5 \\ 4 & 4 & 3 & \infty & 5 & 1 & 1 \\ 1 & 2 & 3 & 1 & \infty & 6 & 6 \\ 6 & 5 & 5 & 3 & 5 & \infty & \infty \\ 6 & 5 & 5 & 3 & 5 & \infty & \infty \end{bmatrix}$$

is constructed. The total changeover time z_0 for the best schedule available equals infinity.

From Step 2, the subset at hand is $T_i = T$. $\hat{\tilde{C}}$ is reduced according to (III-8) to yield

$$\hat{\tilde{C}}' = \begin{bmatrix} \infty & 5 & 0^2 & 5 & 0^1 & 2 & 2 \\ 0^0 & \infty & 3 & 0^0 & 2 & 3 & 3 \\ 0^1 & 2 & \infty & 4 & 1 & 3 & 3 \\ 3 & 2 & 0^0 & \infty & 4 & 0^2 & 0^2 \\ 0^0 & 0^1 & 0^0 & 0^0 & \infty & 5 & 5 \\ 3 & 1 & 0^0 & 0^0 & 2 & \infty & \infty \\ 3 & 1 & 0^0 & 0^0 & 2 & \infty & \infty \end{bmatrix},$$

where the superscripts are the alternate costs to be computed in Step 3. (Non-zero elements have zero alternate cost.) A lower bound on the total changeover time required for any $S \in T_i$ is $b(T_i) = h = 17$, computed from equation (III-9). The progress of the solution procedure can be represented by the

tree given in Figure 1 where $b(T) = 17$ since $T_i = T$.

From Step 3, changeover (4,9) is tied for maximum alternate cost and is arbitrarily selected as the changeover on which to base the first partitioning. (Note that the $\hat{C}'_{4,7}$ element in \hat{C}' represents the (4,9) changeover since changeovers to artificial job n are to be taken as changeovers to final job $M + N + n$.)

According to Step 4, set T_i is partitioned into T_i' which includes all \tilde{S} such that $(4,9) \in \tilde{S}$ and T_i'' which includes all \tilde{S} such that $(4,9) \notin \tilde{S}$. The subsets T_i' and T_i'' are denoted $(4,9)$ and $(\overline{4,9})$, respectively, in Figure 1.

Also $b(T_i'') = b(T_i) + \theta_{4,9} = 17 + 2 = 19$.

From Step 5, the changeover time matrix describing the scheduling problem given by the subset T_i'' is

$$\hat{\tilde{C}} = \begin{bmatrix} \infty & 5 & 0 & 5 & 0 & 2 & 2 \\ 0 & \infty & 3 & 0 & 2 & 3 & 3 \\ 0 & 2 & \infty & 4 & 1 & 3 & 3 \\ 3 & 2 & 0 & \infty & 4 & 0 & \infty \\ 0 & 0 & 0 & 0 & \infty & 5 & 5 \\ 3 & 1 & 0 & 0 & 2 & \infty & \infty \\ 3 & 1 & 0 & 0 & 2 & \infty & \infty \end{bmatrix}.$$

The matrix describing the scheduling problem at T_i' is

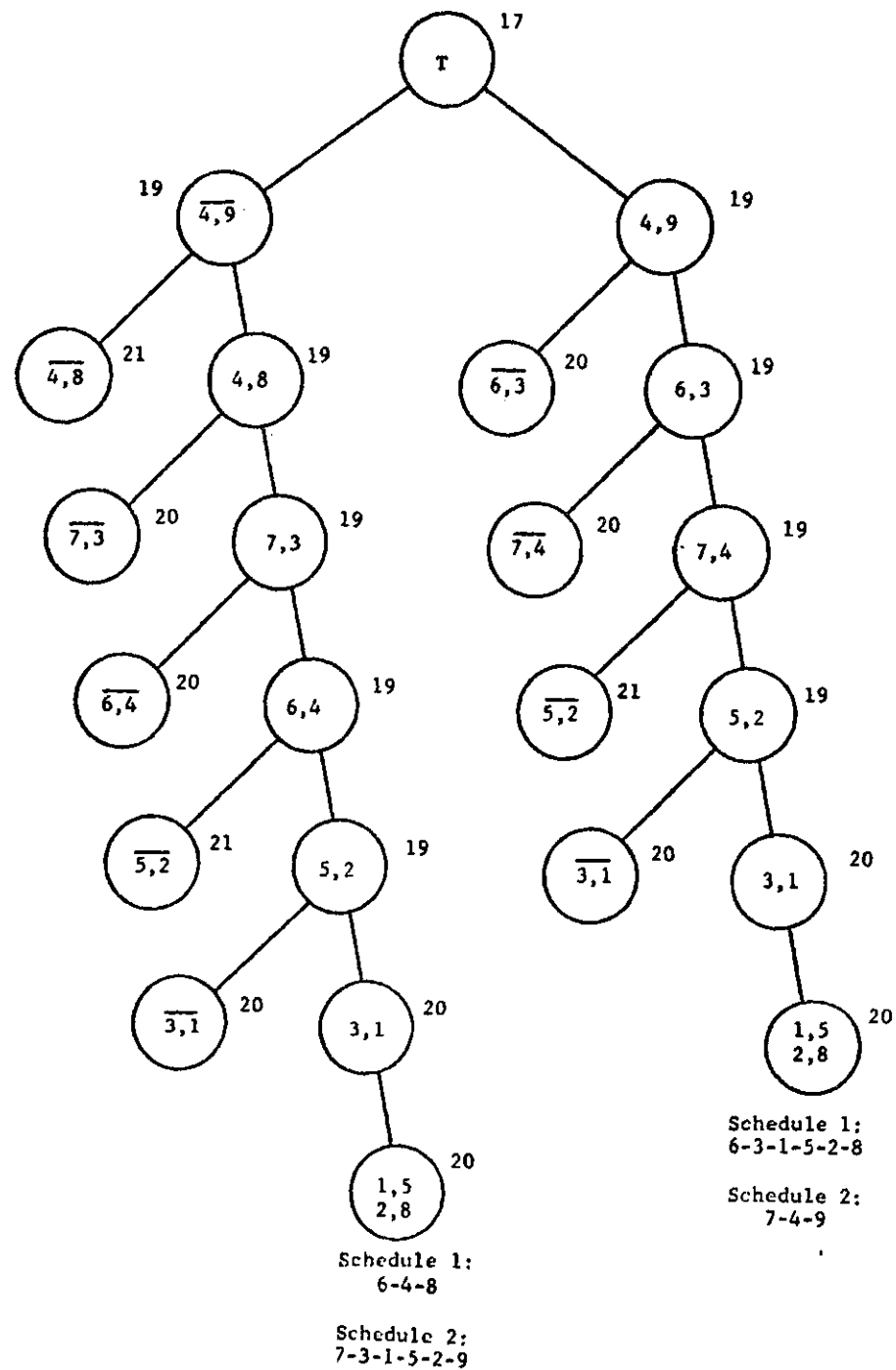


Figure 1. Tree Representation of the Solution to the Identical Processor Example Where N' is Specified

$$\hat{\tilde{C}} = \begin{bmatrix} \infty & 5 & 0 & 5 & 0 & 2 & \infty \\ 0 & \infty & 3 & 0 & 2 & 3 & \infty \\ 0 & 2 & \infty & 4 & 1 & 3 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 0 & 0 & \infty & 5 & \infty \\ 3 & 1 & 0 & \infty & 2 & \infty & \infty \\ 3 & 1 & 0 & 0 & 2 & \infty & \infty \end{bmatrix}$$

which was developed according to Step 6. The only active feasibility condition was (i) which made the (6,4) change-over infeasible.

The new matrix $\hat{\tilde{C}}$ describing T_i' is reduced according to Step 7 and the lower bound $b(T_i') = b(T_i) + h = 17 + 2 = 19$.

Step 8 required that Steps 3 - 7 be performed recursively. The remainder of the iterations are entirely analogous to the one performed above, and are summarized in Figure 1. Note that an optimal solution with a total change-over time of 20 was found on the first pass, but that one backtrack was required to prove optimality. The backtrack terminated with an alternate optimal solution identical to the first except that the processors are reversed.

Identical Processor Example Where N^* is to be Determined

The same problem given above can be solved under the assumption that N^* is to be determined. The only significant difference is the way in which the algorithm is started.

Step 1 now specifies that

$$\hat{C} = \begin{bmatrix} \infty & 8 & 4 & 7 & 2 & 4 & 4 \\ 2 & \infty & 7 & 2 & 4 & 5 & 5 \\ 2 & 5 & \infty & 6 & 3 & 5 & 5 \\ 4 & 4 & 3 & \infty & 5 & 1 & 1 \\ 1 & 2 & 3 & 1 & \infty & 6 & 6 \\ 6 & 5 & 5 & 3 & 5 & 0 & \infty \\ 6 & 5 & 5 & 3 & 5 & \infty & 0 \end{bmatrix} .$$

The iterations are summarized in the tree given by Figure 2. Note that no backtracking was required to find the optimal solution to this problem, which has a total changeover time of 14. It is interesting that the previous example requiring exactly two processors in the final solution required an incremental 6 units of changeover time over the unconstrained solution, which is approximately a 43% increase in total changeover time.

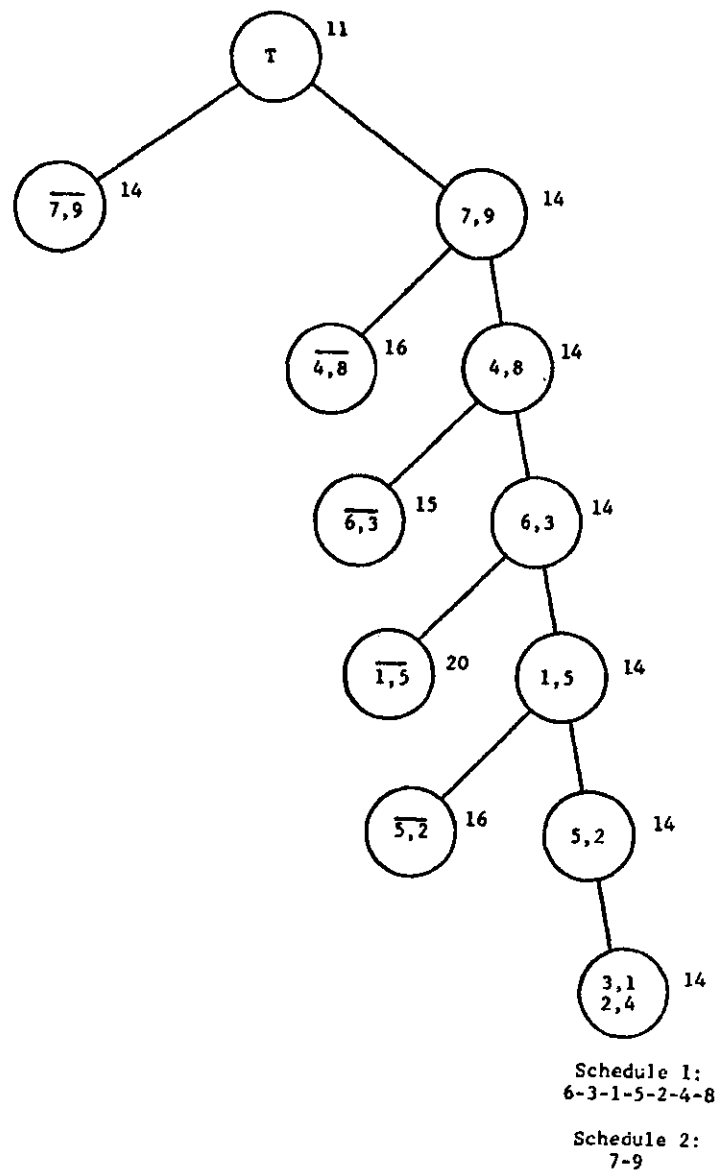


Figure 2. Tree Representation of the Solution to the Identical Processor Example Where N^* is to be Determined

CHAPTER IV

SCHEDULING JOBS WITH ALL INFINITE DUE DATES ON DISTINCT PARALLEL PROCESSORS

The present chapter extends the identical processor scheduling algorithm to admit distinct processor problems. Distinct processors mean that, in general, $c_{ijn} \neq c_{iju}$ when $n \neq u$. It is still assumed that $d_j = \infty$ for all j . The existence of N distinct changeover cost matrices precludes a formulation entirely analogous to the identical processor formulation. Therefore, certain concepts underlying the identical processor algorithm must be extended.

Extensions of Identical Processor Algorithm Components

Let F be the set of all feasible distinct processor schedules. Any admissible schedule is also a feasible schedule if all $d_j = \infty$. Let T be the set of all parallel processor schedules in which an initial job is performed first and a final job is performed last on each processor. The approach is to imbed the problem of finding an optimal feasible parallel processor schedule $\tilde{S}^* \in F$ in the less restrictive problem of finding an optimal parallel processor schedule $\tilde{S}^* \in T$ in such a way as to insure feasibility. Therefore, the better than optimal but infeasible solutions in $T - F$ must be efficiently isolated. In addition, the

branch and bound partitioning scheme and bounding rules must be extended to treat the more generalized problem.

Partitioning Scheme

The partitioning scheme used in the distinct processor branch and bound algorithm always partitions a single nonempty subset T_i whose elements are schedules which have in common a number (possibly zero) of changeovers on specific processors and which prohibit a number (possibly zero) of changeovers on specific processors. The partitioning is made by selecting a changeover $(p,q)_n$ for processor n and dividing T_i into T_i' all of whose elements include the $(p,q)_n$ changeover and T_i'' all of whose elements prohibit the $(p,q)_n$ changeover.

That is, $T_i \subseteq T$, $T_i \neq \emptyset$ and the subset of changeovers common to all $\tilde{S} \in T_i$ is $A = \bigcap_{\tilde{S} \in T_i} \tilde{S}$. If $(p,q)_n \in \{ \bigcup_{\tilde{S} \in T_i} \tilde{S} - A \}$, then the partition is defined as

$$T_i' = \{ \tilde{S} | \tilde{S} \in T_i; (p,q)_n \in \tilde{S} \} \quad (IV-1)$$

$$T_i'' = \{ \tilde{S} | \tilde{S} \in T_i; (p,q)_n \notin \tilde{S} \}. \quad (IV-2)$$

This partitioning scheme differs from the earlier scheme of equations (III-4) and (III-5) since it explicitly assigns changeovers to processors. However, it is a valid partitioning procedure since it partitions a subset T_i into proper subsets and does not add any elements in the partitioning.

Theorem 4.1. Given T_i' and T_i'' as defined by (IV-1) and (IV-2), then

$$T_i' \cup T_i'' = T_i,$$

$$T_i' \subset T_i \text{ and } T_i'' \subset T_i,$$

$$T_i' \neq T_i'' \neq T_i.$$

Proof. For any $\underline{S} \in T_i$, either $(p,q)_n \in \underline{S}$ or $(p,q)_n \notin \underline{S}$. If $(p,q)_n \in \underline{S}$, then $\underline{S} \in T_i'$ and $\underline{S} \notin T_i''$. Otherwise $\underline{S} \in T_i''$ and $\underline{S} \notin T_i'$. Therefore $T_i' \cup T_i'' = T_i$. Also, since $T_i' \cap T_i'' = \emptyset$, then $T_i' \subset T_i$, $T_i'' \subset T_i$ and $T_i' \neq T_i'' \neq T_i$.

The selection procedure for the changeover $(p,q)_n$ is an extension of the alternate cost procedure used for the identical processor problem. If a changeover $(i,j)_n$ is not made on processor n , then exactly one of the following two events may occur: (1) the changeover (i,j) is not made on any processor, necessitating a changeover $(i,u)_r$, $u \neq j$ and a changeover $(v,j)_t$, $v \neq i$; or (2) the changeover $(i,j)_r$, $r \neq n$ is made. The occurrence of exactly one of the events (1) or (2) is necessary for an admissible parallel processor schedule since each job must be processed. These considerations lead to the following lower bounding on the cost of any parallel processor schedule which does not include a given changeover $(i,j)_n$.

changeovers from each $S \in T_i$, increasing the probability that the optimal solution $S^* \in T_i'$.

Feasibility Tests

Since each changeover is explicitly assigned to a specific processor, some new feasibility tests can be stated and some of the identical processor feasibility tests can be extended. The following conditions are necessary for feasibility.

Condition (i). Given set T_i , let $(j,k)_p \in S_p \in T_i$. A partition T_i' of T_i using $(r,s)_n$ is admissible only if $j \neq k \neq r \neq s$ when $p \neq n$. This condition states that changeovers both from and to a given job must be performed on only one processor to insure admissibility.

Condition (ii). Given a set T_i , suppose $(M+n, M+N+n)_n \in S_n \in T_i$. Then any partition T_i' of T_i using changeover $(r,s)_p$ is admissible only if $p \neq n$. This condition states that when an initial - final job pair has been selected for processor n (implying that processor n is not activated) then requiring processor n to make any other changeovers results in infeasible solutions.

Condition (iii). Given a set T_i , suppose there exists a complete schedule $S_p = [(M+n, i_{1,p}), \dots, (i_{m_p,p}, M+N+p)] \in T_i$. Then a partitioning T_i' of T_i based on $(r,s)_n$ is admissible only if $n \neq p$. This condition is actually the general case of Condition (ii) and states that if a complete schedule for any processor is contained in a subset of

Theorem 4.2. Let $(i,j)_n$ be a changeover on processor n such that $c_{ijn} < \infty$. The cost $z(\underline{S})$ of any admissible schedule \underline{S} such that $(i,j)_n \notin \underline{S}$ is bounded as follows:

$$z(\underline{S}) \geq \min [\gamma_{ijn}; \xi_{ijn}] = \theta_{ijn}, \quad (\text{IV-3})$$

$$\text{where } \gamma_{ijn} = \min_{\substack{1 \leq r \leq N \\ \bar{u} \neq \bar{j}}} \{c_{iur}\} + \min_{\substack{1 \leq r \leq N \\ \bar{v} \neq \bar{i}}} \{c_{vjr}\}$$

$$\text{and } \xi_{ijn} = \min_{r \neq n} \{c_{ijr}\}.$$

Proof. If for an admissible schedule \underline{S} , $(i,j)_n \notin \underline{S}$, then either (1) the (i,j) changeover is not made on any processor or (2) the (i,j) changeover is made on processor r , $r \neq n$. If event (1) occurs, a changeover $(i,u)_r$ and a changeover $(v,j)_t$ must be included in \underline{S} since \underline{S} was assumed to be an admissible schedule. Therefore, if event (1) occurs $z(\underline{S}) \geq \gamma_{ijn}$. If event (2) occurs, then $z(\underline{S}) \geq \xi_{ijn}$. Since exactly one of the events (1) and (2) must occur if \underline{S} is to be admissible, then the cost of \underline{S} can be bounded from below by finding the minimum over all outcomes.

In general, the subset T_i' partitioned from T_i will be smaller than T_i'' because the schedules in T_i' have more changeovers in common. Therefore the partition is constructed according to (IV-1) and (IV-2), selecting $(p,q)_n$ so that $\theta_{pqn} = \max \{\theta_{ijr}\}$. This procedure excludes high-time

solutions, then any further partition based on another changeover on that processor is inadmissible.

Condition (iv). Given a set T_i , suppose subsequences $[(M+n, i_{1,n}), \dots, (i_{j-1,n}, i_{j,n})] \in T_i$, $[(i_{k,n}, i_{k+1,n}), \dots, (i_{x_n,n}, M+N+n)] \in T_i$ and $(w,x)_n \in T_i$. A partitioning T_i' of T_i based on $(r,s)_p$ is admissible only if $r \neq i_{j,n}$ and $s \neq i_{k,n}$ when $p = n$. This condition simply states that a complete schedule for processor n which excludes a changeover already assigned to processor n is inadmissible.

Condition (v). Given a set T_i , suppose $(j,k)_p \in S_p \in T_i$. Any partition T_i' of T_i based on a changeover $(M+n, M+N+n)$ is admissible only if $n \neq p$. This condition states that when one or more jobs have already been assigned to a processor, the selection of an initial - final pair implying no work for the same processor results in an infeasible solution.

Condition (vi). If $(r,s)_n \in T_i$, then $(s,r)_n$ is inadmissible in T_i or any of its partitions. This condition relates to inadmissible cycles in a given processor's schedule.

Condition (vii). If N^* is to be determined and if $(N-1)$ initial-final changeovers $(M+n, M+N+n)_n \in \bigcup_{S \in T_i} S$, then the remaining initial-final changeover $(M+u, M+N+u)_u \notin \bigcup_{S \in T_i} S$ is infeasible. Otherwise, there would be an initial-final sequence on all processors.

Condition (viii). If exactly N' processors are to

be activated and if $N - N'$ initial-final changeovers

$(M + n, M + N + n)_n \in \bigcup_{S \in T_i} S$ any initial-final changeover

$(M + u, M + N + u)_u \in \bigcup_{S \in T_i} S$ is infeasible. Otherwise, less than N' processors will be activated.

Condition (ix). Suppose exactly N' processors are to be activated and the set of changeovers $\bigcup_{S \in T_i} S$ place m jobs on p processors. Then if $M - m \leq N' - p$, any changeover $(r, s)_n \notin \bigcup_{S \in T_i} S$ is infeasible if n is one of the p processors. Otherwise, less than N' processors would be activated. Also, if $M - m < N' - p$, the entire subset of solutions T_i' is infeasible. Similarly, if $p = N'$ any $(r, s)_n \notin \bigcup_{S \in T_i} S$ is infeasible if n is not one of the p processors, unless $(r, s)_n$ is an initial-final changeover.

The simultaneous satisfaction of the above conditions is only a necessary condition for admissibility. However, they do identify a large number of inadmissible schedules because they identify entire subsets of inadmissible solutions. Conditions (i) - (vi) combined with the definition of an admissible schedule above provide an efficient mechanism for generating a collection \underline{t}_0 of inadmissible subsets.

Lower Bounds

Suppose a subset of solutions T_i is partitioned into T_i' and T_i'' according to (IV-1) and (IV-2). The identical processor procedure for establishing a lower bound $b(T_i')$ on the total changeover time required by any $S \in T_i$ can be extended to the distinct processor case. Consider an array

$\hat{\tilde{C}} = \{c_{ijn}\}$ constructed from matrices

$$\hat{\tilde{C}}_n = \begin{matrix} & \begin{matrix} 1 & 2 & \dots & M & & M+1 & \dots & M+N \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ \vdots \\ M \\ M+1 \\ \vdots \\ M+N \end{matrix} & \left[\begin{array}{cc|cc} & & & \\ & & & \\ & & & \\ & C_n = \{c_{ijn}\} & & \tilde{B} \\ & & & \\ \hline & & \tilde{A} & \tilde{D} \\ & & & \end{array} \right] \end{matrix} \quad (IV-4)$$

The elements of the upper left submatrix \tilde{C}_n equal those entries in \tilde{C} which give the changeover times between original jobs for processor n . Submatrix \tilde{A} has infinite entries except for row n (row $M + n$ of $\hat{\tilde{C}}_n$) in which element $\hat{c}_{M+n,j,n}$ equals the changeover time from processor n 's initial job $M + n$ to job j . Similarly, submatrix \tilde{B} has infinite entries except for column n (column $M + n$ of $\hat{\tilde{C}}_n$) in which element $\hat{c}_{i,M+n,n}$ equals the changeover time from job i to processor n 's final job $M + N + n$. \tilde{D} is an $(N \times N)$ submatrix with diagonal elements equal to zero and off-diagonal elements equal to infinity.

The distinct processor bounding procedure rests on array reduction of $\hat{\tilde{C}}$. Array $\hat{\tilde{C}}$ is fully reduced if for each

i there exists some h and k such that $\hat{c}_{ihk} = 0$ and for each j there exists some f and g such that $\hat{c}_{fjg} = 0$.

Array reduction can be used to establish lower bounds on any $\tilde{S} \in T_i'$ using a procedure analogous to the identical processor bounding $b(T_i')$. The basis for this extension is that constants can be systematically subtracted from certain elements of array \hat{C} to obtain an array \hat{C}' ; and that the cost of any \tilde{S} under \hat{C}' equals the cost under \hat{C} less the constants subtracted. The following is the result underlying the extension.

Theorem 4.3. Given \hat{C} constructed according to (IV-4), if k_{pn} and k_{qn} are real numbers associated with \hat{c}_{pqn} such that

$$\hat{c}'_{pju} = c_{pju} - k_{pn} \quad j \neq q; 1 \leq u \leq N$$

$$\hat{c}'_{iqu} = c_{iqu} - k_{qn} \quad i \neq p; 1 \leq u \leq N$$

$$\hat{c}'_{pqu} = c_{pqu} - k_{pn} - k_{qn} \quad 1 \leq u \leq N$$

$$\hat{c}'_{rtu} = c_{rtu} \quad r \neq p; t \neq q; 1 \leq u \leq N$$

and if for admissible schedule \tilde{S}

$$z(\tilde{S}) = \sum_{(i,j) \in \tilde{S}} \hat{c}_{ijn}$$

$$z'(\tilde{S}) = \sum_{(i,j)_n \in \tilde{S}} \hat{c}'_{ijn},$$

then
$$z'(\tilde{S}) = z(\tilde{S}) - k_{pn} - k_{qn}.$$

Proof. In any admissible schedule \tilde{S} , all original jobs precede exactly one other job and all original jobs follow exactly one other job. Final jobs precede exactly zero other jobs and initial jobs follow exactly zero other jobs. Therefore, exactly one changeover $(p,j)_n$, $1 \leq j \leq M + N$, $n \in \underline{N}$ will be included in any admissible changeover. Then, if a constant k_{pn} is subtracted from each such $(p,j)_n$, $1 \leq j \leq M + N$, $n \in \underline{N}$, then the cost of any admissible schedule \tilde{S} under the revised costs will be exactly k_{pn} less than under old costs. Similarly, any admissible schedule includes exactly one changeover $(i,q)_n$, $1 \leq i \leq M + N$, $n \in \underline{N}$, and subtracting k_{qn} from each such $(i,s)_n$ reduces the total cost of any admissible schedule \tilde{S} by exactly k_{qn} .

Theorem 4.3 suggests that repeated subtraction of constants from the cost data for any subset of solutions T_i be performed such that $\hat{c}'_{ijn} \geq 0$ and using the sum of the subtraction constants as a lower bound on the cost $z(\tilde{S})$ of any $\tilde{S} \in T_i$.

Corollary 4.1. If

$$\hat{c}'_{ijn} = \hat{c}_{ijn} - \min_{v,r} \hat{c}_{ivr} - \min_{u,r} [\hat{c}_{ujr} - \min_{v,r} \hat{c}_{uvr}], \quad (\text{IV-5})$$

then

$$\begin{aligned} \sum_{(i,j) \in \tilde{S}} \hat{c}_{ijn} &= \sum_{(i,j) \in \tilde{S}} \hat{c}'_{ijn} + \sum_i \min_{v,r} \hat{c}_{ivr} \\ &+ \sum_j \min_{u,r} [\hat{c}_{ujr} - \min_{v,r} \hat{c}_{uvr}], \end{aligned}$$

so that

$$h = \sum_i \min_{v,r} \hat{c}_{ivr} + \sum_j \min_{u,r} [\hat{c}_{ujr} - \min_{v,r} \hat{c}_{uvr}] \quad (\text{IV-6})$$

is a lower bound on the total changeover time for any \tilde{S} under \hat{C} .

Since the minimum over all processors r occurs in each reduction constant in (IV-5) and (IV-6), a $(M + N) \times (M + N)$ composite matrix

$$\tilde{C}^* = \{c_{ij}^*\} = \min_r \{\hat{c}_{ijr}\} \quad (\text{IV-7})$$

can be reduced in the usual way to find h . That is

$$c_{ij}^{*'} = c_{ij}^* - \min_v c_{iv}^* - \min_u [c_{uj}^* - \min_v c_{uv}^*] \quad (\text{IV-8})$$

$$h = \sum_i \min_v c_{iv}^* + \sum_j \min_u [c_{uj}^* - \min_v c_{uv}^*]. \quad (\text{IV-9})$$

is a lower bound $b(F)$ on any $\tilde{S} \in T_i$.

An analogous procedure can be used to bound from below the time required for any subset $T_i' \subset T_i$ defined by (IV-1). Jobs p and q can be treated as a single job on processor n , so that a scheduling problem with one less job is represented by T_i' . Infeasible changeovers identified from feasibility conditions (i) - (vi) can be assigned an infinite cost in array \hat{C} . Matrix \tilde{C}^* can be recomputed according to (IV-7) to describe the new problem at T_i' . Then \tilde{C}^* can be reduced according to (IV-8) and (IV-9) to find the time in excess of $b(T_i)$ incurred by selecting $(p,q)_n$. That is

$$b(T_i') = b(T_i) + h. \quad (IV-10)$$

The lower bounding of any $S \in T_i''$ defined according to (IV-2) is a straightforward application of Theorem 4.2 so that

$$b(T_i'') = b(T_i) + \theta_{pqn} \quad (IV-11)$$

where θ_{pqn} is computed according to equation (IV-3).

Branch and Bound Algorithm

Based on the above extensions, and using the analogous recursive operations, the generalized algorithm can be stated as follows.

Step 1. Construct \hat{C} according to (IV-4) and

feasibility conditions (vii) - (ix). Construct \tilde{C}^* according to (IV-7). Let z_0 be the total changeover time for the best schedule available at any step of the algorithm; initially $z_0 = \infty$.

Step 2. Let $T_i = T$, the set of all parallel processor schedules. Reduce \tilde{C}^* according to (IV-8), performing the same operations on \tilde{C} according to (IV-5). Compute h from (IV-9) and let $b(T_i) = h$.

Step 3. Compute θ_{ijn} for each $(i,j)_n \in \tilde{C}^*$ and let $\theta_{pqn} = \max \{\theta_{ijn}\}$.

Step 4. Partition T_i into T_i' and T_i'' according to (IV-1) and (IV-2). Compute $b(T_i'')$ from (IV-11).

Step 5. Develop the data describing the problem at T_i'' by letting $\hat{c}_{pqn} = \infty$, letting $c_{pq}^* = \min_u \{c_{pqu}\}$.

Step 6. Develop the data describing the problem at T_i' by (a) letting $\hat{c}_{pju} = \infty$, $1 \leq j \leq M + N$, $u \in \underline{N}$, (b) letting $\hat{c}_{iqu} = \infty$, $1 \leq i \leq M + N$, $u \in \underline{N}$, (c) letting $c_{iju} = \infty$ for those $(i,j)_n$ which are known to be infeasible by feasibility conditions (i) - (ix) and (d) recomputing \tilde{C}^* according to equation (IV-7).

Step 7. Reduce \tilde{C}^* according to (IV-8), performing the same operations on \tilde{C} according to (IV-5). Compute h from (IV-6) and $b(T_i')$ from (IV-10). If \tilde{C}^* has exactly two rows and two columns which are not all infinite, complete the single schedule $S \in T_i'$ by adding those changeovers which have one zero in each noninfinite row and column of \tilde{C}^* . If T_i'

$= \{\underline{S}\}$ and if $b(T_i') < z_0$, let $z_0 = b(T_i')$.

Step 8. If $T_i' \neq \{\underline{S}\}$ or if $b(T_i') \geq z_0$, backtrack to the smallest subset, say T_i , for which $b(T_i) < z_0$ and proceed according to Step 3; if no such T_i exists, the current best schedule is optimal and the algorithm is terminated. Otherwise, go to Step 3, letting $T_i = T_i'$.

Illustrative Problems

Two example problems are given below to illustrate the use of the algorithm for the case when the number of processors to be utilized is a given number $N' \leq N$ and when part of the scheduling problem is to determine N^* , the optimal number of processors.

Distinct Processor Example Where N' is Specified

Consider an $M = 5$ job, $N = 2$ available distinct processor problem where the number of processors N' to be activated in the final schedule is given as $N' = 2$. The changeover cost data for processors one and two are, respectively,

$$C_1 = \begin{bmatrix} \infty & 6 & 5 & 9 & 2 \\ 3 & \infty & 3 & 8 & 6 \\ 1 & 7 & \infty & 9 & 7 \\ 2 & 5 & 9 & \infty & 3 \\ 3 & 8 & 9 & 8 & \infty \end{bmatrix}, \quad (\text{IV-12})$$

$$C_2 = \begin{bmatrix} \infty & 4 & 4 & 6 & 1 \\ 2 & \infty & 2 & 8 & 3 \\ 3 & 9 & \infty & 6 & 8 \\ 4 & 3 & 4 & \infty & 4 \\ 8 & 9 & 2 & 5 & \infty \end{bmatrix} . \quad (IV-13)$$

Let the changeover time from initial job $M + n$ to any job j be given by u_{nj} where

$$\underline{U} = \{u_{nj}\} = \begin{bmatrix} 4 & 5 & 9 & 9 & 3 \\ 1 & 3 & 5 & 7 & 1 \end{bmatrix}. \quad (\text{IV-14})$$

Similarly, let the time from any job j to final job $M + N + n$ be v_{nj} where

$$\underline{V} = \{v_{nj}\} = \begin{bmatrix} 1 & 7 & 9 & 7 & 1 \\ 2 & 4 & 8 & 8 & 5 \end{bmatrix} . \quad (\text{IV-15})$$

According to Step 1,

[illegible]

$$\hat{\tilde{C}}_2 = \begin{bmatrix} \infty & 4 & 4 & 6 & 1 & \infty & 2 \\ 2 & \infty & 2 & 8 & 3 & \infty & 4 \\ 3 & 9 & \infty & 6 & 8 & \infty & 8 \\ 4 & 3 & 4 & \infty & 4 & \infty & 8 \\ 8 & 9 & 2 & 5 & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 1 & 3 & 5 & 7 & 1 & \infty & \infty \end{bmatrix} \quad \text{and (IV-17)}$$

$$\tilde{C}^* = \begin{bmatrix} \infty & 4 & 4 & 6 & 1 & 1 & 2 \\ 2 & \infty & 2 & 8 & 3 & 7 & 4 \\ 1 & 7 & \infty & 6 & 7 & 9 & 8 \\ 2 & 3 & 4 & \infty & 3 & 7 & 8 \\ 3 & 8 & 2 & 5 & \infty & 1 & 5 \\ 4 & 5 & 9 & 9 & 3 & \infty & \infty \\ 1 & 3 & 5 & 7 & 1 & \infty & \infty \end{bmatrix} . \quad \text{(IV-18)}$$

Note that the submatrix \tilde{D} in (IV-4) has all infinite elements due to feasibility condition (viii).

From Step 2, \tilde{C}^* and $\hat{\tilde{C}}$ are reduced to yield

$$\tilde{C}^{*'} = \begin{bmatrix} \infty & 2 & 4 & 1 & 0 & 0 & 0 \\ 0 & \infty & 0 & 2 & 1 & 5 & 1 \\ 0 & 4 & \infty & 1 & 6 & 8 & 6 \\ 0 & 0 & 2 & \infty & 1 & 5 & 5 \\ 2 & 6 & 1 & 0 & \infty & 0 & 3 \\ 1 & 1 & 6 & 2 & 0 & \infty & \infty \\ 0 & 1 & 4 & 2 & 0 & \infty & \infty \end{bmatrix} , \quad \text{(IV-19)}$$

$$\hat{\tilde{C}}_1' = \begin{bmatrix} \infty & 4 & 4 & 4 & 1 & 0 & \infty \\ 1 & \infty & 1 & 2 & 4 & 5 & \infty \\ 0 & 5 & \infty & 4 & 6 & 8 & \infty \\ 0 & 2 & 7 & \infty & 1 & 5 & \infty \\ 2 & 6 & 8 & 3 & \infty & 0 & \infty \\ 1 & 1 & 6 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \quad \text{and (IV-20)}$$

$$\hat{\tilde{C}}_2' = \begin{bmatrix} \infty & 2 & 3 & 1 & 0 & \infty & 0 \\ 0 & \infty & 0 & 2 & 1 & \infty & 1 \\ 2 & 7 & \infty & 1 & 7 & \infty & 6 \\ 2 & 0 & 2 & \infty & 2 & \infty & 5 \\ 7 & 7 & 1 & 0 & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 1 & 4 & 2 & 0 & \infty & \infty \end{bmatrix} \quad \text{(IV-21)}$$

A lower bound on the total changeover time required by any feasible schedule is $h = 17$ and the tree of Figure 3 is started.

According to Step 3, element $(6,5)$ of \tilde{C}^* has the maximum alternate cost of one. Therefore $(p,q)_n = (6,5)_1$ since $c_{6,5}^* = c_{6,5,1}$.

By Step 4, the set T_i at hand is partitioned into T_i' , denoted by $(6,5)_1$ in Figure 3, and T_i'' , denoted by $(\overline{6},5)_1$ in Figure 3.

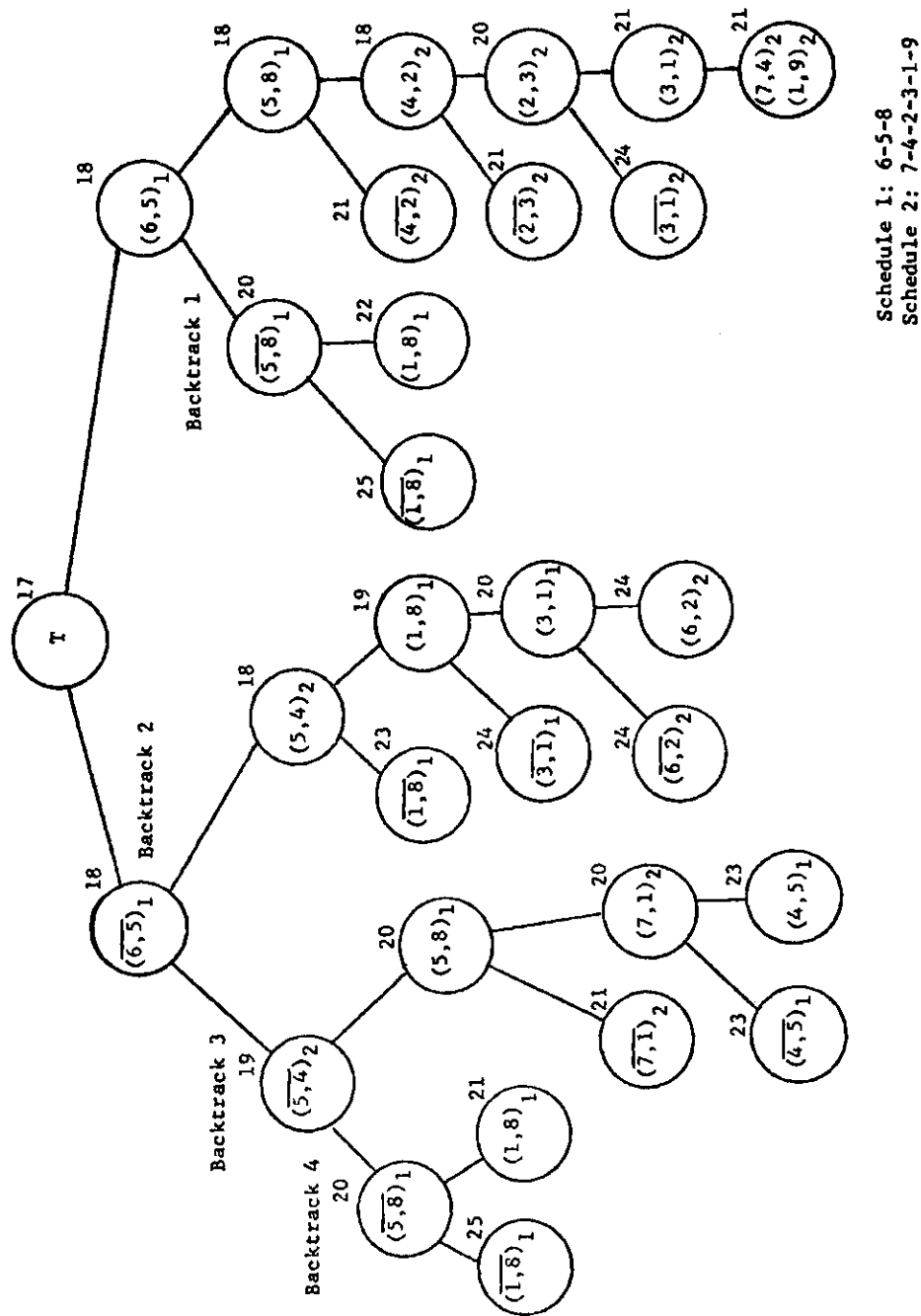


Figure 3. Tree Representation of Solution to the Distinct Processor Example where N' is Specified

The cost data to be computed in Step 5 describing the scheduling problem of subset T_i is identical to equations (IV-19), (IV-20), and (IV-21) except that $c_{6,5}^* = \infty$ and $\hat{c}_{6,5,1} = \infty$.

All changeovers to and from either job 5 or 6 must now be performed on processor 1. The adjustments of Step 5 to (IV-19), (IV-20), and (IV-21) yield

$$\tilde{C}^* = \begin{bmatrix} \infty & 2 & 3 & 1 & \infty & 0 & 0 \\ 0 & \infty & 0 & 2 & \infty & 5 & 1 \\ 0 & 5 & \infty & 1 & \infty & 8 & 6 \\ 0 & 0 & 2 & \infty & \infty & 5 & 5 \\ 2 & 6 & 8 & 3 & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 1 & 4 & 2 & \infty & \infty & \infty \end{bmatrix}, \quad (\text{IV-22})$$

$$\hat{\tilde{C}}_1 = \begin{bmatrix} \infty & 4 & 4 & 4 & \infty & 0 & \infty \\ 1 & \infty & 1 & 2 & \infty & 5 & \infty \\ 0 & 5 & \infty & 4 & \infty & 8 & \infty \\ 0 & 2 & 7 & \infty & \infty & 5 & \infty \\ 2 & 6 & 8 & 3 & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad (\text{IV-23})$$

$$\hat{C}_2 = \begin{bmatrix} \infty & 2 & 3 & 1 & \infty & \infty & 0 \\ 0 & \infty & 0 & 2 & \infty & \infty & 1 \\ 2 & 7 & \infty & 1 & \infty & \infty & 6 \\ 2 & 0 & 2 & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 1 & 4 & 2 & \infty & \infty & \infty \end{bmatrix}. \quad (\text{IV-24})$$

The only reduction on \hat{C}^* of equation (IV-22) required by Step 7 is in column four where each entry can be reduced by a constant one. \hat{C} given by (IV-23) and (IV-24) are similarly reduced. Therefore $b(T_i')$ = 17 + 1 = 18 as shown in Figure 3.

Step 8 determines that neither is T_i' a singleton subset $\{S\}$ nor is $b(T_i') = 18 < z_0 = \infty$. Additional iterations analogous to the one above are required. The results of these iterations are displayed in the tree of Figure 3. Note that the optimal schedule was found on the first pass but that four backtracks were required to prove optimality.

Distinct Processor Example Where N^* is to be Determined

Consider the same problem given by equations (IV-12)-(IV-15) under the assumption that the number of processors activated is unconstrained. The solution proceeds exactly the same except that the \underline{D} submatrices of \hat{C}_1 and \hat{C}_2 in equations (IV-16) and (IV-17) respectively have diagonal

entries of zero and off-diagonal entries of infinity. Therefore \underline{C}^* of (IV-18) is computed accordingly and the algorithm proceeds as usual. The tree representation of the solution is given in Figure 4.

The optimal solution was again found on the first pass of the procedure. However, only one backtrack was required to prove optimality. This reduced the amount of backtracking that was encountered for most problems where the number of processors was unconstrained (see Chapter VI). Also the total changeover time was only 16 units for the unconstrained solution compared to 21 units for the constrained problem.

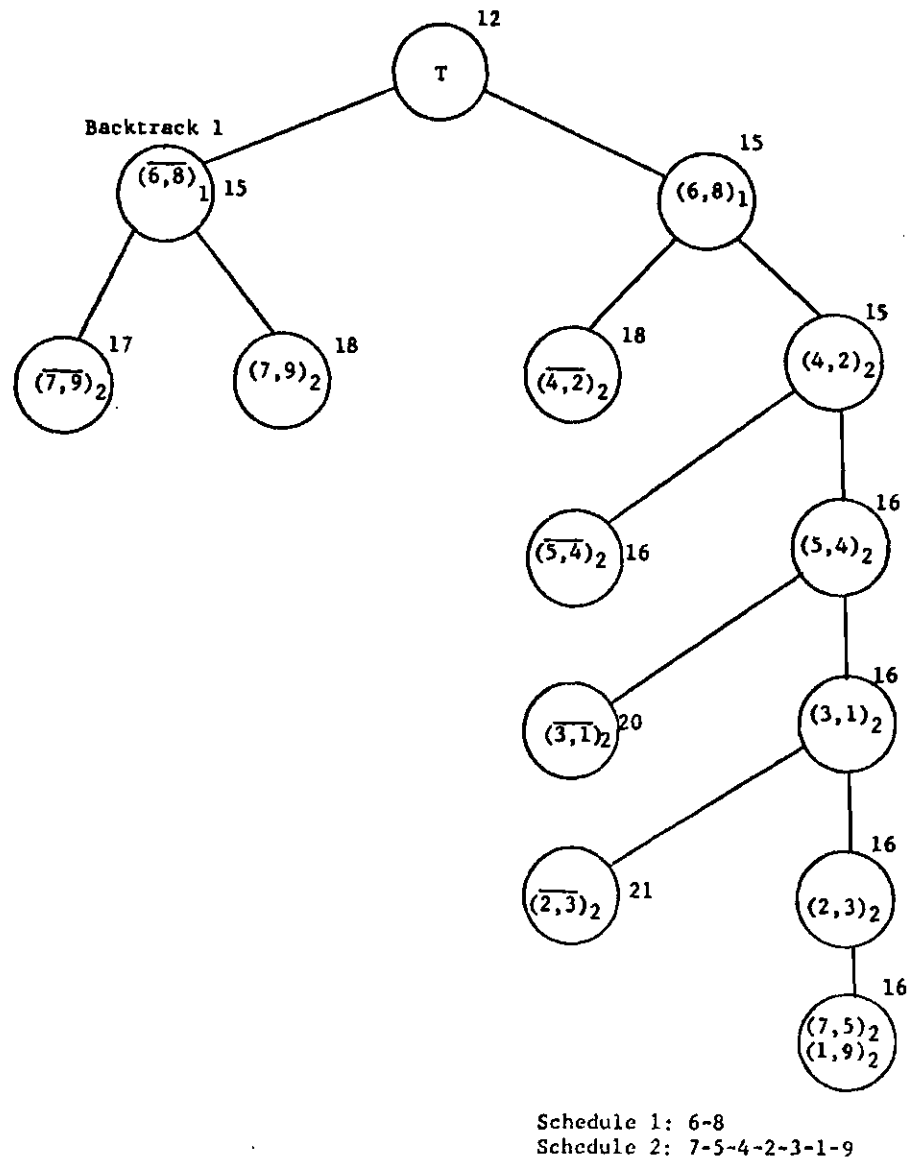


Figure 4. Tree Representation of Solution to the Distinct Processor Example Where N^* is to be Determined

CHAPTER V

SCHEDULING JOBS WITH SOME FINITE DUE DATES

The algorithm given in the previous chapter provides procedures to schedule jobs with all infinite due dates on parallel processors in such a way as to minimize total changeover time. The present chapter extends these algorithms to admit a set of M jobs for which there is at least one finite due date.

The minimization of total setup time is still the criterion, but each job must complete processing before its due date. If the individual job due dates are so restrictive that there exists no parallel processor schedule that meets all due dates, then there exists no feasible solution to the problem. Procedures for the situation where no feasible solution exists are beyond the scope of this study.

The previous algorithm considered the set of feasible schedules F to be imbedded in a larger superset T , and any admissible schedule was also a feasible schedule since all due dates were infinite. Extending the solution procedures to admit job due dates involves devising methods to identify those admissible solutions which are due date infeasible. Obviously, one way of doing this is to check each complete schedule to see if all due dates are met. However, this

approach is not efficient. Ideally, it is desired to eliminate entire subsets of solutions during early stages of the solution procedure. This can be done by identifying changeovers whose inclusion in a subset would result in the entire subset being infeasible. The present chapter develops tests to identify such infeasible changeovers. The basic approach underlying the feasibility tests developed below is due to the single processor results of Pierce and Hatfield [6]. However, in addition to extending their results to the multiprocessor problem, a new lower bounding scheme is given which significantly improves their basic concept.

Development of Feasibility Conditions

Let the M jobs be numbered in nondecreasing due date order so that for any two jobs i and j , $i < j$ implies that $d_i \leq d_j$. Let the notation $[i]_n$ denote the job which has the i th smallest due date of all jobs currently assigned to processor n by some subset T_i at a given stage of the solution procedure. For example, if for T_i at some stage of the algorithm jobs 3, 5 and 10 with respective due dates 12, 13 and 20 are assigned to processor 5, then $[1]_5 = 3$, $[2]_5 = 5$ and $[3]_5 = 10$. Also $d_{[1]_5} = d_3 = 12$, $d_{[2]_5} = d_5 = 13$ and $d_{[3]_5} = d_{10} = 20$. Finally, let k_n be the number of jobs currently assigned to processor n in subset T_i at some stage of the algorithm.

Necessary Conditions

A condition necessary for the feasibility of any solution $\tilde{S} \in T_i$ is the following:

Condition 1. If there exist changeovers $\bigcap_{\tilde{S} \in T_i} \tilde{S}$ that require jobs $[1]_n, [2]_n, \dots, [k_n]_n$ to be performed on processor n , then each $\tilde{S} \in T_i$ is feasible only if

$$\sum_{j=1}^v (b[j]_{n,n} + p[j]_{n,n}) \leq d[v]_n, \quad 1 \leq v \leq k_n,$$

where $b_{j,n} = \min_{1 \leq i \leq N} c_i[j]_{n,n}$

A schedule \tilde{S} is feasible only if all job due dates are met. That is, all processing on a given job must be completed before its due date for feasibility. Furthermore, for any job $[v]_n$, all processing on jobs $[1]_n, [2]_n, \dots, [v-1]_n$ must also be completed before $d[v]_n$ since $d[1]_n \leq d[2]_n \leq \dots \leq d[v]_n$. Suppose there exists some u , $1 \leq u \leq k_n$ for which

$$\sum_{j=1}^u (b[j]_{n,n} + p[j]_{n,n}) \geq d[u]_n.$$

Note that $b[j]_{n,n}$ is a lower bound on the cost of any changeover to job $[j]_n$. Therefore it is impossible to meet job $[u]_n$'s due date unless Condition 1 is met.

The above condition is clearly not sufficient because the jobs already assigned to processor n will, in

general, be a combination of jobs such that the lower bound $b_{[j]_n, n}$ on the cost of the changeover to the job on processor n with the j th smallest due date will not be realized. In fact, knowledge of the changeovers already included in a subset T_i can be used to make the necessary condition more efficient by tightening the lower bounds $b_{[j]_n, n}$.

Suppose a changeover $(w, [j]_n)_n \in \bigcap_{S \in T_i} S$ so that $[j]_n$ must be processed on n . Then it is known that a cost

$c_{w[j]_n, n}$ must be incurred instead of $\min_{1 \leq i \leq N} c_{i[j]_n, n}$.

In general

$$c_{w[j]_n, n} \geq \min_{1 \leq i \leq N} c_{i[j]_n, n}.$$

Therefore $b_{[j]_n, n}$ can be made more efficient (larger) by redefining it as

$$b_{[j]_n, n} = \begin{cases} c_{w[j]_n, n} & \text{if } (w, [j]_n)_n \in \bigcap_{S \in T_i} S \\ \min_{1 \leq i \leq N} c_{i[j]_n, n} & \text{otherwise.} \end{cases}$$

Thus, the quantity

$$e_{v, n} \equiv \sum_{j=1}^v (b_{[j]_n, n} + p_{[j]_n, n})$$

is a lower bound on the time required for both processing and changeovers for jobs $[1]_n, [2]_n, \dots, [v]_n$.

The amount of "slack time" $q_{[v]_n, n}$ available before time $d_{[v]_n}$ for making changeovers $(i, [j]_n)_n$ for $j \leq v$ whose costs $c_{i[j]_n, n}$ are larger than the lower bound $b_{[j]_n}$ and for changing over for and processing jobs $[v+1]_n, \dots, [k]_n$ is important in feasibility tests.

To illustrate the concept, suppose that $T_i = \{\underline{S} \mid (3,5)_2 \in \underline{S}\}$ so that job 3 and job 5 must be completed on processor 2. Figure 5 shows a typical relationship between the time data. In this situation, $q_{5,n} = d_5 - e_{5,n}$ would obviously be available before d_5 and $q_{3,n} = \min(q_{5,n}; d_3 - e_{3,n})$ would be available before time d_3 . Figure 6 shows a different but also typical relationship between the data. In this case a certain amount of job 5's processing must occur before time d_3 in order that job 5's due date d_5 be met. However,

$$q_{5,n} = d_5 - e_{5,n} \text{ and } q_{3,n} = \min(q_{5,n}; d_3 - e_{3,n}).$$

In general

$$q_{[v]_n, n} = \min[q_{[v+1]_n, n}; d_{[v]_n} - e_{[v]_n, n}].$$

Using the concept of slack time $q_{[v]_n, n}$, feasibility Condition 1 can be restated and a number of feasibility tests can be stated to identify infeasible changeovers $(i, j)_n$ for a given processor n :

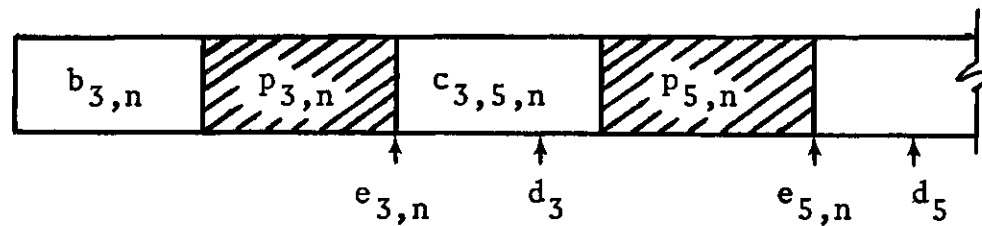


Figure 5. Typical Relationships Between Time Data When Processing on a Job Begins After the Previous Job's Due Date

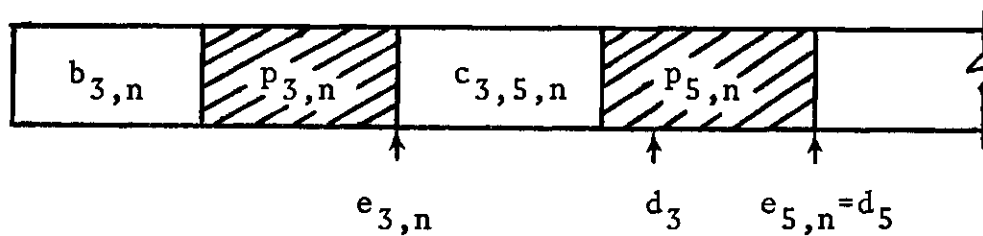


Figure 6. Typical Relationships Between Time Data When Processing on a Job Must Begin Before the Previous Job's Due Date

Condition 1'. If there exist changeovers $\{(i,j)_n\} \in \bigcap_{S \in T_i} S$ that require jobs $[i]_n, [2]_n, \dots, [k]_n$ to be performed on processor n , then each $S \in T_i$ is feasible only if

$$q_{[i]_n, n} \geq 0 \quad i = 1, \dots, k_n.$$

Condition 2. Let $[i]_n$ and $[j]_n$ be any two jobs currently assigned to processor n such that $1 \leq i < j \leq k_n$ and for which $q_{[i]_n, n} < q_{[j]_n, n}$. If $b_{[j]_n, n} + p_{[j]_n, n} > q_{[i]_n, n}$ then job $[j]_n$ cannot precede job $[i]_n$ in any sequence on processor n . Also since $q_{[v]_n, n} \leq q_{[i]_n, n}$ for $v < i$, then all changeovers $([j]_n, [v]_n)_n$ are infeasible for $1 \leq v \leq i$.

Condition 2 follows from the fact that, under the stated assumptions, $e_{[i]_n, n} > d_{[i]_n}$ if $[i]_n$ follows $[j]_n$ on processor n when $b_{[j]_n, n} + p_{[j]_n, n} > q_{[i]_n, n}$.

Condition 3. Under the assumptions of Condition 2, changeovers $([r]_n, [j]_n)_n$ are infeasible for $1 \leq r \leq i$ if $e_{[i]_n} > d_{[r]_n}$.

Condition 3 follows from the requirement that jobs $M+n, [1]_n, \dots, [r-1]_n, [r+1]_n, \dots, [i-1]_n, [i]_n$ must be completed before job $[r]_n$ and the earliest time at which this total processing could be complete is e_i .

Condition 4. For any job, $[i]_n, 1 \leq i < k_n$, currently assigned to processor n , the $([i]_n, M+N+n)_n$ changeover is infeasible if $e_{[k]_n, n} > d_{[i]_n}$.

Condition 4 is obvious from the fact that jobs $[1]_n, \dots, [k_n]_n$ must be included in processor n 's schedule so that $e_{[k_n],n}$ is a lower bound on the completion time for all k_n jobs.

Condition 5. For jobs $[i]_n$ and $[j]_n$, $1 \leq i < j \leq k_n$, if $q_{[i]_n,n} < q_{[j]_n,n}$ and if $b_{[j]_n,n} + p_{[j]_n,n} > q_{[i]_n,n}$, then $([j]_n, [i]_n)_n$ is infeasible.

Condition 5 states that if job $[j]_n$ is to be processed before job $[i]_n$, then the lower bound $b_{[j]_n,n} + p_{[j]_n,n}$ on changing over to and processing job $[j]_n$ must be less than the slack available before job $[i]_n$'s due date.

Condition 6. If $[i]_n$ and $[j]_n$ are two jobs $1 \leq i < j \leq k_n$ for which $q_{[i]_n,n} = q_{[j]_n,n}$ and $d_{[i]_n} < d_{[j]_n}$ then the $([j]_n, [i]_n)_n$ changeover is infeasible if $b_{[j]_n,n} + p_{[j]_n,n} + e_{[i]_n,n} > d_{[i]_n}$.

Condition 6 states that if a lower bound $p_{[j]_n,n} + b_{[j]_n,n} + e_{[i]_n,n}$ on the time required to process and make the required changeovers for jobs $[i]_n$ and $[j]_n$ under the stated assumptions is greater than $d_{[i]_n}$, then $([j]_n, [i]_n)_n$ is infeasible.

Condition 7. If $[i]_n$ and $[j]_n$ are any two jobs $1 \leq i < j \leq k_n$ for which

$$\begin{aligned} \bar{d} &= \min \{d_{[j]_n}; d_{[i]_n} + c_{[i]_n[j]_n,n} + p_{[j]_n,n}\} \\ &= d_{[i]_n} + c_{[i]_n[j]_n,n} + p_{[j]_n,n} \end{aligned}$$

then $([i]_n, [j]_n)_n$ is infeasible if $e_{[v]_n, n} > d_{[i]_n}$ where $d_{[v]_n} \leq \bar{d} < d_{[v+1]_n}$ and $1 \leq v \leq i$.

Condition 7 simply states that, under the given conditions, the selection of the $([i]_n, [j]_n)_n$ changeover will cause some job to be late if the lower bound $e_{[v]_n}$ on the time required to process jobs $[1]_n, \dots, [v]_n$ is greater than job $[i]_n$'s due date.

Sufficient Condition

The above eight conditions are useful in identifying changeovers which, if added to any schedule in a given subset of solutions, would make that schedule infeasible.

If the subset of solutions contains a singular complete parallel processor schedule, then a condition sufficient for feasibility is that each job's due date be met.

Condition 8. If a complete parallel processor schedule \tilde{S} requires k_n jobs on processor n , $1 \leq n \leq N$, then \tilde{S} is feasible if and only if

$$c_{M+n, i_1, n, n} + p_{(i_1, n), n} \leq d_{i_1, n}$$

and

$$c_{M+n, i_1, n, n} + p_{(i_1, n), n} + \sum_{j=2}^v \{c_{i_{j-1}, n, i_j, n, n} + p_{(i_j, n), n}\} \leq d_{i_v, n}; \quad v = 2, \dots, k_n$$

$$n = 1, \dots, N$$

However, even though Condition 8 is both necessary and sufficient, the exclusive reliance on it would result in an inefficient scheme to isolate infeasible solutions since it relates to a singleton subset T_i .

Modification of Solution Procedure

Using Conditions 1-8, the algorithm developed in Chapters III and IV can be modified to schedule jobs with due dates on parallel processors. The required additions are as follows.

Step 7'. Check the changeover selected in Step 3 for feasibility using Conditions 1-7 given above. If the changeover is infeasible, set $b(T_i') = \infty$. If $T_i' = \{\underline{S}\}$, a single solution, check \underline{S} for feasibility using Condition 8. If \underline{S} is infeasible, set $b(T_i') = \infty$.

Illustrative Problems

Distinct Processor Example with Moderately Constraining Due Dates

Consider a $N = 2$ available distinct processor problem in which $M = 5$ jobs have the moderately constraining due dates $\underline{d} = \{d_j\} = [29, 38, 52, 59, 87]$. Let the processing time p_{jn} be given by

$$p = \{p_{jn}\} = \begin{bmatrix} 3 & 5 \\ 2 & 5 \\ 6 & 8 \\ 7 & 7 \\ 1 & 5 \end{bmatrix} .$$

For comparison, let the changeover time data be the same as the examples in Chapter IV given by equations (IV-12) - (IV-15) and assume that N^* is to be determined.

The algorithm of Chapter IV augmented by Step 7' is appropriate. The solution of the present example proceeds exactly as the example in the previous chapter except for the execution of Step 7'. Therefore the tree in Figure 7 describing the present solution begins exactly like the tree of Figure 4, which relates to the previous example. In solving the present problem Step 7' states that each pair $(i,j)_n$ selected is to be tested for feasibility using feasibility conditions 1-7 of this chapter. This involves inspecting the branch to which the pair $(i,j)_n$ selected by Step 3 is appended to determine all jobs assigned to processor n by that branch. The lower bounds $e_{j,n}$ on completion time for each job j on processor n and the slack times $q_{j,n}$ for each job j on processor n are computed and Conditions 1-7 are tested.

The first changeover selected by Step 3 involving real jobs is $(4,2)_2$, the second pair selected after initialization.

Therefore

$$e_{2,2} = b_{2,2} + p_{2,2} = c_{4,2,2} + p_{2,2} = 3+5 = 8$$

and

$$\begin{aligned} e_{4,2} &= e_{2,2} + b_{4,2} + p_{4,2} = e_{2,2} + \min_i c_{i,4,2} + p_{4,2} \\ &= 8+5+7 = 20. \end{aligned}$$

Then

$$q_{4,2} = d_4 - e_{4,2} = 59 - 20 = 39$$

and

$$q_{2,2} = \min \{q_{4,2}; d_2 - e_{2,2}\} = \min \{39; 52 - 8\} = 39.$$

With these data, Tests 1-7 can easily be made and the $(4,2)_2$ changeover cannot be identified as infeasible by any of the necessary conditions.

After $(4,2)_2$ passed the necessary conditions, branching continued to the right in Figure 7 and all nodes passed the necessary conditions until the first complete solution $S = \{6,8; 7,5,4,2,3,1,8\}$ is reached. This solution failed the sufficient condition in Step 7' as illustrated in Figure 8

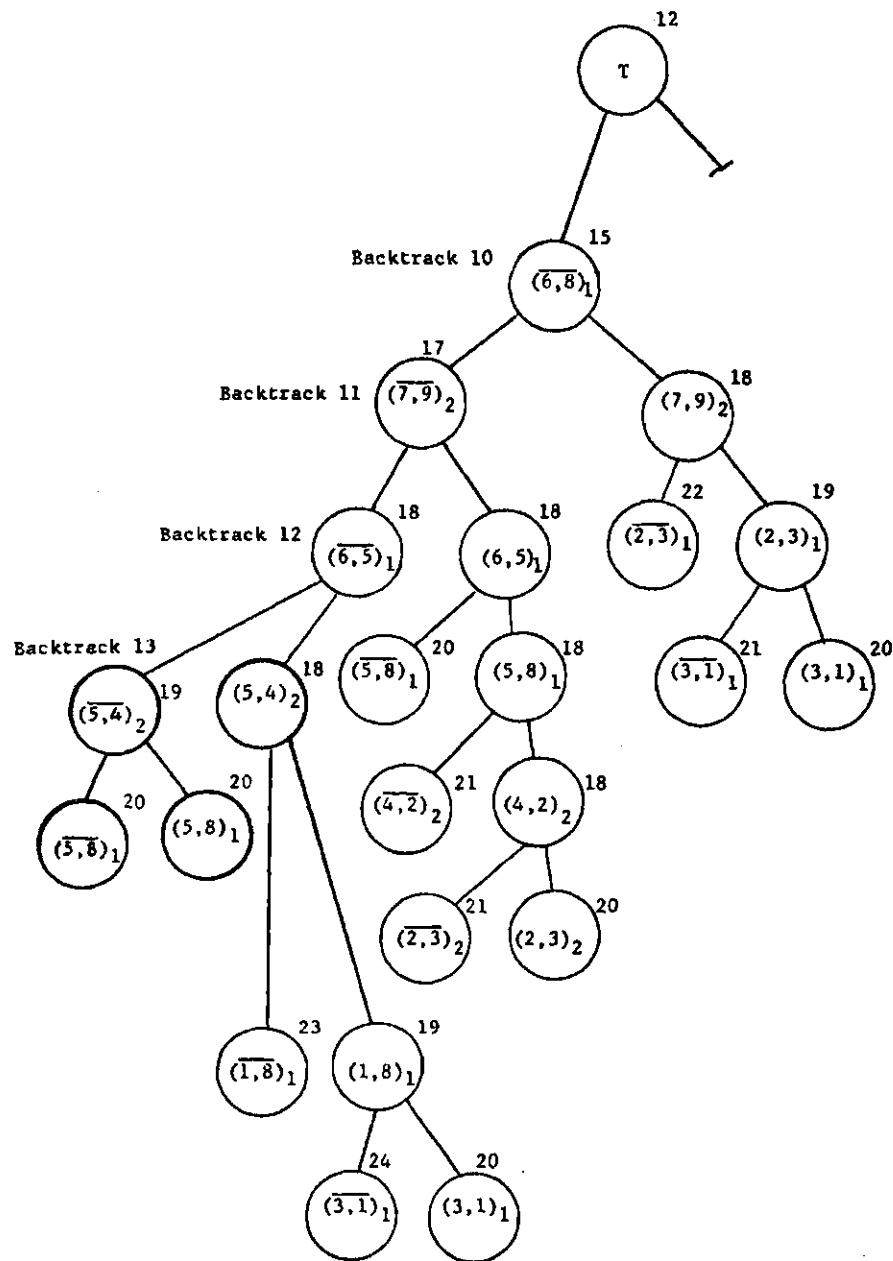


Figure 7. Tree Representation of the Solution to the Distinct Processor Example with Moderately Constraining Due Dates (continued on next page)

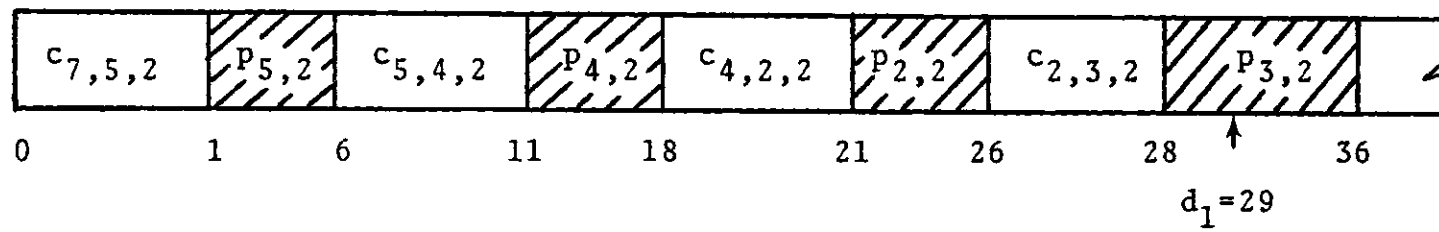


Figure 8. Partial Schedule $\{(7,5), (5,4), (4,2), (2,3)\}$ of S_2

because by time $d_1 = 29$, job 1 had not been processed.

Note from Figure 7 that the optimal schedule $\underline{S} = \{6,8; 7,1,5,4,2,3,9\}$ was found after only two backtracks but that 13 more backtracks were required to prove optimality. Also note that all of the necessary conditions were met at each stage of the branching.

Distinct Processor Example with Highly Constraining Due Dates

The utility of the necessary conditions can be illustrated by resolving Example 1 with the rather restrictive due dates $\underline{d} = \{d_j\} = [18, 30, 49, 61, 72]$.

The tree representation of the solution is given in Figure 9, where some uninteresting branches have been omitted. The first complete solution $\underline{S} = \{6,8; 7,5,4,2,3,1,9\}$ is infeasible by Condition 8.

When $(4,2)_2$ and $(5,1)_2$ are specified for processor 2, condition 2 is failed for $i = 1, j = 5$. Here jobs 1, 2, 4 and 5 are assigned to processor 2 and the lower bounds on completion times are

$$e_{1,2} = c_{5,1,2} + p_{1,2} = 8+5 = 13$$

$$e_{2,2} = e_{1,2} + b_{2,2} + p_{2,2} = 13+3+5 = 21$$

$$e_{4,2} = e_{2,2} + b_{4,2} + p_{4,2} = 21+5+7 = 33$$

$$e_{5,2} = e_{4,2} + b_{5,2} + p_{5,2} = 33+1+5 = 39.$$

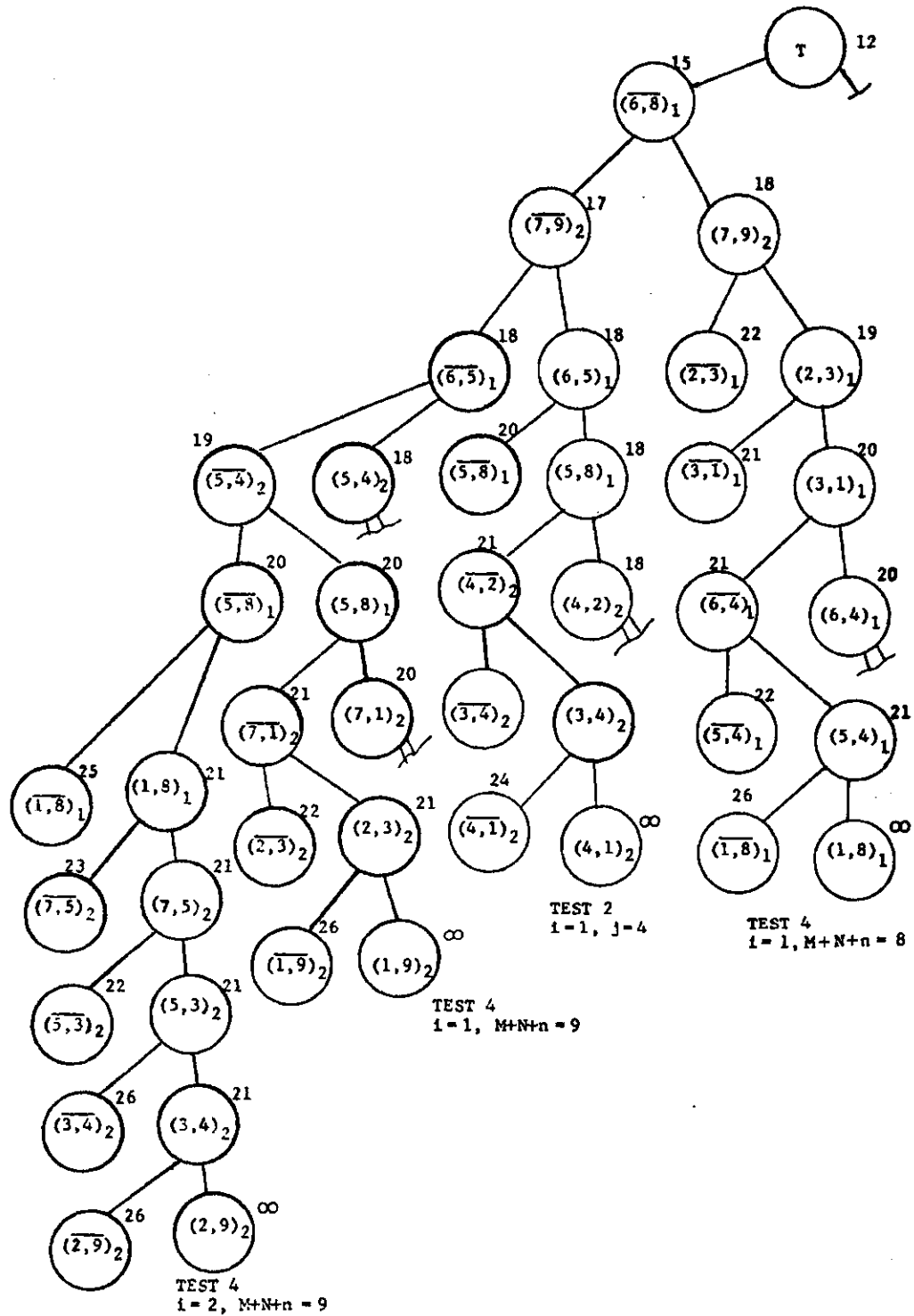


Figure 9. Tree Representation of the Solution to the Distinct Processor Example with Restrictive Due Dates (continued on next page)

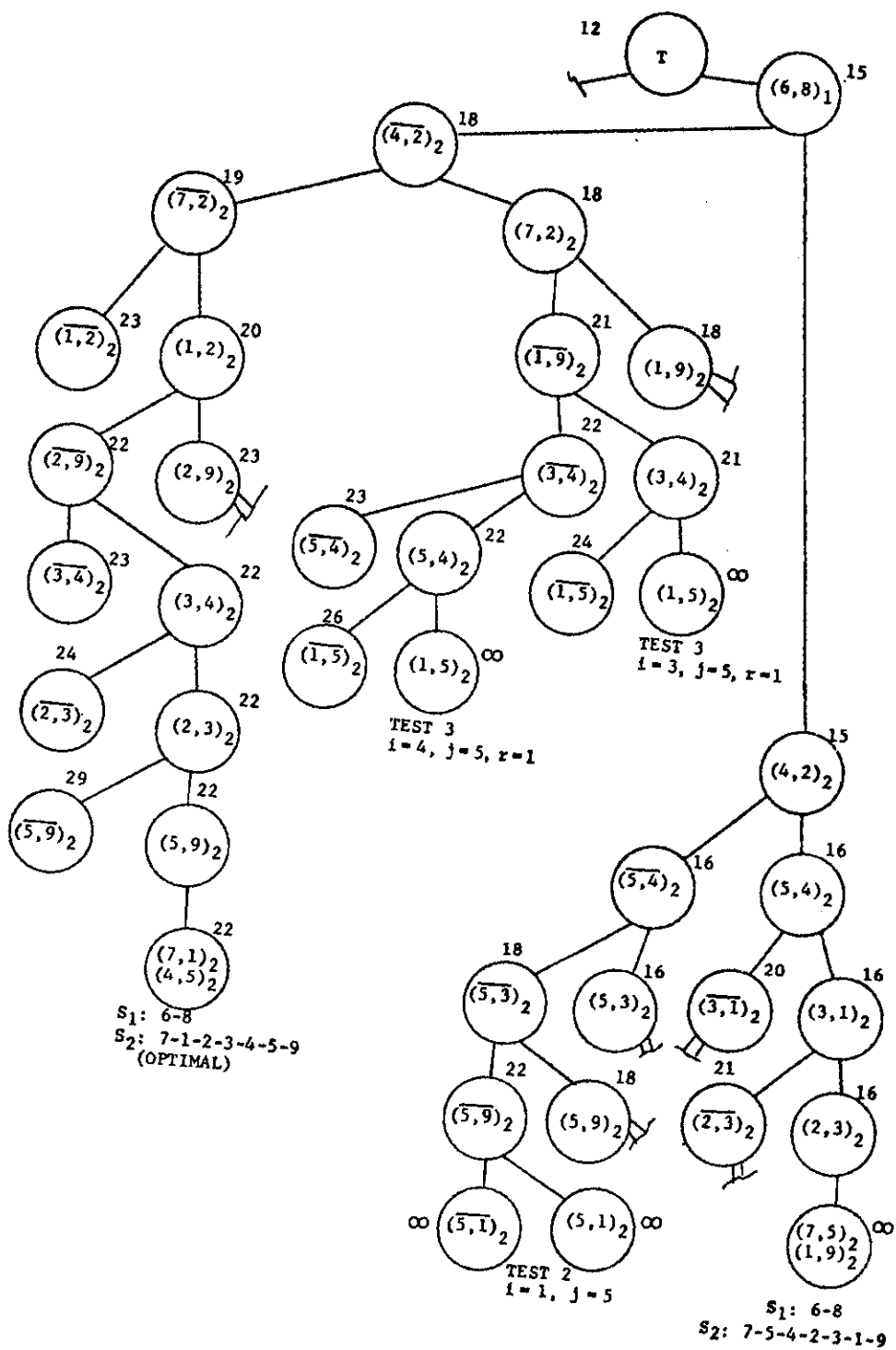


Figure 9. (Concluded)

The slack times are

$$q_{5,2} = d_5 - e_{5,2} = 72 - 39 = 33$$

$$q_{4,2} = \min \{q_{5,2}; d_4 - e_{4,2}\} = \min \{33, 28\} = 28$$

$$q_{2,2} = \min \{q_{4,2}; d_2 - e_{2,2}\} = 9$$

$$q_{1,2} = \min \{q_{2,2}; d_1 - e_{1,2}\} = 5$$

Obviously, the assignments pass Condition 1 since $q_{1,2}, q_{2,2}, q_{4,2}, q_{5,2} > 0$. However, for job 1 and job 5, $q_{1,n} < q_{5,n}$ and $b_{5,2} + p_{5,2} = 6 > q_{1,2} = 5$ and Condition 2 states that job 5 cannot precede job 1 on processor 2.

Figure 10 shows the infeasibility of this assignment which places the earliest time at which job 1 could be completed at time $t = 19$, which is past its due date of time $t = 18$.

The other tests involved in the tree in Figure 9 can be explained similarly. For example, consider the branch which assigns changeovers $(7,2)_2$, $(3,4)_2$, and $(1,5)_2$ to processor 2 and which fails Condition 3 for $i = 3$, $j = 5$ and $r = 1$. Figure 11 shows the relationship of the data. In this case $q_{3,2} = 24$ and $q_{5,2} = 29$. Test 3 states that the $(1,5)_2$ changeover is infeasible. Figure 11 confirms this by inspection since processing on jobs 2 and 3 would have to be completed before job 1 and this processing cannot be complete before $e_3 = 24$ while $d_1 = 18$.

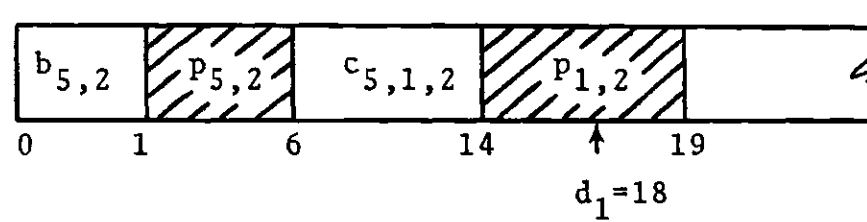


Figure 10. Partial Schedule $\{(5,2), (2,1)\}$ of S_2

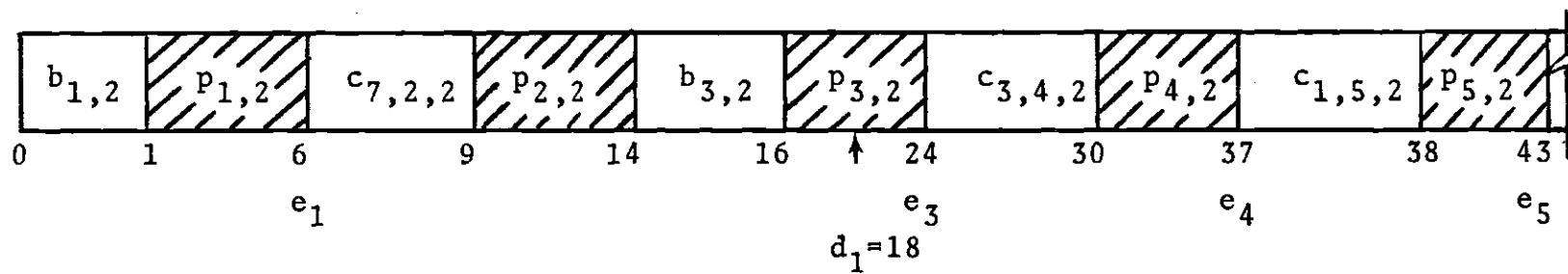


Figure 11. Relationships Between Time Data for S_2
Illustrating Infeasibility

CHAPTER VI

COMPUTATIONAL EXPERIENCE

The algorithm developed in the previous chapters has been coded in FORTRAN V for the Univac 1108. The computer code is given in Appendix A, and incorporates features to take advantage of the structure of the problem being solved by suppressing certain operations when they are not required, e.g., suppressing due date tests when all $d_j = \infty$.

Computational experience concentrated on general cases of the problem under assumptions which typically adversely affect the computing times of branch and bound algorithms. These results are as follows.

Results for Distinct Processor Problems

For a given problem size, the largest number of admissible solutions apparently results when the processors are distinct and when N^* is to be determined. Size of the solution space is frequently an indicator of problem difficulty and computational experience was concentrated on the class of problems defined by this assumption.

Branch and bound algorithms incorporating similar branching and bounding schemes for related problems have shown the worst performance when the data had low variability

[6, 12]. Therefore, the computational experience was concentrated on problems with low changeover time variability.

Problems involving either 2, 3 or 4 processors and from 5 to 15 jobs, inclusive, were solved to investigate the algorithm performance with respect to this class of jobs. Five problems for each assumption on M and N were generated so that 165 problems were solved under the present assumptions. Low data variability was introduced by generating the changeover times for both real and dummy jobs for each problem from a discrete uniform [0, 10] distribution.

Figure 12 shows the computing times averaged over the five problems. (Computing time for each experiment is given in Appendix B.) The average computing times for a given number of processors N appears to lie along a straight line on the semi-log plot. This suggests that the average time t_{MN} in minutes, to find the optimal solution to an M job N processor problem is of the form

$$t_{MN} = e^{a_N} b_N^M, \quad (\text{VI-1})$$

where a_N and b_N are constants. The broken lines in Figure 12 are the least squares lines fitted to equation (VI-1) for a given N. The least squares estimates for a_N and b_N for the three curves are given in Table 1.

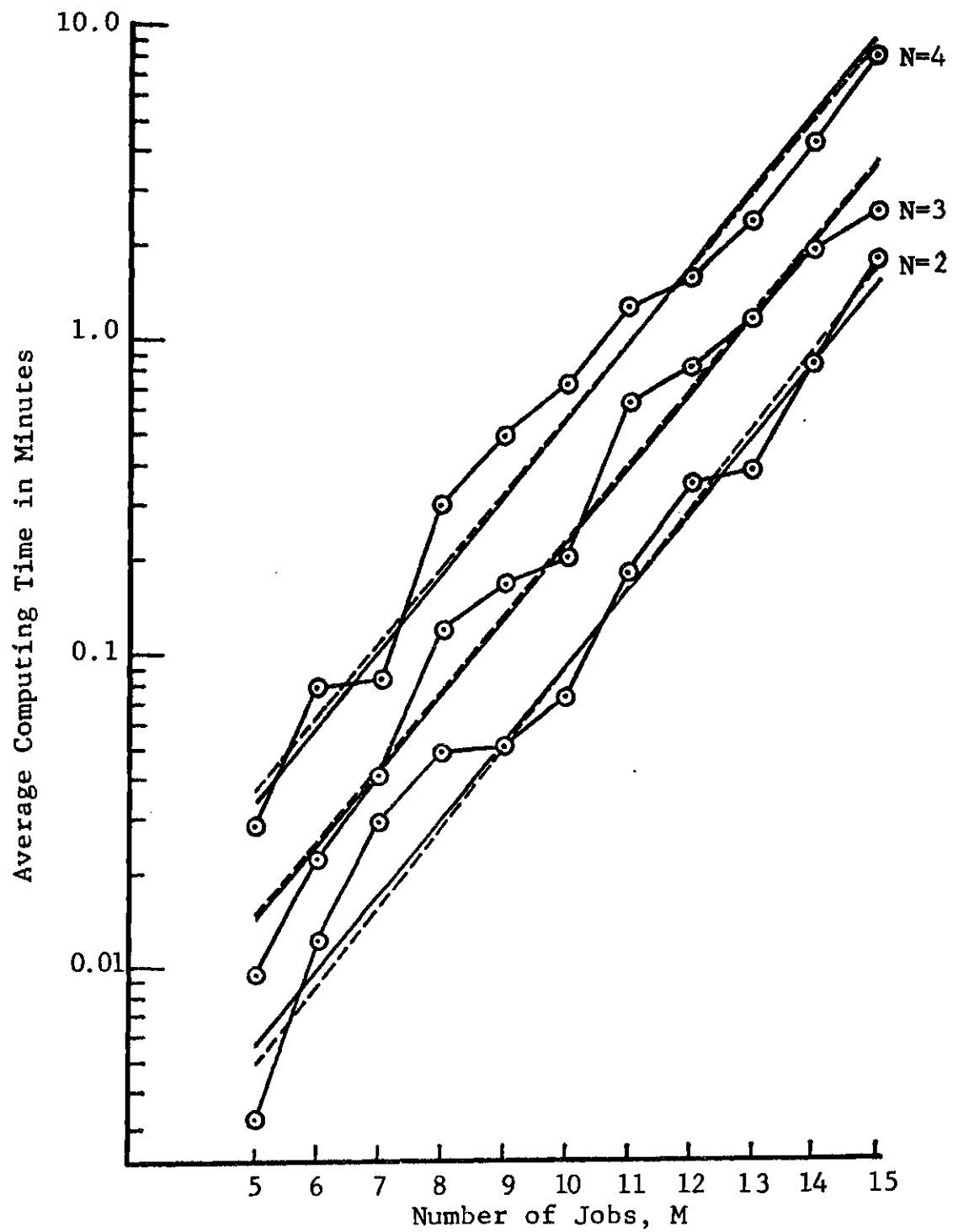


Figure 12. Average Computing Time and Least Squares Lines for Distinct Processor Problems

Table 1. Least Squares Estimates of a_N and b_N

N	\hat{a}_N	\hat{b}_N
2	-8.2022	1.7860
3	-6.9998	1.7390
4	-6.0188	1.7200

The broken lines have approximately equal slopes \hat{b}_N and intercepts $e^{\hat{a}_N}$ which place them an equal distance apart. Neither the null hypothesis that $\hat{b}_2 = \hat{b}_3$ or $\hat{b}_3 = \hat{b}_4$ can be rejected by a t-test (assuming normally distributed regression errors) at the 0.95 confidence level. Furthermore, both the null hypothesis that $e^{a_1} = e^{a_2}$ and $e^{a_2} = e^{a_3}$ are rejected by a t-test at the 0.95 confidence level.

These findings suggest that the average computing time t_{MN} in minutes is actually of the form

$$t_{MN} = e^{\alpha} \beta^M \gamma^N, \quad (\text{VI-2})$$

where α , β and γ are constants. Accordingly, all times in Figure 12 were used to determine the least squares equation

$$\hat{t}_{MN} = e^{-9.7752} (1.7480)^M (2.4600)^N. \quad (\text{VI-3})$$

The solid lines of Figure 12 are the family of lines of equation (VI-3).

Equation (VI-3) provides an adequate predictor of average computing time over the range of parameters studied. The coefficient of multiple determination of (VI-3) is $R^2 = 0.981$. However, it may be unwise to extrapolate (VI-3) to larger problems.

A limited number of additional distinct processor problems were solved under alternative assumptions. Fifteen problems were developed by generating job changeover times from a discrete uniform $[0, 20]$ distribution. Fifteen additional problems with all discrete uniform $[0, 10]$ were developed and solved under the assumption that the number of processors to be activated is specified to be N' . The computing times for these additional problems are given in Table 2 along with the comparable mean computing times from the previous problems. The number of additional problems solved is insufficient to allow a valid comparison of the computing times for the different classes of distinct processor problems. However, it is noted that all but three of the problems with discrete uniform $[0, 20]$ changeover times resulted in computing times below the average time required for problems with discrete uniform $[0, 10]$ changeover times. This is apparently not inconsistent with the experience of others [6, 12] with related algorithms.

Table 2. Computing Times for (a) Problems with Discrete Uniform [0,20] Changeover Times, (b) Problems where $N' = N$ Compared to (c) the Mean Computing Times for Previous Problems

Parameters		Problem Set - Computing Time		
M	N	(a)	(b)	(c)
5	2	.0023	.0015	.0033
8	2	.0487	.0616	.0491
10	2	.0389	.0252	.0734
12	2	.1884	.1603	.3404
15	2	.1696	2.0000	1.8112
5	3	.0067	.0041	.0096
8	3	.0650	.0597	.1192
10	3	1.5049	.3973	.2000
12	3	.7893	.6672	.8076
15	3	.4599	3.6773	2.5608
5	4	.0194	.0306	.0283
8	4	.3330	.1374	.2959
10	4	1.0180	.2410	.7204
12	4	1.3959	1.9668	1.5725
15	4	1.7158	2.0000	8.1371

Results for Problems with Due Dates

The effect of introducing due dates was investigated by solving each of the 55 previous $N = 2$ processor problems under two sets of due dates. That is, 110 problems with finite due dates were solved.

One set of due dates were such that they made only a moderate number of admissible solutions due date infeasible; i.e., the due dates were moderately constraining. These problems would therefore be problems of average difficulty as far as finding a solution which meets all due dates. These moderately constraining due dates were developed by generating the due date for job i from a discrete uniform $[20(i-1), 20i]$ distribution. All changeover times are discrete uniform $[0, 10]$. For convenience, all processing times were also generated from a discrete uniform $[0, 10]$ density. The expected processing time plus the expected changeover time to any job is 10 and the expected time to complete k jobs is $5 + 10k$. Since the expected due date of job i is $10i$, there are only a moderate number of sequences which satisfy job i 's due date.

The other set of due dates were such that a larger subset of the admissible solutions were due date infeasible. This was accomplished by generating job i 's due date from a discrete uniform $[10(i-1), 10i]$ distribution. Therefore, this set of due dates is described as highly constraining.

The average computing times for these problems are

illustrated in Figure 13 along with the average computing times for the same problems unconstrained by due dates. The average computing time for this class of due date problems does not appear to be explained by an equation of the form of (VI-1). Therefore the least squares fit to (VI-1) is not shown in the figure. Admitting moderately constraining due dates increases average computing time, although perhaps not significantly. However, a marked increase in average computing time results when the due dates are highly constraining.

Results for Identical Processor Problems

Computational experience was not concentrated on the identical processor case for the reasons previously enumerated. However, three sets of identical processor problems were developed and solved. Each set of identical processor problems contained 15 problems so that 45 identical processor problems were solved. In the first set, all job changeover times were generated from a discrete uniform $[0, 10]$ distribution and the problems were solved under the assumption that $N' = N$. The second set of problems was developed similarly, and they were solved under the assumption that N^* was to be determined. The third set of problems had discrete uniform $[0, 20]$ changeover times and these problems were solved under the assumption that $N' = N$.

The computing times for the identical processor

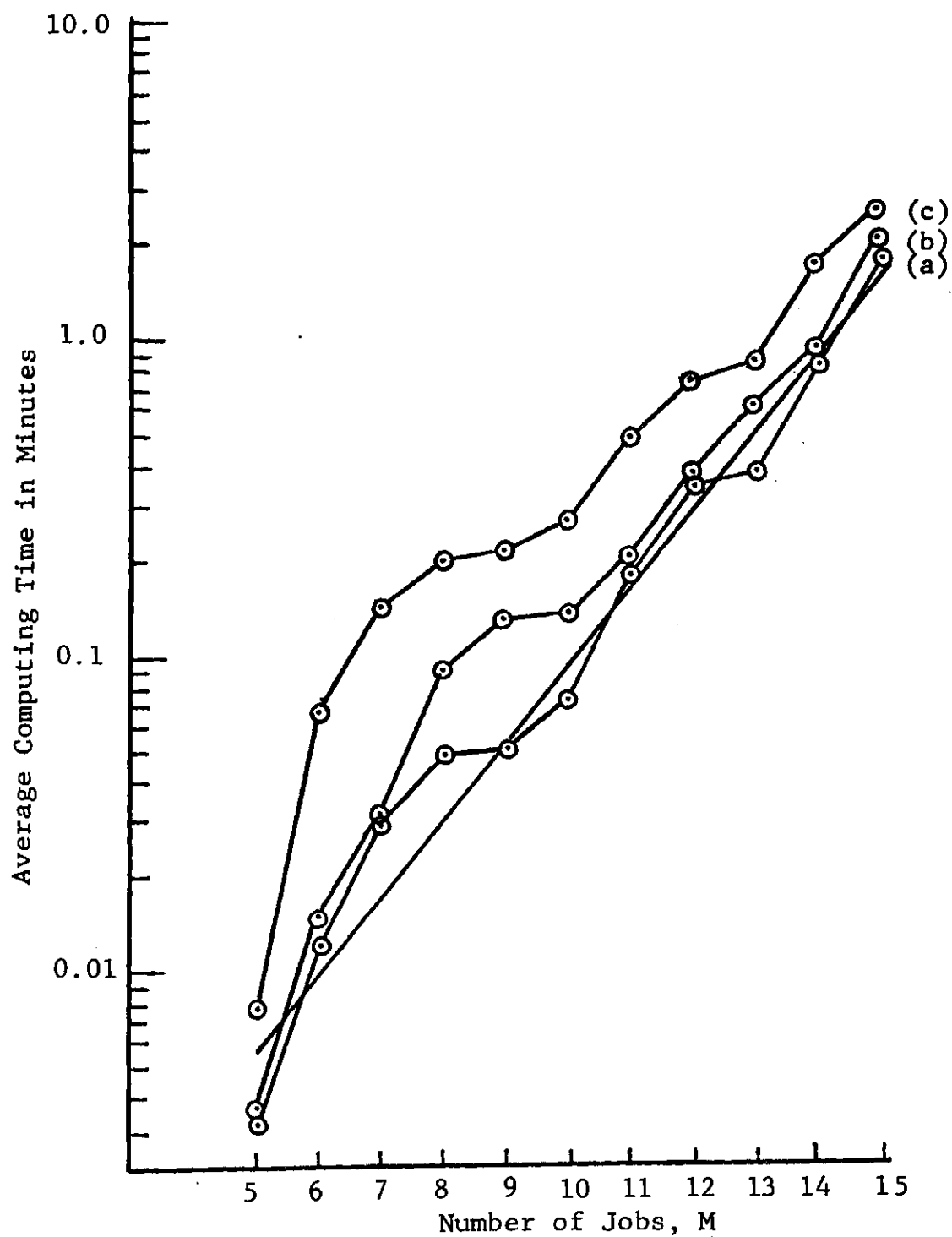


Figure 13. Average Computing Time for Distinct Processor Problems Where $N=2$ for (a) Infinite Due Dates, (b) Moderately Constraining Due Dates, and (c) Highly Constraining Due Dates

problems are given in Table 3. The number of experiments is inadequate to make valid conclusions. However, it is noted that relaxing the assumption that exactly $N' = N$ processors are to be activated resulted in reduced computing times in all but three problems. Also, increasing the changeover time variability resulted in lower computing times in all but one problem.

Table 3. Computing Times for Identical Processor Problems
 Where (a) $N' = N$, (b) N^* is to be Determined and
 (c) Changeover Times are Discrete Uniform $[0,20]$

Parameters		Problem Set - Computing Time		
M	N	(a)	(b)	(c)
5	2	.0027	.0026	.0014
8	2	.0334	.0117	.0030
10	2	.0852	.0276	.0233
12	2	.0325	.5080	.0985
15	2	3.2165	.1835	.5111
5	3	.0378	.0332	.0033
8	3	.0559	.2980	6.4790
10	3	.2750	.3020	.0220
12	3	3.5179	2.8700	.7984
15	3	2.9765	5.2765	.0656
5	4	.0915	.0855	.0266
8	4	2.3200	1.7233	.0854
10	4	5.7984	2.5321	.3302
12	4	8.3765	3.2625	.8249
15	4	9.4771	2.9870	2.0060

CHAPTER VII

HEURISTIC PROCEDURES FOR SCHEDULING PARALLEL PROCESSORS

The computational experience reported in the previous chapter indicates that exact procedures are probably computationally inefficient for many problems of moderate size. The present chapter develops and evaluates several heuristic procedures for solving larger scheduling problems. The procedures given below, except for random scheduling, incorporate heuristics which have shown promise in similar problems and therefore can be considered to be logical extensions of existing results.

The Heuristic Procedures

Random Scheduling

A simple heuristic is to generate a number of random solutions, using the best solution found. The procedure is computationally fast but it has certain disadvantages. Typical problems are structured such that there are only a few near-optimal solutions and the probability of generating one of these on a single iteration is quite small. When a large number of trials are performed to increase the probability of generating a good solution, the procedure becomes inefficient. However, it is included here because the results can be compared against more realistic heuristics.

Random scheduling involves randomly partitioning the M real jobs into \tilde{N} subsets, scheduling the jobs in each subset in random order and attaching an initial job and a final job to the \tilde{N} single processor schedules. If the problem involves determining the number of processors activated, \tilde{N} is a random integer, $1 \leq \tilde{N} \leq \tilde{N}$. Otherwise \tilde{N} is the specified number of processors. A FORTRAN V routine for random scheduling is given in Appendix C.

Shortest Changeover Next

Procedures which build schedules on the basis of shortest changeover next have performed well in the single machine problem involving jobs with sequence-dependent setup times [1, 34]. The shortest changeover next heuristic was extended to the parallel processor problem by successively finding single processor schedules S_n according to the following rules. If the processors are identical select the next (initially the first) processor n and find the minimum changeover time $c_{i^*,j^*,n} = \min_{i,j} \{c_{i,j,n}\}$ between real jobs. If the processors are distinct, select the minimum changeover time $c_{i^*,j^*,n} = \min_{i,j,n} \{c_{ijn}\}$ between real jobs over all processors. Job i^* is the first real job and job j^* is the second real job on the processor n . If g is the last job added to S_n , sequentially add jobs by selecting that changeover (g,k^*) for which $c_{g,k^*,n}$ is a minimum, $g \neq k^*$. Jobs are added to S_n until either a final job is selected for k^* , or until no more jobs can be added if a prescribed

number of processors are to be activated.

Changeover Imbedded in Minimum Time Subsequence Next

The shortest changeover next heuristic tends to be myopic in the sense that it does not consider the time effects of any additional changeovers necessitated by selecting the shortest changeover next. Determining a machine's schedule by successively adding the changeovers which necessitate subsequences of minimum time would seem to overcome this myopic tendency. This can be considered as a "look ahead" scheme. An additional logical basis for considering this heuristic is that an optimal schedule for M' jobs with sequence-dependent setup times on a single machine is a subsequence of M' jobs with minimum total changeover time. This optimality condition will hopefully be approached if changeovers incurring minimum time subsequences are added sequentially to a machine's schedule.

This heuristic was applied to the identical processor problem by selecting the next (initially the first) processor n and then letting the first changeover $(i^*, j^*)_n$ in \underline{S}_n between real jobs be that changeover such that the total subsequence time

$$c_{M+n, i^*, n} + c_{i^*, j^*, n} + \min_{k \neq i^*} \{c_{j^*, k, n}\}$$

is a minimum. If g^* is the last job added to \underline{S}_n , additional jobs are added by selecting that changeover (g^*, k^*) such

that the subsequence time

$$c_{g^*,k^*,n} + \min_{k \neq k^*} \{c_{k^*,k,n}\}$$

is a minimum. If k^* is a final job, it is understood that

$\min_{k \neq k^*} \{c_{k^*,k,n}\} = 0$. Jobs are sequentially added to S_n until either the final job $M+N+n$ is selected or until no more jobs can be added to S_n if a specified number of processors are to be activated. Since the processors are identical, individual schedules are developed in sequential order S_1, S_2, \dots

If the processors are distinct, the schedules are not developed in sequential order. The next processor n^* and the first changeover $(i^*,j^*)_{n^*}$ between real jobs on that processor is determined by selecting $(i^*,j^*)_{n^*}$ such that

$$c_{M+n^*,i^*,n^*} + c_{i^*,j^*,n^*} = \min_{k \neq i^*} \{c_{j^*,k,n^*}\}$$

is a minimum and then sequentially adding jobs as before. A FORTRAN V code for either this heuristic or the shortest changeover next heuristic is given in Appendix D.

Maximum Regret (Branch and Bound Without Backtrack)

Ashour, et al. [35] have reported good results using a tour-building scheme for the traveling salesman problem by linking at any stage those cities which would incur a maximum regret or alternate cost as defined by Little, et al. [12]. Ashour, et al. [35] also experimented with a look

ahead rule to break ties when there exist alternative maximum alternate costs.

The extension of the maximum regret heuristic to the parallel machine problem can best be described as branch and bound without backtrack. Successive changeovers are assigned to specific processors by selecting those admissible changeovers with maximum alternate cost computed by equation (IV-3).

This heuristic seems reasonable based on experience with the branch and bound algorithms developed in the previous chapters. The exact algorithms frequently find either an optimal or near optimal solution on the first iteration even though much backtracking may be required to verify optimality or to make a slight improvement for optimality. The steps of the solution procedure are exactly the same as those of the optimal algorithms, except that no backtracking is required. The FORTRAN V code in Appendix I for the exact algorithm incorporates an indicator variable to suppress backtracking, if desired.

Maximum Regret With Look Ahead

The look ahead scheme proposed by Ashour, et al. involves breaking ties between maximum alternate cost by selecting that changeover which provides the minimum cost reduction if that changeover were selected. That is, if there are ties for maximum alternate cost in Step 3 of the exact algorithm, select that changeover $(i^*, j^*)_n$ such that

$$\sum_{\substack{i \\ i \neq i^*}} [\min_{j \neq j^*} c_{ij}] + \sum_{\substack{j \\ j = j^*}} [\min_{i \neq i^*} \{c_{ij} - \min_{i \neq i^*} c_{ij}\}]$$

is minimum.

The above look ahead scheme is also imbedded in the FORTRAN V code in Appendix A and is controlled by an indicator variable.

Computational Experience

The five scheduling heuristics were evaluated by applying them to various sized parallel processor problems. Heuristic solutions were found for selected problems solved optimally in Chapter VI, allowing the quality of the heuristic solutions to be compared directly with the optimal solutions. Larger problems were also solved heuristically. These solutions were evaluated by approximating the distribution of total changeover time for each large problem and making comparisons in terms of the probability that a random solution yields a better solution. There appear to be significant differences in computing times for heuristic procedures, and computing times apparently do not increase exponentially with problem size.

In general, the maximum regret heuristic provided the best solutions, followed by maximum regret with look ahead, shortest changeover next, minimum time subsequence, and random scheduling.

Comparison With Optimal Solutions

The five heuristics were applied to the distinct processor problems where N^* is to be determined and where the changeover times are randomly selected from a discrete uniform $[0, 10]$ distribution. Table 4 compares these heuristic solutions to the optimal solutions determined by the exact algorithm. The maximum regret heuristic produced the solution nearest the optimal solution in all but one problem. The maximum regret heuristic found the optimal solution in two of the 15 distinct processor problems. Adding the look ahead feature to the maximum regret heuristic did not lead to improved solutions in the test problems.

The shortest changeover next heuristic produced an optimal solution for one of the 15 problems. It produced the same solution as the maximum regret heuristic for one other problem. The shortest changeover next heuristic found solutions of higher total changeover time for all other problems.

A slightly different pattern emerges when the heuristics are applied to identical processor problems. This is evident from Table 5 where the heuristic solutions are compared to the optimal solution for each of 15 identical processor problems. (It was assumed that N^* was to be determined and changeover times were discrete uniform $[0, 10]$.) Adding the look ahead scheme to the maximum regret heuristic provided a better solution to only one test problem.

Table 4. Comparison of Heuristic Solutions to Optimal Solutions of Distinct Processor Problems Where N^* is to be Determined and Where $c_{ijn} \sim U [0,10]$

Parameters		Scheduling Method - Changeover Time					
M	N	Optimal	Max. Regret	Max. Regret Look Ahead	Shortest Change Next	Min. Time Subseq.	Random
5	2	8	21	21	21	21	35
8	2	6	12	12	20	20	46
10	2	12	13	21	29	31	59
12	2	5	9	10	25	27	73
15	2	6	9	9	41	39	56
5	3	7	9	9	18	18	28
8	3	8	11	15	8	29	47
10	3	7	7	7	21	25	73
12	3	8	15	18	22	43	65
15	3	6	15	16	29	33	78
5	4	6	6	6	11	15	25
8	4	10	11	11	21	46	61
10	4	9	18	19	17	29	84
12	4	6	13	14	17	33	89
15	4	5	8	8	35	48	104

Table 5. Comparison of Heuristic Solutions to
Optimal Solution of Identical Processor
Problems Where N^* is to be Determined
and Where $c_{ijn} \sim U [0,10]$

Parameters		Scheduling Method - Changeover Time					
M	N	Optimal	Max. Regret	Max. Regret Look Ahead	Shortest Change Next	Min. Time Subseq.	Random
5	2	11	11	11	22	19	31
8	2	7	7	7	31	33	51
10	2	13	15	15	30	37	58
12	2	7	13	13	16	42	57
15	2	10	10	10	35	44	70
5	3	15	15	15	30	24	50
8	3	12	12	14	21	17	34
10	3	7	8	8	23	34	61
12	3	10	11	11	29	49	59
15	3	8	8	8	41	67	76
5	4	15	15	15	18	24	49
8	4	13	13	13	26	31	68
10	4	9	11	11	28	27	57
12	4	14	17	14	28	43	68
15	4	8	9	9	38	58	96

The difference in computing times for the five heuristics and the optimal procedure is illustrated in Figure 14 along with the least square fit to equation (VI-1). Although the figure only gives the computing times for distinct processor problems in which $N = 3$, the pattern is typical of computing times for other problems. It is noted that the heuristics' computing times are not ranked in the same order as the apparent goodness of their solutions. The typical relationship is that between the shortest changeover next heuristic and the maximum regret heuristic where an increase in computing time provides an improved solution. An anomalous relationship may exist between the maximum regret heuristic and the maximum regret with look ahead heuristic where the increased computing time leads to little or no improvement in the solution.

Computational Results for Large Problems

Some computational experiments with the heuristic procedures were performed on larger scheduling problems. The experiments were directed toward determining (a) the quality of the heuristic solutions, (b) whether the marked difference and linear trend in the computing times indicated in Figure 14 extrapolated to larger problems, and (c) whether the heuristic procedures were ranked the same with respect to quality of solution.

In order to answer these questions, 10 test problems involving up to 40 jobs and 10 processors were attempted.

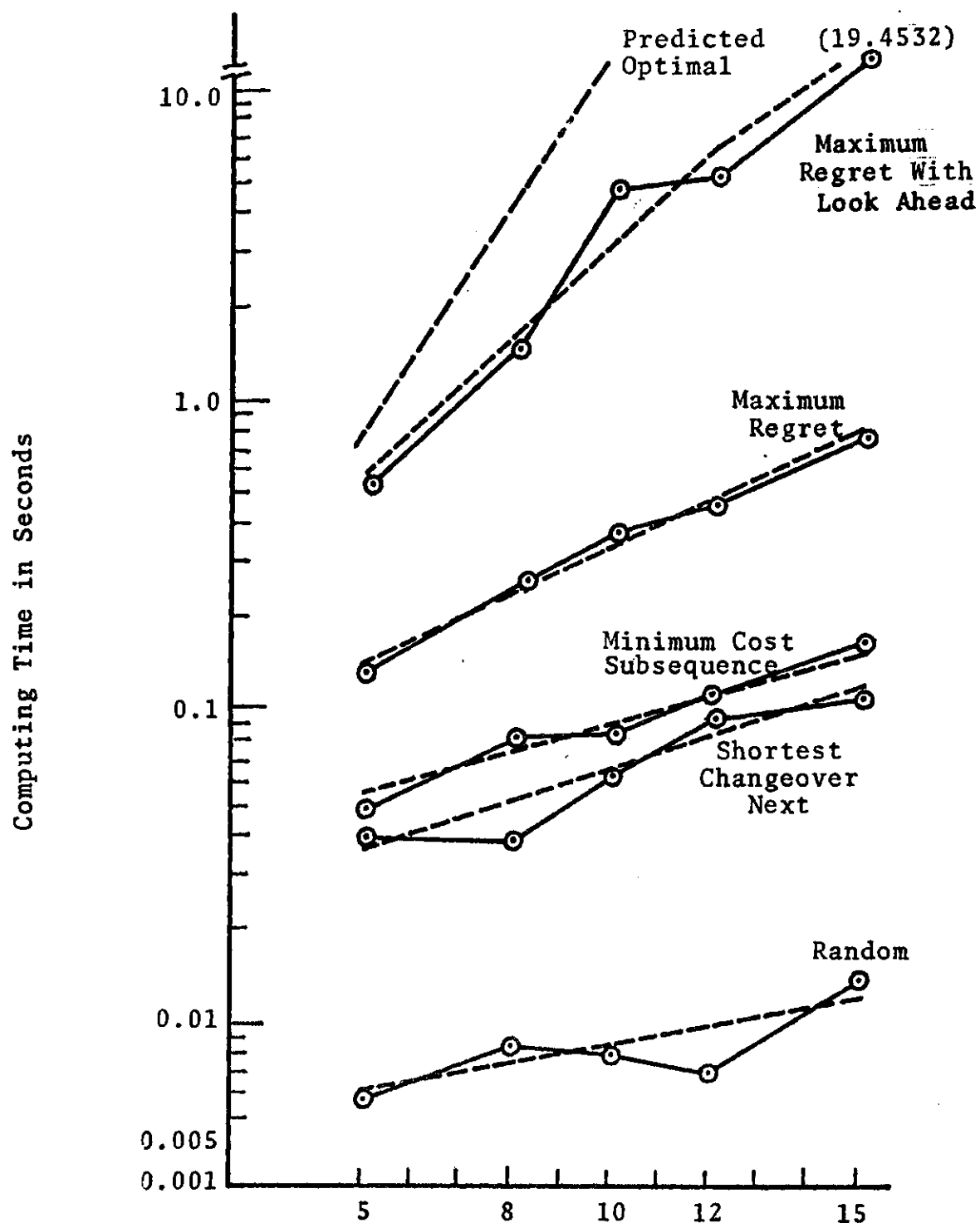


Figure 14. Computing Time for Alternative Scheduling Procedures for Distinct Processor Problems Where $N = 3$, N^* is to be Determined and $c_{ijn} \sim U[0,10]$

These problems were for the most general parallel processor case, involving distinct processors where N^* is to be determined. All changeover times were assumed to follow a discrete uniform $[0, 10]$ distribution and were generated accordingly.

The evaluation of the quality of the heuristic solutions is difficult because the optimal solutions are unknown. It was decided that some idea as to the distribution of total changeover time for each test problem would be helpful. Since M jobs are to be scheduled on N^* processors and since $2N^*$ initial and final jobs must be processed, the total changeover time is a random variable $X = X_1 + X_2 + \dots + X_{M+2N^*}$ where X_i is discrete uniform $[0, 10]$. Attempts to find the distribution function analytically for specific $j = M + 2N^*$ failed because of the discrete nature of the problem.

Therefore, following Lockett and Muhleman [34] on a related problem, schedules were developed at random for each problem and the total changeover time was computed. A histogram was developed from 1000 sampled schedules for each problem. For example, the sampled distributions of total changeover times for the $M = 25, N = 5$ problem and the $M = 30, N = 10$ problem are shown in Figure 15.

Total changeover time appears to be normally distributed. The sample mean and unbiased sample variance were used as estimates of the parameters for each of the 10 distributions.

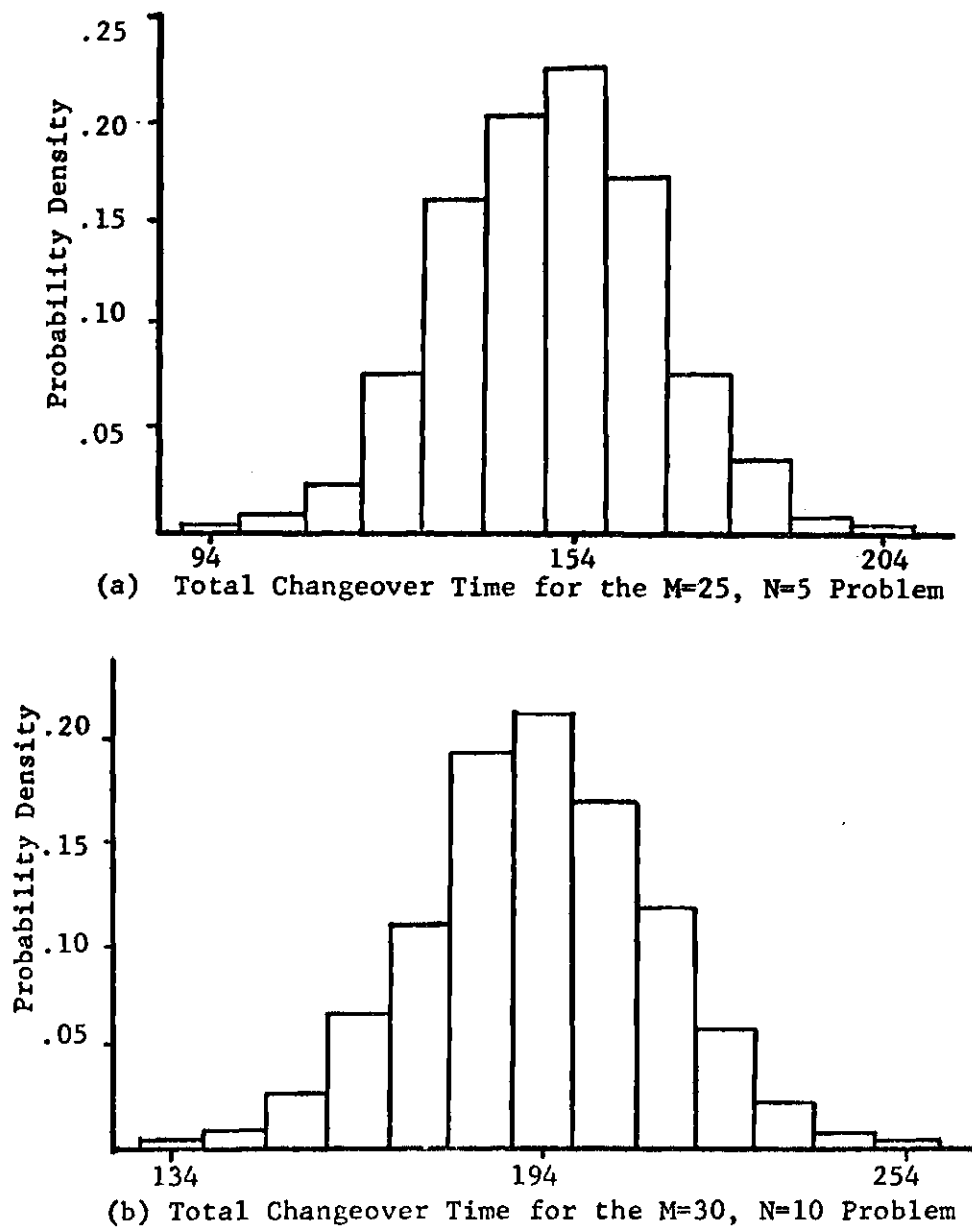


Figure 15. Distributions of Sampled Total Changeover Time

A chi-square goodness of fit test at the 0.95 confidence level did not reject the hypothesis that total changeover time was normally distributed for each problem.

The heuristic solutions to the 10 problems are given in Table 6, which gives the parameters of the sampled distributions. The maximum regret heuristic resulted in the solution with the least total changeover time in over 60% of the problems solved. Adding the look ahead scheme resulted in a lower time solution in three of the six schedules which it developed.

Table 7 expresses the heuristic solutions in standard deviations below the estimated mean total changeover time. The averages are a rough measure of overall goodness and indicate that the difference between maximum regret and shortest changeover next scheduling may be insignificant. However, even if the difference is significant, the cost of additional computing time may offset the reduction obtained in the objective function.

It appears that the same marked difference in computing times for heuristic solutions to large problems exists. This is illustrated in Figure 16 which shows the computing times when $N = 5$ fitted by least squares to equation (VI-1). The rate of growth with problem size appeared to be an extrapolation of the curves in Figure 14, indicating that heuristic solutions for large problems are obtained with approximately the same efficiency as for small problems.

Table 6. Comparison of Heuristic Solutions to Estimated Total Changeover Time Distribution for Selected Large Problems

Problem Number	Number of Jobs M	Available Number of Processors N	Heuristic Procedure - Total Changeover Time					Estimated Normal Parameters	
			Max. Regret	Max. Regret Look Ahead	Shortest Change-over Next	Min. Time Subseq.	Random	Mean	Standard Deviation
1	20	5	12	12	40	81	121	128.1	15.5
2	25	5	15	13	34	69	108	150.1	16.8
3	30	5	15	15	22	42	165	173.9	17.9
4	35	5	13	12	33	33	196	198.2	19.4
5	40	5	21	27	38	38	210	224.9	20.6
6	20	10	17	15	35	65	117	143.6	18.1
7	25	10	18	*	32	70	171	173.8	19.4
8	30	10	14	*	14	72	158	193.7	19.1
9	35	10	*	*	23	90	230	226.6	21.6
10	40	10	*	*	44	86	255	251.7	21.8

*Problem size exceeded allowed storage on the Univac 1108 used.

Table 7. Heuristic Solutions to Selected Large Problems,
Expressed in Deviations Below Estimated Mean
Total Changeover Time

Heuristic Procedure - Standard Deviations Below Sample Mean						Estimated Normal Parameters	
Problem Number	Max. Regret	Max. Regret Look Ahead	Shortest Change- over Next	Min. Time Subseq.	Random	Mean	Standard Deviation
1	7.52	7.52	5.68	3.04	0.46	128.1	15.5
2	8.05	8.16	6.92	4.82	2.50	150.1	16.8
3	9.43	9.43	8.48	7.36	0.50	173.9	17.9
4	9.55	9.61	8.52	8.52	0.11	198.2	19.4
5	9.88	9.60	9.04	9.04	0.72	224.9	20.6
6	7.00	7.10	6.00	4.35	1.47	143.6	18.1
7	8.03	*	7.30	5.35	0.14	173.8	19.4
8	9.40	*	9.40	6.37	1.87	193.7	19.1
9	*	*	9.41	6.32	-0.20	226.6	21.6
10	*	*	9.50	7.60	-0.15	251.7	21.8
Average	8.61	8.57	8.03	6.28	0.75	--	--

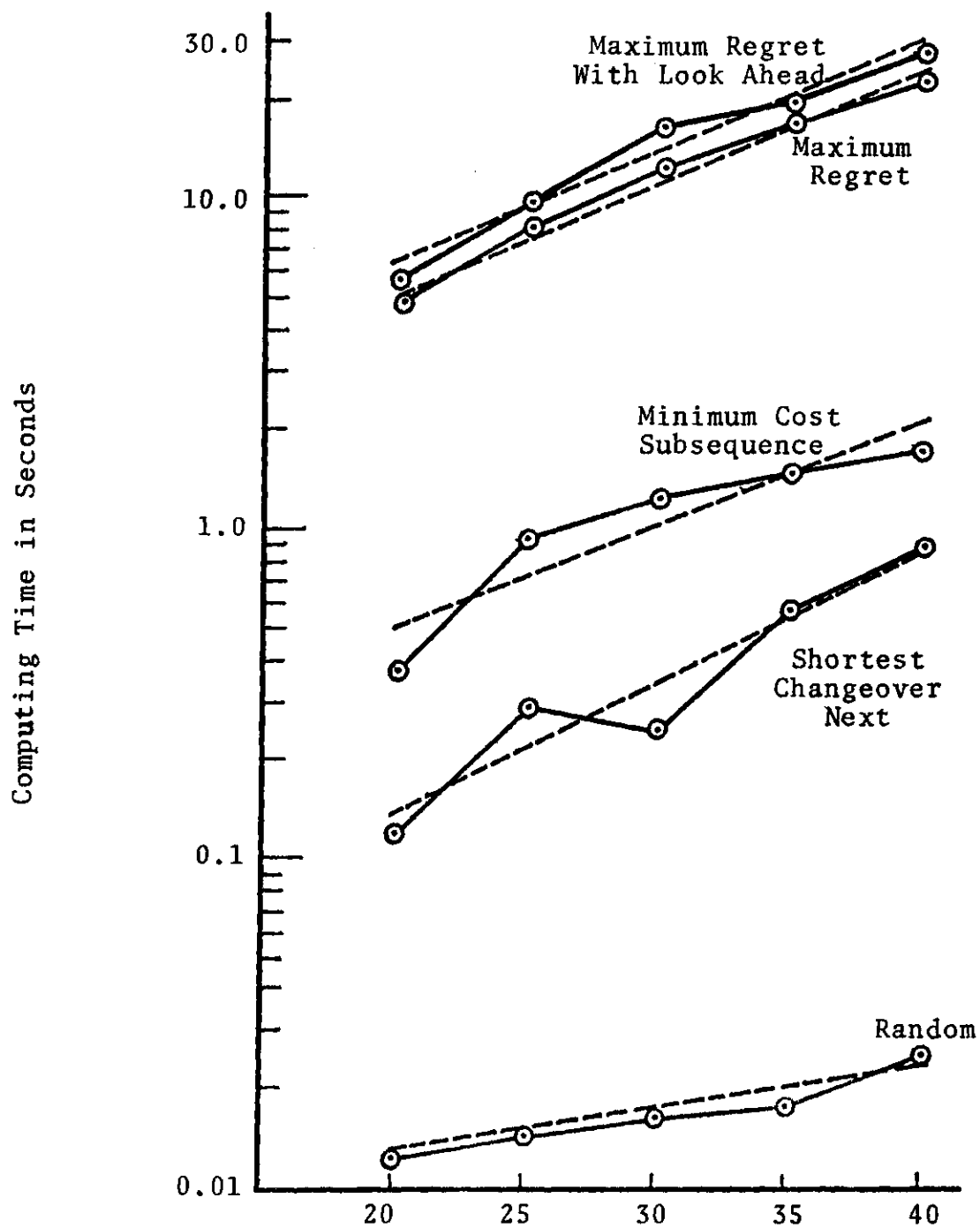


Figure 16. Computing Times for Heuristic Solutions to Selected Large Problems

CHAPTER VIII

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

A branch and bound algorithm was developed for the parallel processor scheduling problem. The algorithm admits parallel processor problems with finite job due dates and distinct processors. Unique features of the algorithm include (a) the lower bounding procedures used in identifying dominated subsets of solutions, (b) the sequential feasibility tests based on conditions necessary for both schedule admissibility and due date feasibility, and (c) a backtracking scheme which minimizes the amount of data required for the recursive operations.

A FORTRAN V computer code was developed for the algorithm and a number of computational experiments were performed. Computational experience was concentrated on distinct processor problems with low changeover time variability under three classes of due dates. Computing time increased exponentially with both M and N for this class of problems. A prediction equation for mean computing time was developed for the special case where all due dates are infinite. Relaxing the due dates decreased computing time, which was a minimum when all due dates are infinite.

A limited number of additional problems were solved. These included distinct processor problems with increased changeover time variability, distinct processor problems under alternative assumptions on the number of processors to be activated and identical processor problems under a variety of assumptions.

In general, optimal solutions appear to be elusive for many parallel processor problems because of either the problem size or structure. Several heuristic procedures were developed for this class of problems. These included adaptations of branch and bound without backtrack, branch and bound with look ahead and without backtrack, shortest changeover next, minimum time subsequence and (for comparison) random scheduling. The heuristic solutions were compared to optimal solutions for small problems and to the estimated distribution of changeover time for selected large problems. It was determined that the branch and bound without backtrack procedure provided good schedules with reasonable computing times.

Recommendations

Several ideas for extensions of the parallel processor results have evolved from the present investigation.

Relaxed Assumptions

Some of the assumptions underlying the present investigation could be relaxed to admit some important variations

of the parallel processor problem. Additional time constraints on both jobs and processors frequently exist. Processor n 's availability is more generally constrained to some time interval $[\alpha_n, \beta_n] \neq [0, \infty]$ to account for noncontinuous operation. In addition to due date d_j , job j generally has an arrival time a_j which is the earliest time that processing can begin. The branch and bound approach is still appropriate for these variations, but some significant extensions would be necessary.

Alternative Criteria

The minimization of total changeover time is taken as one of the most important criteria in scheduling environments [4,5]. There are, however, alternative criteria which could be equally important.

The minimization of total processor use time is appropriate for many problems, especially when the job processing times differ significantly from processor to processor. An important variation is the minimization of maximum completion time. Many of the algorithm components are applicable here, but the extension would rest on the development of efficient lower (for minimization) bounds on the objective function.

Due dates frequently cannot be met and there are a number of criteria that could be appropriate, depending on the penalty cost of late jobs. Ideally, a procedure to minimize job lateness on parallel processors should be

derived. This would provide a conceptual basis on which optimization procedures involving lateness could be developed.

In addition, there is a large class of possible objective functions which, in general, are functions of total processor use time. The branch and bound approach may extend to cases where the functions are linear or perhaps continuous, nondecreasing convex, since lower bounding would be fairly straightforward.

Recursive Operations

The development of alternative branch and bound recursive operations could be important in the parallel processor algorithm, and in similar algorithms. The most elusive development would be that of adaptive recursive operation. It appears that the efficiency of lower bounds is highly dependent on problem structure, which changes during the solution procedure due to partitioning. Also a flooding operation where large subsets with small lower bounds are investigated may be effective, depending on problem structure. The development of logical switching rules to evaluate the structure of the data at hand and then select the most appropriate recursive operation would be challenging.

Heuristic Procedures

The combinatorial nature of the parallel processor problem makes optimality a formidable goal in some cases. In view of this, new heuristics could be developed and the heuristic procedures of Chapter VII could be thoroughly

investigated and perhaps made more powerful (at the expense of computing time) by incorporating more powerful decision rules. Also, optimal stopping procedures could be developed by comparing the estimated computing time to find one more solution to the expected improvement in objective function.

APPENDIX A

FORTRAN V CODE FOR OPTIMAL, MAXIMUM REGRET,
AND MAXIMUM REGRET WITH LOOK AHEAD SCHEDULING

```

      INTEGER CCOST,FINDC,P,D,CDIMEN,THETA,CHOLD
      READ(5,99)M,N,NPRIME,NTYPE,LOOK,IOP,IDUE
99  FORMAT( )
      DIMENSION CCOST(25,25,5),CHOLD(25,25,5),
      XFINDC(25,25),P(15,5),J(15),KREDCU(25,25),NODE(25,25,50),
      XLBOUND(50),KHOLD(25,25),MJOB(25),IEK(25),IQK(25)
      CDIMEN=M+2*N
      INFIN=999
      ICOUNT=0
      IPAIRS=M+N-2
      NFLAG=0
      ITER=1
      CALL LOAD(NTYPE,M,N,CCOST,KREDCU,FINDC,P,D,CHOLD,IX,IS,IA,IB,IC,
      XID)
      NNODE=1
      LCOST=INFIN
190  FORMAT(1H0,'*****',*****,'*****')
      CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCU,ISUM)
      LBOUND(NNODE)=ISUM
      NNODE=NNODE+1
      LBOUND(NNODE)=ISUM
301  CALL ALTER(LOOK,NFLAG,NODE,NNODE,KREDCU,M,N,CHOLD,FINDC,THETA,
      XMROW,MCOL,ICOUNT)
      IF(THETA.EQ.(-1))GO TO 1305
      ICOUNT=ICOUNT+1
      NNEXT=NNODE+1
      LBOUND(NNEXT)=LBOUND(NNODE)
      DO 501 I=1,CDIMEN
      DO 501 J=1,CDIMEN
      NODE(I,J,NNEXT)=NODE(I,J,NNODE)
501  CONTINUE
      NODE(MROW,MCOL,NNODE)=- (100+FINDC(MROW,MCOL))
      LBOUND(NNODE)=LBOUND(NNODE)+THETA
      IF(LBOUND(NNODE).GT.999)LBOUND(NNODE)=999
      KPRSSR=FINDC(MROW,MCOL)
      CALL UPDAT1(KPRSSR,CDIMEN,MROW,MCOL,N,CHOLD)
      IF(IDUE.EQ.0)GO TO 801
      DO 709 I=1,CDIMEN
      MJOB(I)=0
709  CONTINUE
      MJOB(MROW)=1
      MJOB(MCOL)=1
      DO 713 I=1,CDIMEN
      DO 713 J=1,CDIMEN
      IF(NODE(I,J,NNEXT)-100,NE,KPRSSR) GO TO 713
      MJOB(I)=1
      MJOB(J)=1
713  CONTINUE
      DO 715 I=1,M
      IEK(I)=0
715  CONTINUE
      IFLAG=0
      IHOLD=0
      DO 735 I=1,M
      IF(MJOB(I).EQ.0) GO TO 735
      IF(I.NE.MCOL) GO TO 721
      IF(IFLAG.GT.0) GO TO 719
      IEK(I)=CCOST(MROW,MCOL,KPRSSR)+P(I,KPRSSR)
      IFLAG=1

```

```

      IHOLD=IEK(I)
      GO TO 735
719 IEK(I)=IHOLD+CCOST(MROW,MCOL,KPRSSR)+P(I,KPRSSR)
      IHOLD=IEK(I)
721 IFROM=0
      DO 723 K=1,CDIMEN
        IF(NODE(K,I,NNEXT)-100,NE,KPRSSR) GO TO 723
        IFROM=K
723 CONTINUE
        IF(IFROM.EQ.0) GO TO 727
        IF(IFLAG.GT.0) GO TO 725
        IEK(I)=CCOST(IFROM,I,KPRSSR)+P(I,KPRSSR)
        IFLAG=1
        IHOLD=IEK(I)
        GO TO 735
725 IEK(I)=IHOLD+CCOST(IFROM,I,KPRSSR)+P(I,KPRSSR)
        IHOLD=IEK(I)
        GO TO 735
727 IBK=999
        DO 729 J=1,CDIMEN
          IF(CCOST(J,I,KPRSSR)-IBK.GT.0) GO TO 729
          IBK=CCOST(J,I,KPRSSR)
729 CONTINUE
          IF(IFLAG.GT.0) GO TO 731
          IEK(I)=IBK+P(I,KPRSSR)
          IFLAG=1
          IHOLD=IEK(I)
          GO TO 735
731 IEK(I)=IHOLD+IBK+P(I,KPRSSR)
        IHOLD=IEK(I)
735 CONTINUE
        JOBS=0
        DO 736 I=1,CDIMEN
          JOBS=JOBS+MJOB(I)
736 CONTINUE
        DO 738 I=1,CDIMEN
          IQK(I)=0
738 CONTINUE
          IMARK=0
          IHOLD=0
          JIND=M+1
          DO 739 K=1,M
            LJOB=JIND-K
            IF(MJOB(LJOB).EQ.0) GO TO 739
            IF(IMARK.GT.0) GO TO 737
            IQK(LJOB)=0(LJOB)-IEK(LJOB)
            IMARK=1
            IHOLD=IQK(LJOB)
            GO TO 739
737 ICOMP=0(LJOB)-IEK(LJOB)
            IQK(LJOB)=MIN0(ICOMP,IHOLD)
            IHOLD=IQK(LJOB)
739 CONTINUE
          IBACK=0
          DO 741 I=1,M
            IF(MJOB(I).EQ.0) GO TO 741
            IF(IQK(I).GE.0) GO TO 741
            IF(IBACK.EQ.1) GO TO 741
            NNODE=NNODE+1

```



```

      IBACK=1
741 CONTINUE
      IF (IBACK.EQ.1) GO TO 1305
      IBACK=0
      IRANGE=M-1
      DO 749 I=1,IRANGE
      IF (MJOB(I).EQ.0) GO TO 749
      ILARGE=I+1
      DO 748 K=ILARGE,M
      IF (MJOB(K).EQ.0) GO TO 748
      IF (I.EQ.MCOL.AND.K.EQ.MROW) GO TO 743
      IF (NODE(K,I,KPRSSR)-100.NE.KPRSSR) GO TO 748
743 IF (IQK(I).GE.IQK(K)) GO TO 748
      IHOLD=0
      IFLAG=0
      JRANGE=K-1
      DO 747 J=1,JRANGE
      LASTJ=K-J
      IF (MJOB(LASTJ).EQ.0) GO TO 747
      IF (IFLAG.GT.0) GO TO 747
      IFLAG=1
      IHOLD=IEK(LASTJ)
747 CONTINUE
      IF (IEK(K)-IHOLD.LE.IQK(I)) GO TO 748
      IF (IBACK.EQ.1) GO TO 748
      IBACK=1
      NNODE=NNODE+1
748 CONTINUE
749 CONTINUE
      IF (IBACK.EQ.1) GO TO 1305
      IBACK=0
      IRANGE=M-1
      DO 759 I=1,IRANGE
      IF (IBACK.EQ.1) GO TO 759
      IF (MJOB(I).EQ.0) GO TO 759
      ILARGE=I+1
      DO 758 K=ILARGE,M
      IF (MJOB(K).EQ.0) GO TO 758
      IF (IQK(I).GE.IQK(K)) GO TO 758
      ISMALL=I-1
      DO 755 IR=1,ISMALL
      IF (IR.EQ.MROW.AND.K.EQ.MCOL) GO TO 753
      IF (NODE(IR,K,KPRSSR)-100.NE.KPRSSR) GO TO 755
753 IF (IEK(I).LE.D(IR)) GO TO 755
      IF (IBACK.EQ.1) GO TO 755
      IBACK=1
      NNODE=NNODE+1
755 CONTINUE
758 CONTINUE
759 CONTINUE
      IF (IBACK.EQ.1) GO TO 1305
      KLAST=0
      KSTART=M+1
      DO 763 K=1,M
      IF (KLAST.GT.0) GO TO 763
      KNEXT=KSTART-K
      IF (MJOB(KNEXT).EQ.0) GO TO 763
      KLAST=KNEXT
763 CONTINUE

```

```

      IBACK=0
      DO 769 I=1,M
      IF (IBACK.EQ.1) GO TO 769
      IF (MJOB(I).EQ.0) GO TO 769
      IF (I.EQ.KLAST) GO TO 769
      KFINAL=M+N+KPRSSR
      IF (I.EQ.MROW.AND.KFINAL.EQ.MCOL) GO TO 765
      IF (NODE(I,KFINAL,KPRSSR)-100.NE.KPRSSR) GO TO 769
765  IF (IEK(KLAST).LE.D(I)) GO TO 769
      IBACK=1
      NNODE=NNODE+1
769  CONTINUE
      IF (IBACK.EQ.1) GO TO 1305
      IBACK=0
      IRANGE=M-1
      DO 779 I=1,IRANGE
      IF (IBACK.EQ.1) GO TO 779
      IF (MJOB(I).EQ.0) GO TO 779
      ILARGE=I+1
      DO 778 K=ILARGE,M
      IF (IBACK.EQ.1) GO TO 778
      IF (MJOB(K).EQ.0) GO TO 778
      IF (I.EQ.MCOL.AND.K.EQ.MROW) GO TO 771
      IF (NODE(K,I,KPRSSR)-100.NE.KPRSSR) GO TO 778
771  IF (IQK(I).GE.IQK(K)) GO TO 778
      IF (IBACK.EQ.1) GO TO 778
      IF (K.NE.MCOL) GO TO 772
      IBK=CCOST(MROW,MCOL,KPRSSR)
      GO TO 777
772  IFROM=0
      DO 773 K1=1,CDIMEN
      IF (NODE(K1,K,NNEXT)-100.NE.KPRSSR) GO TO 773
      IFROM=K1
773  CONTINUE
      IF (IFROM.EQ.0) GO TO 774
      IBK=CCOST(IFROM,K,KPRSSR)
      GO TO 777
774  IBK=999
      DO 776 J=1,CDIMEN
      IF (CCOST(J,K,KPRSSR)-IBK.GT.0) GO TO 776
      IBK=CCOST(J,K,KPRSSR)
776  CONTINUE
777  IF (IBACK.EQ.1) GO TO 778
      IF (IBK+P(K,KPRSSR).LE.IQK(I)) GO TO 778
      IBACK=1
      NNODE=NNODE+1
778  CONTINUE
779  CONTINUE
      IF (IBACK.EQ.1) GO TO 1305
      IBACK=0
      IRANGE=M-1
      DO 789 I=1,IRANGE
      IF (IBACK.EQ.1) GO TO 789
      ILARGE=I+1
      DO 788 K=ILARGE,M
      IF (MJOB(K).EQ.0) GO TO 788
      IF (I.EQ.MCOL.AND.K.EQ.MROW) GO TO 781
      IF (NODE(K,I,KPRSSR)-100.NE.KPRSSR) GO TO 788
781  IF (IQK(I).NE.IQK(K)) GO TO 788

```

```

      IF(D(I).GE.D(K))GO TO 788
      IF(K.NE.MCOL)GO TO 787
      IBK=CCOST(MROW,MCOL,KPRSSR)
      GO TO 787
782  IFROM=0
      DO 783 K1=1,CDIMEN
      IF(NODE(K1,K,NNEXT)-100.NE.KPRSSR)GO TO 783
      IFROM=K1
783  CONTINUE
      IF(IFROM.EQ.0)GO TO 784
      IBK=CCOST(IFROM,K,KPRSSR)
      GO TO 787
784  IBK=999
      DO 786 J=1,CDIMEN
      IF(CCOST(J,K,KPRSSR)-IBK.GT.0)GO TO 786
      IBK=CCOST(J,K,KPRSSR)
786  CONTINUE
787  IF(IBK+P(K,KPRSSR)+IEK(I).LE.D(I))GO TO 788
      IF(IBACK.EQ.1)GO TO 788
      IBACK=1
      NNODE=NNODE+1
788  CONTINUE
789  CONTINUE
      IF(IBACK.EQ.1)GO TO 1305
      IBACK=0
      IRANGE=M-1
      DO 489 I=1,IRANGE
      IF(IBACK.EQ.1)GO TO 489
      IF(MJOB(I).EQ.0)GO TO 489
      ILARGE=I+1
      DO 488 K=ILARGE,M
      IF(MJOB(K).EQ.0)GO TO 488
      IF(I.EQ.MROW.AND.K.EQ.MCOL)GO TO 481
      IF(NODE(I,K,KPRSSR)-100.NE.KPRSSR)GO TO 488
481  IRHS=D(I)+CCOST(I,K,KPRSSR)+P(K,KPRSSR)
      IDBAR=MIN0(D(K),IRHS)
      IF(IDBAR.NE.IRHS)GO TO 488
      DO 487 J1=1,M
      IF(MJOB(J1).EQ.0)GO TO 487
      IF(IV.GT.0)GO TO 487
      J1N=J1+1
      IVN=0
      DO 483 J2=J1N,M
      IF(IVN.GT.0)GO TO 483
      IF(MJOB(J2).EQ.0)GO TO 483
      IVN=J2
483  CONTINUE
      IF(IVN.EQ.0)GO TO 487
      IF(D(J1).LE.IDBAR.AND.IDBAR.LT.D(IVN))GO TO 484
      GO TO 487
484  IV=J1
487  CONTINUE
      IF(IBACK.EQ.1)GO TO 488
      IF(IEK(IV).LE.D(I))GO TO 488
      IBACK=1
      NNODE=NNODE+1
488  CONTINUE
489  CONTINUE
      IF(IBACK.EQ.1)GO TO 1305

```

```

801 NNODE=NNODE+1
   NODE(MROW,MCOL,NNODE)=100+FINDC(MROW,MCOL)
   CALL UPDAT2(KPRSSR,CDIMEN,MROW,MCOL,M,N,CHOLD,NFLAG,
XICOUNT,NODE,NNODE)
905 IF(IPAIRS-ICOUNT)1005,1005,907
907 CALL UPDATK(NPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
XKPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)
   CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
   LBOUND(NNODE)=LBOUND(NNODE)+ISUM
998 IF(LBOUND(NNODE).GE.LCOST)GO TO 1305
   GO TO 301
1005 CALL UPDATK(NPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
XKPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)
   CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
   IPASS=0
   DO 1019 I=1,CDIMEN
   DO 1019 J=1,CDIMEN
   IF(IABS(KREDCD(I,J)).EQ.999)GO TO 1019
   LROW=I
   LCOL=J
   IF(IPASS.GT.0)GO TO 1016
   ISAVE=KREDCD(LROW,LCOL)
   KREDCD(LROW,LCOL)=999
   DO 1011 K=1,CDIMEN
   IF(IABS(KREDCD(K,LCOL)).EQ.999)GO TO 1011
   KROW=K
   LSAVE=KREDCD(K,LCOL)
   KREDCD(K,LCOL)=999
1011 CONTINUE
   NADD=0
   INUM=0
   ISTART=LROW+1
   DO 1015 IIND=ISTART,CDIMEN
   DO 1015 JIND=1,CDIMEN
   IF(IABS(KREDCD(IIND,JIND)).EQ.999)GO TO 1015
   INUM=INUM+1
1015 CONTINUE
   IF(INUM.GT.0)GO TO 1016
   KREDCD(KROW,LCOL)=LSAVE
   IPASS=IPASS+1
   GO TO 1019
1016 NODE(LROW,LCOL,NNODE)=100+FINDC(LROW,LCOL)
   NADD=NADD+1
   DO 1017 J1=1,CDIMEN
   KREDCD(LROW,J1)=999
1017 CONTINUE
   DO 1018 I1=1,CDIMEN
   KREDCD(I1,LCOL)=999
1018 CONTINUE
   IPASS=IPASS+1
1019 CONTINUE
   IF(NADD.EQ.2)GO TO 1059
   LBOUND(NNODE)=999
   GO TO 1305
1059 LBOUND(NNODE)=LBOUND(NNODE)+ISUM
   IF(IDUE.EQ.0)GO TO 1060
   IBEGIN=M+1
   IEND=M+N
940 DO 949 I=IBEGIN,IEND

```

```

      IJOB=I
      LTIME=0
      IPRSSR=IJOB-M
941  DO 943 J=1,CDIMEN
      IF(NODE(IJOB,J,NNODE).LE.0)GO TO 943
      JJOB=J
943  CONTINUE
      JFINAL=M+N+IPRSSR
      IF(JJOB.EQ.JFINAL)GO TO 949
      LTIME=LTIME+CCOST(IJOB,JJOB,IPRSSR)+P(JJOB,IPRSSR)
      IF(LTIME.GT.D(JJOB)) LBOUND(NNODE)=999
      IJOB=JJOB
      GO TO 941
949  CONTINUE
1060 WRITE(6,190)
      WRITE(6,1598)ITER
      IBEGIN=M+1
      IEND=M+N
      LTOTAL=0
      DO 1089 I=IBEGIN,IEND
      IJOB=I
      IPRSSR=IJOB-M
      WRITE(6,1080)IPRSSR,IJOB
1080 FORMAT(1H0,13X,'SCHEDULE',13,3X,'=',I4)
1085 DO 1086 J=1,CDIMEN
      IF(NODE(IJOB,J,NNODE).LE.0)GO TO 1086
      JJOB=J
1086 CONTINUE
      WRITE(6,1087)JJOB
1087 FORMAT(30X,I3)
      LTOTAL=LTOTAL+CCOST(IJOB,JJOB,IPRSSR)
      JFINAL=M+N+IPRSSR
      IF(JJOB.EQ.JFINAL)GO TO 1089
      IJOB=JJOB
      GO TO 1085
1089 CONTINUE
      WRITE(6,1189)LTOTAL
1189 FORMAT(1H0,13X,'TOTAL COST',4X,'=',I4)
      IF(LBOUND(NNODE).GT.LCOST)GO TO 1305
      LCOST=LBOUND(NNODE)
      WRITE(6,1191)LCOST
1191 FORMAT(1H0,13X,'LCOST',9X,'=',I4)
      IF(IOPT.EQ.0.AND.LCOST.LT.999)GO TO 9999
      DO 1205 I=1,CDIMEN
      DO 1205 J=1,CDIMEN
      NODE(I,J,1)=NODE(I,J,NNODE)
1205 CONTINUE
1305 NEXT=0
      NINDEX=NNODE-2
      DO 1307 K=1,NINDEX
      J=NNODE-K
      IF(LBOUND(J).GE.LCOST)GO TO 1307
      IF(NEXT.GT.0)GO TO 1307
      NEXT=J
1307 CONTINUE
      IF(NEXT.NE.0)GO TO 1505
      WRITE(6,1491)
1491 FORMAT(14X,'CURRENT SOLUTION IS OPTIMAL')
      GO TO 9999

```

```

1505 ISTART=NEXT+1
    DO 1507 I=ISTART,NNODE
        LBOUND(I)=0
1507 CONTINUE
    DO 1509 K=ISTART,NNODE
        DO 1509 I=1,CDIMEN
            DO 1509 J=1,CDIMEN
                NODE(I,J,K)=0
1509 CONTINUE
        NFLAG=0
        NNODE=NEXT
        GO TO 1513
1513 INDEX=N-1
    DO 1515 I=1,CDIMEN
        DO 1515 J=1,CDIMEN
            KREDCD(I,J)=CCOST(I,J,1)
            FINDC(I,J)=1
            DO 1515 K=1,INDEX
                NEXT=K+1
                IF(CCOST(I,J,NEXT)-KREDCD(I,J))1514,1514,1515
1514 KREDCD(I,J)=CCOST(I,J,NEXT)
                FINDC(I,J)=NEXT
1515 CONTINUE
            DO 1519 K=1,N
                DO 1519 I=1,CDIMEN
                    DO 1519 J=1,CDIMEN
                        CHOLD(I,J,K)=CCOST(I,J,K)
1519 CONTINUE
            LOOP=0
            DO 1521 I=1,CDIMEN
                DO 1521 J=1,CDIMEN
                    IF(NODE(I,J,NNODE).EQ.0)GO TO 1521
                    IF(NODE(I,J,NNODE).GT.0)NODE(I,J,NNODE)=0
                    LOOP=LOOP+1
1521 CONTINUE
            CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
            ICOUNT=0
            DO 1559 L=1,LOOP
                CALL ALTER(LOOK,NFLAG,NODE,NNODE,KREDCD,M,N,CHOLD,FINDC,THETA,
                    XMROW,MCOL,ICOUNT)
                IF(NODE(MROW,MCOL,NNODE))1529,1555,1555
1529 KPRSSR=FINDC(MROW,MCOL)
                CHOLD(MROW,MCOL,KPRSSR)=-999
                KREDCD(MROW,MCOL)=-999
                DO 1539 K=1,N
                    IF(IABS(KREDCD(MROW,MCOL)).LE.IABS(CHOLD(MROW,MCOL,K)))
                        XGO TO 1539
1534 KREDCD(MROW,MCOL)=CHOLD(MROW,MCOL,K)
                FINDC(MROW,MCOL)=K
1539 CONTINUE
                GO TO 1589
1555 KPRSSR=FINDC(MROW,MCOL)
                ICOUNT=ICOUNT+1
                NODE(MROW,MCOL,NNODE)=100+KPRSSR
                CALL UPDAT1(KPRSSR,CDIMEN,MROW,MCOL,N,CHOLD)
                CALL UPDAT2(KPRSSR,CDIMEN,MROW,MCOL,M,N,CHOLD,NFLAG,
                    XICOUNT,NODE,NNODE)
                CALL UPDATK(NPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
                    XKPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)

```

```

1589 CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
1559 CONTINUE
      ITER=ITER+1
1598 FORMAT(1H0,'ITERATION:',I3)
      GO TO 301
9999 STOP
      END
      SUBROUTINE LOAD(NTYPE,M,N,CCOST,KCOST,FINDC,P,D,CHOLD,IX,IS,IA,IB,
XIC,IO)
      INTEGER CDIMEN,CCOST,KCOST,FINDC,P,D,CHOLD
      CDIMEN=M+2*N
      DIMENSION CCOST(25,25,5),KCOST(25,25),
XFINDC(25,25),P(15,5),D(5),
XCHOLD(25,25,5)
      READ(5,29)((CCOST(I,J,K),J=1,CDIMEN),
*I=1,CDIMEN),K=1,N)
      READ(5,29)((P(I,J),I=1,M),J=1,N)
      READ(5,29)(D(I),I=1,M)
29  FORMAT( )
40  INDEX=N-1
      DO 50 I=1,CDIMEN
      DO 50 J=1,CDIMEN
      KCOST(I,J)=CCOST(I,J,1)
      FINDC(I,J)=1
      DO 50 K=1,INDEX
      NEXT=K+1
      IF(CCOST(I,J,NEXT)-KCOST(I,J)) 41,41,50
41  KCOST(I,J)=CCOST(I,J,NEXT)
      FINDC(I,J)=NEXT
50  CONTINUE
      DO 69 K=1,N
      DO 69 I=1,CDIMEN
      DO 69 J=1,CDIMEN
      CHOLD(I,J,K)=CCOST(I,J,K)
69  CONTINUE
99  RETURN
      END
      SUBROUTINE REDUCE(CHOLD,CDIMEN,N,IROWS,ICOLS,RMATRX,ISUM)
      INTEGER RMATRX,CHOLD,CDIMEN
      DIMENSION RMATRX(25,25),CHOLD(25,25,5)
      ISUM=0
      DO 50 I=1,IROWS
      MIN=999
      DO 29 J=1,ICOLS
      IF(ABS(RMATRX(I,J))-MIN) 21,29,29
21  MIN=RMATRX(I,J)
29  CONTINUE
      IF(MIN.EQ.0.OR.MIN.EQ.999) GO TO 50
      ISUM=ISUM+MIN
      DO 31 L=1,N
      DO 31 K=1,ICOLS
      IF(ABS(CHOLD(I,K,L))-999)30,31,31
30  CHOLD(I,K,L)=CHOLD(I,K,L)-MIN
      CHOLD(I,K,L)=MAX0(0,CHOLD(I,K,L))
31  CONTINUE
      DO 49 K=1,ICOLS
      IF(ABS(RMATRX(I,K))-999)37,49,49
37  RMATRX(I,K)=RMATRX(I,K)-MIN
49  CONTINUE

```

```

50 CONTINUE
   DO 80 J=1,ICOLS
     MIN=999
     DO 59 I=1,IROWS
       IF (IABS(RMATRX(I,J))-MIN) 51,59,59
51   MIN=RMATRX(I,J)
59 CONTINUE
     IF (MIN.EQ.0.OR.MIN.EQ.999) GO TO 80
     ISUM=ISUM+MIN
     DO 61 L=1,N
       DO 61 K=1,IROWS
         IF (IABS(CHOLD(K,J,L))-999) 60,61,61
60   CHOLD(K,J,L)=CHOLD(K,J,L)-MIN
       CHOLD(K,J,L)=MAX(0,CHOLD(K,J,L))
61 CONTINUE
     DO 79 L=1,IROWS
       IF (IABS(RMATRX(L,J))-999) 67,79,79
67   RMATRX(L,J)=RMATRX(L,J)-MIN
79 CONTINUE
80 CONTINUE
   RETURN
   END
   SUBROUTINE ALTER(LOOK,NFLAG,NODE,NNOUE,KREDCD,M,N,CHOLD,FINDC,
XTHETA,MROW,MCOL,ICOUNT)
   INTEGER THETA1,THETA2,THETA,CDIMEN,FINDC,CHOLD
   CDIMEN=M+2*N
   DIMENSION CHOLD(25,25,5),
XFINDC(25,25),KREDCD(25,25),
XNODE(25,25,50),IMAT(25,25,5),JMAT(25,25),
XJFIND(25,25),LOHOL(25,25)
   THETA=-1
   MROW=0
   MCOL=0
   DO 89 I=1,CDIMEN
     DO 89 J=1,CDIMEN
       IF (KREDCD(I,J).NE.0) GO TO 89
       KREDCD(I,J)=999
       MINROW=999
       DO 19 L=1,CDIMEN
         IF (IABS(KREDCD(I,L))-MINROW) 9,19,19
9     MINROW=KREDCD(I,L)
19 CONTINUE
       MINCOL=999
       DO 39 K=1,CDIMEN
         IF (IABS(KREDCD(K,J))-MINCOL) 29,39,39
29   MINCOL=KREDCD(K,J)
39 CONTINUE
       KREDCD(I,J)=0
       THETA1=MINROW+MINCOL
       IF (THETA1.GE.999) THETA1=999
       NEXT=999
       IPRSSR=FINDC(I,J)
       LSAVE=CHOLD(I,J,IPRSSR)
       CHOLD(I,J,IPRSSR)=999
       DO 59 KPRSSR=1,N
         IF (IABS(CHOLD(I,J,KPRSSR))-NEXT) 49,59,59
49   NEXT=CHOLD(I,J,KPRSSR)
59 CONTINUE

```



```

      CHOLD(I,J,IPRSSR)=LSAVE
      IF(999-NEXT) 61,61,69
61  THETA2=999
      GO TO 79
69  THETA2=NEXT-LSAVE
      IF(THETA2.LT.0)THETA2=0
79  ITEST=MIN0(THETA1,THETA2)
      IF(I.GT.M.AND.J.GT.M)ITEST=0
      IF(ITEST.LT.THETA) GO TO 89
      IF(LOOK.EQ.0)GO TO 199
      IF(ITEST.GT.THETA)GO TO 199
      IXB=MROW
      IXC=MCOL
      IVAR=0
      IF(IXB.EQ.0.OR.IXC.EQ.0)GO TO 199
107 DO 109 I1=1,CDIMEN
      DO 109 J1=1,CDIMEN
      JFIND(I1,J1)=FINDC(I1,J1)
109 CONTINUE
      DO 113 I1=1,CDIMEN
      DO 113 J1=1,CDIMEN
      DO 113 K=1,N
      IMAT(I1,J1,K)=CHOLD(I1,J1,K)
113 CONTINUE
      IABC=FINDC(MROW,MCOL)
      DO 119 I1=1,CDIMEN
      DO 119 J1=1,CDIMEN
      JMAT(I1,J1)=KREDCD(I1,J1)
119 CONTINUE
      NABD=NFLAG
      DO 123 I1=1,CDIMEN
      DO 123 J1=1,CDIMEN
      LOHOL(I1,J1)=0
123 CONTINUE
      MNOD=NNODE
      JCOT=ICOUNT
      CALL UPDATK(JFIND,IXB,IXC,IMAT,CDIMEN,M,N,IABC,JMAT,
      XNABD,NODE,MNOD,JCOT,LOHOL)
      CALL REDUCE(IMAT,CDIMEN,N,CDIMEN,CDIMEN,JMAT,ISUM)
      IVAR=IVAR+1
      IF(IVAR.EQ.2)GO TO 131
      IRED=ISUM
      IXB=I
      IXC=J
      GO TO 107
131 IF(IRED.LT.ISUM)GO TO 89
199 THETA=ITEST
      MROW=I
      MCOL=J
89 CONTINUE
      RETURN
      END
      SUBROUTINE UPDAT1(KPRSSR,CDIMEN,MROW,MCOL,N,CHOLD)
      INTEGER CHOLD,CDIMEN
      DIMENSION CHOLD(25,25,5)
      KSTOP=KPRSSR-1
      KSTART=KPRSSR+1
      IF(KSTOP-1)21,7,7
7 DO 19 K=1,KSTOP

```

```

      DO 9 J=1,CDIMEN
      CHOLD(MROW,J,K)=999
      CHOLD(MCOL,J,K)=999
9    CONTINUE
      DO 19 I=1,CDIMEN
      CHOLD(I,MCOL,K)=999
      CHOLD(I,MROW,K)=999
19   CONTINUE
21   IF(N-KSTART)41,23,23
23   DO 39 K=KSTART,N
      DO 29 J=1,CDIMEN
      CHOLD(MROW,J,K)=999
      CHOLD(MCOL,J,K)=999
29   CONTINUE
      DO 39 I=1,CDIMEN
      CHOLD(I,MCOL,K)=999
      CHOLD(I,MROW,K)=999
39   CONTINUE
41   RETURN
      END

      SUBROUTINE UPDAT2(KPRSSR,CDIMEN,MROW,MCOL,M,N,CHOLD,NFLAG,
XICOUNT,NODE,NNODE)
      INTEGER CDIMEN,CHOLD
      DIMENSION CHOLD(25,25,5),NODE(25,25,50)
      KINTL=M+KPRSSR
      KFINAL=M+N+KPRSSR
      NLEVEL=N-1
      MARK=0
      DO 3 I=1,CDIMEN
      DO 3 J=1,CDIMEN
      IP=NODE(I,J,NNODE)-100
      IF(IP.NE.KPRSSR)GO TO 3
      MARK=MARK+1
3    CONTINUE
      IJOB=KINTL
      NPATH=1
      DO 6 KMARK=1,MARK
      IF(NPATH.EQ.0)GO TO 6
      JJOB=0
      DO 4 J=1,CDIMEN
      IF(NODE(IJOB,J,NNODE).LE.0)GO TO 4
      JJOB=J
4    CONTINUE
      IF(JJOB.EQ.0)GO TO 5
      IJOB=JJOB
      GO TO 6
5    NPATH=0
6    CONTINUE
      IF(NPATH.EQ.1.AND.JJOB.EQ.KFINAL)GO TO 89
37   JSTART=M+N+1
      JSTOP=KFINAL-1
      LSTART=KFINAL+1
      LSTOP=M+2*N
      IF(JSTOP-JSTART)43,39,39
39   DO 41 J=JSTART,JSTOP
      CHOLD(MCOL,J,KPRSSR)=999
41   CONTINUE
43   IF(LSTOP-LSTART)49,45,45
45   DO 47 J=LSTART,LSTOP

```

```

      CHOLD(MCOL,J,KPRSSR)=999
47  CONTINUE
49  MSTART=M+1
      MSTOP=KINTL-1
      NSTART=KINTL+1
      NSTOP=M+N
      IF(MSTOP-MSTART)63,59,59
59  DO 61 I=MSTART,MSSTOP
      CHOLD(I,MROW,KPRSSR)=999
61  CONTINUE
63  IF(NSTOP-NSTART)69,65,65
65  DO 67 I=NSTART,NSTOP
      CHOLD(I,MROW,KPRSSR)=999
67  CONTINUE
69  CHOLD(KINTL,KFINAL,KPRSSR)=999
      CHOLD(MCOL,MROW,KPRSSR)=999
70  IF(NFLAG,LT,NLEVEL)GO TO 78
      DO 73 J=1,CDIMEN
      IF(CHOLD(MROW,J,KPRSSR).NE.999)GO TO 73
      CHOLD(MCOL,J,KPRSSR)=999
73  CONTINUE
      DO 77 I=1,CDIMEN
      IF(CHOLD(I,MCOL,KPRSSR).NE.999)GO TO 77
      CHOLD(I,MROW,KPRSSR)=999
77  CONTINUE
78  DO 79 J=1,CDIMEN
      CHOLD(MROW,J,KPRSSR)=999
79  CONTINUE
      DO 81 I=1,CDIMEN
      CHOLD(I,MCOL,KPRSSR)=999
81  CONTINUE
      GO TO 99
89  DO 91 I=1,CDIMEN
      DO 91 J=1,CDIMEN
      CHOLD(I,J,KPRSSR)=999
91  CONTINUE
99  RETURN
      END
      SUBROUTINE UPDATK(NPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
      XKPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)
      INTEGER CHOLD,CDIMEN,FINDC
      DIMENSION CHOLD(25,25,5),FINDC(25,25),
      XKREDCD(25,25),NODE(25,25,50),
      XKHOLD(25,25)
      KINTL=M+KPRSSR
      KFINAL=M+N+KPRSSR
      NLEVEL=N-1
      DO 2 I=1,CDIMEN
      DO 2 J=1,CDIMEN
      IF(NODE(I,J,NNODE).LE.0)GO TO 2
      K=NODE(I,J,NNODE)-100
      IF(K.EQ.KPRSSR)GO TO 2
      DO 1 LIND=1,N
      CHOLD(J,MROW,LIND)=999
1  CONTINUE
      KREDCD(J,MROW)=999
      FINDC(J,MROW)=999
2  CONTINUE
      MARK=0

```

```

DO 3 I=1,CDIMEN
DO 3 J=1,CDIMEN
IP=NODE(I,J,NNODE)-100
IF(IP.NE.KPRSSR)GO TO 3
MARK=MARK+1
3 CONTINUE
IJOB=KINTL
NPATH=1
DO 6 KMARK=1,MARK
IF(NPATH.EQ.0)GO TO 6
JJOB=0
DO 4 J=1,CDIMEN
IF(NODE(IJOB,J,NNODE).LE.0)GO TO 4
JJOB=J
4 CONTINUE
IF(JJOB.EQ.0)GO TO 5
IJOB=JJOB
GO TO 6
5 NPATH=0
6 CONTINUE
IF(NPATH.EQ.1.AND.JJOB.EQ.KFINAL)GO TO 79
DO 10 I=1,CDIMEN
DO 10 J=1,CDIMEN
KHOLD(I,J)=NODE(I,J,NNODE)
10 CONTINUE
ICHAIN=KINTL
DO 15 ILOOP=1,M
JJOB=0
DO 12 J=1,CDIMEN
IF(KHOLD(ICHAIN,J).LE.0) GO TO 12
K=KHOLD(ICHAIN,J)-100
IF(K.NE.KPRSSR)GO TO 12
JJOB=J
KHOLD(ICHAIN,J)=0
12 CONTINUE
IF(JJOB.EQ.0)GO TO 21
ICHAIN=JJOB
15 CONTINUE
21 JCHAIN=KFINAL
DO 25 JLOOP=1,M
IJOB=0
DO 23 I=1,CDIMEN
IF(KHOLD(I,JCHAIN).LE.0)GO TO 23
K=KHOLD(I,JCHAIN)-100
IF(K.NE.KPRSSR)GO TO 23
IJOB=I
KHOLD(I,JCHAIN)=0
23 CONTINUE
IF(IJOB.EQ.0)GO TO 27
JCHAIN=IJOB
25 CONTINUE
27 KCHECK=0
DO 33 IIND=1,CDIMEN
DO 33 JIND=1,CDIMEN
IF(KCHECK.GT.0)GO TO 35
IF(KHOLD(IIND,JIND).LE.0)GO TO 33
K=KHOLD(IIND,JIND)-100
IF(K.NE.KPRSSR)GO TO 33
KCHECK=1

```

```

33 CONTINUE
   IF(KCHECK.EQ.0.AND,NFLAG.LT,NLEVEL) GO TO 8
35 KREDCD(ICHAIN,JCHAIN)=999
   CHOLD(ICHAIN,JCHAIN,KPRSSR)=999
   FINDC(ICHAIN,JCHAIN)=999
8 DO 9 J=1,CDIMEN
   KREDCD(MROW,J)=999
   FINDC(MROW,J)=999
9 CONTINUE
   DO 19 I=1,CDIMEN
   KREDCD(I,MCOL)=999
   FINDC(I,MCOL)=999
19 CONTINUE
   DO 29 J=1,CDIMEN
   KREDCD(MCOL,J)=CHOLD(MCOL,J,KPRSSR)
   FINDC(MCOL,J)=KPRSSR
29 CONTINUE
   DO 31 I=1,CDIMEN
   KREDCD(I,MROW)=CHOLD(I,MROW,KPRSSR)
   FINDC(I,MROW)=KPRSSR
31 CONTINUE
   JSTART=M+N+1
   JSTOP=KFINAL-1
   LSTART=KFINAL+1
   LSTOP=M+2*N
   IF(JSTOP-JSTART)43,39,39
39 DO 41 J=JSTART,JSTOP
   KREDCD(MCOL,J)=999
   FINDC(MCOL,J)=999
41 CONTINUE
43 IF(LSTOP-LSTART)49,45,45
45 DO 47 J=LSTART,LSTOP
   KREDCD(MCOL,J)=999
   FINDC(MCOL,J)=999
47 CONTINUE
49 MSTART=M+1
   MSTOP=KINTL-1
   NSTART=KINTL+1
   NSTOP=M+N
   IF(MSTOP-MSTART)63,59,59
59 DO 61 I=MSTART,MSSTOP
   KREDCD(I,MROW)=999
   FINDC(I,MROW)=999
61 CONTINUE
63 IF(NSTOP-NSTART)69,65,65
65 DO 67 I=NSTART,NSTOP
   KREDCD(I,MROW)=999
   FINDC(I,MROW)=999
67 CONTINUE
69 KREDCD(KINTL,KFINAL)=999
   GO TO 199
79 DO 83 J=1,CDIMEN
   KREDCD(MROW,J)=999
   FINDC(MROW,J)=999
83 CONTINUE
   DO 84 I=1,CDIMEN
   KREDCD(I,MCOL)=999
   FINDC(I,MCOL)=999
84 CONTINUE

```

```

      DO 91 I=1,CDIMEN
      DO 91 J=1,CDIMEN
      IF(FINDC(I,J).NE.KPR5SR)GO TO 91
      KREDCD(I,J)=999
      DO 89 K=1,N
      IF(IABS(CHOLD(I,J,K))-KREDCD(I,J))81,81,89
81  KREDCD(I,J)=CHOLD(I,J,K)
      FINDC(I,J)=K
89  CONTINUE
91  CONTINUE
      NFLAG=NFLAG+1
      IF(NFLAG.LT.NLEVEL)GO TO 199
      DO 97 K=1,N
      LROW=M+K
      LCOL=M+N+K
      IF(FINDC(LROW,LCOL).EQ.999)GO TO 97
      IPR0=FINDC(LROW,LCOL)
      FINDC(LROW,LCOL)=999
      KREDCD(LROW,LCOL)=999
      CHOLD(LROW,LCOL,IPR0)=999
97  CONTINUE
139 DO 135 I=1,CDIMEN
      DO 135 J=1,CDIMEN
      KHOLD(I,J)=0
135 CONTINUE
      DO 137 K=1,M
      KHOLD(K,K)=999
137 CONTINUE
      KLEFT=0
      DO 141 I=1,CDIMEN
      DO 141 J=1,CDIMEN
      IF(IABS(KREDCD(I,J)).EQ.999)GO TO 141
      IF(KLEFT.GT.0)GO TO 141
      KLEFT=FINDC(I,J)
141 CONTINUE
      DO 149 I=1,CDIMEN
      DO 149 J=1,CDIMEN
      IF(NODE(I,J,NNODE).LE.0)GO TO 149
      K=NODE(I,J,NNODE)-100
      IF(K.NE.KLEFT)GO TO 149
      KHOLD(I,J)=999
      DO 143 J1=1,CDIMEN
      IF(KHOLD(I,J1).NE.999)GO TO 143
      KHOLD(J,J1)=999
143 CONTINUE
      DO 147 I1=1,CDIMEN
      IF(KHOLD(I1,J).NE.999)GO TO 147
      KHOLD(I1,I)=999
147 CONTINUE
149 CONTINUE
      DO 159 I=1,CDIMEN
      DO 159 J=1,CDIMEN
      IF(KHOLD(I,J).NE.999)GO TO 159
      KREDCD(I,J)=999
      CHOLD(I,J,KLEFT)=999
      FINDC(I,J)=999
159 CONTINUE
199 IF(NPRIME.EQ.0)GO TO 299
      NMACH=0

```

```

DO 207 K=1,N
KMACH=0
DO 203 I=1,CDIMEN
DO 203 J=1,CDIMEN
LCOLV=M+N+1
IF(I.GT.M.AND.J.GT.LCOLV)GO TO 203
IF(NODE(I,J,NNODE)-100,NE,K)GO TO 203
IF(KMACH.GT.0)GO TO 203
KMACH=1
203 CONTINUE
NMACH=NMACH+KMACH
207 CONTINUE
IF(NMACH.LT.NPRIME)GO TO 299
DO 269 K=1,N
KMACH=0
DO 217 I=1,CDIMEN
DO 217 J=1,CDIMEN
LCOLV=M+N+1
IF(I.GT.M.AND.J.GT.LCOLV)GO TO 217
IF(NODE(I,J,NNODE)-100,NE,K)GO TO 217
IF(KMACH.GT.0)GO TO 217
KMACH=1
217 CONTINUE
IF(KMACH.GT.0)GO TO 269
NFLAG=NFLAG+1
DO 219 I=1,CDIMEN
DO 219 J=1,CDIMEN
I1=M+K
J1=M+N+K
KREDCD(I1,I)=999
KREDCD(J,J1)=999
CHOLD(I,J,K)=999
IF(FINDC(I,J,NE,K)GO TO 219
KREDCD(I,J)=999
DO 215 K1=1,N
IF(CHOLD(I,J,K1)-KREDCD(I,J).GT.0)GO TO 215
KREDCD(I,J)=CHOLD(I,J,K1)
FINDC(I,J)=K1
215 CONTINUE
219 CONTINUE
IF(NFLAG.LT.NLEVEL)GO TO 269
DO 235 I=1,CDIMEN
DO 235 J=1,CDIMEN
KHOLD(I,J)=0
235 CONTINUE
DO 237 K1=1,M
KHOLD(K1,K1)=999
237 CONTINUE
KLEFT=0
DO 241 I=1,CDIMEN
DO 241 J=1,CDIMEN
IF(KREDCD(I,J).EQ.999)GO TO 241
IF(KLEFT.GT.0)GO TO 241
KLEFT=FINDC(I,J)
241 CONTINUE
DO 249 I=1,CDIMEN
DO 249 J=1,CDIMEN
IF(NODE(I,J,NNODE).LE.0)GO TO 249
K1=NODE(I,J,NNODE)-100

```

```
      IF(K1.NE,KLEFT)GO TO 249
      KHOLD(I,J)=999
      DO 243 J1=1,CDIMEN
      IF(KHOLD(I,J1).NE.999)GO TO 243
      KHOLD(J,J1)=999
243  CONTINUE
      DO 247 I1=1,CDIMEN
      IF(KHOLD(I1,J).NE.999)GO TO 247
      KHOLD(I1,I)=999
247  CONTINUE
249  CONTINUE
      DO 259 I=1,CDIMEN
      DO 259 J=1,CDIMEN
      IF(KHOLD(I,J).NE.999)GO TO 259
      KREDCD(I,J)=999
      CHOLD(I,J,KLEFT)=999
      FINDC(I,J)=999
259  CONTINUE
269  CONTINUE
299  RETURN
      END
```


APPENDIX B

COMPUTING TIMES FOR EXPERIMENTS WITH THE EXACT ALGORITHM

Table 8. Computing Times for Distinct Processor
Problems Where $N = 2$

M	Replication					Average Computing Time
	1	2	3	4	5	
5	.0042	.0018	.0019	.0053	.0034	.0033
6	.0031	.0139	.0189	.0182	.0066	.0121
7	.0441	.0248	.0067	.0582	.0131	.0294
8	.1000	.0030	.0387	.0392	.0647	.0491
9	.0377	.0269	.0393	.1367	.0097	.0501
10	.2074	.0867	.1560	.0591	.0376	.0734
11	.0217	.3264	.1148	.3245	.1232	.1821
12	.1170	.5634	.4029	.3318	.2868	.3404
13	.7224	.3651	.4004	.1942	.2626	.3889
14	1.0696	.9999	.8150	.2839	1.0326	.8402
15	1.3212	2.3999	1.8230	1.1580	2.3538	1.8112

Table 9. Computing Times for Distinct Processor
Problems Where $N = 3$

M	Replication					Average Computing Time
	1	2	3	4	5	
5	.0128	.0097	.0054	.0138	.0061	.0096
6	.0223	.0148	.0218	.0234	.0292	.0223
7	.0293	.0444	.0286	.0575	.0441	.0408
8	.0874	.0966	.1618	.0310	.2193	.1192
9	.2129	.0670	.2868	.0969	.1831	.1693
10	.1591	.4161	.7231	.2264	.1100	.2000
11	1.0397	.6136	.3208	.6107	.5470	.6264
12	.8000	1.2648	.2892	.8526	.8314	.8076
13	1.3289	.2875	1.4156	1.5990	1.3366	1.1935
14	1.7630	.8595	2.1749	1.5339	3.1996	1.9062
15	2.8717	1.2893	3.8757	2.3056	2.4618	2.5608

Table 10. Computing Times for Distinct Processor
Problems Where $N = 4$

M	Replication					Average Computing Time
	1	2	3	4	5	
5	.0157	.0421	.0267	.0367	.0202	.0283
6	.0805	.0549	.1417	.0171	.1044	.0797
7	.0880	.0132	.1143	.0945	.1059	.0832
8	.2271	.2184	.3935	.1879	.4526	.2959
9	.4505	.1438	.7109	.4505	.6536	.4819
10	.7211	.7261	.4662	.3944	1.2942	.7204
11	1.0881	.9028	.5455	1.2959	2.3075	1.2280
12	2.1433	1.8081	1.6916	1.0486	1.1711	1.5725
13	2.0304	2.1644	2.9316	1.9775	2.8996	2.4007
14	4.1284	4.6534				4.3909
15	8.2560	8.0182				8.1371

Table 11. Computing Times for Distinct Processor Problems Where $N = 2$ and Where Due Dates are Moderately Constraining

M	Replication					Average Computing Time
	1	2	3	4	5	
5	.0038	.0025	.0020	.0054	.0049	.0037
6	.0030	.0408	.0196	.0031	.0080	.0149
7	.0453	.0231	.0096	.0560	.0159	.0300
8	.1694	.0506	.0317	.1381	.0552	.0890
9	.0870	.0058	.1151	.1743	.2649	.1294
10	.1234	.0287	.3248	.0975	.0892	.1327
11	.1598	.2536	.2652	.1076	.2008	.1974
12	.2893	.5459	.3368	.1340	.4635	.3539
13	1.2105	.2761	.4065	.4336	.5786	.5811
14	1.0297	.8966	.9244	.6231	1.0215	.8991
15	2.1073	1.3271	2.0213	1.9721	2.2407	1.9337

Table 12. Computing Times for Distinct Processor Problems Where $N = 2$ and Where Due Dates are Highly Constraining

M	Replication					Average Computing Time
	1	2	3	4	5	
5	.0053	.0020	.0195	.0087	.0022	.0075
6	.0668	.2155	.0112	.0266	.0095	.0659
7	.6407	.0284	.0092	.0215	.0063	.1412
8	.1393	.0035	.2460	.5730	.0177	.1959
9	.0205	.1249	.5164	.3657	.0287	.2112
10	.0559	.5149	.0752	.4327	.2448	.2647
11	.4507	.5492	.4129	.2679	.7183	.4798
12	.7899	1.0608	.3109	1.0872	.3762	.7250
13	.8964	1.0239	.5673	.8219	.9611	.8541
14	.6469	1.9771	2.0611	1.3702	2.4062	1.6923
15	2.7193	3.0111	1.9556	1.3675	3.4550	2.5017

APPENDIX C

FORTRAN V CODE FOR RANDOM SCHEDULING

```

      INTEGER CCOST,CDIMEN
      DIMENSION CCOST(25,25,5),MJOB(15),NARRAY(15,5),MACH(5)
      READ(5,29)M,N,NTYPE,NPRIME
29  FORMAT( )
      CDIMEN=M+2*N
      READ(5,29)((CCOST(I,J,K),J=1,CDIMEN),
      *I=1,CDIMEN),K=1,N)
      JOBS=0
      DO 51 I=1,15
      MJOB(I)=0
51  CONTINUE
      DO 55 I=1,15
      DO 55 J=1,5
      NARRAY(I,J)=0
55  CONTINUE
      DO 57 I=1,5
      MACH(I)=0
57  CONTINUE
103 CALL MGEN(M,IS,MNEXT)
      IF(MJOB(MNEXT).EQ.1)GO TO 103
      MJOB(MNEXT)=1
      JOBS=JOBS+1
      CALL NGEN(M,N,NPRIME,JOBS,NTYPE,IU,MNEXT,MACH,NARRAY)
      IF(JOBS.LT.M)GO TO 103
      LTOTAL=0
      DO 131 I=1,N
      IJOB=M+I
      WRITE(6,108)I,IJOB
108  FORMAT(1H0,13X,'SCHEDULE',I3,3X,'=',I4)
111  IF(MACH(I).EQ.0)GO TO 127
      INDEX=MACH(I)
      DO 121 J=1,INDEX
115  DO 121 K=1,M
      IF(NARRAY(K,I).NE.J)GO TO 121
      LTOTAL=LTOTAL+CCOST(IJOB,K,I)
      WRITE(6,119)K
119  FORMAT(30X,I3)
117  IJOB=K
121  CONTINUE
127  JJOB=M+N+I
      WRITE(6,119)JJOB
129  LTOTAL=LTOTAL+CCOST(IJOB,JJOB,I)
131  CONTINUE
      WRITE(6,169)LTOTAL
169  FORMAT(1H0,13X,'TOTAL COST',4X,'=',I4)
      STOP
      END

```


APPENDIX D

FORTRAN V CODE FOR SHORTEST CHANGEOVER NEXT
OR MINIMUM TIME SUBSEQUENCE SCHEDULING

```

      INTEGER CCOST,CDIMEN
      READ (5,29)M,N,IX,IA,IB,NTYPE,NPRIME,LOOK
29  FORMAT( )
      DIMENSION CCOST (25,25,5)
      LTOTAL=0
      JOBS=0
      NTOTAL=0
      IPRSSR=0
      CDIMEN=M+2*N
      TIME=0.
      READ(5,29)((CCOST(I,J,K),J=1,CDIMEN),
        *I=1,CDIMEN),K=1,N)
91  IF(JOBS.GE.M)GO TO 195
      IF(NPRIME.EQ.0)GO TO 93
      IF(M-JOBS.LE.NPRIME-NTOTAL)GO TO 181
93  MINC=999
      IF(NTYPE.EQ.1)GO TO 94
      IPRSSR=IPRSSR+1
      DO 33 I=1,M
      DO 33 J=1,M
      IF(LOOK.EQ.0)GO TO 31
      MINROW=999
      DO 30 K=1,CDIMEN
      IF(K.EQ.I)GO TO 30
      IF(MINROW.LT.CCOST(J,K,IPRSSR))GO TO 30
      MINROW=CCOST(J,K,IPRSSR)
30  CONTINUE
      ITEST=M+IPRSSR
      LTEST=CCOST(I,J,IPRSSR)+CCOST(ITEST,I,IPRSSR)+MINROW
      IF(MINC.LE.LTEST)GO TO 33
      MINC=LTEST
      IJOB=I
      JJOB=J
      GO TO 33
31  IF(MINC.LT.CCOST(I,J,IPRSSR))GO TO 33
      MINC=CCOST(I,J,IPRSSR)
      IJOB=I
      JJOB=J
33  CONTINUE
      GO TO 97
94  DO 95 I=1,M
      DO 95 J=1,M
      DO 95 K=1,N
      IF(ILOOK.EQ.0)GO TO 41
      MINROW=999
      DO 40 KIND=1,CDIMEN
      IF(KIND.EQ.I)GO TO 40
      IF(MINROW.LT.CCOST(J,KIND,K))GO TO 40
      MINROW=CCOST(J,KIND,K)
40  CONTINUE
      ITEST=M+K
      LTEST=CCOST(I,J,K)+CCOST(ITEST,I,K)+MINROW
      IF(MINC.LE.LTEST)GO TO 95
      MINC=LTEST
      IJOB=I
      JJOB=J
      IPRSSR=K
      GO TO 95
41  IF(MINC.LT.CCOST(I,J,K))GO TO 95

```

```

      MINC=CCOST(I,J,K)
      IJOB=I
      JJOB=J
      IPRSSR=K
95  CONTINUE
97  INTL=M+IPRSSR
      IFINAL=M+N+IPRSSR
      WRITE(6,108)IPRSSR,INTL
108 FORMAT(1H0,13X,'SCHEDULE',I3,3X,'=',I4)
      WRITE(6,119)IJOB
      WRITE(6,119)JJOB
119 FORMAT(30X,I3)
      LTOTAL=LTOTAL+CCOST(INTL,IJOB,IPRSSR)+CCOST(IJOB,JJOB,IPRSSR)
      DO 141 J=1,CDIMEN
      DO 141 K=1,N
      CCOST(IJOB,J,K)=999
      CCOST(INTL,J,K)=999
      CCOST(J,IJOB,K)=999
      CCOST(J,JJOB,K)=999
141 CONTINUE
      JOBS=JOBS+2
      NTOTAL=NTOTAL+1
      IJOB=JJOB
161 IF(JOBS.GE.M)GO TO 191
      IF(NPRIME.EQ.0)GO TO 163
      IF(M-JOBS.GT.NPRIME-NTOTAL)GO TO 163
      WRITE(6,119)IFINAL
      LTOTAL=LTOTAL+CCOST(IJOB,IFINAL,IPRSSR)
      GO TO 181
163 KMIN=999
      IDIMEN=CDIMEN
      IF(NTOTAL.EQ.NPRIME.OR.NTOTAL.EQ.N)IDIMEN=M
      DO 165 K=1,IDIMEN
      IF(K.EQ.IJOB)GO TO 165
      IF(M-JOBS.EQ.1)GO TO 44
      IF(LOOK.EQ.0)GO TO 44
      IF(K.GT.M)GO TO 44
      MINROW=999
      DO 43 KIND=1,CDIMEN
      IF(KIND.EQ.IJOB)GO TO 43
      IF(MINROW.LT.CCOST(K,KIND,IPRSSR))GO TO 43
      MINROW=CCOST(K,KIND,IPRSSR)
43  CONTINUE
      LTEST=CCOST(IJOB,K,IPRSSR)+MINROW
      IF(KMIN.LE.LTEST)GO TO 165
      KMIN=LTEST
      JJOB=K
      GO TO 165
44  IF(CCOST(IJOB,K,IPRSSR).GE.KMIN)GO TO 165
      JJOB=K
      KMIN=CCOST(IJOB,K,IPRSSR)
165 CONTINUE
      WRITE(6,119)JJOB
      LTOTAL=LTOTAL+CCOST(IJOB,JJOB,IPRSSR)
      DO 171 J=1,CDIMEN
      DO 171 K=1,N
      CCOST(IJOB,J,K)=999
      CCOST(J,JJOB,K)=999
171 CONTINUE

```

```

      IJOB=JJOB
      IF(IJOB.NE.IFINAL)GO TO 179
      DO 175 I=1,CDIMEN
      DO 175 J=1,CDIMEN
      CCOST(I,J,IPRSSR)=999
175  CONTINUE
      IF(JOBS.LE.M-2)GO TO 91
      GO TO 181
179  JOBS=JOBS+1
      GO TO 161
181  IIND=M+1
      JIND=M+N
      MINC=999
      IF(NTYPE.EQ.1)GO TO 182
      IPRSSR=IPRSSR+1
      DO 186 I=IIND,JIND
      DO 186 J=1,M
      IF(LOOK.EQ.0)GO TO 59
      ITEST=M+N+IPRSSR
      LTEST=CCOST(I,J,IPRSSR)+CCOST(J,ITEST,IPRSSR)
      IF(LTEST.GT.MINC)GO TO 59
      MINC=LTEST
      IJOB=I
      JJOB=J
      GO TO 186
59  IF(MINC.LE.CCOST(I,J,IPRSSR))GO TO 186
      MINC=CCOST(I,J,IPRSSR)
      IJOB=I
      JJOB=J
186  CONTINUE
      GO TO 184
182  DO 183 I=IIND,JIND
      DO 183 J=1,M
      DO 183 K=1,N
      IF(LOOK.EQ.0)GO TO 69
      ITEST=M+N+K
      LTEST=CCOST(I,J,K)+CCOST(J,ITEST,K)
      IF(LTEST.GT.MINC)GO TO 183
      MINC=LTEST
      IJOB=I
      JJOB=J
      IPRSSR=K
      GO TO 183
69  IF(MINC.LE.CCOST(I,J,K))GO TO 183
      IJOB=I
      JJOB=J
      IPRSSR=K
      MINC=CCOST(I,J,K)
183  CONTINUE
184  WRITE(6,108)IPRSSR,IJOB
      WRITE(6,119)JJOB
      IFINAL=M+N+IPRSSR
      WRITE(6,119)IFINAL
      LTOTAL=LTOTAL+CCOST(IJOB,JJOB,IPRSSR)+CCOST(JJOB,IFINAL,IPRSSR)
      DO 185 J=1,CDIMEN
      DO 185 K=1,N
      CCOST(IJOB,J,K)=999
      CCOST(J,JJOB,K)=999
      CCOST(JJOB,J,K)=999

```

```
185 CONTINUE
    JOBS=JOBS+1
    IF (JOBS.LT.M) GO TO 181
    GO TO 195
191 WRITE(6,119) IFINAL
    LTOTAL=LTOTAL+CCOST(IJOB,IFINAL,IPRSSR)
195 WRITE(6,201) LTOTAL
    STOP
```

BIBLIOGRAPHY

1. Conway, R. W., Maxwell, W. L., and Miller, L. W., *Theory of Scheduling*, Addison-Wesley, Reading, Massachusetts, (1967).
2. Panwalkar, S. S., Dudek, R. A., and Smith, M. L., "Sequencing Research and the Industrial Scheduling Problem," presented at the Symposium on Scheduling, North Carolina State University, Raleigh, May 15-17, 1972.
3. Woolsey, R. E. D., "A Survey of Quick-and-Dirty Methods for Production Scheduling," *Production and Inventory Management*, 12(1971), 1(February), 60-68.
4. Smith, M. L., "A Critical Analysis of Flow-Shop Sequencing," Doctoral Dissertation, Texas Tech University, (1968).
5. Prunzel, J. F., "Setup Times and Setup Costs in Sequencing Problems," Master's Thesis, Texas Tech University, (1972).
6. Pierce, J. F. and Hatfield, D. J., "Production Sequencing by Combinatorial Programming," Chapter 17 of J. F. Pierce, *Operations Research and the Design of Management Information Systems*, Technical Association of the Pulp and Paper Industry, New York, (1967).
7. Geoffrion, A. M. and Marsten, R. E., "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," Chapter 5 of A. M. Geoffrion, *Perspectives in Optimization*, Addison-Wesley, Reading, Massachusetts, (1972).
8. Garfinkel, R. S. and Nemhauser, G. L., "The Set-Partitioning Problem: Set Covering with Equality Constraints," *Operations Research*, 17(1969), 5(September-October), 848-856.
9. Lemke, C. E., Salkin, H. M., and Spielberg, K., "Set Covering by Single-Branch Enumeration with Linear-Programming Subproblems," *Operations Research*, 19(1971), 4(July-August), 998-1025.
10. Jensen, P. A., "Optimum Network Partitioning," *Operations Research*, 19(1971), 4(July-August), 916-932.

11. Pierce, J. F., "Application of Combinatorial Programming to a Class of All Zero-One Integer Programming Problems," *Management Science*, 15(1968), 3(November), 191-209.
12. Little, J. D. C., Murty, K. G., Sweeny, D. W., and Karel, C., "An Algorithm for the Traveling Salesman Problem," *Operations Research*, 11(1963), 6(November-December), 972-989.
13. Hu, T. C. "Parallel Sequencing and Assembly Line Operations," *Operations Research*, 9(1961), 6(November), 841-848.
14. McNaughton, R., "Scheduling with Deadlines and Loss Functions," *Management Science*, 6(1959), 1(October), 1-12.
15. Eastman, W. L., Even, S., and Isaacs, I. M., "Bounds for the Optimal Scheduling of n Jobs on m Processors," *Management Science*, 11(1964), 2(November), 268-279.
16. Lawler, E. L., "On Scheduling Problems with Deferral Costs," *Management Science*, 11(1964), 2(November), 280-288.
17. Root, J. G., "Scheduling with Deadlines and Loss Functions on K Parallel Machines," *Management Science*, 11(1965), 3(January), 460-475.
18. Rothkopf, Michael, "Scheduling Independent Tasks on One or More Processors," Doctoral Dissertation, Massachusetts Institute of Technology, (1964).
19. Rothkopf, Michael, "Scheduling Independent Tasks on Parallel Processors," *Management Science*, 12(1966), 5(January), 437-447.
20. Cox, D. R. and Jessop, W. N., "The Theory of a Method of Production When There Are Many Products," *Operational Research Quarterly*, 13(1962), 4(December), 309-328.
21. Elmaghraby, S. E., "The Sequencing of N Jobs of M Parallel Processors," Research Memorandum, North Carolina State University, January, 1968.
22. Elmaghraby, S. E., "The Machine Sequencing Problem-- Review and Extensions," *Naval Research Logistics Quarterly*, 15(1968).

23. Braun, W., "A Computerized Simulation Approach to the Solution of the Carrier Dispatching Problem," Master's Thesis, Kansas State University.
24. Hayes, R. L., "The Delivery Problem," Doctoral Dissertation, Carnegie Institute of Technology, (1967).
25. Newton, R. M. and Thomas, W. H., "Design of School Bus Routes by Digital Computer," presented at the Thirtieth National Meeting of ORSA, October, 1966.
26. Quon, J., Charnes, A., and Werson, S., "Simulation and Analysis of a Refuse Collection System," *Journal of the Sanitary Engineering Division, ASCE*, 91(1965), SA5(October), 17-36.
27. Balinski, M. L. and Quandt, R. E., "On an Integer Program for a Delivery Problem," *Operations Research*, 12(1964), 2(March-April), 300-304.
28. Held, M. and Karp, R. M., "A Dynamic Programming Approach to Sequencing Problems," *Journal of SIAM*, 10(1962), 1(March), 196-210.
29. Clark, G. and Wright, J. W., "Scheduling Vehicles from a Central Depot to a Number of Delivery Points," *Operations Research*, 12(1964), 4(July-August), 568-571.
30. Tillman, F. A. and Cochran, H., "A Heuristic Approach for Solving the Delivery Problem," *Journal of IE*, 19(1968), 7(July), 354-358.
31. Hering, R. W., "Evaluation of Some Heuristic Look-Ahead Rules for Multiple-Terminal Delivery Problems," Master's Thesis, Kansas State University, (1970).
32. Bellmore, M. and Nemhauser, G. L., "The Traveling Salesman Problem: A Survey," *Operations Research*, 16(1968), 3(May-June), 538-558.
33. Hong, S., "A Linear Programming Approach for the Traveling Salesman Problem," Doctoral Dissertation, The John Hopkins University, (1972).
34. Lockett, A. G. and Muhlemann, A. P., "A Scheduling Problem Involving Sequence-Dependent Changeover Times," *Operations Research*, 20(1972), 4(July-August), 895-902.

35. Ashour, S., Vega, J. F., and Parker, R. G., "A Heuristic Algorithm for Traveling Salesman Problems," *Transportation Research*, 6(1972), 187-195.

VITA

Born April 23, 1944, in Leesburg, Florida, Joseph D. Marsh attended public schools there and entered the University of Florida in September, 1962. He completed the liberal arts program of the University College and was named Vice President and Director of True Temp, Inc., a Leesburg, Florida mechanical contractor in September, 1964.

He entered the College of Engineering at the University of Florida in September, 1965, and received the Bachelor of Industrial Engineering degree with honors in December, 1967. During his undergraduate program, he was charter president of the University of Florida Chapter of Alpha Pi Mu, national industrial engineering honorary. He was elected to membership in Sigma Tau and Tau Beta Pi, both national engineering honoraries, and national scholastic honorary, Phi Kappa Phi. He was tapped as a charter member of the University's chapter of leadership honorary, Omicron Delta Kappa.

He entered graduate school at the University of Florida in January, 1968, working as a graduate teaching assistant and as Project Engineer for the University's J. Hillis Miller Health Systems Research Division. He received the Master of Science in Engineering degree in June, 1969, and was elected to associate membership in the

scientific Society of the Sigma Xi.

Joseph D. Marsh began doctoral study in industrial engineering at the Georgia Institute of Technology in June, 1969. He served as both Graduate Teaching Assistant and Instructor in the School of Industrial and Systems Engineering. He is now Assistant Professor in the Department of Operations Research at the George Washington University.