AN ABSTRACT OF THE THESIS OF

GEORGE KWOK WING NG for the degree of Master of Science in Industrial Engineering presented on December 18, 1980

Title: MINIMUM COST SCHEDULING OF RESOURCE CONSTRAINED JOBS ON PARALLEL MACHINES UNDER CONTROL OF INTERCHANGEABLE PROCESSORS

Abstract approved: Redacted for Privacy

Dr. Michael S. Inoue

A special case of a parallel multiprocessor scheduling (MP) problem is investigated. A set of jobs with a known process time and a resource requirement is scheduled on machines controlled by processors, and the total changeover cost between jobs is to be minimized. Each processor may control up to two machines and requires a unit of a type of resource. A job may be processed by the machine provided that the processor is equipped with the appropriate type of resource to handle the job. The changeover cost is the sum of the job grade switching cost and of the resource brand switching cost.

Three heuristic algorithms are developed to solve the MP problem. The first algorithm uses the "minimum cost rule" applied to the machine with the shortest current makespan with an adaptation of the "longest processing time" algorithm with the consideration of resource allocation. The second algorithm includes a planning horizon and assigns jobs to machines based upon the least changeover cost until the planning horizon is exceeded for the machine. The third algorithm is based upon a generalized formulation of the traveling salesman problem with more than one salesman. It is a bin-packing branch-and-bound algorithm using the first-fit-decreasing method to minimize the makespan.

FORTRAN programs are developed and used to process actual industrial data from an aluminum reduction plant. With 13 to 26 jobs of 8 to 16 types, 6 to 8 resource types, 3 processors and 6 machines, the savings in total changeover cost using the best algorithm ranged from 14% to 51% of the cost resulting from the manual scheduling that was actually used. In dollars, the 51% reduction corresponded to about $23,000 for that one schedule.

Minimum Cost Scheduling of Resource Constrained Jobs on Parallel Machines Under Control of Interchangeable Processors

by

George Kwok Wing Ng

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed December 18, 1980

Commencement June 1981

APPROVED:

Redacted for Privacy

Professor of General and Industrial Engineering
in charge of major

Redacted for Privacy

Head of Department of General and Industrial Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented December 18, 1980

Typed by Mary Ann (Sadie) Airth for George Kwok Wing Ng

ACKNOWLEDGMENTS

I am deeply indebted to Dr. Michael S. Inoue for his most helpful advice, guidance, and encouragement throughout my graduate studies.

I would like to express my sincere appreciation and thanks to Dr. James L. Riggs for his advice during my graduate work.

I would also like to thank Drs. Patrick E. Connor, Paul Cull, Carroll W. DeKock and Ken Funk for serving on my committee.

Special appreciation is expressed to Mr. Dave Brent who introduced me to this research problem. Appreciation is also extended to Mr. Don E. Cullen for his personal encouragement and advice when I was an industrial engineer trainee at the Reynolds Metals Company.

The computation and data analysis were made possible by a grant from the Oregon State University Computer Center and the Reynolds Metals Company.

Finally, the author expresses his thanks to his best friend Wai-Ming for her encouragement during the course of this study.

This thesis is dedicated to my mother, brothers and sisters with love.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

LIST OF TABLES

MINIMUM COST SCHEDULING OF RESOURCE CONSTRAINED JOBS ON PARALLEL MACHINES UNDER CONTROL OF INTERCHANGEABLE PROCESSORS

CHAPTER I

INTRODUCTION

Effective management of all resources is a key concern in today's industry. Productivity depends upon the ratio of the values of the output resources over the values of the input resources. Resources include not only materials and energy, but also the equipment and personnel for processing both these physical entities as well as as information.

An effective usage of the processing system means that jobs must be scheduled in such a way as to minimize the total operating cost, maximize the throughput, and provide a reasonable makespan. Above all, such a schedule needs to be flexible, dynamically alterable, and practical. This scheduling problem is further complicated by the fact that modern processing systems include subsystems that must themselves be scheduled effectively. A typical configuration is a system where jobs are processed by several machines sharing the use of a more expensive common processor. Figure 1-1 illustrates such a system.

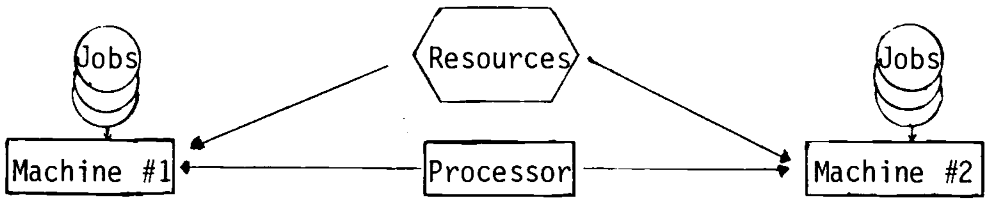


Figure 1-1: A two-machine one processor system.

1.1 Multi-Processor Scheduling

The central problem discussed in this thesis is a special case of multiple processor scheduling, a problem we shall refer to as the "MP" model. More specifically, it involves the effective scheduling of resource constrained jobs on parallel machines under the control of interchangeable processors.

Such problems occur quite frequently in both industrial and social situations. Jobs,machines, and processors can stand for (1) patients, medical equipment, and doctors in a hospital, (2) students, classrooms, and teachers in a school, (3) cargos, ships, and cranes in a port, or (4) jobs, computer terminals, and main frame computers in a multi-processor time-sharing computer network. In each case, jobs require a machine, available time on the processor, and other resources necessary for its processing. A typical resource might be a tool mold, computer memory, an I/O device, etc.

Surprisingly few studies have investigated the problem of analytically finding the minimum cost schedule for multiple machines or processors (Chapter III). None, to the best of the author's knowledge, has addressed specifically the problem of minimizing the changeover cost of resource constrained jobs handled by processor controlled machines.

The problem becomes even more complicated in practice. To be an effective tool to be used in industry, it is not sufficient to minimize the total cost of changeover jobs in machines. The scheduling should permit the consideration of jobs currently in the machines, allow for a reasonably balanced makespan for all machines and processors, and permit as many jobs to be completed by the due dates as possible. In addition, it must be flexible enough so that the management could use it to reschedule the system as new jobs are added, equipment fails, orders are cancelled, or resource shortage occurs.

The purpose of this thesis is to investigate existing algorithms and propose methods that can be effectively used in industry. Current data from an aluminum reduction plant operation are used to validate the effectiveness of the proposed algorithms by computing the labor saving cost resulting from the application of these algorithms.

The term "algorithm" is used loosely to mean any computational method that can reasonably satisfy Knuth's (1975, pp. 4-9) five features of (1) finiteness, (2) definiteness, (3) input, (4) output, and (5) effectiveness.

1.2 Minimum Changeover Cost

The problem of minimizing changeover cost or set-up and tear-down time in sequencing jobs on machines under resource constraints arises quite frequently in various types of industrial operations. Several examples may be cited.

Consider, for instance, a manufacturer of "31-flavor" ice cream mixes with his several ice cream machines. Orders for delivery with specified quantities for each flavor are received. It is then desirable to have a production schedule that will minimize the number of changeovers from one flavor to another while meeting the due date commitments.

Another example may be a printing shop with rotary presses (processors) which must mount different cylinders (machines) to print several different magazines and newsprints. To find a feasible and optimal production schedule to minimize the number of cylinder mounting and color changes may be important. A change of color from red to black may be easier than a change from black to red.

A third example is from the tire industry. Operators (processors) operate general purpose machines, called "cavities" (machines) and a set of transferrable parts called "molds" (resources). To design a production schedule that minimizes the number of set-ups while satisfying the mold availability is important because molds are expensive and cavities should not be kept idle. A schedule that minimizes the amount of resources used, equipment idle time, and the total set-up (or grade change) time means additional production and profit to the company.

Roll in and roll out of jobs on a computer system, heating and cooling of kilns used in ceramic production, and transporting fertilizers and weed killers in the same tank-truck are examples of other changeover costs.

In this thesis, the terms "sequence" and "schedule" are not used synonymously. A sequence is defined as a feasible ordering of a set of jobs to be processed through the machines. A schedule emphasizes the specification of time when the sequenced jobs are started and ended (Elmaghraby, 1968). Job ordering, or sequencing is a binary relationship that is transitive (if $i \prec j$ and $j \prec k$, then $i \prec k$), nonreflexive ($i \not\prec i$, or no job precedes itself), and antisymmetric (if $i \prec j$ then $j \not\prec i$). The changeover cost is associated with each transition.

1.3 Organization of the Thesis

In Chapter II the MP model is formulated and criteria and constraints are described. The characteristics of the problem and the difficulty of its solution are discussed.

Chapter III surveys the past work and investigates methods of solutions to the MP problem. Integer programming, dynamic programming, branch-and-bound, combinatorial analysis, and heuristic approaches are discussed.

Chapter IV prepares the theoretical background necessary to develop the two heuristic algorithms. Numerical examples are included. The "Next Minimum Cost" and the "Longest Processing Time" algorithms are discussed in relation to the proposed algorithms.

Chapter V treats the application of bin-packing and branch-and-bound algorithms to meet processor scheduling, the third algorithm is proposed for the solution of the MP problem.

Chapter VI presents a real life case study involving the design and implementation of a production planning system in an aluminum reduction plant. Real data are used to test the effectiveness of three algorithms and labor cost savings of up to 51% are observed in comparison to manually produced schedules which had been implemented by the industry.

Chapter VII summarizes the findings and draws conclusions from this research as well as to suggest areas for future research efforts.

CHAPTER II

MODEL FORMULATION AND ANALYSIS

2.1 Notation

Throughout this thesis, the regular subscript (e.g. T_j) and the computer language type subscript (e.g. T(j)) are used interchangeably. Most frequently used notations are described below.

Notation	Description and example
t	Discrete time scale, 0, 1, 2, 3,...., ∞ One time unit may correspond to 1/10 day, or 2.4 hours.
n	A finite number of jobs to be considered in a given schedule. E.g. n=30 jobs per monthly schedule.
s	The total number of machines available. E.g. s=8 machines.
ℓ	The total number of processors available. In this thesis, one or two machines are assigned to each processor. E.g. $\ell \geq s/2=4$ processors.
$\left.\begin{matrix} i \\ j \end{matrix}\right\}$	Job identification. Job i is usually considered to be followed by Job j, or $i \prec j$ $1 \leq i \leq n$ and $1 \leq j \leq n$. J_j means job number j.
r	The total number of types of resources. E.g. r=20 resources types.
e	The total number of job types available
T(j) or T_j	The job duration in time units for job j. E.g. T(1)=10 means that job #1 takes 10 time units.
E(j) or E_j	The job type for job j. $1 \leq E(j) \leq e$. E.g. E(3)=5 means that job #3 is of type 5.
R(j) or R_j	The resource type for job j. E.g. R(5)=3 means that job #5 requires resource type #3 to be attached to the processor controlling the machine which operates upon job #5.

Notation	Description and example
β	The machine identification. $1 \leq \beta \leq s$.
$\alpha(\beta)$	The processor which is shared by the machine β. E.g. $\alpha(2)=1$ means that the machine #2 is controlled by the processor #1.
k	Resource identification. $1 \leq k \leq r$. E.g. k=3 means resource type 3.
$M(t,j,\beta)$	A 0/1 integer variable that is set to 1 when the job j is assigned to machine β at time t, and 0 otherwise. E.g. M(4,2,1)=0 means that job #2 at time 4 is not on machine 1.
$P(t,j,p)$	A 0/1 integer variable that is 1, if $p=\alpha(\beta)$ and $M(t,j,\beta)=1$, and is set to zero otherwise. E.g. P(3,4,5)=1 means that at time 3, the job #4 is being operated by a machine which is controlled by the processor 5.
$A(t,j)$	A 0/1 integer variable that is set to 1 if the job j is active at time t, and set to 0 otherwise. E.g. A(2,3)=1 means that job #3 is being processed by one or more machines at time t=2.
$X(i,j)$	A 0/1 integer variable that is set to 1 if job i is followed immediately by job j on the same machine. X(4,5) means that job #5 starts as soon as job #4 is finished.
$a(t,k)$ or $a_k(t)$	The amount of resource of type k available at time t. It is assumed that once a resource is assigned to a processor, that processor will need only one unit of the resource regardless of the number of jobs being processed by machines attached to that processor and that all machines attached to that processor can only process jobs requiring that type of resource. E.g. a(3,1)=3 means that there are enough resource of type 1 available to make three processors dedicated to service machines attached to them. All such machines must process jobs whose resource requirement is of type 1.
C	Total changeover cost for the schedule.
$C(i,j)$ or C_{ij}	Changeover cost for immediately following job i with job j. E.g. C(2,3)=10 means that unloading job #2 and making the machine ready for job #3 takes 10 cost units.

Notation	Description and example
$G(E(i),E(j))$ or $C_{g_{ij}}$	The changeover cost component due to changing the job type between an old job i to the new job j. E.g. $G(E(1),E(2))=5$ means that the cost of changeover is 5 cost units for unloading a job of type 1 and loading a job of type 2 in its place.
$B(R(i),R(j))$ or C_r	The changeover cost component due to switching the resource brand between job i and job j. E.g. $B(R(1),R(2))=4$ means that changing the resource required by job 1 to another required by job 2 costs 4 cost units.
$\tau(\beta)$	The makespan of jobs on machine β. It is equal to the time duration from the beginning of the schedule ($t=0$) to when the machine first becomes idle.
$Z(\alpha)$	The makespan of the processor is the longest makespan of machines it controls. $Z(\alpha)=\max\ (\tau(\beta) \mid \beta$ attached to α).
Z	The makespan of the system. $Z=\max_{\alpha}(Z(\alpha))$
$W(j)$ or W_j	Job status. Set to 1 if the job is in a machine at $t=0$.
Q_{p_α}	A sequence of jobs on processor α. $1 \leq \alpha \leq \ell$
ω_α	A subset of jobs assigned to processor α

2.2 Input Data

For each job j, the following information must be provided before the scheduling activity can commence. These data can be conveniently denoted as an array $J_j(E_j, R_j, T_j, W_j)$ where:

$E_j = E(j)$	the type of job that the job j is. $1 \leq E(j) \leq e$
$R_j = R(j)$	the type of resource that the job j requires $1 \leq R(j) \leq r$
$T_j = T(j)$	the length of processing time for the job j.
$W_j = W(j)$	the processing status of the job j at the beginning

of the schedule. $W(j) = 0$ if the job is new, 1 if currently on a machine being processed. Omitted if $W(j)=0 \ \forall \ j$.

In addition, the following data must also be supplied.

$\alpha(\beta)$	The processor α to which machine β is attached.
$C_{i,j}=C(i,j)$	The cost matrix computed from the job type (grade) change $E(i)$ to $E(j)$ and the resource type change $R(i)$ to $R(j)$, using the grade change cost matrix $G(E(i),E(j))$ and the resource change cost matrix $B(R(i),R(j))$.
$a(t,k)$	The resource availability schedule for resource k at time t, $1 \leq k \leq r$ and $t=0,1,2,\ldots.$

The job description $J_j(E_j, R_j, T_j, W_j)$ is conveniently abbreviated as J_j on Gantt Charts, or expressed only with meaningful parameters when others are not used. The notations P_α and m_β are used to identify the processor α and machine β.

2.3 The Independent Variable

The independent decision variable in the MP formulation is $M(t,j,\beta)$, a 0/1 integer variable which identifies whether a job j is assigned to the machine β at time t or not.

From this information and $\alpha(\beta)$, we know which processor is being engaged. Similarly, $R(j)$ will identify the resource that must be attached to the processor to process this job.

The job sequencing within each machine is controlled by the 0/1 integer variable $X(i,j)$ which is set to 1 only if job j is immediately preceded by job i in the same machine. However, $X(i,j)$ can be expressed as a function of $M(t,i,\beta)$.

$$X(i,j) = \begin{cases} 1 \text{ if } \sum_{t=0}^{\infty} \sum_{\beta=1}^{s} M(t,i,\beta)\, M(t+1,j,\beta)=1 \\ 0 \text{ otherwise.} \end{cases}$$

2.4 Model Formulation

The scheduling problem MP can be formulated as an integer programming model:

$$\text{Minimize } C = \sum_{i=1}^{n} \sum_{j=1}^{n} C(i,j)X(i,j)$$

where C is the total cost of changeovers for the schedule,

$$X(i,j) = \begin{cases} 1 \text{ when a job switch occurs from job i to job j} \\ \text{on the same machine.} \\ 0 \text{ otherwise.} \end{cases}$$

$$C(i,j) = \begin{cases} G(E(i),E(j)) + B(R(i),R(j)) & \text{if } i \neq j \\ \infty \text{ otherwise} \end{cases}$$

$n =$ total number of jobs in the schedule.

Subject to the following constraints:

(i) Each job must be assigned to a machine for its job duration.

$$\sum_{t=0}^{\infty} \sum_{\beta=1}^{s} M(t,j,\beta) = T(j) \text{ for every } j,\ 1 \leq j \leq n$$

(ii) Total usage of a resource cannot exceed its availability at any time.

$$\sum_{j=1}^{n} (P(t,j,p) \mid R(j)=k) \leq a(t,k) \text{ for every } t \text{ and every } k,$$

$$0 \leq t \leq \infty, \; 1 \leq k \leq r.$$

and $$A(t,j) = \bigcup_{\beta=1}^{s} M(t,j,\beta)$$

(iii) A job is assigned to each machine from the very beginning of the schedule.

$$\sum_{j=1}^{n} M(0,j,\beta)=1 \text{ for all } \beta \qquad 1 \leq \beta \leq s.$$

(iv) The job changeover occurs only when a job i is immediately followed by a job j in the same machine.

$$X(i,j) = \begin{cases} 1 \text{ if } \sum_{t=0}^{\infty} \sum_{\beta=1}^{s} M(t,i,\beta)\, M(t+1,j,\beta)=1 \\ 0 \text{ otherwise.} \end{cases}$$

(v) There is no idle time allowed between jobs on machines.

$$\sum_{\beta=1}^{s} \sum_{t=0}^{\tau(\beta)} \sum_{j=1}^{n} M(t,j,\beta) = \sum_{j=1}^{n} T(j)$$

where

$$\tau(\beta) = \min \left(t \mid \prod_{j=1}^{n} M(t,j,\beta)=0\right)$$

2.5 Assumptions

The following assumptions are usually implied, and unless otherwise stated, apply to the remainder of this thesis.

(i) Time zero, t=0 is defined as the instant at which the new schedule commences.

(ii) No job cancellation is allowed after the job schedule has been set up. If it does occur, a rescheduling will be necessitated.

(iii) No preemptive priority is allowed. This means that the job may be split between machines but not interrupted and delayed or discarded.

(iv) No precedence relationship may exist among jobs.

(v) The processing time for each job is finite, deterministic, and known before scheduling.

(vi) Machines, processors, jobs, and resources are assumed to be available throughout the schedule horizon. If availability changes, a rescheduling may become necessary.

2.6 The Complexity of Scheduling Problems

2.6.1 An "easy" vs "hard" problem

How much computation should a problem require before we rate the problem as being easy or difficult? There is a general agreement that if a problem cannot be solved in polynomial time, then the problem should be considered intractable. The following definition is made to measure the complexity of a problem. (Aho *et al*, 1974, p. 364)

Definition A problem is a polynomial time problem if an algorithm exists which can find an optimal (or exact) solution with a number of computations which grows at a rate less than a polynomial function of the "size" of the parameters specifying the instance of the problem. (A problem which is not polynomial time is an exponential time problem.)

Before we can analyze how 'hard' the MP problem is, the concept of a class of problems which are called NP-complete (nondeterministic polynomial-time complete) is needed. Rather than digressing to define it explicitly, only some implications of belonging to that class will be presented. An excellent treatment of NP-complete problems is contained in Garey and Johnson (1979).

To say that a problem is NP-complete implies that the problem has the following two characteristics.

(1) If a polynomial time algorithm can be found to solve the problem, then a polynomial time algorithm exists for all NP-complete problems, which include the linear integer programming problem, the travelling salesman problem, the set covering problem, and many others.

(2) The problem is an exponential time problem.

Since no polynomial time algorithm has been found for an NP-complete problem, it is conjectured that none exists. If this is true, the diagram shown in Figure 2.1 would apply.

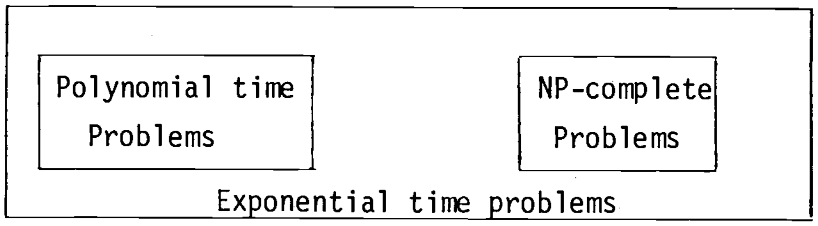


Figure 2.1 Problem Classes

2.6.2 The complexity of the MP problem.

A scheduling problem is easy to state but difficult to solve. "It has been a graveyard for practicing management scientists and problem solvers for many years" (Poole, 1977, p. 49). A more traditional and now classical quote from Conway et al. (1967, p. 103) asserts pessimistically that:

> Many proficient people have considered the problem, and all have come away essentially empty-handed. Since this frustration is not reported in the literature, the problem continues to attract investigators, who just cannot believe that a problem so simply structured can be so difficult, until they have tried it.

A scheduling problem becomes difficult for mainly two reasons:

(1) The combinatorial nature of the problem.

(2) The problem has to satisfy too many objectives at once.

The theory of NP-completeness provides many straight forward techniques for proving that a given problem is "just as hard as" the large number of other problems that are widely recognized as being difficult (Garey and Johnson, 1979). These problems have been challenging the experts for years.

To prove that a problem in NP is NP-complete, it suffices to prove that some other NP-complete problem is polynomial transformable to it since the polynomial transformability relation is transitive (Baase, 1979). "Partition" is an NP-complete problem.

Theorem 2.1 The problem of finding an optimal schedule to a set J of n jobs, on ℓ processors with variable processing time T_j $(1 \leq j \leq n)$ and a time limit D is NP-complete.

This problem can be transformed from Partition. A detailed proof can be found in Garey and Johnson (1979, p. 64).

Theorem 2.2 Scheduling a set J of n jobs, on ℓ processors with variable processing time T_j $(1 \leq j \leq n)$ and r resources; with resource bound a_k $(1 \leq k \leq r)$ and time limit D is NP-complete.

This problem can be transformed from 3-Partition. A detailed proof can be found in Garey and Johnson (1975, p. 408).

Theorem 2.3 MP is NP-complete.

The subproblems of MP from the above two theorems are proved to be NP-complete, therefore MP is also NP-complete.

It is bad news to know that MP is intractable. However, it is felt by this author that this should not be a reason for neglecting this problem. For small problems exponential time algorithms may perform just as well as polynomial time algorithm (e.g., $2^n \leq n^{10}$ for $1 < n < 59$). In addition, by saying that a problem is an exponential time problem implies that an algorithm exists, which can solve even the worst set of values of the parameters of the problem in exponential time.

CHAPTER III

REVIEW OF LITERATURE AND METHODS OF SOLUTION

3.1 Survey of Past Work

Before we go on to review other past research work in this area, it is helpful to identify the relative position of the MP problem among other problems in sequencing theory. A scheme for classifying sequencing problems is shown in Figure 3.1 (Day, 1970, p. 119). The deterministic sequencing problems are divided into those with single processors and those with multiple processors. Compared with the problem of multiple processors, the problem of single machine has received much more attention in the literature. It is worthwhile to review and study some results and solution methods for the problem of a single processor case, mostly because these results and methods have given us ideas about the approaches used in the solution of the MP problem in this research. A formal description of the job shop scheduling problem and an excellent summary of past research works are also given by Conway *et al*. (1967).

Sequencing problems with multiple processors in series have drawn more attention from researchers than those with multiple processors in parallel (Day, 1970). The criteria (measure of performance or objective) proposed in the literature on the parallel case of static sequencing include: (1) Minimize the cost of tardiness and penalty (Elmaghraby and Park, 1974; Schild and Fredman, 1961; Barnes and Brennan, 1977), (2) minimize the makespan (Elmaghraby and Elimam, 1980; McNaughton, 1959;

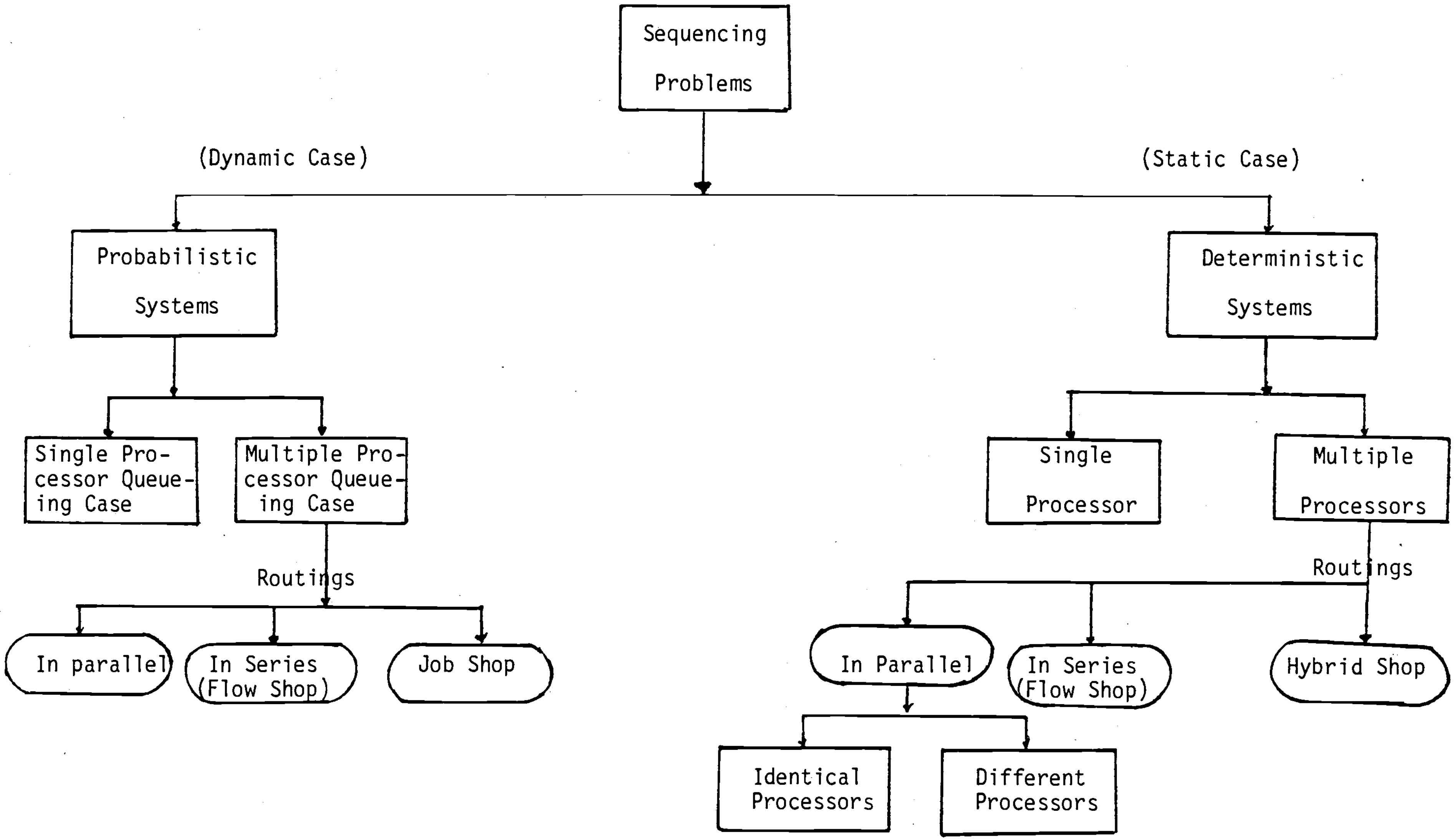


Figure 3.1 Overview of Sequence Theory (Day, 1970, p. 119)

Coffman, 1976; Barker, 1974, p. 116; Graham, 1969), (3) minimize the total cost of production (Gorenstein, 1970, p. 373 and Dinkel et al. 1976), and (4) minimize the maximum flow time (Conway et al. 1967).

One way to solve a difficult problem is to solve a related 'easy' problem and hope that the solution to the easy problem can be shown to be a solution to the difficult problem. There are very few papers which focus on the minimization of changeover cost on multiprocessors or multiple machines. However, there are a lot of algorithms developed for single machine models (Glassey, 1968, p. 342; Driscoll, 1971, p. 388; and Presby, 1967, p. B454). The task of minimizing the sum of the production cost or set-up time on a single machine corresponds to what is usually called the traveling-salesman problem (TSP). The TSP can be stated as follows: a salesman must visit each of n cities once and only once and return to his point of origin and do so in a way that minimizes the total distance traveled (or total time, or cost, etc.). Each city corresponds to a job, and the distance between cities corresponds to the time or cost required to change over from one job to another. A set of nonrepetitive jobs to be scheduled on a single machine is similar to an open-path TSP. There are algorithms to solve it (Gavett, 1965 and Ramalingam, 1969, p. 85). For the close-path TSP, there are many papers discussing how to solve this problem efficiently. A good summary of methodologies may be found in the paper by Bellmore and Nemhauser (1968, p. 538).

In the real world, the constrained version of the TSP is seen to be a generic model for a wide variety of problems. Carpaneto and Toth (1977) developed a branch-and-bound algorithm based on a depth-first technique to solve TSP with due date. Dinkel et al. (1976) and Dantzig and Ramser (1959) used several good approximation methods to solve a constrained TSP. Their constraints are the length of a trip and the capacity of the vehicles.

Bodin and Kursh (1978) solved an m street-sweepers routing problem by using TSP solution technique. The methods of "cluster first,route second" and "route first,cluster second" are introduced. However, they favor the "cluster first,route second" approach. It decomposes the network into a collection of m (the number of vehicles to be used) subclusters and then solves a "one vehicle" routing problem over each of these subclusters.

Frederickson et al. (1978) developed two methods for building k tours for k traveling salespersons. The first method is to build k subtours simultaneously. A set of heuristic rules (the nearest neighbor, the nearest insertion, and the cheapest insertion, etc.) is used to generate an approximate solution to a "one person" problem. The second method is to build k tours by splitting a good tour for one person into k tours.

Another formulation of the multiple salesmen problem is given in the paper by Gorenstein (1970). He regards m traveling salesmen to be the same as a single traveling salesman problem with m-1 additional home visits. However, Svestka and Huckfeldt (1973) solved the m-salesman problem as an (m+n-1) city problem. Their algorithm consists of three main parts; the branch-and-bound scheme, the initial tour generator, and the assignment algorithm. Every solution to the m-salesman problem will contain exactly m sorties, one for each of the m salesmen.

In resource constrained scheduling, Garey and Johnson (1975) showed why resource constrained scheduling is so difficult. They proved that even with just two processors and one resource available, a set of unit execution jobs result in a scheduling problem that is NP-complete. Garey and Graham (1975) studied multiprocessor scheduling with resource constraints and derived a number of close bounds for this system.

A complete and detailed guide to the problem of scheduling under resource constraints can be found in the review paper by Davis (1973) and Moder and Phillips (1970, p. 152).

3.2 Methods of Solution

A survey of the approaches used in solving the scheduling problems reveals that there are mainly five different methods:

(i) Combinatorial analysis

(ii) Integer linear programming

(iii) Branch-and-bound methods

(iv) Dynamic programming

(v) Heuristic methods

Theoretically, the first four methods lead to an exact optimal solution. The remainder of this section will be devoted to reviewing the five approaches for scheduling problems.

i) Combinatorial approach

Methods of combinatorial analysis often turn out to be useful in some scheduling problems. They frequently involve a close examination of the effect of a minor change in a particular schedule (notably the interchange of two possible adjacent jobs) to satisfy a given criteria (Root, 1965). The basis for the works by Smith (1956) and McNaughton (1959) is also this combinatorial approach.

ii) Integer linear programming

A natural way to attack machine scheduling problems is to formulate them as mathematical programming models. Pritsker, Watters and Wolfe (1969, p. 93) proposed a model to solve the multi-project scheduling problem for which several objective functions were allowed; i.e. minimization of the total project throughput time, minimization of total makespan and minimization of the total cost of tardiness.

Garcia (1976) developed an interactive computer system to solve classroom scheduling using integer programming. The objectives were to maximize the number of student requested courses, utilize the classroom facilities as efficiently as possible while keeping the size of the courses within given bounds.

In the case of resource constrained scheduling, numerous integer programming formulations have appeared in the literature (Wagner, 1959; Manne, 1960; Mason and Moodie, 1971). However the solution of real problem using general purpose integer programming code has not appeared computationally feasible (Barker, 1974, p. 286).

Geoffrion and Marsten (1972) gave a good summary of the state-of-the art integer programming techniques. They described what kind of general purpose integer linear programming algorithms existed and their computational success. They remarked that the integer programming generally can not solve many special structured scheduling problems. Problem solvers often turn to more tailor-made forms of implicit enumeration similar to those which are to be discussed next.

iii) Branch-and-bound

Branch-and-bound methods are useful tools for solving many combinatorial problems. They are sometimes also known as "reliable heuristics," "controlled enumeration" or "implicit numeration". There are nine different characteristics of branch-and-bound which are described by Kohler and Sterglitz (Coffman, 1976, Chapter 6). Branch-and-bounds were developed and first used in the context of mixed integer programming (Land and Doig, 1960) and the traveling salesman problem (Eastman, 1959), but soon their wide applicability was perceived.

From the past literature survey, we have mentioned that most of the least cost routing and sequencing problems are closely related to the single and multiple salesmen problem with further constraints (e.g., due date, machine capacity, etc.) to be met. The majority of the literature which has been cited used branch-and-bound methods. However, the bounds for all the least cost branch-and-bound methods (LCBB) using relaxation are calculated based on the availability of one machine only. Therefore, it is not surprising to know that the LCBB methods applying to multi-machines scheduling problems have received little attention in the scheduling literature.

Thompson (1970) developed a general FORTRAN-based package for solving sequencing problems using branch-and-bound methods. His program can solve one resource to many resource constrained project scheduling problems. His program data structures resembled that of GASP II simulation language which was developed by Pritsker and Kiviat (1969).

iv) Dynamic programming

Dynamic programming is closely related to certain branch-and-bound algorithms. It is an algorithm design method that can be used when the solution to a problem may be viewed as the result of a sequence of decisions. It drastically reduces the amount of enumeration by avoiding the enumeration of some decision sequences that can not possibly lead to an optimal solution. In dynamic programming, an optimal sequence of decisions is arrived at by making explicit appeal to the Principle of Optimality (Riggs and Inoue, 1975, p. 296). This principle was developed by Richard Bellman. It states that an optimal sequence of decisions has the property that whatever the initial state and decisions are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

A dynamic programming formulation for the problem of a single processor was given by Held and Karp (1962) and Lawler (1964). Driscoll and Emmons (1977, p. 388) used dynamic programming to find an optimal schedule on one machine. Their objectives were to minimize the total changeover cost and meet the due date of all customers. Their algorithm required 155 CPU seconds (IBM370) to solve 15 jobs.

Horowitz and Sahni (1978, p. 233) showed that an $O(n^2 2^n)$ dynamic programming algorithm solves the traveling salesman problem. Although this represents a considerable improvement over explicit enumeration (e.g. for a 15-job problem, $15^2(2)^{15} = 7,372,800$ where explicit enumeration gives $15! = 1.31 \times 10^{12}$), this method is still computationally infeasible for problems with a large number of jobs.

For the problem of sequencing n immediately available jobs on multiple processors, Rothkopf (1966) presents a formulation. His objective is to minimize the total penalty cost. One noticeable assumption of his model is that the order in which the jobs are considered for scheduling is specified in advance. He mentions that the number of calculations is of the order of $(t)^{n-1}(m-1/2)^n/n4^{n-2}$, where t is the average processing time for the identical machines. For $n=5$, $t=6$ and $m=2$, the number of calculations is approximately 35,000.

v) Heuristic methods

Heuristic methods usually consist of a series of priority rules which, when applied to the basic problem data, give a feasible but not optimal solution. They are characterized by Brown (1971, p. 86) as:

(a) Being derived from the problem environment; and thus

(b) Being highly problem-specific

(c) Giving sub-optimal results with uncontrolled error

(d) Being often intuitive in nature.

In most practical situations, because of the complexity of a problem, to find an optimal solution would often be too time-consuming to be feasible. Under these circumstances, heuristic methods that produce good, but possibly suboptimal solution are of interest to most of the practitioners.

Presby and Wolfson (1967) offered a heuristic approach to sequence jobs on individual machines to minimize job changeover cost. Their algorithm is very similar to dynamic programming. The algorithm starts with four job sequences and from these constructs five job sequences, from the five job sequences, six job sequences are constructed, and so on. At each stage, a large fraction of sequences are eliminated from further consideration. They claimed that the number of considerations (N) are

$$N = k! \sum_{i=1}^{i=k-2} \frac{1}{i!(k-i-2)!} \quad \text{where k is number of jobs.}$$

So, for a list of ten jobs, N=22950, while the number of complete sequences for ten jobs is 10! = 3,628,800.

Gavett (1965) has developed three heuristic rules for choosing a least cost schedule for a single machine situation. The three heuristic methods are:

(i) The "Next Best" rule

(ii) The "Next Best Prime" rule

(iii) The "Next Double Prime" rule.

He has tested the algorithm for a large number of problems in which the elements of the cost matrix are independent identically distributed random variables--in some cases from a normal distribution, in others from a rectangular distribution. Examples of each type of problem are generated and tested. The performance of the algorithm seems to weaken as the number of jobs increases.

Researchers usually have to face another problem when they schedule multiprocessors in parallel, that is the problem of makespan minimization. It appears to be difficult in general because it is known to be NP-complete (Coffman, 1976, Chapter 4). McNaughton (1959) obtained an optimum solution to the makespan problem when job pre-emption is allowed.

Graham (Coffman, 1976, Chapter 5) describes a sequence of algorithm that yields an optimum in a computation time that grows exponentially with number of processors and behaves more and more like exhaustive search as the guaranteed accuracy improves.

Barker (1974, p. 116) refers to Kedia's Longest Processing Time (LPT) algorithm to minimize makespan in multiprocessors. It ranks jobs with the longest processing time first, then assigns a job from the list to the processor with the least amount of processing time already assigned. Graham (1969) showed that the makespan obtained by Kdeia's LPT algorithm is at most 4/3 of the optimum.

Elmaghraby and Elimam (1980) present a knapsack-based heuristic method for makespan problems with large numbers of machines.

CHAPTER IV

ALGORITHMS DEVELOPMENT

4.1 Analytic Models

4.1.1 Makespan Minimization on Parallel Processors.

In the single-machine model, the makespan is equal to a constant for any sequence of n given jobs, therefore the makespan problem in the single-processor case is trivial. In multiple-processor cases, however, this is no longer the case.

An elementary result for the makespan problem was presented by McNaughton (1959) with the assumptions that jobs are independent and preemption is permitted. With preemption allowed, the processing of a job may be interrupted and the remaining processing can be completed subsequently, perhaps on a different machine. Therefore, an optimal schedule would have divided the processing load $\sum_{j=1}^{n} T_j$ evenly among the ℓ processors. The schedule of length D (or planning horizon) is $\sum_{j=1}^{n} T_j/\ell$.

Consider the following job set when $\ell=4$ processors are available:

TABLE 4.1 List of jobs and their processing time.

J_j	1	2	3	4	5	6	7	8
T_j	1	2	3	4	5	6	7	8

The planning horizon D is the same as the optimal makespan Z^*

$D = Z^* = 36/4 = 9$

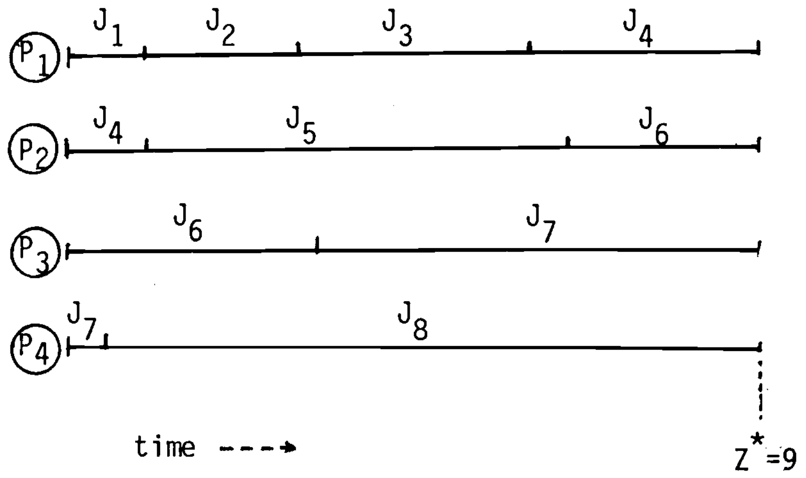


Figure 4-1 Gantt chart shows that a preemptive schedule can achieve an optimal makespan.

If job preemption is prohibited, the problem of minimizing makespan is more difficult. No exact method has been developed to solve this problem optimally. The minimum makespan is obtained by the following formula:

$$\text{Min } Z = \underset{1 \le \alpha \le \ell}{\text{Max}} \left\{ \sum_{j \in \omega_\alpha} T_j \right\}$$

A simple yet effective heuristic procedure called LPT (Longest Processing Time) algorithm was reported by Kedia (Barker, 1974, p. 116). This heuristic can be implemented to run in a time proportional to $n\log(\ell n)$. The algorithm is described as follows:

Step 1: Construct an LPT ordering of the jobs, from the longest to the shortest, e.g. $J_8, J_7, \ldots, J_1$.

Step 2: Schedule the jobs in order, each time assigning a job to the processor (or machine) that has the least amount of processing already assigned.

A nonpreemptive schedule of jobs from Table 4.1 resulting from LPT heuristic procedure is shown in Figure 4.2.

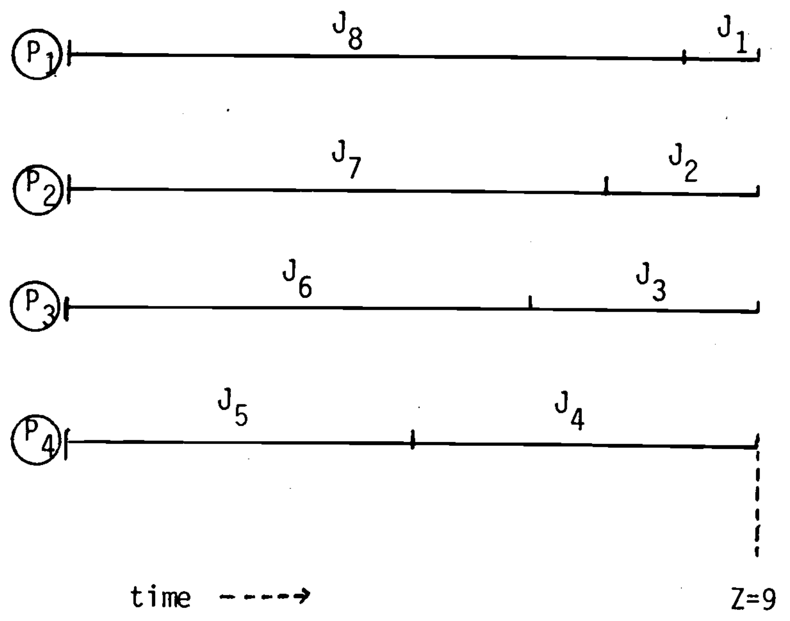


Figure 4.2 Gatt chart shows that a non preemptive schedule from LPT algorithm.

It so happens that, in this example, $Z^* = Z$. Graham (1969, p. 416) shows that the makespan obtained by Kedia's LPT has a bound of $(1/3 - 1/(3\ell))$ in the worst case, i.e.

$$\frac{Z^*(I) - Z(I)}{Z^*(I)} \leq \frac{1}{3} - \frac{1}{3\ell}$$

where ℓ is the number of processors. $Z^*(I)$ is the finish time of an optimal ℓ-processor schedule for instance I of the schedule problem. Z (I) is the finish time of an LPT schedule for the same instance.

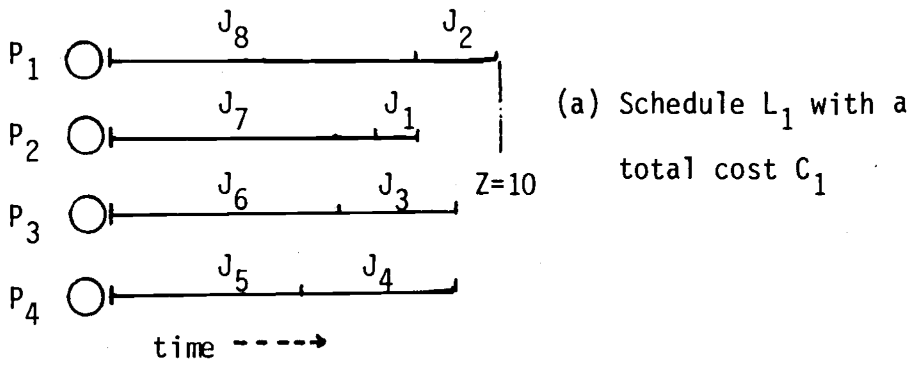
Although the main objective of MP focuses upon the total cost changeover minimization, the makespan consideration is not to be neglected. For example, in Figure 4.3, there are three possible schedules for eight jobs on four processors, Z_1=10, Z_2=12 and Z_3=13 for schedules L_1, L_2 and L_3, respectively. Each schedule has a production cost. In this case, suppose $C_1 > C_2 > C_3$ in dollar value. From the cost reduction scheduling point of view, C_3 is the least cost, so L_3 should be chosen. In the real world situation, however, if $C_1 - C_3 = \varepsilon$, and ε is a small value in dollars, schedule L_1 may be chosen because the percentage of processor utilization is better than any of the other two. Therefore, makespan minimization can not be ignored in identical parallel processors scheduling.

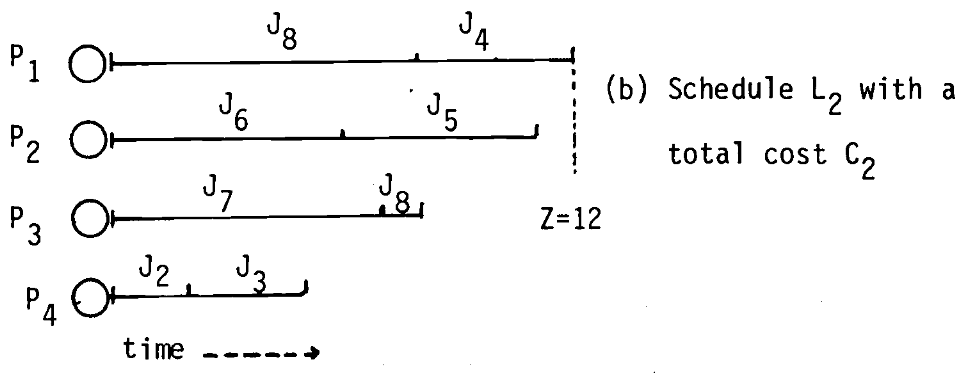
4.1.2 Single machine vs. double machines

Two machines generally share one processor and one resource to perform a task. We assume that if a job may be split over two machines, then that job is completed 50% earlier than on one machine. This can be shown more clearly with a Gantt Chart. Consider the job set in the following table when all jobs use the same kind of resource. The makespan for two cases is shown in Figure 4.4

Table 4.2. A jobs list

J_j	1	2	3	4
T_j	4	2	4	2





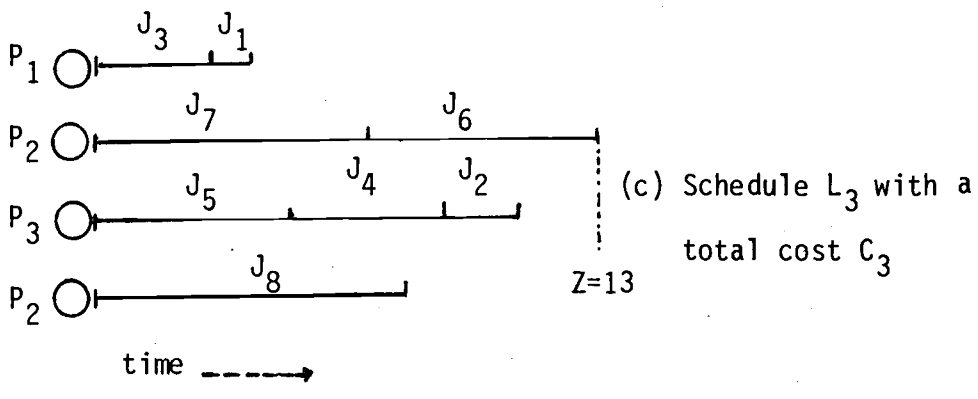
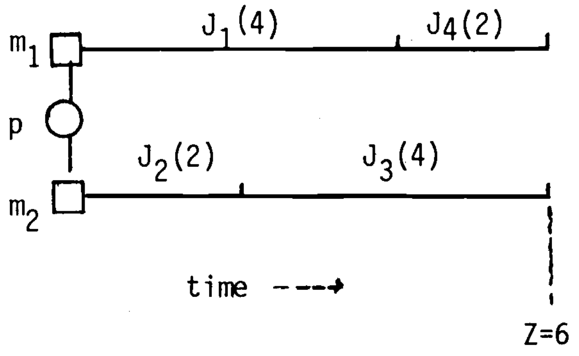


Figure 4.3 Gantt Charts for three possible schedules

Case 1 when $\ell=1$, s=2



Case 2 when $\ell=1$, s=1

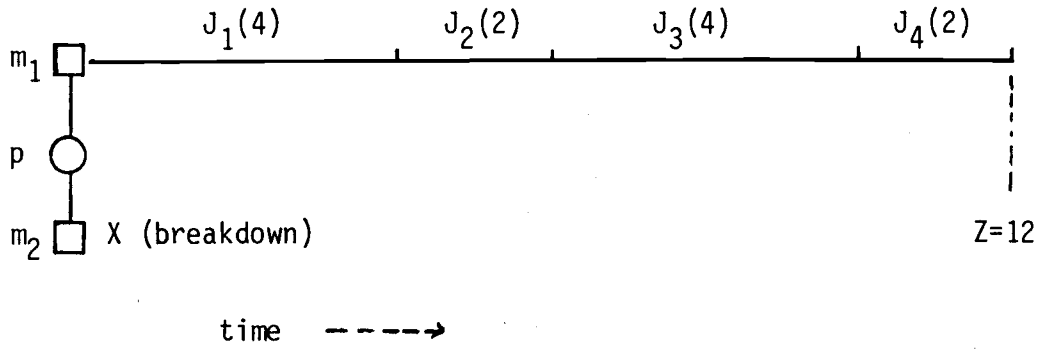


Figure 4.4 Jobs execution Gantt Chart of one machine vs. two machines.

4.1.3 Resource Constraints Consideration

Lemma: 4.1

If $\sum_{u=1}^{t} K(u) > \sum_{u=1}^{t} a(u,k)$ for any t, and k $1 \leq t \leq D$, $1 \leq k \leq r$, then

no feasible schedule of length D exists.

Proof: Assume that a feasible schedule of length D exists. A feasible schedule implies that all resource requirements are met. Thus, any duration t less than or equal to D, we should have $k(u) \leq a(u,k)$ or $\sum_{u=1}^{t} k(u) \leq \sum_{u=1}^{t} a(u,k)$. This contradicts the premise above. Q.E.D.

Consider the following example when s=3, ℓ=2, r=2.

Table 4.3 A jobs list and their attributes.

J_j	1	2	3	4	5
T_j	4	2	6	2	10
R_j	1	1	3	3	3
a_k	1	1	1	1	1

$$Z^* = \sum_{j=1}^{5} Tj/S = \frac{24}{3} = 8$$

Let D=8 which is also equal to the optimal makespan (Z^*) for the schedule. Each resource type has only one unit available; we construct one possible schedule which is shown on the following page.

Although there is a two-unit-time space available on machine m_3 after job J_2, job J_4 can not be placed on machine m_3 because no additional resource of type 3 is available.

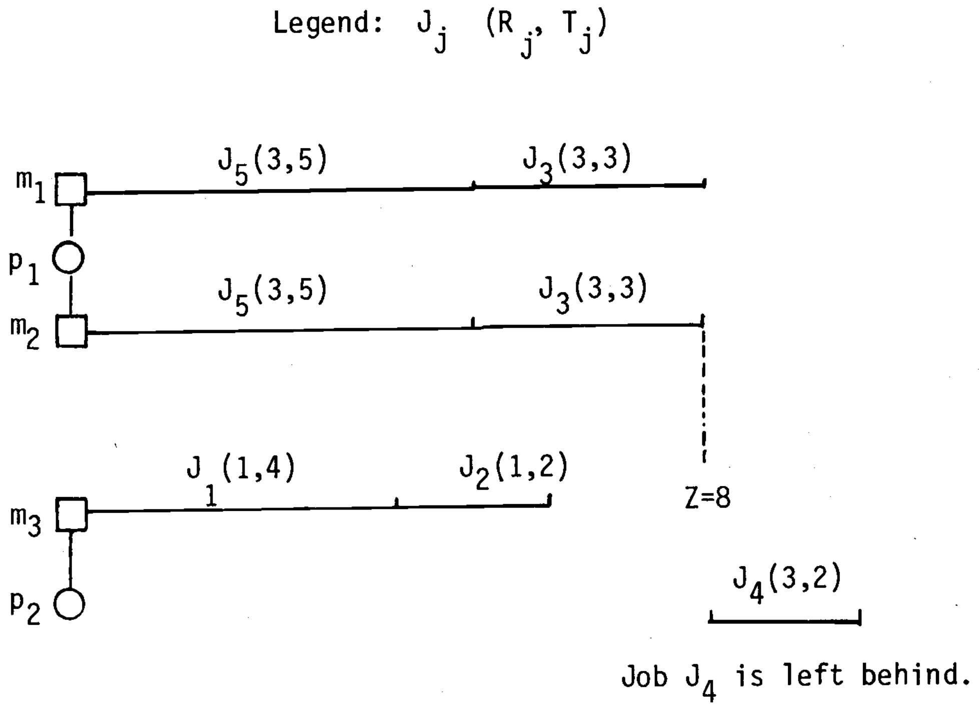


Figure 4.5. D=8 Scheduling dilemma.

Corollary 4.1 $a_k(t) \leq a_k$. The total usage of resource type k $(1 \leq k \leq r)$ at any instant of time t must not exceed its total availability.

4.1.4 Changeover Criteria

In real world situations, a set of jobs J can be classified according to their properties or functions. We consider three possible cases where a cost will be incurred when job J_i changes to job J_j.

Case 1. $J_i(E_i,R_i)$ ---> J_j (E_i,R_j)

i.e. When a job J_i of job type E_i and a resource R_i

changes to job J_j with the same job type E_i but different resource R_j usage. The cost for job change-over will be due simply to the change in resource (tools), and is often a constant.

$$C_{ij} = C_r$$

Where C_r stands for the resource changeover cost.

<u>Case 2</u>: $J_i(E_i,R_i) \dashrightarrow J_j(E_j,R_i)$

i.e. Job J_i and J_j are using the same resource R_i but the job type is different. Therefore the cost of changeover will be the cost incurred in grade change, cleaning or some other technical adjustment, etc.

$$C_{ij} = C_{g_{ij}}$$

Where $C_{g_{ij}}$ stands for the cost due to different job type changes.

<u>Case 3</u>. $J_i\ (E_i,R_i) \dashrightarrow J_j\ (E_j,R_j)$

In this case both jobs have different job type and resource type requirements, therefore the total changeover cost will be

$$C_{ij} = C_r + C_{g_{ij}}$$

Lemma 4.2 If there is no resource conflict, there exists an optimal schedule in which no job is split.

Proof: Let L_1 be a schedule with job-splits. We wish to show that there always exists a better (or equally good) schedule L_2 with no job-split other than L_1. There are three types of job split:

(1) a job is split on the same machine.

(2) a job is shifted from one machine to another adjacent machine which is controlled by the same processor.

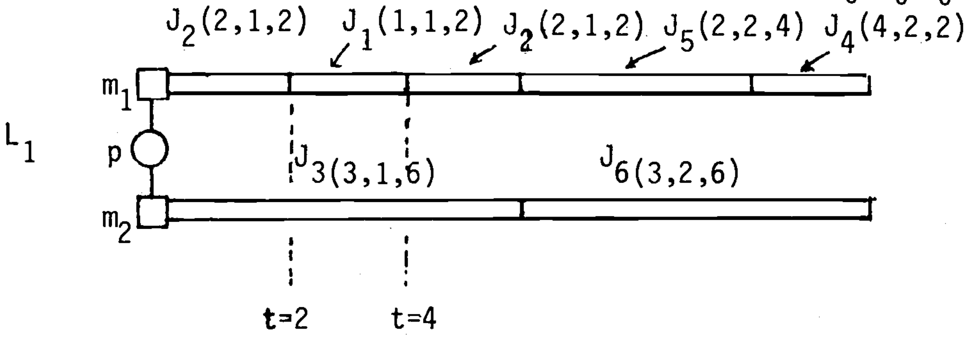
(3) a job is shifted from one machine to another machine which is controlled by a different processor.

Consider an example with a set of job J when n=6, e=4, and s=2. The following table shows the numerical value for J_j $(1 \le j \le n)$, R_j $(1 \le R_j \le r)$ and E_j $(1 \le E_j \le e)$ and each job execution time T_j.

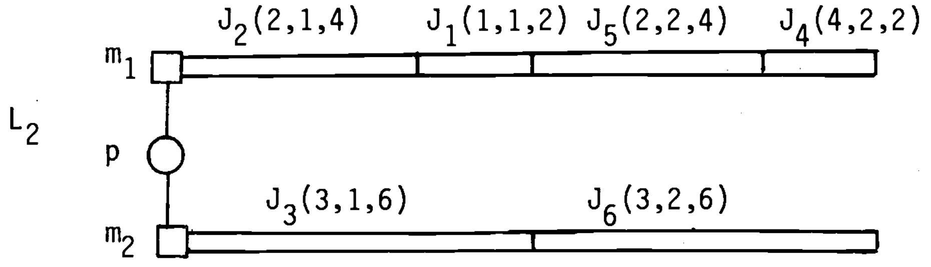
Table 4.4 A Jobs List

J_j	1	2	3	4	5	6	
E_j	1	2	3	4	2	3	When ℓ=1 and s=2,
R_j	1	1	1	2	2	2	then D=12
T_j	2	4	6	2	4	6	

Case 1. A job split on a machine. Legend: $J_j(E_j,R_j,T_j)$

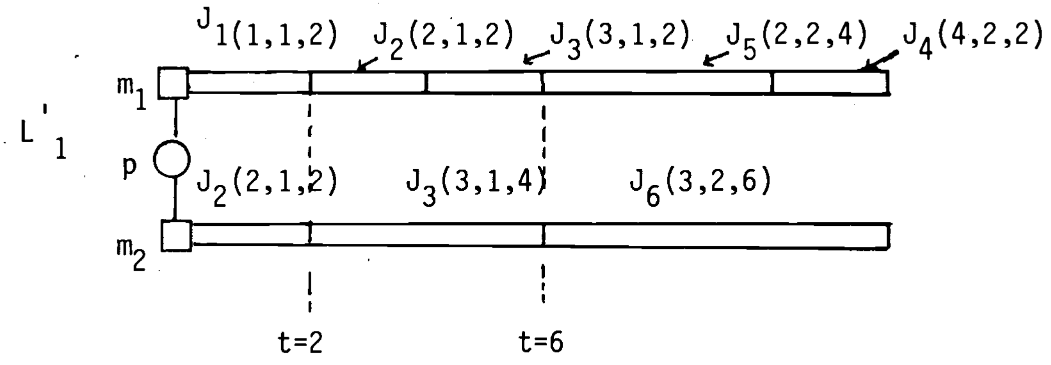


In schedule L_1, we notice that job J_2 is interrupted at t=2 and resumed after job J_1 has been completed. The total number of changeovers in Q_{m_1} is five (including the old job in previous schedule on machine m_1). We can find a better schedule L_2 with no job split.

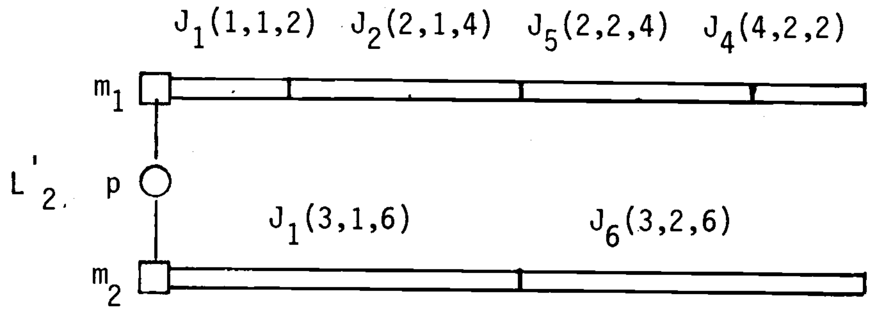


In schedule L_2, the number of changeovers in Q_{m_1} is decreased by one, i.e., four. The least number of changeovers, the better the schedule will be.

Case 2: A job is split between two machines that are controlled by the same processor.



In schedule L_1', job 2 is interrupted at t=2 on machine m_2 and transferred to m_1 after job 1 is completed on machine m_1. Because of this interchange, job 3 has to be split between machines m_1 and m_2 in order that all jobs using the same resource type 1 may finish by t=6. The number of changeovers on m_1 and m_2 are five and three, respectively. We can find a better schedule L'_2 with no job split between the machines.



The number of changeovers on m_1 and m_2 are four and two, respectively. Case 3 can be shown similarly to case 2. By induction, there always exists a better (or equally good) schedule with no job split for any schedule that has a multiple-split schedule on a machine or machines.

Corollary 4.2 If there is a set of jobs J and each job in the set uses a distinct type of resource, then each job has to be split over two machines which are controlled by a single processor.

The above statement can be illustrated more clearly by using an example.

Consider s=2, ℓ=1 n=3 and r=3. A set of jobs J with their attributes is listed below.

Table 4.5 A Jobs List.

J_j	1	2	3
E_j	1	2	3
R_j	4	8	5
T_j	4	2	6

We construct three schedules to distinguish feasible schedules from infeasible schedules.

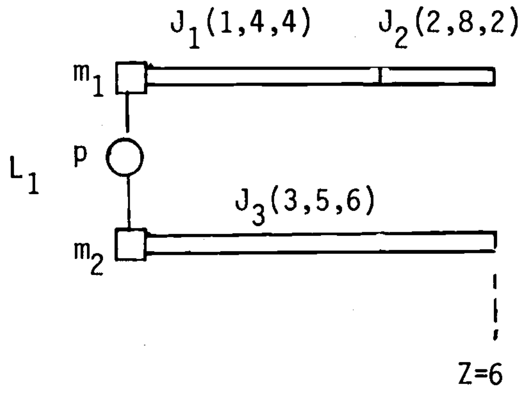


Figure 4.6a An infeasible schedule

Figure 4.6a, Schedule L_1 is an infeasible schedule because one processor can not use two resources (4 and 5, then 8 and 5) at the same time.

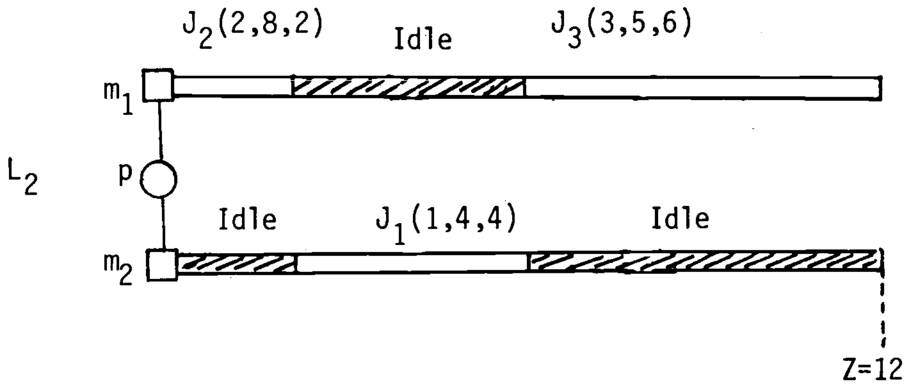


Figure 4.6b. An infeasible schedule

In Figure 4.6b, schedule L_2 does not have a resource conflict problem; however, the machine utilization is very poor. When there is idle time existing in a schedule, we say that that schedule is not feasible.

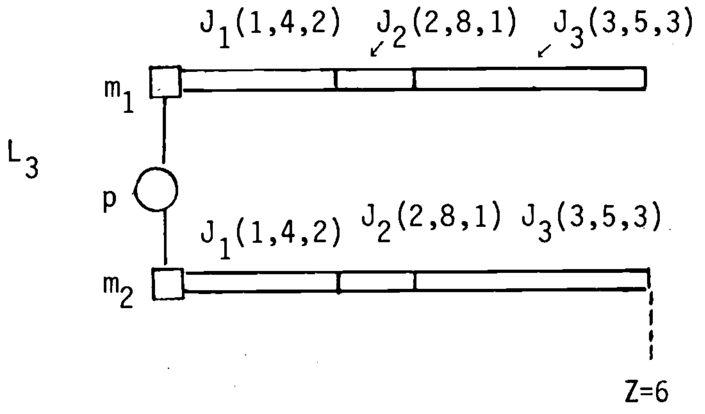


Figure 4.6c. A feasible schedule

In Figure 4.6c, L_3 is a feasible schedule. Each job is split over two machines. The makespan is six. There is no other feasible schedule. L_3 can be represented by a processor Gantt Chart in Figure 4.6d.

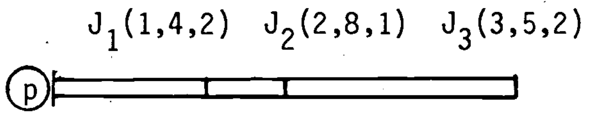


Figure 4.6d. A processor Gantt Chart.

4.1.5 Permutation Schedules

In previous sections (4.1.1 to 4.1.4), we have discussed some characteristics of MP. Here we wish to extend our discussion of what is a feasible or infeasible schedule to cases where set-up costs are associated with the decisions of a schedule. Graphical description is used to show the relationship among the job type, resource type, job processing time, processors and machines.

In many industrial situations, a set of new jobs must be scheduled on the machines or processors which are still processing some of the jobs from the previous schedule. In the following production period, the next set of jobs, including the jobs being processed, is to be sequenced for processing on the same machines so that the total changeover cost (or the total set-up time) for all the machines is minimized.

Consider an example with four machines, s=4, and two processors, ℓ=2. Suppose that seven new jobs are to arrive and their job attributes and changeover cost are shown in Table 4.6 as:

Table 4.6 A from-to cost matrix and the jobs descriptions.

J_1 (1,1,8)

J_2 (2,1,6)

J_3 (1,2,5)

J_4 (3,2,5)

J_5 (5,2,10)

J_6 (4,3,3)

J_7 (2,3,3)

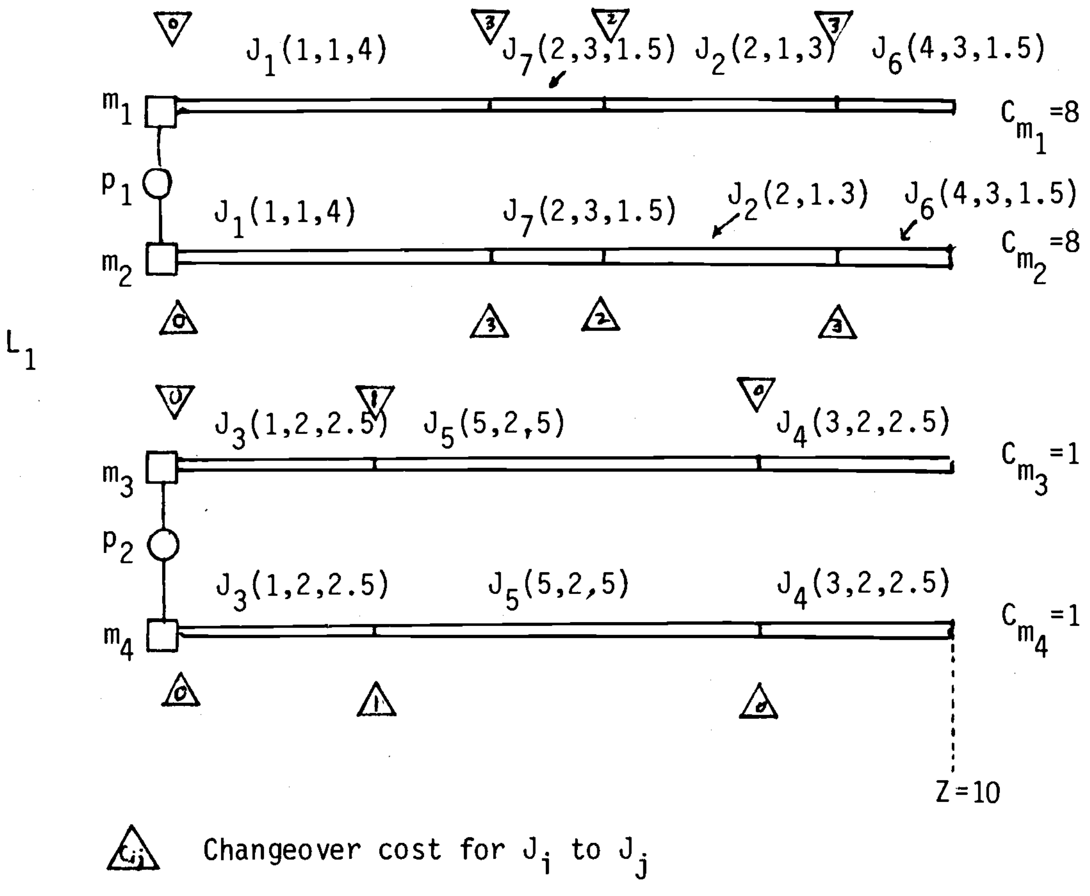
Job changeover Cost Matrix $[C_{ij}]$

J_i \ J_j	1	2	3	4	5	6	7
1	0	5	2	4	6	5	3
2	1	0	3	10	7	3	2
3	2	4	0	4	1	3	6
4	9	3	0	0	1	4	5
5	3	4	1	0	0	3	7
6	2	3	4	3	5	0	1
7	4	2	3	9	4	0	0

Suppose that the new jobs J_1 and J_3 are identical to the jobs which have been completed on processor P_1 with m_1 and m_2, P_2 with m_3 and m_4, respectively. We suppose that the cost of resource change is a constant, \$2. The cost for job type changeover is a variable. Then the cost for changing from J_i to J_j is expressed as the sum of grade change $C_{g_{ij}}$ and resource change C_r

$$C_{ij} = C_{g_{ij}} + C_r$$

The cost matrix in Table 4.6 is completed by the above expression. The optimal makespan of the schedule will be $\sum_{j=1}^{7} T_{j/s} = 40/4 = 10$ execution time units. In minimum changeover cost scheduling, we take advantage of the fact that if we assign a new job to the machines and processors which have just completed the same or similar type of jobs with the same resource, then there will be little cost involved in the job changeover. Schedule L_1 (Figure 4.7) is constructed in this way with job J_1 and job J_3 assigned to p_1 and p_2, respectively. The rest of the jobs are assigned pairwise to the machines. We obtain a feasible schedule with 18 cost units and a makespan of 10 execution time units.



Total cost for schedule L_1 is $C_{m_1} + C_{m_2} + C_{m_3} + C_{m_4} = 18$

Figure 4.7 A feasible schedule and its total changeover cost when each job is split over pairwise on a processor.

L_1 can be represented by a processor Gantt Chart as shown in Figure 4.8.

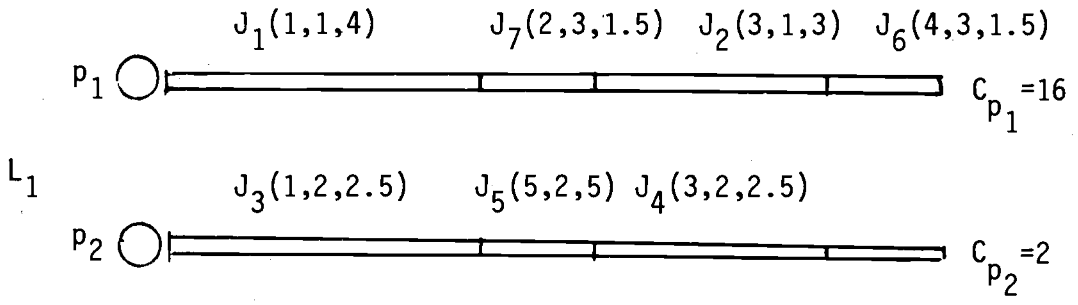


Figure 4.8 A processor Gantt Chart for L_1

If we examine the schedule L_1, we shall notice that the jobs on m_3 and m_4 use the same type of resource. Job J_1 and J_2 also use the same type of resource on m_1 and m_2. We are interested in finding a permutation schedule which is lower in cost than the old schedule without increasing the makespan. Referring to the cost matrix in Table 4.6, we can intuitively see that if jobs 3 and 4 are interchanged with job 5 on m_4 and m_3, and job J_1 is split to machines m_1 and m_2, then we obtain a better schedule with the total cost of 17 units (Figure 4.9).

In Figure 4.10, schedule L_3 has the same cost as L_2 with a makespan of 11. However, it is regarded as an infeasible schedule because the idle time exists on machine m_2. In Figure 4.11, the schedule L_4 is also infeasible because P_1 can not be used as two resources to perform jobs J_2 and J_7 or job J_2 and job J_6 at the same time.

The graphical representation shows that finding an optimal schedule is very difficult, especially when the number of resources and the number of jobs increase. In fact, just for one machine and one processor with 20 jobs available, there will be $20! = 2.45 \times 10^{18}$ possible

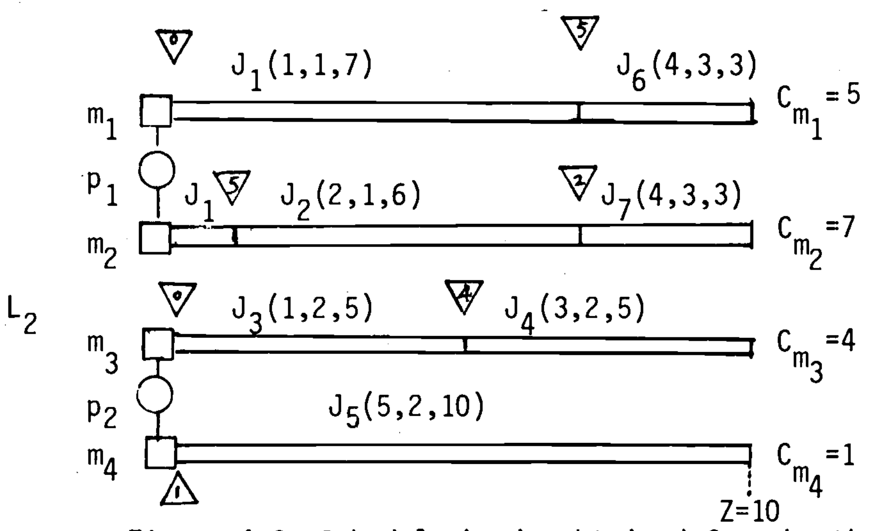


Figure 4.9 Schedule L_2 is obtained from L_1 through jobs permutation.

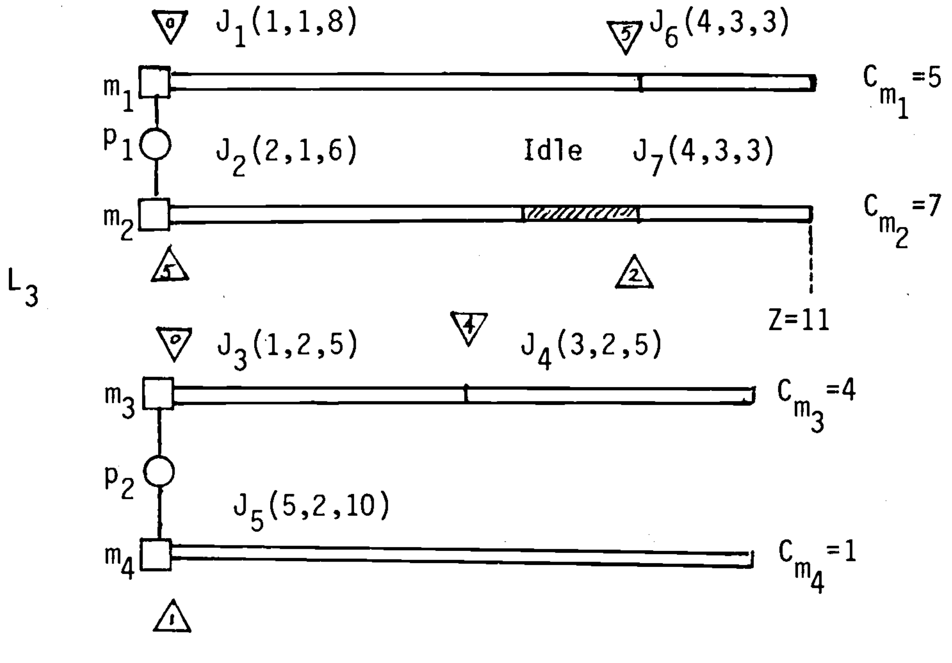


Figure 4.10 A least cost job sequencing that is infeasible because of idleness.

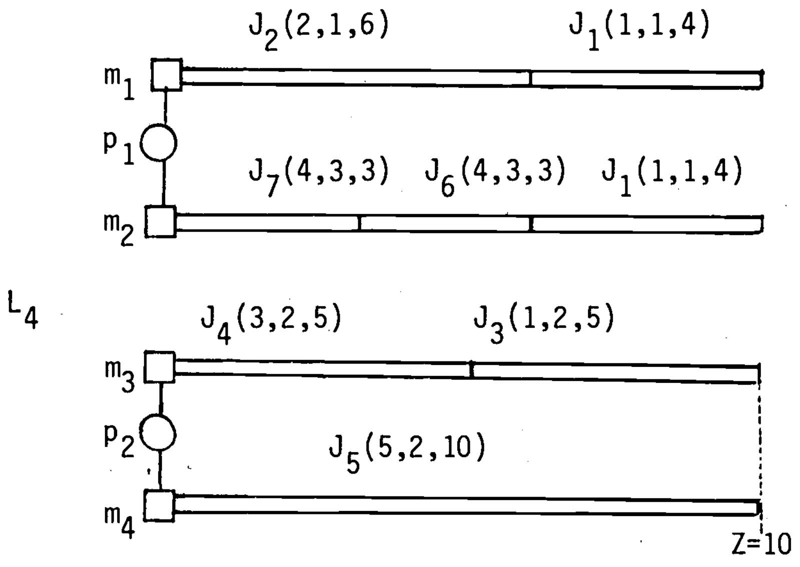


Figure 4.11 An infeasible schedule due to resource conflict.

solutions. If a computer is used to evaluate one solution every microsecond, it would take more than 76,000 years to try all possibilities. Horowitz and Sahni (1974) showed that a makespan minimization on ℓ processors with n variable processing time tasks scheduling will require an enumeration of ℓ^n possible schedules.

Based on the characteristics of the MP, three algorithms are developed. They will be described in the next two sections.

4.2 Two Heuristic Algorithms

From the last section, we have shown that the MP can be solved by intuitive judgments which are hard to program on a computer. The overall strategy of the solution methodology presented here is to obtain a locally feasible and optimal schedule with a minimum amount of computation. There are two stages in solving the problem. The first stage is to construct an initial feasible schedule. The second stage is to modify the initial schedule by applying a series of pairwise interchanges of those jobs which use the same type of resource. An improved permutated schedule can be obtained.

Two heuristic algorithms for scheduling immediately available independent n jobs, with ℓ identical processors and s parallel machines, where the objective is to minimize the total changeover cost, are developed by using three priority rules:

(1) Select the job which has the lowest changeover cost. (Purpose: to minimize total production cost.)

(2) Select the job which has the same resource usage as the previous completed job on the machine and processor. (Purpose: to have an improved and near optimal permutated schedule later.)

(3) Select jobs which have the longest processing time. (Purpose: to minimize the makespan.)

These two algorithms are similar to each other. The differences between the first and second algorithm are that the second one has a planning horizon and all the processors have to compare each other before undertaking the job assignment.

We introduce the following additional notations: $J_j(E_j, R_j, T_j, W_j, P_\alpha)$ where:

$$W_j = \begin{cases} 1 \text{ means job j is an old job.} \\ 0 \text{ means job j is a new job.} \end{cases}$$

e.g. (i) $J_1(2,5,0,1,5)$

(ii) $J_3(3,6,2,0,0)$

(i) means job one is an old job and is currently on processor P_5. But the order of this job has just completed, so $T_1=0$.

(ii) means that job 3 is a new job, it needs two execution time units to complete the order.

$P_\alpha(u)$ is the number of machine(s) controlled by P_α where $(1 < u \leq 2)$

e.g. $P_3(2)$ means processor 3 has two machines.

$f_{p_\alpha}(i,k)$ is the least cost for the current job i on processor P_α changes to job j which correspond to the k^th position C_{ij}, of the $[C_{ij}]$; where $f_{p_\alpha}(i,k) =$

$$\text{Min}\left\{C_{ij};\ \text{for } j=1\ldots n\right\}$$

4.2.1 Heuristic algorithm I

Preparation: Observe what type of jobs and resources are currently on each machine and processor. Obtain a cost matrix $[C_{ij}]$ which consists of the changeover costs for the old jobs to the new jobs.

Input: $n,r,\ell,s,P_\alpha(u)$, $[C_{ij}]$, a_k, R_j, $J_j(E_j,R_j,T_j,W_j,P_\alpha)$.

where $(1 \leq i \leq n,\ 1 \leq j \leq n,\ 1 \leq k \leq r,\ 1 \leq \alpha \leq \ell,\ 1 \leq u \leq 2)$.

Output: Schedule L, Q_{p_α}, $Z(\alpha)$, $Z(\beta)$, where $(1 \leq \beta \leq s)$

C_{p_α} // Cumulative total cost for P_α //

and $\sum_{\alpha=1}^{\ell} C_{p_\alpha}$ // Total changeover cost on all processors and machines //

Step 0: (Initialization). Initialize storage arrays for job attributes in all processors and machines.

Step 1: (Previous unfinished job assignment). Assign all the old jobs to each processor. Update the current $Z(\alpha)$, set $C_{p_\alpha}=0$ for $\alpha = 1$ to ℓ. Set the corresponding columms in the matrix where the jobs are assigned to infinity. Decrement the resource count.

Step 2: (Select a processor). Find P_α which has the shortest current makespan $Z(\alpha)$. Increment the previous resource count on P_α by one.

Step 3: (Select a job). With the current J_i in P_α, select a job J_j such that C_{ij} is the minimum. i.e., $f_{p_\alpha}(i,k) = \min \left\{C_{ij}, \text{for } j=1\ldots.n\right\}$. If $C_{ij} = \infty$ then go to step 6. If there is a tie, choose the job which has a same type of resource usage as the previous job J_i. If the above criterion fails, choose the job which has the longest processing time (T_j). Check whether the resource which is going to be used by J_j is available. If the resource is available, go to step 5; otherwise, go to the next step.

Step 4: (Select the next best job). Find the next job which has the second lowest cost of change. If there is a tie, apply the same criteria as in step 3. Check for resource availability, if there is no resource conflict, then go to the next step; otherwise, repeat Step 4 until a job can be assigned without resource conflict. If no job can be scheduled, then print "not all jobs can be scheduled." Call exit.

Step 5: (Decrement resource count). Assign J_j to P_α , decrement resource count of a_k which is used by J_j by one. Set the column of the $[C_{ij}]$ corresponding to J_j to infinity. Update Q_{p_α} , $Z(\alpha)$ and C_{p_α} . Go to Step 2.

Step 6. (Termination). When $C_{ij} = \infty$, it means that all jobs have been scheduled. Call output to print schedule L. END.

The flowchart for the algorithm is shown in Figure 4.12, the program for the algorithm is shown in the Appendix.

Numerical Example: The foreman of a job shop has received 11 jobs for the next month production. The last jobs in this month being scheduled to be processed on processors P_1 and P_2 are job J_1 and job J_6. Suppose that J_3 had been finished by P_3. In what order may the 11 jobs be scheduled for production so that the total set-up cost for all the machines and processors is minimized? The job attributes, processors, machines and resources availability are given in Table 4.7a and the cost matrix is in Table 4.7b.

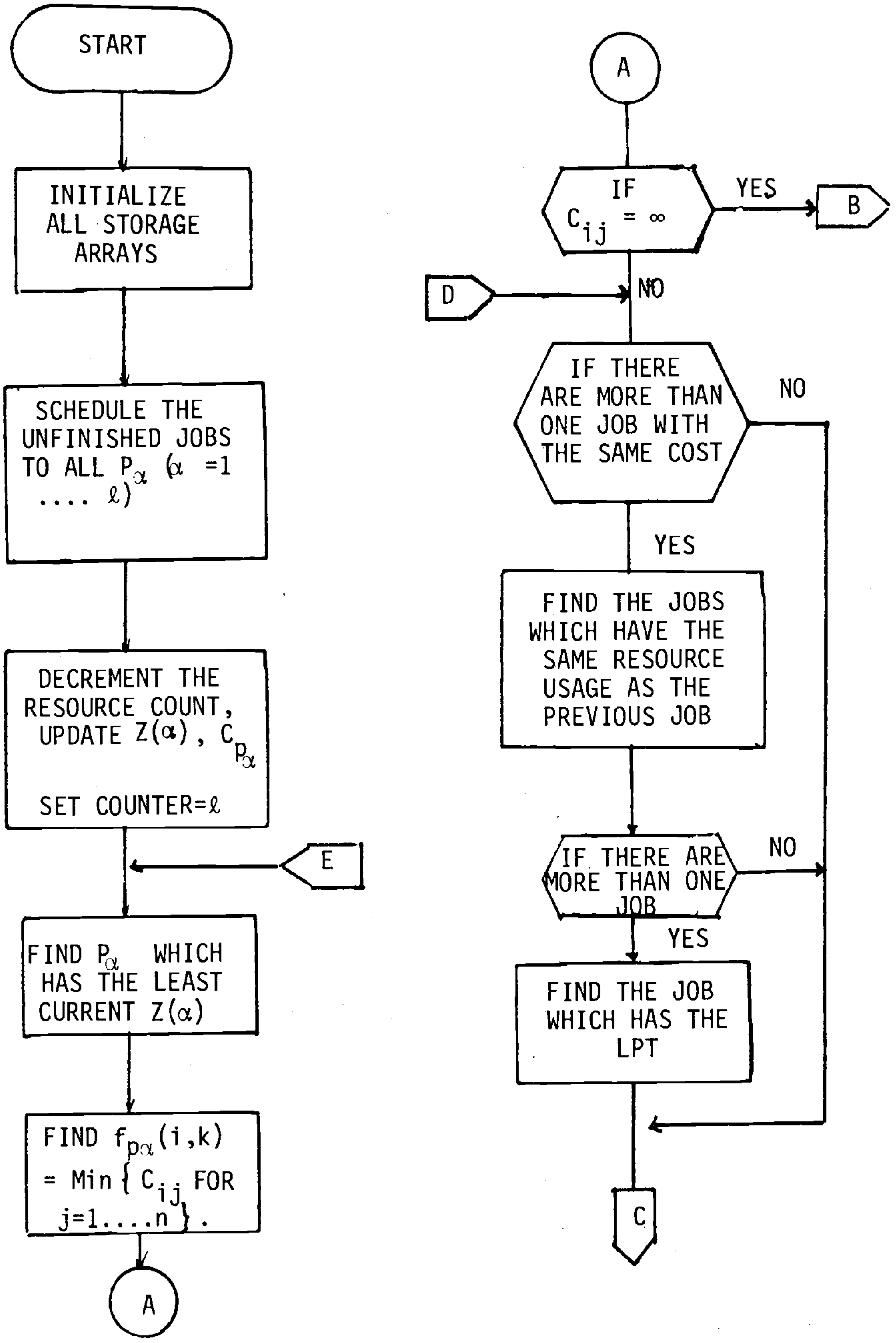

START
INITIALIZE ALL STORAGE ARRAYS
SCHEDULE THE UNFINISHED JOBS TO ALL P_α (α =1 .... ℓ)
DECREMENT THE RESOURCE COUNT, UPDATE $Z(\alpha)$, C_{p_α}
SET COUNTER=ℓ
E
FIND P_α WHICH HAS THE LEAST CURRENT $Z(\alpha)$
FIND $f_{p\alpha}(i,k) = \text{Min}\{C_{ij}$ FOR $j=1....n\}$.
A
A
IF $C_{ij} = \infty$
YES
B
D
NO
IF THERE ARE MORE THAN ONE JOB WITH THE SAME COST
NO
YES
FIND THE JOBS WHICH HAVE THE SAME RESOURCE USAGE AS THE PREVIOUS JOB
IF THERE ARE MORE THAN ONE JOB
NO
YES
FIND THE JOB WHICH HAS THE LPT
C

Figure 4.12 Flow Chart of Heuristic Algorithm I.

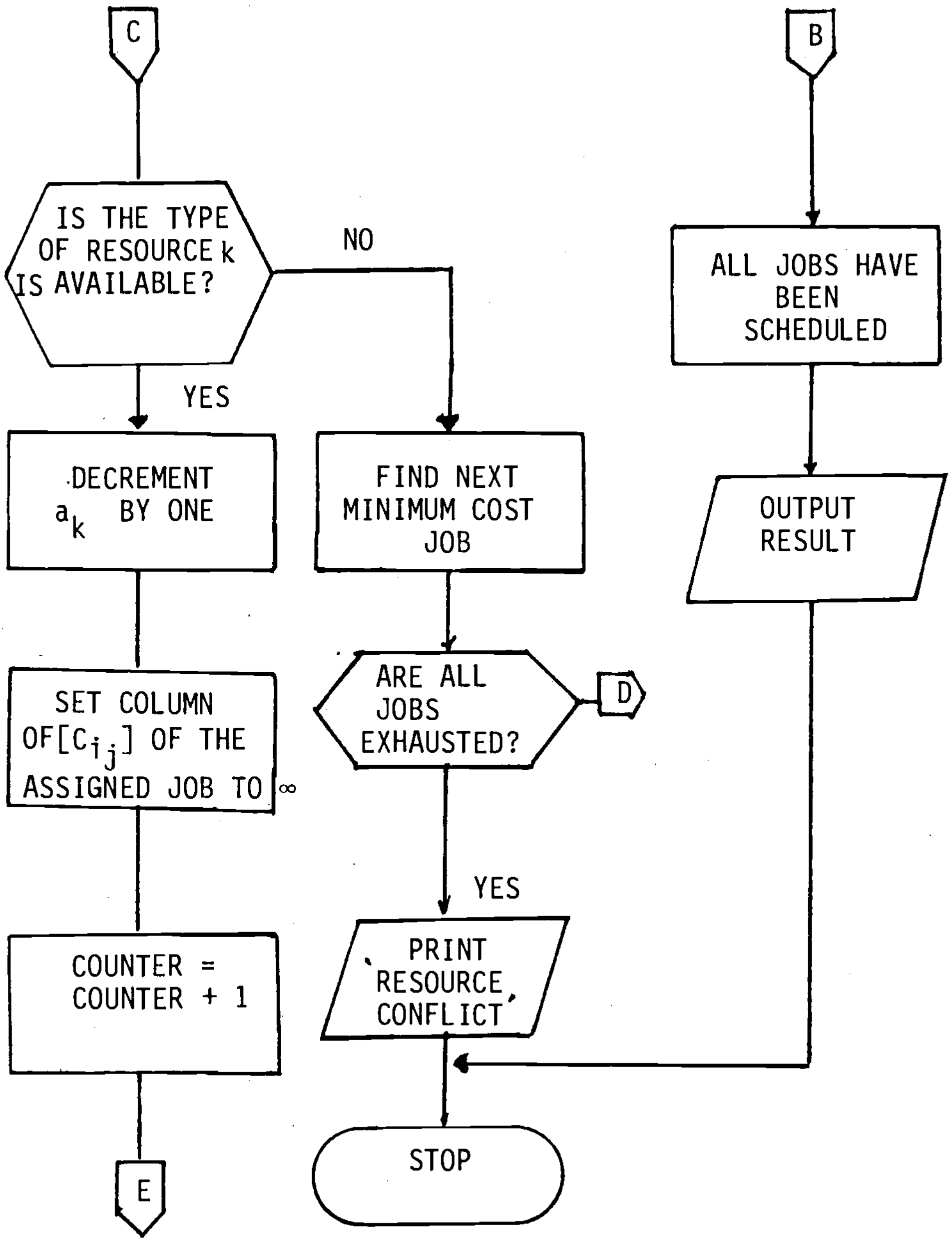

C
IS THE TYPE OF RESOURCE k IS AVAILABLE?
NO
YES
DECREMENT a_k BY ONE
SET COLUMN OF[C_{ij}] OF THE ASSIGNED JOB TO ∞
COUNTER = COUNTER + 1
E
FIND NEXT MINIMUM COST JOB
ARE ALL JOBS EXHAUSTED?
D
YES
PRINT 'RESOURCE CONFLICT'
STOP
B
ALL JOBS HAVE BEEN SCHEDULED
OUTPUT RESULT

Figure 4.12 (Continued)

Table 4.7a Jobs list, processors, machines and resources availability

Job Description

Legend: J_j (E_j, R_j, T_j, W_j, P_α)

J_1(1, 1, 8, 1, 1)

J_2(2, 1, 6, 0, 0)

J_3(1, 2, 0, 1, 3)

J_4(3, 2, 5, 0, 0)

J_5(5, 2, 10, 0, 0)

J_6(4, 3, 3, 1, 2)

J_7(2, 3, 3, 0, 0)

J_8(6, 5, 4, 0, 0)

J_9(7, 4, 4, 0, 0)

J_{10}(6, 4, 7, 0, 0)

J_{11}(8, 5, 2, 0, 0)

J_{12}(4, 1, 3, 0, 0)

J_{13}(9, 4, 4, 0, 0)

J_{14}(6, 3, 6, 0, 0)

Resources availability

r=5

R_j=k	a_k
1	1
2	1
3	1
4	1
5	1

Processors and machines status

p_1 has 2 machines: p_1(2)

p_2 has 2 machines: p_2(2)

p_3 has 1 machine: p_3(1)

ℓ=3 s=5

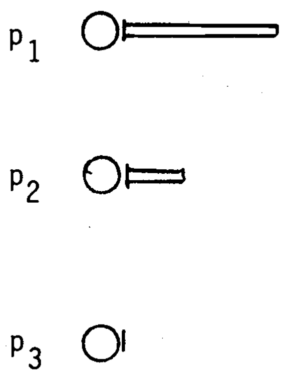
Table 4.7b Cost Matrix

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
(1)	999	5	2	4	6	5	3	3	9	4	3	1	2	3
(2)	1	999	3	10	7	3	2	4	3	3	3	1	3	2
(3)	2	4	999	4	1	3	6	2	3	6	2	9	2	2
(4)	9	3	0	999	1	4	5	2	3	2	3	6	2	3
(5)	3	4	1	0	999	3	7	6	2	8	11	3	13	4
(6)	2	3	4	3	5	999	1	7	6	2	13	2	3	1
(7)	4	2	3	9	4	0	999	2	4	5	7	2	2	0
(8)	5	10	4	6	10	2	7	999	2	2	4	2	8	2
(9)	3	4	2	7	11	6	10	5	999	8	2	6	0	11
(10)	2	11	5	2	4	8	2	2	0	999	5	3	6	2
(11)	4	2	7	12	2	11	16	0	7	9	999	4	6	0
(12)	0	0	4	13	2	2	7	11	3	2	3	999	3	2
(13)	2	4	2	5	8	6	3	2	1	0	2	10	999	8
(14)	8	9	5	8	3	0	4	2	8	2	2	8	4	999

Solution procedure

The (14 x 14) cost matrix in Table 4.7b consists of the previous scheduled jobs, J_1, J_3 and J_6. There is only one job of each type and resource hence we cannot process J_i after processing J_i. This is avoided by setting $C_{ii} = \infty$.

We know that no preemptive priorities are allowed. All the current unfinished jobs on each processor and machine have to be continued to be processed in order that no cost will be involved at time zero of the new schedule. Therefore, J_1, J_3 and J_6 have to be assigned to p_1, p_3 and p_2, respectively. We get the cost matrix in Table 4.8 and processors Gantt Chart (Figure 4.3) after the previous scheduled jobs assignment in step 1 of algorithm I.



$Q_{p_1} = \{J_1\}$ $Z(1) = 4$ $C_{p_1} = 0$

$Q_{p_2} = \{J_6\}$ $Z(2) = 1.5$ $C_{p_2} = 0$

$Q_{p_3} = \{J_3\}$ $Z(3) = 0$ $C_{p_3} = 0$

Figure 4.13 Processors Gantt Chart after J_1, J_3 and J_6 have been assigned.

Table 4.8 The cost matrix after the previous jobs have been assigned.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	∞	5	∞	4	6	∞	3	3	9	4	3	1	2	3
2	∞	∞	∞	10	7	∞	2	4	3	3	3	1	3	2
3	∞	4	∞	4	1	∞	6	2	3	6	2	9	2	2
4	∞	3	∞	∞	1	∞	5	2	3	2	3	6	2	3
5	∞	4	∞	0	∞	∞	7	6	2	8	11	3	13	4
6	∞	3	∞	3	5	∞	1	7	6	2	13	2	3	1
7	∞	2	∞	9	4	∞	∞	2	4	5	7	2	2	0
8	∞	10	∞	6	10	∞	7	∞	2	2	4	2	8	2
9	∞	4	∞	7	11	∞	10	5	∞	8	2	6	0	11
10	∞	11	∞	2	4	∞	2	2	0	∞	5	3	6	2
11	∞	2	∞	12	2	∞	16	0	7	9	∞	4	6	0
12	∞	0	∞	13	2	∞	7	11	3	2	3	∞	3	2
13	∞	4	∞	5	8	∞	3	2	1	0	2	10	∞	8
14	∞	9	∞	8	3	∞	4	2	8	2	2	8	4	∞

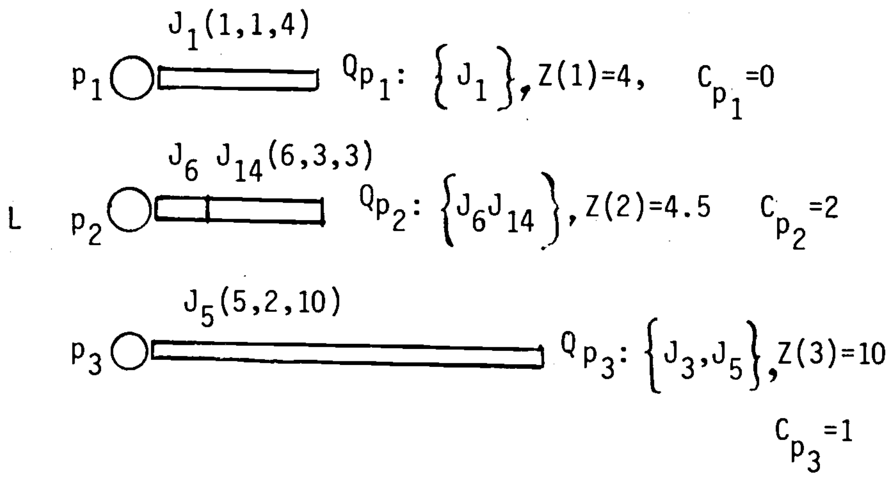
p_3 now has the shortest current Z, therefore p_3 gets the next job assignment. The minimum changeover cost from J_3 to any other job is expressed as $f_{p_3}(3,k) = \min\left\{ C_{3,j}; j=1...n \right\}$ and $f_{p_3}(3,5) = C_{3,5}=1$. This means that J_3 changes to J_5 incurs the minimum cost. Since p_3 has one machine only, the execution time unit for J_5 will be $(T_5/p_1(1) = 10)$. There is no resource conflict for resource type 2, so J_5 is assigned to p_3. C_{i5} is set to ∞ for $(i=1...n)$. We have a partial schedule with

$$Q_{p_1} : \{J_1\}, \quad Z(1) = 4, \quad C_{p_1} = 0$$

$$Q_{p2} : \{J_6\}, \quad Z(2) = 1.5, \quad C_{p_2} = 0$$

$$Q_{p_3} : \{J_3, J_5\}, \quad Z(3) = 10, \quad C_{p_3} = 1$$

Among the three processors, p_2 has the shortest Z, so this time p_2 gets the next job assignment. $f_{p_2}(6,k) = \min\left\{C_{6j};\ j=1\ldots n\right\} = C_{6,7} = C_{6,14} = 1$ cost unit. There is a tie. Both J_7 and J_{14} use the same resource type 3, but $T_{14} > T_7$, therefore, without resource conflict, J_{14} is assigned to p_2 with $f_{p_2}(6,14) = 1$. The execution time is 3 units. The cost matrix is shown in Table 4.9 with $C_{i,14} = \infty$ (for i=1..n). The processor Gantt Chart is shown in Figure 4.14.



$J_1(1,1,4)$
p_1
$Q_{p_1}: \{J_1\}, Z(1)=4, \quad C_{p_1}=0$
$J_6 \; J_{14}(6,3,3)$
L
p_2
$Q_{p_2}: \{J_6 J_{14}\}, Z(2)=4.5 \quad C_{p_2}=2$
$J_5(5,2,10)$
p_3
$Q_{p_3}: \{J_3, J_5\}, Z(3)=10$
$C_{p_3}=1$

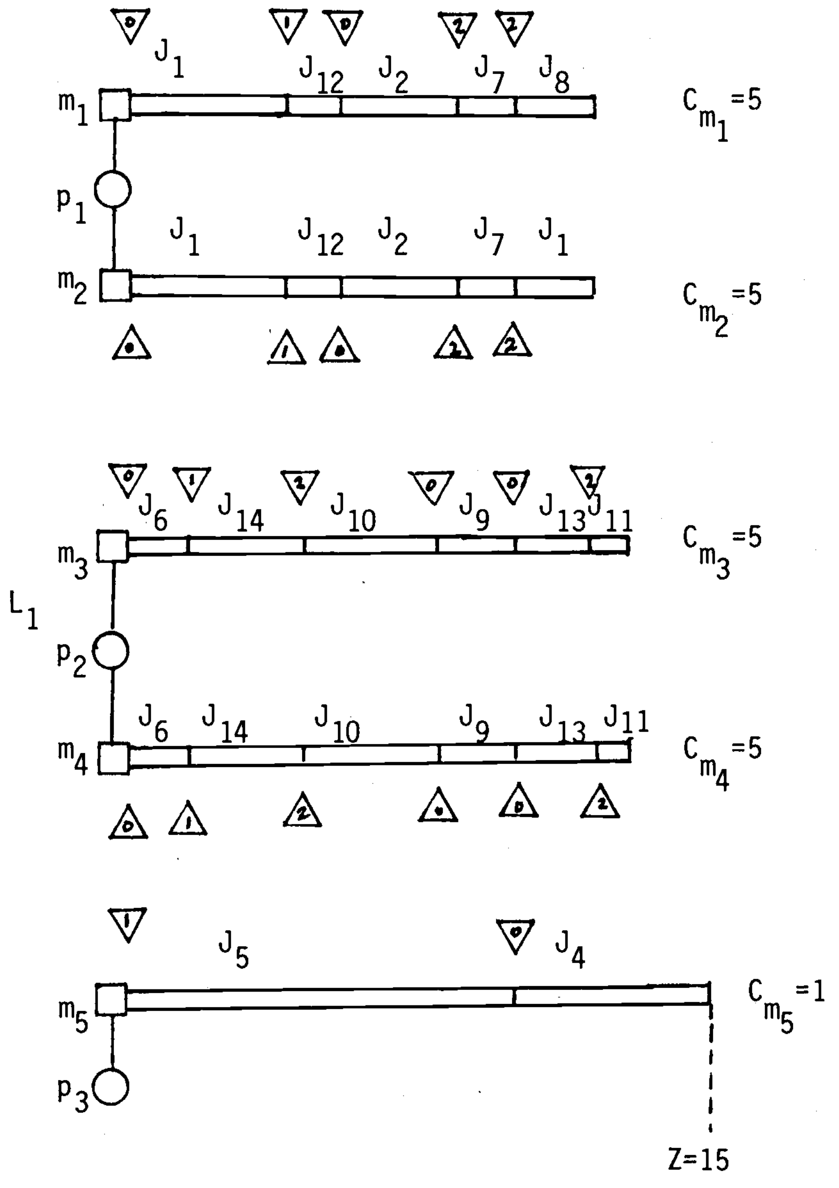
Figure 4.14 A partial schedule L.

Table 4.9 The cost matrix after the 5th job has been assigned.

	2	4	7	8	9	10	11	12	13	14
1	5	4	3	3	9	4	3	1	2	∞
2	∞	10	2	4	3	3	3	1	3	∞
3	4	4	6	2	3	6	2	9	2	∞
4	3	∞	5	2	3	2	3	6	2	∞
5	4	0	7	6	2	8	11	3	13	∞
6	3	3	1	7	6	2	13	2	3	∞
7	2	9	∞	2	4	5	7	2	2	∞
8	10	6	7	∞	2	2	4	2	8	∞
9	4	7	10	5	∞	8	2	6	0	∞
10	11	2	2	2	0	∞	3	6	0	∞
11	2	12	16	0	7	9	∞	4	6	∞
12	0	13	7	11	3	2	3	∞	3	∞
13	4	5	3	2	1	0	2	10	∞	∞
14	9	8	4	2	8	2	2	8	4	∞

This procedure is repeated until all jobs have been assigned. We then obtain schedule L_1 which is shown in Figure 4.15. The total changeover cost is 21 cost units. The makespan is 15 execution time units.

An attempt was made to pairwise interchange those jobs which use the same resource and are adjacent to each other. We cannot produce



Total cost = 21

Figure 4.15 Result of the schedule L_1 when algorithm I is used

a better schedule than schedule L_1 at this time, and this schedule is said to be locally optimal in this sequence.

4.2.2 Heuristic Algorithm II

Algorithm II has only a few changes from Algorithm I. We shall state the differences.

In order to make the algorithm clear, we add one additional notation.

Let $G_y(p_\alpha(k))$ be the least cost of job k to be assigned to P_α after y jobs have been assigned.

and $G_y(P_\alpha(k)) = \min\left\{f_{p_\alpha}(i,k);\ \text{for}\ \alpha = 1 \ldots \ell\right\}$

<u>Preparation</u>: The same as in algorithm I.

Input: The same. Add the planning horizon D.

Output: The same.

Steps 1, 2, 5, 6: same.

Step 0: (Compute the optimal makespan and initialization)

$$Z^* = \sum_{j=1}^{n} T_j/s$$

If $D \leq Z^*$ then print "not all jobs can be scheduled within the time limit of D." Call exit. Otherwise, initialize storage arrays.

Step 3: (Select a job for each processor)

With the current job J_i in each p_α $(\alpha = 1 \ldots \ell)$, find the least cost job J_j.

$$f_{p_\alpha}(i,k) = \min \left\{ C_{i,j};\ \text{for } j=1.....n \right\}$$

If there are jobs with the same minimum cost for a given p_α, break the tie by using the priority rules (jobs with the same resource type and LPT is scheduled first) as in algorithm I. Go to next step.

Step 4: (Assign a job to a processor)

Find the minimum cost for each processor; i.e.

$$G_y(p_\alpha(k)) = \min \left\{ f_{p_\alpha}(i,k) \quad \alpha = 1....\ell \right\} .$$

If there is a tie, break the tie arbitrarily.

Check whether the type of resource which is going to be used by J_j is available and the cumulative processing time (Z) of p_α is less than the planning horizon D. If it is true, go to next step; otherwise mark the J_j which can not be processed by p_α at that particular time. Go to step 3 to find next job. If no jobs can be assigned to any of the p_α, print message and go to step 6 to print out the partial schedule.

We do not exhibit the flow chart for this algorithm, because it is easily constructed from algorithm I. The program for this algorithm is shown in the Appendix.

Numerical Example: We use the same example in algorithm I to illustrate how the algorithm works. We add the planning horizon D to be 15.5 execution time units.

Solution Procedure

Step 0: $Z^* = \sum_{j=1}^{n} T_j/s = 65/5 = 13$

Since $D > Z^*$, there may be a schedule existing such that all jobs can be finished at or before the time limit D.

Steps 1 & 2: J_1, J_6 and J_3 are assigned to p_1, p_3 and p_2, respectively. This is the same as in algorithm I. The processors Gantt chart and matrix are shown in Figure 4.13 and Table 4.8.

Step 3: y=3 (3 jobs have been assigned.) Iteration 1: (a) Find the least cost job for each processor.

$f_{p_1}(1,k) = \min \left\{ C_{1,j} \text{ ; for } j = 1 \ldots n \right\}$

$= C_{1,12} = 1$

$\therefore f_{p_1}(1,12) = 1$

$f_{p_2}(6,k) = \min \left\{ C_{6,j} \text{ ; for } j = 1 \ldots n \right\}$

$= C_{6,7} = 1 \text{ ; } C_{6,14} = 1$

J_{14} has the LPT. So J_{14} is selected $\therefore f_{p_2}(6,14) = 1$

$f_{p_3}(3,k) = \min \left\{ C_{3,j} \text{ ; for } j = 1 \ldots n \right\}$

$= C_{3,5} = 1$

$\therefore f_{p_3}(3,5) = 1$

(b) find the best assignment for a job to a particular p_α

Step 4: $G_3\ (P_\alpha(k)) = \min\left\{P_1(12) = 1,\ P_2(14) = 1,\ P_3(5) = 1\right\}$

There is a tie. Although J_{12}, J_{14} and J_5 have the same resource type as the previous jobs, and $T_5 > T_{14} > T_{12}$, we break the tie arbitrarily. J_5 is chosen.

Before assigning J_5 to P_3, check whether the resource type 2 is available and the current makespan of Z(3) is less than D.

$\because$ Resource type (k=2) is available and $Z(3)=10 < D$.

J_5 is assigned to P_3, set $C_{i5} = \infty$ (for i=1...n)

Decrement resource amount a_k by 1 with a duration of 10 execution time units.

$y = y+1$. Go to Step 3.

Iteration 2. (repeat Step 3 and Step 4)

(a) find a least cost job

$$f_{p_1}\ (1,k) = \min\left\{C_{1,j};\ j = 1 \ldots n\right\}$$

$$= C_{1,12} = 1 \quad \therefore\ k = 12$$

$$f_{p_2}\ (6,k) = \min\left\{C_{6,j};\ j = 1 \ldots n\right\}$$

$$= C_{6,14} = 1\ \therefore\ k = 14$$

$$f_{p_3}\ (5,k) = \min\left\{C_{5,j};\ j = 1 \ldots n\right\}$$

$$= C_{5,4} = 0 \quad \therefore\ k = 4$$

(b) find the best assignment

$$G_4(P_{\alpha}(k)) = \min\left\{P_1\ (1,12) = 1,\ P_2\ (6,14) = 1,\ P_3\ (5,4) = 0\right\}$$

$\therefore$ $G_4\ (P_3(4)) = 0$ is the lowest cost.

So J_4 is assigned to P_3 after checking

R_2 is available and $Z(3) = 10 + 5 < D$. Set $C_{i4} = \infty$

now

$Q_{p_1} = \{J_1\}$, $Z(1) = 4$ $C_{p_1} = 0$

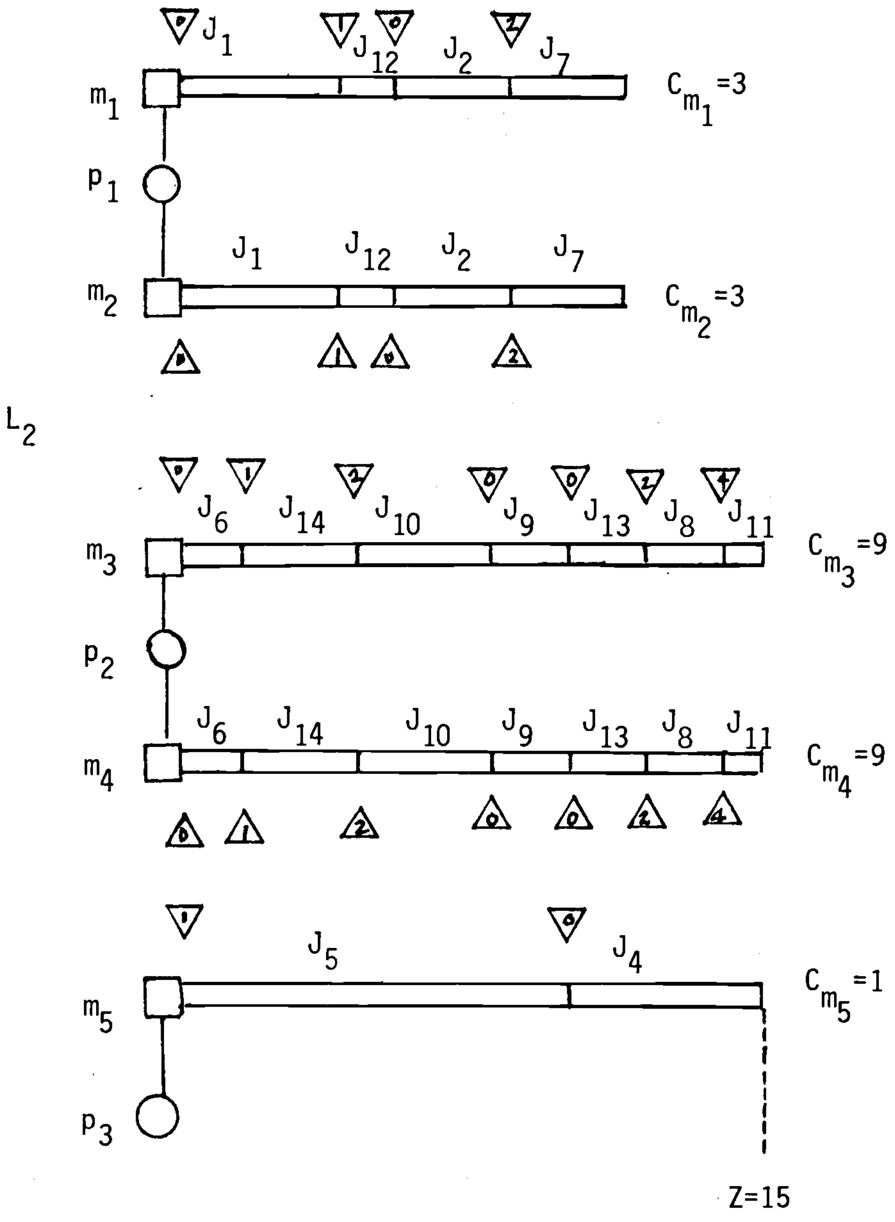
$Q_{p_2} = \{J_6\}$, $Z(2) = 1.5$ $C_{p_2} = 0$

$Q_{p_3} = \{J_3, J_5, J_4\}$, $Z(3) = 15$ $C_{p_3} = 1$

Iteration 3 $y = 5$; $G_5\ (P_2(14))$ Set $C_{i,14} = \infty$

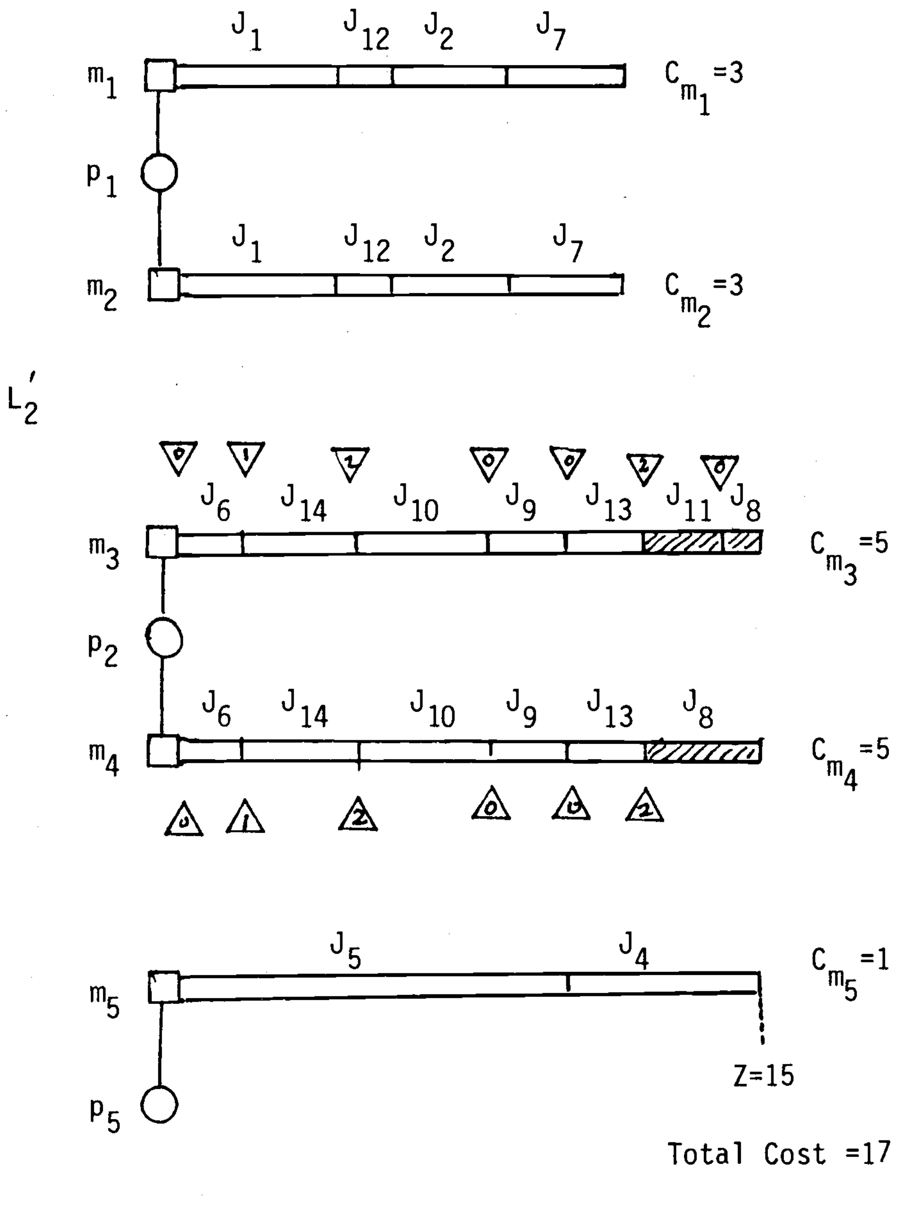
Iteration 4 $y = 6$; $G_6\ (P_1(12))$ Set $C_{i,12} = \infty$

After $(n-\ell)$ iterations, all jobs have been assigned. The machines' Gantt chart is shown in Figure 4.16. The total cost for schedule L_2 is 25 cost units. The makespan Z is 15. In this example, the result of algorithm II is worse than algorithm I, due mainly to the choice of the last job J_{11}. This also illustrates how a heuristic algorithm can lead to poor decisions toward the end of sequence. However, if we interchange J_{11} with J_8, we get a much better schedule L_2' with a total cost of 17 units; the processor's Gantt chart is shown in Figure 4.17.



Total cost = 25

Figure 4.16 Algorithm II produces an initial schedule L_2



▨ jobs have been permutated

Figure 4.17 An improved schedule L'_2 from schedule L_2

4.23 Discussion

The computational experience of these two algorithms is given in section 5.4.

These two algorithms suffer two disadvantages.

(i) If there is only one processor controlling two machines, and each job has a different resource type requirement, then all jobs have to be split over two machines in order to satisfy the constraints stated in Corollary 4.2.

(ii) Both algorithms will fail when:

(a) One type of resource is used by many jobs and its amount of availability is limited (i.e., "saturated").

(b) The number of processors increases, the makespan becomes shorter, this will make the resource availability of each type tighter.

The branch-and-bound algorithm discussed in the next chapter will alleviate the above difficulties.

CHAPTER V

APPLICATION OF BIN PACKING AND BRANCH AND BOUND ALGORITHMS TO MULTIPROCESSOR SCHEDULING - THE THIRD ALGORITHM

5.1 Introduction

Since the subproblems of the MP problem are NP-complete, it is hard to find an optimal solution for medium size of jobs in a reasonable time of computation. Here we present an algorithm based on the common-sense philosophy that a complex problem may be decomposed into several less complex problems. If there are several algorithms which exist to solve the subproblems of the complex problem, then these algorithms may be combined together to form a new algorithm which may be bounded by addition, multiplication and composition of the complexities of its component algorithms.

The philosophy for the third algorithm developed can be summarized by three main points:

(1) To design a procedure for partitioning n jobs into mutually exclusive subsets called classes.

(2) To design a procedure for specifying a sequence and priority of the classes.

(3) To design a procedure for sequencing and packing jobs into bins within each class.

The proposed algorithm is abbreviated BINBAB (Bin Packing and Branch and Bound methods.)

This chapter contains a review of existing heuristics and branch and bound algorithms for solving the least cost scheduling and makespan minimization on multiprocessors. BINBAB algorithm is presented and is followed by a numerical example. Finally, comparison of computational results among three algorithms are presented.

5.2 Previous Algorithms

5.2.1 Minimum Cost Sequencing

From the literature review, we noticed that the minimum cost sequencing "routing" problems to which branch and bound algorithms have traditionally been applied were all based on an availability of a single processor (or a single machine, or traveling salesperson). A number of branch and bound algorithms to find the exact solution for small-to-moderate-size traveling salesperson problems (fewer than 50 cities) appeared in the literature during the past 17 years. However, most, if not all, are based on the algorithm by Eastman (1959) or Little *et al*. (1963, p. 972). The work of Little, *et al*. is a tour-building algorithm, while the work of Eastman is subtour elimination algorithms. However, the former may be considered a modification of the branching and bounding procedure used by Eastman. The Eastman algorithm is extended by Shapiro and the computational experience of his algorithm makes using Little's algorithm less desirable (Bellmore and Nemhauser, 1968, p. 550). Ramalingam (1969, p. 81) showed how to modify Little *et al*.'s algorithm for solving sequencing problems with nonrepetitive jobs.

Bellmore and Hong (1974) used graph theory to show that a multisalesmen problem can be transformed to a single traveling salesman problem. The multisalesmen problem can be stated as follows. Given m salesmen who are required to visit n "customer cities" from a "base city" and return to the base city with a minimum total distance (or cost) traveled incurred by all salesmen. Each city must be visited exactly once by exactly one of the m salesmen. Thus the multisalesmen problem is as hard as the single salesmen problem. In fact, if m=1 then the problem is reduced to a standard traveling salesman problem.

Svestka and Huckfeldt (1973, p. 798) presented a generalization algorithm to the multisalesmen case. Their branch and bound scheme was based on the Bellmore and Malone Model (1971, p. 278) and it is of a subtour elimination type. Their computational experience showed that the multisalesmen in fact is faster in computation time than the single salesman. They observed that the minimum computation time occurs when the integer [n/m] lies between three to seven. However, their algorithm can not be applied to the MP, because their algorithm produces closed sub-tours and the length of each tour for each salesman is not considered. The running time for their algorithm is worth noting. They claimed that for m=1, their algorithm execution time is $t=e^{0.074\,n}$ while Little _et_ _al_.'s algorithm is $t=e^{0.115\,n}$ where n is the number of cities.

The author observed that no algorithm has been reported on scheduling n independent jobs with variable execution time on multiprocessors where the objective is to minimize the changeover cost.

5.2.2 Makespan Minimization

The bin packing problem is similar to the problem of makespan minimization of identical parallel processors problem. The bin packing problem can be described as follows (Horowitz and Sahni, 1978, p. 572):

> If we are given n objects which have to be placed in bins of equal capacity L. Object i requires ℓ_i units of bin capacity. The objective is to determine the minimum number of bins needed to accommodate all n objects. No object may be placed in one bin and partly in another.

Horowitz and Sahni also showed that the bin packing problem is NP-hard (1978, p. 573). They stated four simple known heuristic algorithms to solve it. They are:

(i) First Fit (FF)

(ii) Best Fit (BF)

(iii) First Fit Decreasing (FFD)

(iv) Best Fit Decreasing (BFD)

The LPT algorithm can be applied to solve the bin packing problem. It has been described in section 4.1.1.

Coffman, et al. (1978, p. 1) introduced a comparably fast procedure named MULTIFIT (Multiple fit) algorithm which is based on the First Fit Decreasing (FFD) bin packing technique to solve the multiprocessor scheduling problem. The basic algorithm is as follows:

(i) Construct an LPT ordering of jobs.

(ii) Start with known upper and lower bounds on the makespan

Z, and at each step come up with a value, D, midway between the current upper and lower bounds.

(iii) Schedule the jobs in order, each time assigning a job to the lowest index processor without violating the deadline D.

(iv) If all jobs are assigned such that the load on each processor P_α, $Z_{p_\alpha} \leq D$, then we succeed in constructing a schedule with a makespan

$$Z = \max_{p_\alpha} Z_{p_\alpha}$$

and D becomes the new upper bound. If necessary, go to (ii) to start another interation.

(v) Otherwise D becomes the new lower bound (we have not obtained a complete schedule yet) and go to (ii) to start another iteration.

(vi) Stop when the desired number of iterations is completed. At each iteration, the potential range is halved, and a good makespan value is approximated very rapidly.

The authors proved that the MULTIFIT algorithm satisfies the worst-case performance bound of 1.22. This is precisely the best possible bound for the algorithm when $m \leq 7$. (Where m is number of processors). Coffman, et al. (1978, p. 1), conjectured that the best possible general bound for their algorithm is 20/17.

Elmaghraby and Elimam (1980, p. 94) presented a knapsack-based algorithm (KOMP) which requires more computational effort than either LPT or MULTIFIT. However, the efficiency of their multiprocessor's schedule appears to be superior to that of either LPT or MULTIFIT. Their algorithm is quite long. KOMP is based on the simple observation that a two-machine makespan problem is equivalent to a knapsack problem. A "crude" heuristic is used to yield a feasible schedule. The makespan machine teams up with the shortest processing time machine to form a knapsack which is solved to yield a lower makespan. The process is iterated until a good, if not optimal, makespan is reached. They claimed that KOMP yields an optimum schedule most of the time.

5.3 BINBAB Algorithm

KOMP and MULTIFIT algorithms are both effective. They can be applied to MP under the following assumptions:

(i) The previous jobs are not necessary to be scheduled first at the beginning of a scheduling period.

(ii) There are no resource constraints. If this assumption is held, ℓ processors will become resources, we then seek to find a schedule which meets a common deadline D for S identical machines.

BINBAB algorithm can be summarized into three steps. First, the jobs with the same type resource usage are grouped together into classes. This may help to eliminate the resource conflict. Second, each class of jobs is assigned to a processor by using the FFD bin packing techniques, the makespan minimization can be achieved. After the second step, we have a subset of jobs in each processors and they are mutually exclusive ($\omega_1 \cup \omega_2$ ----- $\omega_\ell = J$). Third, each subset of jobs is solved as a single machine case by using the algorithm described by Ramalingam (1968, p. 81) and the branch and bound method by Little et al. (1963, p. 979). We will obtain an optimal sequence of jobs for each processor.

The step-by-step BINBAB algorithm is described as follows:

Preparation:	Same as algorithms I and II.
Input:	Same as algorithms I and II.
Output:	Z^*, $Z(\alpha)$, C_{p_α}, subcost-matrix for each subset of jobs, $Q_{p\alpha}$.

Step 0. (Find the optimal makespan)

$$Z^* = \sum_{j=1}^{n} T_j/s$$

Round off Z^* to its greatest integer. Z^* is the lowest bound of the completion time for the schedule.

Step 1. (Find the height of a stack, h)

Set $h = Z^*$, if there is a processor which has only one machine.

or set $h = 2Z^*$, if there is a tight resource situation and one processor two machines situation.

Step 2. (Sort J into classes). Each class of job needs the same type of resource. The jobs in each class are arranged by a decreasing order of its processing time T_j.

Step 3. (Place jobs into stacks). Put the jobs in each class into a stack with a stack height limitation, h. The unfinished jobs of the previous schedule have to be put in each stack first. The remaining jobs are placed into the stack by using the First Fit Decreasing method (FFD). (Baase, 1978, p. 268). i.e., the longest processing time job is filled in the stack, then find the second longest to fit the remaining stack level. If all jobs in that class are exhausted before the stack is full, name the stack, otherwise continue to put the remaining jobs of that class into a new stack and name the stack. Update the number of stacks. This procedure is applied to all classes of jobs until all jobs have been put into stacks with each stack height less than h. After this step, the total number of stacks is always equal to or greater than the number of resource type (r) available.

Step 4. (Assign previous jobs to processors). Index and treat each processor as a bin. If a processor has two machines, then the processor capacity $B_p = 2Z^*$ otherwise, $B_p = Z^*$. Assign the stacks which have the previous jobs to processors.

Step 5. (Pack each processor with stacks). Arrange the remaining stacks according to their decreasing order of stack height. Apply FFD algorithm again. Afterwards, we would have two cases:

Case 1. There are no more stacks, all stacks have been assigned. Go to step 7.

Case 2. There are stacks left behind. Go to next step.

Step 6. (Assign the remaining stack to processors). Assign the tallest stack to the processor with the biggest amount of remaining capacity B_r until all stacks are assigned to the processors. Go to next step.

Step 7. (Find the optimal sequencing) for $\alpha = 1$ to ℓ.
Sort out the subcost matrix for the jobs in each P_α .
Call branch and bound procedure (BANDB) to find the optimal job sequencing. END.

Procedure BANDB ($[C_{kl}]$, NUM)

$[C_{kl}]$ is the subcost matrix. NUM is number of jobs on the processor P_α.

This section has been lifted from Ramalingam (1968, p. 83).

Step 1. Set $C_{k1} = \infty$, because job 1 is always the job which is left behind at the last schedule.

Step 2. Reduce matrix $[C_{kl}]$ by finding the smallest number in each column and subtract each column with that number. Subtract the smallest number from the first row of the $[C_{kl}]$ only.

Step 3. We obtain a reduced cost matrix $[C_{kl}']$. Let S=1 be the cost of all possible schedules. Label S with

V(S) = sum of reducing constants.

Step 4. For each cell (a,b) with zero cost in the reduced matrix $[C_{kl}']$, compute the cost penalty $(P_{a,b})$ of not using it, where

$$(P_{a,b}) = \min_{k \neq a} [C_{kl}'] + \min_{l \neq b} [C_{kl}']$$

Enter the value of $(P_{a,b})$ in the cell (a,b)

Step 5. Choose a cell (c,d) such that $P_{c,d} = \text{Max}\ (P_{a,b})$

for all a and b values. Ties, if any, may be broken arbitrarily. We branch the set of all possible schedules from S into those that contain the route (c,d) and those that do not. Let us denote these subsets by Y and $\overline{Y}$. Delete row c and column d.

Step 6. The lower bounds for subsets Y and $\overline{Y}$ may be calculated as follows: For the subsets $\overline{Y}$, v $(\overline{Y})$ = v (S) + (Pc,d), determine the starting job s and ending job e of the schedule containing (c,d) among schedules generated by the selected pairs of Y. Record in the matrix $[C_{kl}']$, set $C(e,s) = \infty$. Reduce the matrix $[C_{kl}']$ by columns and the first row only. v (Y) = v (S) + (sum of reducing constants).

Check if the reduced matrix is of size 2 x 2. If yes, complete the single route and continue, otherwise go to next step.

Step 7. Examine the lower bounds of the nodes obtained so far and choose the one with the smallest value.

Step 8. Check if the best schedule found so far has a cost (Z_o). less than or equal to the lower bounds on all terminal nodes of the decision tree. If yes, the sequence established in step 7 is the optimal schedule.

If the lower bound of some other artibrary node X has less value than that of the last node Y, go to step 9. Otherwise, go to step 4.

Step 9. In the original cost matrix $[C_{kl}]$, choose the pairs (c,d) that are previously selected in the route of S. Compute $g = \Sigma C_{c,d}$

For each of (C,d), delete the row c and column d. For each route among the (c,d) group, find the starting jobs s and the ending job e and set $C_{(e,s)} = \infty$. For each $(\overline{e,d})$ that is not included in the schedules of S, set $C_{(\overline{c,d})} = \infty$. Reduce the remaining matrix $[C_{kl}]$ if possible.

The lower bound of X, v (X) = g + sum of reducing constants. Then, go to step 4.

<u>Numerical example</u>: We use the example in Table 4.7 a and b for illustrating how the BINBAB algorithm works.

Solution Procedure:

Step 0. $Z^* = 65/5 = 13$

Step 1. In this example, we set $h= 2Z^*$, because we have a tight resource availability.

Step 2. Sort J into classes, we have five types of resource, therefore we have five classes of jobs. Arrange the job in each class by the decreasing order of T_j.

Class 1

$J_1(1,1,8,1,1) > J_2(2,1,6,0,0) > J_{12}(4,1,3,0,0)$

Class 2

$J_5(5,2,10,0,0) > J_4(3,2,5,0,0) > J_3(1,2,0,1,3)$

Class 3

$J_{14}(6,3,6,0,0) > J_7(2,3,3,0,0) > J_6(4,3,3,1,2)$

Class 4

$J_{10}(6,4,7,0,0) > J_9(7,4,4,0,0) > J_{13}(9,4,4,0,0)$

Class 5

$J_8(6,5,4,0,0) > J_{11}(8,5,2,0,0)$

Step 3. Each job is placed into a stack with the previous unfinished job first. Apply FFD algorithm to stack the remaining jobs. Afterwards, we have five stacks with various stack heights (Figure 5.1). The height of each stack is less than 26.

Step 4. Assign stack 1, stack 3 and stack 2 to processor #1, #2, and #3, respectively. Stack 4 and stack 5 are left behind.

Steps 5 & 6. Since processor #2 has more room in it, $B_{r_2}=26-12=14$. Stack 4 is assigned to processor #2 and stack 5 is assigned to processor #1. We have all processors filled with jobs. (Figure 5.2).

Step 7. Sort out the sub-cost matrix for the jobs in P_1. It is shown in Table 5.1. Call procedure BANDB.

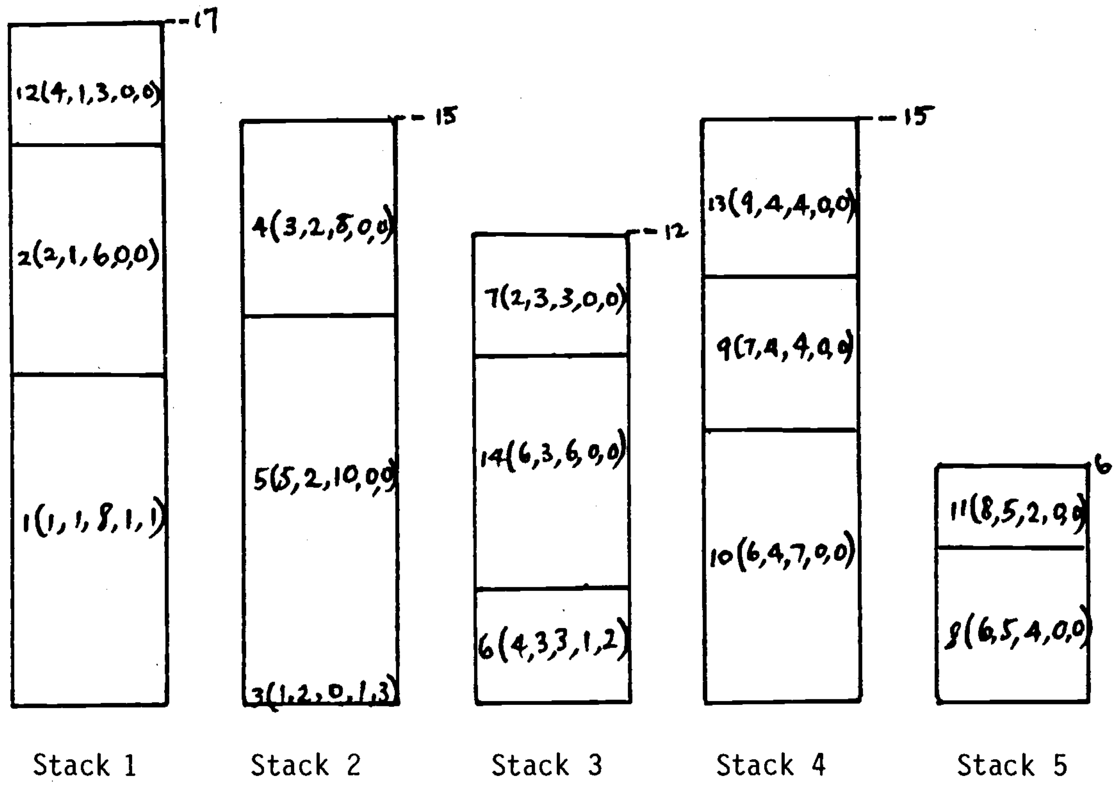
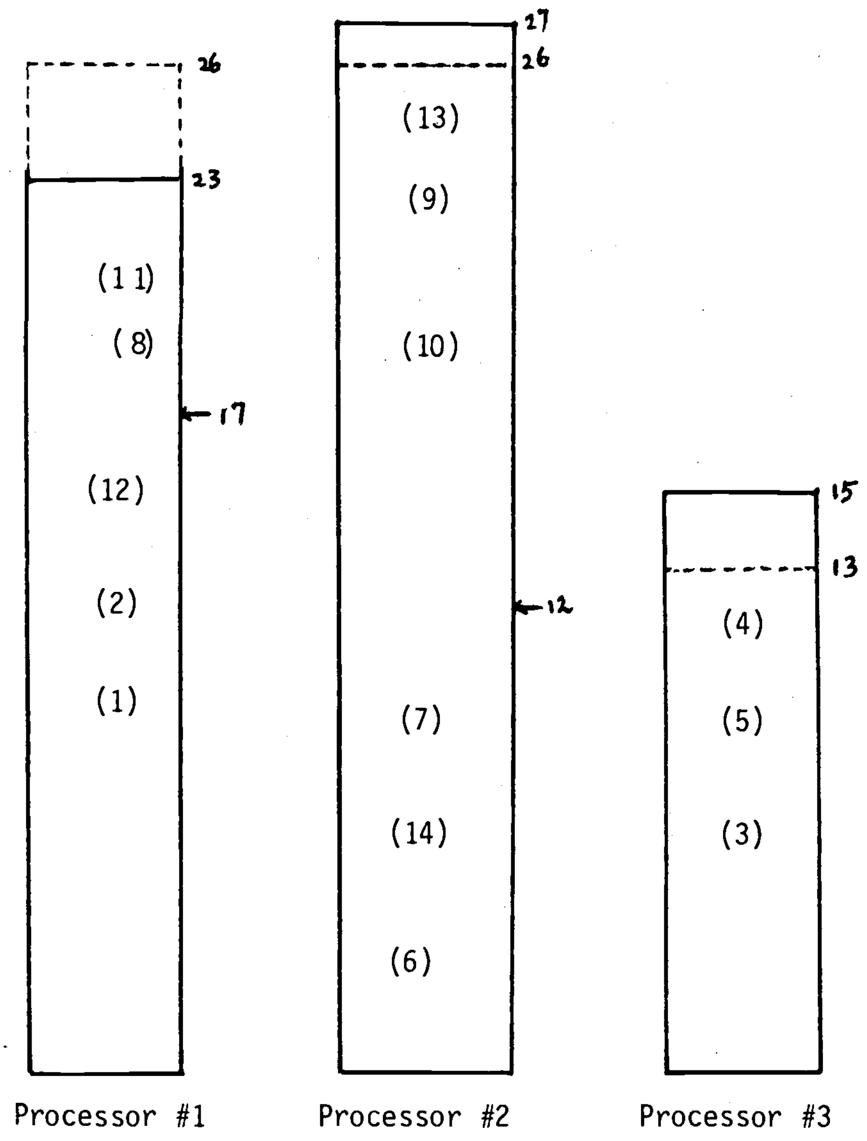


Figure 5.1 Jobs with the same class are put into stacks



———	Actual required duration for processing all jobs assigned to each processor.
------	Optimum makespan ($Z^*=13$)

Figure 5.2 Jobs are assigned to each processor.

Table 5-1. Cost matrix for the jobs in Processor #1.

i\j	(1)	(12)	(2)	(11)	(8)
(1)	∞	1	5	3	3
(12)	0	∞	0	3	11
(2)	1	1	∞	3	4
(11)	4	4	2	∞	0
(8)	5	2	10	4	∞

The following are branch and bound procedures.

Steps 1 & 2. Set $C_{k1} = \infty$ and reduce the cost matrix. We obtain Table 5-2

Table 5-2. The reduced cost matrix No. 1

i\j	(1)	(12)	(2)	(11)	(8)
(1)	∞	0	5	0	3
(12)	∞	∞	0	0	11
(2)	∞	0	∞	0	4
(11)	∞	3	2	∞	0
(8)	∞	1	10	1	∞

Step 3. $S=1$, $v(1) = 4$

Steps 4-9. Table 5-3 to Table 5-6 show the results of each step in the branch and bound algorithm. The final decision tree is shown in Figure 5-3.

Table 5-3. The reduced cost matrix No. 2.

i\j	(12)	(2)	(11)	(8)
(1)	0^0	5	0^0	3
(12)	∞	0^2	0^0	11
(2)	0^0	∞	0^0	4
(11)	3	2	∞	0^5
(8)	1	10	1	∞

Table 5-4. The reduced cost matrix No. 3.

i\j	(12)	(2)	(11)
(1)	0^0	5	0^0
(12)	∞	0^5	0^0
(2)	0^0	∞	0^0
(8)	1	10	∞

Table 5-5. The reduced cost matrix No. 4.

i\j	(12)	(11)
(1)	0^1	0^0
(2)	∞	0^∞
(8)	1	∞

Table 5-6. The reduced cost matrix No. 5.

i\j	(12)
(1)	0^1
(8)	1

The results of this problem are shown in Figure 5.4 and Figure 5.5. The total cost for the initial schedule L_1 and permutated schedule L'_1 is the same. In this example, BINBAB produces the best answer comparing with algorithm I and algorithm II.

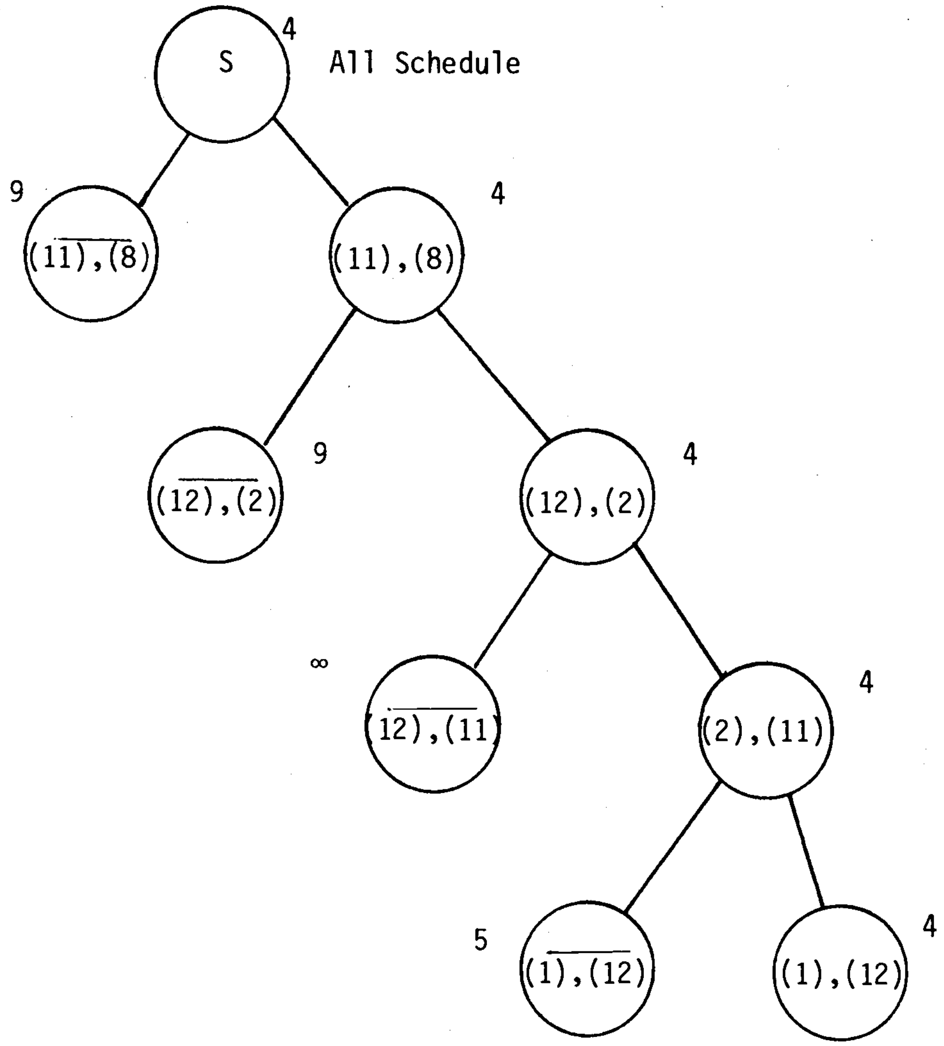
The flow chart for the BINBAB algorithm is shown in Figure 5-6. However, the flow chart for the procedure of branch-and-bound is not shown here because the detail flow chart can be found in Little et al. (1963, p. 978). The program for the BINBAB algorithm is shown in the Appendix.

5.4 Computational Experience

All three proposed algorithms were coded in FORTRAN IV. Approximately 9 problems were run on CDC Cyber 17720 at Oregon State University. Only the problem with a successful result produced by the algorithms I and II are summarized in Table 5.7. The cost matrix data are either selected from Gillett(1976, p. 503) or generated by the random number subroutine.

From the results and observations of computation of these three algorithms, we have the following conclusions:

(1) The solution time of heuristic algorithm I is faster than



Optimal Schedule for processor #1 is

(1) - (12) - (2) - (11) - (8) with total cost = 4

Figure 5.3. Decision tree for the sequencing of non-repetitive jobs on processor #1.

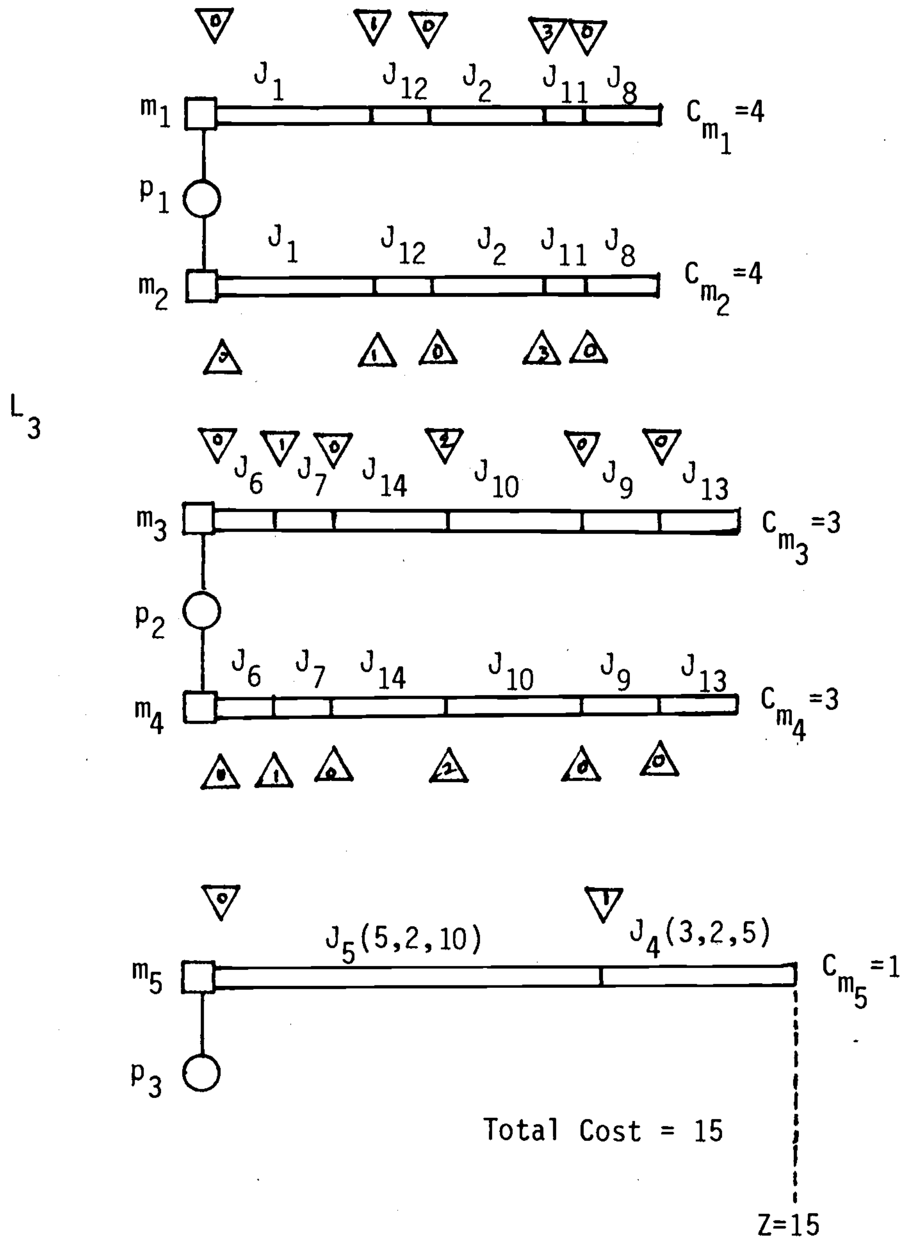


Figure 5.4 Schedule L_3 is constructed by branch and bound method

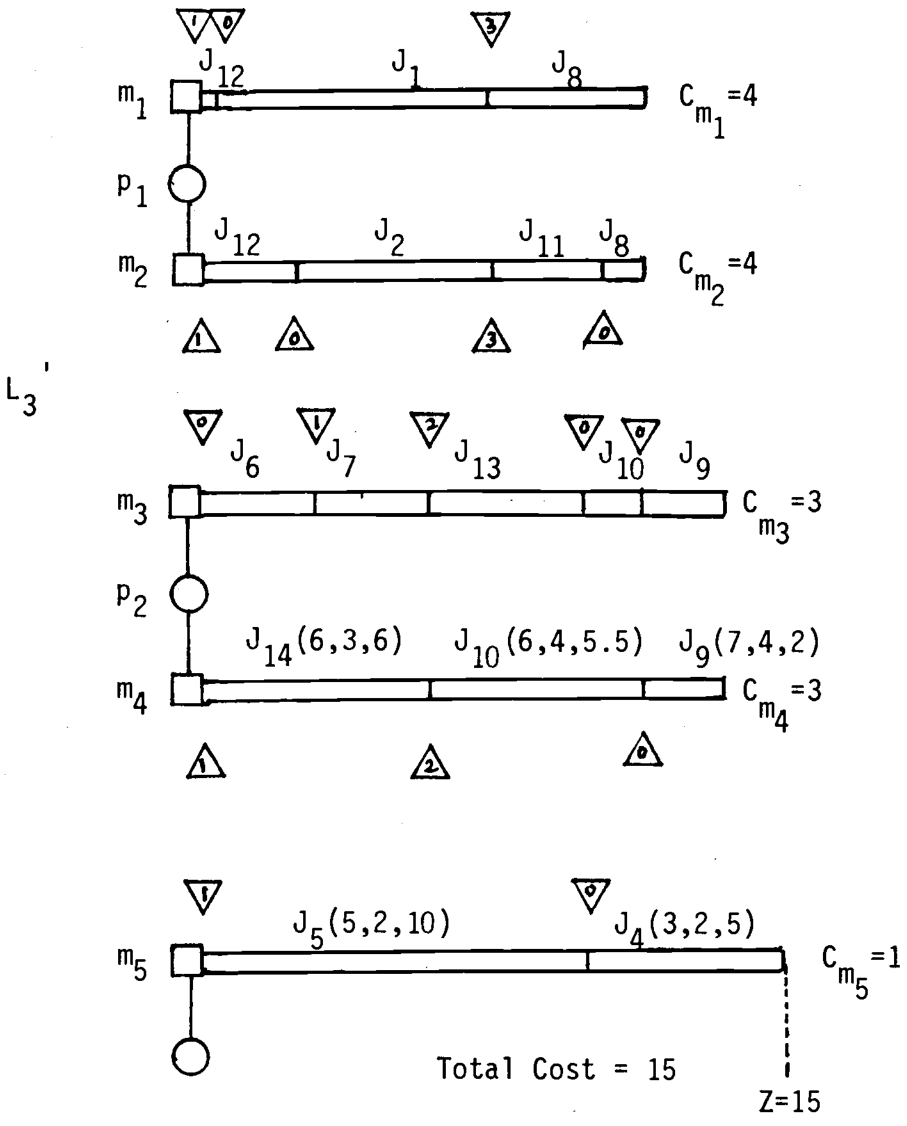


Figure 5.5 A Permutated schedule from L_3

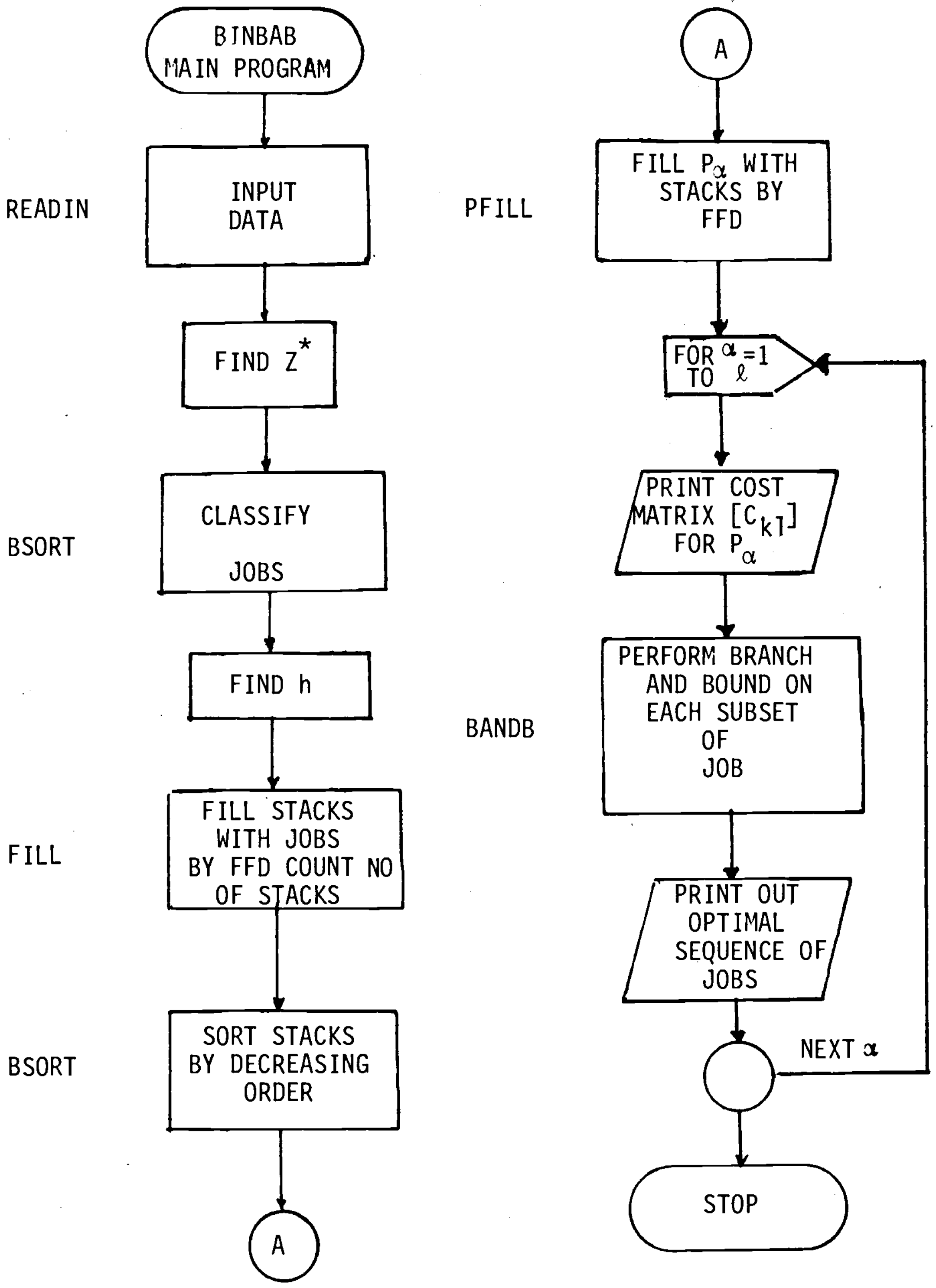

BINBAB MAIN PROGRAM
READIN
INPUT DATA
FIND Z^*
BSORT
CLASSIFY JOBS
FIND h
FILL
FILL STACKS WITH JOBS BY FFD COUNT NO OF STACKS
BSORT
SORT STACKS BY DECREASING ORDER
A
A
PFILL
FILL P_α WITH STACKS BY FFD
FOR α =1 TO ℓ
PRINT COST MATRIX $[C_{kl}]$ FOR P_α
BANDB
PERFORM BRANCH AND BOUND ON EACH SUBSET OF JOB
PRINT OUT OPTIMAL SEQUENCE OF JOBS
NEXT α
STOP

Figure 5.6 Flow Chart of the BINBAB Algorithm

Table 5.7 Computational Results

					Algorithm I				Algorithm II				Algorithm III			
Problem No.	n	ℓ	S	Z^*	Solution in CPU sec	Total Cost	Z = Max [Z(α)]	$\frac{Z}{Z^*}$	Solution in CPU sec	Total Cost	Z = Max [Z(α)]	$\frac{Z}{Z^*}$	Solution in CPU sec	Total Cost	Z = Max [Z(α)]	$\frac{Z}{Z^*}$
1	14	2	4	25	.04	180	28.5	1.14	.09	191	30	1.20	.84	185	27.5	1.10
**2	14	3	5	13	.04	21	15	1.15	.16	25	15.5	1.19	.40	15	15.0	1.15
3	16	3	5	25.5	.05	220	27.6	1.08	.15	187	28.75	1.13	.56	178	28.0	1.10
4	18	3	6	35.0	.05	140	37.5	1.07	.16	153	39.75	1.14	1.14	136	40.5	1.16
5	25	4	8	15.0	.06	650	17.0	1.13	.18	620	17.0	1.13	1.48	589	16.5	1.10
6	30	3	6	31.0	.07	790	34.0	1.09	.31	720	34.5	1.11	3.04	701	36.5	1.18

**The data of this problem were not randomly generated.

algorithm II because algorithm I is of order $O(n)$ and algorithm II is of order $O(\ell n)$, where ℓ is the number of processors. The third algorithm is the slowest because the amount of work done is much more than the other two. The amount of operations are due mainly to the sorting, tree branching and searching.

(2) Algorithm III produced the least cost schedule when randomized data were used. However, when the data were not randomly generated, it did not always produce the least cost schedule. More will be discussed on this in the next chapter. Algorithm II seemed to give lower cost results than algorithm I. However, from the observations of the results, the makespan of the schedule obtained from algorithm II is usually poorer than algorithm I and the chance of failure (i.e. an infeasible schedule) is higher than algorithm I. The failure often occurred at the end of the schedule where the last one or two jobs could not be scheduled. The infeasible schedule was due to either an insufficient resource or beyond the given planning horizon. Generally it is possible to distinguish good and bad heuristics by making a number of experimental trials.

(3) Three algorithms produce a schedule with the assumption that all jobs have to be split over two machines equally in order to have a feasible and tight schedule.

(4) It is difficult to investigate how the solutions obtained from these three methods compare to the optimal solution because the latter is difficult to obtain. An exhaustive search program is hard to program, because we have to consider the resource constraints, cost and makespan at the same time. If we ignore the makespan and resource constraints consideration, we can solve the MP as an assignment problem by a modified transportation algorithm of Ford and Fulkerson (1962, p. 95) or by the Hungarian method (Gillett, 1978, p. 112). An improved lower bound for the cost will be obtained but it is not guaranteed to be the optimum.

CHAPTER VI

APPLICATION OF THE ALGORITHMS --- A CASE STUDY

6.1 Introduction

We present a real life case study of how the developed algorithms function in the design and implementation of a production planning system in an aluminum reduction plant. The plant, the largest in the Northwestern part of the U.S., is strategically located in the State of Washington to take advantage of cheap electrical power. The plant produces alloyed and unalloyed sheet, plate, foil and foundry ingots, T-ingots and extrusion billet. Products from the plant are shipped to other fabricating facilities of the company or to customers both at home and abroad. The plant employs about 1,020 people with an annual production capacity of 210,000 tons.

6.2 Brief Description of the Plant Operation

A. Raw Material Flow

The process of aluminum reduction runs 24 hours a day, seven days a week. This means that raw materials must be in constant supply, pots must be kept operating at all times, and the pouring operation and handling of the finished product must be maintained around the clock.

The basic raw material is a fine, white powder called alumina ore which has about the consistency of sugar. It is brought into the plant by ship. The ore unloading system at the plant dock features a long 150 foot-high gantry crane and suction nozzles to suck up the ore from a ship's hold. The system is designed to eliminate this alumina ore dust in the air and water. The ore, which is now stored in two huge silòs at the end of each reduction building, is transported into the potroom through pipes.

B. Reduction

The plant has six production lines which are called the "potlines." The potlines reduce aluminum oxide (alumina) into molten metal through an electrolytic process that is considered to be both highly efficient and low in cost. This high productivity is accomplished by the use of proper materials, equipment, and manpower. Under normal operating conditions, raw materials and manpower usage in the reduction processes are predictable by the plant's management.

Operation of a potline can be broken down into three basic activities: working, oreing up, and tapping.

"Working the pots" is the term applied to breaking up the crust of the pot with a poker prepatory to adding ore to make the molten bath. "Oreing up" is done when each pot is worked. The pots are then fed with alumina. "Tapping" is the final operation performed on the potlines. This process, drawing the aluminum from the bottom of the reduction cell, is illustrated in Figure 6.1. A large crucible is brought in from the pouring room on a low trailer. The

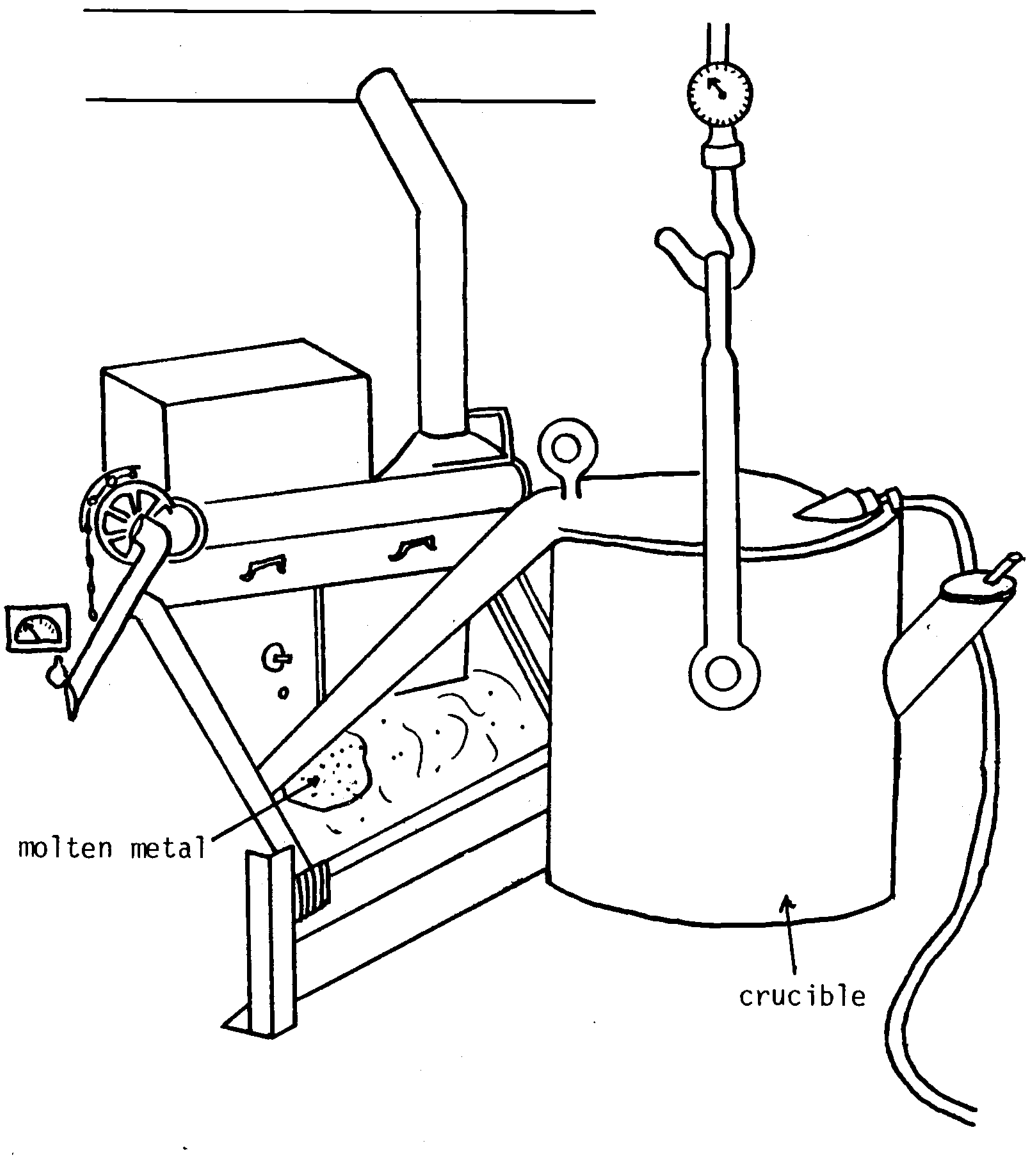


Figure 6.1. Transferring molten metal into a crucible for transportation to the cast house

crucible is equipped with a siphon lid and a long tube which is inserted through a hole punched in crust, and lowered to the bottom of the vessel suspended by an overhead crane. The crucible is placed into position; a workman places a cover over the pouring spout and attaches a vacuum hose to a fitting on the lid. Attached to the crane is a scale, which is used to determine how much metal is flowing into the crucible.

C. Pouring (casting)

i) Aluminum ingots--the crucible containing the molten metal is transported on the trailer to the cast facility. The crucible, equipped with pouring handle, is then picked up by an overhead crane and is guided by an operator into proper position to pour into a furnace. The molten aluminum must be cleaned and then poured into ingots or "pigs." The latter weigh between 50 and 1200 lbs. These aluminum ingots are up to 99.6 percent pure.

ii) Alloy ingots--the company produces sheet ingots and billets depending upon what kinds of alloys are being produced. Alloying ingredients are added to the melt in the furnace. Regardless of the alloy, the molten aluminum must first be cleaned and degassed in the filter box; then various sizes of ingots are produced according to the customer's specifications.

6.3 System Boundary and Processes Description

A. System Boundary

The cast house located in the reduction plant is composed of nine holding furnaces, four vertical casting units (VDC units) and a pigging wheel. These units are arranged in the cast house as indicated in Figure 6.2. Each holding furnace that feeds a vertical casting unit can be operated in conjunction with a molten metal filter box.

B. Processes Description

In order to present the problem more clearly, the activities in the cast house (system boundary) are divided into the following processes:

i) Molten metal arrives at the cast house--Crucibles of molten metal arrive at the north and south cast houses. The metal arrives at the south cast house from the south potline, and the crucible average net capacity is 5,600 lbs. The molten metal from the south potline cannot be used in the north cast house furnaces because of its low grade in purity. The metal arrives to north cast house from the north potline, and the crucible average net capacity is 8,600 lbs.

ii) Molten metal is charged into furnaces--upon arrival the overhead craneman hooks up the full crucible and

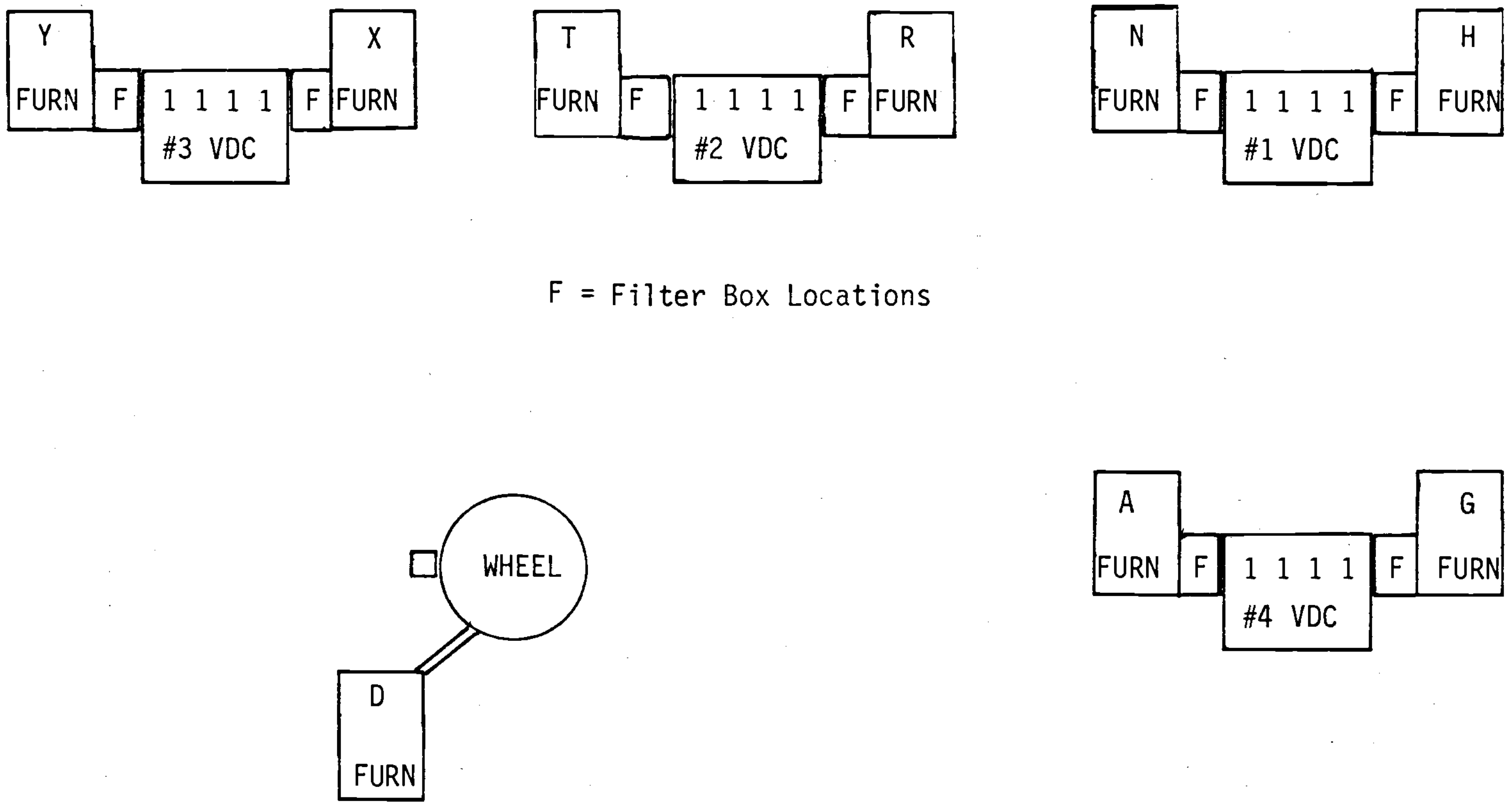


Figure 6.2 Cast house holding furnace casting unit layout.

moves it to a scale. A scaleman weighs and records the value.

iii) Melting--each furnace has a maximum capacity of 95,000 lbs. They are usually operating in conjection with a 5,000 lb. filter box. There is always a minimum of 15,000 lbs. to 13,000 lbs. molten metal left in the furnace after each casting (called a "drop"). Sometimes it will be more, depending on the drop weight (the amount of molten metal poured out during a drop). So a furnace has a usable maximum melt capacity of 80,000 lbs. When a furnace is full, an alloyman charges a calculated amount of various alloying ingredients into the furnace according to the particular alloy to be cast. Then the alloyman stirs the furnace with a boom. The molten metal is then fluxed with chlorine gas for half an hour to get rid of the alkaline metallic elements. Upon completion of this fluxing, the alloyman skims the furnace to get rid of the dross (non-metallic oxide from the molten metal) and takes samples from the furnace.

iv) Casting--when the metal in the furnace is on grade and the vertical casting unit is ready, a crew consisting of a furnace operator and a casting attendent starts the drop (a casting) by removing the plug from the furnace. They tap the furnace to induce the molten metal to the trough (Figure 6.3).

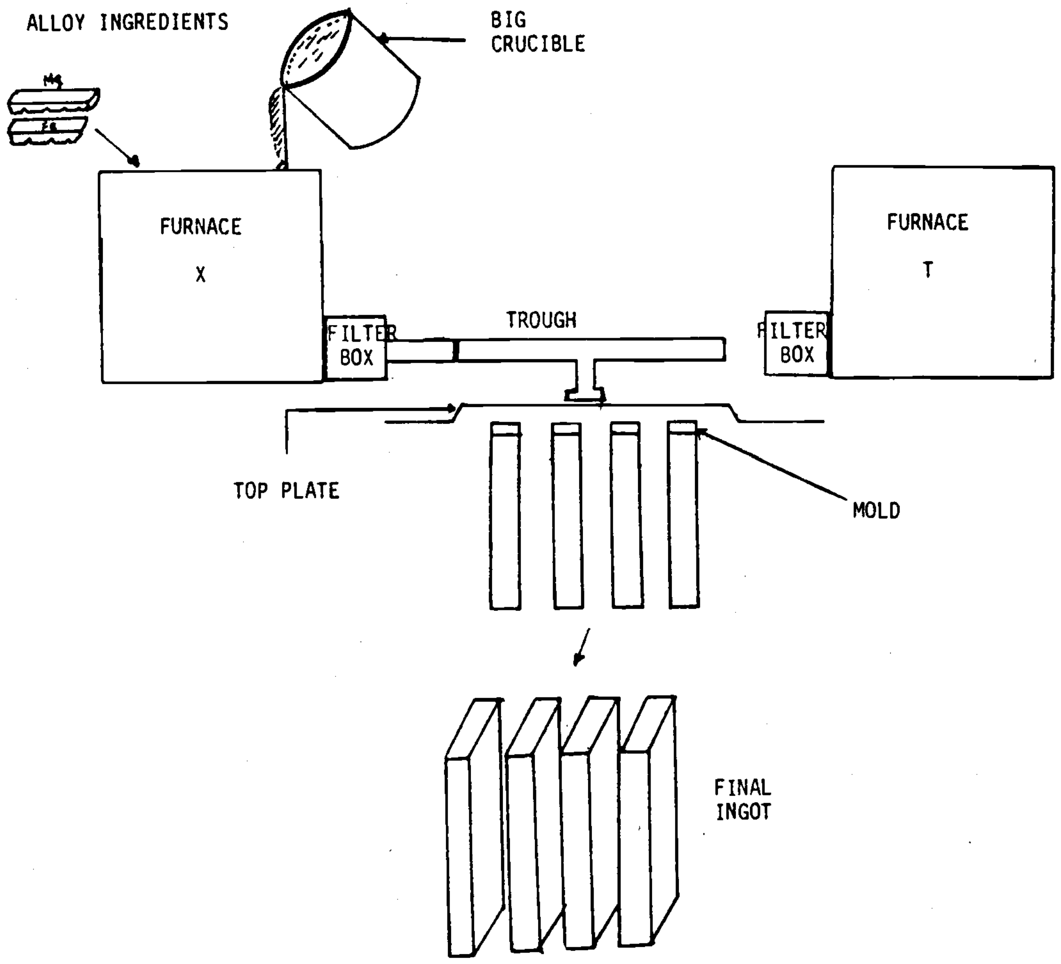


Figure 6.3 Melting, casting and removal of ingot.

Various size alloys have different casting speeds which are expressed in inches per minute. The casting time can be represented by the following formula:

$$\text{cast time (hours)} = \text{cast length (inches)} \div [(\text{casting speed}) \times 60 \text{ minutes}]$$

v) Ingot Removal--immediately after a drop, the ingots are removed to storage by an overhead craneman with the assistance of the casting attendants.

vi) Tool Change--when an alloy of size S_1 is changed to size S_2, the mold in the VDC has to be changed before a new casting.

vii) Furnace "Wash" and Filter Box "Wash"--different alloys have different chemical composition. (Refer to Table 6.1). For example, a can-stock alloy (5052), which is used to make beverage cans, has a high magnesium content. If the production of this can-stock alloy is followed by the production of a cable alloy (1100, a magnesium free material for electrical power cable), then the furnace has to be drained, diluted, and cleaned with pure molten aluminum. This pure molten aluminum becomes scrap (off-grade metal). The scrap generated from the cleaning process can be computed and considered a part of the changeover cost. If a filter box is used with the furnace, it must also be "cleaned". However, in the case of the filter box, the washing process

Table 6.1

Alloy Chemical Composition

ALLOY	ID	FE	FE	SI	SI	CU	CU	MN	MN	MG	MG	ZN	ZN	CR	CR
		MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1050	1	.28	.35	.08	.12	0.00	.03	0.00	.03	0.00	.01	0.00	.04	0.00	.03
1100	2	.55	.65	.10	.15	.10	.20	0.00	.05	0.00	0.00	0.00	.10	0.00	0.00
.															
.															
.															
.															
.															
.															
.															
5052	10	.40	.65	0.00	.12	0.00	.10	0.00	.10	1.30	1.70	0.00	.20	0.00	0.00

continues after the furnace wash is completed. Some alloy changes may not require a furnace "wash" (e.g., from a low concentration to high concentration), but a filter box "wash" is almost always required. The filter "washing" process is very similar to the continuous process, and the scrap produced is also predictable (Figure 6.4).

The scrap from a washing process can be re-used at any time. After a dilution process, the scraps are cut into small pieces and transferred to a remelt process.

6.4 Problem Identification

Each month the plant receives a list of customers' orders from headquarters. These orders contain what type of products and specifications, order quantity and desired shipping dates (week ending). The cast house general foreman schedules the production of the products ordered by intuitive judgment and experience. He will try to balance and consider all factors (e.g. furnaces makespan minimization, mold availability, etc.). He manually constructs an acceptable schedule for a month by using a Gantt chart and load diagrams. At present, there is no quantitative technique used to evaluate how good or optimal a schedule is. The company management feels that this is a weak point in the company's structure from the risk management point of view. The complete production planning system depends on an experienced foreman.

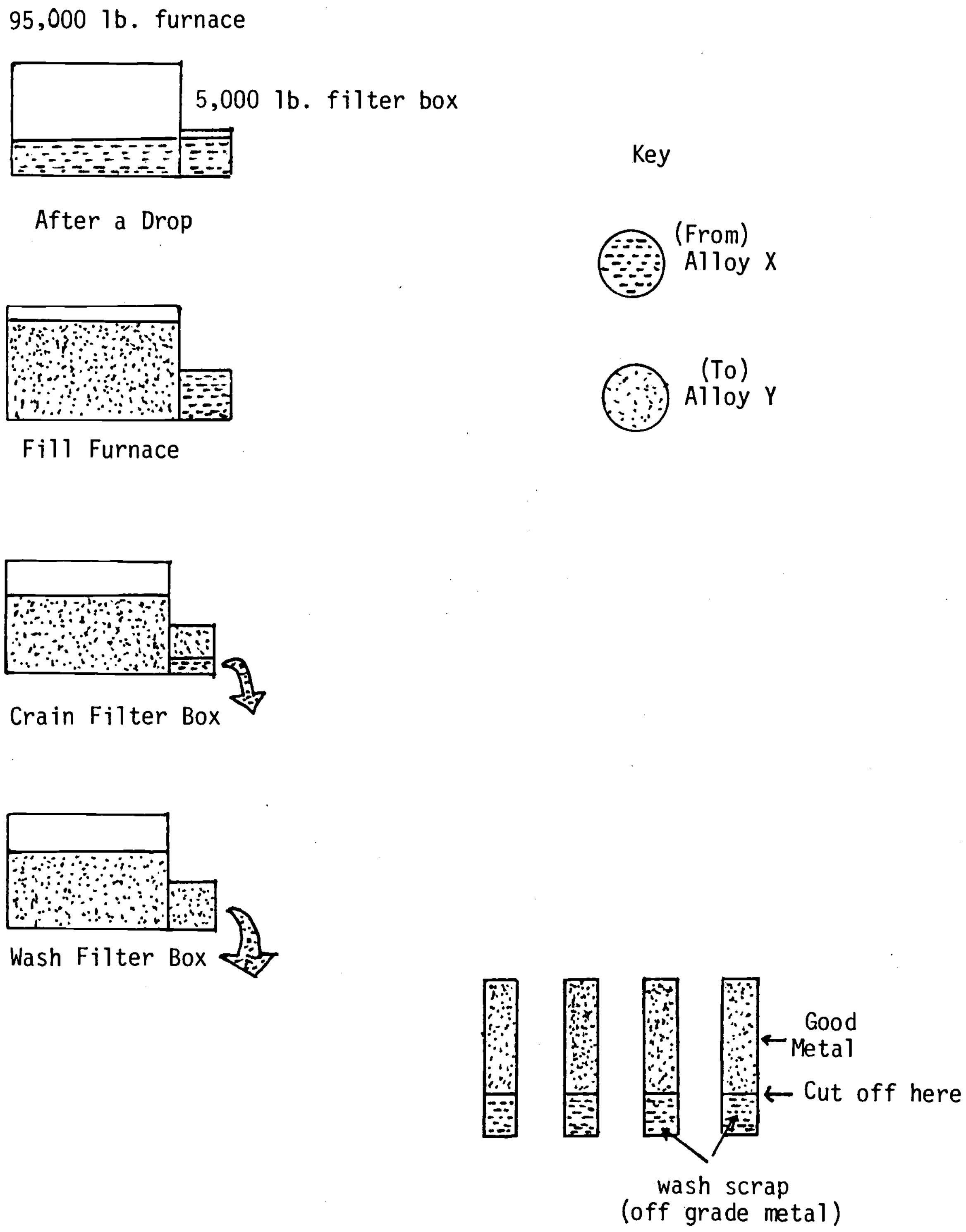


Figure 6.4 Filter box wash procedure

At the moment, the cast house is expanding and new furnaces are being built. As a consequence, the management is interested in finding a good scheduling procedure which can minimize the total setup and scrap cost with fixed availability of tools and molds while at the same time balancing furnace utilization.

6.5 The Production Planning System

As the result of this study, we have proposed to introduce a two-level computerized production planning system.

(1) The aggregate production planning - A computer program has been developed to compute the job changeover cost, the processing time of each job and furnace capacity..

(2) Feasibility and optimization scheduling - To the results of (1) above, we apply the three heuristic algorithms to find the best minimum cost scheduling. Figure 6.5 shows the detail of the system.

6.6 The Result of the Case Study

Past historical data are used to evaluate the effectiveness of these three algorithms. The changeover cost of each job is computed and all values are scaled (divided by 15) in order to have the unit costs to have numerical values less than 999. Since all three programs use the same input format, the actual data used are shown in Table 6.2. Table 6.3 shows the cost matrix. The results are listed on Table 6.4. Because of limited computer funds, the three algorithms were

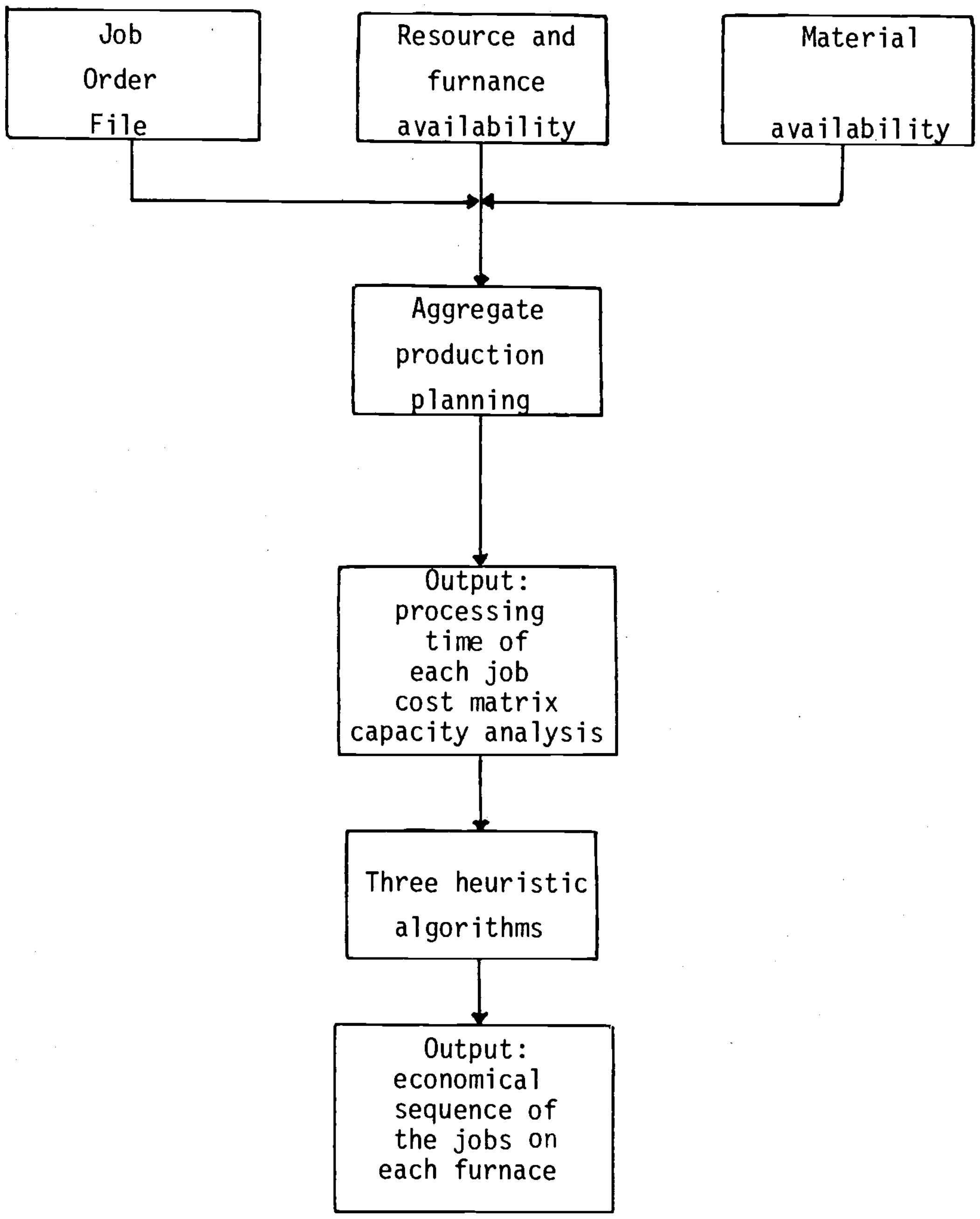

Job Order File
Resource and furnance availability
Material availability
Aggregate production planning
Output: processing time of each job cost matrix capacity analysis
Three heuristic algorithms
Output: economical sequence of the jobs on each furnace

Figure 6.5 The production scheduling system.

Table 6.2 Job order of October 1980.

NO. OF PROCESSORS ARE 3

NO. OF JOBS ARE 24

NO. OF RESOURCE TYPE = 8

	TYPE	QUANT
(20 x 54)	2	2
(20 x 60)	3	2
(24 x 41)	5	2
(24 x 45)	6	1
(24 x 53)	7	1
(24 x 60)	8	1
(12" dia)	13	1
(18 x 61)	16	1

PROCESSOR	MACHINES
1	1
2	2
3	2

JOB DESCRIPTION INPUT

		Alloy	Size	Order Quantity in M lb.	Required Processing Days
(1)	(13., 5.,11.0,1.,1.)	5352	24 x 41 x 182	900 + 200 (outstanding order)	11
(2)	(13., 6., 4.0,0.,0.)	5352	24 x 45 x 182	400	4
(3)	(12., 6., 5.0,0.,0.)	5252	24 x 45 x 150	500	5
(4)	(12., 5., 4.0,0.,0.)	5252	24 x 41 x 150	500	4
(5)	(12., 7., 3.0,0.,0.)	5252	24 x 53 x 150	400	5
(6)	(15., 7.,15.0,0.,0.)	5657	24 x 53 x182	1800	15
(7)	(15., 8., 7.0,0.,0.)	5657	24 x 60 x 172	1000	7
(8)	(15., 6., 4.0,0.,0.)	5657	24 x 45 x 164	400	4
(9)	(15., 5., 2.0,0.,0.)	5657	24 x 41 x 164	200	2
(10)	(27.,16.,10.0,1.,2.)	R396	18 x 61 x 164	1200 + (outstanding order)	10
(11)	(11., 5., 8.0,0.,0.)	5052RO	24 x 41 x 164	800	8
(12)	(25., 5., 3.0,0.,0.)	MD192	24 x 41 x 164	300	3
(13)	(7., 5., 5.0,0.,0.)	3003F	24 x 41 x 164	500	5
(14)	(7., 7., 8.0,0.,0.)	3003F	24 x 53 x 182	1100	8
(15)	(7., 2., 2.0,0.,0.)	3003F	20 x 54 x 164	300	2
(16)	(4., 7., 0.0,1.,3.)	1235	24 x 53 x 195	0	0
(17)	(4., 2., 2.0,0.,0.)	1235	20 x 54 x 118	300	2
(18)	(32.,13.,18.0,0.,0.)	7029	12" dia. x 253	1500	18
(19)	(10., 6.,14.0,0.,0.)	5050	24 x 45 x 195	1000	14
(20)	(22., 3., 4.0,0.,0.)	MD221	20 x 60 x 130	250	4
(21)	(24., 6., 8.0,0.,0.)	MD175	24 x 45 x 164	600	8
(22)	(8., 8., 4.0,0.,0.)	4343	24 x 60 x 150	400	4
(23)	(20., 3., 2.0,0.,0.)	8079	24 x 60 x 195	200	2
(24)	(19., 2., 3.0,0.,0.)	7072	20 x 54 x 118	300	3

Table 6.3 Changeover cost of the alloys for the month of October, 1980.

COST MATRIX

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)	(22)	(23)	(24)
(1)	999	10	2	2	12	38	38	38	28	50	2	37	200	210	210	270	270	32	30	190	73	230	130	130
(2)	10	999	0	12	12	38	38	28	38	50	12	47	210	210	210	270	270	32	20	190	63	230	130	130
(3)	10	0	999	12	12	38	38	28	38	50	12	47	210	210	210	270	270	32	20	190	63	230	130	130
(4)	0	10	10	999	12	38	38	38	28	50	2	37	200	210	210	270	270	32	30	190	73	230	130	130
(5)	10	10	10	12	999	28	38	38	38	50	12	47	210	200	210	260	270	32	30	190	73	230	130	130
(6)	16	16	16	16	16	999	10	10	10	24	14	22	50	40	50	128	138	61	14	22	14	73	55	50
(7)	16	16	16	16	16	10	999	10	10	24	14	22	50	50	50	138	138	61	14	22	14	60	55	50
(8)	16	6	6	16	16	10	10	999	10	24	14	22	50	50	50	138	138	61	4	22	4	70	55	50
(9)	6	16	16	6	16	10	10	10	999	24	4	12	40	50	50	138	138	61	14	22	14	20	55	50
(10)	19	19	19	19	19	20	20	20	20	999	18	27	90	90	90	80	80	25	18	19	38	90	30	35
(11)	21	31	30	20	30	25	25	25	15	55	999	34	200	210	210	260	260	32	30	130	30	130	130	130
(12)	32	42	40	30	40	35	35	35	25	20	32	999	57	67	67	38	38	39	48	50	14	50	60	26
(13)	271	281	288	278	278	268	268	268	258	190	58	180	999	10	10	264	264	290	94	70	160	288	60	70
(14)	281	281	268	288	268	268	268	268	268	190	68	190	10	999	10	254	264	290	94	70	160	288	60	70
(15)	281	281	288	288	278	268	268	268	268	190	68	190	10	10	999	264	254	290	94	70	160	288	60	60
(16)	40	46	35	35	45	30	40	48	40	27	38	16	52	42	52	999	10	28	32	27	38	42	30	28
(17)	40	40	35	35	35	40	40	40	40	27	38	16	52	52	42	10	999	28	32	27	38	42	30	18
(18)	150	150	155	155	155	160	160	160	160	260	198	32	300	300	308	250	250	999	198	298	380	231	310	327
(19)	31	21	25	38	38	33	33	23	33	10	25	80	150	150	150	160	160	16	999	120	14	150	135	30
(20)	100	100	110	110	110	115	115	115	115	88	110	20	50	50	50	40	40	50	90	999	126	60	29	40
(21)	480	470	460	470	450	450	440	450	360	475	380	455	455	455	455	490	490	526	480	440	999	257	360	370
(22)	400	400	405	405	405	395	385	385	180	340	200	210	210	210	410	410	450	340	340	454	2	999	508	380
(23)	40	40	36	38	38	42	42	42	42	38	44	28	45	45	45	31	31	50	20	50	30	60	999	27
(24)	120	120	130	130	130	125	125	125	125	100	136	100	182	182	172	150	140	12	136	115	120	182	160	999

Table 6.4 Comparison of schedules obtained by manual methods and three algorithms

			Manual Method		Algorithm I			Algorithm II			Algorithm III (BINBAB)				
Month/ Year	Job Status	North Cast House Status	Total Cost in $ Value	$\frac{Z}{Z^*}$	Total Cost in $ Value	Total Cost After Schedule Permu- tated	$\frac{Z}{Z^*}$	Total Cost in $ Value	Total Cost After Schedule Permu- tated	$\frac{Z}{Z^*}$	Total Cost in $ Value	Total Cost After Schedule Permu- tated	$\frac{Z}{Z^*}$	Total Cost Saved in $ Value	Percent Reduc- tion
Oct. 1980	n = 24 e = 15 r = 8	s = 5 ℓ = 3	3040 x 15 = 45,600	$\frac{31}{29.2}$ = 1.06	1516 x 15 = 22,470	1498 x 15 = 22,470	$\frac{30}{29.2}$ = 1.03	2273 x 15 = 34,095	2273 x 15 = 34,095	$\frac{34}{29.2}$ = 1.16	1672 x 15 = 25,080	1672 x 15 = 25,080	$\frac{30}{29.2}$ = 1.03	45,600 - 22,470 = 23,130	$\frac{23,130}{45,600}$ x 100% = 51%
Jan. 1980	n = 13 e = 8 r = 6	s = 6 ℓ = 3	1904 x 15 = 28,560	$\frac{15}{14.3}$ = 1.05	1675 x 15 = 25,125	1645 x 15 = 25,125	$\frac{15.5}{14.3}$ = 1.08	1655 x 15 = 24,825	1632 x 15 = 24,480	$\frac{16.5}{14.3}$ = 1.15	1803 x 15 = 27,045	1660 x 15 = 24,900	$\frac{15.5}{14.3}$ = 1.08	28,560 - 24,480 = 4,080	$\frac{40,080}{28,560}$ x 100% = 14%
Feb. 1978	n = 26 e = 8 r = 6	s = 6 ℓ = 3	3398 x 15 = 50,970	$\frac{28}{27.33}$ = 1.02	2266 x 15 = 33,990	2236 x 15 = 33,540	$\frac{28}{27.33}$ = 1.02	2546 x 15 = 38,190	2217 x 15 = 33,255	$\frac{31.5}{27.33}$ = 1.153	1890 x 15 = 28,350	1816 x 15 27,240	$\frac{28}{27.33}$ = 1.02	50,070 - 27,240 = 23,730	23,730 x 100% = 47%

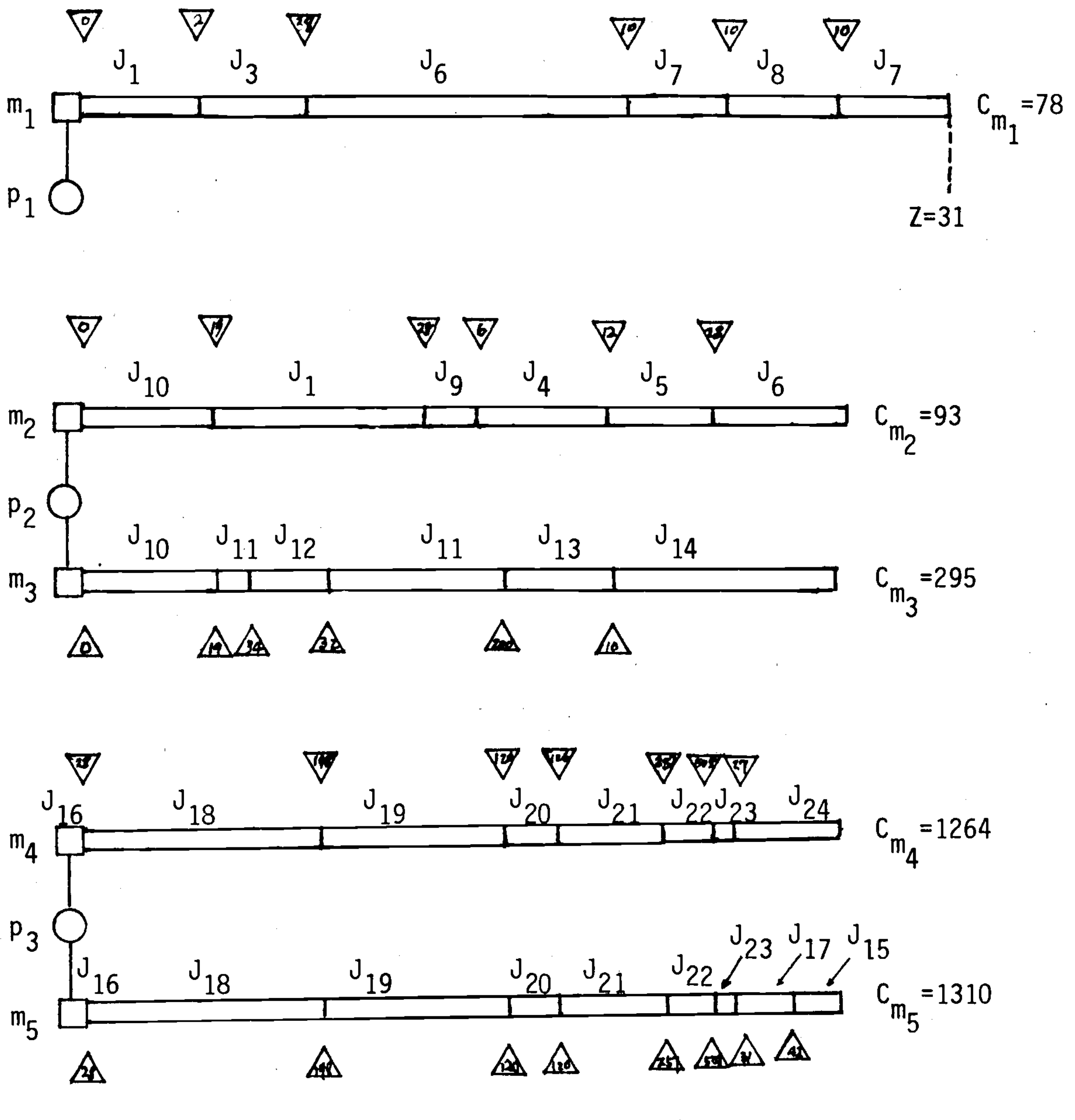
where n = number of alloys (jobs)

e = number of different types of alloys (job types)

r = total number of different molds used in that month (resource types)

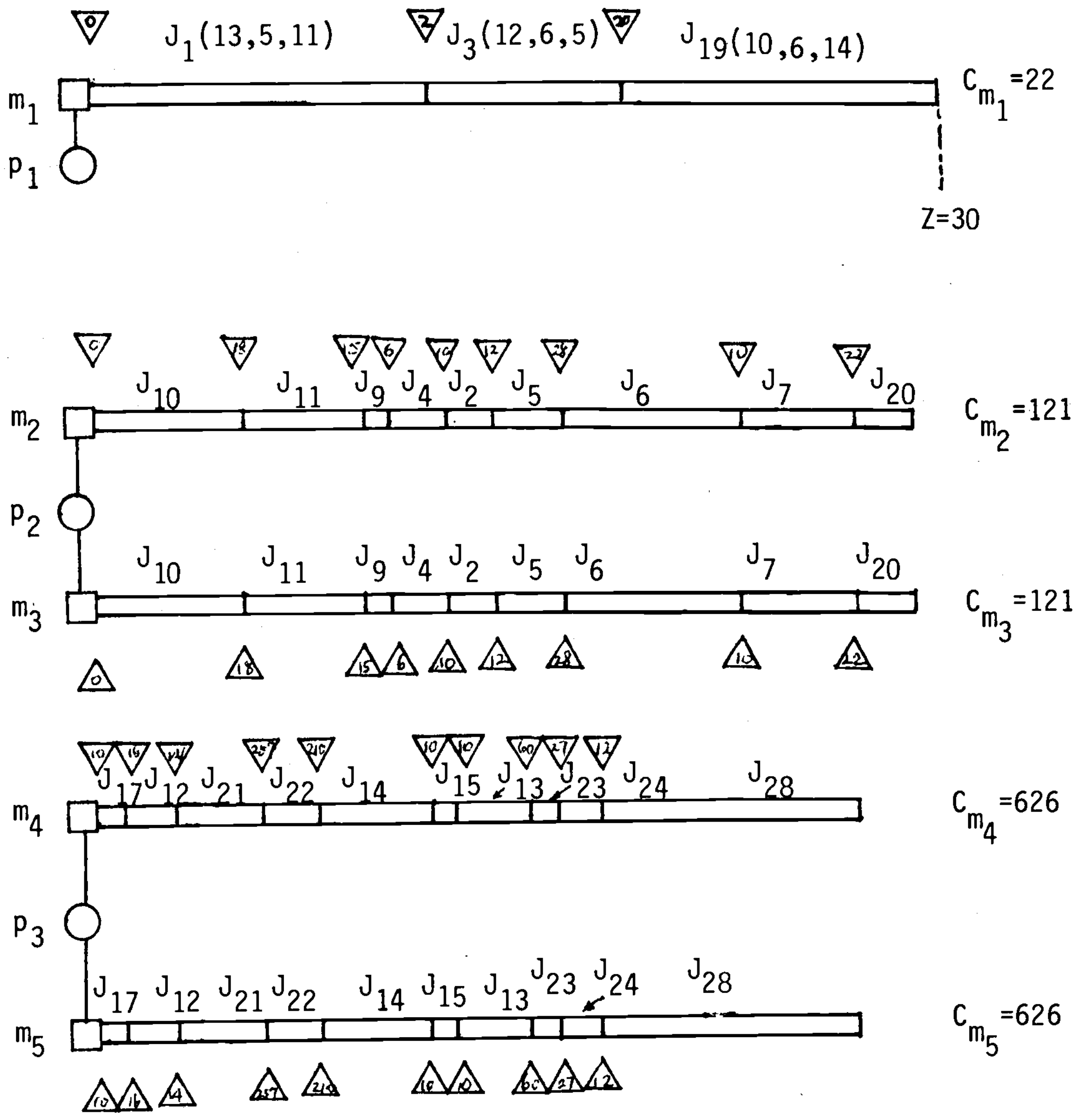
s = number of furnaces (machines)

ℓ = number of vertical casting units (processors)



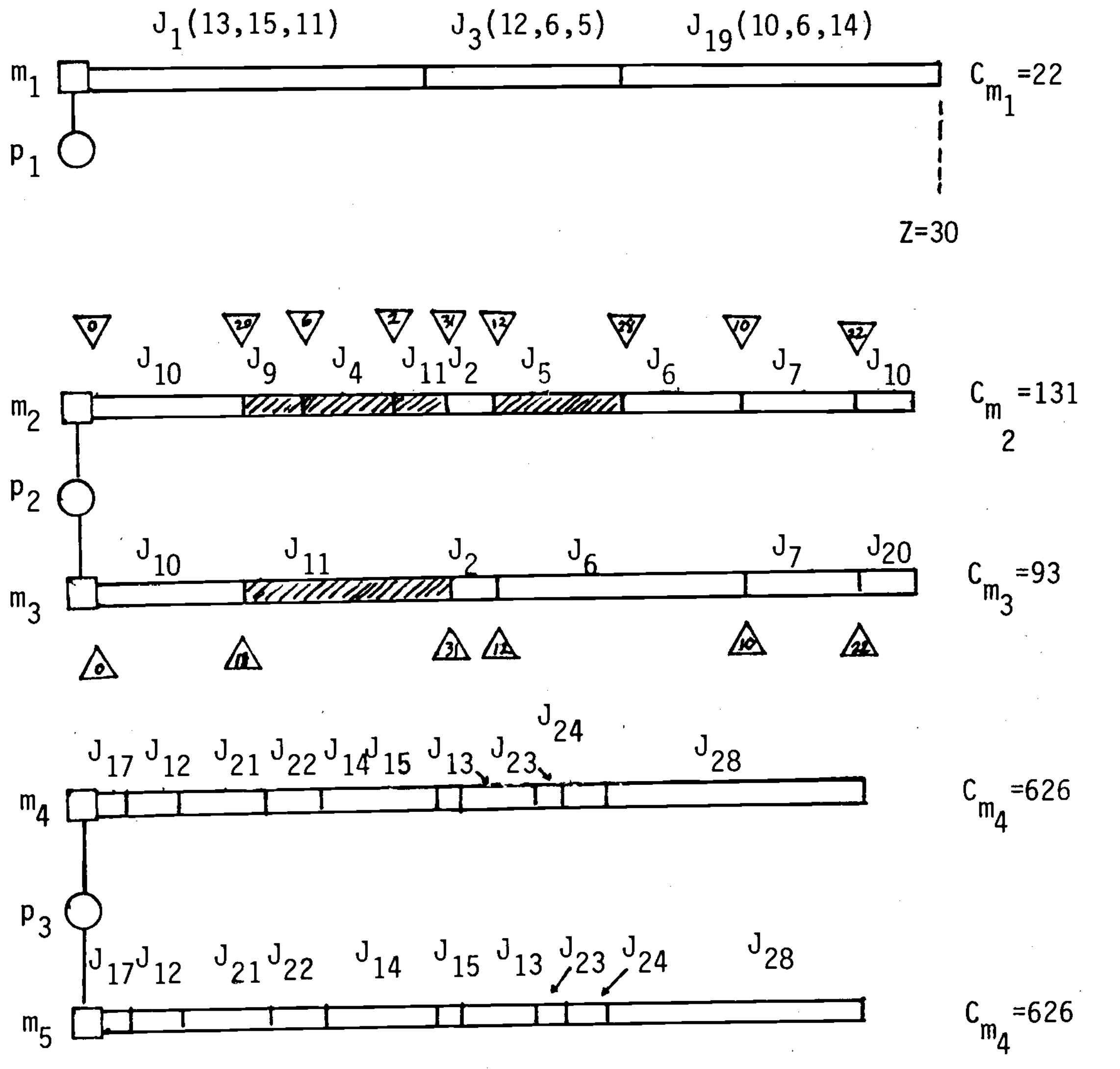
Total Cost = 3040

Figure 6.6 The production schedule produced by manual method.



Total Cost = 1516

Figure 6.7a A schedule is produced by algorithm I.

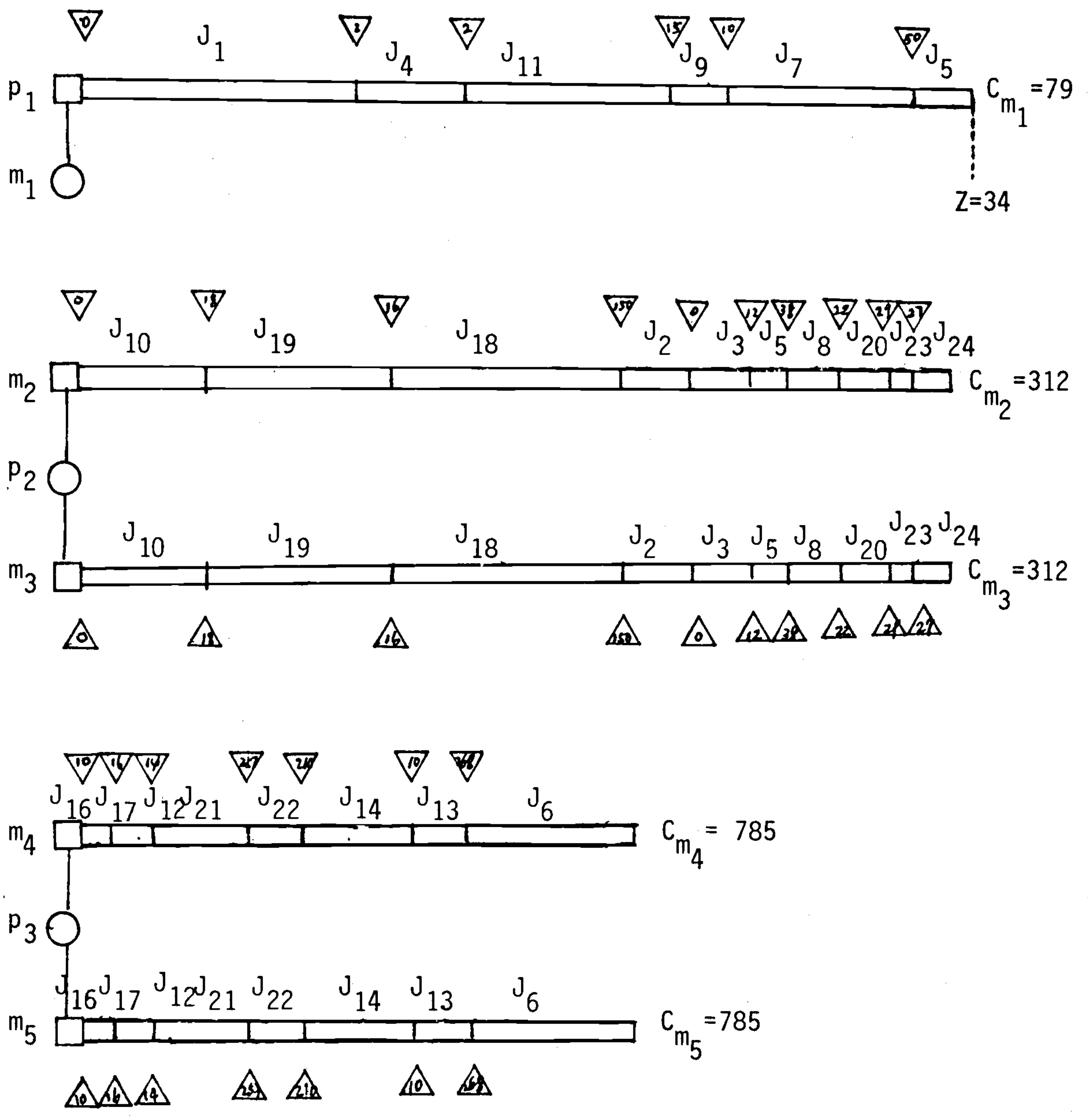


$J_1(13,15,11)$
$J_3(12,6,5)$
$J_{19}(10,6,14)$
m_1
p_1
$C_{m_1}=22$
$Z=30$
J_{10} J_9 J_4 J_{11} J_2 J_5 J_6 J_7 J_{10}
m_2
$C_{m_2}=131$
p_2
J_{10} J_{11} J_2 J_6 J_7 J_{20}
m_3
$C_{m_3}=93$
J_{17} J_{12} J_{21} J_{22} J_{14} J_{15} J_{13} J_{23} J_{24} J_{28}
m_4
$C_{m_4}=626$
p_3
J_{17} J_{12} J_{21} J_{22} J_{14} J_{15} J_{13} J_{23} J_{24} J_{28}
m_5
$C_{m_4}=626$

Total cost = 1498

▨ Jobs have been permutated.

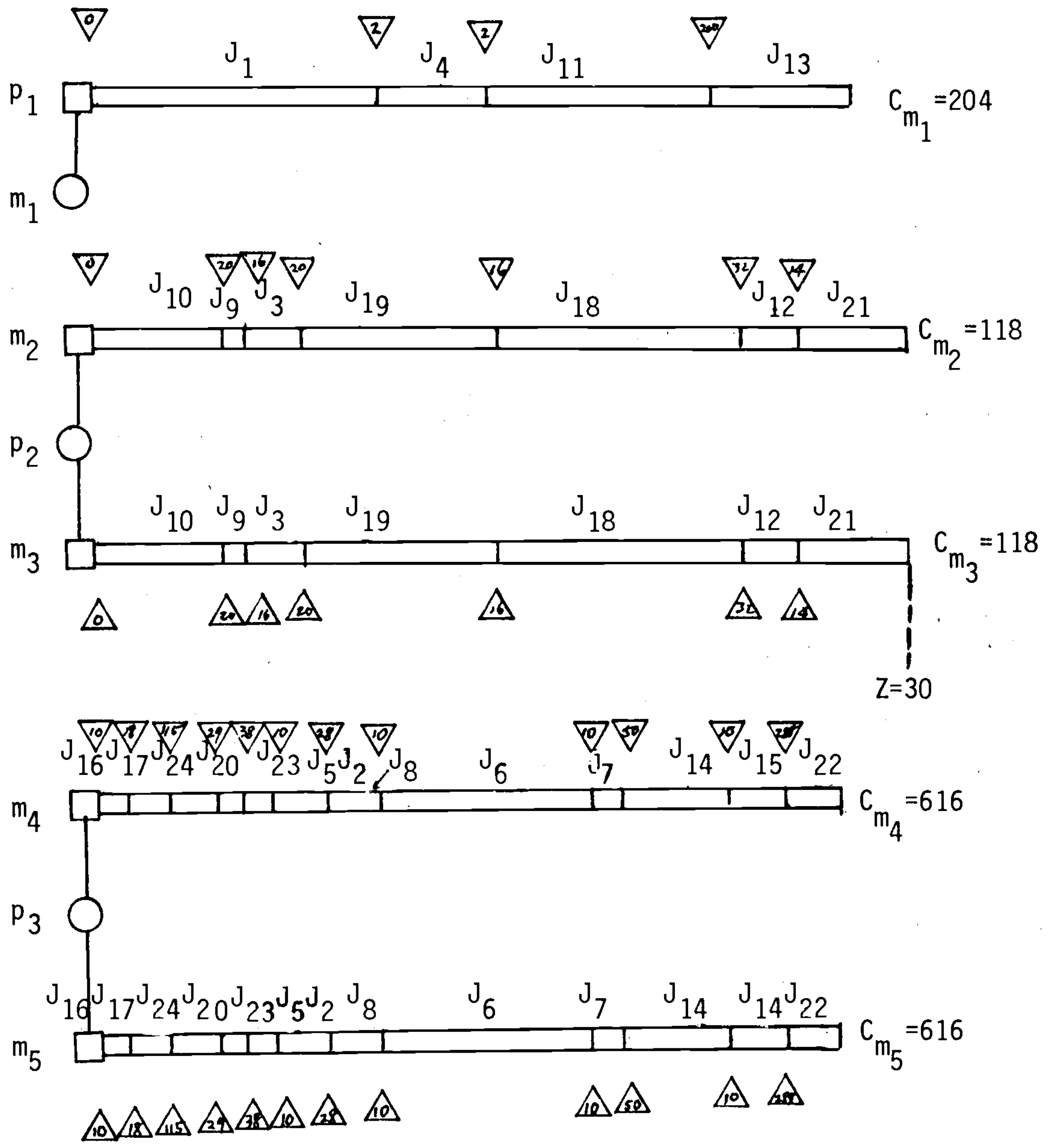
Figure 6.7b A permutated schedule from figure 6.7a.


p_1
m_1
J_1 J_4 J_{11} J_9 J_7 J_5
C_{m_1}=79
Z=34
m_2
p_2
m_3
J_{10} J_{19} J_{18} J_2 J_3 J_5 J_8 J_{20} J_{23} J_{24}
C_{m_2}=312
C_{m_3}=312
m_4
p_3
m_5
J_{16} J_{17} J_{12} J_{21} J_{22} J_{14} J_{13} J_6
C_{m_4} = 785
C_{m_5}=785

Total Cost = 2273

Planning horizon = 34 days

Figure 6.8 A schedule is produced by algorithm II


J_1
J_4
J_{11}
J_{13}
p_1
m_1
$C_{m_1}=204$
J_{10} J_9 J_3 J_{19} J_{18} J_{12} J_{21}
m_2
$C_{m_2}=118$
p_2
m_3
$C_{m_3}=118$
Z=30
J_{16} J_{17} J_{24} J_{20} J_{23} J_5 J_2 J_8 J_6 J_7 J_{14} J_{15} J_{22}
m_4
$C_{m_4}=616$
p_3
m_5
$C_{m_5}=616$

Total Cost = 1672

Figure 6.9 A schedule is obtained from algorithm III.

run on only three sets of data. The results show that all heuristic methods do better than the trial-and-error manual scheduling method. Only the results of the month of October, 1980 is shown in Gantt charts (Figure 6.6 to Figure 6.9).

From the results of this computer analysis, we have made the following observations.

(1) Although Algorithm III implicitly enumerates a subset of jobs to find the best sequence, it has fallen below our expectations when real data were used. (October 1980 and January, 1980). The reason may be because dissimilar alloys with the same size are grouped together thus producing a lot of scrap. Therefore another method worth trying may be to group similar alloys together and disregard the resource availability; then we may use a branch-and-bound method to implicitly enumerate each subset of jobs and find the best sequence. If it arbitrarily comes out with no resource conflict, then we have obtained a near-optimal solution.

(2) The percentage of total cost reduction is different each month. It is because scheduling by experience may sometimes produce an optimal solution. However, if the number of jobs, types of job and types of resource increase, human intuitive judgment becomes increasingly difficult.

(3) The author has tried to use different priorities for

scheduling. For example, if there is a tie, schedule the job with the shortest processing time (SPT) instead of the longest processing time (LPT). Sometimes a better solution is obtained.

(4) At present, the cast house general foreman takes four to six hours to produce a manual Gantt chart schedule at the beginning of each month. With the computerized system, about one hour is required to gather the necessary information to execute both the aggregate planning program and three heuristic programs. The time required to do one schedule permutation is about thirty minutes. Therefore, the total time required to produce a good production schedule by computer is about 2½ hours. This is a reduction of up to 50% in clerical work.

(5) During the year 1978, the cast house produced about 20 million pounds of furnace scrap at the cost of 2 million dollars. If this can be reduced by 20%, a real savings of $400,000/year will result. This also represents an increase in productivity for the plant.

6.7 Discussion

The scheduling of production and control of inventory are becoming more and more important to manufacturing companies. Often the volume and variety of products make the production scheduling computation difficult to perform manually. Furthermore, since more than one satisfactory schedule may be possible, the computer is useful in performing the complex calculations necessary to discover the best schedule for reducing costs and effectively utilizing scarce production resources. Computer scheduling is also more dynamic since it facilitates quick responses to changes in the availability of or demand for materials and facilities after production has started.

The benefits of the proposed computerized production planning system can be summarized as follows:

(1) Yield is improved by scrap reduction because of better scheduling and fewer errors.

(2) Small fluctuation in alloy quality and a tight, uniform furnace schedule is obtained.

CHAPTER VII

SUMMARY, CONCLUSION, AND EXTENSIONS

The MP problem presented in this thesis is but a sample of the type of problems that are becoming increasingly frequent in industry. This is expected to become even more important as robots and computerized controls start replacing the more traditional man-machine systems. Sharing of a "processor" or a pool of processors becomes a vital issue as all segments of production must feed data to, and receive information from, centralized or distributed data base systems.

The MP problem in this thesis was limited to two machines per processor and one resource type per job. Other restrictions were also imposed to make the model practical for use in the aluminum industry. Some of those restrictions can be removed easily, others will need restructuring of the model and of solution approach.

The difficulty of solving an MP model became evident. A simple model with a single objective of minimizing the total changeover cost in scheduling n resource constrained jobs on s parallel machines with ℓ interchangeable processors proved to be a challenging problem even for computers, and we now believe that the use of heuristics is inevitable.

Three methods were examined in this thesis. Algorithm I, the Least Cumulative Processing Time model, focused on always assigning jobs to the processor with the least cumulative processing time assigned. This proved to be a simple, economical, and reliable method that yielded reasonable total cost and makespan. Algorithm II, the Planning Horizon model, assigned jobs to the processor based upon the least changeover cost criteria until the planning horizon is reached. Algorithm III, the Bin-Packing Branch-and-Bound method was the most elegant approach combining decomposition with branch-and-bound algorithm. It was designed to provide a good feasible solution even when both Algorithms I and II fail to do so.

Algorithm III is developed based on the simple observation that if jobs with the same resource type usage are grouped together into a class and assigned to a processor, then we can eliminate the resource conflict. Algorithm III also serves as a comparison with Algorithm I and II and helps us to make a better decision to select a schedule. The reason is that both Algorithm I and II make a decision by choosing the next job with the least cost in the row of a cost matrix, but certain types of data may trap the algorithms into a bad solution. In order to avoid this situation, Algorithm III applies branch and bound methods to find the best sequence in a given subset of jobs.

A summary of the results of each of the three methods is given below.

(1) Algorithm I behaves consistantly well, it usually produces a least cost with minimum makespan, when n is small.

Algorithm II behaves inconsistantly. Sometimes it is good, sometimes it is bad. The bad result occurs very often because of poor decisions at the end of the sequence. The chance of failure is higher than with Algorithm I, when we give a planning horizon D which is close to the optimal makespan Z^*, however, if $D \gg Z^*$, a very poor makespan may occur.

Algorithm III uses First Fit Decreasing (FFD) method to achieve a good makespan. This algorithm may not be used under the following conditions:

(i) One machine is attached to one processor only, - in this case, we lose the advantage of permutating the job to achieve a better schedule.

(ii) When there is a great contrast in the property of jobs which uses the same resource type.

(2) The execution time of Algorithm I is faster than Algorithms II and III, Algorithm III is the slowest.

(3) In order to have a feasible and tight schedule, three algorithms produce a schedule with the assumption that all jobs have to be split over two machines equally. A manual permutated schedule is achieved by switching adjacent jobs which use the same resource type. A better schedule is usually obtained.

(4) These three algorithms are applied to a real industry scheduling problem. The results show that all three algorithms are better than the manual scheduling method.

Conclusions drawn from this research are given below.

The three heuristic methods presented here will help in finding a schedule that is better than a shop foreman can make up by hand and more economical. After a good and feasible schedule is obtained, any person will be able to improve the schedule so that more cost will be saved.

There is a healthy interaction between scheduling theory and practice in the field of operations research. This will continue to make scheduling problems a challenging research area.

Suggested future extensions of this research are:

(1) The manual permutation schedule procedure can be eliminated by modifying the heuristic algorithm developed by Armour (1961). Jobs with the same resource type and processing time can be pairwise interchanged. An improved schedule can be obtained after a series of sequential moves.

(2) Job priority or due dates are included in the scheduling.

(3) Removal of the requirement that all machines and processors must be identical.

(4) Consideration of precedence relationships among jobs.

As a final note to this thesis, the author wishes to point out that the insights gained concerning the MP-type problems and their significance in industry have both surpassed any expectation he had when the research began. The advances in hardware technology must be matched by our enhanced ability to handle the scheduling of increasingly costly and complex systems. The savings generated in our case study, up to a quarter of million dollars per year, are not trivial, but insignificant when compared to the potential that this type of research could lead to in all segments of our economy. Of an even greater importance is the hope that this research has given by making us realize that we can continue to create algorithms to match the complexity of future industrial systems.

BIBLIOGRAPHY

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. 1974. The Design and Analysis of Computer Algorithm, Addision Wesley. 470 p.

Armour, G. 1961. "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities," Unpublished Doctoral Dissertation, UCLA. Los Angeles, California.

Baase, S. 1978. Computer Algorithms: Introduction to Design and Analysis, First Edition, Addision Wesley. 286 p.

Barnes, J.W. and Brennan, J.J. 1977. "An Improved Algorithm for Scheduling Jobs on Identical Machines," A.I.I.E. Transactions. Vol. 9, No. 1. pp. 25-30.

Barker, K.B. 1974. Introduction to Sequencing and Scheduling, First Edition. John Wiley and Sons. New York. 299 p.

Bellmore, M. and Hong, S. 1974. "Transformation of Multisalesmen Problem to the Standard Traveling Salesman Problem," J. ACM, Vol. 21., No. 3. pp. 500-505.

Bellmore, M. and Malone, J.C., 1969. "Pathology of Traveling Salesman Subtour - Elimination Algorithms," Operations Research. Vol. 19, No. 2. pp. 278-300.

Bellmore, M. and Nemhauser, G. 1968. "The Traveling Salesman Problem: A Survey." Operations Research. Vol. 16. pp. 538-558.

Bodin, L.D. and Kursh, S.J. 1978. "A Computer Assisted System for the Routing and Scheduling of Street Sweepers." Operations Research. Vol. 26, No. 4. pp. 525-537.

Brown, A.R. 1971. Optimum Packing and Depletion: The Computer in Space and Resource Usage Problems. American Elsevier, Inc. New York. 107 p.

Carpaneto, G. and Toth P. 1977. "A New Algorithm for the Traveling Salesman Problem with Due Dates." Advances in Operations Research Proceedings of Euro II. The Second European Congress on Operations Research. North-Holland. pp. 95-102.

Coffman, E.G. 1976. "Computer and Job/Shop Scheduling Theory." John Wiley and Sons. New York. 299 p.

Coffman, E.G., Garey, M.R. and Johnson, D.S. 1978. "An Application of Bin-Packing to Multiprocessors Scheduling." SIAM J. Comput., Vol. 7, No. 1. pp. 1-17.

Conway, R.W., Maxwell, W.L. and Miller, L.W. 1967. Theory of Scheduling. Addison Wesley. 294 p.

Dantzig, J.D. and Ramser, J. 1959. "The Truck Dispatching Problem." Management Science. Vol. 6. pp. 80-91.

Davis, E.W. 1973. "Project Scheduling Under Resource Constraints: Historical Review and Categorization of Procedures." A.I.I.E. Transactions. Vol. 5, No. 4. pp. 297-313.

Day, J.E. and Hottenstein, M.P. 1970. "Review of Sequencing Research." Naval Research Logistics Quarterly. March. pp. 118-146.

Dinkel, J.J., Kleindorfer, G.B., Kochenberger, G.A. and Wong, S.N. 1976. "Environmental Inspection Routes and the Constrained Traveling System Salesman Problem." Computer and Operations Research. Pergamon Press. pp. 269-282.

Driscoll, W.C. and Emmons, H. 1977. "Scheduling Production on One Machine with Changeover Costs." A.I.I.E. Transactions. Vol. 9, No. 4. pp. 385-395.

Eastman, W.L. 1959. "A Solution to the Traveling Salesman Problem." Econometrica. Vol. 27. pp. 282-289.

Elmaghraby, S.E. 1968. "The Machine Sequencing Problem - Review and Extensions". Naval Research Logistic Quarterly. Vol. 15, No. 2, June. pp. 205-232.

Elmaghraby, S.E. 1968. "The One Machine Sequencing Problem with Delay Costs." Jounral of Industrial Engineering. Vol. XIX. No. 2. February. pp. 105-108.

Elmaghraby, S.E. and Elimam, A.A. 1980. "Knapsack-based Approaches to the Makespan Problem on Multiple Processors." A.I.I.E. Trasnsactions. Vol. 12, No. 1. pp. 87-96.

Elmaghraby, S.E. and Park, S.H. 1974. "Scheduling Jobs on a Number of Identical Machines." A.I.I.E. Transaction. Vol. 6, No. 1. pp. 1-13.

Ford, L.R. and Fulkerson, D.R. 1962. Flows in Networks. Princeton University Press. Princeton, New Jersey. 194 p.

Frederickson, G.N., Hecht, M.S. and Kim, C.E. 1978. "Approximation Algorithms for Some Routing Problem." Siam J. Compt. Vol. 7, No. 2. pp. 178-193.

Garcia, A.S. 1976. "School Scheduling on an Interactive Computer System." Unpublished Doctoral Dissertation. Stanford University, California. 98 p.

Garey, M.R., and Graham, R.L. 1975. "Bounds for Multiprocessor Scheduling with Resource Constraints," Siam J. Compt. Vol. 4, No. 2, June. pp. 187-200.

Garey, M.R., and Johnson, D.S. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. First Edition. W.H. Freeman and Company. 338 p.

Garey, M.R. and Johnson, D.S. 1975. "Complexity Results for Multiprocessor Scheduling under Resource Constraints," Siam J. Comput. Vol. 4, No. 4, Dec. pp. 397-411.

Geoffrion, A.M. and Marsten, R.E. 1972. "Integer Programming Algorithms: A Framework and State-of-Art Survey," Management Science, Vol. 18. No. 9. pp. 465-491.

Gavett, J. William. 1965. "Three Heuristic Rules for Sequencing Jobs," Management Science. Vol. 11, No. 8, June. pp. B166-B176.

Gillett, B.E. 1976. Introduction to Operations Research: A Computer Oriented Algorithmic Approach. First Edition, McGraw-Hill. 617 p.

Glassey, C.R. 1968. "Minimum Changeover Scheduling of Several Products on One Machine," Operations Research. Vol. 16, No. 2. pp. 343-352.

Gorenstein, S. 1970. "Printing Press Scheduling for Multi-Edition Periodicals," Management Science. Vol. 16, No. 6, Feb. pp. B373-B383.

Graham, R.L. 1969. "Bounds on Multiprocessing Timing Anomalies," SIAM J. on Applied Math. Vol. 17, No. 2. pp. 416-429.

Held, M. and Karp, R. 1962. "A Dynamic Programming Approach to Sequencing Problems," Siam J. Comput. Vol. 10, No. 1, March. pp. 25-60.

Horowitz, E. and Sahni, S. 1978. Fundamentals of Computer Algorithms. Computer Science Press. 626 p.

Horowitz, E. and Sahni, S. 1974. "Computing Partitions with Applications to the Knapsack Problem," J. ACM. Vol. 21, No. 2. pp. 277-292.

Knuth, P.E. 1975. The Art of Computer Programming, Vol. 1: Fundamental Algorithms. Addison-Wesley. 634 p.

Land, A.H. and Doig, A.G. 1960. "An Automatic Method for Solving Discrete Programming Problems," Econometrica. Vol. 28. pp. 497-520.

Lawler, E.L. 1964. "On Scheduling Problems with Deferral Cost," Management Science. Vol. 11, No. 2. Nov. pp. 280-288.

Little, J.D., Murty, K.G., Sweeny, D.W., and Karel, C. 1963. "An Algorithm for the Travelling Salesman Problem," Operations Research. Vol. 11. pp. 972-989.

Manne, A.S., 1960. "On the Job-Shop Scheduling Problem," Operations Research. Vol. 8, No. 2. pp. 219-223.

Mason, A.T. and Moodie, C.L. 1971. "A Branch and Bound for Minimizing Cost in Project Scheduling," Management Science. Vol. 18, No. 4, Dec. pp. B158-B173.

McNaughton, R. 1959. "Scheduling with Deadline and Loss Function," Management Science. Vol. 6, No. 1, Oct. pp. 1-12.

Moder, J.J. and Phillips, C.R. 1970. "Project Management with CPM and PERT," Van Nostrand, Reinhold Company. Second Edition. Chapter 8.

Poole, J.G. and Szymankiewicz. 1977. Using Simulation to Solve Problems. McGraw-Hill. 333 p.

Presby, J.T. and Wolfson, M.L. 1967. "An Algorithm for Solving Job Sequencing Problems," Management Science. Vol. 13, No. 8, April. pp. B454-B464.

Pritsker, A. and Kiviat, P.J. 1969. Simulation with GASP II. Englewood Cliffs, New Jersey: Prentice-Hall. 332 p.

Pritsker, A., Watters, L. and Wolfe, P. 1969. "Multiproject Scheduling with Limited Resources, A Zero-one Programming Approach," Management Science., Vol. 11. No. 3. Jan. pp. 93-108.

Ramalingam, P. 1969. "Optimizer for Single and Multi-Stage Job Shop Scheduling Problems," Unpublished Master Thesis. Oregon State University. Chapter 3.

Riggs, J. L. and Inoue, M.S. 1975. Introduction to Operations Research and Management Science: A General System Approach. McGraw-Hill. 497 p.

Root, J.G. 1965. "Scheduling with Deadlines and Loss Functions on the Parallel Machines," Management Science. Vol. 11, No. 3, Jan. pp. 460-475.

Rothkopf, M.H. 1966. "Scheduling Independent Tasks and Parallel Processors," Management Science. Vol. 12. No. 5. Jan. pp. 437-456.

Schild, A. and Fredman, I.J. 1961. "On Scheduling Tasks with Associated Linear Loss Functions," Management Science. Vol. 7. No. 3. pp. 280-285.

Schild, A. and Fredman, I.J. 1962. "Scheduling with Deadline and Nonlinear Loss Functions," Management Science. Vol. 9. No. 1. pp. 73-81.

Smith, W. 1956. "Various Optimizers for Single Stage Production", Naval Research Logistic Quarterly. Vol. 3. No. 1. March. pp. 59-66.

Svestka, J.A. and Huckfeldt, V.E. 1973. "Computational Experience with an M-Salesman Travling Salesman Algorithm," Management Science. Vol. 19. No. 7. March. pp. 790-799.

Thompson, W.J. 1970. "A General FORTRAN-based Package for Solving Sequencing Problems using Branch and Bound," Unpublished Ph.D. Dissertation. Arizona State University. 326 p.

Wagner, H.M. 1959. "An Integer Linear-programming Model for Machine Scheduling," Naval Research Logistic Quarterly. Vol. 6, No. 2. June. pp. 131-140.

APPENDIX A

ALGORITHM I

```
      PROGRAM ALGOP1(INPUT,OUTPUT,TAPE60=INPUT,TAPE61=OUTPUT)
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IR,INUM,ICM(30,30)
      COMMON/BLOCKX/IRES(20)
      COMMON/BLOCK3/IPST(20)
      COMMON/BLOCK4/IJOB(90),IPPCS(7),IPR(7),IPRS(7),T1
      COMMON/BLOCK5/OSCH(560,3),MOPR1(7,2),OPR2(7,3),IND(14)
      DIMENSION  ITEMP(7,2)
C
C *     DEFINITION  OF VARIABLES
C *     SJOBS CONTAINS MAXIMIUM OF 80 JOBS, EACH JOB IS CHARACTERERIZED
C *     SJOBS(I,1)  =  JOB TYPE
C *     SJOBS(I,2)  =  RESOURCE TYPE.
C *     SJOBS(I,3)  =  NO OF PROCESSING DAYS
C *     SJOBS(I,4)  =  TAG   = 0  NEW JOB
C *                          = 1  OLD JOB
C *     SJOBS(I,5)  = PROCESSOR IDENTIFICATION P:
C *                   P = PROCESSOR IDENTIFICATION , IF TAG = 1
C *     ICM   =  COST MATRIX
C *     IRES(I) = NO. OF RESOURCES OF TYPE I.
C *     IPROC(I) = NO OF MACHINES IN PROCESSOR I, 1≤IPROC≤2.
C *     IJOB = CURRENT JOBS TO BE SCHEDULED.
C *     IPPS(I)   =   THE CURRENT JOB IN PROCESSOR I.
C *     MOPR1(I,J) = STARTING LOCATION OF JOB SCHEDULE IN OSCH FOR
C *                  PROCESSOR I, MACHINE J
C *     OPR2(I,1)   = ACCUMULATED TOTAL COST ON PROCESSOR I.
C *     OPR2(I,2)   = TOTAL NO. OF DAYS SCHEDULED ON MACHINE ONE OF
C *                  PROCESSOR I.
C *     OPR2(I,3)   = TOTAL NO. OF DAYS SCHEDULED ON MACHINE 2 OF
C *     PROCESSOR I.
C *     OSCH(I,1) =  JOB TYPE
C *     OSCH(I,2) = RESOURCE TYPE.
C *     OSCH(I,3)   =   NO. OF PROCESSING DAYS.
C *
      IN=60
      JO=61
C      READ IN DATA
C
      CALL READIN
      CALL SECOND(T)
C
C
      TBEGIN = T
       INF = 999
C
C   INITIALIZE OUTPUT ARRAYS.
C
      CALL INITIAL
      CALL ASSIGN
C      ASSIGN JOBS TO OUTPUT ARRAYS
C      UPDATE THE QUEUE
      L=0
      DO 85 I=1,M
        JB=IPPS(I)
         L=L+1
          K=IND(L)
           OSCH(K,1)  =SJOBS(JB,1)
           OSCH(K,2)  =  SJOBS(JB,2)
        IR=SJOBS(JB,2)
C       DECREMENT RESOURCE TYPE BY 1
        IRES(IR) = IRES(IR)-1
        T1  =  SJOBS(JB,3)
        IF(IPROC(I)  .EQ.  2)  T1=T1/2.0
      OSCH(K,3)  =  T1
      IND(L)  =  K+1
      IF(IPROC(I)  .EQ.  1)  GO TO 82
      L=L+1
      K=IND(L)
      OSCH(K,1)=SJOBS(JB,1)
      OSCH(K,2)=SJOBS(JB,2)
      OSCH(K,3)  =  T1
      IND(L)  =K+1
   82  CONTINUE
C
C**    UPDATE TOTAL NO. OF DAYS ON   PROCESSORS I
C
      OPR2(I,2)=OPR2(I,2)+T1
      IF(IPRCC(I) .EQ. 2) OPR2(I,3) = OPR2(I,3) +T1
   85 CONTINUE
C
C**   SCHEDULE THE REMAINING JOBS TO MINIMIZE COST . A JOB IS ASSIGNED TO
C**     A PROCESSOR THAT WILL COMPLETE FIRST, THAT IS THE ONE  WHOSE
C**     ACCUMULATED NO. OF DAYS IS LEAST.
C**     THE NEXT JOB TO ASSIGNED TO A PROCESSOR IS THE ONE THAT WILL
C**     RESULT IN LEAST COST.  IN CASE, MORE THAN ONE JOB RESULT IN THE
C**     THE SAME LEAST COST, THEN CHOSE THE ONE THAT USE THE SAME
C**     RESOURCE TYPE AS THE PREVIOUS JOB ON THE PROCESSOR.
C**     IF THIS CONDITION DOES NOT HOLD THEN CHOOSE THE ONE WITH
C**     LONGEST PROCESSING DAYS.
      ITOT=M
   87 CONTINUE
C
C**     FIND A   PROCESSOR WITH LEAST NO. OF PROCESSING DAYS.
C
      T=OPR2(1,2)
      IP = 1
      DO 90 I= 1,M
      IF(OPR2(I,2) .GE. T) GO TO 90
      T = OPR2(I,2)
      IP = I
   90 CONTINUE
C**    SCHEDULE PROCESSOR IP
      K= 0
      DO 92 I=1,IP
      K=K+1
      IF(IPRCC(I) .EQ. 2) K=K+1
   92 CONTINUE
      KK= K
      K = IND(K) -1
C**   PREVIOUS RESOURCE USED WAS OF TYPE ITYPE.
       ITYPE = DSCH(K,2)
      IRES(ITYPE)  = IRES(ITYPE)+1
C
C**    FIND PREVIOUS JOB NO. SCHEDULED ON PROCESSOR IP ; JB.
C
      JB=IPRS(IP)
C
C**    SEARCH ALONG ROW JB OF COST MATRIX TO LOCATE THE NEXT JOB TO BE
C**    SCHEDULED ON PROCESSOR IP.
C**   FIND JOB(S) WITH LEAST COST AND STORE IN ITEMP (ITEMP(I,1)=COST,
C***   ITEMP(I,2)=JOB NO.)
      JNN= 0
      ITTOT=ITOT
      IPTYPE = 0
   93 CONTINUE
      ITMP = INF
      DO 95 I = 1,N
      ITT = SJOBS(I,2)
      IF(ITT .EQ. IPTYPE) GO TO 95
      IF(I .EQ. JNN) GO TO 95
      IF(ICM (JB,I) .GE. ITMP) GO TO 95
      ITMP=ICM(JB,I)
   95 CONTINUE
C**   IF ITMP = INIFINITY ALL JOBS HAVE BEEN SCHEDULED THEN GO TO
C**    PRINT OUTPUT.
C
      IF(ITMP .EQ. INF) GO TO 125
C      FIND ALL JOB WITH THE LEAST COST
      L = 0
       DO 97 I=1,N
        ITT = SJOBS(I,2)
         IF(ITT .EQ. IPTYPE) GO TO 97
          IF(I .EQ. JNN) GO TO 97
          IF(ICM(JB,I) .NE. ITMP) GO TO 97
          L=L+1
       ITEMP(L,1) = ITMP
      ITEMP(L,2) = I
   97 CONTINUE
C      IF L= 1  THEN ONLY ONE CANDIDATE JOB TO BE SCHEDULED.
      IF(L .EQ. 1) GO TO 110
C**   CHECK JOB THAT HAS SAME RESOURCE TYPE AS THE ONE PREVIOUSLY SCHED-
C**    ULED
      DO 100 I = 1,L
      L1 = ITEMP(I,2)
      ISJB = SJOBS(L1,2)
      IF(ISJB .EQ. ITYPE) GO TO 105
  100 CONTINUE
C      SCHEDULE JOB WITH LONGEST PROCESSING DAYS.
C
      L1 = ITEMP(1,2)
      TMD=SJOBS(L1,3)
      L2=1
      DO 102 I= 1,L
       L1 = ITEMP(I,2)
        IF(SJOBS(L1,3) .LE. TMD) GO TO 102
      TMD = SJOBS(L1,3)
      L2 = I
  102 CONTINUE
      ICOST = ITEMP(L2,1)
      JN = ITEMP(L2,2)
       GO TO 115
  105 ICOST=ITEMP(I,1)
      JN=ITEMP(I,2)
      GO TO 115
  110 ICOST = ITEMP(1,1)
      JN = ITEMP(1,2)
  115 CONTINUE
C     SCHEDULE JOB NO. JN. RESULTING IN ACOST OF ICOST
C      DECREMENT NO. OF RESOURCES OF THE RESOURCE TYPE USED BY JOB JN
C
      IR=SJOBS(JN,2)
C     CHECK IF RESOURCE TYPE HAS ANY AVAILABLE RESOURCE.
C
      IF(IRES(IR) .GT. 0) GO TO 113
      JNN = JN
       IRTYPE = IR
       ITTOT = ITTOT + 1
      IF(ITTOT .GT. N) GO TO 123
      GO TO 93
  113 IRES(IR) = IRES(IR)-1
C    UPDATE THE TOTAL COST ON PROCESSOR IP
      OPR2(IP,1)=OPR2(IP,1) + ICOST
C      UPDATE TOTAL NO. OF DAYS ON PROCESSOR IP
      T1 = SJOBS(JN,3)
      IF(IPROC(IP) .EQ. 2) T1 = T1/2.0
       OPR2(IP,2)= OPR2(IP,2) + T1
       IF(IPROC(IP) .EQ. 2) OPR2(IP,3) = OPR2(IP,3) + T1
C     UPDATE CURRENT JOB SCHEDULED ON PROCESSOR IP
      IPRS(IP) = JN
C     UPDATE THE OSCH QUEUE
      IF(IPROC(IP) .EQ. 2) KK=KK-1
      IND1=IND(KK)
      OSCH(IND1,1) = SJOBS(JN,1)
      OSCH(IND1,2)=SJOBS(JN,2)
      OSCH(IND1,3) = T1
      IND(KK) = IND1+1
      IF(IPROC(IP) .EQ. 1) GO TO 120
       KK = KK+ 1
      IND1= IND(KK)
      OSCH(IND1,1) = SJOBS(JN,1)
      OSCH(IND1,2) = SJOBS(JN,2)
      OSCH(IND1,3) = T1
      IND(KK) = IND1 + 1
  120 CONTINUE
C     SET COLUMN CORRESPONDING TO JOB IN TO INFINITY.
      DO 122 I=1,N
      ICM(I,JN) = INF
  122 CONTINUE
C      GO BACK AND SCHEDULE REMAINING JOBS
      ITOT = ITOT + 1
      GO TO 97
  123  WRITE(61,600)
  600  FORMAT(/// 5X,≠ NO. OF RESOURCE NOT  ENOUGH ≠)
      GO TO 130
  125  CONTINUE
C     UPDATE THE NO. OF RESOURCE TYPES USED LAST
C
      DO 130 I= 1,M
      IF(I .EQ. IP) GO TO 130
       L= IPRS(I)
```

```
            L1=SJOBS(L,2)
            IPES(L1) = IPES(L1) + 1
  130     CONTINUE
C**         PRINT RESULTS
          CALL SECOND(T)
          TOTALT = T-TBEGIN
          WRITE(JO,611) TOTALT
  611     FORMAT(//10X, *TIME FOR EXECUTION IS *,F10.2,* SECONDS*)
C           OUTPUT RESULT
          CALL OTPUT
  180     CONTINUE
            STOP
            END


          SUBROUTINE INITIAL
          COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(30,5)
          COMMON/BLOCK4/IJOB(20),IPPOS(7),IPR(7),IPRS(7),T1
          COMMON/BLOCK5/OSCH(560,3),MOPR1(7,2),OPR2(7,3),IND(14)
C
C       THIS ROUTINE INITIALIZE ALL ARRAYS
C
C
C       OUTPUT ARRAYS
C
          DO 35 I=1,560
          DO 35 J=1,3
          OSCH(I,J)=0
   35     CONTINUE
          DO 40 I=1,7
          DO 40 J=1,2
          MOPR1(I,J)=0
   40     CONTINUE
          DO 45 I=1,7
          DO 45 J=1,3
          OPR2(I,J)=0.0
   45     CONTINUE
          L=-19
          L1=0
          DO 50 I=1,M
          L=L+20
          MOPR1(I,1)=L
           L1=L1+1
          IND(L1)=L
          IF(IPROC(I) .EQ. 1) GOTO 50
          L=L+20
          MOPR1(I,2)=L
          L1=L1+1
          IND(L1)=L
   50     CONTINUE
          IF(M .GT. 1) GOTO 55
          MOPR1(1,1)=1
          IND(1)=1
            IF(IPROC(1) .EQ. 1) GOTO 55
            MOPR1(1,2)=31
          IND(2)=31
   55     CONTINUE
C
C       INITIALIZE IJOB AND IPROS
C
          DO 60 I=1,N
          IJOB(I)=0
          IPROS(I)=0
          IF(I.GT.M) GOTO 60
          IPR(I)=0
          IPRS(I)=0
   60     CONTINUE
          RETURN
          END
```

```
      SUBROUTINE READIN
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IR,INUM,ICM(80,80)
       COMMON/BLOCKX/IRES(20)
      DIMENSION IBLK(32)
C
C**   READ THE NUMBER OF PROCESSORS M AND NO. OF JOBS N
C*
      READ(IN,1000)M,N
 1000 FORMAT(2I5)
C
      WRITE(JO,1)M,N
    1 FORMAT(≠1≠//35X,≠THE SCHEDULING OUTPUT≠//10X,≠NO. OF PROCESSORS
     .ARE ≠,I5//10X,≠NO. OF JOBS ARE ≠,I5/)
      WRITE(JO,2)
    2 FORMAT(//10X,≠PROCESSOR≠,6X,≠MACHINES≠/)
C
C**   READ THE PROCESSOR IDENTIFICATION AND THE NO. OF MACHINE ON EACH P  C
C
      DO 10 I=1,M
      READ(IN,1000)IP,IM
      IPROC(IP)=IM
      WRITE(JO,3)IP,IM
    3 FORMAT(12X,I5,3X,I5/)
   10 CONTINUE
C     READ JOB DESCRIPTIONS
      WRITE(JO,13)
   13 FORMAT(//10X,≠JOB DESCRIPTION INPUT≠/)
      DO 20 I=1,N
       READ(IN,1300)(SJOBS(I,J),J=1,5)
       WRITE(JO,11)I,(SJOBS(I,J),J=1,5)
   11 FORMAT(//10X,≠(≠,I2,≠)≠,3X,≠(≠,2(F3.0,≠,≠),F4.1,≠,≠,F2.0,≠,≠,F2.0
     .,≠)≠)
   20 CONTINUE
 1300 FORMAT(2F5.0,F5.1,2F5.0)
C
C**   READ NUMBER OF RESOURCE TYPE,IR
C
      READ(IN,1000)IR
      WRITE(JO,16)IR
   16 FORMAT(////10X,≠NO. OF RESOURCE TYPE =≠,I5//10X,≠TYPE≠,2X,≠QUANT≠
     ./)
C**   READ NUMBER OF RESOURCE TYPE,IR.
C
      DO 33 I=1,IR
      READ(IN,1000)IPT,INUM
      WRITE(JO,17)IPT,INUM
   17 FORMAT(10X,I5,2X,I5)
      IRES(IPT)=INUM
   33 CONTINUE
C
      IF(N .GT. 32) GOTO 99
       DO 33 I=1,N
       IBLK(I)=I
   33 CONTINUE
      WRITE(JO,4)
    4 FORMAT(//34X,≠COST MATRIX≠,/1H,34X,≠-----------≠///)
      WRITE(JO,9)(IBLK(J),J=1,N)
    9 FORMAT(5X,32(≠(≠,I2,≠)≠)///)
C
C**    READ THE COST MATRIX WHICH N IS LESS THEN 21
C
       DO 15 I=1,N
       READ(IN,1200)(ICM(I,J),J=1,N)
       WRITE(JO,12)I,(ICM(I,J),J=1,N)
   12  FORMAT(1H0,≠(≠,I2,≠)≠,1X,32(I3,1X))
   15  CONTINUE
 1200  FORMAT(32I3)
       GOTO 111
   99  CONTINUE
       DO 41 I=1,N
        DO 42 J=1,N
       ICM(I,J)=0
       CALL RAN1(UR)
       ICM(I,J)=INT(UR*100)
       IF(I.EQ.J) ICM(I,J)=999
   42  CONTINUE
   41  CONTINUE
  111  RETURN
       END
      SUBROUTINE ASSIGN
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IR,INUM,ICM(80,80)
      COMMON/BLOCKX/IRES(20)
      COMMON/BLOCK3/IRST(20)
      COMMON/BLOCK4/IJOB(80),IPROS(7),IPP(7),IPRS(7),T1
      DIMENSION DAYS(80,2)
C
C     THIS ROUTINE IS TO FIND OLD JOBS LEFT OVER AND ASSIGN
C     THEM TO RESPECTIVE PROCESSORS FIRST
C
C     INF=999
C
      INF=999
      K=0
      DO 65 I=1,N
      IF(SJOBS(J,4) .NE. 1.0) GOTO 65
      K=K+1
      IJOB(I)=I
      IPROS(I)=SJOBS(I,5)
      IPP(K)=SJOBS(I,5)
   65 CONTINUE
C
C     ASSIGN INITIAL JOBS TO UNOCCUPIED  PROCESSORS.  INITIAL JOBS ARE
C     ASSIGNED TO THE PROCESSOR WITH THE LEAST CUMULATIVE PROCESSING
C     TIME
C
      IF(K.GE.M) GOTO 78
      L=0
      I=0
   66 I=I+1
      IF(I .GT. N) GOTO 67
      IF(IJOB(I) .NE. 0) GOTO 66
      L=L+1
      DAYS(L,1)=SJOBS(I,3)
      DAYS(L,2)=I
      GOTO 66
   67 CONTINUE
      N1=L-1
      DO 68 II=1,N1
      IPLUS1=II+1
      DO 68 JJ=IPLUS1,L
      IF(DAYS(II,1) .LE. DAYS(JJ,1)) GOTO 68
      T1=DAYS(II,1)
      T2=DAYS(II,2)
      DAYS(II,1)=DAYS(JJ,1)
      DAYS(II,2)=DAYS(JJ,2)
      DAYS(JJ,1)=T1
      DAYS(JJ,2)=T2
   68 CONTINUE
      IF(K .EQ. 0) GOTO 70
       DO 69 I=1,K
       I1=IPP(I)
       IRST(I)=IRES(I1)-1
   69 CONTINUE
   70 CONTINUE
      I=0
   71 K1=0
   72 I=I+1
      T1=DAYS(I,1)
      IT=DAYS(I,2)
      IF(K .EQ. 0) GOTO 75
      DO 73 L=1,K
      L1=IPP(L)
      IF(SJOBS(IT,2) .EQ. SJOBS(L1,2)) GOTO 74
   73 CONTINUE
      GOTO 75
   74 IF(IRST(L) .LE. 0) GOTO 72
   75 CONTINUE
      IJOB(IT)=IT
   76 K1=K1+1
      IF(K .LT. 1) GO TO 88
      DO 77 J=1,K
      IF(IPP(J) .EQ. K1) GO TO 76
   77 CONTINUE
   88 CONTINUE
      IPROS(IT)=K1
      K=K+1
      IPR(K)=K1
      IF(K .LT. 9) GOTO 71
   78 CONTINUE
      DO 79 I=1,N
      IF(IPROS(I) .EQ. 0) GO TO 79
      K1=IPROS(I)
      IPPS(K1)=IJOB(I)
   79 CONTINUE
C
C     SET THE COLUMN OF THE COST MATRIX CORRESPONDING TO THE
C     JOB TO INFINITY
C
      DO 81 I=1,N
       IF(IJOB(I) .EQ. 0) GOTO 81
       JJ=IJOB(I)
       DO 80 J=1,N
       ICM(J,JJ)=INF
   80 CONTINUE
   81 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE RAN1(UR)
C     RANDOM NUMBER GENERATOR
      IR=IIIIIII
      IR=AND(AND(2069*IR,8388607)+1772721,8388607)
      UR=FLOAT(IR)/8388607
      RETURN
      END
```

```
      SUBROUTINE OTPUT
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK5/OSCH(360,3),MOPR1(7,2),OPR2(7,3),IND(14)
C
C     THIS ROUTINE IS TO PRINT RESULTS
C
      TCOST=0.
      DO 150 I=1,4
      WRITE(JO,500)I
  500 FORMAT(//5X,*PROCESSOR NO.*, I3)
      WRITE(JO,510)
  510 FORMAT(//5X,*MACHINE NO. 1.*//)
      WRITE(JO,520)
  520 FORMAT(/15X,*JOB TYPE*,5X,*RES. TYPE*,5X,*NO. OF PROCESSING DAYS*
     ./)
      L1=MOPR1(I,1)
  135 WRITE(JO,530)OSCH(L1,1),OSCH(L1,2),OSCH(L1,3)
  530 FORMAT(8X,F5.0,9X,F5.0,9X,F10.2)
      L1=L1+1
      IF(OSCH(L1,1) .NE. 0.0) GOTO 135
      IF(IPROC(I) .EQ. 1) GOTO 145
      WRITE(JO,540)
  540 FORMAT(//5X,*MACHINE NO. 2*/)
      WRITE(JO,520)
      L1=MOPR1(I,2)
  140 WRITE(JO,530)OSCH(L1,1),OSCH(L1,2),OSCH(L1,3)
      L1=L1+1
      IF(OSCH(L1,1).NE. 0.0) GOTO 140
  145 WRITE(JO,550)I,OPR2(I,1)
  550 FORMAT(//5X,*TOTAL COST ON PROCESSOR NO. *,I2,* = *,F10.2)
      TCOST=TCOST+OPR2(I,1)
  150 CONTINUE
      WRITE(JO,560)TCOST
  560 FORMAT(///5X,*TOTAL PROCESSING COST =*,F10.2)
      RETURN
      END
```

APPENDIX B

ALGORITHM II

```
      PROGRAM PLANHOR(INPUT,OUTPUT,TAPE60=INPUT,TAPE61=OUTPUT)
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IR,INUM,ICM(30,80)
      COMMON/BLOCKX/IRUSE(80,20),IRES(20)
      COMMON/BLOCK3/IPST(20)
      COMMON/BLOCK4/IJOB(30),IPRCS(7),IFF(7),IPRS(7),T1
      COMMON/BLOCK5/OSCH(500,3),MOPR1(7,2),OPR2(7,3),IND(14)
      DIMENSION ITEMP(7,2),ICPR(7,4),NPS(7,80)
C*****************************************************************************
C *    DEFINITION  OF VARIABLES
C *    SJOBS CONTAINS MAXIMIUM OF 80 JOBS, EACH JOB IS CHARACTERERIZED
C *    SJOBS(I,1) = JOB TYPE
C *    SJOBS(I,2) = RESOURCE TYPE.
C *    SJOBS(I,3) = NO OF PROCESSING DAYS.
C *    SJOBS(I,4) = TAG  = 0  NEW JOB
C *                      = 1  OLD JOB
C *    SJOBS(I,5) = PROCESSOR IDENTIFICATION P*
C *               0 = PROCESSOR IDENTIFICATION , IF TAG = 1
C *    NPS = USED TO KEEP TRACK OF JOBS THAT CANNOT BE PROCESSED ON A
C *     PARTICULAR PROCESSOR DUE TO INSUFFICIENT RESOURCES.
C *    ICM  =  COST MATRIX
C *    IRES(I) = NO. OF RESOURCES OF TYPE I.
C *    IPROC(I) = NO OF MACHINES IN PROCESSOR I, 1≤IPROC≤2.
C *    IJOB = CURRENT JOBS TO BE SCHEDULED.
C *    IPRS(I)  =  THE CURRENT JOB IN PROCESSOR I.
C *    MOPR1(I,J) = STARTING LOCATION OF JOB SCHEDULE IN OSCH FOR
C *                 PROCESSOR I, MACHINE J
C *    OPR2(I,1)  = ACCUMULATED TOTAL COST ON PROCESSOR I.
C *    OPR2(I,2)  = TOTAL NO. OF DAYS SCHEDULED ON MACHINE ONE OF
C *                 PROCESSOR I.
C *    OPR2(I,3)  = TOTAL NO. OF DAYS SCHEDULED ON MACHINE 2 OF
C *    PROCESSOR I.
C *    OSCH(I,1) = JOB TYPE
C *    OSCH(I,2) = RESOURCE TYPE.
C *    OSCH(I,3)  =  NO. OF PROCESSING DAYS.
C *    ICPR(I,1) = PROCESSOR I
C *    ICPR(I,2) = JOB NUMBER.
C *    ICPR(I,3) = ICOST
C *    ICPR(I,4) = PREVIOUS RESOURCE TYPE.
C *
C*****************************************************************************
      IN=60
      JO=61
C
C*       INPUT ALL PARAMETERS
C
      CALL READIN
C       INPUT THE DESIRE PLANNING HORIZON
C
      READ(IN,1)TOTDYS
    1 FORMAT(F5.1)
      WRITE(JO,2)TOTDYS
    2 FORMAT(//10X,* THE SCHEDULING HORIZON IS *,F5.1,* DAYS*)
C     FIND THE OPTIMAL FINISHING TIME
C
      XSUM=0.
      DO 3 I=1,N
    3 XSUM=XSUM+SJOBS(I,3)
      MACH=0
      DO 4  I=1,M
    4 MACH=MACH+IPROC(I)
      FT = XSUM/MACH
C     CHECK FOR FEASIBLE SCHEDULE
      IF(TOTDYS .GT. FT) GOTO 5
      WRITE(JO,6)TOTDYS
    6 FORMAT(///10X,*  NOT ALL JOBS CAN BE SCHEDULED WITHIN THE DURATION OF *,
     1  DURATION OF *,F5.1,*  DAYS *)
      GO TO 999
C
C         COMPUTE THE RUNNING TIME
C
    5 CALL SECOND(T)
      TBEGIN = T
C
C       INITIALIZATION
      CALL INITIAL
C
```

```
      CALL ASSIGN
      ITJ = 0
       INF = 999
C
C**    ASSIGN JOBS TO OUTPUT ARRAYS
C**    UPDATE THE QUEUE OSCH
C
      L=0
      DO 85 I=1,M
       JB=IPRS(I)
        L=L+1
         K=INO(L)
          OSCH(K,1) =SJOBS(JB,1)
          OSCH(K,2) = SJOBS(JB,2)
        IR=SJOBS(JB,2)
       T1 = SJOBS(JB,3)
       IF(IPROC(I) .EQ. 2) T1=T1/2.0
      OSCH(K,3) = T1
      INO(L) = K+1
      IF(IPROC(I) .EQ. 1) GO TO 82
      L=L+1
      K=INO(L)
      OSCH(K,1)=SJOBS(JB,1)
      OSCH(K,2)=SJOBS(JB,2)
      OSCH(K,3) = T1
      INO(L) =K+1
   82  CONTINUE
C
C**    UPDATE TOTAL NO. OF DAYS ON  PROCESSORS I
C
      TO = OPR2(I,2)
      OPR2(I,2)=OPR2(I,2)+T1
      IF(IPROC(I) .EQ. 2) OPR2(I,3) = OPR2(I,3) +T1
C**    UPDATE RESOURCE MATRIX FOR RESOURCE TYPE IR.
      CALL UPRES(TO,T1,IR)
   85 CONTINUE
C
      ITJ = M
C**   SCHEDULE THE REMAINING JOBS TO MINIMIZE COST . A JOB IS ASSIGNED TO
C**    A PROCESSOR THAT WILL COMPLETE FIRST, THAT IS THE ONE  WHOSE
C**    ACCUMULATED NO. OF DAYS IS LEAST.
C**    THE NEXT JOB TO ASSIGNED TO A PROCESSOR IS THE ONE THAT WILL
C**    RESULT IN LEAST COST.  IN CASE, MORE THAN ONE JOB RESULT IN THE
C**    THE SAME LEAST COST, THEN CHOSE THE ONE THAT USE THE SAME
C**    RESOURCE TYPE AS THE PREVIOUS JOB ON THE PROCESSOR.
C**    IF THIS CONDITION DOES NOT HOLD THEN CHOOSE THE ONE WITH
C**    LONGEST PROCESSING DAYS.
C
   85 CONTINUE
      DO 93 II = 1,M
      DO 93 JJ = 1,N
       NPS(II,JJ) = 0
   93 CONTINUE
   87 CONTINUE
C
      DO 90 I=1,M
       DO 90 J=1,4
        ICPR(I,J)=0
   90  CONTINUE
      DO 200 IP = 1,M
C
C**    SCHEDULE PROCESSOR IP
C
      K= 0
      DO 92 I=1,IP
      K=K+1
      IF(IPROC(I) .EQ. 2) K=K+1
   92 CONTINUE
      K = INO(K) -1
C**   PREVIOUS RESOURCE USED WAS OF TYPE ITYPE.
       ITYPE = OSCH(K,2)
C
C**    FIND PREVIOUS JOB NO. SCHEDULED ON PROCESSOR IP : JB.
C
      JB=IPRS(IP)
C
C**    SEARCH ALONG ROW JB OF COST MATRIX TO LOCATE THE NEXT JOB TO BE
```

```
C**        SCHEDULED ON PROCESSOR IP.
C**        FIND JOB(S) WITH LEAST COST AND STORE IN ITEMP (ITEMP(I,1)=COST,
C***        ITEMP(I,2)=JOB NO.)
           ITMP = INF
           DO 95 I = 1,N
           IF(NPS(IP,I) .EQ. 1) GO TO 95
           IF(ICM (JB,I) .GE. ITMP) GO TO 95
           ITMP=ICM(JB,I)
        95 CONTINUE
C
C**         IF ITMP = INFINITY, NO MORE JOBS CAN BE SCHEDULED ON PROCESSOR IP
C
           IF(ITMP .EQ. INF) GO TO 190
C           FIND ALL JOB WITH THE LEAST COST
           L = 0
            DO 97 I=1,N
           IF(NPS(IP,I) .EQ. 1) GO TO 97
               IF(ICM(JB,I) .NE. ITMP) GO TO 97
               L=L+1
            ITEMP(L,1) = ITMP
           ITEMP(L,2) = I
        97 CONTINUE
C           IF L= 1  THEN ONLY ONE CANDIDATE JOB TO BE SCHEDULED.
           IF(L .EQ. 1) GO TO 110
C**        CHECK JOB THAT HAS SAME RESOURCE TYPE AS THE ONE PREVIOUSLY SCHED-
C**         ULED
           DO 100 I = 1,L
           L1 = ITEMP(I,2)
           ISJB = SJOBS(L1,2)
           IF(ISJB .EQ. ITYPE) GO TO 105
       100 CONTINUE
C         SCHEDULE JOB WITH LONGEST PROCESSING DAYS.
C
           L1 = ITEMP(1,2)
           TMO=SJOBS(L1,3)
           L2=1
           DO 102 I= 1,L
            L1 = ITEMP(I,2)
             IF(SJOBS(L1,3) .LE. TMO) GO TO 102
           TMO = SJOBS(L1,3)
           L2 = I
       102 CONTINUE
           ICOST = ITEMP(L2,1)
           JN = ITEMP(L2,2)
            GO TO 115
       105 ICOST=ITEMP(I,1)
           JN=ITEMP(I,2)
           GO TO 115
       110 ICOST = ITEMP(1,1)
           JN = ITEMP(1,2)
       115 CONTINUE
C          SCHEDULE JOB NO. JN, RESULTING IN ACOST OF ICOST
C           DECREMENT NO. OF RESOURCES OF THE RESOURCE TYPE USED BY JOB JN
C
           IR=SJOBS(JN,2)
           IOPR(IP,1) = IP
           IOPR(IP,2)=JN
           IOPR(IP,3) = ICOST
           IOPR(IP,4) = ITYPE
           GO TO 200
       190 IOPR(IP,1) = 0
           IOPR(IP,2) = 0
           IOPR(IP,3) = INF
           IOPR(IP,4) = 0
       200 CONTINUE
           M1 = M-1
           DO 202 I= 1,M1
           IPL1 = I+ 1
           DO 202 J= IPL1,M
           IF(IOPR(I,3) .LE. IOPR(J,3)) GO TO 202
           IT1 = IOPR(I,1)
           IT2 = IOPR(I,2)
           IT3 = IOPR(I,3)
           IT4 = IOPR(I,4)
           IOPR(I,1) = IOPR(J,1)
           IOPR(I,2)=IOPR(J,2)
           IOPR(I,3)=IOPR(J,3)
```

```
            IOPR(I,4) = IOPR(J,4)
            IOPR(J,1)=IT1
            IOPR(J,2) = IT2
            IOPR(J,3)=IT3
            IOPR(J,4) = IT4
     202    CONTINUE
            IT1 = 0
            IF(IOPR(1,3) .EQ. INF) GO TO 123
            IT1 = IT1 + 1
            IF (IT1 .GT. M) GO TO 123
            IF(IOPR(IT1,3) .EQ. INF) GO TO 123
            IP = IOPR(IT1,1)
            JN=IOPR(IT1,2)
            ICOST=IOPR(IT1,3)
            ITYPE=IOPR(IT1,4)
            T1 = SJOBS(JN,3)
            IF(IPROC(IP) .EQ. 2) T1=T1/2.0
            TDAYS = OPR2(IP,2) + T1
            IF(TDAYS .GT. TOTDYS) GO TO 204
            IK=SJOBS(JN,2)
C**         CHECK IF JOB JN CAN BE SCHEDULED ON PROCESSOR IP.
            TO = OPR2(IP,2)
            IF(ICRES(IR,TO,T1) .EQ. 0) GO TO 210
     204    NFS(IP,JN) = 1
            GO TO 47
C         UPDATE THE TOTAL COST ON PROCESSOR IP
     210    OPR2(IP,1)=OPR2(IP,1) + ICOST
C             UPDATE TOTAL NO. OF DAYS ON PROCESSOR IP
             OPR2(IP,2)= OPR2(IP,2) +T1
             IF(IPROC(IP) .EQ. 2) OPR2(IP,3) = OPR2(IP,3) + T1
C           UPDATE CURRENT JOB SCHEDULED ON PROCESSOR IP
            IPPS(IP) = JN
C            UPDATE THE OSCH QUEUE AND THE RESOURCE ARRAY
            CALL UPRES(TO,T1,IP)
            KK= 0
            IP1 = IP-1
            IF(IP1 .LE. 0) GO TO 206
            DO 205 I= 1,IP1
            KK = KK+ 1
            IF(IPROC(I) .EQ. 2) KK=KK+1
     205    CONTINUE
     206    KK = KK+1
            IND1=IND(KK)
            OSCH(IND1,1) = SJOBS(JN,1)
            OSCH(IND1,2)=SJOBS(JN,2)
            OSCH(IND1,3) = T1
            IND(KK) = IND1+1
            IF(IPROC(IP) .EQ. 1) GO TO 120
             KK = KK+ 1
            IND1= IND(KK)
            OSCH(IND1,1) = SJOBS(JN,1)
            OSCH(IND1,2) = SJOBS(JN,2)
            OSCH(IND1,3) = T1
            IND(KK) = IND1 + 1
     120    CONTINUE
C           SET COLUMN CORRESPONDING TO JOB IN TO INFINITY.
            DO 122 I=1,N
            ICM(I,JN) = INF
     122    CONTINUE
C            GO BACK AND SCHEDULE REMAINING JOBS
            ITJ=ITJ+1
            GO TO 66
     123    CONTINUE
            IF(ITJ .EQ. N) GO TO 125
            WRITE(61,600)
     600    FORMAT(///5X,≠***   NOT ALL JOBS COULD BE SCHEDULED   ***≠)
     125     CONTINUE
C**           PRINT RESULTS
            CALL SECOND(T)
            TOTALT = T-TBEGIN
            WRITE(JO,611) TOTALT
     611    FORMAT(//10X, ≠TIME FOR EXECUTION IS ≠,F10.2,≠ SECONDS≠)
C
C           PRINT RESULTS
            CALL OTPUT
     999    STOP
             END
```

```
      SUBROUTINE READIN
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IP,INUM,ICM(80,80)
       COMMON/BLOCKX/IRUSE(60,20),IRES(20)
      DIMENSION IBLK(32)
C
C**   READ THE NUMBER OF PROCESSORS M AND NO. OF JOBS N
C*
      READ(IN,1000)M,N
 1000 FORMAT(2I5)
C
      WRITE(JO,1)M,N
    1 FORMAT(*1*//35X,*THE SCHEDULING OUTPUT*//10X,*NO. OF PROCESSORS
     .ARE *,I5//10X,*NO. OF JOBS ARE *,I5/)
      WRITE(JO,2)
    2 FORMAT(//10X,*PROCESSOR*,6X,*MACHINES*/)
C
C**   READ THE PROCESSOR IDENTIFICATION AND THE NO. OF MACHINE ON EACH P  C      PROCE
C
      DO 10 I=1,M
      READ(IN,1000)IP,IM
      IPROC(IP)=IM
      WRITE(JO,3)IP,IM
    3 FORMAT(12X,I5,3X,I5/)
   10 CONTINUE
C     READ JOB DESCRIPTIONS
      WRITE(JO,13)
   13 FORMAT(//10X,*JOB DESCRIPTION INPUT*/)
      DO 20 I=1,N
       READ(IN,1300)(SJOBS(I,J),J=1,5)
       WRITE(JO,11)I,(SJOBS(I,J),J=1,5)
   11 FORMAT(//10X,*(*,I2,*)*,3X,*(*,2(F3.0,*,*),F4.1,*,*,F2.0,*,*,F2.0
     .,*)*)
   20 CONTINUE
 1300 FORMAT(2F5.0,F5.1,2F5.0)
C
C**   READ NUMBER OF RESOURCE TYPE,IR
C
      READ(IN,1000)IR
      WRITE(JO,16)IR
   16 FORMAT(////10X,*NO. OF RESOURCE TYPE =*,I5//10X,*TYPE*,2X,*QUANT*
     ./)
C**   READ NUMBER OF RESOURCE TYPE,IR.
C
      DO 30 I=1,IR
      READ(IN,1000)IRT,INUM
      WRITE(JO,17)IRT,INUM
   17 FORMAT(10X,I5,2X,I5)
      IRES(IRT)=INUM
   30 CONTINUE
C
      IF(N .GT. 32) GOTO 99
       DO 33 I=1,N
       IBLK(I)=I
   33 CONTINUE
      WRITE(JO,4)
    4 FORMAT(//34X,*COST MATRIX*,/1H,34X,*-----------*///)
      WRITE(JO,9)(IBLK(J),J=1,N)
    9 FORMAT(5X,32(*(*,I2,*)*)//)
C
C**    READ THE COST MATRIX WHICH N IS LESS THEN 21
C
       DO 15 I=1,N
       READ(IN,1200)(ICM(I,J),J=1,N)
       WRITE(JO,12)I,(ICM(I,J),J=1,N)
   12  FORMAT(1H0,*(*,I2,*)*,1X,32(I3,1X))
   15  CONTINUE
 1200  FORMAT(32I3)
       GOTO 111
   99  CONTINUE
       DO 41 I=1,N
        DO 42 J=1,N
       ICM(I,J)=0
       CALL RAN1(UR)
       ICM(I,J)=INT(UR*100)
       IF(I .EQ. J) ICM(I,J)=999
   42  CONTINUE
   41  CONTINUE
  111  RETURN
       END
      SUBROUTINE INITIAL
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCKX/IRUSE(60,20),IRES(20)
      COMMON/BLOCK4/IJOB(60),IPROS(7),IPR(7),IPRS(7),I1
      COMMON/BLOCK5/OSCH(560,3),MOPF1(7,2),OPF2(7,3),INO(14)
C
C     THIS ROUTINE INITIALIZE ALL ARRAYS
C
      DO 32 I=1,60
      DO 32 J=1,20
        IRUSE(I,J)=0
   32 CONTINUE
C
C     OUTPUT ARRAYS
C
      DO 35 I=1,560
      DO 35 J=1,3
      OSCH(I,J)=0
   35 CONTINUE
      DO 40 I=1,7
      DO 40 J=1,2
      MOPF1(I,J)=0
   40 CONTINUE
      DO 45 I=1,7
      DO 45 J=1,3
      OPF2(I,J)=0.0
   45 CONTINUE
      L=-19
      L1=0
      DO 50 I=1,M
      L=L+20
      MOPF1(I,1)=L
      L1=L1+1
      INO(L1)=L
      IF(IPROC(I) .EQ. 1) GOTO 50
      L=L+20
      MOPF1(I,2)=L
      L1=L1+1
      INO(L1)=L
   50 CONTINUE
      IF(M .GT. 1) GOTO 55
      MOPF1(1,1)=1
      INO(1)=1
       IF(IPROC(1) .EQ. 1) GOTO 55
       MOPF1(1,2)=31
      INO(2)=31
   55 CONTINUE
C
C     INITIALIZE IJOB AND IPROS
C
      DO 60 I=1,N
      IJOB(I)=0
      IPROS(I)=0
      IF(I.GT.M) GOTO 60
      IPR(I)=0
      IPRS(I)=0
   60 CONTINUE
      RETURN
      END


      SUBROUTINE UPRES(TO,T1,IR)
      COMMON/BLOCKY/IRUSE(60,20),IRES(20)
      I=TO*2
      J= T1*2
      IJ = I+J
      I=I+1
      DO 10 L=I,IJ
      IRUSE(L,IR) = IRUSE(L,IR) + 1
   10 CONTINUE
      RETURN
      END

      FUNCTION ICRES(IR,TO,T1)
      COMMON/BLOCKX/IRUSE(60,20),IRES(20)
      ICRES = 0
      I= TO*2
      J=T1*2
      IJ= I+ J
      I=I+1
      DO 10 L=I,IJ
      IF(IRUSE(L,IR) .GE. IRES(IR)) ICRES = 1
   10 CONTINUE
      RETURN
      END
```

APPENDIX C

ALGORITHM III

(BINBAB)

```
      PROGRAM BINBAB(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      COMMON/MATCOM/ ICOST(20,20),K
      COMMON/BLOCK1/ IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IR,INUM,IRES(20),ICM(80,80)
      COMMON/BLOCK3/LINK(80),LIST(80),ITOP(20),IPTOP(7),TOTSTK(20)
      COMMON/BLOCK4/TOTP(7),TOTMAX(7)
      EXTERNAL TIMF,STKO
      DATA IN,JO/5,6/,TOTSTK/20*0/
C
C     CALL INPUT SUBROUTINE
      CALL READIN
C       COMPUTE RUNNING TIME
C
      CALL SECOND(T)
      TBEGIN=T
C
C     FIND OPTIMAL FINISH TIME
      SUM=0.
      DO 100 I=1,N
  100 SUM=SUM+SJOBS(I,3)
      MACH=0
      DO 105 I=1,M
  105 MACH=MACH+IPROC(I)
      IFT=(SUM/MACH)*2+0.9999
      WRITE(JO,*)*THE OPTIMAL FINISH TIME IS *,IFT
      DO 108 I=1,M
  108 TOTMAX(I)=(IFT+1)/(3-IPROC(I))
C
C     SORT THE JOBS INTO THE LIST
      DO 110 I=1,N
  110 LIST(I)=I
      CALL BSORT(LIST,TIMF,N)
C
C     FILL THE STACKS USING FIRST FIT DECREASING
      DO 115 I=1,20
      ITOP(I)=0
  115 CONTINUE
C      IF ONE OR MORE PROCESSORS HAVE ONE MACHINE ATTACH TO IT
C      SET  ISINGLE=1.  OTHERWISE ISINGLE=0
      READ(IN,1)ISINGLE
    1 FORMAT(I1)
      IF(ISINGLE .NE. 1)GOTO 10
      IFT=IFT/2
   10 CALL FILL(IFT,NUMSTK)
C     FIND THE PREVIOUS JOBS EXECUTING ON EACH PROCESSOR.
C     PUT THE STACKS THAT CONTAIN THEM FIRST ON THE PROCESSOR STACKS AND
C     PUT THOSE JOBS FIRST IN EACH STACK
      DO 200 I=1,NUMSTK
      J=ITOP(I)
  120 IF(J.EQ.0)GOTO 200
      IF(SJOBS(J,4).NE.1.) GOTO 140
      IF(J.EQ.ITOP(I)) GOTO 130
      LINK(LSTJ)=LINK(J)
      LINK(J)=ITOP(I)
  130 IPTOP(SJOBS(J,5))=J
      ITOP(I)=999
      TOTP(SJOBS(J,5))=TOTSTK(I)
      J=0
      GOTO 120
  140 LSTJ=J
      J=LINK(J)
      GOTO 120
  200 CONTINUE
C
C      SORT THE STACKS  INTO THE LIST
       DO 210 I=1,NUMSTK
  210 LIST(I)=I
      CALL BSORT(LIST,STKO,NUMSTK)
C
C      FILL THE PROCESSOR STACKS ACCORDING TO THE
C      FIRST FIT DECREASING METHOD
      CALL PFILL(NUMSTK)
C
C     ASSIGN THE SUBCOST MATRIX
      DO 590 IP=1,M
      L=1
      K=1
      I=IPTOP(IP)
      J=I
  510 IF(J.EQ.0)GOTO 540
  520 IF(I.EQ.0)GOTO 530
      ICOST(K,L)=ICM(I,J)
      I=LINK(I)
      K=K+1
      GOTO 520
  530 CONTINUE
      I=IPTOP(IP)
      K=1
      J=LINK(J)
      L=L+1
      GOTO 510
  540 CONTINUE
C
C     PRINT OUT THE SUB COST MATRIX
C      ALONG WITH THE LIST OF JOBS IN THE PROCESSOR STACK
      J=IPTOP(IP)
      K=1
  610 IF(J.EQ.0)GOTO 640
      LIST(K)=J
      J=LINK(J)
      K=K+1
      GOTO 610
  640 K=K-1
      WRITE(JO,*)≠JOBS FOR PROCESSOR ≠,IP,≠ ARE -≠,(LIST(J),≠-≠,J=1,K)
      WRITE(JO,900)IP,IPROC(IP),TOTP(IP)
  900 FORMAT(≠ FINISH TIME FOR PROCESSOR ≠,I2,≠ WITH ≠,I3,
     1      ≠   PROCESSOR IS ≠,F6.2//)
      WRITE(JO,901)(LIST(J),J=1,K)
  901 FORMAT(20X,20(1X,≠(≠,I3,≠)≠,1X))
      DO 650 J=1,K
  650 WRITE(JO,902)LIST(J),(ICOST(J,I),I=1,K)
C
  902 FORMAT(13X,≠(≠,I3,≠)≠,20(2X,I3,2X)///)
C
C     USE BRANCH AND BOUND TO FIND OPTIMAL SEQUENCE OF EACH SET OF JOBS
      CALL BANDB
  590 CONTINUE
C
      CALL SECOND(T)
      TOTALT=T-TBEGIN
      WRITE(JO,666)TOTALT
  666 FORMAT(//10X,≠TIME FOR EXECUTION IS ≠,F10.2,≠ SECONDS≠)
      STOP
      END
```

```
      SUBROUTINE BSORT(LIST,TEST,LENGTH)
      EXTERNAL TEST
      LOGICAL TEST
      DIMENSION LIST(LENGTH)
C     THIS SUBROUTINE SORTS THE ARRAY LIST ACCORDING TO ≠TEST≠
C     USING A BUBBLE SORTING TECHNIQUE
      L=LENGTH-1
      DO 500 I=1,L
      DO 400 K=1,I
      J=I-K+1
      IF(.NOT.TEST(LIST(J),LIST(J+1)))GOTO 500
      ITEMP=LIST(J)
      LIST(J)=LIST(J+1)
      LIST(J+1)=ITEMP
  400 CONTINUE
  500 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE RAN1(UR)
C     RANDOM NUMBER GENERATOR
      DATA IR/1111111/
      IR=AND(AND(2069*IR,8388607)+1772721,8388607)
      UR=FLOAT(IR)/8388607
      RETURN
      END
      SUBROUTINE PFILL(NUMSTK)
      COMMON/BLOCK1/ IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK3/LINK(80),LIST(80),ITOP(20),IPTOP(7),TOTSTK(20)
      COMMON/BLOCK4/TOTP(7),TOTMAX(7)
C      THIS ROUTINE FILLS THE PROCESSOR STACKS USING THE
C      FIRST FIT DECREASING METHOD
C
C      FILL THEM UP TO THE PROCESSOR LIMIT
      LEFT=NUMSTK-M
      DO 180 I=1,NUMSTK
      IF(ITOP(LIST(I)).EQ.999)GOTO 180
      J=1
  150 IF(TOTP(J)+TOTSTK(LIST(I)).LE.TOTMAX(J))GOTO 160
      J=J+1
      IF(J.GT.M)GOTO 160
      GOTO 150
C
C      ADD THE STACK TO THE BOTTOM OF THE PROCESSOR STACK
  160 L=IPTOP(J)
  165 IF(LINK(L).EQ.0)GOTO 170
      L=LINK(L)
      GOTO 165
  170 LINK(L)=ITOP(LIST(I))
      ITOP(LIST(I))=999
      TOTP(J)=TOTP(J)+TOTSTK(LIST(I))
      LEFT=LEFT-1
  180 CONTINUE
C      IF ANY ARE LEFT, PUT THEM IN THE PROCESSOR STACK THAT HAS THE MOST
C      ROOM.
  205 IF(LEFT.EQ.0)RETURN
      J=1
      DO 210 I=1,NUMSTK
      IF(ITOP(I).EQ.999)GOTO 210
      PRINT *,I
      LIST(J)=I
      J=J+1
  210 CONTINUE
C
C      PICK A PROCESSOR STACK
      IPUT=MINS(LEFT)
      ISPOT=MINP(DUMMY)
      IWHER=IPTOP(ISPOT)
C
C      FIND THE LAST ELEMENT IN THE STACK
  230 IF(LINK(IWHER).EQ.0)GOTO 240
      IWHER=LINK(IWHER)
      GOTO 230
C
C      ADD THE STACK TO THE PROCESSOR STACK
  240 LINK(IWHER)=ITOP(IPUT)
      ITOP(IPUT)=999
      LEFT=LEFT-1
      TOTP(ISPOT)=TOTP(ISPOT)+TOTSTK(IPUT)
      GOTO 205
      END


      SUBROUTINE FILL(IFT,NUMSTK)
C     THIS ROUTINE FILLS EACH OF THE STACKS USING THE FIRST FIT DECFEASING METHOD.
C
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK3/LINK(80),LIST(80),ITOP(20),IPTOP(7),TOTSTK(20)
      COMMON/BLOCK4/TOTP(7),TOTMAX(7)
      NXJ=1
      MAXJ=1
      DO 180 I=1,N
      IF(I.EQ.1)GOTO 140
      IF(SJOBS(LIST(I),2).NE.SJOBS(LIST(I-1),2))NXJ=MAXJ
  140 J=NXJ
C
C      FIND THE STACK TO PUT THE JOB IN
  160 IF(.NOT.(TOTSTK(J)+SJOBS(LIST(I),3).GT.IFT).OR.(TOTSTK(J).EQ.0.))
     1     GOTO 150
      J=J+1
      GOTO 160
  150 IF(J+1.GT.MAXJ)MAXJ=J+1
C
C      INSERT THE JOB AT THE TOP OF THE STACK
      LINK(LIST(I))=ITOP(J)
      ITOP(J)=LIST(I)
      TOTSTK(J)=TOTSTK(J)+SJOBS(LIST(I),3)
  180 CONTINUE
      NUMSTK=MAXJ-1
      RETURN
      END
      SUBROUTINE READIN
      COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
      COMMON/BLOCK2/IR,INUM,IRES(20),ICM(80,80)
      DIMENSION IBLK(32)
C
C**   READ THE NUMBER OF PROCESSORS M AND NO. OF JOBS N
C*
      READ(IN,1000)M,N
 1000 FORMAT(2I5)
C
      WRITE(JO,1)M,N
    1 FORMAT(≠1≠//35X,≠THE SCHEDULING OUTPUT≠//10X,≠NO. OF PROCESSORS
     .ARE ≠,I5//10X,≠NO. OF JOBS ARE ≠,I5/)
      WRITE(JO,2)
    2 FORMAT(//10X,≠PROCESSOR≠,6X,≠MACHINES≠/)
C
C**   READ THE PROCESSOR IDENTIFICATION AND THE NO. OF MACHINE ON EACH P  C   PROCESSORS
C
      DO 10 I=1,M
      READ(IN,1000)IP,IM
      IPROC(IP)=IM
      WRITE(JO,3)IP,IM
    3 FORMAT(12X,I5,8X,I5/)
   10 CONTINUE
C     READ JOB DESCRIPTIONS
      WRITE(JO,13)
   13 FORMAT(//10X,≠JOB DESCRIPTION INPUT≠/)
      DO 20 I=1,N
       READ(IN,1300)(SJOBS(I,J),J=1,5)
       WRITE(JO,11)I,(SJOBS(I,J),J=1,5)
   11 FORMAT(//10X,≠(≠,I2,≠)≠,3X,≠(≠,2(F3.0,≠,≠),F4.1,≠,≠,F2.0,≠,≠,F2.0
     .,≠)≠)
   20 CONTINUE
 1300 FORMAT(2F5.0,F5.1,2F5.0)
C
C**   READ NUMBER OF RESOURGE TYPE,IR
C
      READ(IN,1000)IR
      WRITE(JO,16)IR
   16 FORMAT(////10X,≠NO. OF RESOURCE TYPE =≠,I5//10X,≠TYPE≠,2X,≠QUANT≠
     ./)
C**   READ NUMBER OF RESOURCE TYPE,IR.
C
      DO 30 I=1,IR
      READ(IN,1000)IRT,INUM
      WRITE(JO,17)IRT,INUM
   17 FORMAT(10X,I5,2X,I5)
      IRES(IRT)=INUM
   30 CONTINUE
C
      IF(N .GT. 32) GOTO 99
       DO 33 I=1,N
       IBLK(I)=I
   33 CONTINUE
      WRITE(JO,4)
    4 FORMAT(//34X,≠COST MATRIX≠,/1H,34X,≠-----------≠///)
      WRITE(JO,9)(IBLK(J),J=1,N)
    9 FORMAT(5X,32(≠(≠,I2,≠)≠)//)
C
C**   READ THE COST MATRIX WHICH N IS LESS THEN 32
C
       DO 15 I=1,N
       READ(IN,1200)(ICM(I,J),J=1,N)
       WRITE(JO,12)I,(ICM(I,J),J=1,N)
   12  FORMAT(1H0,≠(≠,I2,≠)≠,1X,32(I3,1X))
   15  CONTINUE
 1200  FORMAT(32I3)
       GOTO 111
   99  ICM(I,J)=0
       DO 41 I=1,N
        DO 42 J=1,N
       CALL RAN1(UR)
       ICM(I,J)=INT(UR*100)
         IF (I .EQ. J) ICM(I,J)=999
   42  CONTINUE
   41  CONTINUE
  111  RETURN
       END
```

```
          LOGICAL FUNCTION STKD(I,J)
          COMMON/BLOCK3/LINK(80),LIST(80),ITOP(20),IPTOP(7),TOTSTK(20)
C          THIS ROUTINE IS THE CRITERION FOR SORTING THE STACKS
C          INTO DESCENDING ORDER BY THEIR TOTALS
          STKD=.FALSE.
          IF(TOTSTK(I).LT.TOTSTK(J))STKD=.TRUE.
          RETURN
          END

          LOGICAL FUNCTION TIMF(J,K)
          COMMON/BLOCK1/IN,JO,M,N,IPROC(7),SJOBS(80,5)
          TIMF=.FALSE.
C          THIS FUNCTION IS THE CRITERIA FOR SORTING THE JOB NUMBERS
C          INTO INCREASING ORDER OF RESOURCE TYPE AND DECREASING ORDER
C          OF PROCESSOR TIME
          IF(SJOBS(J,2).EQ.SJOBS(K,2))GOTO 110
          IF(SJOBS(J,2).GT.SJOBS(K,2))TIMF=.TRUE.
          RETURN
    110   IF(SJOBS(J,3).LT.SJOBS(K,3))TIMF=.TRUE.
          RETURN
          END

          INTEGER FUNCTION MINP(DUMMY)
          COMMON/BLOCK1/ IN,JO,M,N,IPROC(7),SJOBS(80,5)
          COMMON/BLOCK4/TOTP(7),TOTMAX(7)
C
C          THIS FUNCTION RETURNS THE SUBSCRIPT OF THE PROCESSOR THAT HAS
C          THE MOST ROOM LEFT
          MINP=1
          DO 100 I=1,M
          IF(TOTMAX(I)-TOTP(I).GT.TOTMAX(MINP)-TOTP(MINP))MINP=I
    100   CONTINUE
          RETURN
          END

          INTEGER FUNCTION MINS(LEFT)
          COMMON/BLOCK3/LINK(80),LIST(80),ITOP(20),IPTOP(7),TOTSTK(20)
C
C          THIS FUNCTION RETURNS THE SUBSCRIPT OF THE
C          STACK IN LIST WITH THE MOST PROCESSOR TIME
          MINS=LIST(1)
          DO 100 I=1,LEFT
          IF(TOTSTK(LIST(I)).GT.TOTSTK(MINS))MINS=LIST(I)
    100   CONTINUE
          RETURN
          END
```

```
      SUBROUTINE BANDB
      COMMON/MATCOM/ ICMAT(20,20),N
      COMMON/LOCAL/ITREE(500,7),ICPEN(20,20),IBESTT(20,2),
     1 ITEMP(20,20),IPEN(20,3),ICOMIT(20,2),NEWCM(20,20),ICOM,
     2 ITOUR(20,2),NONCOM(20,2),IBESTC,IPREV,LEFT,INEXT,INODE,IRGHT
C     IBEST-BEST ROUTE SO FAR,  IBESTC-BEST COST ON THIS
C     ICOMIT - COMMITTED ROUTE
      INF=999
      WRITE(6,530)N
  530 FORMAT(*1*,5X,*N = *,I3)
      DO 5 I=1,N
      WRITE(6,540) (ICMAT(I,J),J=1,N)
    5 CONTINUE
  540 FORMAT(5X,20I5)
      CALL INITIAL(INF)
C      SAVE COST MATRIX
      DO 28 I=1,N
       DO 28 J=1,N
       ITEMP(I,J)=ICMAT(I,J)
   28 CONTINUE
      ICOM=0
C     REDUCE MATRIX ICMAT BY FINDING THE SMALLEST NUMBER IN EACH COLUMN
C     AND SUBTRACT EACH COLUMN WITH THE SMALLEST NUMBER
      CALL MREDUCE(N,ICMAT,IRCONST,INF)
      IPREV=IRCONST
C     LABEL THE IST NODE OF TREE AND INITIALIZE PTRS.
        ITREE(1,1)=IRCONST
       WRITE(6,510) ITREE(1,1)
       INODE=1
       INEXT=2
C      COMPUTE THE COST PENALTY
C
       IPREV=0
    6 CONTINUE
      CALL PENALTY(MAX,K,L,INF)
      IP=0
C     BRANCHING
      LEFT=INEXT
      ITREE(INODE,2)=LEFT
      ITREE(LEFT,1)=IRCONST+MAX+IPREV
      IF(ITREE(LEFT,1) .GT. INF ) ITREE(LEFT,1)=INF
      ITREE(LEFT,4)=K
      ITREE(LEFT,5)=L
      ITREE(LEFT,6)=INODE
      ITREE(LEFT,7)=1
       WRITE(6,600) ITREE(LEFT,1)
  600 FORMAT(*  LEFT BRANCH :  *,I5)
      INEXT=INEXT+1
      IRGHT=INEXT
      ITREE(INODE,3)=IRGHT
      IPREV=ITREE(INODE,1)
      ITREE(IRGHT,4)=K
      ITREE(IRGHT,5)=L
      ITREE(IRGHT,6)=INODE
      ITREE(IRGHT,7)=0
      INEXT=INEXT+1
C      DELETE ROW K AND COLUMN L FROM ICMAT
      DO 70 I=1,N
      ICMAT(I,L)=INF
      ICMAT(K,I)=INF
   70 CONTINUE
C      FIND P=BEGINING OF JOB P AND M ENDING JOB (K,L)
C        AMONG THE ROUTE GENERATED BY THE COMMITTED JOB PAIR
      CALL GENROUT(M,MP,K,L,INF)
      ICOM=ICOM+1
      ICOMIT(ICOM,1)=K
      ICOMIT(ICOM,2)=L
C      SET ICMAT(M,P)=INF
      ICMAT(M,MP)=INF
C     REDUCE ICMAT
      CALL MREDUCE(N,ICMAT,IRCONST,INF)
  120 ITREE(IRGHT,1)=IRCONST+IPREV
      INODE=IRGHT
      WRITE(6,610) ITREE(IRGHT,1),K,L
  610 FORMAT(*  RIGHT BRANCH : *,10I5)
      IPREV=IPREV+IRCONST
C     CHECK IF ICMAT IS A 2X2 MATRIX
```

```
                                                FIN M.0+518        80/12/11.
        IT=0
      00 125 I=1,N
       00 125 J=2,N
       IF (ICMAT(I,J) .NE. INF) IT=IT+1
  125 CONTINUE
      IF (IT .GT. 2) GO TO 135
      IF (IPREV .LT. IBESTC) GO TO 126
      GO TO 135
  126 CONTINUE
C     SAVE THE COST ANC ROUTE
      CALL SAVE(ITOT,INF)
  135 CONTINUE
C     EXAMINE THE LOWER BOUNDS OF THE TERMINAL NOOES OBTAINEO
C     SO FAR ANO CHOOSE THE ONE WITH THE SMALLEST VALUE TO BRANCH
C
      IN=INEXT-1
      MINLB=INF
      00 140 I=2,IN
      IF((ITREE(I,2).EQ.0).ANO.(ITREE(I,3).EQ.0))GO TO 137
      GO TO 140
  137 CONTINUE
      IF (ITREE(I,1).GE.MINLB) GO TO 140
      MINLB=ITREE(I,1)
      MINOOE=I
  140 CONTINUE
      INOOE=MINOOE
      IPREV=MINLB
      WRITE(6,650) IBESTC,MINLB,MINOOE,IRGHT
  650 FORMAT(≠  IBESTC= ≠,I10,≠ MINLB = ≠,I5,≠ MINOOE=≠,I5,≠ IRGHT=≠,I5)
      IF (IBESTC .LE. MINLB) GO TO 250
      IF (MINOOE.EQ.IRGHT) GO TO 6
      CALL NXTBNO(MINOOE,IG,NCOM,INF)
C      FOR EACH NEWCM(K,L) PROHIBITED FROM ROUTE IN CURRENT ICOMIT,
C      SET NEWCM(K,L) TO INF
  195 CONTINUE
      IF(NCOM.EQ.0) GO TO 210
       00 200 I=1,NCOM
       K=NONCOM(I,1)
       L=NONCOM(I,2)
       NEWCM(K,L)=INF
  200  CONTINUE
  210  CONTINUE
C      REOUCE NEWCM
      CALL MREDUCE(N,NEWCM,IRCONST,INF)
       IRCONST=IRCONST+IG
       IPREV=0
         ITREE(MINOOE,1)=IRCONST
         00 245 I=1,N
         00 245 J=1,N
         ICMAT(I,J)=NEWCM(I,J)
  245  CONTINUE
       GO TO 6
C    PRINT OUTPUT
  250  CONTINUE
       WRITE(6,500)
  500  FORMAT(≠0≠,5X,≠OPTIMAL SCHEOULE //≠)
       DO 255 I=1,ITOT
        WRITE(6,510) IBESTT(I,1),IBESTT(I,2)
  255  CONTINUE
  510  FORMAT(5X,2I5)
       WRITE(6,520) IBESTC
  520  FORMAT(// 5X,≠OPTIMAL COST OF SCHEOULE  =  ≠,I5)
       STOP
       END
      SUBROUTINE INITIAL(INF)
      COMMON/MATCOM/ ICMAT(20,20),N
      COMMON/LOCAL/ITREE(500,7),ICPEN(20,20),IBESTT(20,2),
     1 ITEMP(20,20),IPEN(20,3),ICOMIT(20,2),NEWCM(20,20),ICOM,
     2 ITOUR(20,2),NONCOM(20,2),IBESTC,IPREV,LEFT,INEXT,INODE,IRGHT
C      ROUTINE INITIALIZES ARRAYS
      DO 10 I=1,20
      DO 10 J=1,20
       ICPEN(I,J)=0
   10 CONTINUE
      DO 15 I=1,500
      DO 15 J=1,7
      ITREE(I,J)=0
   15 CONTINUE
      ITREE(1,4)=1
      ITREE(1,5)=1
      ITREE(1,7)=1
      IBESTC=INF
C     SET C(I,1) TO INF
      DO 20 I=1,N
      ICMAT(I,1)=INF
   20 CONTINUE
C     SET DIAGONAL ELEMENTS TO INFINITY
      DO 25 I=1,N
       DO 25 J=1,N
        IF(I .EQ. J) ICMAT(I,J) = INF
   25 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MREDUCE(N,MAT,IRCONST,INF)
      DIMENSION MAT(20,20)
C      ROUTINE DOES THE MATRIX REDUCTION
C      RETURNS REDUCTION CONSTANT, IRCONST
       IRCONST=0
       DO 225 J=2,N
       MIN=INF
       DO 220 I=1,N
       IF(MAT(I,J).LT.MIN) MIN=MAT(I,J)
  220  CONTINUE
       IF (MIN.EQ.INF) GO TO 225
       IRCONST=IRCONST+MIN
       DO 222 I=1,N
       IF(MAT(I,J).EQ.INF) GO TO 222
       MAT(I,J)=MAT(I,J)-MIN
  222  CONTINUE
  225  CONTINUE
C      REDUCE CURRENT FIRST ROW
       DO 230 I=1,N
       ISW=0
       DO 228 J=2,N
       IF (MAT(I,J).NE.INF) ISW=1
  228  CONTINUE
       IF (ISW.EQ.1) GO TO 235
  230  CONTINUE
       GO TO 242
  235  MIN=INF
      IF(I .NE. 1) GO TO 242
        DO 238 J=2,N
        IF (MAT(I,J).LT.MIN) MIN=MAT(I,J)
  238  CONTINUE
       IF (MIN.EQ. INF) GO TO 242
       IRCONST=IRCONST+MIN
       DO 240 J=2,N
        IF (MAT(I,J).EQ.INF) GO TO 240
        MAT(I,J)=MAT(I,J)-MIN
  240  CONTINUE
  242  CONTINUE
      RETURN
      END
      SUBROUTINE PENALTY(MAX,K,L,INF)
      COMMON/MATCOM/ ICMAT(20,20),N
      COMMON/LOCAL/ITREE(500,7),ICPEN(20,20),IRESTT(20,2),
     1 ITEMP(20,20),IPEN(20,3),ICOMIT(20,2),NEWCM(20,20),ICOM,
     2 ITOUR(20,2),NONCOM(20,2),IBESTC,IPREV,LEFT,INEXT,INODE,IRGHT
C      ROUTINE COMPUTES THE COST PENALTY
       IP=0
      DO 60 I=1,N
      DO 60 J=2,N
       IF (ICMAT(I,J).NE. 0) GO TO 60
       IRMIN=INF
       ICOLMIN=INF
       DO 52 K=2,N
        IF (K .EQ. J) GO TO 52
        IF (ICMAT(I,K) .LT. IRMIN) IRMIN=ICMAT(I,K)
   52  CONTINUE
      DO 54 K=1,N
      IF (K .EQ. I) GO TO 54
      IF (ICMAT(K,J) .LT. ICOLMIN) ICOLMIN=ICMAT(K,J)
   54 CONTINUE
      ICPEN(I,J)=IRMIN+ICOLMIN
      IP=IP+1
      IPEN(IP,1)=ICPEN(I,J)
      IPEN(IP,2)=I
      IPEN(IP,3)=J
   60 CONTINUE
C     FIND MAXIMUM COST OF PENALTY
      MAX=IPEN(1,1)
      K=IPEN(1,2)
      L=IPEN(1,3)
      DO 65 I=1,IP
      IF (IPEN(I,1).LE.MAX) GO TO 65
      MAX = IPEN(I,1)
      K=IPEN(I,2)
      L=IPEN(I,3)
   65 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE GENROUT(M,MP,K,L,INF)
      COMMON/MATCOM/ ICMAT(20,20),N
      COMMON/LOCAL/ITREE(500,7),ICPEN(20,20),IRESTT(20,2),
     1 ITEMP(20,20),IPEN(20,3),ICOMIT(20,2),NEWCM(20,20),ICOM,
     2 ITOUR(20,2),NONCOM(20,2),IBESTC,IPREV,LEFT,INEXT,INODE,IRGHT
C      ROUTINE FINDS P = BEGINNING OF JOB P AND M ENDING
C      JOB(K,L) AMONG THE ROUTE GENERATED BY THE JOB PAIR
C      K,L
      IF (ICOM .GT. 1) GO TO 75
      MP=K
      M=L
      GO TO 95
   75 CONTINUE
C     FIND P
      MP=K
   78 CONTINUE
      DO 80 I=1,ICOM
      IF (ICOMIT(I,2) .EQ. MP) GO TO 82
   80 CONTINUE
      GO TO 85
   82 MP=ICOMIT(I,1)
      GO TO 78
C       FIND M
   85 M=L
   88 CONTINUE
       DO 90 I=1,ICOM
       IF(ICOMIT(I,1) .EQ. M) GO TO 92
   90 CONTINUE
      GO TO 95
   92 M=ICOMIT(I,2)
      GO TO 88
   95 CONTINUE
      RETURN
      END
      SUBROUTINE SAVE(ITOT,INF)
      COMMON/MATCOM/ ICMAT(20,20),N
      COMMON/LOCAL/ITREE(500,7),ICPEN(20,20),IBESTT(20,2),
     1 ITEMP(20,20),IPEN(20,3),ICOMIT(20,2),NEWCM(20,20),ICOM,
     2 ITOUR(20,2),NONCOM(20,2),IBESTC,IPREV,LEFT,INEXT,INODE,IRGHT
C      ROUTINE SAVES THE BEST SO FAR AND THE ROUTE
C      GIVING THAT COST
      IBESTC=IPREV
      WRITE(6,620) IBESTC
  620 FORMAT(*  BEST COST SO FAR I*,I10)
      N1=ICOM+1
      DO 128 I=1,N
      DO 128 J=2,N
      IF (ICMAT(I,J) .EQ. INF) GO TO 128
      ICOM=ICOM +1
      ICOMIT(ICOM,1)=I
      ICCMIT(ICOM,2)=J
      WRITE(6,610) I,J
  128 CONTINUE
C     SAVE THE ROUTE
      DO 129 I=N1,ICOM
      LEFT = INEXT
      ITREE(LEFT,1)=INF
      ITREE(LEFT,4)=ICCMIT(I,1)
      ITREE(LEFT,5)=ICCMIT(I,2)
      ITREE(LEFT,5)=INODE
      ITREE(LEFT,7)=1
      INEXT=INEXT+1
      IRGHT=INEXT
      ITREE(IRGHT,1)=IBESTC
      ITREE(IRGHT,4)=ICOMIT(I,1)
      ITREE(IRGHT,5)=ICOMIT(I,2)
      ITREE(IRGHT,6)=INODE
      ITREE(IRGHT,7) =0
      INODE=IRGHT
      INEXT=INEXT+1
  129 CONTINUE
C     READ OFF ROUTE AND SAVE
      ITOT=1
      ITOUR(ITOT,1)=ITREE(IRGHT,4)
      ITOUR(ITOT,2)=ITREE(IRGHT,5)
      N1=ITREE(IRGHT,6)
  130 CONTINUE
      IF (N1 .EQ. 0) GO TO 133
      IF(ITREE(N1,7) .EQ. 1) GO TO 132
      ITOT=ITOT+1
      ITOUR(ITOT,1)=ITREE(N1,4)
      ITOUR(ITOT,2)=ITREE(N1,5)
  132 N1=ITREE(N1,6)
      GO TO 130
  133 CONTINUE
      DO 134 I=1,ITOT
      N1=ITOT-I+1
      IBESTT(I,1)=ITOUR(N1,1)
      IBESTT(I,2)=ITOUR(N1,2)
  134 CONTINUE
  610 FORMAT(*  RIGHT BRANCH 1 *,10I5)
      RETURN
      END
      SUBROUTINE NXTBNO(MINOOE,IG,NCOM,INF)
      COMMON/MATCOM/ ICMAT(20,20),N
      COMMON/LOCAL/ITREE(500,7),ICPEN(20,20),IBESTT(20,2),
     1 ITEMP(20,20),IPEN(20,3),ICOMIT(20,2),NEWCM(20,20),ICOM,
     2 ITOUR(20,2),NONCOM(20,2),IBESTC,IPREV,LEFT,INEXT,INODE,IRGHT
C     STEP 11
C     SET UP ORIGINAL COST MATRIX
      OO 145 I=1,N
      OO 145 J=1,N
       NEWCM(I,J)=ITEMP(I,J)
  145 CONTINUE
C     READ PAIRS (I,J) COMMITTED TO BE IN THE ROUTE OF MINODE
      ICOM=0
      NCOM=0
      IF(ITREE(MINOOE,7).EQ.1) GO TO 146
      ICOM=ICOM+1
      ITOUR(ICOM,1)=ITREE(MINOOE,4)
      ITOUR(ICOM,2)=ITREE(MINOOE,5)
      GO TO 148
  146 NCOM=NCOM+1
      NONCOM(NCOM,1)=ITREE(MINOOE,4)
      NONCOM(NCOM,2)=ITREE(MINOOE,5)
  148 IN=ITREE(MINOOE,6)
  150 CONTINUE
      IF (IN.EQ.0) GO TO 153
      IF (ITREE(IN,7).EQ.1) GO TO 151
      ICOM=ICOM+1
      ITOUR(ICOM,1)=ITREE(IN,4)
      ITOUR(ICOM,2)=ITREE(IN,5)
      GO TO 152
  151 NCOM=NCOM+1
      NONCOM(NCOM,1)=ITREE(IN,4)
      NONCOM(NCOM,2)=ITREE(IN,5)
  152 IN=ITREE(IN,6)
      GO TO 150
  153 CONTINUE
      IG=0
      IF(ICOM.EQ. 0) GO TO 195
      OO 154 I=1,ICOM
      IN=ICOM-I+1
      ICOMIT(I,1)=ITOUR(IN,1)
      ICOMIT(I,2)=ITOUR(IN,2)
  154 CONTINUE
      OO 190 IM=1,ICOM
      I=ICOMIT(IM,1)
      J=ICOMIT(IM,2)
      IG=IG+NEWCM(I,J)
C     DELETE ROW I ANO COLUMN J
      OO 158 I1=1,N
      NEWCM(I,I1)=INF
      NEWCM(I1,J)=INF
  158 CONTINUE
C     FOR EACH ROUTE AMONG THE (I,J) FINO THE STARTING JOB P AND ENOING
C     JOB M ANO SET NEWCM(M,P) TO INFINITY
C
      CALL GENROUT(M,MP,I,J,ICOM,INF)
C      SET NEWCM(M,P)=INF
      NEWCM(M,MP)=INF
  190 CONTINUE
  195 CONTINUE
      RETURN
      ENO
```