

OPTIMAL RULES AND ALGORITHMS FOR
SOME PARALLEL PROCESSOR SCHEDULING PROBLEMS

By

LOUIS ANTHONY MARTÍN-VEGA

A DISSERTATION PRESENTED TO THE GRADUATE COUNCIL OF
THE UNIVERSITY OF FLORIDA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1975

*Para mi querida Magali y
nuestras dos bendiciones
Louisito y Mónica*

*Preguntó el joven al sabio: "¿De las virtudes de la vida,
será inteligencia la que domina?"*

*Respondió el sabio al joven: "No te engañes hijo mío,
que en la persistencia y confianza en Dios está
el verdadero poderio."*

ACKNOWLEDGEMENTS

I wish to acknowledge the numerous and significant contributions to this study by Dr. H. Donald Ratliff, who served as Chairman of my Doctoral Committee and provided the original motivation for this research. His guidance and insight have contributed considerably to both the form and contents of this dissertation and his dedication to the development of this work is sincerely appreciated.

I thank the other members of my committee, Dr. Richard L. Francis, Dr. Thom J. Hodgson and Dr. Ira Horowitz for their constructive comments and suggestions leading to the completion of this work. I am also grateful to Dr. M.E. Thomas and Dr. Eginhard Muth for their assistance and support throughout my stay at the University of Florida.

I also acknowledge the financial assistance received from the Economic Development Administration of Puerto Rico throughout the course of my graduate studies and the support of Messrs. Jorge Vega, Joaquin Flores and Luis V. Martín in the acquisition of this aid.

Thanks are also extended to Mrs. Edna Larrick, Miss Mary Van Meer and Mrs. Pat Whitehurst for the typing of the manuscript and to Mr. Jeff Whitehurst, for his artful preparation of all figures and illustrations.

Finally, I bow my head in loving respect to my dearest wife Maggi for her patience, understanding and ever-present gentle encouragement. To my fine son, Louisito, I extend my token of pride and

appreciation which is sure to grow with his years. And to my pequeña Mónica goes my delight at her unique insistence in sharing in the final stages of this work. To this my family, as well as to my parents Louis and Eva Martín, I extend my gratefulness for a moment which is as much theirs as it is mine.

This dissertation was supported in part under grants from the Office of Naval Research and the Army Research Office, Durham, N.C.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	viii
ABSTRACT	ix
 CHAPTER	
1 INTRODUCTION AND OVERVIEW.....	1
1.1 In the Beginning	1
1.2 The Parallel Processor Scheduling Problem	2
1.3 Representation of a Schedule	4
1.4 Scope of the Research	5
1.5 Overview	5
2 LITERATURE REVIEW	8
2.1 Classification Scheme	10
2.2 Optimality Criteria	10
2.3 Organization of the Review	13
2.4 Path 9	13
2.5 Path 8	14
2.5.1 Min-Sum Criteria	15
2.5.2 Min-Max Criteria	17
2.6 Path 7	18
2.7 Path 6	21
2.7.1 Min-Sum Criteria	21
2.7.2 Min-Max Criteria	23
2.8 Path 5	26
2.9 Paths 4 and 3	27
2.9.1 Path 4	27
2.9.2 Path 3	28
2.10 Paths 2 and 1	32
2.10.1 Min-Sum Criteria	33
2.10.2 Min-Max Criteria	39
3 SCHEDULING RULES FOR SOME PARALLEL PROCESSOR PROBLEMS. 41	41
3.1 Introduction	41
3.2 Scheduling with Job Splitting and No Restrictions on Job Availabilities	43
3.2.1 General Formulation	43
3.2.2 Minimization of Maximum Lateness	49
3.2.3 The "Due Date" Rule as a Special Case of the "Greatest Potential Lateness" Rule. 53	53

	<u>Page</u>
3.3 Nonsimultaneous Job Arrivals	54
3.3.1 Minimization of Maximum Flow Time	54
3.3.2 Minimization of Maximum Lateness	59
3.4 Simultaneous Processing by More than One Machine..	62
3.5 Two Min-Sum Problems	64
3.5.1 Limited Machine Availabilities	64
3.5.2 Jobs with Common Due Dates	65
3.6 Minimization of Maximum Deferral Cost	66
3.7 Conclusions	67
4 FIXED ROUTE PARALLEL PROCESSOR SCHEDULING PROBLEMS	69
4.1 Introduction	69
4.2 The One Train-One Track Problem	72
4.3 The M Train-One Track Problem	76
4.3.1 Problem Formulation	77
4.3.2 Maximization of the Number of Jobs Taken to their Destinations by the First K Trains	79
4.3.3 Minimization of the Sum of Completion Times	83
4.4 Restrictions on Job Availabilities	86
4.4.1 Jobs Not Simultaneously Available	86
4.4.2 Jobs Restricted by Due Dates	92
4.4.3 Jobs Not Simultaneously Available and Sub- ject to Due Dates	97
4.4.4 Synopsis	98
4.5 Minimization of Maximum Lateness	98
4.5.1 Minimization of L_{\max} with Job Splitting ...	98
4.5.2 Feasible Schedules Without Job Splitting...	103
4.6 M Trains on One Track with Switchyards	107
4.6.1 Train Schedules Based on Demand	108
4.6.2 Fixed Train Schedules	109
4.7 Routes Consisting of a Directed Network	117
4.8 Conclusion	120
5 MIN-MAX PARALLEL PROCESSOR ALLOCATION PROBLEMS	122
5.1 Introduction	122
5.2 Formulation	125
5.3 Case 1: Minimization of Maximum Loss Probability.	126
5.4 Case 2: Minimization of Maximum Penalty	132
5.5 Conclusions	135
6 SUMMARY AND SUGGESTIONS FOR FUTURE RESEARCH	137

	<u>Page</u>
APPENDICES	
1 COMPUTATIONAL RESULTS	143
2 REDISCOVERING THE SQUARE WHEEL	154
3 THE OPERATION OF THE SEABOARD COAST LINE RAIL- ROAD (SCL)	166
BIBLIOGRAPHY	172
BIOGRAPHICAL SKETCH	178

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
1	Classification of Deterministic Scheduling Problems	9
2	A Three Job, Two Machine Problem	16
3	Example of a Two Job, Three Period Problem with M Identical Processors	124
4	Network Representation of a Two Port Three Period Min- Max Loss Probability Allocation Problem	128
5	An Example of a Min-Max Allocation Problem on a Directed Network	130
6	Network Representation of a Two Port, Three Period Min-Max Penalty Allocation Problem	134
7	Plot of CPU Time Versus Problem Size	152
8	Freight Railway Network	167

ABSTRACT OF DISSERTATION PRESENTED TO THE GRADUATE COUNCIL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

OPTIMAL RULES AND ALGORITHMS FOR
SOME PARALLEL PROCESSOR SCHEDULING PROBLEMS

By

Louis Anthony Martín-Vega

March, 1975

Chairman: H. Donald Ratliff
Major Department: Industrial and Systems Engineering

A study of some deterministic parallel processor scheduling problems is presented. Optimal rules and algorithms (i.e., constructional methods which lead directly to an optimal solution) are developed for problems motivated by the theory of parallel processor scheduling itself as well as for problems motivated by their potential application in transportation systems. This is in contrast to enumeration type approaches (e.g., dynamic programming, branch and bound) which have been utilized for problems of this nature.

With respect to the classical parallel processor environment, a basic problem of scheduling identical processors without restrictions on job or machine availabilities is first considered. A priority based rule is shown to minimize maximum lateness when a limited form of job splitting is allowed. Optimality of the rule is shown to extend for minimizing maximum flow time when jobs are not simultaneously available. Algorithms and properties are presented for special cases of more general problems and other optimality criteria. These

developments are also used to tie in previously unrelated results in the area.

Optimal rules are then presented for various criteria involving the scheduling of freight on a class of fixed route railway systems. Train movement is modeled as a series of parallel processors whose availability is subject to a predetermined train schedule. Situations ranging from maximization of the number of jobs taken to their destinations on one train and multiple train systems to the scheduling of jobs on systems consisting of directed networks are considered. Besides presenting a new scenario for scheduling research, these results represent one of the first developments of nonsimulation type models for scheduling problems in the railway industry.

An extensive review of the literature is also included which serves as a unified framework for future work in the area of parallel processor scheduling.

CHAPTER 1

INTRODUCTION AND OVERVIEW

1.1 In the Beginning

The origin of scheduling problems can be traced to the annals of primitive man. Surely his initial observation of the difficulties involved in trying to accomplish more than one task simultaneously presented him with the "first" scheduling problem. While our unsung "first scheduler" must by necessity remain anonymous, what is clear is that attempts to quantify the scheduling decision process began around the turn of the twentieth century. Such ventures, being mainly confined to the manufacturing shop environment, led to the phrase "machine scheduling problem" as a descriptor of such processes. These attempts and their documentation served to create what is known as the "theory of scheduling," a field of study whose present applicability transcends its original environment. While scheduling problems continue to attract increasing numbers of researchers, their complexity is such that even today, existing results consist mainly of a great number of individual contributions lacking, for the most part, any unifying theory.

Interest in scheduling problems has grown appreciably within the last two decades. Certainly a great deal of this interest is spurred by the countless applications that have arisen for problems initially relegated to a "machine shop" environment as well as the theoretical

challenge that this apparently well-structured and yet difficult to solve problem presents. The development of numerous formulations and techniques for dealing with scheduling problems has expanded knowledge in this once highly specialized area to the point where the "theory of scheduling" is now a general term and notions such as "job shop" and "flow shop" problems constitute specializations of the general theory. This dissertation will concentrate on a particular class of deterministic scheduling problems, namely, the problem of scheduling jobs on parallel processors.

1.2 The Parallel Processor Scheduling Problem

Informally, the deterministic scheduling problem can be described as follows: (Rinnooy-Kan [61])

Given a set of n jobs (tasks, events, products, . . .) that have to pass through m machines (processors) in a given prescribed order (which may be no order) under certain restrictive assumptions (which may be no assumptions), what is according to some criterion, the optimal schedule for handling the jobs.

The parallel processor problem is a subset of deterministic scheduling specifically characterized by the assumption that for any job j , $j = 1, \dots, n$ and machine l , $l = 1, \dots, m$, either machine l is capable of completely processing job j or machine l does not have the capability of processing job j at all. This implies that the only ordering restrictions which exist are those that may be specified among the jobs themselves. This is in contrast to other deterministic problems where some of the jobs may require processing on more than one

machine before they are completed in which case machine ordering restrictions may also exist.

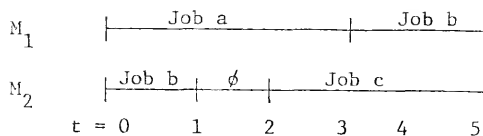
Interest in parallel processor systems has blossomed with the advent of multi-processor computer systems. As Muntz [56, p. 1] notes, "these systems have been shown to have several advantages over more conventional single processor organizations. The one of most importance to a large class of users is the potential increase in computing speed achievable by parallel programming. This is particularly of interest for real time applications and also for lengthy computations, such as weather forecasting, when the results are needed more quickly than they can be provided by a single processor." Other areas of applicability that appear in the literature include manufacturing systems [63, 77] and problems concerning project planning under scarce resources [26].

The impact of a scheduling decision varies from one situation to the next. In many cases it is impossible to isolate the scheduling process. As Rinnooy-Kan [61, p.4] points out, the scheduling decision is generally preceded by planning activity and followed by control activity, both of them involving economic and technological judgments that strongly influence the scheduling decision itself. Pounds [59] reports that management is often not even aware that a scheduling problem exists; there are so many decisions to be made that the order that jobs are processed is not perceived as an influenceable and relevant variable. However, as Elmaghraby [26] notes, scheduling decisions are likely to get more and more important as the computer takes over many routine decisions and as improved operations research techniques perfect other ones. And

Mellor [53] quotes a list of no less than 27 goals that can be attained by good scheduling, including items as diverse as day-to-day stability of work force and anticipation of price changes. Moreover, there are situations in which the scheduling decision becomes the principal factor affecting daily operation as is the case in crew scheduling on airlines. While the argument over the impact of scheduling decisions continues, there appears to be general agreement in that oftentimes significant differences exist in the use of one schedule over another. It is this factor that constitutes the basic "raison d'etre" for work in this area.

1.3 Representation of a Schedule

The objective of a parallel processor problem centers around determining an optimal schedule. Informally, a schedule is simply a description of the work to be done by each machine at each moment in time. The simplest way of specifying the schedule is through the use of a Gantt Chart. The chart consists of a time axis for each processor with intervals marked off and labeled with the name of the job being processed. A symbol of \emptyset can be used to represent an idle period [56]. For example, the following schedule for two jobs shows that Job a is processed from $t = 0$ to $t = 3$ on machine one, Job b from $t = 0$ to $t = 1$ on machine two and $t = 3$ to 5 on machine one, and Job c from $t = 2$ to $t = 5$ on machine two. The period $t = 1$ to $t = 2$ is an idle period on machine two.



1.4 Scope of the Research

This dissertation presents a study of deterministic parallel processor scheduling problems. The primary contribution of the study is the development of optimal rules and algorithms (i.e., constructive methods which lead directly to an optimal solution) for some parallel processor problems. This is in contrast to enumeration type approaches (e.g., dynamic programming, branch and bound) which have also been utilized for problems of this nature. The research effort is divided between problems motivated essentially by the theory of parallel processor scheduling itself, as well as problems motivated by their potential application in transportation and military systems. The effort is brought together by the fact that the insight developed in the treatment of the former provides the foundation for the solution of the latter.

A secondary contribution is the interlacing of a number of heretofore unrelated results which appear in the literature on single and parallel processor problems and the establishment of a common, unified framework for future work in the area.

1.5 Overview

In Chapter 2, a classification scheme for parallel processor problems is developed. The scheme serves as the basic outline for an extensive review of the state of the art of parallel processor scheduling and provides a framework for identification of areas for future development. The review emphasizes and provides details for those problems where scheduling rules and network formulations suffice as solution

procedures although enumerative approaches are mentioned when applicable.

In Chapter 3 problems which arise from the theory of parallel processor scheduling are studied. The basic problem of scheduling identical processors without restrictions on job or machine availabilities is considered. An optimal rule is developed for minimizing maximum lateness when jobs are assumed simultaneously available and a limited type of job splitting is allowed. Optimality of the rule is shown to extend for the problem of minimizing maximum flow time when jobs are not simultaneously available. Development of rules for more general problems is hindered by the lack of a dominant priority selection criterion. Algorithms and properties are also presented for special cases of more general problems and other optimality criteria. The developments of this chapter are also used to tie in previously unrelated results in the area of parallel processor scheduling.

Chapter 4 provides the most important contribution of this dissertation. Optimal rules are developed for a multiplicity of criteria in problems involving the scheduling of freight on a class of fixed route railway systems. In these problems the trains are modeled as parallel processors whose availability is subject to a predetermined train schedule. Situations ranging from maximization of the number of jobs taken to their destinations on one train and multiple train systems to the scheduling of jobs on systems consisting of directed networks are considered. Besides presenting a new scenario for scheduling research, this chapter represents one of the first developments of non-simulation type models for scheduling problems in the railway industry.

In Chapter 5, min-max parallel processor problems of the type generally found in production scheduling contexts are considered. These problems have the particular characteristic that penalties are assessed as a function of how much of a job has been completed by a certain period rather than as a function of a job's completion time as is the case in previous chapters. Optimal algorithms are presented for two cases. The problems considered in this chapter, while not of the same spirit as those in the previous chapters, chronologically were the first approached in this study. The notions utilized in their solution opened the door to many of the developments of Chapters 3 and 4.

Finally, Chapter 6 presents a summary and discussion of areas for future research.

CHAPTER 2

LITERATURE REVIEW

2.1 Classification Scheme

The accompanying diagram (Figure 1) presents a schematic classification of deterministic scheduling problems upon which the organization of this review is based. The scheme is based on identifying the restrictive assumptions currently imposed on deterministic scheduling problems and the level of difficulty they create in the solution of the same. Each path in the drawing represents one particular scheduling problem (e.g., Path 6 denotes the m identical parallel processor problem with sequence independent setup times and precedence constraints among the jobs). In general, the higher up in the tree an assumption occurs, the more critical it is to the computational efficiency of currently available algorithms for solving problems under the assumption. Underlying the classification scheme are the optimality criteria which in this review will be partitioned into two major divisions: min-max and min-sum criteria.

2.2 Optimality Criteria [16, 61]

We first establish the following definitions:

Let

r_j = the release time of job j (the earliest time that processing could start on job j where $r_j = 0$ for all j)

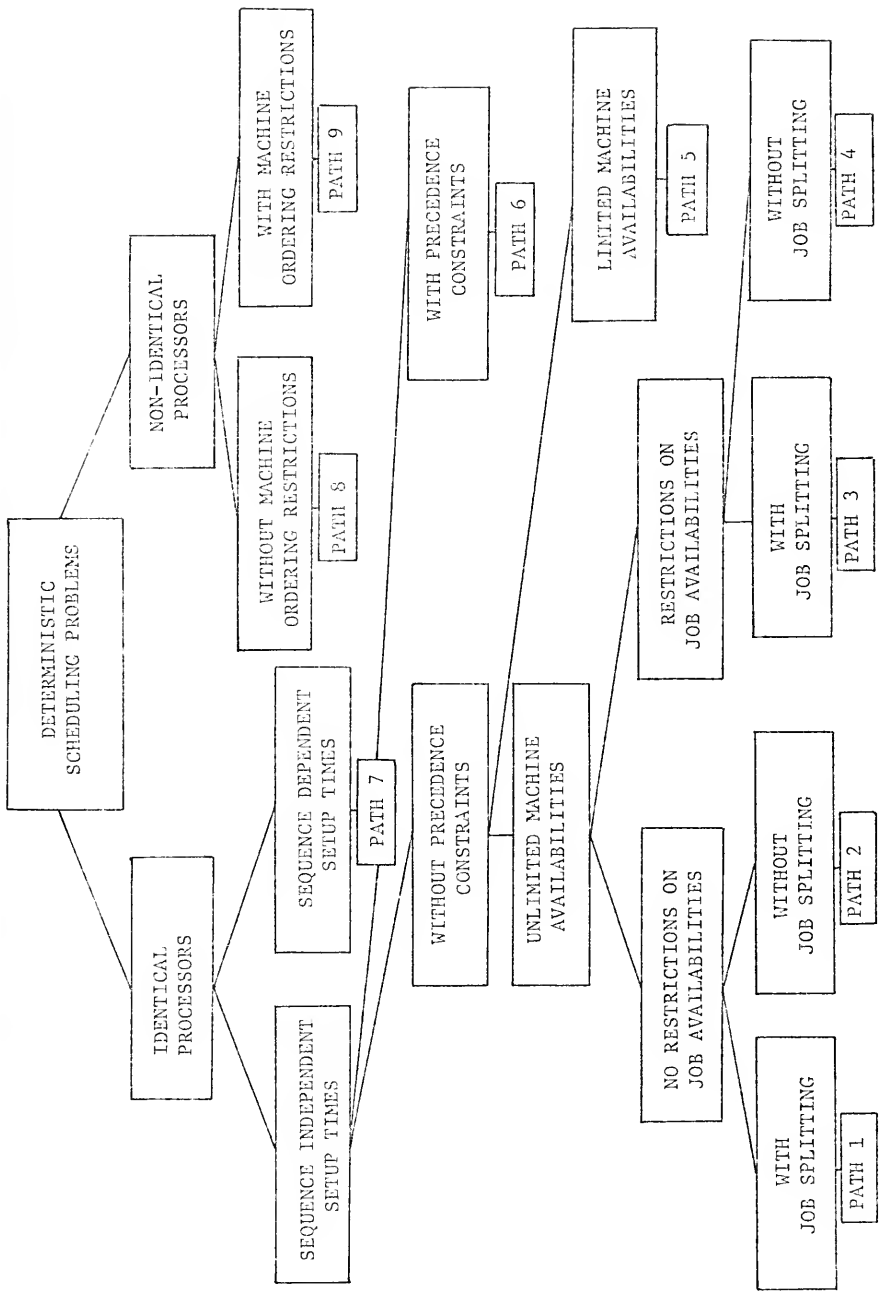


Figure 1: Classification of Deterministic Scheduling Problems

implies that all jobs are simultaneously available
for processing)

p_j = the total processing time for job j

W_j = the total waiting time for job j

C_j = completion time of job j (time at which job j is finished).

F_j = flow time of job j (the total time job j spends in the shop)

These concepts are related as follows:

$$C_j = r_j + W_j + p_j$$

$$F_j = W_j + p_j$$

Setting due dates for each job allows us to define:

d_j = due date or desired completion date of job j

$L_j = (C_j - d_j)$, the lateness of job j

$T_j = \max(0, L_j)$, the tardiness of job j .

Based on these definitions we have that min-max optimality criteria commonly considered in the literature are:

- | | |
|---|---------------------------|
| (1) minimization of maximum completion time | $C_{\max} = \max_j (C_j)$ |
| (2) minimization of maximum flow time | $F_{\max} = \max_j (F_j)$ |
| (3) minimization of maximum waiting time | $W_{\max} = \max_j (W_j)$ |

When due dates are set for all jobs two other criteria of interest are:

- | | |
|---------------------------------------|-----------------------------|
| (4) minimization of maximum lateness | $L_{\max} = \max_j (L_j)$ |
| (5) minimization of maximum tardiness | $T_{\max} = \max_j (T_j)$. |

Note that any schedule minimizing L_{\max} also minimizes T_{\max} (but not necessarily vice versa).

Defining $c_j(t)$ = the deferral cost or penalty associated
with the completion of job j at time t

we have as a final min-max criterion

$$(6) \text{ minimization of maximum deferral cost } c_{\max} = \max_j (c_j(t))$$

of which (1) through (5) are all special cases.

The min-sum optimality criteria commonly considered in parallel processor problems are:

$$(7) \text{ Minimization of mean completion time } \bar{C} = \frac{1}{n} \sum_j C_j$$

(sum or total completion time)

$$(8) \text{ Minimization of mean flow time } \bar{F} = \frac{1}{n} \sum_j F_j$$

(sum or total flow time)

$$(9) \text{ Minimization of mean waiting time } \bar{W} = \frac{1}{n} \sum_j W_j$$

(sum or total waiting time)

(7), (8), and (9) are really special cases of:

$$(10) \text{ Minimization of the weighted sum of completion } \sum_j \alpha_j C_j$$

times where α_j denotes the relative importance
of job j

$$(11) \text{ Minimization of the weighted sum of flow times } \sum_j \alpha_j F_j, \text{ and}$$

$$(12) \text{ Minimization of the weighted sum of waiting times } \sum_j \alpha_j W_j.$$

Since $C_j = r_j + W_j + p_j$ and $F_j = W_j + p_j$ we have:

$$\sum_j \alpha_j F_j = \sum_j \alpha_j W_j + \sum_j \alpha_j p_j \text{ and } \sum_j \alpha_j C_j = \sum_j \alpha_j r_j + \sum_j \alpha_j F_j$$

which given that $\sum_j \alpha_j p_j$ and $\sum_j \alpha_j r_j$ are schedule independent constants, imply that (10), (11), and (12) are equivalent criteria, as are (7), (8), and (9).

When due dates d_j are set for the jobs, criteria of interest are:

$$(13) \text{ Minimization of mean or total lateness } \bar{L} = \frac{1}{n} \sum_j L_j \text{ and}$$

$$(14) \text{ Minimization of mean or total tardiness } \bar{T} = \frac{1}{n} \sum_j T_j,$$

as well as,

$$(15) \text{ Minimization of total weighted lateness } \sum_j \alpha_j L_j \text{ and}$$

$$(16) \text{ Minimization of total weighted tardiness } \sum_j \alpha_j T_j$$

$$\text{Since } \sum_j \alpha_j L_j = \sum_j \alpha_j C_j - \sum_j \alpha_j d_j \text{ and } \sum_j \alpha_j d_j \text{ is}$$

a schedule independent constant, we conclude that (15) is equivalent to (7), (8), and (9) and (16) is equivalent to (10), (11), and (12).

Another criterion of special interest is:

$$(17) \text{ Minimization of the number (\#) of tardy jobs.}$$

Finally, we note that all eleven min-sum criteria are special cases of

$$(18) \text{ Minimization of } \sum_j c_j(t) \text{ where } c_j(t) \text{ is the penalty or}$$

cost associated with the completion of job j at time t .

A final criterion which arises in problems of a production scheduling context concerns the minimization of total penalty cost associated with processing a set of jobs where the cost is a function

of how much of each job has been completed by time t , ($t = 1, \dots, H$, where H is the scheduling horizon).

2.3 Organization of the Review

The review itself will start with the most general parallel processor scheduling problem (Path 8) and work its way downwards to the most basic parallel processor problem (Path 1). Path 9, while not a parallel processor problem, is included for the sake of completeness and will be commented upon only briefly. For each problem, current results may range from enumeration approaches to network formulations to the existence of scheduling rules be they one pass or iterative. We again note that the emphasis in this review will be on rules and network formulations (i.e., constructional methods which lead directly to an optimal solution).

The results in each section (Path) will be dichotomized into those involving min-max or min-sum optimality criteria. This is done as much to highlight the surprising fact that the majority of the results found in the literature are for parallel processor problems with min-sum optimality criteria as well as providing natural subheadings for the presentation of the results themselves.

2.4 Path 9

Deterministic scheduling problems can be initially delineated into situations where machines are identical versus situations when they are not. The system consists of non-identical processors if the processing time of job j , $p_{j\lambda}$, varies with its machine(s) assignment (i.e., $p_{j\lambda}$ denotes the processing time of job j on machine λ). When

the processors are not identical, machine ordering restrictions may exist which define the various patterns the jobs or any subset of the jobs may follow in their progress towards completion. Two particular cases of interest within this category concern the job shop and flow shop problems. In both cases, it is assumed that some of the jobs require processing on more than one machine before they are completed. Job shop problems are those in which the machine ordering restrictions may take on any form. Flow shop problems require that all jobs be processed in the same machine order and in some instances impose the additional condition that the same job order be preserved on all machines.

As noted previously, a distinguishing feature of parallel processor systems is that machine ordering restrictions are not imposed on the jobs. It then follows that the problems of interest in this review pertain to Paths 8 through 1 in which case we conclude with Path 9. (Reviews of deterministic scheduling problems including those in Path 9 are found in Ashour [3], Bellman [8], Conway, Maxwell and Miller [16], Elmaghraby [26], Maxwell [48], Rinnooy-Kan [61] and Spinner [75].)

2.5 Path 8

This most general parallel processor problem has received scant attention in the literature. The scarcity of results is a by-product of the complexity introduced when job processing times are dependent on the machine assignment as well as the jobs themselves.

2.5.1 Min-Sum Criteria

The most significant result in this area is provided by Horn [40] who shows that when all jobs are simultaneously available the problem of minimizing \bar{F} can be formulated as an assignment problem. An example network for a 3 job two machine problem is given in Figure 2. To construct the network define a node j corresponding to each job and a node $k\ell$ corresponding to the k^{th} from the last position on machine ℓ . From each node j , construct a forward arc $(j, k\ell)$ to each node $k\ell$ for all $j = 1, \dots, n$, $k = 1, \dots, n$ and $\ell = 1, \dots, m$. A cost of $kp_{j\ell}$ is assigned to each arc $(j, k\ell)$, where $p_{j\ell}$ denotes the processing time of job j on machine ℓ . For example, if job 2 is assigned to the third from the last position on machine one its contribution to the total flow time (cost) is $3p_{21}$. Define a source node S and a sink node T where a forward arc (S, j) is constructed for all $j = 1, \dots, n$ and a forward arc $(k\ell, T)$ is constructed for all $k = 1, \dots, n$ and $\ell = 1, \dots, m$. Since each job can only be assigned once a capacity of one is placed on all arcs (S, j) . Analogously, since each position on each machine can only be occupied by one job, a capacity of one is placed on each arc $(k\ell, T)$. When n , the number of jobs, is large, computational difficulties may arise since it is difficult to determine "a priori" a bound on the number of jobs to be assigned to each machine and n positions must be allocated to each machine. Horn provides means for reducing computational difficulties caused by the size of the formulation.

Another result in this area is given by Gupta and Maykut [35] who present a dynamic programming formulation for the problem of minimizing $\sum_j c_j(t)$.

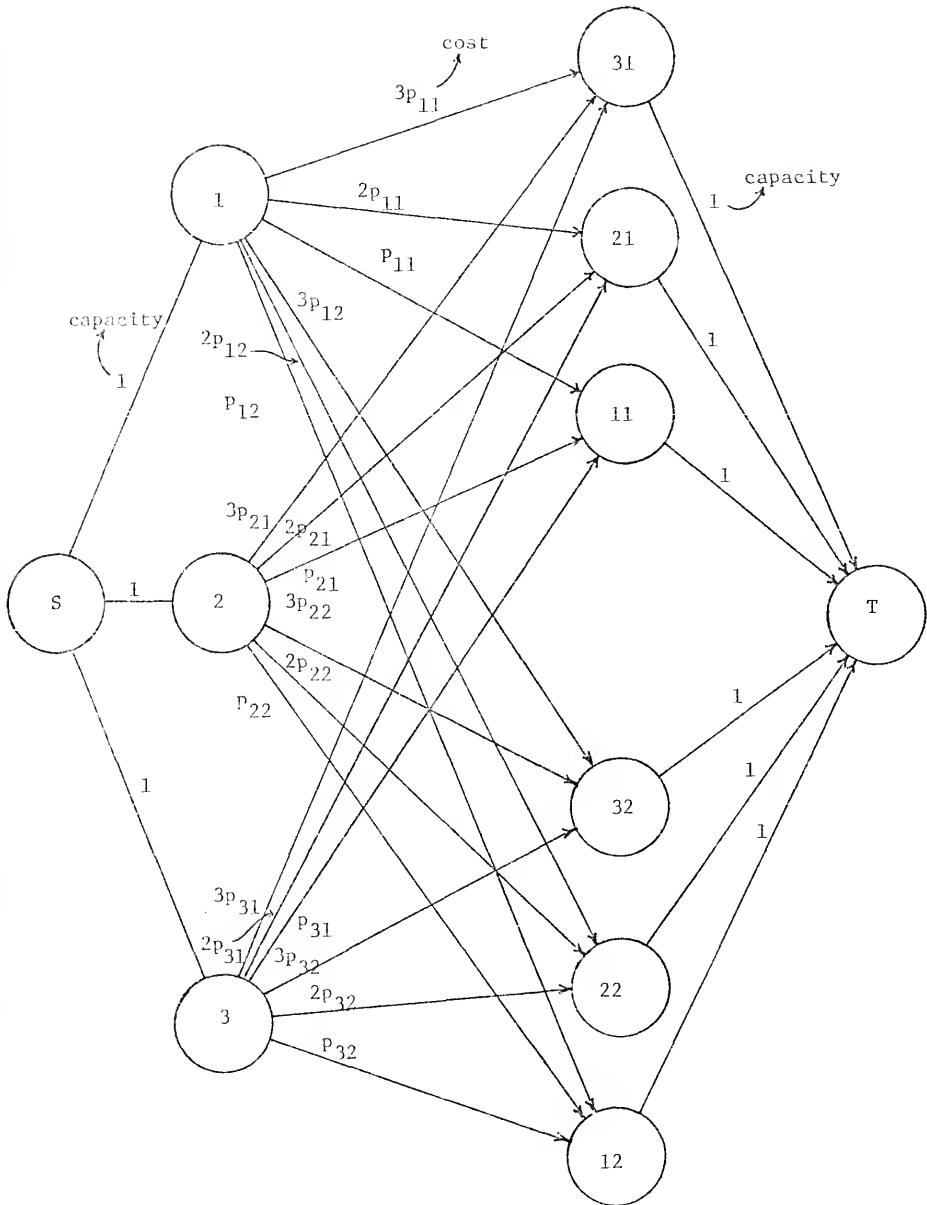


Figure 2: A Three Job, Two Machine Problem

2.5.2 Min-Max Criteria

The problem of minimizing F_{\max} was looked at briefly by McNaughton [50] for the simplest case of three jobs and two machines. In order to solve this case he had to consider 13 different subproblems, an occurrence that did not inspire him (nor any other researchers for that matter) to continue onward. However, there is a special case for which results are attainable.

Consider the problem of scheduling n jobs with identical processing times on m non-identical processors so as to minimize F_{\max} . Since the jobs are identical we have the $p_{j\ell}$ is the same for all j and a given ℓ , $\ell = 1, \dots, m$. (i.e., $p_{j\ell} = p_{\ell}$ for all $j = 1, \dots, n$).

Let x_{ℓ} = the number of jobs assigned to machine ℓ and

p_{ℓ} = the processing time of any job on machine ℓ .

The problem of minimizing F_{\max} can be formulated as:

$$(P-1) \quad \begin{array}{l} \min P(X) \\ \text{subject to} \quad \sum_{\ell=1}^m x_{\ell} = n \\ \\ x_{\ell} \geq 0 \text{ and integer} \end{array}$$

where:

$$P(X) \equiv \max\{p_1 x_1; p_2 x_2; \dots, p_m x_m\}.$$

(P-1) is a single constraint min-max problem similar to those considered by Jacobsen [43] and can be optimally solved using a marginal allocation procedure as follows:

- (i) Set $x_{\ell} = 0$ for all ℓ , and set $i = 1$ (i.e., $P_1(X) = 0$).
- (ii) Assign any job j , from the set of jobs not yet assigned, to machine ℓ^* if:
- $$P_{j^*}(x_{\ell^*} + 1) - P_i(X) = \min_{\ell=1, \dots, m} \{p_{\ell}(x_{\ell} + 1) - P_i(X)\}$$
- (iii) Once a job is assigned to ℓ^* , x_{ℓ^*} is increased by one and all other values of x_{ℓ} remain the same.
- (iv) Let $i = i + 1$ and recompute $P_i(X)$.
- (v) Repeat steps (ii) through (iv) until all jobs are assigned in which case the current value of $P_i(X)$ is optimal.

2.6 Path 7

Even though we now assume that the processors are identical (i.e., $p_{j\ell} = p_j$ for $\ell = 1, \dots, m$), the inclusion of sequence dependent setup times makes this a very difficult problem. In the literature that we considered, no results were found for problems involving min-max criteria although this was not the case for min-sum criteria as we shall now see.

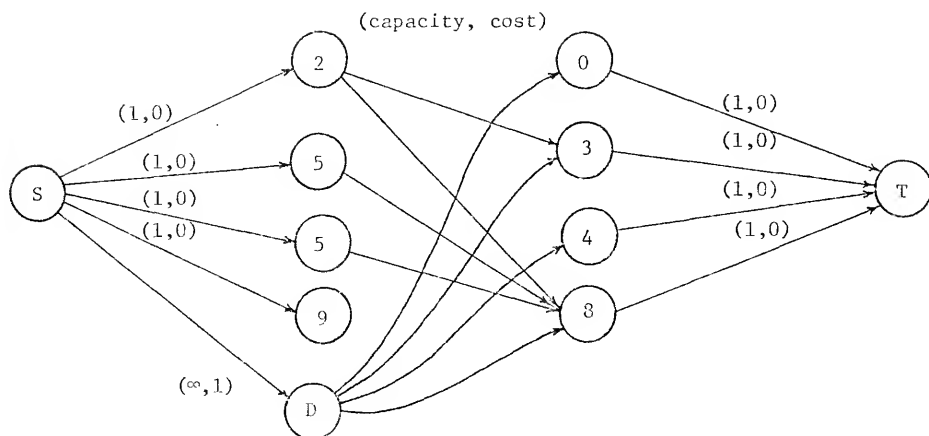
The only solution for the case of more than one processor is provided by Dantzig and Fulkerson [17]. They show that the special case where the objective is the minimization of the number of resources required to complete all jobs by their due dates if $p_j = d_j - r_j$ (i.e., no job slack) can be formulated and solved as a minimum cost network flow problem. This problem is referred to in the literature as the "Tanker Scheduling" problem.

For example, consider the following problem with four jobs where release times, due dates and sequence dependent setup times are given by the following data:

j	r_j	d_j
1	0	2
2	3	5
3	4	5
4	8	9

		s_{ij}				
		j	1	2	3	4
i	j					
1	1	-	1	3	2	
2	1	-	-	-	3	
3	1	-	-	-	1	
4	1	-	-	-	-	

The network is constructed by creating a node d_j corresponding to each job due date, and defining a node r_j corresponding to each job release time. A directed arc (d_i, r_j) is drawn if and only if $d_i + s_{ij} \leq r_j$. The network is completed by adding a source node S , a sink node T and a dummy node D such that a directed arc (S, d_j) with unit capacity is drawn from S to all nodes d_j , a directed arc (D, r_j) is drawn from D to all nodes r_j with infinite capacity and unit cost, and an arc (r_j, T) with unit capacity is drawn from r_j to T for all nodes r_j .



The minimum number of resources required to process all jobs can then be determined by solving a minimal cost flow problem on the above network such that all arcs (r_j, T) from r_j to T are saturated. The number of resources required is given by the flow out of the dummy node D .

The inclusion of sequence dependent setup times is quite a formidable barrier since no longer can the setup times be included as part of the job processing times. The difficulty of this class of problems is appreciated by noting that for the one machine case where the objective is the minimization of the sum of setup times, the problem is equivalent to what is commonly referred to in the literature as the "traveling salesman" problem. Because of the fame or the notoriety, as you wish, of this problem, no need will be found for expounding upon it in this survey. A thorough review of this problem is provided by Bellmore and Nemhauser [9]. The specific role played by the problem in scheduling theory is elaborated upon by Conway, Maxwell and Miller [16] and Elmaghraby [26].

A final criterion considered by Glassey [33] concerns the minimization of the number of changeovers required to have produced $d_j(t)$ units of product j by time t ($t = 1, \dots, T$) where the machine produces one unit of product per time unit and T denotes the scheduling horizon. He combines graph-theoretical and dynamic programming arguments in order to solve the problem. Both the formulation and the results are discussed by Rinnooy-Kan [61].

2.7 Path 6

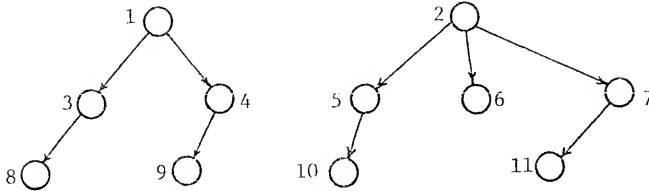
We now assume that setup times are sequence independent in which case they can be included as part of the job processing times. We also assume that the jobs are related by precedence constraints such as those found in CPM and PERT which create a partial ordering of the jobs defined by the precedence relation. As Baker [4] notes, the most commonly used convention " $i < j$ " is used to denote the fact that job i precedes job j (i.e., job j can not begin processing until job i is completed). When $i < j$, job i is called a predecessor of job j and job j is called a successor of job i . Job i is a direct predecessor of job j if $i < j$ and there is no job k such that $i < k < j$.

2.7.1 Min-Sum Criteria

Baker [4] advances the opinion that a collection of precedence related jobs may not be a realistic context for dealing with a mean flowtime (\bar{F}) criterion. He states that in many cases, a predecessor job is important only to the extent that it allows a later job to be processed; the flowtime of the predecessor job may itself be of no importance. He suggests that a more meaningful context might be one in which contributions to the \bar{F} criterion arise only from jobs that have no successors in which case the $\sum_j \alpha_j F_j$ criterion would be more appropriate with $\alpha_j = 0$ for those jobs whose contribution is not meaningful. This belief is apparently shared by those who have researched this problem since current results deal basically with this criterion.

Horn [39] provides an algorithm in which the directed graph representing the precedence constraints is a forest; i.e., a collection

of trees, each with a root node. For example, given the following precedence relationships



there are only two candidates for the first job, jobs 1 and 2. When one of the two is selected, the node as well as all branches leading from it are deleted generating a new set of trees with candidate root nodes. In order to solve the $\sum_j \alpha_j F_j$ problem for precedence constraints of the above form, Horn introduces the notion of a successor set S_j to job j where if:

(i) $k \in S_j$ and $k \neq j$, then $k < j$ and

(ii) if $k \in S_j$ and $i < j$, then either $i \in S_j$ or $i < j$.

His algorithm consists of calculating for each root j

$$\gamma_j = \min_{S_j} (\sum p_j) / (\sum \alpha_j)$$

and scheduling the root job with minimal γ_j . The procedure is then repeated with the new set of roots. Based on Conway, Maxwell and Miller's [16] observation that the pair of sequences that minimize and maximize $\sum_j \alpha_j F_j$ will be antithetical, Horn's algorithm is also optimal for situations where the precedence constraints are upside down trees. The algorithm can be applied after turning the trees upside down again, reversing all directed arcs and replacing α_j by $-\alpha_j$. [61]

Sidney [72] considers the $\sum_j c_j(t)$ problem when $c_j(t)$ is linear. He presents a combinatorial algorithm and proves that a permutation is optimal if and only if it can be generated by the algorithm. He also develops more efficient procedures for special precedence structures such as parallel chains, parallel networks, job modules and rooted trees.

2.7.2 Min-Max Criteria

A fundamental result in this area is provided by T. C. Hu [41] for the special case when all jobs have unit processing time and the graph of ordering restrictions is a tree with all arcs directed towards the root node. Hu's algorithm for minimizing F_{\max} consists of:

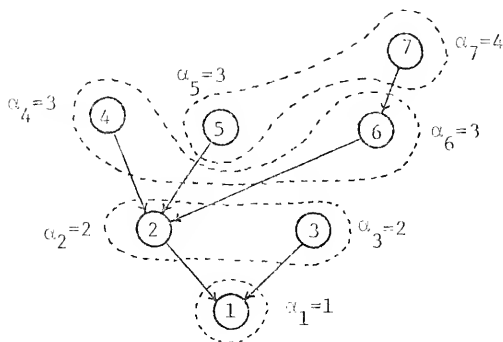
- (i) Labeling the job without any successor by $\alpha_j = 1$.
- (ii) Labeling all other jobs by

$$\alpha_j = 1 + \alpha_k$$

where k is the job that directly succeeds job j in the tree.

- (iii) a. If the number of jobs without predecessors is less than or equal to m , the number of machines, then these are scheduled leaving any excess machines idle.
- b. If the number of jobs without predecessors exceeds m , then those m jobs with the largest α_j values are scheduled.
- (iv) All scheduled jobs and branches incident to them are omitted, and step (iii) is repeated until all jobs are scheduled.

For example, consider the following problem with $n = 7$ jobs and $m = 2$ machines.



Since we have two machines then according to the algorithm we should choose job 7 and either job 4 or 5. Here we arbitrarily choose job 5 and enclose jobs 7 and 5 in a dotted curve. The successive steps are self-explanatory. The jobs are completed in $P_{\max} = 4$ units of time which is clearly minimal.

As Baker [4] observes, Hu's algorithm takes on added significance when it is noted that any job j with integer processing time p_j can be modeled as a series of p_j jobs with unit processing time where each except the last has one direct predecessor. In this case, Hu's algorithm is optimal for jobs with arbitrary processing times and a tree precedence graph when a limited form of job splitting (i.e., one unit of each job can be processed in any period) is allowed.

Zaloom [79] has observed that any general set of precedence constraints can be converted to a tree by removing all but one arc leading out of each node. Therefore, Hu's algorithm provides a lower bound on the problem with general precedence constraints when the same limited form of job splitting is allowed. If Hu's solution to the problem on the tree does not violate any of the extirpated precedences, then it is

optimal for the general network. If some precedences are violated, then what is lacking in this instance is an algorithm to optimally replace precedence arcs that are initially removed.

For the case of arbitrary job processing times and a general graph, Coffman and Muntz [15] present an algorithm for minimizing F_{MAX} on two processors. They allow job splitting and divide the jobs into independent (non-interfering) subsets, ordering within the groups and then combining them. The algorithm does not extend for any larger values of m .

For the single machine problem, Lawler [46] has presented a very efficient algorithm for minimizing $\max c_j(t)$ when $c_j(t)$ is a monotonically non-decreasing penalty assessed if job j is completed at time t . His procedure consists of sequencing job k last if

$$c_k(T) = \min_{j \in S} c_j(T) \quad \text{where } T = \sum_{j \in J} p_j$$

where S denotes the subset of jobs without successors and J denotes the set of all jobs. After job k is selected, it is removed from J and the procedure is repeated to find the job to be scheduled next to last. This process is continued until all jobs have been scheduled. The L_{max} problem can now be solved by letting $c_j(t) = t - d_j$. The T_{max} problem is solved by letting $c_j(t) = \max(t - d_j, 0)$. In these two cases the algorithm becomes equivalent to sequencing the jobs from last to first, always choosing next from among the jobs without successors a job with the latest possible due date.

The difficulty of the m identical parallel processor problem with precedence constraints is brought into perspective by noting that

even with job splitting, the problem is equivalent to what is referred to in the literature as the capacitated or constrained CPM problem with a single type of resource. No attempt will be made here to delve further into this extensively investigated area. The interested reader is referred to Beaulieu and McGinnis [10], Davis [18] and Herroelen [38] for recent state of the art reviews.

2.3 Path 5

Problems in which not all machines are available during the scheduling horizon represent a relatively untouched area in scheduling on parallel processors. Such a condition may arise as an "a priori" restriction on the problem or in instances where the machines themselves move in time as may be the case in the scheduling of jobs in transportation systems. A major contribution of this dissertation is the development of optimal rules for minimizing \bar{L} , L_{\max} , F_{\max} , C_{\max} and other criteria for problems involving the scheduling of freight on a class of fixed route railway systems. In these problems the trains play the role of parallel processors whose availability is subject to a predetermined train schedule. These results are contained in Chapter 4.

With respect to more classical situations we find that in a recent paper Gelders and Kleindorfer [31] consider the problem that arises in trying to minimize overtime and scheduling costs on a single processor with varying capacity. They develop a branch and bound algorithm for dealing with this detailed and complex scheduling problem. We also find that the problem of minimizing $\sum_j c_j(t)$ when the number of machines available varies from period to period can be formulated as an

assignment problem for the special case of jobs with unit processing time. This formulation is presented in Chapter 3.

2.9 Paths 4 and 3

The availability of a job for processing is typically restricted by its arrival or release time into the system r_j , by its due date d_j or both. When $r_j \neq 0$ for some j , the situation is commonly referred to as the case when jobs are not simultaneously available for processing. An analogous situation occurs when jobs must be processed by their due dates (i.e., $T_{\max} \leq 0$).

2.9.1 Path 4

In this section we consider problems where in addition to job availability restrictions it is also required that once processing has commenced on any job it must be carried through until the job is completed. The only results we uncovered here were based on the following min-max problems.

For a single processor and $r_j \neq 0$ for some j it can be noted that the F_{\max} problem is trivially solved by scheduling jobs as soon as possible after they become available. The L_{\max} problem for one machine is not quite so straightforward. Besides being of interest in itself, its importance is also due to its applicability in computing lower bounds for the general job shop problem [14]. Dessouky and Margenthaler [19] solve the problem by initially scheduling the jobs such that job k precedes job q if:

$$r_k + d_k \leq r_q + d_q$$

Branch and bound is then used to move from this initial solution to optimality in a finite number of steps. Other implicit enumeration schemes for this problem and the T_{\max} problem are found in Bratley et al. [13] and Baker and Su [7]. More recently, McMahon and Florian [49] presented a very efficient branch and bound solution such that a complete solution is associated with each node of the enumeration tree.

When both availability restrictions are imposed a problem considered in this area concerns the minimization of F_{\max} subject to $T_{\max} = 0$. A branch and bound procedure is developed by Bratley, Florian and Robillard [12, 13] based on their work for the job splitting version of the same problem to be discussed in Path 3.

2.9.2 Path 3

When job splitting is allowed the question of how often jobs can be split comes into play. Conway et al. [16] have observed that if all the jobs can be processed simultaneously on all m machines, the situation becomes equivalent to processing on a single machine. The m machines working simultaneously on a single job can be viewed as a single machine with m times the power of the basic machines. If the m machines are identical, a pseudo-processing time \hat{p}_j is defined for each job where:

$$\hat{p}_j = \frac{p_j}{m}$$

and these \hat{p} are used whenever the scheduling procedure calls for a processing time.

If it is the case that the jobs have integer processing times and some (or all) jobs can be processed by more than one machine

simultaneously, then it is possible to transform the problem into one where no job is processed simultaneously on more than one machine in each period. (See Chapter 3)

A. Min-Sum Criteria

Dorsey, Hodgson and Ratliff [20, 21, 22] present a network approach for the problem of minimizing the total penalty cost associated with producing a set of products over a finite planning horizon. Letting each product be a job, and the time it takes to produce a product be the processing time, allows their network approach to solve an equivalent parallel processor scheduling problem. The problem they solve concerns the minimization of total penalty cost associated with processing a set of jobs on m identical processors over a fixed time horizon H where:

- (a) It is known how much of each job should be completed by period t ($t = 1, \dots, H$) (i.e., there exists a fixed schedule for completing each job).
- (b) The penalty cost for each job is a function of how much of the job has actually been completed by time t (i.e., a penalty is assessed for being off the fixed schedule).

This formulation is discussed in detail in Chapter 5 where algorithms are developed for parallel processor problems with the same type of penalty cost under min-max criteria.

With the exception of the above problem, results for parallel processor problems within this category are virtually non-existent. For the case of a single processor Schrage and Miller [70] in looking at

single server queuing problems show that a "shortest remaining processing time" (SRPT) rule will minimize \bar{F} . Unfortunately, the rule does not extend to $m \geq 2$ as shown by the following example:

Job (j)	r_j	p_j	$m = 2$
1	0	4	
2	0	5	
3	0	1	
4	1	1	
5	2	1	
6	6	1	
7	6	1	

Extension of the SRPT rule for this example implies that among those jobs available for processing priority is given to those jobs with shortest remaining processing time.

For the data given, the rule yields the following schedule:

	t =	1	2	3	4	5	6	7	8	9	
Machine 1	1	1	1	1	2	2	2	6	2	2	with $\bar{F} = 18$
Machine 2	3	4	5	1	ϕ	ϕ	7	ϕ	ϕ	ϕ	

However, a better schedule is:

	t =	1	2	3	4	5	6	7	
Machine 1	1	2	2	1	1	1	1	6	with $\bar{F} = 17$
Machine 2	3	4	5	2	2	2	2	7	

So while Conway, Maxwell and Miller [16] state that rules for single machine problems with simultaneous release times extend to non-simultaneous release times with job splitting, such fortune does not generalize to parallel processor systems.

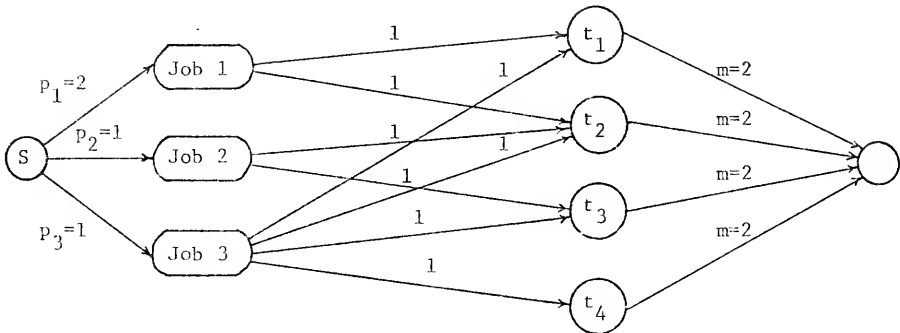
B. Min-Max Criteria

When both nonsimultaneous release times and due date restrictions are imposed on the problem, we find that Bratley, Florian and Robillard [13] have developed a network formulation for minimizing F_{\max} subject to $T_{\max} = 0$. It is best illustrated through the following example.

Let $n = 3$ and $m = 2$ with the following data:

j	p_j	d_j	r_j
1	2	2	1
2	1	3	2
3	3	4	1

Letting T denote the length of the scheduling horizon, suppose $T = \max_j d_j = 4$. Then the problem of scheduling the jobs subject to release time and due date constraints is equivalent to finding a feasible flow on the following network.



That is, all arcs of the form (S, Job j) have capacity p_j , all arcs of the form (t_i, F) have capacity m , and an arc with capacity one is drawn from Job j to t_i if job j can be processed in period t_i (i.e. if $r_j \leq t_i \leq d_j$). If no feasible flow is obtained then T is increased and the procedure is repeated. If a feasible flow is obtained, then T is decreased. The minimum value of T for which a feasible flow is obtained is the optimal value of F_{\max} .

When due date restrictions are not imposed on the jobs a more efficient rule can be applied for minimizing F_{\max} when the jobs are not simultaneously available. This result is presented in Chapter 3.

2.10 Paths 2 and 1

Scheduling with jobs simultaneously available presents the simplest and by far most thoroughly understood problem in the field. Most of the research effort on parallel processor problems in this area is motivated by the success encountered in the solution of single machine problems although the results have not been as gratifying. Inability to extend single machine results stem in part from the fact that properties of single machine problems do not carry over in general for more than one processor. For example, Conway et al. [16] have shown that for any optimality criterion that is strictly a function of job completion times (i.e., regular measures of performance), it is not necessary to consider schedules which involve job splitting. It then follows that the optimal schedule will be one of the $n!$ permutations of the job numbers 1, . . . , n (e.g., SPT rule in section 2.10.1 or "due date" rule in section 2.10.2). This convenient result does

not extend to $m \geq 2$ processors for which job splitting is a relevant consideration in developing optimal schedules even when the jobs are all simultaneously available.

2.10.1 Min-Sum Criteria

A. \bar{F} and $\sum \alpha_j F_j$

Perhaps schedulings' most well known result was first established by Smith [74] who showed that the \bar{F} problem for one machine is solved by the sequence [1], [2], ..., [n] with $p_{[1]} \leq p_{[2]} \leq \dots \leq p_{[n]}$. This result is more commonly known as the SPT, or shortest processing time rule. The rule also minimizes \bar{W} , \bar{L} , \bar{C} and $\frac{1}{n} \sum F_j^\alpha$ ($\alpha > 0$). Smith [74] also showed that the rule can be generalized to solve the $\sum \alpha_j F_j$ problem. The solution consists of the sequence [1], [2], ..., [n] with

$$\frac{p_{[1]}}{\alpha_{[1]}} \leq \frac{p_{[2]}}{\alpha_{[2]}} \leq \dots \leq \frac{p_{[n]}}{\alpha_{[n]}}$$

This "weighted SPT rule" also minimizes $\sum \alpha_j W_j$, $\sum \alpha_j L_j$ and $\sum \alpha_j C_j$.

Smith also considers the problem of minimizing \bar{F} subject to the condition that no job be tardy. His algorithm (generalized by Rinnooy-Kan [61] to cover $\sum \alpha_j F_j$) consists of initially scheduling the jobs according to the "due date" rule (see Section 2.10.2) to determine if at least one schedule exists such that no jobs are tardy. If a schedule does exist, he then reorders the jobs such that job k is assigned to be last in the sequence if

$$(a) \quad d_k \geq \sum_{j=1}^n p_j \quad \text{and}$$

$$(b) \frac{p_k}{\alpha_k} > \frac{p_j}{\alpha_j} \text{ for all } j \text{ with } d_j \geq \sum_{j=1}^n p_j.$$

That is, he allows a job to be in the last position only if (a) this does not cause it to have non-zero tardiness; and (b) if the job has the greatest p_j/α_j ratio of all jobs that could be last without being tardy. Once a job is selected to be last, the procedure is repeated with the remaining $n - 1, n - 2, \dots$, jobs until all jobs are ordered.

In the case of parallel processors McNaughton [50] proved that there exists an optimal schedule without job splitting for both the \bar{F} and $\sum \alpha_j F_j$ problems.

For the \bar{F} problem, Conway et al. [16] have shown that an extension of SPT rule for single machines yields an optimal schedule. Their modified SPT rule initially assigns the m jobs with shortest processing time to the machines. Each time a machine finishes a job, it would be assigned from among those jobs waiting, the job with the shortest processing time.

An interesting property of this rule is that jobs in equivalent positions on different machines can be interchanged without affecting \bar{F} in which case several alternate assignments will yield optimum schedules. The rule also minimizes \bar{W} and \bar{L} . As we have seen in the previous section however, this rule will not extend to parallel processor problems with nonsimultaneous job arrival times even if job splitting is allowed.

The $\sum \alpha_j F_j$ problem on parallel processors was first approached by Eastman, Even and Isaacs [23] who showed that a lower bound on the problem is given by

$$\bar{F}_\alpha(m) \geq \frac{m+n}{m(n+1)} \bar{F}_\alpha(1)$$

where $\bar{F}_\alpha(m)$ is the mean weighted flow time with m machines. Moreover, they show that, in general, no greater lower bound is attainable by exhibiting a set of jobs that attain this bound. At present a solution to this problem has not been obtained. As Baker [4] notes, an optimal solution to the general problem will be characterized by the "weighted SPT" rule on each machine, but the problem arises in deciding how to partition the jobs to the machines. Baker and Merten [6] have explored some elementary properties of this problem and use them to develop three heuristic rules. An experimental comparison of the rules yielded no dominating results.

In a recent paper Miller [54] considers the more complex problem of scheduling n jobs on m parallel processors with the added restriction that exactly q_j machines must be assigned to job j for a continuous interval p_j where job splitting is not allowed. He presents a branch and bound argument for minimizing \bar{F} to arrive at optimal solutions for small problems ($n = 8$).

B. Number of Tardy Jobs

A rather ingenious procedure for solving the problem of minimizing the number of tardy jobs on a single machine was presented by Moore [55]. His algorithm as modified by Hodgson [55] consists of

- (i) Scheduling the jobs by the "due date" rule. If no jobs are tardy, then we terminate.
- (ii) If the job $\{k\}$ is the first tardy job in the due date sequence, then discard job $\{k\}$ where

$$P_{[l]} = \max\{P_{[1]}, P_{[2]}, \dots, P_{[k]}\}.$$

Repeat steps (i) and (ii) until no jobs are tardy.

(iii) Once a non-tardy subsequence has been found, all discarded jobs are added on in any order to complete the optimal sequence.

Sidney [73] has presented a variation of this algorithm for solving this same problem subject to the constraint that some subset of the jobs must be completed on time.

Lawler and Moore [47] developed a functional equation which allows for the formulation of the $\alpha_j^{\#}$ tardy (weighted number of tardy jobs) problem through dynamic programming.

Moore's rule does not extend to the general parallel processor problem. When job splitting is allowed an extension of Moore's rule can be used to solve the special case when all job due dates are identical (i.e., $d_j = d$ for all j). This result is presented in Chapter 3.

$$C. \frac{\sum_j c_j(t)}{j}$$

The problem of minimizing $\sum_j c_j(t)$ and its special cases \bar{T} and $\sum_j \alpha_j T_j$ on a single processor has received extensive treatment in the literature. The $\sum_j \alpha_j T_j$ was first postulated by McNaughton [50] who noted the solution for two extreme cases:

- (a) If a solution exists such that no jobs are tardy, then Jackson's "due date" is optimal.
- (b) If it is the case that for all feasible solutions all jobs would be tardy (i.e. $d_j \geq p_j$ for all j), then Smith's weighted SPT is optimal.

In the absence of these two occurrences the problem becomes formidable even though the generality of the criteria has motivated a multitude of researchers to try their hand at its solution.

The first approach on the \bar{T} problem was carried out by Emmons [28]. He develops two basic theorems that can be used to eliminate subsets of sequences from consideration. He then uses the theorems as elimination rules in the implementation of a branch and bound solution procedure. While various other approaches have been tried (all procedures for solving $\sum_j \alpha_j T_j$ and $\sum_j c_j(t)$ of course apply to \bar{T}), it should be pointed out that Baker and Martin [5] have found a heuristic developed by Wilkerson and Irwin [78] to be the most efficient of current techniques for the \bar{T} problem. Lawler and Moore [47] present a dynamic programming formulation for the case when all jobs are subject to primary and secondary due dates.

Research on the $\sum_j \alpha_j T_j$ problem was initiated by Schild and Fredman [68]. Despite their initial claims, their procedure was found to be non-optimal due to the inadequacy of the pairwise comparison method utilized in their decision rule [16]. Dynamic programming solutions for this problem are in Held and Karp [37], Lawler [45] and Srinivasan [76]. Branch and bound procedures have been developed by Elmaghraby [25], Fisher [29] and Slwimer [71]. Petersen [57] has presented an efficient heuristic based on the use of reordering operations.

The case of non-linear but monotone non-decreasing deferral costs is formulated by Lawler [45] as a dynamic programming problem. Rinnooy-Kan, Lagewag and Lenstra [62] develop a branch and bound approach.

The general non-linear case has been approached by Gotterer [34] and Schild and Fredman [69] through dynamic programming and by Shwimer [71] using branch and bound.

The problem for parallel processors was first addressed by McNaughton [50] who showed that if the cost functions are linear no job splitting is required. Rothkopf [66] came up with the first optimal procedure via a dynamic programming algorithm. Gupta and Walvekar [36] formulated the problem as a 0-1 mixed integer program. Arthanari and Ramamurthy [2] and Elmaghraby and Park [27] have devised branch and bound schemes that generate optimal solutions. Computational results are provided only in the latter reference.

Lawler [45] looked at the problem of non-linear but monotone nondecreasing deferral costs and was able to formulate the special case of uniform processing times as a transportation problem. He also formulated the general non-identical job case as a capacitated transportation problem. The solution to this latter formulation, while not necessarily optimal, provides a lower bound on the optimal solution value.

Root [65] considers the special case when all jobs are subject to a common due date. He develops eight theorems by which non-optimal schedules are eliminated and generates a reduced set of schedules containing an optimal one. However, no systematic procedure is provided for determining which schedule(s) in the set is optimal.

Despite the assault that has been carried out on this problem, the lack of non-enumerative solutions is sufficient testimony to its complexity. This is further highlighted by Karp [44] having included

the $\sum_j \alpha_j T_j$ problem for a single machine on his list of problems that are not expected to have a polynomial bound on their solution time. Even in the \bar{T} problem, it appears that serious theoretical complications are caused by the non-linearity of T_j .

2.10.2 Min-Max Criteria

A. F_{\max}

The F_{\max} problem with job splitting on m identical parallel processors was first solved by McNaughton [50] who observed that a necessary condition for a schedule of value $F_{\max} = T^*$ to be optimal is that:

$$T^* = \max \left(\max (p_j); \left\langle \frac{\sum_{j=1}^n p_j}{m} \right\rangle \right)$$

where $\langle \cdot \rangle$ denotes the smallest integer greater than or equal to \cdot . McNaughton's procedure for generating a schedule of value T^* (as explained by Baker [4]) consists of:

- (i) Selecting any job to begin on machine one at time zero.
- (ii) Choosing any unscheduled job and scheduling it as early as possible on the same machine. This step is repeated until the machine is occupied beyond T^* , or until all jobs are scheduled.
- (iii) Reassigning the processing scheduled beyond T^* to the next machine instead, starting at time zero and returning to step (ii).

In Chapter 3 a "longest remaining processing time" rule for minimizing F_{\max} even when jobs are not simultaneously available is

developed which provides an alternate procedure to McNaughton's rule for simultaneous job availabilities.

B. L_{\max}

For single machine problems, Jackson [42] has shown that both the L_{\max} and T_{\max} problems are solved by the sequence [1], [2], ..., [n] where:

$$d_{[1]} \leq d_{[2]} \leq \dots \leq d_{[n]}$$

(i.e., schedule the jobs in nondecreasing due date order).

The classic result is commonly referred to as the "due date" rule.

When job splitting is allowed it is possible to develop an efficient optimal rule for minimizing L_{\max} on parallel processors when all jobs are simultaneously available. It is at this point that our review terminates and the development of rules for classical parallel problems begins in Chapter 3.

CHAPTER 3

SCHEDULING RULES FOR SOME PARALLEL PROCESSOR PROBLEMS

3.1 Introduction

When scheduling on a single machine, Conway et al. [16] have shown that situations where job splitting is allowed need not be considered. The most basic single machine problem is then the no job splitting problem without restrictions on job availabilities. Such is not the case for parallel processors where it is often true that better schedules can be obtained by permitting jobs to be processed by more than one machine or by different machines in different periods. The job splitting assumption, while uncommon in problems of a machine shop nature, acquires relevance in instances where machine change over times are practically instantaneous (e.g., computer systems) or where job processing times are relatively long (i.e., days, weeks) as often-times occurs in resource allocation problems modeled as parallel processor systems.

In Section 3.2 we consider the most basic parallel processor problem where job splitting is allowed and no restrictions are imposed on job availabilities. The problem is formulated in a general context which is used to develop an optimal rule for minimizing maximum lateness when all jobs are assumed simultaneously available for processing. The

rule consists essentially of assigning priority to those jobs which have the "greatest potential lateness" or the "smallest remaining slack" at the beginning of each period in the scheduling horizon. It is also shown that Jackson's "due date" algorithm (Section 2.10.2.B) for one machine problems is a special case of this rule.

In Section 3.3 we drop the assumption that all jobs are simultaneously available. In this situation the results of Section 3.2 extend only to the problem of minimizing maximum flow time where a "longest remaining processing time" rule is found optimal. Properties and relationships of this rule to other problems in the literature are discussed. Finally a very efficient heuristic is developed for the maximum lateness problem based on the results of Section 3.2. Computational results are also presented.

As noted in Section 2.9.2, an important question in job splitting concerns how often the jobs can be split. In Section 3.4, a straightforward transformation is presented which allows the results of Sections 3.2 and 3.3 to be applied to situations where some or all of the jobs can be processed by more than one machine simultaneously.

In Section 3.5 two min-sum parallel processor problems are presented. It is first shown that the problem of $\min \sum_j c_j(t)$ for the special case of unit processing time can be formulated as an assignment problem. An algorithm for minimizing the number of tardy jobs when jobs are subject to a common due date and job splitting is allowed is also presented based on the results of Section 3.3.

In Section 3.6 repeated application of a necessary condition is used to present an algorithm for minimizing $\max_j c_j(t)$.

Finally, a brief discussion of the results of this chapter is presented in Section 3.7.

3.2 Scheduling with Job Splitting and No Restrictions on Job Availabilities

In this section a scheduling rule for minimizing maximum lateness on m parallel processors is presented. We first consider the scheduling horizon to be divided into discrete periods such that the length of any period corresponds to one unit of processing. We then allow the jobs to be split in the sense that any job can be processed by any machine in any given period and it is not required that once a job's processing has commenced it must be left on the machine until completed. It is required, however, that no job be processed by more than one machine in the same period.

The rule consists of determining the "potential lateness" or "remaining slack" for all jobs not yet completed at the beginning of each period. The rule then assigns priority to those on jobs with "greatest potential lateness" or "smallest remaining slack." The values are updated at the beginning of each period and the procedure is repeated until all jobs have been scheduled.

3.2.1 General Formulation

Given n jobs to be processed on m identical parallel processors such that job splitting is allowed and all jobs are simultaneously available (i.e., $r_j = 0$ for all j):

Define: $\underline{p}_j(t)$ = the processing time remaining for job j at the beginning of period t .

A feasible schedule at the beginning of t: one that satisfies the following restrictions:

- (i) No job is processed on more than one machine during the same period;
- (ii) Jobs are changed only at the beginning of periods; and
- (iii) $p_j(T + 1) = 0$, for all j , where T is the final period.

Let $X(t) = \|\|x_{jt'}\|\|$; denote a $(t - 1)$ by m matrix representing a feasible schedule generated up to the beginning of period t where:

$$x_{jt'} = \begin{cases} 1 & \text{if job } j \text{ is processed on a machine during period } t' \\ & (t' \leq t - 1) \\ 0 & \text{otherwise, and,} \end{cases}$$

$\sum_{t'=1}^{t-1} x_{jt'}$ = the number of periods that job j has been processed in a feasible schedule $\lambda(t)$

Let

$$C(X(t)) = \{j | p_j(t) > 0 \text{ in a schedule } X(t)\}$$

We define

$I(X(t))$ = the total idle time on all m machines in a schedule $X(t)$ where:

$$I(X(t)) = m(t - 1) - \sum_{j=1}^n \sum_{t'=1}^{t-1} x_{jt'}$$

Associated with each job j is a "potential lateness function"

$L_j(X(t))$ defined recursively as follows:

$L_j(X(1)) = b_j$ where b_j is a known constant, and,

$$L_j(X(t)) = \begin{cases} L_j(X(t-1)) & \text{if job } j \text{ is either on a machine} \\ & \text{during period } t-1 \text{ or if} \\ & j \notin C(X(t)) \\ L_j(X(t-1)) + 1 & \text{otherwise} \end{cases}$$

A schedule $\bar{X}(t)$ will be called a t -optimal schedule if for a given t and for all j :

$$\mathcal{L}(\bar{X}(t)) \leq \mathcal{L}(X(t))$$

for all feasible schedules $X(t)$ where:

$$\mathcal{L}(X(t)) = \max_{j=1, \dots, n} \{L_j(X(t))\}$$

It will subsequently be shown that for any problem adhering to the previous formulation, the following "greatest potential lateness" rule will generate a t -optimal schedule.

Let $|\cdot|$ indicate the number of elements in the set \cdot . For any $t' = 1, 2, \dots, t$ schedule all $j \in C(\bar{X}(t'))$ so that:

- (i) If $|C(\bar{X}(t'))| \leq m$, then process all jobs in $C(\bar{X}(t'))$.
- (ii) If $|C(\bar{X}(t'))| > m$, then process any m jobs in $C(\bar{X}(t'))$ such that if job q is processed and job r is not, then

$$L_q(\bar{X}(t')) \geq L_r(\bar{X}(t'))$$

(i.e., process jobs $1, \dots, m$ where

$$L_{[1]}(\bar{X}(t')) \geq L_{[2]}(\bar{X}(t')) \geq \dots \geq L_{[m]}(\bar{X}(t')) \geq \dots \geq L_{[l]}(\bar{X}(t))$$

where

$l \leq n$ and $[j]$ denotes the job with the j^{th} largest "potential lateness function" value.

Properties of Schedules Generated by the "Greatest Potential Lateness" (GPL) rule:

Suppose t^* is any period such that at the beginning of t^* , there exist at least $m + 1$ jobs with $L_{[1]}(X^*(t^*)) = L_{[2]}(X^*(t^*)) = \dots = \dots = L_{[m+1]}(X^*(t^*)) = \mathcal{L}(X^*(t^*))$ where $X^*(t^*)$ denotes any schedule generated by the GPL rule up to the beginning of t^* . It then follows that an increase in the value of $(X^*(t))$ occurs at the beginning of $t^* + 1$ (i.e., $\mathcal{L}(X^*(t^* + 1)) > \mathcal{L}(X^*(t^*))$).

Define Q^* to be the set of jobs j such that $L_j(X^*(t^*)) = \mathcal{L}(X^*(t^*))$, where $|Q^*| \geq m + 1$.

Property 3.1: If there exists a schedule $\hat{X}(t^*) \neq X^*(t^*)$ such that

$$\mathcal{L}(\hat{X}(t^* + 1)) < \mathcal{L}(X^*(t^* + 1))$$

then it must be the case that for at least on job $j \in Q^*$,

$$\sum_{t=1}^{t^*-1} \hat{x}_{jt} > \sum_{t=1}^{t^*-1} x_{jt}^* .$$

Proof: If $\sum_{t=1}^{t^*-1} \hat{x}_{jt} \leq \sum_{t=1}^{t^*-1} x_{jt}^*$ for all $j \in Q^*$, then

$$L_j(\hat{X}(t^*)) \geq \mathcal{L}(X^*(t^*)) \text{ for all } j \in Q^* . \text{ Since } |Q^*| \geq m + 1,$$

processed during period f .) Let $X'(t^*)$ be a schedule identical to $X^*(t^*)$ except that job i is not processed in period f .

Since
$$\sum_{t=1}^{t^*-1} x'_{it} = \sum_{t=1}^{t^*-1} x^*_{it} - 1,$$
 then

$L_i(X'(f)) \geq L_i(X^*(f))$ which implies $L_i(X'(f)) \geq L_j(X^*(f))$ for some job $j \in Q^*$. Since job i is not processed from f to t^* , then for all \hat{t} , where $f \leq \hat{t} \leq t^*$, $L_i(X'(\hat{t} + 1)) = L_i(X'(\hat{t})) + 1$, whereas $L_j(X^*(\hat{t} + 1)) \leq L_j(X^*(t)) + 1$. It then follows that for $t = t^*$, $L_i(X'(t^*)) \geq L_j(X^*(t^*))$ for all $j \in Q^*$. Finally, let $\hat{X}(t^*) \neq X'(t^*)$ be any other schedule such that
$$\sum_{t=1}^{t^*} \hat{x}_{it} < \sum_{t=1}^{t^*} x^*_{it}.$$

Note that for i incomplete by t^* , $L_i(\hat{X}(t^*))$ is solely a function of the number of periods i is worked on up to the beginning of t^* , but not of when i is worked on. Therefore

$$L_i(\hat{X}(t^*)) \geq L_i(X'(t^*)) \geq \mathcal{L}(X^*(t^*)). \quad \square$$

Property 3.3: In order to show that the GPL rule is t^* -optimal, it suffices to show that for any other schedule $\hat{X}(t^*)$ having
$$\sum_{t=1}^{t^*-1} x_{jt} > \sum_{t=1}^{t^*-1} x^*_{jt}$$
 for some job $j \in Q^*$, it must be the case that
$$\sum_{t=1}^{t^*-1} x_{it} < \sum_{t=1}^{t^*-1} x^*_{it}$$
 for some job i .

Proof: The result follows directly from properties 3.1 and 3.2. \square

Property 3.3 provides a condition whereby the "greatest potential lateness" rule is t -optimal for all periods t with the property of

t^* . In general, it is not a straightforward task to show that Property 3.3 will hold. However, the following property characterizes a situation under which the conditions of Property 3.3 will always hold.

Property 3.4: If $X^*(t^*)$ is such that $I(X^*(t^*)) \leq I(X(t^*))$ for all $X(t^*) \neq X^*(t^*)$ then property 3.3 holds and $X^*(t^*)$ is a t^* -optimal schedule.

Proof: Let $X^*(t^*)$ be a schedule generated by the GPL rule such that $I(X^*(t^*)) \leq I(X(t^*))$ for all $X(t^*) \neq X^*(t^*)$. Assume there exists a schedule $\hat{X}(t^*) \neq X^*(t^*)$ such that $\mathcal{L}(\hat{X}(t^* + 1)) < \mathcal{L}(X^*(t^* + 1))$ that does not satisfy property 3.3 (i.e., $\hat{X}(t^*)$ is such that

$$\sum_{t=1}^{t^*-1} x_{jt}^* \quad \text{for some } j \in Q^* \text{ and } \sum_{t+1}^{t^*-1} \hat{x}_{jt} = \sum_{t=1}^{t^*-1} x_{jt}^* \quad \text{for all other jobs}$$

i). Then by processing job j at least one more period before t^* , total idle time is reduced and hence, a contradiction exists. \square

Finally, it should be noted that for any $t \neq t^*$, $\mathcal{L}(X^*(t)) = \mathcal{L}(X^*(t + 1))$ and the GPL rule is clearly optimal.

3.2.2 Minimization of Maximum Lateness (L_{\max})

It will now be shown that the problem of minimizing L_{\max} on m parallel processors with job splitting and simultaneous job availabilities can be adapted to the previous formulation and optimally solved with the GPL rule.

For all jobs $j = 1, \dots, n$, let:

$p_j(t)$ = processing time remaining for job j at the beginning of period t such that:

$p_j(1)$ = total processing time for job j , and for $t \geq 2$,

$$p_j(t) = \begin{cases} p_j(t-1) - 1 & \text{if job } j \text{ is on a machine during} \\ & t-1 \text{ or if } j \in C(X(t)). \\ p_j(t-1) & \text{otherwise} \end{cases}$$

In addition let:

$d_j(t)$ = the number of periods remaining at the beginning of period t until job j is due where:

$d_j(1)$ = the original due date for job j and

$d_j(t) = d_j(t-1) - 1$, for $t \geq 2$.

Finally, define the "potential lateness function" $L_j(X(t))$ such that:

$$L_j(X(1)) = p_j(1) - d_j(1) \text{ and}$$

$$L_j(X(t)) = p_j(t) - d_j(t) \text{ for } t \geq 2$$

It then follows from the definitions of $p_j(t)$ and $d_j(t)$ that:

$$L_j(X(t)) = \begin{cases} L_j(X(t-1)) & \text{if job } j \text{ is either on a machine} \\ & \text{during period } t-1 \text{ or if } j \notin C(X(t)) \\ L_j(X(t-1)) + 1 & \text{otherwise} \end{cases}$$

Rule 3.1: An optimal rule for the L_{\max}, T_{\max} problem consists of scheduling for any $t' = 1, 2, \dots, t$ all $j \in C(\bar{X}(t'))$ so that

- (i) If $|C(\bar{X}(t'))| \leq m$, then all jobs in $C(\bar{X}(t'))$ are processed.
- (ii) If $|C(\bar{X}(t'))| > m$, then process any m jobs in $C(\bar{X}(t'))$ such that if job q is processed and job r is not, then

$$L_q(\bar{X}(t')) \geq L_r(\bar{X}(t')) \text{ or equivalently}$$

$$p_q(t') - d_q(t') \geq p_r(t') - d_r(t') \text{ in the } t' \text{ optimal schedule } \bar{X}(t').$$

Proof: Optimality of the rule follows from the observation that since $r_j = 0$ for all j , $I(\bar{X}(t^*)) = 0 \leq I(X(t^*))$ for all feasible schedules $X(t^*) \neq \bar{X}(t^*)$ and any period t^* (where t^* is as defined previously). Therefore, Property 3.4 holds and the optimality condition of Property 3.3 is satisfied. \square

We also note that an equivalent formulation of Rule 3.1 consists in giving priority to job q if:

$$d_q(t') - p_q(t') \leq d_r(t') - p_r(t')$$

in which case the decision is based on favoring those m jobs with smallest remaining slack at the beginning of each period.

An Example: Consider the following problem requiring the scheduling of $n = 5$ jobs on $m = 2$ parallel processors where the data generated by the GPL rule are displayed below.

t	1	2	3	4	5
$p_a(t)$	3	3	3	3	
$p_b(t)$	1	0	-	-	
$p_c(t)$	2	1	0	-	
$p_e(t)$	1	1	1	0	
$p_f(t)$	5	5	4	3	

$d_a(t)$	5	4	3	2
$d_b(t)$	2	-	-	-
$d_c(t)$	2	1	-	-
$d_e(t)$	3	2	1	-
$d_f(t)$	6	5	4	3
$p_a(t) - d_a(t)$	-2	-1	0	1
$p_b(t) - d_b(t)$ *	-1	-	-	-
$p_c(t) - d_c(t)$ *	0	*	0	-
$p_e(t) - d_e(t)$	-2	-1	*	0
$p_f(t) - d_f(t)$ *	-1	*	0	0
$ C(X(t)) $	5	4	3	2

where * denotes the jobs which are candidates for scheduling at each period.

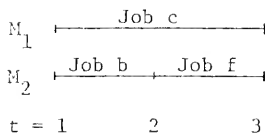
t = 1: Job c and either job b or f must be processed during t = 1.

Arbitrarily select job b.

t = 2: Job b is now completed and need no longer be considered (i.e., $C(\bar{X}(t)) = 4$).

Jobs c and f must be processed during t = 2.

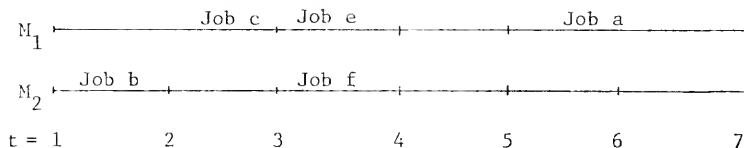
t = 3: Job c is now completed and is no longer considered. $\mathcal{L}(X(t^*)) = 0$ and $|Q^*| = 3 \neq m = 2$, so t = 3 is a period t* at which an increase in $\mathcal{L}(X(t^*))$ will occur at t* + 1 = 4. Jobs a, e and f $\in Q^*$. The schedule up to the beginning of t* = 3 consists of:



$I(X(t^*)) = 0$ and it can be readily verified that if either jobs a, e, or f were worked on one more period before $t^* = 3$, it would imply that $\mathcal{L}(X(t^*)) \geq 0$. (Property 3.2).

Arbitrarily select jobs e and f for processing at $t = 3$.

$t = 4$: Job e is completed and no longer considered. At this point $|C(X(4))| = 2 = m$ and $r_j = 0$ for all j imply that no further decisions are necessary. Both remaining jobs a and f are processed until completed. An optimal schedule with $L_{\max} = T_{\max} = 1$ is given by:



3.2.3 The "Due Date" Rule as a Special Case of the "Greatest Potential Lateness" Rule

In Section 2.10.2.B, Jackson's "due date" rule for minimizing $L_{\max}(T_{\max})$ on a single machine was shown to consist of sequencing the n jobs so that $d_{[1]} \leq d_{[2]} \leq \dots \leq d_{[n]}$. While his result is based on the assumption of no job splitting, application of the GPL rule to the one machine problem allows one to arrive at the same conclusion.

Property 3.5: Any n job single machine problem with simultaneous job arrivals and job splitting can be converted into an equivalent N job problem without job splitting as follows:

Split job j (for all $j = 1, \dots, n$) into p_j identical jobs each with processing time $p_{j,1} = 1$ and due date $d_{j,1} = d_j$. The original L_{\max} problem with job splitting is equivalent to an N ($N = \sum_{j=1}^n p_j$) job single machine problem subject where all jobs have unit processing times and due dates $d_{j,1}$.

Applying the GPL rule to the N job problem results in scheduling, for any period t , job q' versus job r' if:

$$p_{q'} - d_{q'} \geq p_{r'} - d_{r'} \quad \text{for all } r' \neq q' .$$

Since $p_{q'} = p_{r'} = 1$ for all q' and r' , the rule becomes equivalent to selecting job q if

$$d_{q'} \leq d_{r'} \quad \text{for all } r' \neq q' .$$

Since there is only one machine, the original job q will have priority until it is completed. Therefore, no job in the optimal schedule will be split and all original jobs will be scheduled in due date order.

3.3 Non-Simultaneous Job Arrivals

When the jobs are not assumed to be simultaneously available (i.e., $r_j \neq 0$ for some j), the rule of Section 3.2 minimizes L_{\max} only for the special case of a single machine. We will show however that the problem of minimizing F_{\max} can be solved by an extension of the GPL rule.

3.3.1 Minimization of Maximum Flow Time (F_{\max})

When due date restrictions are not imposed on the jobs but the jobs are not simultaneously available for processing, the following

"longest remaining processing time" (LRPT) rule provides an efficient procedure for minimizing F_{\max} .

Consider the scheduling horizon to be divided into discrete periods such that the length of each period corresponds to one unit of processing time. We again define:

$p_j(t)$ = the processing time remaining for job j at the beginning of period t , where

$$p_j(t) = \begin{cases} p_j(t-1) - 1 & \text{if job } j \text{ is processed during period} \\ & t-1 \\ p_j(t-1) & \text{otherwise} \end{cases}$$

$$C(X(t)) = \{j \mid p_j(t) > 0 \text{ in the schedule } X(t)\}$$

In addition define:

$$A(t) = \{j \mid r_j \leq t\} \quad (\text{i.e., } A(t) \text{ is the set of all jobs available for processing at or before the beginning of period } t.)$$

Rule 3.2: Any schedule $X^*(t)$ which has the property that at the beginning of any period t , all jobs available and not yet completed (i.e., all $j \in \{A(t) \cap C(X^*(t))\}$) are scheduled such that:

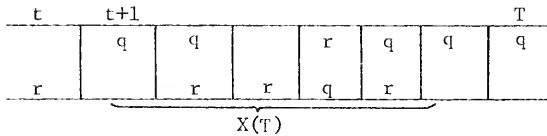
- (i) If $|A(t) \cap C(X^*(t))| \leq m$, then all jobs in $A(t) \cap C(X^*(t))$ are processed.
- (ii) If $|A(t) \cap C(X^*(t))| > m$, then any m jobs in $A(t) \cap C(X^*(t))$ are processed such that if job q is processed and job r is not,

$$p_q(t) \geq p_r(t)$$

minimizes F_{\max} .

The rule translates to assigning priority at the beginning of each period to those m available jobs with longest remaining processing time. The procedure starts with $t = 1$ and is repeated until the beginning of period T where T is such that $|C(X^*(T))| = 0$.

Proof: Consider the beginning of any period t and any two jobs q and r such that either q or r may be processed on a machine during t but not both. Assume that $p_r(t) < p_q(t)$ but that r is selected for processing during t . Let $X(T)$ be any arbitrary schedule after t . For example,



where $p_q(t) = 6 > 5 = p_r(t)$.

At the beginning of t , the maximum number of overlaps in $X(T)$ between jobs q and r (i.e., the maximum number of periods that q and r could be processed simultaneously) is given by:

$$\max OL(t) \leq \min(p_q(t), p_r(t)) = p_r(t).$$

However, since r is processed during t and q is not, the maximum number of overlaps from $t + 1$ to T in $X(T)$ is given by:

$$\max OL(t + 1) \leq p_r(t + 1) = p_r(t) - 1.$$

This implies that there is at least one period after t where q is processed and r is not. Hence r and q can be interchanged at t , the schedule $X'(T)$ after the interchange is also feasible and F_{\max} in $X'(T)$ is less than or equal to F_{\max} in $X(T)$.

Now given any arbitrary schedule, starting at $t = 1$ the jobs can be interchanged so that they satisfy the "longest remaining processing time" rule. The interchange procedure is repeated for $t = 2, 3, \dots, T$ to generate a schedule $X^*(T)$. Since all schedules which satisfy the LRPT rule have the same value of F_{\max} , the result follows. \square

This "longest remaining processing time" rule for minimizing F_{\max} when jobs are not simultaneously available of course provides an alternate procedure to McNaughton's rule (Section 2.10.2.A) for the F_{\max} problem with simultaneous job availabilities. In this case the LRPT rule simplifies to:

Starting with $t = 1$

- (i) Rank the jobs so that $p_{[1]}(t) \geq p_{[2]}(t) \geq \dots \geq p_{[n]}(t)$.
- (ii) Schedule the first m jobs on the machines in period t .
- (iii) Let $t = t + 1$, and update all processing times.

where again

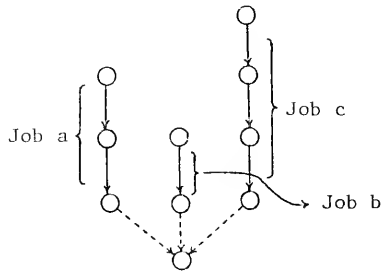
$$p_j(t) = \begin{cases} p_j(t-1) - 1 & \text{if job } j \text{ is} \\ & \text{processed during} \\ & \text{period } t - 1 \\ p_j(t-1) & \text{otherwise} \end{cases}$$

- (iv) Repeat steps (i), (ii), (iii) until all jobs are completed.

In Section 2.7.2, a procedure by T. C. Hu for minimizing F_{\max} when jobs are subject to precedence constraints was discussed. Hu's algorithm was based on the assumption that all jobs have unit processing time and the precedence graph is a tree with all arcs directed towards the root node. In the problem considered in this section we assume that the jobs are not constrained by any precedence relationships, but that

job processing times are arbitrary integers. However, it can be shown that T. C. Hu's procedure can be applied to the problem in this section and that Hu's algorithm is equivalent to Rule 3.2.

Again observe that any job j with integer processing time p_j can be modeled as a series of p_j jobs with unit processing time, where each except the last has one direct predecessor. For example, if we have three jobs with $p_a = 3$, $p_b = 2$ and $p_c = 4$, the problem is equivalent to minimizing F_{\max} on the following precedence graph:

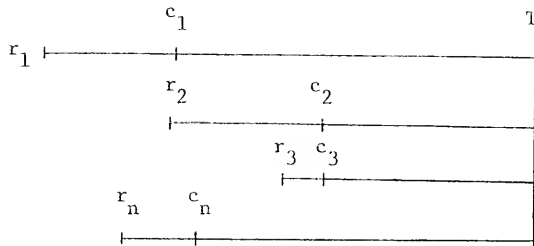


where each original job is divided into p_j subjobs of unit processing time. The precedence relationships assure that no original job is worked on more than once in the same period and the jobs are tied to a root node to complete the tree. Each path in the precedence graph corresponds to a particular job in the original problem. T. C. Hu's algorithm leads one to select, at each step, those m subjobs without predecessor from each of the m longest paths. With respect to the original problem without precedences, Hu's algorithm is equivalent to processing, at each step, one unit of those m jobs with longest remaining processing time, in which case the procedure is equivalent to Rule 3.2.

To complete our discussion on the F_{\max} problem with non-simultaneous job arrivals we note that an alternate approach has been suggested

by Robillard [64]. Let r_1, r_2, \dots, r_n denote the job release times.

Let T denote any horizon length so that all jobs can be completed by T



Starting from T and working backwards, minimize the maximum lateness of all jobs (L_{\max}) by letting the job release times play the role of due dates. Let c_1, c_2, \dots, c_n denote the completion times of the jobs after a schedule minimizing L_{\max} has been constructed. A schedule that minimizes F_{\max} is obtained by shifting the L_{\max} schedule to the left by an amount equal to $\min_{j=1, \dots, n} c_j - r_j$. It then follows

that the L_{\max} problem with simultaneous job arrivals and the F_{\max} problem with non-simultaneous job arrivals are equivalent. If job splitting is allowed, then the "greatest potential lateness" rule (Rule 3.1) can be used with Robillard's observation to provide yet another way of minimizing F_{\max} when $r_j \neq 0$ for some j . Robillard's procedure does not require job splitting but is in fact limited to job splitting cases since optional rules have yet to be developed for either the L_{\max} or non-simultaneous F_{\max} problems when job splitting is not allowed.

3.3.2 Minimization of Maximum Lateness (L_{\max})

The "greatest potential lateness" rule does not generalize to problems where $m \geq 2$ for minimizing L_{\max} . Failure of the rule when

$r_j \neq 0$ for some j centers around a tradeoff between potential lateness $p_j(t) - d_j(t)$ and remaining processing time $p_j(t)$ which cannot be resolved unilaterally. Since neither criterion is dominant an optimal measure for assigning job priorities cannot be attained. (See Appendix 2 for a discussion of this occurrence).

Rule 3.1 is extended to provide what results in a very efficient heuristic for the L_{\max} problem when $r_j \neq 0$ for some j . The heuristic employs the following rule:

For any $t' = 1, 2, \dots, t$, schedule all $j \in \{A(t') \cap C(X(t'))\}$ such that:

- (i) If $|A(t') \cap C(X(t'))| \leq m$, then process all jobs in $A(t') \cap C(X(t'))$.
- (ii) If $|A(t') \cap C(X(t'))| > m$, then process any m jobs in $A(t') \cap C(X(t'))$ such that if job q is processed and r is not, then either
 - (a) $p_q(t') - d_q(t') > p_r(t') - d_r(t')$ or
 - (b) $p_q(t') - d_q(t') = p_r(t')$ and $p_q(t') \geq p_r(t')$

where the arbitrariness of selecting jobs with equal potential lateness functions is resolved by picking the one with longer remaining processing time.

In order to test the efficiency of the heuristic an experiment was conducted in which 279 different problems were run. The heuristic was programmed in APL/360 [32] and problem sizes of up to 80 jobs on 5 machines were generated. Solutions obtained by the algorithm were initially compared to a lower bound on the problem. Those that did not

obtain the lower bound value were tested for optimality by not allowing any job to be processed beyond the furthest period such that $L_j < L_{\max}$ generated by the heuristic, for all j . If no feasible schedule exists under these conditions, then the schedule generated by the heuristic is optimal. The heuristic proved optimal in 277 out of the 279 problems. Computations efficiency of the heuristic was tested by running an 80 job 2 machine problem which was optimally solved in 16 seconds on an IBM/370 system. Complete details and discussion of the experiment are found in Appendix 1.

For the case of a single machine, the following property can be used to show that the above heuristic is in fact optimal when jobs are not simultaneously available.

Property 3.6: Any n job single machine problem with job splitting and non-simultaneous release times can be converted into an equivalent N job problem as follows:

Split job j (for all $j = 1, \dots, n$) into p_j identical jobs with processing time $p_j = 1$, due date $d_j = d_j$ and release time $r_j = r_j$. The original L_{\max} problem is equivalent to an N ($N = \sum_{j=1}^n p_j$) job single machine problem subject to the new processing times, due dates and release times.

The two selection criteria call for favoring job q over job r for any $t' = 1, \dots, t$ of:

$$(a) \quad p_{q'}(t') - d_{q'}(t') \geq p_{r'}(t) - d_{r'}(t) \text{ or}$$

$$(b) \quad p_{q'}(t) \geq p_{r'}(t)$$

where both $q', r' \in \{A(t') \cap C(X(t'))\}$. However, since $p_{q'} = p_{r'} = 1$

no tradeoff between (a) and (b) will ever occur and the optimal priority criteria selects q' over r' if

$$d_{q'}(t) \leq d_{r'}(t)$$

a result previously noted in its latter form in [24].

3.4 Simultaneous Processing by More than One Machine

In Section 2.9.2 we discuss Conway, Maxwell and Miller's [16] observation with respect to situations where all jobs can be processed by all m identical machines simultaneously. In that instance it is shown that such an assumption is equivalent to scheduling on a single machine with m times the power of the basic machine.

We again consider our scheduling horizon to be divided into discrete periods where the length of each period equals one unit of processing time. If it is the case that some (or all) jobs can be processed by more than one machine in the same period, then the following transformation can be used to convert the problem into one where no job is processed by more than one machine in each period.

Consider any job j ($j = 1, \dots, n$) with integer processing time p_j which can be processed by up to q_j ($q_j \leq m$) machines simultaneously in any period. Assume that if \bar{q}_j ($\bar{q}_j \leq q_j$) machines process job j in a period then the processing time for job j is reduced by \bar{q}_j . Then job j can be divided into q_j jobs such that:

- (i) $p_j - q_j \left\lfloor \frac{p_j}{q_j} \right\rfloor$ of these jobs are assigned a pseudo-processing time of $\left\lfloor \frac{p_j}{q_j} \right\rfloor + 1$.
- (ii) The remaining jobs are assigned a processing time of $\left\lceil \frac{p_j}{q_j} \right\rceil$

where $\lfloor \cdot \rfloor$ denotes the largest integer less than or equal to \cdot .

The transformed problem becomes one of scheduling N ($N = \sum_{j=1}^n q_j$) jobs on m identical machines subject to the condition that none of the N jobs may be processed on more than one machine simultaneously.

For example, consider the problem of scheduling $n = 2$ jobs on $m = 2$ machines such that:

$$\begin{aligned} p_a &= 5 & q_a &= 2 \\ p_b &= 3 & q_b &= 2 \end{aligned}$$

Since $q_a = 2$ we have that job a is divided into two jobs such that $p_{a1} = q_a \lfloor \frac{p_a}{q_a} \rfloor$ or one job p_{a1} is assigned a processing time of $\lfloor \frac{p_a}{q_a} \rfloor + 1$ or 3. The other job p_{a2} is assigned a processing time of two.

Since $q_b = 2$ we have that job b is similarly divided into two jobs with processing times $p_{b1} = 2$ and $p_{b2} = 1$.

The transformed problem becomes one of scheduling $N = 4$ jobs with processing times $p_{a1} = 3$, $p_{a2} = 2$ and $p_{b1} = 2$, $p_{b2} = 1$ on two identical machines so that no job is processed by more than one machine in each period.

Correctness of the transformation follows from the observation that a solution to the transformed problem is feasible to the original problem. No more than q_j machines work on the original job j simultaneously (since it is divided into q_j jobs none of which can be processed simultaneously on more than one machine in any period). Moreover, since

all jobs in the transformed problems are completed and $\sum_{i=1}^{q_j} p_{ji} = p_j$, then job j in the original problem is completely processed. The completion time of job j is given by the last of the completion times of the q_j transformed jobs.

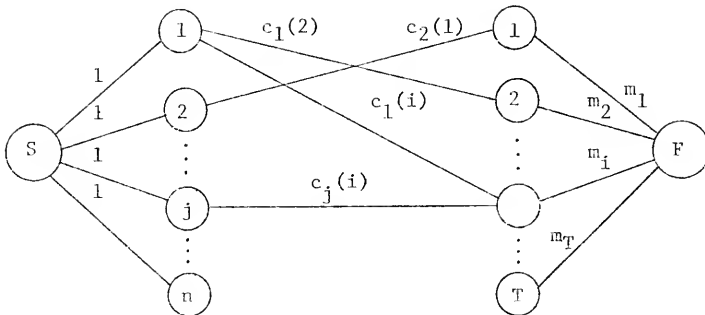
It then follows from this transformation that the results in Sections 3.2 and 3.3 also apply to the situation where simultaneous processing is allowed by more than one machine.

3.5 Two Min-Sum Problems

3.5.1 Limited Machine Availabilities

In Section 2.8 it was noted that very few results have been obtained for problems with limitations on job availabilities. One special case for which results are attainable is the following:

Consider the situation where we have unit processing times and the number of machines available varies from period to period. In this case the problem of minimizing $\sum_j c_j(t)$ can be formulated as follows:



That is, all arcs of the form (S, j) have capacity $p_j = 1$, all arcs of the form (i, F) have capacity m_i (the number of machines available in

period i) and an arc with cost $c_j(i)$ is drawn from job j to period i if job j can be processed in period i (i.e., if $r_j \leq i$). The solution to the above assignment problem provides a schedule which minimizes $\sum_j c_j(t)$. Extension of this notion to arbitrary processing times is stymied by the inability to construct appropriate costs for the jobs.

3.5.2 Jobs with Common Due Dates

In Section 2.10.1.B, Moore's algorithm for minimizing the number of tardy jobs was presented. While Moore's rule does not extend to general parallel processor problems, when job splitting is allowed the special case when all jobs have identical due dates (i.e., $d_j = d$ for all j) can be solved using the following algorithm:

Let d = due date for all jobs

$$(i) \text{ Compute } F_{\max} = \left\{ \max \left\{ \frac{\sum_j p_j}{m} ; \max p_j \right\} \right\}$$

where m denotes the number of identical processors and J the set of all jobs.

(ii) (a) If $F_{\max} \leq d$, then use either the "longest remaining processing time" rule (Rule 3.2) or McNaughton's algorithm [50] to construct a schedule such that no jobs are tardy.

(b) If $F_{\max} > d$, then discard job k where k is such that:

$$p_k = \max \{p_j\} \text{ for all } j \in J.$$

Remove job k from the set J and repeat (i) and (ii) until $F_{\max} \leq d$.

- (iii) Once a non-tardy subschedule has been found, all discarded jobs are added on in any order such that no job is worked on by more than one machine in each period, thereby completing the optimal schedule.

The application of the above rule for no job splitting is of course contingent upon a procedure for determining a schedule of value $F_{\max} \leq d$ which has yet to be found.

3.6 Minimization of Maximum Deferral Cost

Given any parallel processor scheduling problem for which a feasible solution exists, repeated application of the following property can be used to generate a schedule that minimizes $\max_j c_j(t)$.

Property 3.7. Let $X(T)$ be any feasible solution with value $K = \max_j c_j(t)$. A necessary condition for generating a strictly better schedule $X'(T)$ is that $c_j(t) < K$ for all $j = 1, \dots, n$ in $X'(T)$.

The interpretation of this deceptively simple condition in a scheduling context leads to the subsequent iterative procedure for generating an optimal schedule:

- (i) Generate any feasible schedule with value $K = \max_j c_j(t)$.
- (ii) Determine the latest period each job can be completed so that $\max_j c_j(t) < K$.
- (iii) Impose due dates on the jobs equal to this latest period.
- (iv) Determine a feasible schedule for all jobs subject to their newly assigned due dates.

- (v) If a feasible schedule exists, update K and repeat (ii) through (v). If no feasible schedule exists, then stop with $\min \max_j c_j(t) = K$.

It should be noted that the procedure, while completely independent of how a feasible schedule is generated, does require the existence of a method for arriving at a feasible schedule. For example, if $r_j = 0$ for all j and job splitting is allowed the results of Section 3.2 can be used at each iteration to determine the existence of a schedule with $L_{\max} = 0$ for each new set of due dates. If $r_j \neq 0$ for some j , then Bratley et al.'s network approach (Section 2.9.2.B) can be used for the same purpose. The lack of efficient methods for generating feasible schedules when job splitting is not allowed places limitations on the use of the above procedure for problems of this gender.

3.7 Conclusions

This chapter has been concerned with the development of optimal rules for problems that originate from the theory of parallel processor scheduling. A fundamental parallel processor problem was isolated and formulated. The general formulation presented for this basic problem of scheduling identical processors was motivated by the feeling that optimal rules could be developed for a class of problems adhering to this structure. A rule was then developed (Section 3.2) for minimizing maximum lateness. When job arrivals were allowed to be non-simultaneous, development of an optimal rule was limited to the criteria of maximum flow time (Section 3.3). However, extensions to more general cases were hindered by the lack of a dominant priority measure (Section 3.3.2)

as well as complexities introduced when various assumptions were relaxed (Appendix 2).

The relative lack of results for min-max problems in the literature (Chapter 2) prompted one to concentrate on this criterion. Min-sum problems were also considered but with the exception of two special cases (Section 3.5) little success was attained. Perhaps the most intractable of the problems studied were situations when job splitting is not allowed for which no results were attained for either min-max or min-sum criteria.

While utilization of the rules in this chapter is constrained by the rigidity of the assumptions invoked, it may very well be the case that these assumptions mark the perimeter beyond which development of optimal rules for parallel processor problems is highly improbable. Nonetheless attempts to develop rules for both the successful situations presented in this chapter as well as the unsuccessful ventures contribute to a fundamental understanding of the parallel processor problem. It is this insight which assists greatly in Chapter 4, where rules are developed for problems motivated by their potential application in scheduling on railways.

CHAPTER 4

FIXED ROUTE PARALLEL PROCESSOR SCHEDULING PROBLEMS

4.1 Introduction

While scheduling problems have received prominent attention in the airline industry, comparatively little analytical work has been done concerning such problems in the railroad industry. This seems particularly strange when one considers the observation by Mehl [52, p. 2] that "Almost everywhere rail transportation systems are infested with problems. Transit systems all over the country lose money. Even the more profitable railroads could have earned more by investing in a passbook savings account." Mehl's remark takes on added significance when he notes that such problems arise equally in freight service as well as in the more publicized area of passenger service.

Most of the work related to scheduling problems in the railroad industry has concerned itself with simulating various independent activities within the structure of a particular railway system. A compendium of simulation models currently in use is summarized in the Association of American Railroads' Application Digest [1] and in the Railroad Research Developmental Issue [60]. Some recent analytical approaches that, according to Mehl [52], incorporate more practical considerations are a Network Design Model by Billheimer [11] and a Suburban Railway Fleetsize Model by Salzborn [67]. In general, however, the amount of analytical work done with respect to schedule generation is scarce.

The primary interest of this chapter centers around the development of solution procedures for problems involving both a scheduling and routing aspect. The specific problems dealt with concern the movement of freight on railway networks although the notions developed are applicable in other contexts as well.

The emphasis will be on the simplest scheduling situation that arises when the route (network) consists of a single track connecting two major switchyards. This is akin to the operation of a "local line" of freight service (see Appendix 3 for a brief description of freight service operation on a railway system). The behavior of a local line follows two basic modes of operation:

(i) Based on demand or traffic available.

In this situation trains are formed and sent off as the need may arise. Interest here is focused on determining the size or number of trains required to carry all traffic when time is ignored.

(ii) Based on train movement according to a fixed schedule along the route. Although we will deal with mode (i) for certain route configurations the single track problems to be considered herein will concentrate on mode (ii) within which two major questions arise:

(a) What the train schedule is to be, and

(b) Given a train schedule, what should be transported on which train and what should be dropped off at each station.

In order to key on the most basic problem, we assume that the arrival of the trains at the stations along the line is predetermined (i.e., (b) the train(s) schedule is fixed a priori). The train(s) will either slow down or speed up accordingly to meet the schedule.

A basic assumption that will be maintained throughout is that: loading or unloading will not affect the schedule of the trains. The feasibility of such an assumption is enhanced by the operation of freight trains that usually run at intervals of every two or three days. In this context, loading and unloading times play roles of less significance.

Section 4.2 will deal with the simplest problem, namely the scheduling of one train on one track. The notions developed here are extended in Sections 4.3 through 4.5 to generate a series of basic feasibility results and optimal rules for the operation of M trains on one track subject to various restrictions on job availabilities and optimality criteria. Sections 4.6 and 4.7 allow for systems extending beyond a local line and networks with more than a single route, respectively, in which cases additional feasibility results are expounded.

Finally, we relate the problems of this chapter to the more classical parallel processor scheduling problems of Chapters 2 and 3 by noting that in the fixed route case, the trains play the role of processors and the freight that of the jobs. A basic difference which is exploited is that in the classical problems of the previous chapters jobs "queue up and wait" if not processed, whereas in the fixed route problems the machines move in time causing any job which is not processed to wait for a subsequent machine. This last characteristic allows one to classify fixed route problems as pertaining to problems with limited machine availabilities, i.e., Path 5 in the classification scheme of Chapter 2.

4.2 The One Train - One Track Problem
 (Alias the Waldo Train Scheduling Problem)

There exists a train station at Waldo, Florida, that is unique in a variety of aspects, many of which are not really relevant to the following discussion. Two relevant aspects are, however,

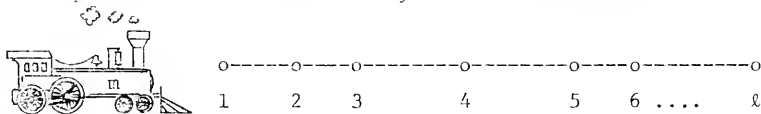
- (i) That the station is serviced by only one track, and
- (ii) That only one train per day passes through Waldo in a given direction. (While it is true that the same train passes through Waldo on its return trip, it suffices to view this occurrence as the same problem in the opposite direction.)

Based on this orientation we are interested in solving a scheduling problem that adheres to the following general description.

We have a single train with a fixed capacity (in boxcars), traveling a single fixed route subject to a fixed timetable. It encounters a series of local (Waldo-like) stations where jobs are waiting to be picked up and delivered to their destinations. All origin and destination stations for all jobs lie along the fixed route. Furthermore, we assume that all job units are commensurable with those of the train capacity (i.e., each job requires one boxcar).

Given that the train makes only one pass through the route, it is desired to determine which jobs are to be picked up or dropped off at the various stations so as to maximize the number of jobs that are taken to their destinations.

The problem can be schematically described as follows:



where the stations along the route are numbered 1, 2, ..., l . We define:

m = the capacity of the train (boxcars)

o_j = the origin station of job j , $j = 1, \dots, n$

f_j = the final or destination station of job j , $j = 1, \dots, n$

$p_j(i, k)$ = the time it takes to transport job j from station i to station k ($i \leq k$) (i.e., the time it takes the train to travel from station i to station k).

Furthermore, letting s denote any station we define:

$O(s) = \{j | o_j = s\}$ (i.e., $O(s)$ is the set of jobs whose origin is station s)

$F(s) = \{j | f_j = s\}$ (i.e., $F(s)$ is the set of jobs whose destination is station s)

$\bar{F}(s) = \{j | f_j \neq s\}$

$T(s) = \{j | \text{job } j \text{ is on the train when it arrives at station } s\}$

$A(s) = \{j | \text{job } j \text{ is available to be transported from } s \text{ to } s+1\}$ where

$A(s) = \{(T(s) \cap \bar{F}(s)) \cup O(s)\}$

for all stations $s = 1, \dots, \ell$ and all jobs $j = 1, \dots, n$, and

$\sum_{s=1, \dots, \ell} | \cup O(s) | = n$ (where $|\cdot|$ denotes the number of elements in the set).

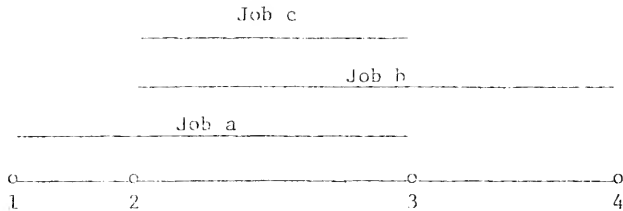
Finally, let

$X(\ell) =$ any schedule or assignment of the jobs to the train for all station intervals $s, s+1$ ($s = 1, \dots, \ell$).

We first note that the effect of the jobs upon the system consists in creating an overlap (or demand) for service between any two stations along the route. Let $\delta(s, s+1)$ denote the number of jobs (loads) which overlap between stations s and $s+1$ where formally

$$\delta(s, s+1) = \left| \bigcup_{v=1, \dots, s} O(v) \cap \left(\bigcup_{v=s+1, \dots, \ell} F(v) \right) \right|$$

For example, given the following route and jobs, where each job can be represented by an interval starting at o_j , ending at f_j , of length $p_j(o_j, t_j)$, we have



$$\delta(1, 2) = 1; \quad \delta(2, 3) = 3; \quad \text{and} \quad \delta(3, 4) = 1.$$

Property 4.1. The minimum train capacity, m_{\min} , required to transport all jobs to their destinations is

$$m_{\min} = \max_{s=1, \dots, \ell} \delta(s, s+1)$$

It then follows that if $m \geq \max_{s=1, \dots, \ell} \delta(s, s+1)$, no scheduling problem

exists since all jobs can be transported to their destinations. Therefore, we focus attention on the only case for which a scheduling problem arises.

Theorem 4.1. Let $m < \max_{s=1, \dots, \ell} \delta(s, s+1)$. Then any schedule that adheres

to the following rule maximizes the number of jobs taken to their destinations.

When the train arrives at any station s , $s = 1, \dots, \ell$,

- (i) If $|\Lambda(s)| \leq m$, then transport all jobs in $\Lambda(s)$ to station $s+1$.
- (ii) If $|\Lambda(s)| > m$, then transport any m jobs in $\Lambda(s)$ to station $s+1$ such that if job q is transported and job r is not, then $p_q(s, f_q) \leq p_r(s, f_r)$

(i.e., select those m jobs which are presently closest to their destinations).

Proof. Let $X^*(\ell)$ denote a schedule generated by the rule. (Note that $X^*(\ell)$ is not necessarily unique since ties are broken arbitrarily.)

Assume that $X^*(\ell)$ is not optimal. Let $X'(\ell) \neq X^*(\ell)$ be a schedule that does not satisfy the property of the rule. That is, in $X'(\ell)$ there exists a station s^* such that job q is transported from s^* to s^*+1 and job r is not and

$$p_q(s^*, f_q) > p_r(s^*, f_r).$$

Assume that $|J'| > |J^*|$, where $|J'|$ and $|J^*|$ denote the number of jobs transported to their destinations in $X'(\ell)$ and $X^*(\ell)$, respectively.

If q arrives at its destination in $X'(\ell)$, then q occupies a box-car from s^* for $p_q(s^*, f_q)$ units of time. Since $p_q(s^*, f_q) > p_r(s^*, f_r)$, interchanging r for q at s^* allows r to arrive at its destination without affecting any other jobs.

If q does not arrive at its destination in $X'(\ell)$, then there must exist a station s' , where $s' < f_q$, where q was taken off the train. If $s' < f_r$, interchanging r for q at s^* does not decrease the number of jobs that arrive at their destinations. If $s' \geq f_r$, then after the interchange r will be able to arrive at its destination thus increasing the number of jobs taken to their destinations by one.

In either case, interchanging r for q at s^* results in the number of jobs taken to their destinations after the interchange being greater than or equal to the number before.

If we start with $s^* = 1$, after at most $m\ell$ interchanges of the above form, any schedule $X'(\ell)$ can be converted to a schedule $X^*(\ell)$ such that $|J^*|_s \geq |J'|_s$ for all $s = 1, \dots, \ell$. Therefore, $|J^*| \geq |J'|$, contradicting the initial assumption.

It now remains to show that all different schedules generated by the rule have the same value. Let $X_1^*(\ell)$ and $X_2^*(\ell)$ be two schedules that have the property of Theorem 4.1 but such that $X_1^*(\ell) \neq X_2^*(\ell)$.

We wish to show that $|J_1^*| = |J_2^*|$.

Since $X_1^*(\ell) \neq X_2^*(\ell)$, then there must exist a station s^* such that:

- (a) r is transported from s^* to s^*+1 in $X_1^*(\ell)$ while q is not,
- (b) q is transported from s^* to s^*+1 in $X_2^*(\ell)$ while r is not, and
- (c) $p_q(s^*, f_q) = p_r(s^*, f_r)$.

Then clearly interchanging q and r at s^* does not affect the number of jobs taken to their destinations. Again starting with $s^* = 1$ after at most m^ℓ interchanges, the schedule $X_2^*(\ell)$ can be made identical to $X_1^*(\ell)$ and $|J_1^*| = |J_2^*|$. Repetition of this argument for any schedule $X_u^*(\ell) \neq X_1^*(\ell)$, allows us to conclude that for all schedules $X_u^*(\ell)$ with the property of Theorem 4.1, $|J_u^*| = |J_1^*|$. \square

4.3 The M Train - One Track Problem

We now extend the formulation of Section 4.1 to the case where M trains service the stations along the route. In this situation it is important to distinguish between instances where job splitting is or is not allowed. If job splitting is permissible, then any job can be carried from its origin to its destination on any combination of trains. No job splitting requires that once a job is loaded on a train it must be carried its entire distance on the same train. (Note that in the One Train - One Track case it is irrelevant whether job splitting is or is not allowed since all jobs that are not carried their entire distance need never be transported at all.)

4.3.1 Problem Formulation

The M train problem can be schematically described as follows:



where again the stations along the route are numbered 1, 2, ..., l (the above schematic illustrating the case where $M = 3$).

We assume that all definitions and assumptions of Section 4.1 still hold, although we redefine

m_i = the capacity in boxcars of train i , $i = 1, 2, \dots, M$

$T_i(s) = \{j \mid \text{job } j \text{ is on train } i \text{ when it arrives at station } s\}$

$A_i(s) = \{j \mid \text{job } j \text{ is available to be transported on train } i \text{ from } s \text{ to } s+1\}$ where

$A_i(s) = \{(T_i(s) \cap \bar{F}(s)) \cup O(s)\}$.

$X(l)$ = any feasible schedule or assignment of the jobs to the trains for all station intervals $s, s+1$ ($s = 1, \dots, l$), where by feasible we mean that given any station s , any job j which is transported to $s+1$ on train Y , either originates at s or is transported to s on one of the trains $1, \dots, Y-1$.

We also define

t_i = the interarrival time between trains i and $i+1$ at all stations along the route ($i = 1, 2, \dots, M$)

a_j = the time at which train 1 arrives at o_j (i.e., the earliest departure time of job j).

Theorem 4.2. Let $\delta(s, s+1) = \left| \left(\bigcup_{v=1, \dots, s} O(v) \right) \cap \left(\bigcup_{v=s+1, \dots, l} F(v) \right) \right|$ again denote the overlap or demand for boxcars between stations s and $s+1$.

Furthermore, let $\sum_{i=1}^Y m_i$ = the total capacity of the first Y trains.

It then follows that a necessary and sufficient condition for all jobs to arrive at their destinations is that at least M^* trains travel the route where

$$\sum_{i=1}^{M^*-1} m_i < \max_{s=1, \dots, \ell} \delta(s, s+1) \leq \sum_{i=1}^{M^*} m_i$$

Proof. The condition is clearly necessary. Sufficiency is established

by assuming that $\sum_{i=1}^{M^*} m_i \geq \max_{s=1, \dots, \ell} \delta(s, s+1)$. A schedule can now be con-

structed in which all jobs arrive at their destinations by simply starting with the first station and arbitrarily assigning the jobs to any unfilled train. Then repeat the process at each station encountered

along the route. If a station s' is reached where all loads cannot be

assigned, it follows that $\sum_{i=1}^{M^*} m_i < \delta(s', s'+1)$ which is a contradiction. \square

Corollary 4.1. Assume the necessary and sufficient condition of

Theorem 4.2 is satisfied. Then, any schedule which has the property

that no job which can be transported is left uncarried (i.e., train i

travels empty between s and $s+1$ only if $|A_i(s)| = \phi$ for all $i = 1, \dots, M$)

will take all jobs to their destinations on M^* trains irrespective of whether job splitting is or is not allowed.

Proof. Follows directly from the proof of Theorem 4.2. \square

We also note that without loss of generality we can assume that $m_i = 1$ for all trains $i = 1, \dots, M$ (i.e., each original train with capacity m_i can be conceptualized as m_i "trains" with unit capacity and

interarrival time of ϵ between any two boxcars of the original train i).
Therefore, Theorem 4.2 translates to

$$M^* = \max_{s=1, \dots, \ell} \delta(s, s+1)$$

4.3.2 Maximization of the Number of Jobs Taken to Their Destinations by the First K ($K \leq M^*$) Trains

In this section we will be concerned with generating schedules that not only transport all jobs to their destinations on M^* trains, but at the same time maximize the number of jobs transported to their destinations by the first K trains for all $K = 1, 2, \dots, M^*$. We assume that the necessary and sufficient condition of Theorem 4.2 is satisfied and will first consider the situation when any job can be carried from its origin to its destination on any combination of trains.

Theorem 4.3. Assume that job splitting is allowed. Then any schedule having the property that given the arrival of train Y ($1 \leq Y \leq K$) at any station s ($s = 1, \dots, \ell$),

- (i) If $|A_Y(s)| \leq 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y .
- (ii) If $|A_Y(s)| > 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y such that if job q is transported and job r is not, then

$$p_q(s, f_q) \leq p_r(s, f_r)$$

maximizes the number of jobs taken to their destinations by the first K trains for all $K = 1, 2, \dots, M^*$.

Proof. Let $X_K^1(\ell)$ be any schedule that maximizes the number of jobs taken to their destinations by the first K trains. Disregard all jobs not taken to their destinations on the first K trains in $X_K^1(\ell)$. Let $D_K^1(\ell)$ be the set of jobs transported to their destinations by the first K trains in $X_K^1(\ell)$ and let $\delta_D(s, s+1)$ denote the overlap between s and $s+1$ of the jobs in $D_K^1(\ell)$ for all $s = 1, \dots, \ell$.

From Theorem 4.2 we have that $\max_s \delta_D(s, s+1) = K$. It then follows from Corollary 4.1 that any schedule which has the property that train Y travels empty between s and $s+1$ only if $|A_Y(s)| = \phi$ for all $Y = 1, \dots, K$, will transport all the jobs in $D_K^1(\ell)$ to their destinations on the first K trains. Note that any schedule generated by conditions (i) and (ii) of Theorem 4.3 satisfies the property of Corollary 4.1. Therefore, use conditions (i) and (ii) to schedule the jobs in $D_K^1(\ell)$ on the first K trains and denote this schedule by $X_K^0(\ell)$.

Let $X_K^2(\ell)$ be any schedule generated by conditions (i) and (ii) of Theorem 4.3 for the whole set of jobs and the first K trains. Disregard all jobs not taken to their destinations on the first K trains in $X_K^2(\ell)$. Let $D_K^2(\ell)$ be the set of jobs transported to their destinations on the first K trains in $X_K^2(\ell)$. Starting with $s = 1$ and train one compare $X_K^2(\ell)$ and $X_K^0(\ell)$ as follows:

- (a) If train Y travels empty from s to $s+1$ in both schedules continue to $s+1$.
- (b) If train Y travels empty from s to $s+1$ in $X_K^0(\ell)$ but does not travel empty in $X_K^2(\ell)$, then transport the job carried from s to $s+1$ in $X_K^2(\ell)$ from s to $s+1$ in $X_K^0(\ell)$. Continue to $s+1$.

(c) Assume that train Y transports a job from s to $s+1$ in both schedules. Let q and r be the jobs transported in $X_K^0(\ell)$ and $X_K^*(\ell)$ respectively. If $q = r$, continue to $s+1$. Assume $q \neq r$.

(1) If $p_r(s, f_r) < p_q(s, f_r)$, it must be true that r is not transported at all in $X_K^0(\ell)$ since the jobs transported in $X_K^0(\ell)$ satisfy conditions (i) and (ii). Discard q and transport r from s to $s+1$ on Y in $X_K^0(\ell)$. Since $p_r(s, f_r) < p_q(s, f_r)$, r can be transported from $s+1$ to f_r in $X_K^0(\ell)$ using the same trains that q traveled on previously. After this interchange $X_K^0(\ell)$ remains optimal since the number of jobs transported in $X_K^0(\ell)$ remains the same.

(2) If $p_r(s, f_r) = p_q(s, f_q)$, then transport r from s to $s+1$ on Y in $X_K^0(\ell)$, and take r to its destination by interchanging q and r everywhere they appear in $X_K^0(\ell)$. If r was not processed in $X_K^0(\ell)$, then discard q . We again have that after the interchange the number of jobs transported in $X_K^0(\ell)$ remains identical.

Either (a), (b) or (c) is applied for $s = 2, \dots, \ell$. The procedure is repeated for trains $2, 3, \dots, K$. At each step we have that both schedules are identical up to train Y and station s and that after train K is scheduled at station ℓ , $X_K^0(\ell) \equiv X_K^*(\ell)$. Since the number of jobs transported in $X_K^0(\ell)$ remains identical after either (a), (b) or (c), $X_K^0(\ell) \equiv X_K^*(\ell)$ implies that $|D_K^0(\ell)| = |D_K^1(\ell)|$ and $X_K^0(\ell)$ is optimal.

Finally, it is obvious that if $X_{1K}^*(\ell)$ and $X_{2K}^*(\ell)$ are any two

schedules generated by conditions (i) and (ii) for the whole set of jobs and the first K trains but such that $X_{1K}^*(\ell) \neq X_{2K}^*(\ell)$, that $|D_{1K}^*(\ell)| = |D_{2K}^*(\ell)|$ where $D_{1K}^*(\ell)$ and $D_{2K}^*(\ell)$ denote the set of jobs transported to their destinations on trains $1, \dots, K$ in $X_{1K}^*(\ell)$ and $X_{2K}^*(\ell)$, respectively. \square

Theorem 4.3 allows us to present the following algorithm for generating a schedule that maximizes the number of jobs taken to their destinations by the first K trains (for all $K = 1, \dots, M^*$) when job splitting is now allowed.

(i) Use Theorem 4.3 to determine the maximum number of jobs taken to their destinations by the first K trains when job splitting is allowed. Let $X_K^*(\ell)$ be the schedule generated by Theorem 4.3 and $D_K^*(\ell)$ be the set of jobs taken to their destinations in $X_K^*(\ell)$.

(ii) Disregard all jobs not taken to their destinations in $X_K^*(\ell)$. Schedule all jobs in $D_K^*(\ell)$ on the first K trains by starting with the first station and arbitrarily assigning the jobs to any unfilled train. Repeat the procedure for stations $2, \dots, \ell$ after which a schedule $X_K^o(\ell)$ without job splitting has been obtained which maximizes the number of jobs transported to their destinations by trains $1, \dots, K$.

Proof. Let $\delta_{D_K^*(\ell)}(s, s+1)$ denote the overlap between s and $s+1$ of the jobs in $D_K^*(\ell)$ in $X_K^*(\ell)$ for all $s = 1, \dots, \ell$. Since $\max_s \delta_{D_K^*(\ell)}(s, s+1) \leq K$, it follows from Theorem 4.2 and Corollary 4.1 that in step (ii) all jobs in $D_K^*(\ell)$ are transported to their destinations by $X_K^o(\ell)$ without job splitting. Since $|D_K^*(\ell)|$ is a lower bound on the no job splitting case, we have that $X_K^o(\ell)$ is optimal. \square

4.3.3 Minimization of the Sum of Completion Times

We assume in this instance that job splitting is allowed and that the trains travel the route according to a fixed schedule with train interarrival times t_i . We are interested in generating schedules that minimize the mean or sum of job completion times or equivalently minimize the mean lateness of all jobs.

We first observe that the completion time C_j of any job j , is solely a function of the train that it arrives on at its destination f_j . That is, if job j arrives at f_j on train one, then

$$C_j(1) = a_j + p(o_j, f_j)$$

If it arrives at f_j on train two, then

$$C_j(2) = a_j + p(o_j, f_j) + t_1;$$

and in general, if it arrives at f_j on train Y , then

$$C_j(Y) = a_j + p(o_j, f_j) + \sum_{i=1}^{Y-1} t_i$$

Let $W_j(Y)$ denote the waiting time of job j given that it arrives at its destination on train Y where

$$W_j(Y) = \sum_{i=1}^{Y-1} t_i.$$

Property 4.2. Let $X'(\ell)$ be any schedule that delivers all jobs to their destinations on M^* trains. Let $\sum_{j=1}^n C_j(X'(\ell))$ and $\sum_{j=1}^n W_j(X'(\ell))$

denote the sum of completion times and sum of waiting times, respectively, for all jobs in $X'(\ell)$. Then

$\sum_{j=1}^n C_j(X'(\ell)) \leq \sum_{j=1}^n C_j(X(\ell))$ if and only if

$\sum_{j=1}^n W_j(X'(\ell)) \leq \sum_{j=1}^n W_j(X(\ell))$ for all other schedules $X(\ell)$ that transport all jobs to their destinations on M^* trains.

Proof. $C_j(Y) = a_j + p(o_j, f_j) + \sum_{i=1}^{Y-1} t_i = a_j + p(\theta_j, f_j) + W_j(Y)$

$$\therefore \sum_{j=1}^n C_j(X'(\ell)) = \sum_{j=1}^n a_j + \sum_{j=1}^n p(o_j, f_j) + \sum_{j=1}^n W_j(X'(\ell))$$

where $\sum_{j=1}^n a_j + \sum_{j=1}^n p(o_j, f_k)$ are constants independent of the schedule. \square

Our objective reduces to one of minimizing $\sum_{j=1}^n W_j$. Since W_j is strictly a function of the train that j arrives at its destination on, it is intuitively appealing that what we would like to do is deliver as many jobs as possible to their destinations on train one, then on train two, etc., in order to minimize $\sum_{j=1}^n W_j$. In Theorem 4.3 we found that this is optimally accomplished by giving priority to that job which is closest to its destination whenever a decision arises. We now restate Theorem 4.3 as a scheduling rule and show that it also provides us with a procedure for minimizing the sum of waiting times.

Rule 4.1. Assume that job splitting is allowed. Then any schedule having the property that given the arrival of train Y ($Y = 1, \dots, M^*$) at any station s ($s = 1, \dots, \ell$)

- (i) If $|A_Y(s)| \leq 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y .
- (ii) If $|A_Y(s)| > 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y such that if job q is transported and job r is not, then

$$p_q(s, f_q) \leq p_r(s, f_r)$$

minimizes $\sum_{j=1}^n W_j$.

Proof. The proof will be by contradiction. Let $X^*(\ell)$ be any schedule generated by Rule 4.1 where $N_1^*, N_2^*, \dots, N_{M^*}^*$ denote the number of jobs taken to their destinations by trains 1, 2, ..., M^* in $X^*(\ell)$. Let $X^o(\ell)$ be any optimal schedule where $N_1^o, N_2^o, \dots, N_{M^*}^o$ denote the number of jobs taken to their destinations by trains 1, 2, ..., M^* in $X^o(\ell)$. Let $\sum_j W_j^*$ and $\sum_j W_j^o$ denote the sum of waiting times in $X^*(\ell)$ and $X^o(\ell)$, respectively.

Assume that $X^*(\ell)$ is not optimal. We then have that

$$\sum_j W_j^o < \sum_j W_j^*$$

or equivalently that

$$\begin{aligned} N_2^o(t_1) + N_3^o(t_1+t_2) + \dots + N_{M^*}^o(t_1+t_2+\dots+t_{M^*-1}) \\ < N_2^*(t_1) + N_3^*(t_1+t_2) + \dots + N_{M^*}^o(t_1+t_2+\dots+t_{M^*-1}) \end{aligned}$$

which can also be expressed as

$$\begin{aligned} t_1(N_2^o+N_3^o+\dots+N_{M^*}^o) + t_2(N_3^o+N_4^o+\dots+N_{M^*}^o) + \dots + t_{M^*-1}(N_{M^*}^o) \\ < t_1(N_2^*+N_3^*+\dots+N_{M^*}^*) + t_2(N_3^*+N_4^*+\dots+N_{M^*}^*) + \dots + t_{M^*-1}(N_{M^*}^*) \end{aligned}$$

By Theorem 4.3 we have that

$$(i) \quad N_1^* \geq N_1^o \quad \text{which implies that} \quad (N_2^*+N_3^*+\dots+N_{M^*}^*) \leq (N_2^o+N_3^o+\dots+N_{M^*}^o)$$

$$(ii) \quad (N_1^*+N_2^*) \geq (N_1^o+N_2^o) \quad \text{which implies that} \quad (N_3^*+N_4^*+\dots+N_{M^*}^*) \leq (N_3^o+N_4^o+\dots+N_{M^*}^o)$$

$$\begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array}$$

$$(M^*-2)(N_1^*+N_2^*+\dots+N_{M^*-2}^*) \geq (N_1^o+N_2^o+\dots+N_{M^*-2}^o) \quad \text{which implies that}$$

$$(N_{M^*-1}^*+N_{M^*}^*) \leq (N_{M^*-1}^o+N_{M^*}^o)$$

in which case $\sum_j W_j^o < \sum_j W_j^*$ only if

$$N_{M^*}^o < N_{M^*}^*$$

However, $\sum_{i=1}^{M^*} N_i^* = \sum_{i=1}^{M^*} N_i^o$ and (i) through (M^*-2) imply that $N_{M^*}^* \leq N_{M^*}^o$

and hence a contradiction ensues. \square

where (E) denotes the first or earliest train that a particular job can go on. Define:

$\delta_E(s, s+1)$ = the number of jobs which overlap between stations
s and s+1 from among those jobs which must go on
train E or later,

where in the above diagram we have

$$\delta_1(1,2) = 0; \quad \delta_1(2,3) = 2; \quad \delta_1(3,4) = 2; \quad \delta_1(4,5) = 2; \quad \delta_1(5,6) = 2$$

and $\delta_2(s, s+1) = 1$ for all $s = 1, \dots, 6$.

Theorem 4.4. Assume that job splitting is allowed. Then a necessary and sufficient condition for all jobs to be transported to their destinations on M trains (where $m_i = 1$ for all $i = 1, \dots, M$) is that

$$\begin{aligned} \text{(i)} \quad & \max_s \delta_M(s, s+1) = 1 \\ \text{(ii)} \quad & \max_s \delta_{M-1}(s, s+1) = 2 \\ & \vdots \\ \text{(M)} \quad & \max_s \delta_1(s, s+1) = M \end{aligned}$$

Proof of Theorem 4.4.

Theorem 4.4 is clearly necessary. Sufficiency will be shown by constructing a schedule that delivers all jobs to their destinations on M trains.

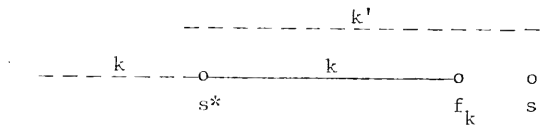
Lemma 4.1. Suppose that the first Y trains have been scheduled and that Theorem 4.4 is satisfied with respect to the remaining jobs or portions of jobs that were not transported to their destinations on trains 1, ..., Y. Considering the stations in the order 1, 2, ..., at any station s let q and r be any jobs such that both q and r are available for transporting

to $s+1$ on train $Y+1$ (i.e., $q, r \in \{A_{Y+1}(s)\}$). Transport q to $s+1$ on train $Y+1$ if

$$p_q(s, f_q) \geq p_r(s, f_r)$$

that is, give priority to that job which is currently farthest away from its destination. Then after train $Y+1$ completes the route all jobs or portions of jobs that remain to be transported on trains $Y+2, \dots, M^*$, satisfy the condition of Theorem 4.4.

Proof. First consider $Y = 1$. Let k denote any job or portion of job that remains to be transported on trains $2, \dots, M^*$ after train one completes the route using the above procedure for loading the jobs. Let s^* denote the last station k was carried to by train one where $s^* \geq o_k$. Since k was not taken to its destination on train one, then there exists a job k' such that $o_{k'} = s^*$ that is transported from s^* to s' ($s' \geq f_k$) on train one. This is illustrated in the following illustration where the broken line indicates that portion of the job transported on train one and the solid line that portion of the job remaining to be transported on after train one completes the route.



Assume that after train one completes the route, the maximum overlap between any two stations along the segment s^* to f_k is M . Since any job that overlaps with k between s^* and f_k also overlaps with k' , this would imply that before train one is scheduled, the maximum overlap between any two stations from s^* to f_k is $M+1$ which is a contradiction. Now label all jobs or portions of jobs with $(E) = 1$ that were not taken

to their destinations on train one with $(E) = 2$, and leave the same label as before on all other jobs. It then follows that the condition of Theorem 4.4 is satisfied for this new set of jobs, that is,

$$\begin{aligned} \text{(i)} \quad & \max_s \delta_M(s, s+1) = 1 \\ & \vdots \\ \text{(M*-1)} \quad & \max_s \delta_2(s, s+1) = M-1 \end{aligned}$$

The argument is now repeated for train two through M by simply renumbering the trains after each train is scheduled. \square

From Lemma 4.1 it follows that if each train is scheduled by using the rule, then after trains $1, \dots, M-1$ complete the route, train M can pick up all remaining jobs and a schedule that transports all jobs to their destinations on M trains has been constructed. \square

It now follows from Lemma 4.1 that a procedure for transporting all jobs to their destinations on M trains is given by the following rule.

Rule 4.2. Assume that job splitting is allowed and that the necessary and sufficient condition of Theorem 4.4 holds. Then any schedule having the property that given the arrival of train Y (for all $Y = 1, \dots, M^*$) at any station s ($s = 1, \dots, \ell$),

- (i) If $|A_Y(s)| \leq 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y .
- (ii) If $|A_Y(s)| > 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y such that if job q is transported and job r is not, then

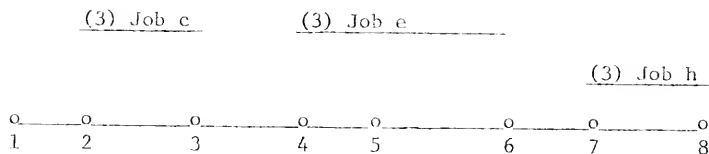
$$p_q(s, f_q) \geq p_r(s, f_r)$$

will transport all jobs to their destinations on M^* trains.

Train two transports Job c from $s = 1$ to $s = 2$. At $s = 2$,

$$p_a(2, f_a) = p_a(2, 4) > p_c(2, 3) = p_c(2, f_c)$$

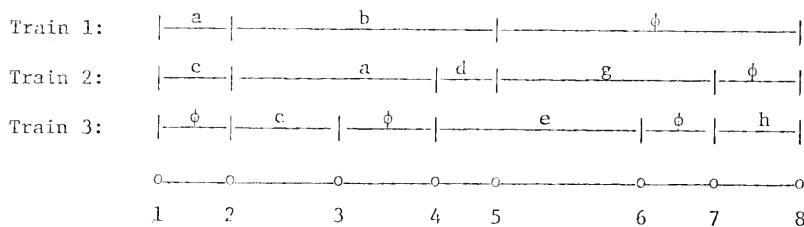
Therefore Job a is transported from $s = 2$ to $s = 4$ on train two. Train two completes its route by transporting Job d from $s = 4$ to $s = 5$ and Job g from $s = 5$ to $s = 7$. After train two reaches station 8 we have



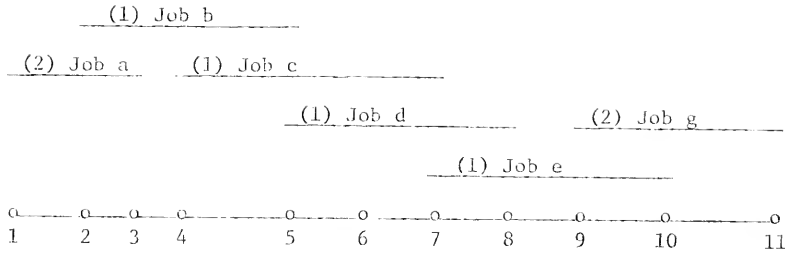
where the remaining portion of Job c has been relabeled and all other job labels remain the same. We now have

$$(i) \max_s \delta_3(s, s+1) = 1$$

in which case train three picks up Jobs c, e, and h. The constructed schedule is summarized as follows:



We mentioned previously that job splitting plays a crucial role in problems with restrictions on job availabilities. The following example will show that when job splitting is not allowed, Theorem 4.4 while still necessary is no longer sufficient to ensure that the M trains can in fact transport all jobs to their destinations.



We have that

$$(i) \max_s \delta_2(s, s+1) = 1$$

$$(ii) \max_s \delta_1(s, s+1) = 2$$

However, since job splitting is not allowed, jobs b and e must be completely carried by train one (since they overlap with Jobs a and g, respectively, which can only go on train two). This implies that Jobs c and d must both be transported on train two and hence no schedule exists in which all jobs can be taken to their destinations on train two subject to the no job splitting and job availability constraints.

4.4.2 Jobs Restricted by Due Dates

We now assume that although all jobs are at their origin when train one arrives, it is required that a subset of the jobs be delivered to their destinations on some train before train M. Such a situation may arise when due dates are imposed on the jobs and our objective is one of delivering all jobs to their destinations on or before their due dates. For example, letting d_j denote the due date of job j , we observe that the lateness L_j of any job j , is solely a function of the last train it is carried to its destination on. That is, if job j arrives at f_j on train one, then

$$L_j(1) = (a_j + p(o_j, f_j)) - d_j$$

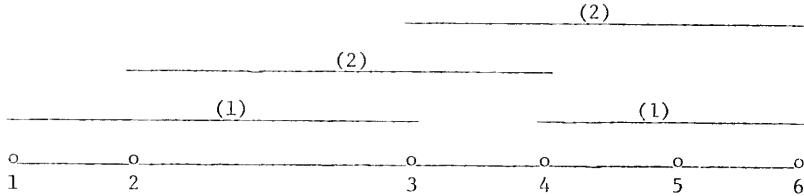
if it arrives at its destination on train two, then

$$L_j(2) = (a_j + p(o_j, f_j) + t_1) - d_j;$$

and in general, if it arrives at its destination f_j on train Y , then

$$L_j(Y) = (a_j + p(o_j, f_j) + \sum_{i=1}^{Y-1} t_i) - d_j.$$

The problem of finding a schedule such that all due dates are met is therefore equivalent to that of scheduling the jobs so that $L_j \leq 0$ for all j , where the last train each job can travel on is predetermined by its last nonnegative value of L_j . This situation, the opposite of the one encountered in 4.4.1 can be illustrated as follows:



where (L) now denotes the last train that a particular job can go on.

Define

$\delta_{(L)}(s, s+1)$ = the number of jobs which overlap between stations s and $s+1$ from among those jobs which must be delivered to their destinations on train L or earlier,

where in the previous illustration we have

$\delta_{(2)}(1,2) = 1$; $\delta_{(2)}(2,3) = 2$; $\delta_{(2)}(3,4) = 2$; $\delta_{(2)}(4,5) = 2$; $\delta_{(2)}(5,6) = 2$
and

$\delta_{(1)}(1,2) = 1$; $\delta_{(1)}(2,3) = 1$; $\delta_{(1)}(3,4) = 0$; $\delta_{(1)}(4,5) = 1$; $\delta_{(1)}(5,6) = 1$.

The analogy between this situation and the case when jobs are not simultaneously available can be used to assert:

Theorem 4.5. Assume that job splitting is allowed. Then a necessary and sufficient condition for all jobs to be transported to their destinations on M trains (where $m_i = 1$ for all $i = 1, \dots, M$) is that

$$\begin{aligned} \text{(i)} \quad & \max_s \delta_{(1)}(s, s+1) = 1 \\ \text{(ii)} \quad & \max_s \delta_{(2)}(s, s+1) = 2 \\ & \vdots \\ \text{(M)} \quad & \max_s \delta_{(M)}(s, s+1) = M \end{aligned}$$

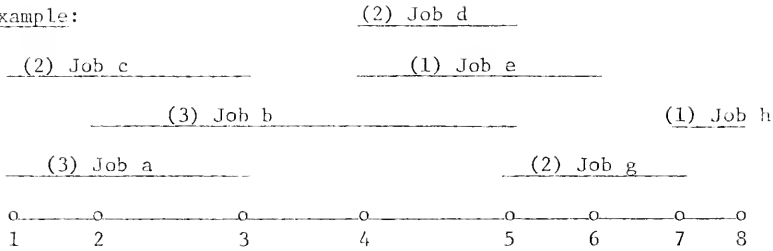
Proof. The proof is completely analogous to the proof of Theorem 4.4. A schedule that delivers all jobs to their destinations on or before their due dates on M trains can be constructed by inverting the procedure of Lemma 4.1 as follows:

Starting with train M (instead of train one) and station l (instead of station 1) consider any station s having jobs q and r available for transporting to $s-1$ on train M (i.e., q and r are available if given that they arrive at their destination on M , they are not late). Transport q to $s-1$ on train M if

$$p_q(s, o_q) \geq p_r(s, o_r)$$

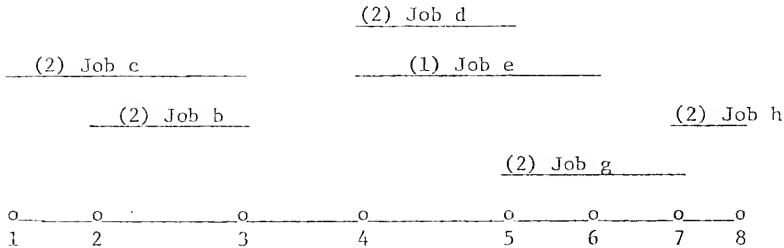
that is, give priority to that job currently farthest away from its origin. After train M completes its inverted route, all jobs or portions of jobs that remain to be transported on trains $M-1, \dots, 1$, satisfy conditions (i) through $(M-1)$ of Theorem 4.5. This inverted procedure is repeated for trains $M-1, M-2, \dots, 1$ after which all jobs will have been transported "from their destinations to their origins" on the M trains. \square

Example:



- where
- (i) $\max_s \delta_{(1)}(s, s+1) = 1$
 - (ii) $\max_s \delta_{(2)}(s, s+1) = 2$
 - (iii) $\max_s \delta_{(3)}(s, s+1) = 3$

Train three is scheduled starting at station $s = 8$. It travels empty from $s = 8$ to $s = 5$ at which time it picks up Job b. At $s = 3$, $p_a(3, o_a) = p_a(3, 1) > p_b(3, 2) = p_b(3, o_b)$. Therefore, Job a is transported from $s = 3$ to $s = 1$ on train three. After train three is scheduled we have



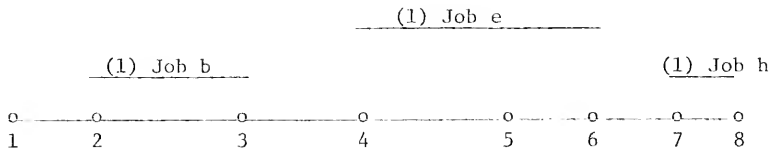
where the remaining portion of Job b has been relabeled and all other jobs retain their original label. We now have

- (i) $\max_s \delta_{(1)}(s, s+1) = 1$
- (ii) $\max_s \delta_{(2)}(s, s+1) = 2$

Train two travels empty from $s = 8$ to $s = 7$. At $s = 7$, Job g is picked up and taken to $s = 5$. At $s = 5$, Job d is transported to $s = 4$. After traveling empty from $s = 4$ to $s = 3$ we find that that

$$p_c(3, o_c) = p_c(3,1) > p_b(3,2) = p_b = (3, o_b)$$

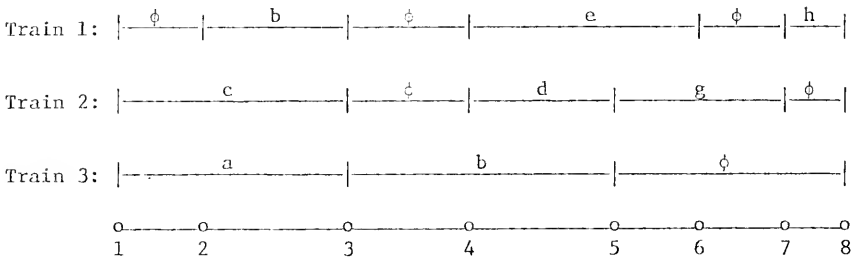
Therefore, Job c is transported from $s = 3$ to $s = 1$ and train two completes its route. We now have



where the remaining portion of Job b has been relabeled and all other job labels remain the same. Finally, we have

$$(i) \max_s \delta_{(1)}(s, s+1) = 1$$

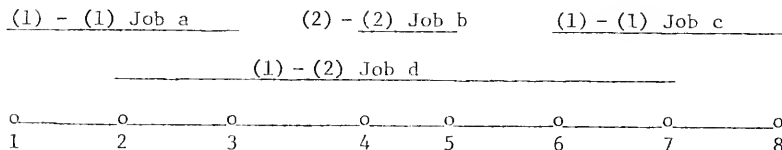
in which case train one picks up Jobs h , e and b . The constructed schedule is summarized as follows:



The awkwardness of developing a schedule in the previous inverted manner makes it desirable to develop procedures for accomplishing the same objective starting with train one, train two, etc. This is discussed in Section 4.5 where the analogous problem of minimizing the maximum lateness of all jobs is presented for both job splitting and no job splitting situations.

4.4.3 Jobs Not Simultaneously Available and Subject to Due Dates

When restrictions exist on both the first and last trains that a job may travel on, we find that even when job splitting is allowed, necessary and sufficient conditions for generating a feasible schedule are hard to come by. The situation can be illustrated as follows:



where (E) - (i) denote, respectively, the first and last trains a particular job can go on. When Theorem 4.4 and Theorem 4.5 are combined they result in a necessary condition for generating a schedule that satisfies the job availability restrictions with M trains. Lack of sufficiency is shown by the problem above where $M = 2$. We have that

$$\text{Theorem 4.4} \quad \left\{ \begin{array}{l} \text{(i)} \quad \max_s \delta_2(s, s+1) = 1 \\ \text{(ii)} \quad \max_s \delta_1(s, s+1) = 2 \end{array} \right.$$

and

$$\text{Theorem 4.5} \quad \left\{ \begin{array}{l} \text{(i)} \quad \max_s \delta_{(1)}(s, s+1) = 1 \\ \text{(ii)} \quad \max_s \delta_{(2)}(s, s+1) = 2 \end{array} \right.$$

However, Jobs a and c can only go on train one, and Job b can only go on train two. This implies that Job d must be transported from $s = 2$ to $s = 3$ on train two, $s = 4$ to $s = 5$ on train one and $s = 6$ to $s = 7$ on train two again. Hence no feasible schedule can be constructed on two trains regardless of which end of the route we start from even if job splitting is allowed.

4.4.4 Synopsis

The results of this section show that even the question of determining a feasible schedule to transport all jobs to their destinations becomes complicated if restrictions on job availabilities are imposed. In order to look at other optimality criteria we revert to the situation where no job availability restrictions are imposed.

4.5 Minimization of Maximum Lateness

We assume that there are no restrictions on job availabilities, we are interested in this instance in generating schedules that minimize L_{\max} or equivalently C_{\max} .

In Section 4.4.2, we defined d_j to be the due date of job j and noted that L_j , the lateness of job j , is solely a function of the last train job j is carried to its destination on. This observation allows us to establish the following property with respect to the L_{\max} problem.

Property 4.3. Let $X(\ell)$ be any solution to the L_{\max} problem that transports all jobs to their destinations on M trains. Let job q be such that $L_q = L_{\max}$ and assume that q arrives at f_q on train Y ($Y > 1$) in $X(\ell)$. A necessary and sufficient condition for any other schedule $X'(\ell) \neq X(\ell)$ to be better than $X(\ell)$ is that $L_j(X'(\ell)) \leq L_j(X(\ell))$ for all $j \neq q$ and that q arrive at f_q on either of trains $Y-1, Y-2, \dots, 1$ in $X'(\ell)$.

4.5.1 Minimization of L_{\max} with Job Splitting

The job splitting assumption for the M train one track problem implies that any job j may be transported between o_j and f_j by using any combination of available trains.

Given the arrival of train Y at station s, the potential lateness of all jobs $j \in A_Y(s)$ is defined as

$$L_j(Y) = (a_j + p(o_j, f_j) + \sum_{i=1}^{Y-1} t_i) - d_j,$$

that is

$$L_j(Y) = \text{lateness of job } j \text{ if it were to arrive at its destination on train } Y.$$

Since the L_{\max} criterion is dependent upon a single job (the latest one), it is intuitively appealing that whenever a decision is made between two jobs, priority should be given to that job exhibiting the greatest potential lateness of the jobs considered. In this case, the notion translates into the following rule which can be used to generate a schedule $X(\ell)$ that minimizes $L(X(s)) \forall s = 1, \dots, \ell$ (where $L(X(s)) = \max_j L_j(X(s)) \forall j = 1, \dots, n; s = 1, \dots, \ell$).

Theorem 4.6. Any schedule having the property that given the arrival of train Y at station s for $Y = 1, \dots, M$ and $s = 1, \dots, \ell$.

- (i) If $|A_Y(s)| \leq 1$, then any job in $A_Y(s)$ is transported to s+1 on train Y,
- (ii) If $|A_Y(s)| > 1$, then any job in $A_Y(s)$ is transported to s+1 on train Y such that job q is transported and job r is not, then

$$L_q(Y) \geq L_r(Y),$$

that is

$$(a_q + p_q(o_q, f_q) + \sum_{i=1}^{Y-1} t_i) - d_q \geq (a_r + p_r(o_r, f_r) + \sum_{i=1}^{Y-1} t_i) - d_r$$

minimizes $L(X(\ell))$ for all schedules $X(\ell)$.

Proof. Optimality of Theorem 4.6 will be shown by induction on the number of stations s . The rule is obviously optimal for $\ell = 1$ and $\ell = 2$. Assume that the rule is true for $\ell = s^*$ and consider $\ell = s^*+1$.

Since we assume Theorem 4.6 to be optimal at s^* , then $L_j(X(s^*)) = L_j(X(s^*+1))$ for all jobs j whose destination is s^* . Therefore, consider the jobs that go from s^* to s^*+1 .

- (i) If all jobs transported from s^* to s^*+1 are taken on the same train that they were transported from s^*-1 to s^* on then the rule is clearly optimal at s^*+1 since $L(X(s^*+1)) = L(X(s^*))$.
- (ii) If all jobs transported from s^* to s^*+1 are not taken on the same train that they were transported from s^*-1 to s^* on and $L(X(s^*+1)) > L(X(s^*))$, then it must be the case that job k , where $L_k(X(s^*+1)) = L(X(s^*+1))$ is either
- (a) A new job (i.e., $o_k = s^*$). In this case if k is transported from s^* to s^*+1 on train $Y = 1$, the rule is clearly optimal. If k is transported on train $Y > 1$, then it must be shown that $L_j(Y) \geq L_k(Y) \forall j$ transported from s^* to s^*+1 on one of the trains $1, 2, \dots, Y-1$ in $X(s^*+1)$. But this follows by the rule since the new job was at the station when each of these jobs was sent.
- (b) An old job (i.e., $o_k < s^*$). In this instance k could not have gone on train $Y = 1$ from s^* to s^*+1 . Again it must be shown that $L_j(Y) \geq L_k(Y)$ for all j transported from s^* to s^*+1 on one of the trains $1, 2, \dots, Y-1$.

Let V be the train that k arrived at s^* on. Then $L_j(X(s^*+1)) \geq L_k(X(s^*+1))$ for all j transported from s^* to s^*+1 on one of the trains $V, V+1, \dots, Y$ since k was at s^* between trains V and Y .

Let q be the job that replaced k on train V at s^* . Let q^1 be the job carried on train $V-1$ from s^* to s^*+1 . Then again by the rule $L_{q^1}(V) \geq L_k(V)$.

If q^1 is such that $o_{q^1} = s^*$, then for all j on trains $1, \dots, V-1$ from s^* to s^*+1 , $L_j(V) \geq L_k(B)$.

If q^1 is such that $o_{q^1} < s^*$, then let V^1 be the train that q^1 arrived on at s^* . Repeating the above argument gives us

$L_j(X(s^*+1)) \geq L_k(X(s^*+1))$ for all j transported from s^* to s^*+1 on one of the trains $V^1, V^1+1, \dots, V, V+1, \dots, Y$;

$L_j(X(s^*+1)) \geq L_k(X(s^*+1))$ for all j transported from s^* to s^*+1 on one of the trains $V^2, V^2+1, \dots, V^1, V^1+1, \dots, V, V-1, \dots, Y$.

At some point one of the jobs q^2, q^3, \dots, q^u , is such that $o_{q^u} = s^*$ in which case

$$L_j(Y) \geq L_k(Y) = L_k(X(s^*+1))$$

for all j transported from s^* to s^*+1 on one of the trains $1, \dots, Y-1$.

If $o_{q^u} \neq s^*$ for some q^u , then it must be true that all jobs transported from s^* to s^*+1 are taken on the same train they were transported from s^*-1 to s^* on and by case (i) the rule is true. \square

Rather than having to compute $L_j(Y)$ when Y arrives at s for all j in $\Lambda_Y(s)$, we find that implementation of Theorem 4.6 is further facilitated by the following property:

Property 4.4. If $L_q(1) \geq L_r(1)$, then $L_q(Y) \geq L_r(Y)$ for all trains $Y = 2, \dots, M$ and any station s .

Proof. $L_j(Y) = L_j(1) + \sum_{i=1}^{Y-1} t_i$. Therefore, if $L_q(1) \geq L_r(1)$ we have that

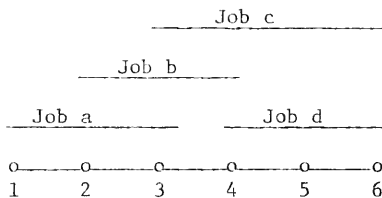
$$L_q(Y) = L_q(1) + \sum_{i=1}^{Y-1} t_i \geq L_r(1) + \sum_{i=1}^{Y-1} t_i = L_r(Y). \quad \square$$

It then follows from Theorem 4.6 and Property 4.4 that a schedule which minimizes $L_{\max}(C_{\max})$ is generated by the following procedure:

- (i) Compute $L_j(1)$ (or $C_j(1)$) for all jobs j .
- (ii) Rank the jobs in order of nonincreasing values of $L_j(1)$ (or $C_j(1)$).
- (iii) Whenever a decision has to be made between two jobs select q over r if q is ranked higher than r .

The procedure represents a more straightforward way of answering the question posed in 4.4.2 of generating a feasible schedule such that $L_j \leq 0$ for all k . By minimizing L_{\max} with M^* trains a schedule such that $L_{\max} \leq 0$ is generated which would then, of course, satisfy the due date restrictions.

Example: Consider the following problem involving four jobs and six stations where two trains travel the route with unit travel time between stations.



Based on the following data

j	d_j	a_j	$p(o_j, f_j)$	$L_j(1)$	Rank
a	4	0	2	-2	#3
b	3	1	2	0	#1
c	6	2	2	-1	#2
d	8	3	2	-3	#4

We have $b > c > a > d$ and an optimal schedule is given by:

Train 1: $\cdot \text{---} a \text{---} | \text{---} b \text{---} | \text{---} b \text{---} | \text{---} d \text{---} | \text{---} d \text{---} |$

Train 2: $| \text{---} \phi \text{---} | \text{---} a \text{---} | \text{---} c \text{---} | \text{---} c \text{---} | \text{---} c \text{---} |$

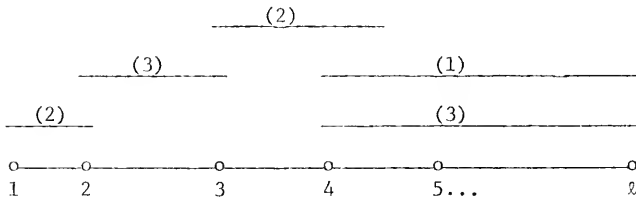
with $b > a$ at $s = 2$; $b > c$ at $s = 3$, and $l_{\max} = l_c = 1$.

4.5.2 Feasible Schedules Without Job Splitting

No job splitting for the M train one track problem requires that once a job j is picked up by train Y at o_j , it must arrive at f_j on the same train for all $j = 1, \dots, n$ and $Y = 1, \dots, M$. The problem we are interested in is:

(P-1) Given M trains with capacity $m_i = 1$ for all $i = 1, \dots, M$, can all the jobs be scheduled so that all satisfy their due dates when job splitting is not allowed?

In 4.4.2 we noted that the last train any job j can be delivered at f_j on so it arrives on or before its due date is easily determined in which case the situation can be illustrated by



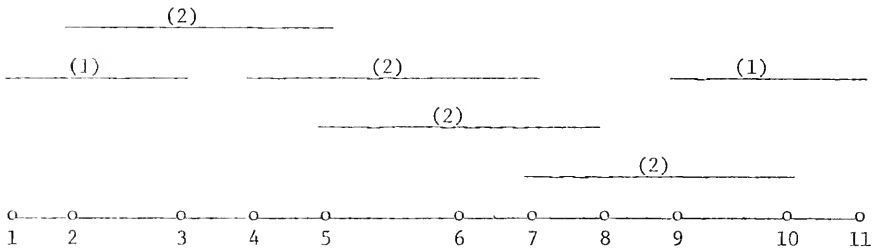
where (L) denotes the last train a job can go on in order to arrive by its due date. While efficient rules have been developed for job splitting problems, results in this section for no job splitting are limited to the case when $M = 2$. Again define

$\delta_{(L)}(s, s+1)$ = the number of jobs which overlap between stations s and $s+1$ from among those jobs which must be delivered to their destinations on train L or earlier.

Property 4.5. A necessary condition for all jobs j ($j = 1, \dots, n$) to be completed on or before their due dates is that

$$\begin{aligned} \text{(i)} \quad & \max_s \delta_{(1)}(s, s+1) = 1 \\ \text{(ii)} \quad & \max_s \delta_{(2)}(s, s+1) = 2 \\ & \vdots \\ \text{(M)} \quad & \max_s \delta_{(M^*)}(s, s+1) = M \end{aligned}$$

If Property 4.5 were sufficient, then a feasible solution to (P-1) could be generated by constructing a schedule of the same nature as generated in the job splitting case. However, Property 4.5 is not sufficient as illustrated by the following example for $M = 2$, $s = 11$.



$M = \max_{s=1, \dots, 11} \delta(s, s+1) = 2$. However, no feasible schedule exists so that

$L_j \leq 0$ for all j , $j = 1, \dots, 6$, with $M = 2$ trains.

The following algorithm can be used to generate a feasible schedule to (P-1) when $M = 2$.

(i) Check to see if the necessary condition of Property 4.5 is satisfied. That is,

$$\max_s \delta_{(1)}(s, s+1) = 1$$

and

$$\max_s \delta_{(2)}(s, s+1) = 2$$

- (ii) Eliminate all jobs that do not overlap with any other jobs from the problem by assigning them to train one.
- (iii) Assign train one to all remaining jobs with $(L) = 1$.
- (iv) Assign train two to all jobs that overlap with jobs already assigned to train one.
- (v) Assign train one to all jobs that overlap with jobs assigned in (iv).
- (vi) Repeat steps (iv) and (v) until either:
 - (a) Any assignment which must be made creates a conflict, in which case there does not exist a solution such that all job due dates are satisfied.
 - (b) All jobs are assigned and a solution such that all due dates are satisfied has been found.
 - (c) No overlap exists but there remain jobs to be assigned. In this case all remaining unassigned jobs are due on train two and can be assigned to either trains one or two. Since these jobs do not overlap with any job previously assigned, then the problem of assigning these remaining jobs is identical to that of finding a feasible solution such that all jobs arrive at their destinations. Since $\max_s \delta_{(2)}(s, s+1) = 2$, Corollary 4.1 can be used to generate a feasible schedule for the remaining unassigned jobs. A feasible solution with all due dates satisfied will then have been found.

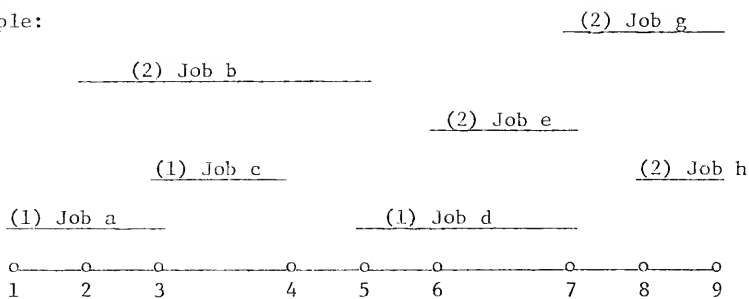
Proof. At each step, a necessary condition for the existence of a schedule where all jobs arrive by their due dates is satisfied. The condition for jobs that must arrive on train one is initially satisfied. The jobs that

must go on train one imply that a necessary condition for overlapping jobs must also be satisfied (i.e., these must go on train two), etc.

The algorithm either terminates due to lack of a feasible solution or with all jobs scheduled such that $L_j \leq 0$ for all j implying feasibility of the generated schedule. \square

Repeated application of the above algorithm for different values of L_j allows one to arrive at an optimal solution for L_{\max} when $M = 2$.

Example:



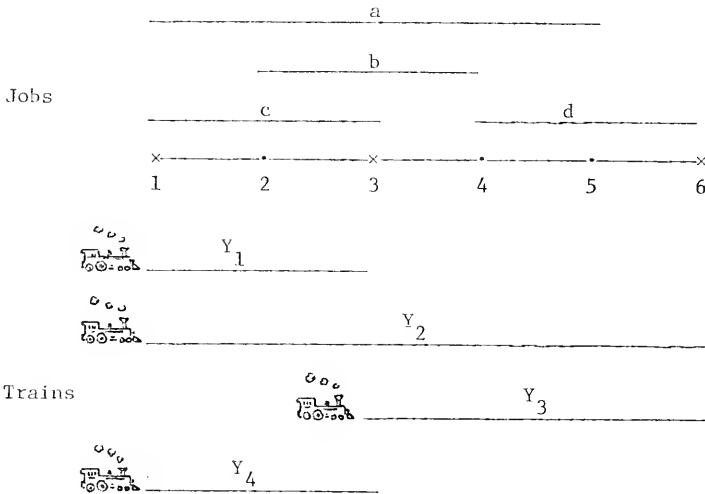
$$(i) \quad \max_{s=1, \dots, 9} \delta_{(1)}(s, s+1) = 1$$

$$\max_{s=1, \dots, 9} \delta_{(2)}(s, s+1) = 2$$

- (ii) Not applicable since all jobs have at least one overlap.
- (iii) Assign Jobs a, c, d to train one.
- (iv) Assign Jobs b and e to train two.
- (v) No further assignments can be made.
- (vi) At this point, both Jobs g and h remain unassigned. Assign the jobs arbitrarily, say g to train one and h to train two. All jobs are assigned and a feasible schedule has been constructed.

4.6 M Trains on One Track with Switchyards

In the development of all previous sections, it has been the case that all M trains travel the complete route from station $s = 1$ to $s = \ell$. However, when the route includes switchyards, this behavior may no longer hold. The switchyards play the role of stations at which new trains may originate or at which the movement of certain trains may cease. The inclusion of switchyards in the problem creates the concept of a route associated with each train, a situation which can be schematically modeled as follows:



where stations 1, 3, and 6 represent switchyards. The jobs $\{a, b, c, d\}$ play the same role as before. However, the trains $\{Y_1, Y_2, Y_3, Y_4\}$ now originate and terminate at various switchyards along the route. The basic difference between this situation and those modeled previously is that the carrying capacity provided by the trains is no longer constant for all track segments. If we assume that $m_Y = 1$ for all trains Y and let

$c(s, s+1)$ denote the carrying capacity between any two stations s and $s+1$ ($s = 1, \dots, \ell$) we have for this example that:

<u>$s, s+1$</u>	<u>$\delta(s, s+1)$</u>	<u>$c(s, s+1)$</u>
1 - 2	2	3
2 - 3	3	3
3 - 4	2	2
4 - 5	2	2
5 - 6	2	2

This is in contrast to the previous cases where $c(s, s+1) = M$ for all $s = 1, \dots, \ell$.

4.6.1 Train Schedules Based on Demand

We first consider the case where trains are scheduled based on demand (i.e., the movement of the trains is not based on an "a priori" fixed schedule). In this situation, trains do not move unless they can, in fact, transport a job.

Property 4.6. A necessary and sufficient condition for all jobs to be transported to their destinations is that

$$c(s, s+1) \geq \delta(s, s+1) \quad \text{for all } s = 1, \dots, \ell.$$

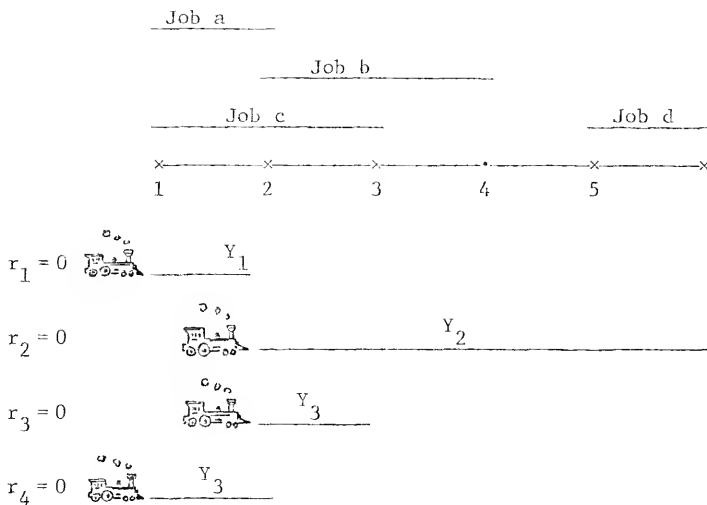
where we recall that $\delta(s, s+1)$ denotes the number of jobs (loads) which overlap between stations s and $s+1$.

Proof. The condition is clearly necessary. Sufficiency is established by assuming that $c(s, s+1) \geq \delta(s, s+1)$ for all $s = 1, \dots, \ell$. A schedule can now be constructed in which all jobs arrive at their destinations by simply starting with the first station and arbitrarily assigning the jobs

to any unfilled train. Then repeat the process at each station encountered along the route. If a station s' is reached where all loads cannot be assigned, it follows that $c(s', s'+1) < \delta(s', s'+1)$ which is a contradiction.

4.6.2 Fixed Train Schedules

We now consider the situation that arises when the schedule of the trains is fixed "a priori." In this case Property 4.6 is no longer sufficient to ensure the delivery of all jobs to their destinations. For example, consider the following problem where we assume that $m_Y = 1$ for all trains Y , and the travel time between any two stations is identical (i.e., one unit of time).



Since there is unit travel time between any two stations, the train schedules are specified by the start times of the trains. Letting r_Y denote the start time for train Y , where in this example the data are as follows:

$s, s+1$	$c(s, s+1)$	$\delta(s, s+1)$	Y	r_Y
1 - 2	2	2	1	0
2 - 3	2	2	2	0
3 - 4	1	1	3	0
4 - 5	1	1	4	0

We note that the necessary condition of Property 4.6 is satisfied. However, there does not exist a schedule which delivers all jobs to their destinations on trains Y_1, Y_2, Y_3 and Y_4 . The earliest moment Job c can arrive at station 2 is at $t = 1$, at which time because of the train schedule there is no train to transport it to station 3.

If, however, there does exist a schedule that transports all jobs to their destinations, then they can be delivered as follows:

Theorem 4.7. Assume that the necessary condition of Property 4.6 is satisfied. Furthermore, assume that all trains travel at the same constant speed (i.e., train passing is not allowed). Let $m_Y = 1$ for all trains $Y (Y = 1, \dots, M')$, where M' denotes the number of trains which travel various sections of the route). Then any schedule having the property that given the arrival of train Y at station $s, s = 1, \dots, \ell$.

- (i) If $|A_Y(s)| < 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y .
- (ii) If $|A_Y(s)| > 1$, then any job in $A_Y(s)$ is transported to $s+1$ on train Y such that if job q is transported and job r is not, then

$$p_q(s, f_q) \geq p_r(s, f_r)$$

(i.e., give priority to that job currently farthest away from its destination), transports all jobs to their destinations if at least one such schedule exists.

Proof of Theorem 4.7. Let $X'(\ell)$ be any schedule that delivers all jobs to their destinations. Let $t_Y(s)$ denote the arrival time of train Y at station s for any train $Y(Y = 1, \dots, M')$ and $s = 1, \dots, \ell$.

Lemma 4.2. Let q and r be any two jobs available for transporting from s to $s+1$ on train Y at $t_Y(s)$ (i.e., $q, r \in A_Y(s)$). Assume that q is transported to $s+1$ on train Y in $X'(\ell)$ but that $p_q(s, f_q) < p_r(s, f_r)$. If q and r are interchanged (that is, r is transported on Y to $s+1$ instead of q) both jobs can still be transported to their destinations without affecting the schedule of any other jobs or any of the trains that leave s on or before $t_Y(s)$.

Proof. Assume that r is transported from s to $s+1$ on train Z in $X'(\ell)$ where $t_Z(s) > t_Y(s)$. If q and r are interchanged at s , then after the interchange q travels on Z and r on Y from s to $s+1$. Consider station $s+1$ at which point either

- (i) q arrives at its destination or
- (ii) both q and r continue to $s+2$.

Case (i) If $f_q = s+1$, then clearly after the interchange q still arrives at its destination. Since $t_Y(s) < t_Z(s)$, r will arrive at $s+1$ earlier than in $X'(\ell)$ and can be transported from $s+1$ to f_r as prior to the interchange.

Case (ii) If $f_q > s+1$, let Y' and Z' be the trains that transport q and r , respectively, from $s+1$ to $s+2$ in $X'(\xi)$, where Y' leaves $s+1$ either before or after train Z arrives.

(a) Assume Y' leaves $s+1$ after Z arrives (i.e.,

$$t_{Y'}(s+1) \geq t_Z(s+1).$$

After the interchange q arrives at $s+1$ on train Z in which case it can still be transported from $s+1$ to $s+2$ on train Y' . It can then be transported from $s+2$ to f_q as before the interchange. Since r arrives at $s+1$ on train Y after the interchange, $t_Y(s+1) < t_Z(s+1)$ implies that r arrives at $s+1$ earlier than before the interchange in which case it can clearly be transported from $s+1$ to $s+2$ on Z' and from $s+2$ to f_r as before.

(b) Assume Y' leaves $s+1$ before Z arrives (i.e.,

$$t_{Y'}(s+1) < t_Z(s+1)).$$

In this case q arrives at $s+1$ too late to catch Y' . However, by interchanging q and r again at $s+1$ (i.e., q is transported on Z' from $s+1$ to $s+2$ and r is transported on Y') both q and r can be transported to $s+2$ without affecting any other jobs or any other trains except the ones they traveled on in $X'(\xi)$. After they are interchanged at $s+1$, the argument is repeated until either case (i) or case (ii) (a) occurs after which no further interchanges are required. Termination of the procedure is assured since $p_q(s, f_q) < p_r(s, f_r)$ (i.e., case (i) will always occur eventually unless (ii) (a) occurs earlier). \square

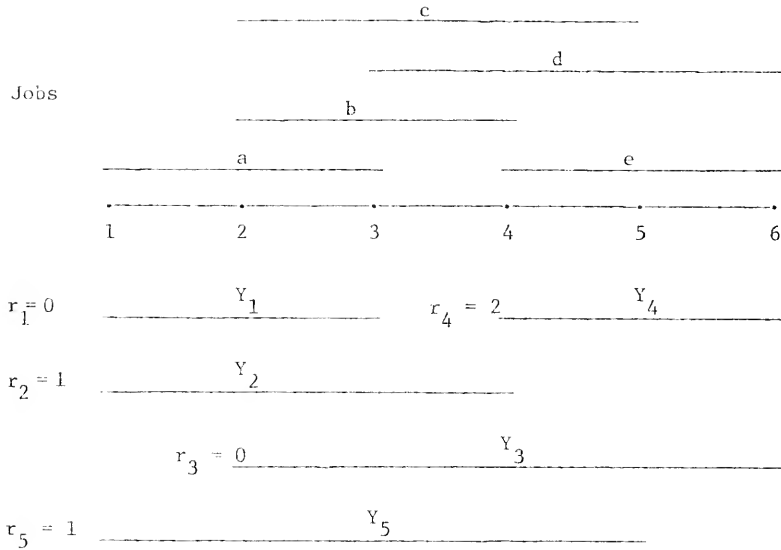
Now let $X^*(\xi)$ denote any schedule that adheres to the property of Theorem 4.7. Let $X'(\xi) \neq X^*(\xi)$ be any schedule that delivers all jobs to their destinations on M' trains. Let $t_1 < t_2 < \dots < T$ denote any

moment in time at which a train either leaves or arrives at a station where T is the total time required for all M' trains to complete their routes in $X'(\ell)$.

Consider t_1 and station $s = 1$. If no trains leave $s = 1$ at t_1 , then continue to $s = 2$. If a train Y leaves $s = 1$ at t_1 , let q be the job transported on it to $s = 2$. If $p_q(1, f_q) \leq p_j(1, f_j)$ for all $j \in A_Y(1)$, then continue to $s = 2$. If not, schedule r on train Y at t_1 and q on train Z at t_1 where r is such that $p_r(1, f_r) = p_j(1, f_j)$ for all $j \in A_Y(1)$ and Z transports r from $s = 1$ to $s = 2$ in $X'(\ell)$. It follows from Lemma 5 that after the interchange both q and r can still be delivered to their destinations. Repeat the procedure for $s = 2, 3, \dots, \ell$. When $s = \ell$, it will be the case that the schedule for t_1 has the property of Theorem 4.7. Consider now t_2 and $s = 1, \dots, \ell$. Since all interchanges occurring at t_2 will not affect the schedule at t_1 (Lemma 4.2) it follows that t_1 need never be reconsidered. After the schedule for t_2 is determined, it remains unaffected, and so on. The procedure terminates when the schedule for T is established in which case a schedule $X^*(\ell)$ has been generated. \square

It should be noted that it is not necessary to carry out Theorem 4.7 for all time periods $t_1 < t_2 < \dots, T$ unless it is the case that a feasible schedule is in fact generated. At any moment in time when the necessary condition of Property 4.6 is not satisfied, the procedure of Theorem 4.7 can be terminated since it would then be the case that no feasible solution exists.

Example: Consider the following jobs and trains where r_Y denotes the start time for train Y . We assume unit travel time for all trains between stations.



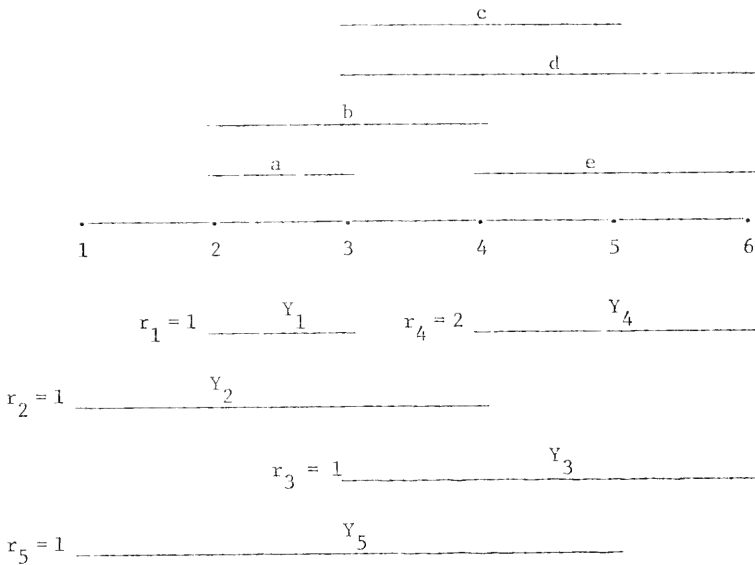
We first note that the necessary condition of Property 4.6 is satisfied, that is, $c(s, s+1) \geq \delta(s, s+1)$ for all $s = 1, \dots, 6$.

$s, s+1$	$\delta(s, s+1)$	$c(s, s+1)$
1 - 2	1	3
2 - 3	3	4
3 - 4	3	3
4 - 5	2	3
5 - 6	2	2

$t_1 = 0$: $s = 1$; Job a is transported from $s = 1$ to $s = 2$ on train 1.

$s = 2$; since $p_c(2, f_c) > p_b(2, f_b)$, Job c is transported from $s = 2$ to $s = 3$ on train 3.

After the trains move the necessary condition of Property 4.6 still holds and we have

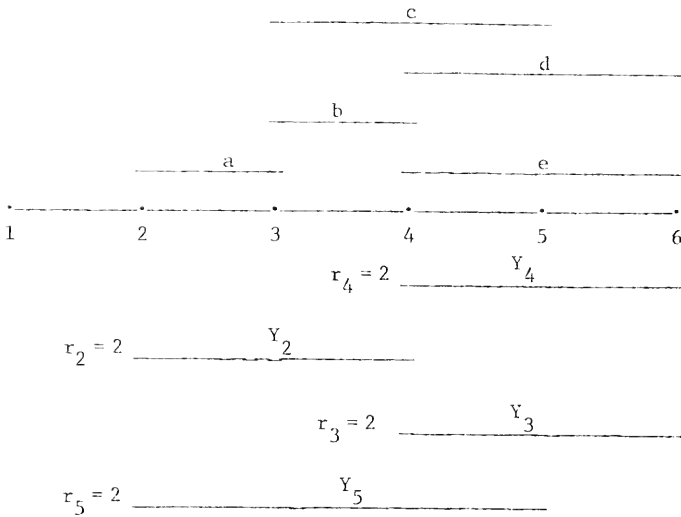


$t_2 = 1$: $s = 1$; no jobs are available.

$s = 2$; since $p_b(s, f_b) > p_a(2, f_a)$, Job b is transported from $s = 2$ to $s = 3$ on train 1.

$s = 3$; since $p_d(s, f_d) > p_c(s, f_c)$, Job d is transported from $s = 3$ to $s = 4$ on train 3.

After the trains move, train one completes its route, the necessary conditions still hold and we have

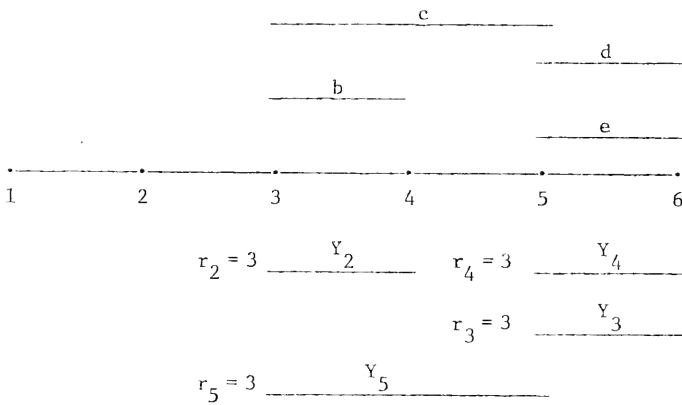


$t_3 = 2$: $s = 1$; N/A.

$s = 2$; Job a is transported on train two to $s = 3$.

$s = 3$; No trains available

$s = 4$; Jobs d and e are transported on trains 4 and 3 to $s = 3$.

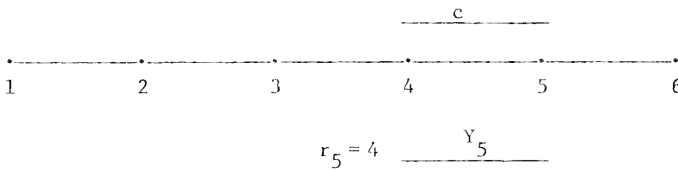


$t_4 = 3$: $s = 1, 2$: N/A.

$s = 3$: Since $p_c(3, f_c) > p_b(3, f_b)$, c is transported on train 2 to $s = 4$. Job b is picked up by train 5.

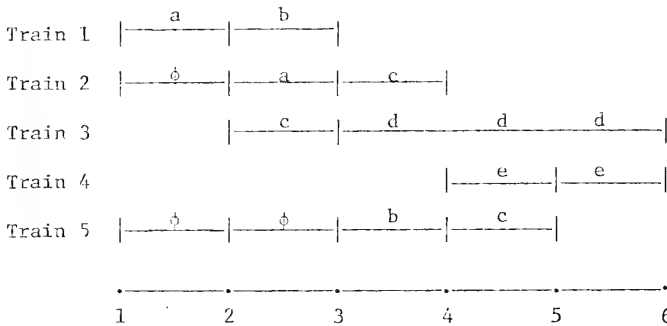
$s = 5$: Train 3 and 4 complete their routes by transporting d and e .

Finally, we have



and train 5 completes its route with Job c .

The schedule is summarized as follows:



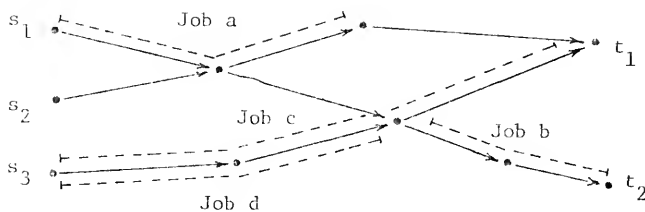
4.7 Routes Consisting of a Directed Network

In all previous sections it has been assumed that the movement of jobs and trains is along a single fixed route with one initial and one final station. Another way of describing this assumption is that our freight transportation network consists of a series of directed arcs with one source and one sink node. The case of parallel but noninteracting routes can be decomposed into independent single fixed route problems.

If the freight transportation network does not have the above form, the problem becomes quite complex (this is the situation faced by an actual railway system when all lines are considered). In this case, the results obtained are limited to that of determining the minimum number of trains required to deliver all jobs to their destinations for a freight transportation system consisting of a directed network.

Specifically, given a system which is a directed network $G = [N, A]$ where N denotes the set of stations and A the various track sections, let $S = \{s_1, s_2, \dots, s_n\}$ denote the stations at which trains originate and $T = \{t_1, t_2, \dots, t_m\}$ the set of stations at which trains terminate their runs.

For example:



in the above freight transportation system, trains leave either stations s_1 , s_2 or s_3 , pass through intermediate stations and eventually terminate their routes at either station t_1 or t_2 .

We again note that the role of the jobs consists in creating an overlap upon the system. If $m_j = 1$ for all trains, then each job creates the need for a train to travel along all arcs from a job's origin station to its destination.

Property 4.7. Let $\delta_{x,y}$ denote the total demand between any two stations x and y (i.e., along any arc (x, y)). Then $\delta_{x,y}$ is a lower bound on the number of trains required to cross arc (x, y) .

We also note that for any intermediate node $x \in \{N \cap (\bar{S} \cup \bar{T})\}$ it must be the case that the number of trains entering the station x must equal the number of trains which leave (basic conservation of trains property). If the trains are scheduled on demand (that is, there is no "a priori" train schedule), we find that:

Theorem 4.8. The minimum number of trains required to transport all jobs to their destinations is given by the minimum feasible flow from source to sink.

Proof. First model the system as a directed network with lower bounds $\delta_{x,y}$ for each arc (x, y) . Let S be a supersource and T be a supersink. Use any available algorithm [30] to find the minimum feasible flow M^* from S to T . We must show that all jobs can in fact be taken to their destinations on M^* trains (i.e., how to assign the jobs to the trains).

Consider any intermediate node x and consider the incoming arcs (i.e., the set $B(x) = \{y \in N \mid (y, x) \in A\}$). Take k to be any job arriving on a train at x . Either

- (i) k passes through the node and leaves on an outgoing arc. Then by leaving it on the same train it arrived on (i.e., assigning the train it arrived on to carry the job through its outgoing arc) we assure that k is taken beyond node x .
- (ii) $f_k = x$ (node x is job k 's destination). In this case then k has been taken to its destination.

After all jobs that are to be carried through node x have been assigned (i.e., these train assignments have been made), then all jobs which originate at x remain to be assigned. By assigning all remaining unassigned trains so as to complete the flow requirements on the outgoing arcs, we assure that these jobs are taken beyond x . \square

An assignment of trains to jobs which delivers all jobs to their destinations can be made by labeling the nodes (stations) from $1, 2, \dots, \ell$. For each node, assignments are made in accordance with the procedure in the above proof. The procedure terminates with node ℓ at which time all jobs are assigned in deliverable fashion.

4.8 Conclusion

Problems with scheduling and routing aspects exert a vital role in the efficient operation of transportation systems. In this paper very elementary systems based on the movement of freight along a local line of a railway system have been considered. In such a situation, the routing aspect is fixed, allowing one to concentrate exclusively on the scheduling portion of the problem. This simplification allows one to develop scheduling rules based on "locally" optimal system performance. These rules have the particularly nice property that they can be updated as jobs enter or leave the system.

The motivation for the rules developed is based on conceptualizing the problem as one of scheduling jobs on a system of parallel processors. For the single train case, the m boxcars are considered to be m identical processors whose function it is to process a set of jobs. The jobs are fixed and the processors move in time in such a way that any job not scheduled when the processors arrive is relegated to being

performed at some later date. The M train case then consists of a series of M parallel processors adding greater flexibility as well as complexity to the scheduling problem. However, insistence upon the use of a single fixed route allows efficient optimal rules to be developed for various criteria and for assumptions on job availabilities. The applicability of solution procedures common to more classical parallel processor problems for the solution of these "single fixed route" problems can be noted in the similarity of the decision rules developed in Sections 4.3 through 4.5 and those of Chapter 3. When the "single fixed route" assumption is relaxed we find that even the question of generating feasible schedules is not straightforward although results were attained for particular network structures.

The complexity of most transportation systems presents a serious challenge to those attempting to analytically model and perhaps solve some of the problems inherent in their operation. However the underlying mechanism of many a system can only be identified by stripping it to its most basic elements and building from there on. It is indeed fortunate that many of the problems considered in this chapter were amenable to efficient optimal rules, since we feel that the systems considered constitute the most basic structure of freight transportation. It remains to be seen whether the insight developed will carry onward to problems of a more general nature.

CHAPTER 5

MIN-MAX PARALLEL PROCESSOR ALLOCATION PROBLEMS

5.1 Introduction

This chapter is concerned with a class of parallel processor problems characterized by optimality criteria that are a function of how much of a job has been completed by a certain period. This is in contrast to criteria considered in previous chapters which are a function of job completion times. Specifically the problem of interest concerns the minimization of a cost function associated with processing a set of jobs on m identical processors over a fixed time horizon H where:

- (a) It is known how much of each job should be completed by period t ($t = 1, \dots, H$) (i.e., there exists a fixed schedule for completing each job).
- (b) The penalty cost for each job is a function of how much of the job has been completed by time t (i.e., a penalty is assessed for being off the fixed schedule).

In Chapter 2 (section 2.9.2.A) it was mentioned that Dorsey, Hodgson and Ratliff [20, 21, 22] have shown that problems adhering to the above description which arise in production scheduling contexts can be modeled as minimal cost network flow problems. An example of their formulation for a two job, three period problem such that job

two is available only in periods two and three is given in Figure 3. To construct the network, define a node m_t corresponding to each period t and define a node n_{jt} corresponding to each job j and period t in which j is available. From each node m_t , construct a forward arc (m_t, n_{jt}) to each node n_{jt} for all $j = 1, \dots, n$. Since only one machine can be assigned to each node in each period, an upper bound of one is placed on each arc (m_t, n_{jt}) . Next connect each node n_{jt} to node $n_{j,t+1}$ by two parallel forward arcs $(n_{jt}, n_{j,t+1})$ and one reverse arc $(n_{j,t+1}, n_{jt})$. An upper and lower bound \bar{w}_{jt} is assigned to one of the forward arcs corresponding to the number of machine periods of production of job j required by period t to meet the production schedule. If more machine periods of production are completed by period t than required, this flow on the other forward arc is assessed a cost representing the penalty for exceeding the schedule. Analogously, flow on the reverse arc is assessed a cost representing the penalty for having completed less than required by period t . Finally define a source node S and a sink node T where a forward arc (S, m_t) is constructed for all $t = 1, \dots, H$ and a forward arc (n_{jH}, T) is constructed for all $j = 1, \dots, n$. A capacity of M , the number of processors is placed on the flow through each of the arcs (S, m_t) . Since each job j must be completed at the end of the horizon, a lower bound of w_{jH} denoting the number of machine periods required to completely process job j is placed on the flow through the arc (n_{jH}, T) . The problem is then solved using any of the algorithms for finding minimal cost flows in single commodity networks [30].

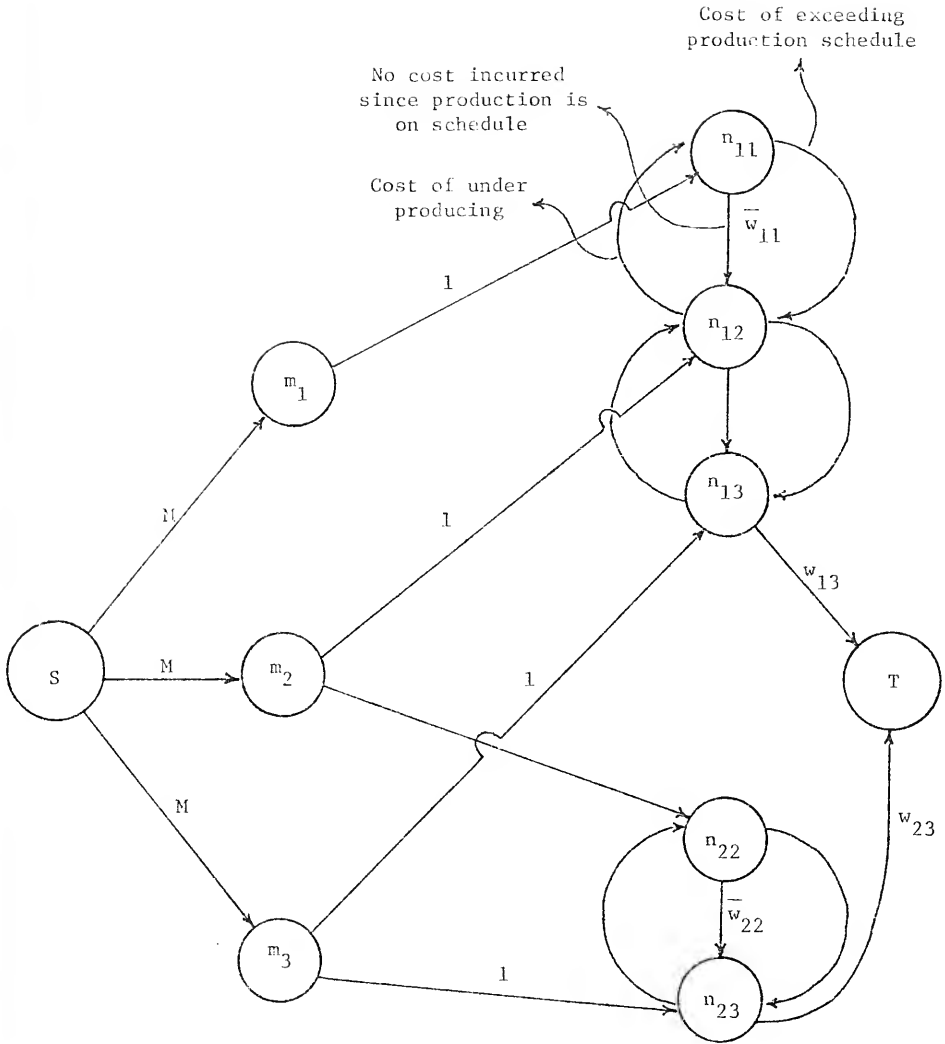


Figure 3: Example of a Two Job, Three Period Problem with M Identical Processors.

This network formulation extends beyond problems of a production scheduling context. In this chapter we shall apply the network flow formulation to problems involving the allocation of mine countermeasures which we find can be modeled as parallel processor problems of this form.

5.2 Formulation

The context of the problems to be considered here will be that of providing the "defender" of a logistics system under mining attack with a quantitative basis for allocating his mine countermeasures (MCMs). The "defender" plays the role of the "scheduler" and the countermeasures which consist primarily of helicopters and their support equipment are the parallel processors. The logistics system consists of a set of ports which are mined. The clearing of each port represents a job to be carried out whose completion requires the assignment of a minimum number of MCMs. A penalty is incurred depending on the latest time each job is completed, where the processing of the job may be split among the parallel processors (the MCMs). The responsibility of the defender is to allocate his MCMs in such a way that the losses (penalties) incurred by the system are not extreme.

McWhite and Ratliff [51] have developed optimal and near optimal procedures for various problems in which the measure of effectiveness for MCM allocation consists of minimizing total shipping losses. Here the prime concern will be in obtaining optimal min-max allocations for the following two cases:

Case 1:

We assume that there is a loss probability associated with getting goods out of a port. The probability of loss is a nonincreasing function of the number of countermeasures assigned to the port. In addition, we have a schedule of what has to be shipped out of the ports and we assume that things go out on schedule. The objective in this case is to deploy the MCMs so that the maximum loss probability of the ports is minimized.

Case 2:

We assume that the goods in a port cannot leave until the port is reasonably cleared (i.e., until the loss probability of a port is less than or equal to some satisfactory level). A penalty is incurred for each period a shipment is delayed. The penalties are assumed to be nondecreasing functions of time. The objective for this situation is to minimize the maximum penalty incurred by all ports.

5.3 Case 1: Minimization of Maximum Loss Probability

To facilitate modeling this case we will consider our planning horizon to consist of H time periods. We will allow the MCMs to be redeployed among ports but only between periods. For $i = 1, 2, \dots, m$

and $t = 1, 2, \dots, H$, let $\ell_{it} \left(\sum_{k=1}^{t-1} q_{ik} \right)$ represent the loss probability for port i in period t where q_{ik} is the number of MCMs assigned to port i in period k . Implicit in this definition of the loss probability is the assumption that the effect of using an MCM is independent of the period. This min-max allocation problem under these assumptions

can be formulated as:

$$\begin{aligned} & \min \max_{i, t} \ell_{it} \left(\sum_{k=1}^t q_{ik} \right) \\ & i = 1, \dots, m \\ & t = 1, \dots, H \\ \\ & \text{subject to} \quad \sum_{i=1}^m q_{it} \leq Q_t \quad \text{for } t = 1, 2, \dots, H \\ \\ & 0 \leq q_{it} \leq \bar{q}_{it} \quad \text{and integer for all} \\ & i = 1, 2, \dots, m \text{ and} \\ & t = 1, 2, \dots, H \end{aligned}$$

where Q_t is the number of MCMS available during period t and q_{i0} is defined to be zero for $i = 1, 2, \dots, m$.

The system can be modeled as a directed network by creating nodes n_t for $t = 1, 2, \dots, H$ and p_{it} for $i = 1, 2, \dots, m$ and $t = 1, 2, \dots, H$. For $t = 1, 2, \dots, H$, construct an arc from n_t to p_{it} with capacity \bar{q}_{it} and associate a loss probability of zero with each of these arcs.

For $t = 1, 2, \dots, H-1$, and $i = 1, 2, \dots, m$, construct an arc from p_{it} to

$p_{i,t+1}$ with capacity $\sum_{k=1}^t \bar{q}_{ik}$ and loss probability $\ell_{it}(f(p_{it}, p_{i,t+1}))$,

where $(f(p_{it}, p_{i,t+1}))$ is the arc flow. Construct a source node S with arcs from S to n_t having capacities Q_t and loss probabilities of zero for $t = 1, 2, \dots, H$, and a sink node T with arcs from $p_{i,H}$ to

T with capacities $\sum_{t=1}^H \bar{q}_{it}$ and loss probabilities $\ell_{it}(f(p_{iH}, T))$

(Loss Probability, Capacity, Lower Bound)

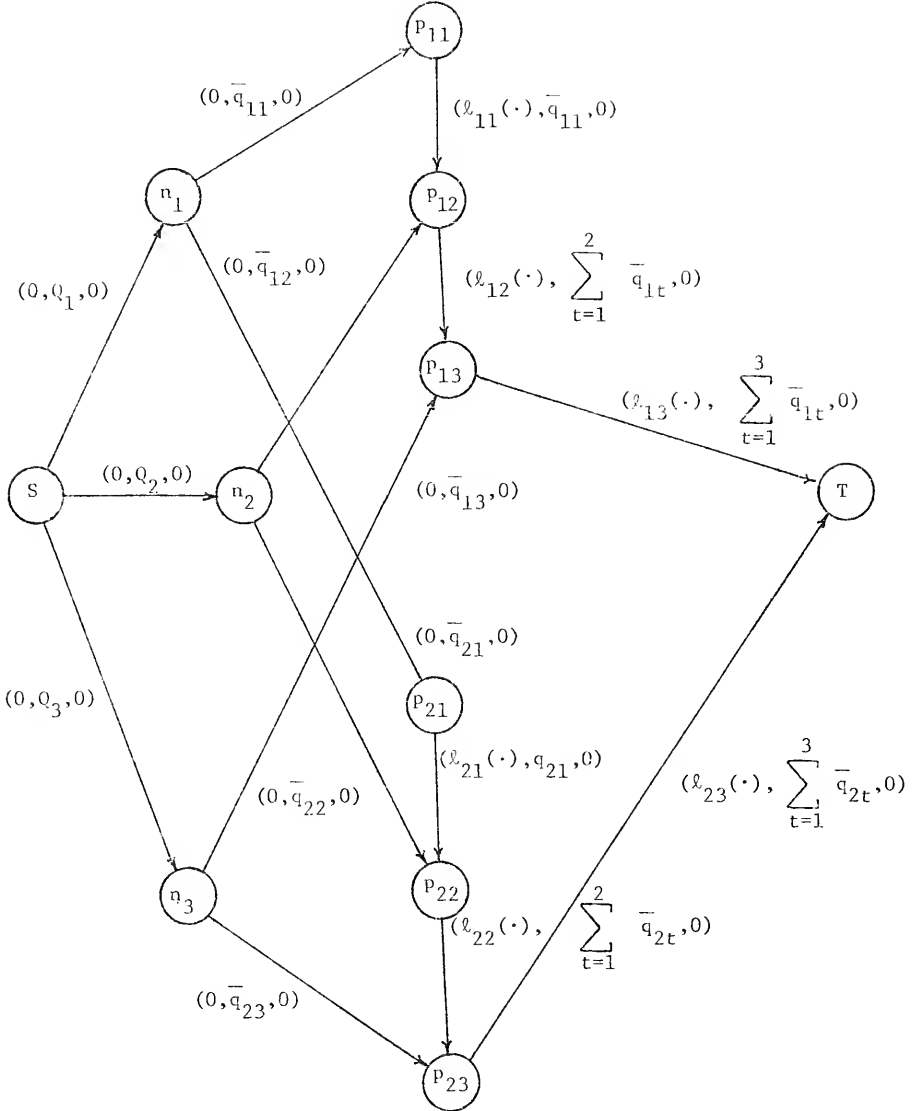


Figure 4: Network Representation of a Two Port Three Period Min-Max Loss Probability Allocation Problem

for $i = 1, 2, \dots, m$. Finally, assign a lower bound of zero to all arcs. An example network for a 2 port 3 period system is shown in Figure 4. (This network structure is the same as that presented in McWhite and Ratliff [51].)

If we assume that $\ell_{it}(\cdot)$ is a monotonically nonincreasing function of the number of MCMs assigned to port i in period t for $i = 1, 2, \dots, m$ and $t = 1, 2, \dots, H$, then the following procedure provides an efficient method for minimizing the maximum loss probability:

- (i) Construct a feasible flow from S to T .
- (ii) Let $\ell_{it}^* = \max \ell_{it}(\cdot)$ for all $i = 1, \dots, m$ and $t = 1, \dots, H$.
- (iii) A necessary and sufficient condition for there to exist a better solution is that the loss probability on every arc be strictly less than ℓ_{it}^* . Therefore, increase the lower bound on each arc to the smallest integer value it can assume so that $\ell_{it} < \ell_{it}^*$ for all $i = 1, \dots, m$ and $t = 1, \dots, h$.

(iv) Repeat steps (i), (ii) and (iii) until no feasible flow exists. The deployment corresponding to the last set of feasible flows found minimizes the maximum loss probability.

Optimality of the algorithm is insured because at each step a necessary and sufficient condition for improving the current solution is generated. The algorithm terminates because the maximum flow is finite and at least one lower bound is increased by an integer at each step.

It should be noted that the algorithm used for the min-max allocation problem can be extended to any general directed network with monotonically nonincreasing functions on the arcs. (See Figure 5 for

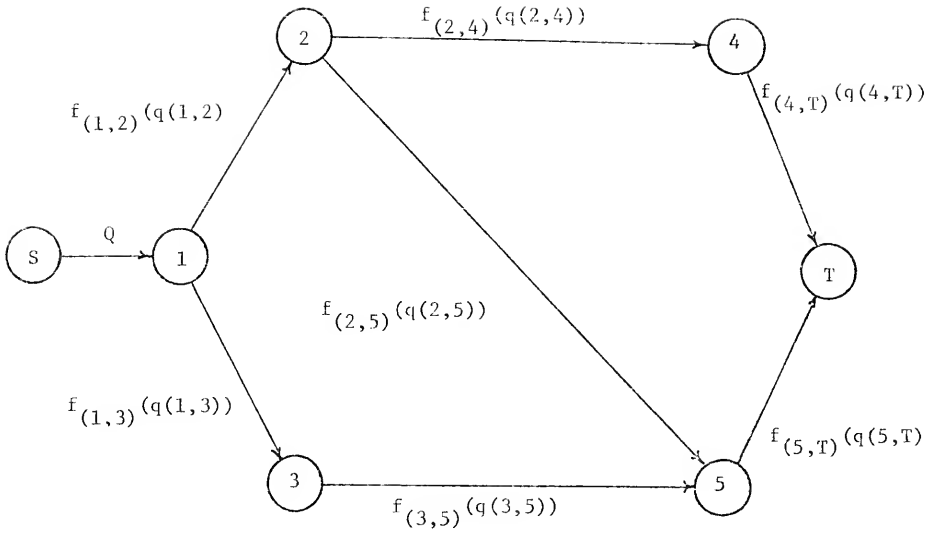


Figure 5: An Example of a Min-Max Allocation Problem on a Directed Network

an example network.) That is, given a directed network $G = [N, A]$, suppose that each arc $(x,y) \in A$ has associated with it a monotonically nonincreasing function $f_{(x,y)}(q(x,y))$ where $q(x,y)$ denotes the number of resources assigned to arc (x,y) . Then an optimal solution to the problem:

$$\min_q \max_{(x,y) \in A} f_{(x,y)}(q(x,y))$$

subject to:
$$\sum_{(x,y) \in A} q(x,y) \leq Q$$

$$\sum_{y \in A(x)} q(x,y) - \sum_{y \in B(x)} q(y,x) = \begin{cases} Q & x=S \\ 0 & x \neq S, T \\ -Q & x=T \end{cases}$$

$$q(x,y) \geq 0 \text{ and integer}$$

where: $A(x) = \{y \in N \mid (x,y) \in A\}$

$$B(x) = \{y \in N \mid (y,x) \in A\}$$

is obtained by

- (i) Constructing a feasible flow from S to T.
- (ii) Let $K^* = \max f(q(x,y))$ for all (x,y)
- (iii) Increase $q(x,y)$ for all (x,y) to the smallest value it can take such that $f(q(x,y)) < K^*$ for all (x,y) .
- (iv) Repeat steps (i), (ii) and (iii) until no feasible flow exists. The last set of feasible flows is optimal.

This algorithm is an extension of methods used by Jacobsen [43] and Porteus and Yormak [58] for the solution of min-max marginal allocation problems of the form:

$$\min_q \quad \max_i \quad f_i(q_i)$$

subject to:
$$\sum_{i=1}^n q_i = Q$$

$$q_i \geq 0 \text{ and integer for all } i=1, \dots, n$$

where f_1, f_2, \dots, f_n are n real-valued monotonically nonincreasing functions defined on the nonnegative reals and Q is a positive integer denoting the number of units of scarce resource.

5.4 Case 2: Minimization of Maximum Penalty

Again we allow our planning horizon to be made up of H time periods and we allow the MCMs to be redeployed among ports but only between periods. Let Q_t denote the maximum number of MCMs which can be allocated to all ports in period t and q_i denote the number of MCM days required to work on port i for it to be cleared. Also let \bar{q}_{it} denote the maximum number of MCMs which can be assigned to port i in period t and r_i denote the first period that port i can be worked on. In this case we will assume that a penalty $c_i(t_i)$ is incurred if port i is not cleared until period t_i . We also require that all ports be completely cleared by period H .

This problem can be formulated as:

$$\min_t \quad \max_{i=1, \dots, m} \quad c_i(t_i)$$

subject to:
$$\sum_{t=r_i}^{t_i} q_{it} = q_i \quad i = 1, 2, \dots, m$$

$$\sum_{i=1}^m q_{it} \leq Q_t \quad t = 1, 2, \dots, H$$

$$0 \leq q_{it} \leq \bar{q}_{it} \quad \text{for all } i = 1, 2, \dots, m \\ t = 1, 2, \dots, H$$

$$r_i \leq t_i \leq H \quad \text{for } i = 1, 2, \dots, m$$

and q_{it} and t_i integer.

This system can be modeled as a network by creating nodes n_t for $t = 1, \dots, H$ and p_i for $i = 1, 2, \dots, m$. For $t = r_i, r_{i+1}, \dots, H$, construct an arc from n_t to p_i with capacity \bar{q}_{it} . Construct a source node S with arcs from S to n_t having capacities Q_t and a sink node T with arcs from p_i to T with both an upper and lower bound of q_i . An example network for a 2 port, 3 period system where $r_1 = 1$ and $r_2 = 2$ is shown in Figure 6.

If we assume that $c_i(t)$ is a monotonically nondecreasing function of t for all $t = 1, \dots, H$ and $i = 1, \dots, m$, then the following observation can be made:

Given any feasible deployment and its associated objective function value, a necessary and sufficient condition for generating a strictly better deployment is that the penalty costs for all ports in the new solution be less than the current objective function value. This condition, in terms of the network, implies that for a better deployment to be obtained, flow may be allowed only on arcs from p_i to t for any t such that $c_i(t)$ is strictly less than the current objective function value. (This notion is utilized by Bratley et al. [13] to solve a related parallel processor scheduling problem.)

(Capacity, Lower Bound)

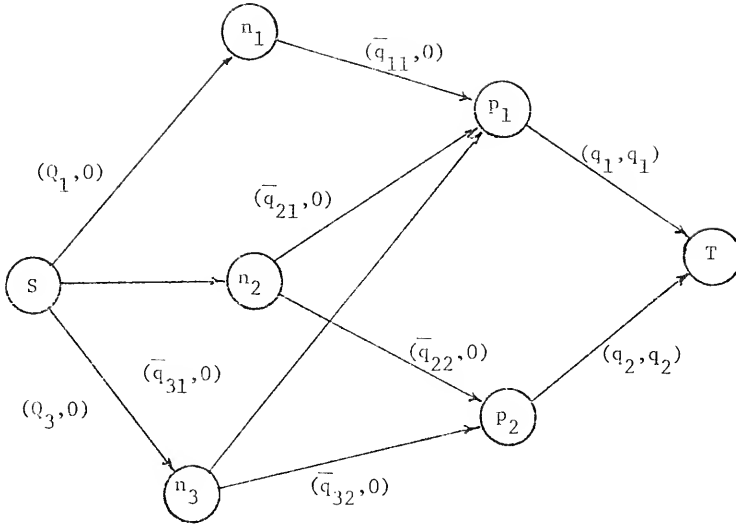


Figure 6: Network Representation of a Two Port, Three Period Min-Max Penalty Allocation Problem

Based on this condition, the following algorithm can be used to find a deployment that minimizes penalty cost.

- (i) Generate a feasible flow for the network and calculate the penalty costs associated with the feasible solution.
- (ii) Let $K = \max_i c_i(t)$ for all $i = 1, \dots, m$ and $t = 1, \dots, H$.
- (iii) For all $i = 1, \dots, m$, determine the latest period that p_i can be cleared so that $c_i(t)$ is strictly less than K .
- (iv) Delete all arcs from p_i to all periods which extend beyond the latest period determined in (iii) for all $i = 1, \dots, m$.
- (v) Repeat steps (i) through (iv) with the new network generated at each iteration until no feasible flow can be attained. The deployment corresponding to the last feasible flow is optimal.

As in Case 1, optimality of the algorithm follows from the fact that at each step a necessary and sufficient condition for improving the current solution is satisfied. Termination of the algorithm upon arriving at the first infeasible solution implies that the solution before the last step is optimal.

5.5 Conclusions

In this chapter two cases of min-max parallel processor allocation problems have been considered. Both problems are found to adhere to a network flow formulation of the type utilized for the solution of related problems which arise in production scheduling contexts. While these problems differ from those considered in the previous chapters,

chronologically they represent the first parallel processor situations approached in this study. They can be related to the previous work by observing that the notion used in developing algorithms for both cases was found to extend to the solution of parallel processor problems such as those considered in sections 3.6 and 4.5.2.

CHAPTER 6

SUMMARY AND SUGGESTIONS FOR FUTURE RESEARCH

In this dissertation a study of some parallel processor scheduling problems has been presented. We now summarize and comment upon the main contributions of this study which are contained in Chapters 3 and 4.

Chapter 3 presents some optimal rules and algorithms for problems originating from the theory of parallel processor scheduling. The developments of this chapter were motivated by the belief that more efficient methods could be constructed for dealing with parallel processor problems, some of which had been considered previously in the literature; thus the emphasis on optimal rules rather than enumerative type algorithms. Moreover, an extensive review of the previous literature (Chapter 2) served to support an initial observation that many current results in the area were isolated ventures with little interrelationship. This led to the objective that identification of some basic problems in the area would not only lead to the development of optimal priority type rules for the solution of the same, but also serve to link previously unrelated observations and results for both single and multiple processor problems. The results of Chapter 3 can be considered as achieving but partial success with respect to this original, albeit, optimistic objective.

By allowing a limited form of job splitting, a priority based rule was shown to minimize maximum lateness for the parallel processor problem without restrictions on job or machine availabilities. Unfortunately, the rule did not extend optimally to the solution where jobs were not simultaneously available although a rule for minimizing maximum flow time for this more general problem was found optimal. Both results are contributions to the theory for a very specialized parallel processor situation but lack the impact and generality attainable if say the rule for minimizing lateness had also been shown optimal for nonsimultaneous release times where current results consist of a network formulation. Nonetheless, the rules do tie in many other results in the area.

A by-product of this part of the study has been to identify the extent to which optimal priority based procedures will play a role in the solution of parallel processor problems. It may well be the case that the assumptions imposed on the problems for which rules were attained may mark the perimeter beyond which the development of optimal rules becomes highly improbable. This is certainly a strong conjecture for problems where job splitting is not allowed (partitioning problems with a high degree of discreteness and complexity). While efforts will surely continue to encompass more general problems than those considered here with optimal rules, it is felt that even when job splitting is allowed, this does not constitute a very promising area for future research.

The most attractive area for future development concerns the class of problems presented in Chapter 4. In this chapter, the

scheduling of freight on a class of "fixed route" railway systems was modeled as a parallel processor problem. It is important to emphasize that the only distinction between the problems in this chapter and those considered in Chapter 3 concerns the limited availability of the machines. In the fixed route problems jobs not carried by a machine (i.e., freight not carried by a train) at a particular moment in time can no longer be processed by the same machine and must wait until the next processor arrives or becomes available. This is in contrast to the more classical formulation of the parallel processor problem where jobs not processed at any point in the scheduling horizon "queue up" and wait until they are processed by either the same or another machine. Yet it is this property that opens up a new area of scheduling problems amenable to priority based optimal rules for the variety of criteria and assumptions considered in Chapter 4.

The development of this new area proceeds from the simplest case of one train (processor) traveling a single fixed route to the situation where M trains travel the route. It is shown that by giving priority to jobs currently closest to their destinations, the number of jobs taken to their destinations by the single train as well as the first K trains ($K \leq M$) of an M train system is maximized. An important characteristic of this criterion is that the rule, while dependent on job splitting for its implementation, provides sufficient information to develop solutions for no job splitting with the same optimal value. This is especially of interest when it is remembered that the no job splitting assumption provided an insurmountable obstacle for the development of optimal rules in the more classical

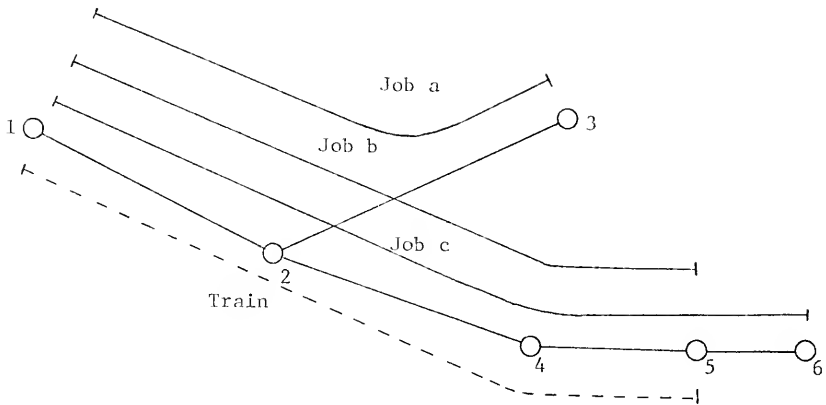
problems of Chapter 3. One then observes that an approach for dealing with other parallel processor problems without job splitting could consist of looking first at the job splitting case and trying to identify those situations where the value of the relaxed problem is in fact optimal when jobs cannot be split. In any case, optimal solutions to job splitting cases will always provide bounds on the optimal values of their no job splitting counterparts.

When restrictions are imposed on job availabilities, the analysis was not so straightforward. When job splitting was allowed, feasibility conditions and rules for transporting all jobs to their destinations were developed for cases when jobs are not simultaneously available or job due dates are invoked. However, inclusion of both constraints and/or a no job splitting assumption makes even the question of feasibility presently intractable. This latter occurrence identifies a niche for future analysis within the framework of fixed route problems.

Relaxing arrival restrictions on the jobs allowed one to minimize lateness for the M train case when again job splitting is allowed. No job splitting results were confined to the case of two trains and attempts for $M \geq 3$ were thwarted by the highly combinatorial and unstructured nature of the problem that evolved.

The last two sections of Chapter 4 present scenarios of increasing practicality and not too surprisingly, of increasing difficulty. Allowing trains to originate and/or terminate their progress at different stations introduces the notion of a route associated with each train although all train routes are still confined to the same

single track. Enlarging the situation to systems of tracks (networks, in essence) increases dramatically the complexity of the problem. In making any decision concerning the loading of a job on a train, it no longer suffices to consider only the remaining processing time of a job. One must also take into account where the job is to go (i.e., the job route) and the train route. For example, in the one train system below where the solid lines denote the job route and the broken line the train route,



maximizing the number of jobs taken to their destinations requires that priority be given to Job b which is neither the job closest to or farthest away from its destination at station 1. In general, the inclusion of non-fixed routing aspects into the problem destroys much of the "locally optimal" properties of the fixed routing case. However, it is quite probable that the special structure of networks such as trees can be capitalized upon to generate efficient rules and algorithms.

In conclusion, it is felt that there exist greater incentive and promise for analyzing and exploring more general situations than those considered in Chapter 4. This study has served to introduce and develop a new area of scheduling on parallel processors. Further extensions of this research not only serve the goal of breaching the gap between the simplified but insightful special cases we have considered and the highly complicated situations encountered in freight transportation, but also add to an increased understanding of parallel processor systems, be they of theoretical or applied origin.

APPENDIX 1

COMPUTATIONAL RESULTS

A.1.1 Introduction

As noted in Chapter 3, the shortest remaining slack time rule does not extend optimally to the situation where jobs are not simultaneously available. However, experience with the problem led the author to believe that with a slight modification the heuristic might demonstrate near optimal behavior as well as a very high degree of computational efficiency. To test this conjecture, an experiment was carried out. The global result for 279 different problems was that in 277 cases the rule yielded an optimal solution. A description of the experiment and details of the results obtained is now presented.

A.1.2 Description of the Experiment

The heuristic tested for solving the n job m machine problem with job splitting and nonsimultaneous release times consists of scheduling the jobs so that:

For any $t' = 1, 2, \dots, t$ schedule all $k \in \{A(t') \cap C(X(t'))\}$

so that

- (i) If $|A(t') \cap C(X(t'))| \leq m$, then process all jobs in $A(t') \cap C(X(t'))$.
- (ii) If $|A(t') \cap C(X(t'))| > m$, then process any m jobs in $A(t') \cap C(X(t'))$ such that if job q is processed and s is not, then either

$$(a) d_q(t') - p_q(t') < d_s(t') - p_s(t')$$

or

$$(b) d_q(t') - p_q(t') = d_s(t') - p_s(t')$$

and

$$p_q(t') \geq p_s(t')$$

The algorithm was coded in APL/360 [32] and consists of three functions:

1. Function "DGEN"

The function "DGEN" makes use of the built-in random number generator of APL to generate

- (i) Random processing times, P, between 1 and 20.
- (ii) Random release time, R, between 1 and 15, and
- (iii) Random due dates between $P + R$ and $P + R + \text{TABLE}$. The smallest due date for any job is at least as large as its processing time plus release time (i.e., no job initially starts off late). The TABLE entry is a random number whose upper limit is fixed before each set of runs allowing one to exercise control over whether job due dates are "tight" or "not so tight." For example, TABLE=15 causes due dates to be randomly generated between $P + R$ and $P + R + 15$. The minimum values of R plus 1 is subtracted from all values generated in (ii) before the random due dates are generated.

The function is activated by specifying the values m, the number of machines, and n the number of jobs, as well as the value of TABLE for a particular set of runs. Initialization of COUNTER = # allowed # n job m machine sets of data to be generated.

```

          EDGEN[ $\bar{E}$ ]
V 1 DGEN N
[1] DL:P*?N; 20
[2] PTEMP←P
[3] R←?N; 15
[4] R←R-(1/R)
[5] R←R+1
[6] D←R+P+?N; TABLE[N-1]
[7] PP
[8] COUNTER←COUNTER-1
[9] --DL≠0≠COUNTER

```

As each problem is generated it is fed into:

2. Function "PP"

Function "PP" executes the scheduling algorithm by initially comparing all values in the vector P-D and selecting the m jobs which have priority. Then all processing times are updated and the procedure is repeated until $|C(X(t'))| = 0$.

```

VPP[ $\bar{Q}$ ]
V PP
[1] LB← $\bar{I} / ( \bar{I} / ((R-1)+P-D) , \bar{I} / ( \bar{I} / ((+/P) : M) , ( \bar{I} / (R-1) + P ) ) - \bar{I} / D )$ 
[2] MAS←(2,M)∅0
[3] T← $\bar{I} / R$ 
[4] LOOP:FLAG←(0<P) ^ (T≥R)
[5] S←(P-D)
[6] ST←FLAG/S
[7] PT←FLAG/P
[8] JOB←FLAG/((1∅FLAG)×FLAG)
[9] →ZERO×10 = RO+∅ST
[10] →((RO=M) , (RO>M) , RO<M)/EQL,CONT,LSS
[11] LSS:TEMP←JOB, ( M-∅ST)∅0
[12] →PR
[13] EQL:TEMP←JOB
[14] →PR
[15] CONT:ST+ST[TT+∅ST]
[16] PT←PT[TT]
[17] JOB←JOB[TT]
[18] TEMP←CNT+TIE+∅I+1
[19] SET: >ALGO×1ST[I]=ST[I+1]
[20] →NEXT×1TIE=0
[21] TEMP←TEMP, JOB[(I-TIE) TO I]
[22] →BY
[23] NEXT:TEMP←TEMP, JOB[I]
[24] BY: CNT←CNT+TIE+1

```

```

[25] →((CNT>M), (CNT<M), CNT=M)/GREAT, LESS, EQUAL
[26] ALGO: TIE+TIE+1
[27] →THRU
[28] LESS: TIE+0
[29] THRU: I+1+1
[30] →TAKE: I=L+ST
[31] →SET
[32] TAKE: TI+φ(TIE+1)+φJOB
[33] CNT+φST
[34] →FIN
[35] GREAT: TI+φ(TIE+1)+φTEMP
[36] TEMP+φ(TIE+1)+φTEMP
[37] FIN: EPT+P[TI]
[38] OEPT+ EPT[XX+φEPT]
[39] TEMP TEMP, (M-(CNT-(TIE+1))) +OTI+TI[XX]
[40] EQUAL: TEMP+L+TEMP
[41] PR: MAS ROWCAT TEMP
[42] P←(P-((IOP)εTEMP))
[43] D←(D-1)
[44] T←T+1
[45] →LOOP
[46] ZERO: →PPSTOP×I=Λ/0=P
[47] TEMP+Mφ0
[48] →PR
[49] PPSTOP: MAXL MAS

```

∇

Once all jobs have been scheduled the program passes on to:

3. Function "MAXL"

Function "MAXL" computes the lateness of all jobs in the schedule by comparing the job completion times with their due dates. Once the lateness vector L is generated, it is scanned to identify the value of the maximum lateness.

```

V MAXL[ ]∇
V MAXL MAS
[1] DIM←ρMAS
[2] ITER←DIM[1]-2
[3] MAS←(0 0 , ITERρ1)/[I] MAS
[4] CPT←(ρP)ρ0
[5] CTR←ITER
[6] LIST←(~(DIM[2]ρ0=MAS[CTR; ]))/MAS[CTR; ]
[7] →MLZ×I=ρLIST
[8] LIST←L LIST[ΔLIST]
[9] MLZ: CPT[LIST]+CTR

```

```

[10] ML:CTR+CTR-1
[11] →MLY×10=CTR
[12] →ML×11=(A/MAS{CTR;}=0)
[13] MASTEMP-(~(DIM[2]0-MAS{CTR;})) /MAS{CTR;}
[14] ABSENT-~MASTEMPcLIST
[15] →ML→~1≠v/ABSENT
[16] LIST-LIST,TP-ABSENT/MASTEMP
[17] LIST-LIST[LIST]
[18] TP-TP[TP]
[19] CPT[TP]+CTR
[20] →ML
[21] MLY:D-D+ITER
[22] L+CPT-D
[23] LMAX+1/L
[24] JMAX+L+LMAX
[25] →0×1LB=LMAX
[26] TMAS+QMAS
[27] 'PROCESSING TIMES'
[28] PTEMP
[29] PSUM+1/PTEMP
[30] 'SUM OF PROCESSING TIMES'
[31] PSUM
[32] 'DUE DATES'
[33] D
[34] 'RELEASE TIMES'
[35] R
[36] 'SCHEDULE'
[37] TMAS
[38] 'LOWER BOUND'
[39] LB
[40] 'LATENESS VECTOR'
[41] L
[42] 'MAXIMUM LATENESS'
[43] LMAX
[44] 'JOB WITH MAX LATENESS'
[45] JMAX

```

∇

Once the value of L_{\max} is determined it is compared with LB which is a lower bound on the value of L_{\max} . If $LB = L_{\max}$, then no results are printed out, since the "rule" behaved optimally. If $LB < L_{\max}$, then the schedule and other relevant information are printed out. The printed information is analyzed to determine if the rule is optimal or not.

An example of a run of 5, 20 job, 3 machine problems with loose due dates (TABLE = 135), now follows:

Table = 135 74 30 17
 Counter = 5
 3 DGEN 20

Loose Due Dates
 4 of 5 hit LB

Processing Times

9 10 4 18 14 5 19 8 9 4 6 5 18 14 17 16
 1 11 8 6

Sum of Processing Times

202

Due Dates

71 90 71 59 41 22 43 33 40 83 18 81 81 74 22 27
 50 25 71 39

Release Times

14 15 5 3 8 11 10 10 2 13 9 12 1 3 3 4
 12 8 9 9

Schedule

13 9 15 15 15 15 15 15 15 15 15 15 15 15 15 6
 8 16 15 8 18 18 18 7 5 5 5 5 5 5 5
 5 5 5 7 20 9 5 5 7 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 1 13 8 14 14 14
 13 13 2 2 2 2 2 2 2 2 2 2 2 2 2

 0 13 9 16 16 16 16 18 11 11 11 11 11 6 16 15
 15 18 16 15 16 16 8 8 7 7 7 7 7 7 7
 7 7 7 5 7 20 9 7 4 14 14 14 14 14 13
 1 13 1 13 1 19 14 8 18 3 14 18 13 13 13
 10 10 10 10 13 13 0 0 0 0 0 0 0
 0 0 4 9 9 3 9 18 18 18 18 16 18 11 18 16
 18 16 18 6 18 16 7 7 18 16 8 8 8 8 8
 20 20 20 20 9 5 7 7 4 17 1 1 19 1 19
 14 19 14 19 14 3 13 1 19 14 19 1 13 12 12
 12 12 13 13 0 0 0 0 0 0 0 0 0

Lower Bound

-3

Lateness Vector

-14 -23 -13 -4 -2 -2 -3 -3 -2 -18 -4 -18 -14
 -13 -2 -3 -10 -2 -13 -2

Maximum Lateness

-2

Job with Max Lateness

5

In this instance only one problem is printed out, since 4 of the 5 problems achieved their lower bound.

A lower bound LB on the value of L_{\max} is determined by computing:

$$LB_1 = \max_k \{r_k - 1 + p_k - d_k\} \quad \text{and}$$

$$LB_2 = \max_k \left\{ \max \left(\left\langle \frac{\sum p_k}{m} \right\rangle ; \max_k (r_k - 1 + p_k) \right) - \max_k (d_k) \right\}$$

and letting $LB = \max (LB_1; LB_2)$.

LB_1 is a trivial lower bound which holds if all jobs are processed continuously from the first moment they become available.

LB_2 is obtained by computing minimum length of the schedule and subtracting from this value the maximum due date. Since at least one job

is still being processed at time $\max \left[\left\langle \frac{\sum p_k}{m} \right\rangle ; \max_k (r_k - 1 + p_k) \right]$,

the best that could happen is that the job is the one with maximum due date.

Those schedules that do not achieve their lower bound are analyzed for optimality by not allowing any job to be processed beyond the furthest period such that $L_j < L_{\max}$ generated by the rule, for all j . If no feasible schedule exists under these conditions, then the schedule generated by the heuristic is optimal.

A.1.4 Computational Efficiency of the Algorithm

It was initially suspected that the number of computations required by the algorithm would be of the $O(n^2)$. This was arrived at by noting that at most n comparisons are required in each of the periods

$t, t = 1, \dots, T$. Since $T \geq \frac{\sum_k p_k}{m}$ and p_k is generated randomly between 1 and 20 and $2 \leq m \leq 5$, the number of computations, COMP is approximately,

$$\text{COMP} \sim n T \sim n \frac{\sum_k p_k}{m} \sim \frac{n \cdot 10 \cdot n}{m} \sim O(n^2).$$

However, run times for $m = 2$ yielded the following data:

n	CPU time (sec)
5	0.40
10	1.06
20	2.80
25	3.92
30	4.33
35	5.02
40	6.73
55	9.42
60	11.06
80	15.92

which when plotted (Figure 7) revealed an almost linear relationship between CPU time and n , the number of jobs.

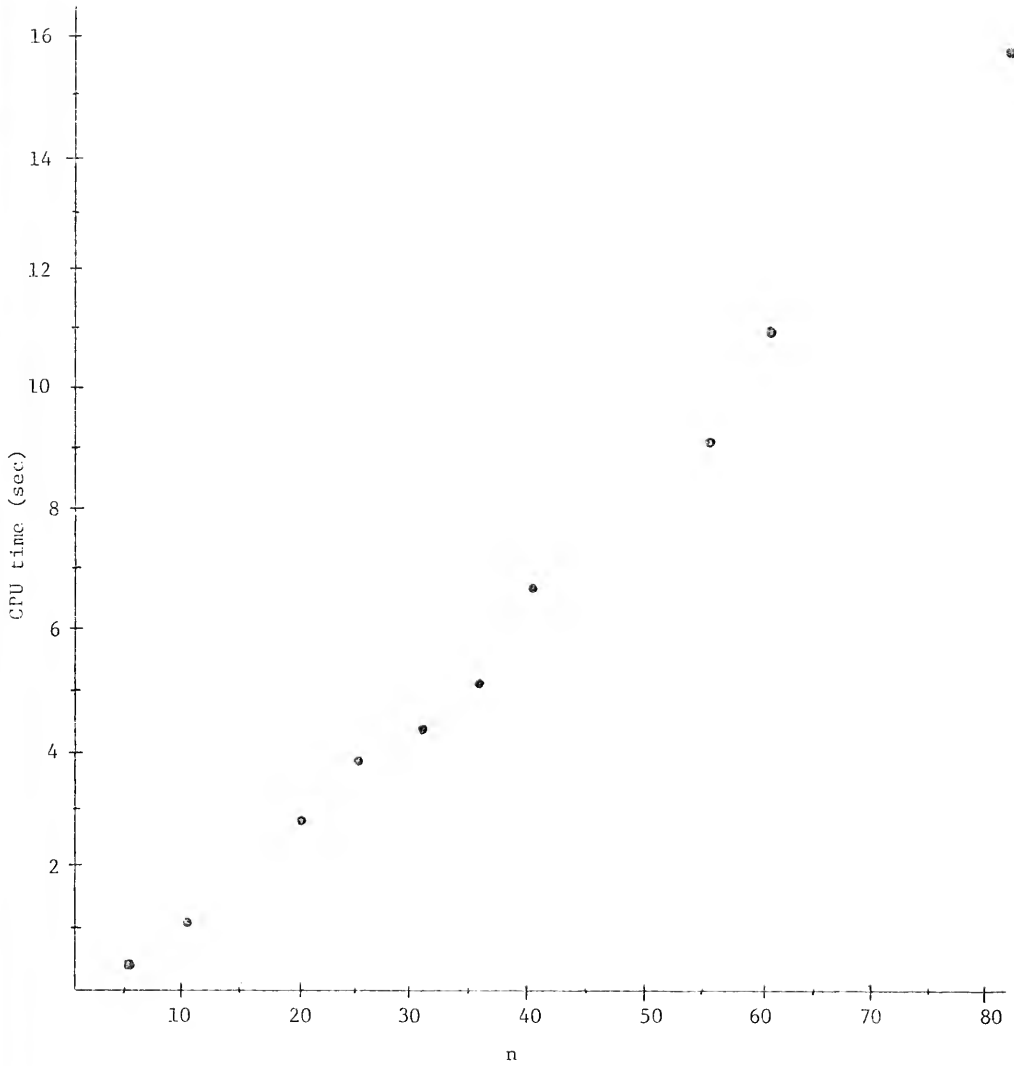


Figure 7: Plot of CPU Time Versus Problem Size

A least squares fit for the data yielded the following functional relationship:

$$\text{CPU (sec)} = -1.50 + 0.21 n$$

where $r = .994$ (Correlation Coefficient)

and $\hat{\sigma} = 1.605$ (Estimate of Standard Deviation).

And so, for all practical purposes, the algorithm can be considered to be of $O(n)$.

A.1.5 Summary

Based on the results of this experiment it may be concluded that the original conjecture still stands as valid. The algorithm was non-optimal in but 2 of 279 problems and never failed to arrive at an optimal solution for "not so tight" due dates (i.e., $\text{TABLE} \geq 10$). Computation times are extremely efficient for $m = 2$ and should decrease for $m > 2$, since T decreases with the number of machines for a given value of n .

Finally, the author wishes to express his gratitude to Mr. Jesús Sáenz for his generous assistance in the coding of the algorithm as well as his role in the author's conversion to an "APL" believer.

APPENDIX 2

REDISCOVERING THE SQUARE WHEEL

A.2.1 Introduction

The multiplicity of technical journals available to practitioners and researchers in the management sciences and related areas serves many purposes. One of the most important concerns the dissemination of solution procedures that are developed for problems of interest. Indeed, the first step in any investigative endeavor centers around a compilation of known results in the area to be considered. A comprehensive survey serves the objective of providing an orientation to the researcher of the area in question which in turn allows him to appraise his conjectures or hypotheses in light of the current "state of the art." The more complete the survey the less risk there is that the researcher's efforts will conclude in a "rediscovery of the wheel," that is a duplication of previous successful efforts.

While a great deal of incentive exists for publishing successful research, very little encouragement is provided for publishing information based on conjectures that resulted incorrect under analysis. This is unfortunate because such negative results are also a part of the "state of the art" in that they also provide insight into the mechanism behind research problems. Since these results are generally known only to those who have personally encountered them, it is the author's

belief that a great deal of time (research \$, etc.) is expended in literally "rediscovering the square wheel," that is, duplicating previously known but undisseminated negative results.

This appendix is directed towards a partial rectification of this situation. In this current research effort, large amounts of time have been dedicated to eventually disproving a series of "heuristics" related to parallel processor scheduling problems. It has been the case, at times, that the development of these counterexamples has required as much effort as the development of proofs for those rules that have been found to be optimal. Moreover, it is debatable which of the two occurrences has provided more insight into the structure of the problems at hand. With this in mind the following "rediscoveries of square wheels" are now presented.

A.2.2 Minimization of L_{\max} with Non-Simultaneous Job Arrivals

The efficiency of the "heuristic" of Section 3.32 for the above problem may raise speculation concerning the possible optimality of the rule for all but certain pathological cases. The following example provides one of the two instances where the rule was not optimal.

$m = 2$	<u>Job (j)</u>	<u>r_j</u>	<u>$P_j (r_j)$</u>	<u>d_j</u>
	1	1	1	1
	2	1	2	4
	3	1	1	2
	4	2	1	10
	5	3	1	4
	6	3	1	4
	7	4	1	4
	8	4	1	4

at $t = 1$, jobs 1, 2, 3 are considered where:

$$p_1(1) - d_1(1) = 1 - 1 = 0 \quad *$$

$$p_2(1) - d_2(1) = 2 - 4 = -2 \quad \text{and jobs 1 and 3 are scheduled.}$$

$$p_3(1) - d_3(1) = 1 - 2 = -1 \quad *$$

at $t = 2$, only jobs 2 and 4 are available for scheduling

at $t = 3$, jobs 2, 5, 6 are considered where

$$p_2(3) - d_2(3) = 1 - 2 = -1 \quad *$$

$$p_5(3) - d_5(3) = 1 - 2 = -1 \quad *$$

$$p_6(3) - d_6(3) = 1 - 2 = -1$$

Since $p_2(3) = p_5(3) = p_6(3) = 1$, the choice is arbitrary,
say, 2 and 5.

at $t = 4$, jobs 6, 7, 8 are considered where

$$p_6(4) - d_6(4) = 1 - 1 = 0 \quad *$$

$$p_7(4) - d_7(4) = 1 - 1 = 0 \quad * \quad \text{and jobs 6 and 7 are scheduled.}$$

$$p_8(4) - d_8(4) = 1 - 1 = 0$$

Finally, at $t = 5$, only job 8 remains with

$$p_8(5) - d_8(5) = 1 - 0 = 1.$$

The schedule generated by the rule is:

	t:	1	2	3	4	5
Machine One		1	2	2	6	8
Machine Two		3	4	5	7	∅

where $L_8 = 1 = L_{\max}$.

However, the following schedule is optimal:

	t:	1	2	3	4	5
Machine One		1	3	5	7	4
Machine Two		2	2	6	8	∅

where $L_{\max} = L_7 = L_8 = 0$.

The failure of the scheduling rule is centered around the trade-off between potential lateness ($p_j(t) - d_j(t)$) and remaining job processing time $p_j(t)$. In this example it is better to schedule job 2 at $t = 1$, even though $p_3(1) - d_3(1) > p_2(1) - d_2(1)$. It appears that $p_2(1) = 2 > 1 = p_3(1)$ dominates at $t = 1$, since when 3 is scheduled at $t = 1$, job 4 is forced into the schedule at $t = 2$ prematurely. If it were the case for all t and for any jobs s and q , that when $p_s(t) - d_s(t) \geq p_q(t) - d_q(t)$ we also had $p_s(t) \geq p_q(t)$, then the rule would be optimal. Clearly no trade-off exists in the F_{\max} problem, since $d_j(t) = 0$ for all k and it was shown in Section 3.31 that the rule is optimal in this instance. Attempts at using a combination of the two priorities have not yielded any positive results.

A.2.3 Precedence Constraints

The lack of results through this proposal for parallel processor problems with precedence constraints is not by design but by default. The inclusion of precedence constraints for identical processors converts the problem into the " m identical resource constrained CPM" problem. A number of promising rules were applied to the F_{\max} and L_{\max} problems for general precedence networks. None,

however, returned optimal results.

The motivation behind looking at this problem is based on the following property:

Property A.1: Any n job m machine problem with job splitting can be converted into an equivalent N job problem where all jobs are identical as follows:

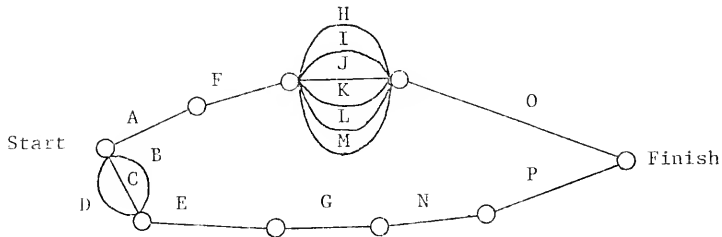
Split job j (for all $j = 1, \dots, n$ such that $p_j \geq 2$), into p_j identical jobs in series, each with processing time $p_j / p_j = 1$. It then follows that any n job problem without precedence constraints and non-identical jobs can be converted into a problem of scheduling $N = \sum_{j=1}^n p_j$ identical jobs on m processors with precedence constraints consisting of n parallel strings. Any n job problem with precedence constraints can be converted into an N job problem with identical jobs on an enlarged but equivalent precedence network.

It then follows that T. C. Hu's [41] algorithm for the F_{\max} problem with identical jobs will solve any F_{\max} problem with the above type of job splitting and precedence constraints in the form of a tree. However, the only algorithm for the F_{\max} problem and general precedence constraints is given by Coffman and Muntz [15] for $m = 2$. The L_{\max} problem is only solved for $m = 1$ by Lawler [46]. Therefore in trying to arrive at rules for more general cases only problem with identical jobs were looked at (i.e., $p_j = 1$ for all j).

A.2.3.1 F_{\max}

Rule A-1: Let $Q(t)$ = the set of all unprocessed jobs at t without predecessors. For any t , schedule all $j \in Q(t)$ so that:

- (i) If $|Q(t)| \leq m$, then process all jobs in $Q(t)$.
- (ii) If $|Q(t)| > m$, then process any m jobs in $Q(t)$ such that if job q is processed and job s is not, then $\ell_q \geq \ell_s$, where ℓ_j denotes the length of the longest path from j to the end of the precedence graph. That the rule will not work for $m \geq 3$ is shown by the following example where $m = 3$.



at $t = 1$, $A, B, C, D \in Q(t)$ where

$$\ell_A = 3$$

$$\ell_B = \ell_C = \ell_D = 4 \quad \therefore \text{Process } B, C, D.$$

at $t = 2$ $A, E \in Q(t)$

at $t = 3$ $F, G \in Q(t)$

at $t = 4$ $H, I, J, K, L, M, N \in Q(t)$ all with length = 1,

so we arbitrarily pick H, I, J

at $t = 5$ $K, L, M, N \in Q(t)$ and pick K, L, M

at $t = 6$ $N, O \in Q(t)$

at $t = 7$ $P \in Q(t)$ and the schedule results in

	t = 1	2	3	4	5	6	7	
Machine One	B	A	F	H	K	N	P	
Machine Two	C	E	G	I	L	O	\emptyset	with $F_{\max} = 7.$
Machine Three	D	\emptyset	\emptyset	J	M	\emptyset	\emptyset	

However, an optimal schedule is given by:

	t = 1	2	3	4	5	6	
Machine One	A	D	E	J	L	O	
Machine Two	B	F	H	K	M	P	with $F_{\max} = 6.$
Machine Three	C	\emptyset	I	G	N	\emptyset	

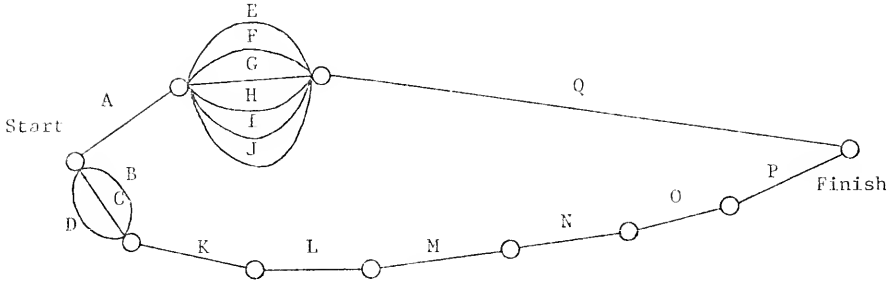
Rule A-2:

For any t , schedule all $j \in Q(t)$ so that:

- (i) If $|Q(t)| \leq m$, then process all jobs in $Q(t)$.
- (ii) If $|Q(t)| > m$, and q is processed while s is not, then

$$|S_q| \geq |S_s| \quad \text{where}$$

$|S_j|$ = the number of successors to j (equivalent to the sum of remaining processing times for all jobs dependent on j 's completion since $p_j = 1$ for all j). Again, this rule will not work for $m \geq 3$.



at $t = 1$, $A, B, C, D, \in Q(t)$ where

$$|S_A| = 7$$

$$|S_B| = |S_C| = |S_D| = 6 \quad \therefore \text{Process } A, B, C$$

at $t = 2$ $D, E, F, G, H, I, J \in Q(t)$ where

$$|S_D| = 6$$

$$|S_E| = |S_F| = |S_G| = |S_H| = |S_I| = |S_J| = 1 \quad \therefore \text{Process } D, E, F$$

at $t = 3$ $G, H, I, J, K \in Q(t)$ where

$$|S_G| = |S_H| = |S_I| = |S_J| = 1$$

$$|S_K| = 5 \quad \therefore \text{Process } K, G, H$$

at $t = 4$ $I, J, L \in Q(t)$

at $t = 5$ $M, Q \in Q(t)$

at $t = 6$ $N \in Q(t)$

at $t = 7$ $O \in Q(t)$

at $t = 8$ $P \in Q(t)$ and the schedule is:

	t = 1	2	3	4	5	6	7	8
Machine One	A	D	K	I	M	N	O	P
Machine Two	B	E	G	J	Q	\emptyset	\emptyset	\emptyset where $F_{\max} = 8$
Machine Three	C	F	H	L	\emptyset	\emptyset	\emptyset	\emptyset

However, an optimal schedule is given by:

	t = 1	2	3	4	5	6	7
Machine One	B	A	E	C	I	Q	P
Machine Two	C	K	F	H	J	O	\emptyset
Machine Three	D	\emptyset	L	M	N	\emptyset	\emptyset

Lawler [46] was able to make use of the notion of scheduling jobs from last to first in arriving at his algorithm for the one machine case. However, both Rule A-1 and Rule A-2 fail if they are applied from last to first. The preceding examples can be used to show this by simply switching the start and finish nodes on both graphs.

A.2.3.1 L_{\max}

Rule A-3: Sequence the jobs from last to first, always choosing next from among the jobs currently available (i.e., those without successors), m jobs with the latest possible due dates. (This rule is a direct extension of Lawler's optimal procedure for the n job one machine problem [46].

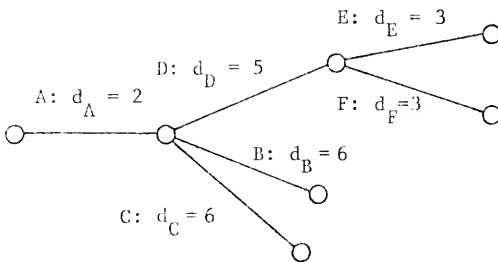
The rule will not work since if $d_j = 0$ for all j , then the L_{\max} problem is equivalent to minimizing F_{\max} . The rule would translate to scheduling from last to first any m jobs currently without

successor. However, T. C. Hu [41] has shown that even for a tree precedence structure the choice is not arbitrary.

Rule A-4: Sequence the jobs from last to first, always choosing from the jobs currently available, job q over job s if

$$l_q^{-1} \geq l_s^{-1},$$

where l_j^{-1} denotes the length of the longest path from j to the beginning of the graph.



where $m = 2$

at T jobs E, F, B, C, are available where

$$l_E^{-1} = 2 = l_F^{-1} \quad \text{and} \quad l_B^{-1} = 1 = l_C^{-1} \quad \text{and so E and F are chosen}$$

at $T-1$ jobs E, C, D, are available where

$$l_B^{-1} = l_C^{-1} = l_D^{-1} = 1 \quad \text{and so B and C are chosen}$$

at $T-2$ job D is available

at $T-3$ job A is available. The resulting schedule is

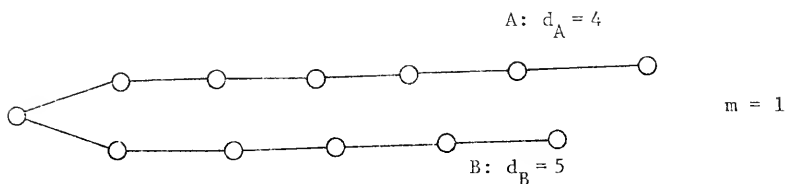
	t =	1	2	3	4	= T	
Machine One		A	D	B	E		with $L_{\max} = L_E = L_F = 1$
Machine Two		\emptyset	\emptyset	C	F		

However, an optimal schedule is given by:

	t =	1	2	3	4	= T	
Machine One		A	D	E	B		with $L_{\max} = L_E = L_F = 0$.
Machine Two		\emptyset	\emptyset	F	C		

Rule A-5: Sequence the jobs from last to first, always choosing from the jobs currently available, job q over job s if:

$$\ell_q^{-1} - d_q \geq \ell_s^{-1} - d_s.$$



at T: choose A over B since

$$\ell_S^{-1} - d_A = 5 - 4 = 1 > -1 = 4 - 5 = \ell_B^{-1} - d_B.$$

However, this contradicts Lawler's optimal rule for $m = 1$ which picks B over A.

While certainly the previous examples are not general proofs, they lend credence to the author's conjecture that no one pass rule exists for either the F_{\max} or L_{\max} problem with precedence constraints. The rules presented represent both the logical extension of known optimal results for more special cases as well as the extension of the results obtained in this dissertation without precedence constraints.

APPENDIX 3

THE OPERATION OF THE SEABOARD COAST LINE RAILROAD (SCL)

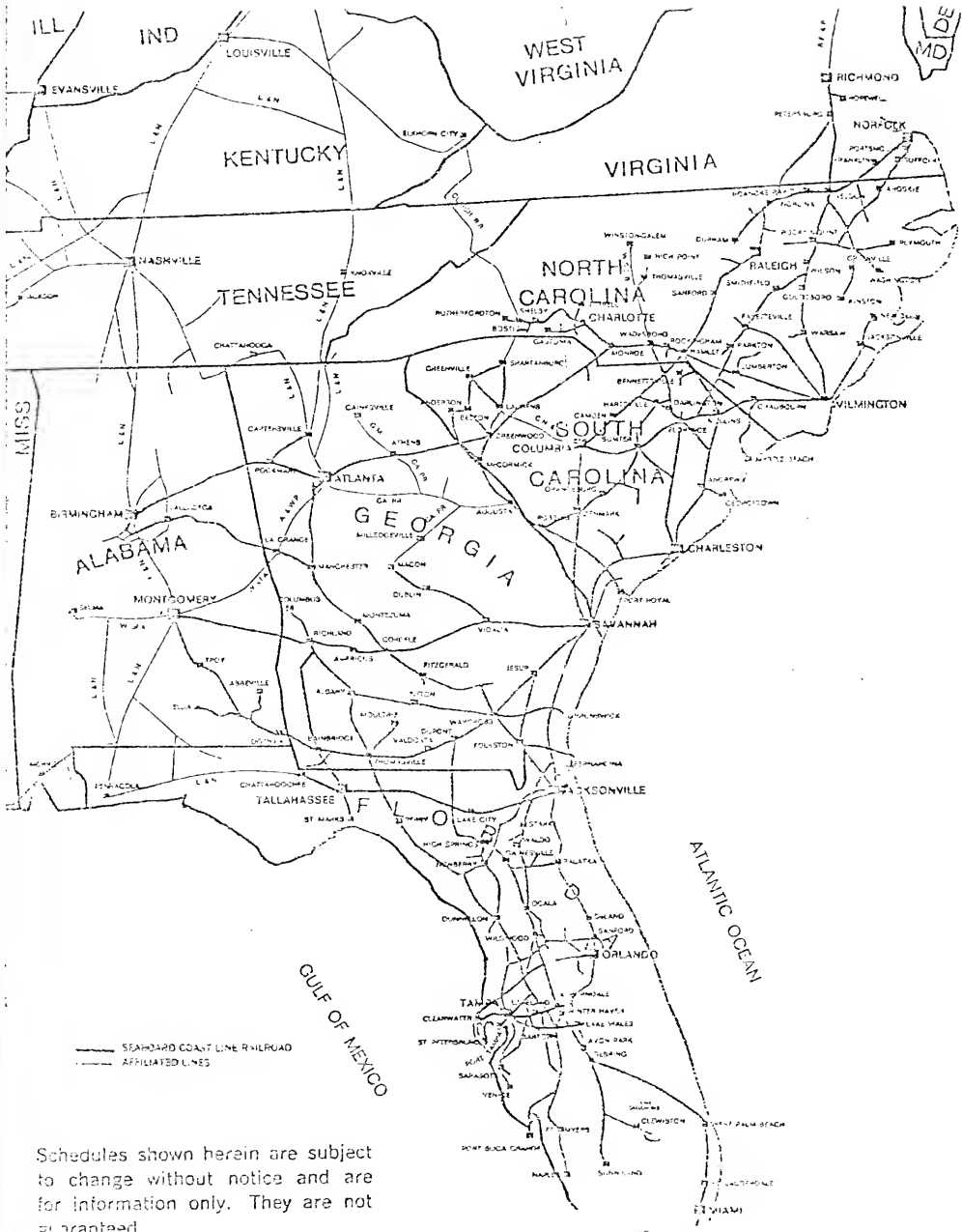
A.3.1 Introduction

Although passenger services are the most widely publicized examples of inefficient railway operation, freight services have a far greater impact on the nation's economy. Typical problems of rail freight transport include low equipment utilization, long and variable origin/destination transit time, congested and costly terminals, expensive and loosely controlled local movements, and ever-present high labor costs.

In an effort to develop insight into the particular problems encountered by the railway industry a visit was paid to the main headquarters of the Seaboard Coast Line Railroad in Jacksonville, Florida. The following description of operations is based on interviews with Mr. J. P. Roberts, Manager of Sales and Service, and Mr. T. J. Waddington, Director of Freight Scheduling, both of whose willing cooperation is hereby duly acknowledged.

A.3.2 The Freight Transportation Network

The freight services of the SCL encompass an area bordered on the northeast by Washington, D.C. (Potomac Yard), on the southeast by Miami, Florida, extending northwest to Chicago, Illinois, west to St.



Schedules shown herein are subject to change without notice and are for information only. They are not guaranteed.

Figure 8: Freight Railway Network

Louis, Missouri and Memphis, Tennessee and southwest to New Orleans, Louisiana. A map of the freight transportation network is found in Figure 8. The thick arcs denote the "parent" line and the narrow ones the annexed lines acquired through merger with the Louisville and Nashville railway. The whole network constitutes what is known as the set of "family" lines. The lines or routes are divided into three categories:

a. Run Lines: These are lines that pass through major switchyard centers where trains are broken up or service is added depending on the circumstances (e.g., Washington, D.C. to Jacksonville, Florida). Trains along these routes usually travel directly between the major switchyards along the route with few, if any, stops at smaller stations along the way.

b. Run-Through Lines: These are lines that run partially through the SCL network and continue through the service of another railway. Resources along this line are pooled with other participating railways (e.g., Jacksonville to New Orleans segment). These lines also pass through major switchyards.

c. Local Lines: These are lines that serve small facilities. Trains running these lines stop to either pick up or drop off cars at local sidings. These lines usually originate and terminate at major switchyards. The formulation of Chapter 4 provides a description of railway operations along a local line.

A.3.3 The Capacity of a Train

A prime concern of much of the development of this study has been the determination of the minimum train capacity required to deliver all freight to its destination. The capacity of a train is mainly restricted by one of the following factors:

a. The engine size: In order for a certain engine to pull "x" boxcars on a given route at an acceptable speed, it is necessary that "x" not exceed a certain pre-determined value.

b. The "capacity" of a passing track: SCL was originally a "two-track" railway--that is, it was generally the case that most routes were connected by two tracks, each track strictly uni-directional in use. At present SCL is a "one-track" railway--routes consist of one track where trains run in opposite directions. At different locations along a route, there are sidings or passing tracks (originally part of the second track), where a train may wait while another passes in the opposite direction. While a train(s) in one direction may have unlimited capacity, the train(s) in the opposite direction cannot be longer than the smallest passing track along the route.

c. An additional restriction imposed externally is that some trains may not leave their origins until they are at least "y" cars long.

A.3.4 The Development of Freight Schedules

Schedules are based primarily on the needs for transporting freight. The idea is to provide enough capacity throughout the

system to move everything, every day. Arrival and departure times are determined so as to grab as much freight as possible and then the trains are run as fast as possible from station to station. Since schedules are set to meet peak demands, the situation of having to choose which of two cars to add to a train does not occur often. However, a critical problem, from management's viewpoint, is the existence of a shortage of boxcars. They point out that the demand for boxcars exceeds the available supply; that if they had more boxcars, they could move more freight. However, it is not clear that efficient use is made of the boxcars on hand since an average per day per boxcar utilization rate of 0.7 reveals that on the average 30% of all boxcars travel empty all the time.

Management acknowledges that the delivery of some commodities is of greater urgency than others, and hence the concept of a "due date" does exist. However, current operations do not take this into account. The goal is to get everything to its destination as soon as possible whether it has to be there or not. The possible use of any slack in the system is completely negated by this approach.

The complexity of moving over 50 different commodities is simplified by giving equal priority to all boxcars, thereby paying little real attention to the particular commodity being moved.

A.3.5 Summary

In summary, a scheduling problem as concerns "what" to put on or drop off does not often arise in practice because capacities

are geared to meet peak demands. The scheduler views all the freight as a single commodity to be moved as quickly as possible. The scheduling function is, at present, essentially one of creating new routes or eliminating stops as demand fluctuates. Timetables are set in accordance with demand.

It is clear that the problems considered in this report are a simplification of current operations. However, inefficiencies in current scheduling operations of SCL bring to light the need for perhaps considering such factors as "due dates" in order to more effectively utilize current equipment. Certainly scheduling problems do exist and the ramifications of more efficient scheduling methods would most likely be in more efficient equipment utilization.

BIBLIOGRAPHY

- [1] Applications Digest, Association of American Railroads, Washington, D.C.
- [2] Arthanari, T.S. and Ramamurthy, K. G., "A Branch and Bound Algorithm for Sequencing n Jobs on m Parallel Processors," Opsearch, 7, 1970, 3, September, pp. 147-156.
- [3] Ashour, S., Sequencing Theory, Springer Verlag (1972).
- [4] Baker, K.R., "Procedures for Sequencing Tasks with One Resource Type," Int. Journal of Prod. Research, 11 (1973), 125-138.
- [5] Baker, Kenneth R. and Martin, James B., "An Experimental Comparison of Solution Algorithms for the Single-Machine Tardiness Problem," Technical Report No. 72-76, Department of Industrial and Operations Engineering, University of Michigan (Presented at the Symposium on Scheduling, Raleigh, N.C., May, 1972).
- [6] Baker, K. R. and Merten, A. G., "Scheduling with Parallel Processors and Linear Delay Costs," Nav. Res. Log. Quart., Vol. 20, 4, 1973.
- [7] Baker, K. R. and Su, Z., "Sequencing with Due-Dates and Early Start Times to Minimize Maximum Tardiness," Manuscript, June, 1972.
- [8] Bellman, R., "Some Mathematical Aspects of Scheduling Theory," J. Soc. Ind. and Appl. Math., 4, No. 3, September, 1956.
- [9] Bellmore, M. and Nemhauser, G. L., "The Traveling Salesman Problem, A Survey," Operations Research, 16, July-August, 1968.
- [10] Bennington, G. E. and McGinnis, L. G., "A Critique of Project Planning Under Constrained Resources," Symposium on the Theory of Scheduling and Its Applications, Springer Verlag (1973).
- [11] Billheimer, J. W., "Network Design with Fixed and Variable Cost Elements," Transportation Science (7) (1973), 47-74.
- [12] Bratley, P., Florian, M. and Robillard, P., "Scheduling with Earliest Start and Due Date Constraints," Nav. Res. Log. Quart., Vol. 18, No. 4, December, 1971.
- [13] Bratley, P., Florian, M. and Robillard, P., "Scheduling with Earliest Start and Due Date Constraints on Multiple Machines," Technical Report #111, Université de Montréal, Département d'Informatique, January, 1973.

- [14] Bratley, P., Florian, M. and Robillard, P., "On Sequencing with Earliest Starts and Due Dates with Application to Computing Bounds for the (u/m/G/F_{max}) Problem," Nav. Res. Log. Quart. 20 (1973) 57-67.
- [15] Coffman, E. and Muntz, R., "Optimal Pre-emptive Scheduling on Two-Processor Systems," IEEE Transactions Comput., 18, 1014.
- [16] Conway, R. W., Maxwell, W. L. and Miller, L. W., Theory of Scheduling, Addison-Wesley, August, 1967.
- [17] Dantzig, G. B., and Fulkerson, D. R., "Minimizing the Number of Tankers to Meet a Fixed Schedule," Nav. Res. Log. Quart., 1, 217-222 (1954).
- [18] Davis, Edward W., "Resource Allocation in Project Network Models-- A Survey," AIIE Transactions, Vol. 5, No. 4, December, 1973.
- [19] Dessouky, M. E. and Margenthaler, C. R., "The One Machine Sequencing Problem with Early Starts and Due Dates," AIIE Transactions, 4,3, September, 1972.
- [20] Dorsey, R. C., Hodgson, T. J. and Ratliff, H. D., "A Network Approach to a Multi-Facility, Multi-Production Scheduling Problem without Backordering," Research Report No. 73-5, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, January, 1973.
- [21] _____, "A Network Approach to a Multi-Facility, Multi-product Production Scheduling Problem with Backordering," Research Report No. 73-6, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, January, 1973.
- [22] _____, "A Multiple Facility, Multiple Product Production Scheduling Problem with Overtime," Research Report No. 73-7, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, January, 1973.
- [23] Eastman, W. L., Even, S. and Isaacs, I. M., "Bounds for Optimal Scheduling of Jobs," Management Science, Vol. 11, 2, December (1964).
- [24] Elmaghraby, S. E. (ed.), Symposium on the Theory of Scheduling and Its Applications, Springer Verlag (1973).
- [25] _____, "The One-Machine Sequencing Problem with Delay Costs," Journal of Industrial Engineering, Vol. 19, No. 2, February, 1968.

- [26] Elmaghraby, S. E. (ed.), "The Machine Sequencing Problem-- Review and Extensions," Nav. Res. Log. Quart., Vol. 15, June, 1968.
- [27] Elmaghraby, S. E. and Park, S. H., "On the Scheduling of Jobs on a Number of Identical Machines," AIIE Transactions, Vol. 6, No. 1, March, 1974.
- [28] Emmons, H., "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness," Operations Research, Vol. 17, No. 4, July, 1969.
- [29] Fisher, M., "Optimal Solution of Scheduling Problems Using Lagrange Multipliers," Report 7210, Center for Math. Studies, University of Chicago, March (1972).
- [30] Ford, L. R., Jr. and Fulkerson, D. R., Flows in Networks, Princeton University Press, Princeton, N.J., 1962.
- [31] Gelders, L., and Kleindorfer, P. R., "Coordinating Aggregate and Detained Scheduling Decisions in the One Machine Job Shop: Part I Theory," Operations Research, Vol. 22, No. 1, Jan. - Feb., 1974.
- [32] Gillman, L. and Rose, A., APL/360: An Interactive Approach, John Wiley & Sons, 1970.
- [33] Glassey, C. R., "Minimum Changeover Scheduling of Several Products on One Machine," Operations Research, Vol. 16, No. 2, March-April, 1968.
- [34] Cotterer, M. H., "Scheduling with Deadlines, Priorities and Non-Linear Loss Functions," Research Report, Dept. of Industrial Engineering, Georgia Institute of Technology, Atlanta, Georgia.
- [35] Gupta, J. and Maykut, A., "Scheduling Jobs on Parallel Processors With Dynamic Programming," Decision Sciences, November, 1973.
- [36] Gupta, J. and Walvekar, A., "Sequencing n Jobs on m Parallel Processors," Opsearch, 6, 1969, 4, December, pp. 295-298.
- [37] Held, M. and Karp, R. M., "A Dynamic Programming Approach to Sequencing Problems," J. SIAM, Vol. 16, No. 2, March (1962).
- [38] Herroelen, Willy S., "Resource-Constrained Project Scheduling-- the State of the Art," Operational Research Quarterly, Vol. 23,
- [39] Horn, W. A., "Single-Machine Job Sequencing with Treelike Procedure Ordering & Linear Delay Penalties," J. SIAM, 23 (1972), 189-202.

- [40] Horn, W. A., "Minimizing Average Flow Time with Parallel Machines," Technical Note, Operations Research, Vol. 21, No. 3, May-June (1973).
- [41] Hu, T. C., "Parallel Sequencing and Assembly Line Problems," Operations Research, 9, No. 6, November, 1961.
- [42] Jackson, J. R., "Scheduling a Production Line to Minimize Maximum Tardiness," Research Report 43, Management Sciences Research Project, UCLA, January, 1955.
- [43] Jacobsen, S., "On Marginal Allocation in Single Constraint Min-Max Problems," Management Science, July, 1971.
- [44] Karp, R. M., "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations, R. E. Miller, J. W. Thatcher and J. D. Bohlinger (eds.), Plenum Press, New York, 1972, pp. 85-103.
- [45] Lawler, E. L., "On Scheduling Problems with Deferral Costs," Management Science, 11, No. 2, November, 1964.
- [46] _____, "Optimal Sequencing of a Single Machine Subject to Precedence Constraints," Management Science, Vol. 19, No. 5, January, 1973.
- [47] Lawler, E. L. and Moore, J. M., "A Functional Equation and Its Application to Resource Allocation and Sequencing Problems," Management Science, Vol. 16, No. 1, September, 1969.
- [48] Maxwell, W. L., "The Scheduling of Single Machine Systems-- A Review," The Int. Journal of Prod. Research, Vol. 3, No. 3, 1964.
- [49] McMahon, G. and Florian, M., "On Scheduling With Ready Time and Due Dates to Minimize Maximum Lateness," Report #159, Université de Montreal, Department d'Informatique, January, 1974.
- [50] McNaughton, R., "Scheduling with Deadline and Loss Functions," Management Science, 6, No. 1, October, 1959.
- [51] McWhite, P. B. and Ratliff, H. D., "Allocation of Mine Countermeasure Resources When Shipping Flow is Fixed," Research Report No. 73-12, Industrial and Systems Engineering Department, University of Florida, Gainesville, Florida, January, 1973.
- [52] Mehl, Roger H., "A Call for Practical Rail Transportation System Models," Presented at ORSA/TIMS Joint National Meeting, Boston, Mass., April 22-24, 1974.

- [53] Mellor, P., "A Review of Job Shop Scheduling," Operational Research Quarterly, 17 (1966), 161-171.
- [54] Miller, Louis W., "Branch and Bound and Heuristic Approaches to a Sequencing Problem with Team Requirements," AIIE Transactions, Vol. 6, No. 3, Sept. 1974.
- [55] Moore, J. Michael, "An n Job-One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs," Management Science, Vol. 15, No. 1, September, 1968.
- [56] Muntz, R. R., Scheduling of Computations on Multiprocessor Systems: The Preemptive Assignment Discipline, Ph.D. Dissertation, Princeton University, Princeton, N. J., April, 1969.
- [57] Petersen, Clifford C., "Solving Scheduling Problems Through Re-ordering Operations," Symposium on Scheduling, Raleigh, N. C., May 15-17, 1972.
- [58] Porteus, E. L. and Yormak, J. S., "More on Min-Max Allocation," Management Science, Vol. 18, No. 9, May, 1972.
- [59] Pounds, W. F., "The Scheduling Environment" in Industrial Scheduling, J. F. Muth and G. L. Thompson (eds.), Prentice-Hall, New York, 1963.
- [60] Railroad Research Bulletin Developmental Issue, Railroad Research Information Service (1973), U. S. Department of Transportation.
- [61] Rinnooy-Kan, A. H. G., "The Machine Scheduling Problem," Report BW 27/73, Mathematisch Centrum, Amsterdam, 1973.
- [62] Rinnooy-Kan, A. H. G., Lagewag, B. J. and Lenstra, J. K., Report BW 33/74, Mathematisch Centrum, Amsterdam, April, 1974.
- [63] Roberts, S. D. and Moodie, C. L., "Experiments with Priority Dispatching Rules in a Parallel Processor Shop," Int. Journal of Prod. Research, 6 (1968), 4, pp. 303-312.
- [64] Robillard, P., Private Communication, Oct. 1973
- [65] Root, J. G., "Scheduling with Deadlines and Loss Functions on k Parallel Machines," Management Science, 11, No. 3, January, 1965.
- [66] Rothkopf, Michael H., "Scheduling Independent Tasks on Parallel Processors," Management Science, Vol. 12, No. 6, January, 1966, pp. 437-447.
- [67] Salzborn, F. J. M., "Minimum Fleetsize for a Suburban Railway System," Transportation Science, 4 (1970), 383-402.

- [68] Schild, A. and Fredman, I. J., "Scheduling Tasks with Linear Loss Functions," Management Science, 7, No. 3, April, 1961.
- [69] _____, "Scheduling Tasks with Deadlines and Non-Linear Loss Functions," Management Science, 9, No. 1, October, 1962.
- [70] Schrage, L. and Miller, L. W., "The Queue M/G/1 with Shortest Remaining Processing Time Discipline," Operations Research, Vol. 14, No. 4, July-August, 1966.
- [71] Shwimer, J., "On the N-Job, One Machine Sequence Independent Scheduling Problem with Tardiness Penalties: A Branch and Bound Solution," Management Science, Vol. 18, No. 6, February, 1972, pp. 301-313.
- [72] Sidney, Jeffrey B., "One Machine Sequencing with Precedence Relations and Deferral Costs--Parts I, II," Working Papers, Nos. 124, 125, Faculty of Commerce and Business Administration, University of British Columbia, April, 1972.
- [73] _____, "An Extension of Moore's Due Date Algorithm," Working Paper No. 126, Faculty of Commerce and Business Administration, University of British Columbia, April, 1972.
- [74] Smith, W. E., "Various Optimizers for Single State Production," Nav. Res. Log. Quart., 3, No. 1, March, 1956.
- [75] Spinner, Allen H., "Sequencing Theory--Development to Date," Nav. Res. Log. Quart., 15, 319-330, June, 1963.
- [76] Srinivasan, V., "A Hybrid Algorithm for the One Machine Sequencing Problem to Minimize Total Tardiness," Nav. Res. Log. Quart., Vol. 18, No. 3, September, 1971, pp. 317-327.
- [77] Walter, J. M., "Scheduling Parallel Processors with Job Precedence Costs Using a Weighted Criterion," M. S. Thesis, Purdue University, August, 1969.
- [78] Wilkerson, L. J. and Irwin, J. D., "An Improved Method for Scheduling Independent Tasks," AIIE Transactions, Vol. 3, No. 3, September, 1971.
- [79] Zaloom, Victor A., "Research Abstract on a Feasibility Approach to Optimal Schedules," Symposium on the Theory of Scheduling and Its Applications, Springer Verlag, 1973.

BIOGRAPHICAL SKETCH

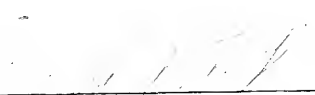
Louis Anthony Martín-Vega was born on December 14, 1947 in New York, New York. He received his elementary and secondary education in New York, Florida and Puerto Rico graduating from Franklin D. Roosevelt High School in Guánica, Puerto Rico, as president and valedictorian of the class of 1964. He attended the University of Puerto Rico at Mayagüez from 1964 to 1968, receiving a Bachelor's degree in Industrial Engineering, magna cum laude, in December, 1968. Upon graduation he was employed as an Assistant Systems Analyst by the Banco Crédito y Ahorro Information Systems Center in Hato Rey, Puerto Rico. In September, 1969 he was awarded a scholarship by the Economic Development Administration of Puerto Rico to pursue graduate studies at New York University. He left graduate school in May, 1970 to serve as Assistant Director of a nine-month Traffic Safety Study for the Department of Public Works in San Juan, Puerto Rico. He returned to New York University in February, 1971, completing a Master of Science degree in Operations Research in June of the same year.

In June, 1971, Louis Martín-Vega entered the graduate program in Industrial and Systems Engineering at the University of Florida. He received a Master of Engineering degree from the University of Florida in March, 1973.

Louis Martín-Vega is a registered Professional Engineer and is a member of the Tau Beta Phi and Alpha Pi Mu honorary engineering societies as well as the American Institute of Industrial Engineers,

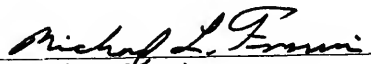
The Institute of Management Science, Operations Research Society of America and College of Engineers and Architects of Puerto Rico professional societies. He is married to the former Emed Magali Flores of Guánica, Puerto Rico, and is the proud father of one son, Louisito and one daughter, Mónica.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



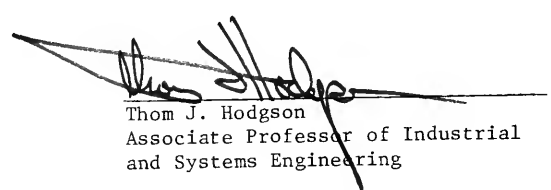
H. Donald Ratliff, Chairman
Associate Professor of Industrial
and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



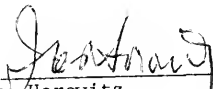
Richard L. Francis
Professor of Industrial and
Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Thom J. Hodgson
Associate Professor of Industrial
and Systems Engineering

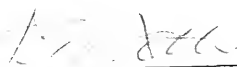
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Ira Horowitz
Professor of Management

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate Council, and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

March, 1975



Dean, College of Engineering

Dean, Graduate School



UNIVERSITY OF FLORIDA



3 1282 08553 3163