BICRITERIA OPTIMIZATION OF SCHEDULES ON ONE AND TWO MACHINES

by

Rema Hariharan

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering and Operations Research

APPROVED:

Subhash.C.Sarin, Chairman

Hanif.D.Sherali

Jerry.R.Rakes

May 19, 1988

Blacksburg, Virginia

Table of Contents

List of Illustrations

List of Tables

Acknowledgements

I sincerely extend my gratitude to all those who were directly and indirectly connected with the development of this work.

I would first like to thank my advisor Dr.Subhash.C.Sarin for his patience, cooperation and help at every stage of the work. But for his encouragement, guidance and willingness to help it wouldn't have been possible to complete this thesis.

My thanks are due to Dr.Hanif.D.Sherali and Dr.Terry R.Rakes for serving on my committee and for making helpful suggestions.

I must thank my roommates and friends in the department for their company and for helping me with various things.

I thank my parents whose encouragement and moral support has stood by me all through out.

And of course, thanks to GOD, the superpower, for everything!

Chapter 1

Introduction

The practical applications of scheduling generally involve the optimization of more than one criterion. However, most of the research related to scheduling has considered the optimization of only a single performance measure. Only for certain special cases have dual criteria job scheduling algorithms been developed.

The early work in this direction due to Smith[14] dealt with the criteria of minimizing flowtime subject to the minimum value of the maximum job tardiness, Tmax. In this context, we refer to Tmax as the primary criterion and the total flow time as the secondary criterion for optimization. In another work, Emmons[7] developed an algorithm which produces a schedule with minimum total job flow time subject to the number of tardy jobs determined by Moore's algorithm[10].

Recent studies that have attacked the bicriteria scheduling problem, have used trade off approach (i.e. optimization of the secondary criterion subject to some fixed value for the primary criterion), leading to efficient schedule generation. This includes the

study by Cottingham[5] for the primary and secondary criteria as the number of tardy jobs and flow time respectively and the research by Nelson, Sarin and Daniels [11], which presents a number of generalized dual criteria single machine job scheduling algorithms.

Other research attempts have been in the direction to build a goal function giving suitable weights to the two criteria to be optimized and then employ goal programming approaches to reach the solution. It was shown by Sen and Gupta[15] that the schedules obtained corresponding to different weights correspond to the various efficient schedules.

1.1 Problem Definition

In this thesis, we address the bicriteria problem of minimizing the number of tardy jobs subject to the Tmax value. Algorithms are developed for both single machine and two machine problems.

For the single machine problem, dominance rules and branch termination rules are developed that speed up the search tree procedure developed by Nelson, Sarin and Daniels. For the two machine problem, an algorithm is developed to minimize the secondary criterion of the number of tardy jobs.

The practical application of this problem would include the minimization of the number of dissatisfied customers while maintaining the maximum level of dissatisfaction

at a minimal value. Since the procedure suggested can be used to generate efficient solutions, it would form a convenient method for managerial decision making.

1.2 Literature Survey

Recent work by Bulfin[2] classifies various kinds of single machine bicriteria problems according to the computational complexity involved. An assignment model approach has been suggested for the special case of equal processing times of jobs.

Most of the work related to the single machine bicriteria scheduling that has been reported in the literature is on the criteria of the minimization of the weighted completion times subject to the minimal value of Tmax. This problem was first considered by Smith, who conjectured an algorithm of complexity O(n logn). Smith's conjecture was that an optimal schedule to this problem can be obtained by scheduling job k last such that

$$\frac{p_k}{w_k} = \max_{j \in L}\{\frac{p_j}{w_j}\}$$

where p_j represents the processing time of job j, w_j represents the weight corresponding to job j, L is the set of jobs whose tardiness will not be more than Tmax when scheduled at the last position and Tmax is the minimum possible value of the maximum tardiness. However, counter-examples to this conjecture were presented by Emmons [6] and Burns[1]. Two heuristic algorithms of $O(n^3)$ complexity that improve upon Smith's algorithm are proposed by Burns[1] and Miyazaki[9]. Bansal[3],

presents a branch and bound approach using dominance rules to solve this problem. Another approach to the same problem is presented by Potts and VanWassenhove[19]. This approach employs adjusted deadlines and lower bounds obtained by performing a Lagrangean relaxation of the deadline constraints. Posner's work [12] on the minimization of the weighted flow times with deadlines develops precedence conditions and lower bounds obtained by splitting the job into two parts and scheduling them based on the weights associated with them. The lower bounds obtained are shown to be superior to those obtained by Potts and Van Wassenhove[20]. Chand and Schneeberger[4] identify several cases of the same problem for which Smith's heuristic [14] would generate an optimal solution.

Another set of conditions that has been dealt with frequently in the literature is the minimization of job flow time subject to a fixed value of the number of tardy jobs. Cottingham[5] presents a comparison of the branch and bound method based on Emmon's algorithm[7] to obtain the trade off points with two heuristic methods.

Shantikumar[16] has developed a branch and bound method to minimize the maximum tardiness subject to the minimum number of tardy jobs. This approach is based on the division of jobs into an early set and a late set. However, due to the computational aspects of the algorithm it can only be applied to problems with smaller number of jobs. Also, the algorithm cannot be extended to the generation of efficient solutions. As opposed to this, Nelson,Sarin and Daniels[11] have developed a generalized algorithm to obtain the efficient solutions. Other research on bicriteria scheduling focuses on the minimization of the deviation of completion times about their deadlines, using penalty functions for both earliness and tardiness. This problem has

special applications in the JIT (Just-In-Time) environment where it is necessary to minimize the inventory as well as the lateness of the jobs.

However, it is apparent that hardly any research has been reported on the bicriteria scheduling as applied in a multi-machine environment.

1.3 Thesis Outline

The presentation in Chapter 2 deals with the single machine problem. First of all, the derivation of the dominance rules is presented and following this is a presentation of the corresponding search tree algorithm. The later portion of the work presented in this thesis deals with the optimization of the same set of criteria for the two machine problem. In Chapter 3, detailed analysis has been carried out for the minimization of the primary criterion Tmax and an algorithm is presented to get the minimal value of Tmax. Following this, a branch and bound based method is developed to optimize the secondary criterion (number of tardy jobs). The approach suggested proceeds parallel to the single machine algorithm to minimize the number of tardy jobs subject to a given level of maximum tardiness. However, none of the dominance rules that are applicable to the single machine problem could be extended to the two machine case.

A detailed analysis was carried out to test the peformance of the algorithm to minimize Tmax as well as for the algorithm to minimize NT. Results regarding the CPU time required and the accuracy of the final solution obtained have been presented in

Chapter 4. Since the storage requirements for the proposed algorithm would restrict its application for larger number of jobs, an experiment was carried out to test the effectiveness of the upper bound value obtained by applying the single machine scheduling algorithm to a given assignment of jobs on the two machines. The results are presented in Chapter 4.

1.3.1 Glossary of Notations

- p_j - processing time of job j
- d_j - due date of job j
- Tmax - Maximum tardiness value.
- NT- Number of tardy jobs

Chapter 2

Algorithm for the single machine case

This chapter deals with the solution of a bicriteria single machine scheduling problem with the primary objective of minimizing the maximum tardiness and the secondary objective of minimizing the total number of tardy jobs.

Recently, Nelson, Sarin and Daniels [11] have developed a search tree based algorithm, which makes use of a few dominance conditions to reduce the number of branches. In this research, the same algorithm is improved by the introduction of two new dominance conditions and a branch terminating rule.

2.1 Definitions and Terminologies

In the algorithm presented, the jobs are scheduled beginning from the last position. Only those jobs are considered for scheduling at a position that are eligible. The eligibility of a job to be scheduled is defined as follows :

Eligibility of a job: A job *i* is considered to be eligible to be scheduled to end at time *T* if $T \leq d_i + T_{max}$ where d_i is the due date of job i and T_{max} is the maximum tardiness value as obtained in the EDD sequence.

The processing time for job i, is designated by p_i .

2.1.1 Dominance Rules

Theorem 1

If $p_i < p_j$, $d_i < d_j$ and $p_i - d_i < p_j - d_j$, then there exists an optimal schedule which job i is scheduled at a position before job j.

Proof

See [11]

Theorem 2

If for the eligible jobs i and j such that both i and j are not early, $p_i \leq p_j$ and $d_i > d_j$, then there exists an optimal schedule where job i is scheduled at a position before job j.

Proof

Since jobs i and j are eligible to end at time T,

$T \leq T_{max} + d_i$

and

$T \leq T_{max} + d_j$.

Since, $p_i \leq p_j$

$T - p_i \geq T - p_j$

The proof is by contradiction. Let job 'j' not be the last job in the optimal sequence and instead let job 'i' be the last job in the optimal sequence. We have two cases.

a. If $T - p_i > d_j$, we have $NT \geq 2$ because job i is tardy at the last position (by assumption) and job j would be tardy at the last but one position. But this schedule could be improved by putting job j as the last job instead because $p_j > p_i$ and $d_i > d_j$.

b. If $T - p_i \leq d_j$, job j would be an early job at the last but one position, $\Rightarrow$ j can also be the last job without making NT = 2 because in this case job i would be an early job at the last but one position.

Therefore, there exists an optimal schedule in which job j comes after job i.

Hence the proof.

Theorem 3

Let p_{max} denote the maximum processing time amongst the jobs eligible to end at time T. Let d_k denote the maximum of the due dates amongst the jobs eligible to end at T. If $T - p_{max} > d_k$, then there exists an optimal schedule, in which we have the job with the maximum processing time is scheduled to end at time T. However, if $T - p_{max} \leq d_k$, then in the optimal schedule the p_{max} job need not be scheduled to end at T.

Proof

If $T - p_{max} > d_k$, then counting the number of tardy jobs before time T, note that in the last and the last but one position any eligible job scheduled would be tardy. Hence by scheduling the largest job to end at T, we would make the finish time of the other jobs early by a maximum possible value, thereby potentially making them early in the schedule.

However, if $T - p_{max} \leq d_k$ then, we know that $NT \geq 1$ (counting the number of tardy jobs before T). Now, if p_{max} corresponds to p_k then if the job k is scheduled to end at time

T, then it is possible that $NT \geq 2$ if $T - p_{max} > d_{k'}$ where $d_{k'}$ is the next highest due date. However, there is a possibility that by scheduling a job j of a smaller processing time such that $T - p_j \leq d_k$, job k becomes early at the previous position to that of job j thereby reducing the number of tardy jobs. This completes the proof.

2.1.2 Branch Termination Rule

Theorem 4

If the total number of tardy jobs in any branch at any level is greater than the number of tardy jobs in the EDD sequence, then that branch can be fathomed.

Proof

The number of tardy jobs in the EDD sequence lays an upper bound on the minimal value of the number of tardy jobs in a sequence that satisfies the primary criterion and hence the proof.

2.2 *Algorithm to Minimize NT with Tmax as the Primary Criterion*

Notations: The matrix AFT(i,j) denotes the dominance relation between jobs i and j. The matrix is set up in such away that if the dominance rules suggest that job j is followed by job i in an optimal sequence, then the entry AFT(i,j) is set equal to 1. Otherwise, the entry AFT(i,j) is set equal to 0. Tminus is an array to store the time value obtained in different branches after the respective eligible jobs are scheduled.

Step 0(Initialization) :

Arrange all the jobs in EDD. Set T_{max} = Maximum tardiness level in the sequence. Set UNT = Number of tardy jobs in the EDD sequence. Set the dominance matrix AFT(I,J) by applying the rules in Theorem 1 and Theorem 2.Set UB = UNT, NT=0, and $T = \sum_{i=1}^{N} p_i$ where N is the total number of jobs.

Step 1(Apply dominance conditions)

Step 1(a): If for any eligible job/jobs, $T \leq d_i$, then schedule all such jobs (in any sequence) to end at T and set $T = T - \sum_{i \in E} p_i$ where E denotes the set of jobs early at T; update the set of eligible jobs for new T and Tminus and go to step 3. Else, Go to Step 1 (b).

Step 1(b)

Among the set of eligible jobs to end at T, if for any pair i and j, of eligible jobs, AFT(i,j) = 1, then eliminate job i from the list of eligible jobs. Go to Step 1(c).

Step 1(c)

Get the highest due date d_{max} amongst the due dates of the remaining set of eligible jobs and p_{max}, the highest processing time amongst the processing times of the the remaining set of eligible jobs. If $T - p_{max} \geq d_{max}$, then schedule the largest job to end at T. Set $T = T - p_{max}$, $NT = NT + 1$. If new T = 0, and $NT \leq UB$, set UB = NT and store the corresponding schedule, go to step 3. If new$T \geq 0$ and $NT > UB$ do not need to store this complete or partial solution; go to step 3. If new$T > 0$ and $NT \leq UB$ update the set of eligible jobs for new T and add T to the set Tminus, and Go to step 3. Else, Go to Step 2.

Step 2(Branching)

: For the current T value generate a branch for each eligible job i by scheduling job i at the last position. Set $T = T - p_i$,$NT = NT + 1$(for each branch). If T = 0, and $NT \leq UB$, set UB = NT. Store the corresponding solution. If T = 0 and $NT > UB$ do not need to store this solution. If $T > 0$ and$NT \leq UB$ then, add T to the set Tminus(for each branch) and store the corresponding partial solution. If $T > 0$ and $NT > UB$ do not need to store solution. Go to step 3.

Step 3(Node Selection)

: From the set Tminus, select the highest value of T and remove that value of T from the set Tminus. If $T > 0$,go to step 1, and if T = 0, then stop. The current UB gives the required NT value and the corresponding sequence gives the required optimal sequence.

2.3 Computation and Results

To judge the effectiveness of the algorithm presented above, it was run on several problems and its CPU times were compared with those of the procedure of Nelson, Sarin and Daniels (herewith called as 'Old Procedure'). This old procedure is like the proposed algorithm except that it does not use the dominance rules due to Theorems 2 and 3.

For each problem size, 5 problems were generated using two different due date generation procedures. In the first procedure, (due to Emmons), processing times are selected as integers, uniformly distributed between 1 and 10. The due date for a job i is determined by adding to the processing time a number randomly selected from a uniform distribution of integers between 0 and 2n.

$$d_i = p_i + U(0,2n)$$

In the second procedure, the due dates are determined according to the following model.

$$d_i = p_i + C_1(AWT) + N(0, C_2(AWT))$$

where, AWT = Average waiting time = (n-1)/2 * Average processing time and $N(0, C_2(AWT))$ represents normal distribution with zero mean and $C_2(AWT)$ variance. The values of the constants that were used are $C_1 = 0.33$ and $C_2 = 0.33$. Nelson, Sarin and Daniels have used other values of C_1 and C_2 also. However, it was observed that neither C_1 nor C_2 causes a major difference in the computational time. Both sets of problems were run for 50, 100, 150, 200, 250, 300, 400 number of jobs. For each

problem size, (for each type of due date), at least 5 problems were generated. So, 35 problems were run for each set. The average CPU times in seconds for the execution of the algorithm are reported in Tables 1 and 2 for the Emmons type of due dates and Normal distribution of due dates, respectively. In each of these tables, corresponding values are reported for the algorithm due to Nelson, Sarin and Daniels as well. It was found that the range of the execution times obtained for any particular size of the problem was approximately 10% about the average value. A comparison of the CPU times of the two algorithms shows that the new algorithm is several folds faster than the old one thereby indicating the effectiveness of the dominance rules developed. To further display the superiority of the new algorithm, Figures 1 and 2 show the graph of the CPU times .vs. number of jobs for the old and the new algorithm, respectively, for the two types of due dates considered.

Table 1. Average CPU times for Emmon's Problems

# of jobs	Old algorithm (Average CPU time in secs)	New Algorithm (Average CPU time in secs)
50	2	0
100	8	2
150	19	4
200	39	8
250	85	15
300	140	26
400	341	63

Table 2. Average CPU times for Normal distribution of due dates

# of jobs	Old Algorithm (Average CPU time in secs	New Algorithm Average CPU time in secs
50	2	0
100	8	2
150	22	5
200	66	5
250	152	26
300	219	43
400	443	103

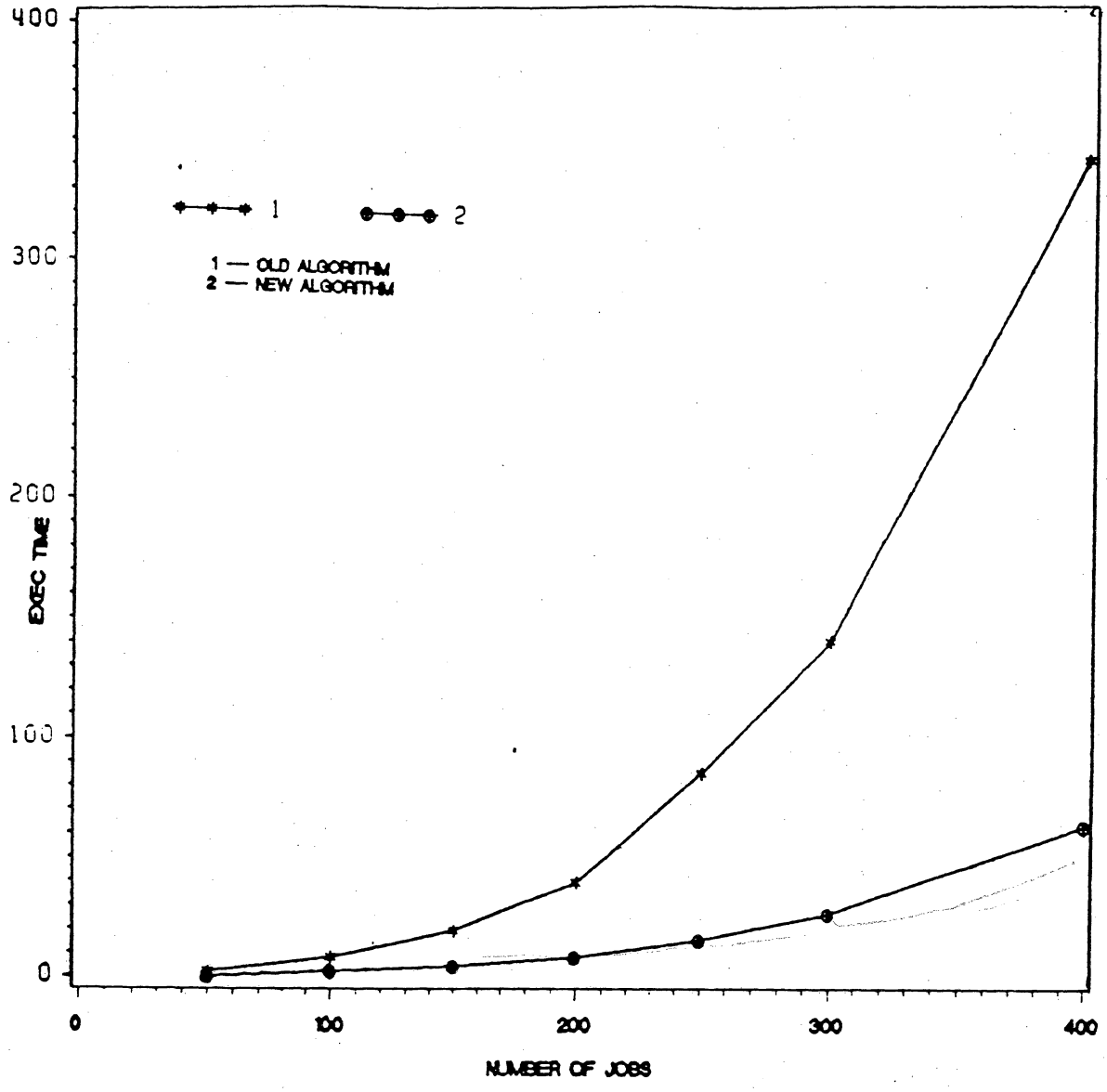


Figure 1. Graph For the Emmons type of due dates.

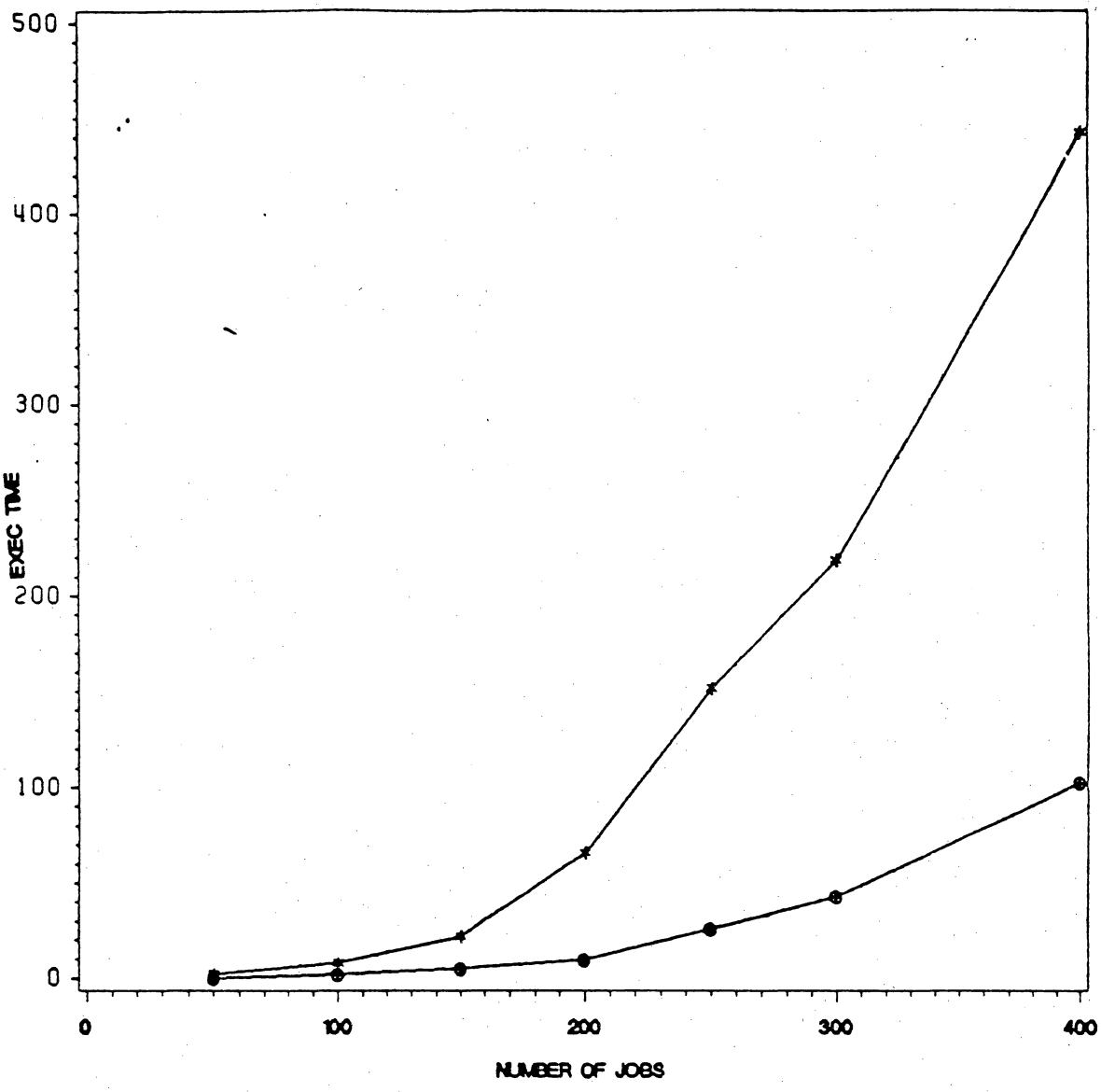


Figure 2. Graph For the Normal distribution of due dates.

Chapter 3

The Two Machine Bicriteria Problem

3.1 *Introduction*

In this chapter, we analyze the bicriteria problem for the two machine case. The primary criterion considered is that of minimizing Tmax i.e. maximum tardiness and the secondary criterion considered is that of minimum NT i.e. the total number of tardy jobs. Detailed analysis has been carried out for the optimization of the primary criterion of Tmax, and then a branch and bound procedure is presented for the secondary criterion of NT.

Unlike in a single machine case, there does not exist a simplistic rule that minimizes the Tmax value in the two machine case. However, given an assignment of jobs to the two machines, there exists an EDD sequence (separately on the two machines)

which minimizes the Tmax values of jobs individually on the two machines. This is a direct application of the EDD rule in a single machine problem.

The algorithm presented in this chapter for the primary criterion first determines the starting upper and lower bound values of Tmax for the two machine problem. Further manipulation of the schedules of jobs on the two machines is carried out to eventually converge to the optimal value of Tmax. These manipulations are based on the exchange of jobs (both single and multiple job exchanges) between the two machines.

The algorithm presented to minimize the secondary criterion NT (knowing the optimal value of the primary criterion, Tmax), follows a branch and bound search tree procedure.

3.1.1 Notations

1. At any stage of the algorithm, the jobs assigned to each machine are arranged in an EDD sequence. This is because, EDD sequence minimizes Tmax for the single machine problem. In order to minimize Tmax on two machines, it is necessary that Tmax is minimized on the two machines individually. Also, this helps us in carrying out the job exchange between the two machines in a systematic manner.

2. In the algorithm presented, we begin by arranging the jobs in an EDD sequence such that if two jobs i and j have equal due dates then the job with a higher processing time precedes the job with a lower processing time. In such a sequence, if job i precedes job j then we denote this by i .b. j. Conversely, the

notation i .b. j also means that $d_i < d_j$, and if $d_i = d_j$ then $p_i \geq p_j$. Also, by the notation i .b. k .b. j, we imply that job i precedes job k and job k precedes job j in that sequence.

3. 'k' refers to the job with the maximum tardiness value on machine 1 and 'l' refers to the job with maximum tardiness on machine 2. Machine 1 and 2 are labelled such that k .b. l.

3.2 *Motivation Regarding the Development of the Algorithms*

3.2.1 Motivation for the Algorithm to minimize Tmax

Given an allocation and arrangement of jobs to the two machines, let Tmax1 and Tmax2 represent the Tmax values obtained respectively on machines 1 and 2. Even though any arrangement of jobs can be considered on the two machines, we shall see later on that arranging the jobs in EDD sequence to start with always gives good starting Tmax values. Also, as discussed above, this helps in carrying out the job exchange between the two machines in a systematic manner. Hence, we will consider EDD sequence as the starting sequence on each machine. Also, due to similar reasons, the jobs will be allocated in EDD order to the two machines; assigning a job with the next earlier due date to the earliest available machine. Now, the Tmax value for the two machine problem equals max(Tmax1,Tmax2), and this maximum value

sets an upperbound on the required optimal value of Tmax. The procedure that is developed first aims at the determination of an assignment of jobs to the two machines such that minimum of the Tmax values of the jobs on the two machines sets a lower bound on the required optimal value of Tmax. This would result in the closed interval [Tmax1,Tmax2] (or the range) in which the optimal value of Tmax lies. This is called the 'range location phase' or Phase I of the algorithm. Starting from a given assignment of jobs to the two machines, this is accomplished by exchanging jobs between the two machines so that no further exchange leads to a value of Tmax less than the current minimum of the Tmax values on the two machines. Both single and multiple job exchanges between the two machines are considered. All such exchanges are made depending upon the locations of the jobs on each machine that correspond to the Tmax values. A set of necessary conditions developed in section 3.3 are used to determine jobs resulting in successful exchanges.

After obtaining a closed interval [Tmax1,Tmax2] by the process of range location, we next attempt to reduce this range such that the optimal Tmax continues to lie in it. This is called Phase II. Such a reduction can be performed in three ways.

- By reducing the higher of the two Tmax values while keeping the other one the same (here to fore referred to as Strategy A). If $Tmax_1$ is the higher value among $Tmax_1$ and $Tmax_2$ values, then according to this strategy, $Tmax_1$ is decreased to $Tmax_{1'}$ while $Tmax_2$ remains the same. This is shown in the figure 3-a.

- By reducing the higher and increasing the lower of the two values (here to fore referred to as Strategy B). That is, $Tmax_2$ is increased to $Tmax_{2'}$ and $Tmax_1$ is decreased to $Tmax_{1'}$. This is shown in figure 3-b.

- By increasing the lower of the two Tmax values (here to fore referred to as Strategy C). That is $Tmax_2$ is increased to $Tmax_2'$. $Tmax_1$ remains the same. This is shown the figure 3-c.

A range relocation may be necessary within the range obtained as a result of Phase II after this process of range reduction. However, the motivation behind this process of range reduction is to know a smaller interval in which the optimum value of Tmax lies.

The procedure adopted to accomplish range reduction involves a series of job exchanges between the two machines including single job exchanges as well as multiple job interchanges. The process of considering all possible multiple job exchanges is equivalent to solving a knapsack problem to obtain all feasible solutions. Due to the tremendous amount of computational efforts involved to do this, we consider instead exchanges of a particular type. This makes the algorithm a heuristic procedure, even though as is shown later, the solutions obtained are found to be almost optimal. After range reduction, the algorithm then tries range relocation within the range obtained as a result of Phase I.

Figure 4 gives the overall logic of the procedure.

3.2.2 Motivation for the Algorithm to minimize NT

The algorithm developed follows a search tree procedure. The scheduling of jobs is carried out beginning from the last position. Hence, the makespan value on each machine needs to be known. These makespan values are determined based on the

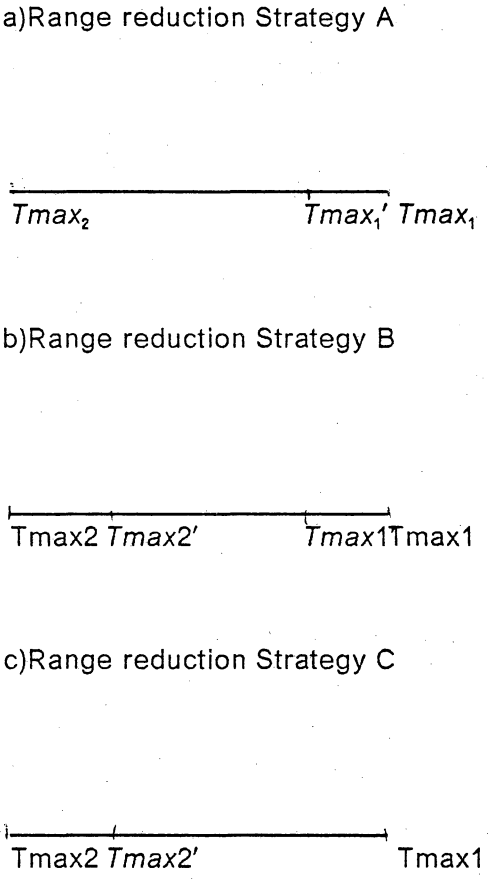


Figure 3. Range reduction strategies

Tmax value and the maximum due date. Since due to scheduling backward, the finish time of a job is known when it is scheduled, its corresponding tardiness value can be determined. Hence, only those jobs that do meet the primary criterion at a position are considered for that position. The allocation of each eligible job there gives rise to a branch of the search tree and a corresponding node of the partial sequence with a knowledge of the tardiness value of the jobs in that partial sequence. To select a node to branch from, the downtrack scheme is used where we select the partial sequence with the highest NT value.

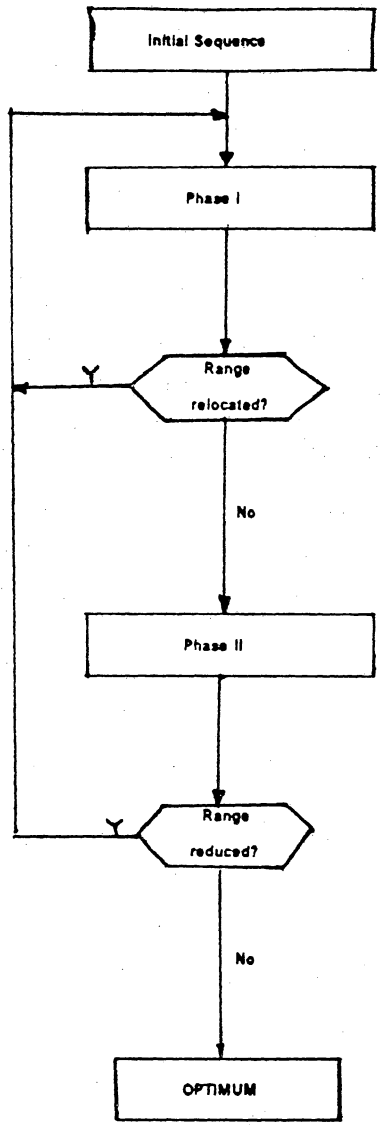


Figure 4. Overall logic of the procedure used

3.3 Exchange Procedure Analysis

Next, we present analysis to determine the sets of jobs to be exchanged between the two machines to implement both the range relocation (Phase I) and the range reduction (Phase II) steps. The range once again corresponds to the $Tmax_1$ and $Tmax_2$ values of the jobs allocated to the machines 1 and 2, respectively. Since the requirements of each of these steps are different, they lead to different sets of conditions to select the sets of jobs to be exchanged that can assure range relocation or range reduction as the case may be. Next, we develop these conditions.

3.3.1 Development of conditions for range relocation

The analysis is divided into two cases depending upon whether or not the jobs corresponding to the Tmax values on each machine are considered as part of the set of jobs to be interchanged.

3.3.1.1 Analysis for the case when the jobs with Tmax are not exchanged

Consider any assignment of jobs to machine 1 and machine 2. Let the jobs assigned to each machine be arranged in an EDD sequence to minimize the Tmax value on each machine individually. Let Tmax1 be the maximum tardiness value on machine 1 and Tmax2 be the maximum tardiness value on machine 2. The maximum tardiness value, Tmax, corresponding to the schedule = Max(Tmax1,Tmax2).

Now, consider the interchange of job i from machine 1 with job j from machine 2 such that i .b. j. and i and j are not the jobs with the $Tmax_1$ and the $Tmax_2$ values respectively. Let p_i and p_j be their respective processing times. Such an interchange is shown in Figure 5. The analysis is carried out depending upon the relative values of p_i and p_j

Case a

$p_i < p_j$

Machine 1

a. If Tmax1 belongs to a job k such that $d_i > d_k$, then Tmax1 remains the same or increases after the interchange. (Refer to figure 6-a)

b. If Tmax1 belongs to a job k such that $d_i \leq d_k \leq d_j$, then Tmax1 decreases if the same job k has the maximum tardiness value on machine 1 again. However, now Tmax1 may belong to some other job after the interchange of jobs i and j. (Refer to figure 6-b)

c. If Tmax1 belongs to a job k such that $d_k \geq d_j$, then Tmax1 increases. (Refer to figure 6-c)

Machine 2

a. If Tmax2 belongs to a job l such that $d_l < d_i$, then Tmax2 remains the same.

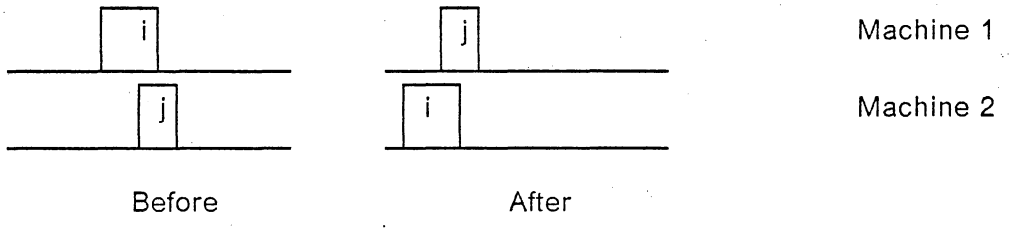


Figure 5. Exchange Procedure Analysis

However, it could increase if Tmax2 belongs to some other job after the interchange. (Refer to figure 7-a)

b. If Tmax2 belongs to a job l such that $d_i \leq d_l \leq d_j$, then Tmax2 increases. (Refer to figure 7-b)

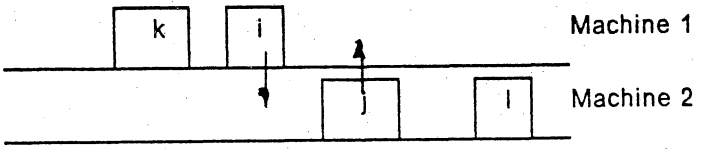
c. If Tmax2 belongs to a job l such that $d_l \geq d_j$, then Tmax2 decreases. However, Tmax2 may belong to some other job after the interchange, in which case Tmax2 may increase, decrease or remain the same. (Refer to figure 7-c)

Note that, in this case, any interchange of jobs between the two machines would lead to a reduction in the value of Tmax beyond Min(Tmax1,Tmax2) only if both Tmax1 and Tmax2 decrease i.e. when $d_i \leq d_k \leq d_j$ on machine 1 and $d_l \geq d_j$ on machine 2. Note that it is not necessary to consider this case when $d_k = d_j$ because then the tardiness values of both jobs k and j will increase irrespective of which job is put first on machine 1. Hence, the condition can be reduced to $d_i \leq d_k < d_j$

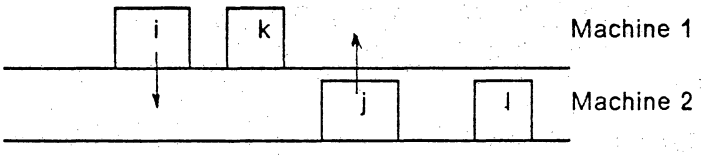
Case b

$p_i \geq p_j$

a)Interchange of job i on machine 1 with job j from machine 2 such that $d_i > d_k$



b)Interchange of job i on machine 1 with job j from machine 2 such that $d_i \leq d_k \leq d_j$



c)Interchange of job i on machine 1 with job j from machine 2 such that $d_k \geq d_j$

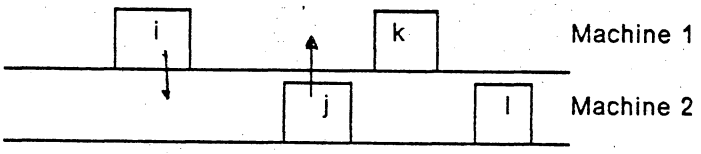
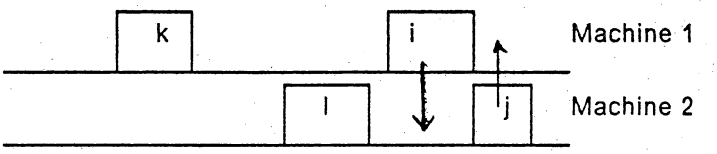
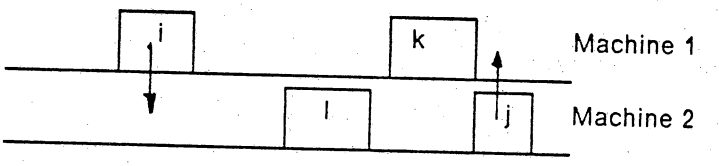


Figure 6. Exchange procedure analysis for tardiness values on machine 1

a)Interchange of job i on machine 1 with job j from machine 2 such that $d_l < d_i$



b)Interchange of job i on machine 1 with job j from machine 2 such that $d_i \leq d_l \leq d_j$



c)Interchange of job i on machine 1 with job j from machine 2 such that $d_l \geq d_j$,

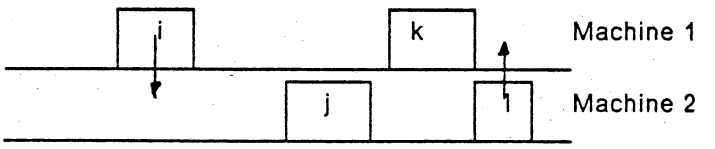


Figure 7. Exchange procedure analysis for tardiness values on machine 2

Machine 1

a. If Tmax1 belongs to a job k such that $d_k < d_i$ then Tmax1 remains the same.

b. If Tmax1 belongs to a job k such that $d_i \leq d_k$ then Tmax1 decreases.

Machine 2

a. If Tmax2 belongs to a job l such that $d_l < d_i$, then Tmax2 remains the same. However, it could increase if Tmax2 belongs to some other job after the interchange.

b. If $d_l > d_i$ then Tmax2 increases.

Therefore, we observe that, in this case, it is not possible to decrease Tmax1 and Tmax2 at the same time.

Thus, from among Cases a and b, we have a possibility of an interchange of jobs i and j leading to a Tmax value less than Min(Tmax1,Tmax2) when Tmax1 belongs to a job k on machine 1 and Tmax2 belongs to a job l on machine 2 such that $p_i < p_j$, $d_i \leq d_k \leq d_j$ and $d_l \geq d_j$. Next, we precisely derive the necessary conditions for an interchange of jobs i and j to result in the Tmax values less than min(Tmax1, Tmax2). Such an interchange of jobs i or j will be called the successful interchange. To that end we need to consider two cases.

Case 1

$T_k > T_l$

Let k denote the job on machine 1 which has the maximum tardiness value and let l denote the job on machine 2 with the maximum tardiness value. (The machines are designated as 1 and 2 such that k .b. l)

Since $d_i \leq d_k < d_j$, (by the condition stated above), T_k would decrease by a value of p_i . Hence, Tmax would decrease beyond T_l if

$$T_k - p_i < T_l$$

Considering the processing times and due dates of jobs to be integer values, we require that

$$p_i \geq T_k - T_l + 1 \qquad [3.3.1]$$

Now, consider the jobs 'p' such that i .b. p .b. j and p belongs to machine 2. Let C_p and C_p' denote the completion times of these jobs before and after the interchange of jobs i and j Since the completion times of these jobs would increase by an amount p_i ,

$$C_p' = C_p + p_i \qquad [3.3.2]$$

From equations 3.3.1 and 3.3.2 we get

$$C_p' \geq C_p + T_k - T_l + 1 \qquad [3.3.3]$$

Denoting by L_p and L_p' the lateness of job p before and after the interchange, the minimum value of L_p' would be

$$L_p' = L_p + T_k - T_l + 1$$

In order to obtain a 'successful interchange', we require that

$$L_p' < T_l$$

i.e. we require that

$$L_p + T_k - T_l + 1 < T_l$$

$$T_l - L_p > T_k - T_l + 1$$

Thus we require that for all jobs 'p' such that i .b. p .b. j and p belonging to machine 2 ,

$$\min_p [T_l - L_p] > T_k - T_l + 1$$

Now, for any job 'm' on Machine 1 such that $d_m \geq d_j$ the completion time after the interchange increases by a value of $p_j - p_i$. Note that $p_j > p_i$ because that is the condition which can lead to a reduction in the Tmax value (See Case a). Hence, $C_m' = C_m + (p_j - p_i)$, where C_m and C_m' are the completion times before and after the interchange. Since $p_j > p_i$, this further implies that

$$C_m' \geq C_m + 1$$

$$\Rightarrow L_m' \geq L_m + 1 \qquad [3.3.4]$$

Since for a successful interchange we require that

$$L_m' < T_l \qquad [3.3.5]$$

where, L_m' is the lateness value of job m after the interchange, we have from equations (3.3.1) and (3.3.2)

$$T_l > L_m' \geq L_m + 1 \qquad [3.3.6]$$

$$\Rightarrow T_l > L_m + 1$$

$$\Rightarrow L_m < T_l - 1$$

Now, based on the above analysis we list the set of necessary conditions for a 'successful interchange' when $T_k > T_l$. (Refer to figure 6-b)

a. There should exist a job i on machine 1 such that

$p_i \geq T_k - T_l + 1$ and

$$d_i \leq d_k \qquad [3.3.7]$$

.

b. There should exist a job j on machine 2 such that

$d_k \leq d_j \leq d_l$ and

$$p_j > p_i \qquad [3.3.8]$$

c. For jobs p on machine 2 such that $d_i \leq d_p \leq d_j$

$$\min_p [T_l - L_p] \geq T_k - T_l + 2 \qquad [3.3.9]$$

d. For jobs m on machine 1 such that $d_m \geq d_j$

$$L_m \leq T_l - 1 \qquad [3.3.10]$$

Case 2

$T_k \leq T_l$

Since, $d_i \leq d_k$ and $d_k < d_j \leq d_l$ (by the conditions stated earlier), for the job l on machine 2 (with the highest tardiness value), T_l decreases by $p_j - p_i$. Now, in order to obtain new Tmax less than min(Tmax1,Tmax2), we require that

$$T_l - (p_j - p_i) < T_k$$

$$\Rightarrow p_j - p_i \geq T_l - T_k + 1$$

(again by assuming all p and T values to be integers) Now, for any job m on machine 1 such that $d_m \geq d_j$, the value of lateness after interchange $L_m' = L_m + p_j - p_i$, where L_m is the lateness value before the interchange. In order with the above requirement, we have,

$$L_m + (p_j - p_i) < T_k$$

$$\Rightarrow T_k - L_m > p_j - p_i$$

or

$$T_k - L_m \geq p_j - p_i + 1$$

Since, $p_j - p_i \geq T_l - T_k + 1$ (from above), we have $T_k - L_m \geq T_l - T_k + 2$

Since the latest possible position of job j can be l, the necessary condition is that for all jobs m on machine 1 such that $d_m \geq d_l$, we require that

$$L_m \leq (T_k - 2) - (T_l - T_k)$$

Next, consider any job p on machine 2 such that $d_i \leq d_p \leq d_j$. The lateness value L_p' after the interchange would be $L_p' = L_p + p_i$ where L_p is the lateness value before the interchange.

Again, we require that

$$L_{p'} < T_k$$

$$\Rightarrow L_p + p_i < T_k$$

$$\Rightarrow T_k - L_p > p_i$$

$$\Rightarrow T_k - L_p \geq p_i + 1$$

Thus the necessary condition becomes

$$L_p \leq T_k - (p_i + 1) \qquad [3.3.11]$$

where p_i is the minimum processing time amongst all jobs m such that $d_m \leq d_k$ and m belongs to machine 1 because if this condition is not satisfied even for minimum p_i job then it is not possible to reduce Tmax value beyond T_k.

Hence, the set of necessary conditions for a successful interchange when $T_k \leq T_l$ is as follows.

a. There should exist a job i on machine 1 such that $d_i < d_k$ and a job j on machine 2 such that

$d_k \leq d_j \leq d_l$ and

$$p_j \geq \min(p_i) + (T_l - T_k + 1) \qquad [3.3.12]$$

b. For any job m on machine 1 such that $d_m > d_l$

$$L_m \leq (T_k - T_l) + (T_k - 2) \qquad [3.3.13]$$

c. If 'i' is a job on machine 1 to be interchanged with a job j on machine 2, then we require that for any job p on machine 2 such that $d_i \leq d_p \leq d_j$

$$L_p \leq T_k - p_i - 1 \qquad [3.3.14]$$

Note that the above condition can be relaxed by considering job p such that $d_i \leq d_p \leq d_s$ (instead of $d_i \leq d_p \leq d_j$), where $d_s \leq d_j$ for all jobs j on machine 2 such that $d_k < d_j < d_l$. This will be referred to later as the Relax1 condition.

This indicates that if any one of these necessary conditions is violated, then it is not possible to get a schedule with Tmax less than $\min(T_k, T_l)$.

3.3.2 Extension to multiple job exchange

In the above discussion we restricted our attention to the exchange of a single job 'i' on machine 1 with a single job 'j' on machine 2. However, we can easily extend the arguments to the cases where jobs 'i' or 'j' would represent a set of jobs on machines

1 and 2 respectively. These are developed based on the conditions for the single job interchange developed in section 3.3.1.

Case 1

$T_k > T_l$

Considering condition 3.3.7 we define job 'i' to be a set of jobs A such that every job $\alpha \in A$ belongs to machine 1 and $d_\alpha \leq d_k$ for all $\alpha \in A$. Hence, by the virtue of the arguments presented in the case of single job exchange, we would require that $\sum_{\alpha \in A} p_\alpha \geq T_k - T_l + 1$.

Considering now 3.3.8 instead of a single job 'j', we can have a set of jobs B such that every job $\beta \in B$ belongs to machine 2 and $d_k \leq d_\beta \leq d_l$. Hence the requirement with respect to the processing times of these jobs translates as $\sum_{\beta \in B} p_\beta > p_i$ or $\sum_{\beta \in B} p_\beta > \sum_{\alpha \in A} p_\alpha$ depending upon whether a single or multiple job exchange is considered with respect to job 'i'. (Note that we can have the exchange of a single job 'i' with a set of jobs 'j' and also the exchange of multiple jobs 'i' with a single job 'j'). In case of the interchange of multiple jobs 'i' (or a set of jobs A) with a single job 'j', we require that $p_j > \sum_{\alpha \in A} p_\alpha$.

Condition 3.3.9 can be extended to the multiple job exchange as follows. Let j_1 represent the job amongst the jobs B on machine 2 which has the minimum due date. Since for all the jobs P on machine 1 such that $d_i < d_p < d_{j_1}$, $(p \in P)$ (when a single job 'i' is being interchanged with a set of jobs P) the lateness would increase by a value equal to p_i, we would require that for all these jobs P

$$\min_{p \in P} [T_l - L_p] \geq T_k - T_l + 2$$

Note that this requirement with respect to the lateness values is the same in the case when we interchange a set of jobs A on machine 1 with a set of jobs B on machine 2 because $\sum_{\alpha \in A} p_\alpha \geq T_k - T_l + 1$.

Condition 3.3.10 can be extended as follows. Let j_2 represent the jobs amongst the set of jobs B with the highest due date. Now, note that for all job M on machine 1 such that $d_m \geq d_{j_2}$, $m \in M$, the lateness value after the interchange increases by an amount equal to $\sum_{\beta \in B} p_\beta - p_i$ (when a single job i is interchanged with a set of jobs B) and by $\sum_{\beta \in B} p_\beta - \sum_{\alpha \in A} p_\alpha$ (when a set of jobs A on machine 1 is being interchanged with a set of jobs B on machine 2.) In either case, the requirement with respect to the lateness values of these jobs M becomes

$$L_m \leq T_l - 1$$

.

Case 2

$T_k \leq T_l$

In this case, note that we are looking for a job i on machine 1 such that $d_i < d_k$ and the processing time of job i is minimum. This is of course motivated by the fact that we would like to keep the increase in lateness of jobs on machine 2 to a minimum level. Hence, note that this is equivalent to a set of jobs A on machine 1 and a set of jobs B on machine 2 such that for every job $\alpha \in A$ and every job $\beta \in B$ we have $d_\alpha < d_k$ and $d_\beta < d_k$ and $\sum_{\alpha \in A} p_\alpha > \sum_{\beta \in B} p_\beta$ such that $\sum_{\alpha \in A} p_\alpha - \sum_{\beta \in B} p_\beta$ is minimum. (In the algorithm presented, we approximate this multiple job search to the search of a single job α and a single job β.)

Considering now Condition 3.3.12 we seek a set of jobs B_2 on machine 2 such that $\sum_{\beta \in B_2} p_\beta \geq p_i + (T_l - T_k + 1)$ (when a single job 'i' is being exchanged) or $\sum_{\beta \in B} p_\beta \geq (\sum_{\alpha \in A} p_\alpha - \sum_{\beta \in B_2} p_\beta) + (T_l - T_k + 1)$ (when we have multiple jobs A and B_2 as described above).

Similarly, referring to condition 3.3.13, for all the jobs M on machine 1 such that $d_m > d_l$, $m \in M$ the lateness value would increase, the requirement with respect to the lateness values of these jobs is the same thereby resulting in,

$$L_m \leq (T_k - T_l) + (T_k - 2)$$

Condition 3.3.14 of the single job exchange case, can be extended to the multiple job scenario as follows. Note that if δ represents the highest of the due dates amongst the jobs α and β (as described above), (δ is equivalent to job 'i' in the case of single job), and if γ represents the job with the highest due date in the set of jobs B_2 (described above), for jobs 'p' on machine 2 such that $d_\delta \leq d_p < d_\gamma$, the lateness value would increase. Hence, the requirement with respect to the lateness values of these jobs becomes

$$L_p \leq T_k - (\sum_{\alpha \in A} p_\alpha - \sum_{\beta \in B_2} p_\beta) - 1$$

(in the case when job sets A and B are considered.) or

$$L_p \leq T_k - p_i - 1$$

(when a single job i is considered.).

3.3.3 Analysis for the case when the jobs with the Tmax values are exchanged

Now, let us consider the case when the jobs considered for exchange namely, job i or job j (or both) correspond to the jobs with tardiness value Tmax1 and Tmax2 respectively. In general, this can lead to the consideration of the following three cases

Case 1 Job k is moved from machine 1 to machine 2 but job l is not .

Case 2 Job l is moved from machine 2 to machine 1 but job k is not.

Case 3 Job k is moved from machine 1 to machine 2 and job l is moved from machine 2 to machine 1.

Our objective is to switch jobs so that min (T_k, T_l) is decreased at least by a unit value to obtain a new range location. Thus, whether a)$T_k < T_l$ or b)$T_k \geq T_l$, we require that

a. Some job 'm' such that m .b. k should be shifted from machine 2 to machine 1.

b. If job k is the first job in the sequence to be shifted from one machine to another, then it should be moved such that it gets an earlier start time on machine 2. However, in the algorithm suggested, we begin by assigning the jobs to the two machines such that each job (in the EDD sequence) gets the earliest possible start. Hence, it would be necessary to move a job 'm' from machine 2 to machine 1 with m .b. k. Also, we note that we would require that $p_m < p_k$, because, otherwise for some job 'p' on machine 1 such that p .b. k, the completion time value obtained after the interchange could be greater than the completion time value of job k before the interchange thus leading

to a higher tardiness value. In case job k is the first job on machine 1, then note that there exists no job m such that m .b. k and hence the range canno be relocated.

c. If job k is moved from machine 1 to machine 2 and job m from machine 2 to machine 1 with m .b. k and $p_m < p_k$, then it is necessary to move job l from machine 2 to machine 1 because otherwise there would be an increase in the completion time of job l on machine 2 and this is not desired.

Thus we note that case 1 becomes identical with case 3.

As regards case 2, we note that it is necessary to move some job 'i' with i .b. k from machine 1 to machine 2 so that the tardiness of job k decreases. Now, considering the situation pertaining to a)$T_k < T_l$ and b)$T_k \geq T_l$, the analysis becomes identical to that of the analysis presented in section 3.3.1 (when jobs with Tmax values are not interchanged.) with only job j being replaced by job l.

Hence, we note that in the siuation when jobs pertaining to Tmax values are considered for exchange we need to consider only Case 3, that is, when both job k and job l are shifted. To implement Case 3 we follow the procedure of Case 1 (as Case 1 was shown to be the same as Case 3). Thus we would shift job m such that m .b. k from machine 1 to machine 2, job k from machine 1 to machine 2 and job l from machine 2 to machine 1. Extension of this argument to a case of multiple job exchange would require the interchange of a set of jobs A on machine 1 such that for every job $\alpha \in A$, $d_\alpha < d_k$, with a set of jobs B on machine 2 such that for every job $\beta \in B$ $d_\beta < d_k$, we would require that $\sum_{\beta \in B} p_\beta - \sum_{\alpha \in A} p_\alpha < p_k$. However, this extension has not been adopted in the algorithm presented in order to avoid high computational complexity.

3.4 Exchange procedure Analysis To Constrict the Range Obtained

Now that we have an assignment of jobs to the two machines such that no change in the assignment would lead to a value of Tmax less than $\min(T_k, T_l)$, we reassign the jobs so as to constrict the range over which the Tmax value lies. The aim is to constrict it to an extent that either it becomes small enough or it cannot be reduced any more indicating the attainment of the optimal Tmax value. This is achieved by alternatively using procedures to constrict the range (Phase II) and to relocate the range (Phase I).

Constriction of the range is done in three different ways depending upon the locations of the jobs from machines 1 and 2 considered for exchange.

1. *Strategy a* By decreasing the value of $\max(T_k, T_l)$ while keeping the value of $\min(T_k, T_l)$ to be the same.

2. *Strategy b* By decreasing the value of $\max(T_k, T_l)$ and increasing the value of $\min(T_k, T_l)$ simultaneously.

3. *Strategy c* By increasing the value of $\min(T_k, T_l)$ while keeping the value of $\max(T_k, T_l)$ to be the same.

If the exchange procedure uses Strategy a then it is obvious that we again have the set of values T_k and T_l such that the optimal value of $Tmax \in [T_k, T_l]$ because the Tmax value obtained as a result of strategy a is an upper bound on the optimal value. In

case of the use of strategies b or c it is necessary to verify once again if the optimal value of Tmax lies over the new range corresponding to the new T_k and T_l values obtained as a result of strategies b or c. This is checked by applying Phase I. This results in the identification of a range smaller than before over which the optimal Tmax value lies. In the following section, an analysis is presented w.r.t the jobs being interchanged between the two machines in accordance with each of the above strategies.

The analysis is divided into two cases depending upon whether $T_k < T_l$ or $T_k > T_l$

Case a

$T_k < T_l$

Strategy a (Refer to figure 8)

Since T_k should not be increased we require that the job/jobs 'j' transferred to machine 1 should lie such that $d_k < d_j$. Also, in the process of interchange we require that for all jobs 'P' on machine 1 such that $d_j < d_p$ for all $p \in P$, where j is the job being transferred,

$$L_p + p_j \leq T_l \qquad [3.4.1]$$

Also, we would require that the value of T_l does not decrease beyond the value of T_k. Strategy a is helpful in a faster constriction of the range. Hence a required condition on the processing time of the job 'j' transferred from machine 2 to machine 1 would be that $d_k < d_j \leq d_l$, $j \in machine2$ and j .b. l, and

$$T_l - T_k \geq p_j \qquad [3.4.2]$$

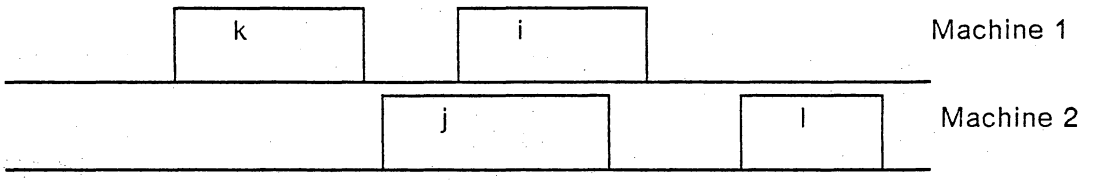


Figure 8. Case a - Strategy a

A more general way of following Strategy a is by the interchange of job/jobs 'i' on machine 1 with job/jobs 'j' on machine 2 such that $\sum p_i < \sum p_j$ and $T_l - T_k \geq \sum p_j - \sum p_i$ However, this leads to an excessive number of possibilities that can be considered. In the algorithm presented, we restrict the search to the interchange of single job i with a single job j only. Even though this is an approximation to the general case, it was found to be quite effective.

Strategy b (Refer to figure 9)

For a job exchange that follows this Strategy, we would consider the exchange of jobs 'i' belonging to Machine 1 and 'j' belonging to Machine 2 such that $d_i \leq d_k$ and $d_j < d_k$ designated as exchange type 1 or the transfer of a single job j from machine 2 to machine 1 such that $d_j < d_k$ designated as exchange of type 2.

Since we are interested in having the final values of T_k and T_l (i.e the values after the interchange of jobs) to lie in the closed interval $[T_k, T_l]$, in the exchange of type 1 we would require that

$$T_k + (p_j - p_i) < T_l$$

OR

$$p_j - p_i \leq T_l - T_k \qquad [3.4.3]$$

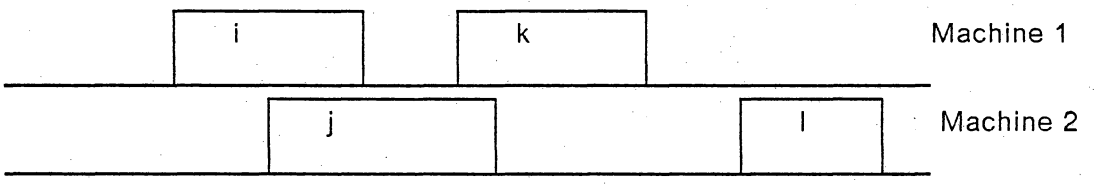


Figure 9. Case a - Strategy b

and in the exchange of type 2, we require that

$$T_k + p_j < T_l$$

OR

$$p_j < T_l - T_k \qquad [3.4.4]$$

Also note that the exchange of type 1 can be generalized to the interchange of a group of jobs $i \in Machine1$ with a group of jobs $j \in Machine2$ such that

$$\sum p_j - \sum p_i < T_l - T_k \qquad [3.4.5]$$

However, such a generalized exchange procedure would result in an exponential increase in the algorithmic complexity. Also, by not verifying for all possible interchanges of this kind would not affect the optimal value of the solution obtained if the values of T_k and T_l differ by a maximum amount equal to the minimum processing time of the jobs. Hence, the algorithm presented restricts the verification of the possible interchanges to the interchange of a single job j on Machine 2 with a single job i on Machine 1.

Strategy c In this case, we would require the jobs to be exchanged such that job i belongs to machine 1 and job j belongs to machine 2 , $p_i = p_j$, $d_j < d_k$ and $d_k < d_i < d_l$.

Case b

$T_l < T_k$

Strategy a In this case, we would require the jobs to be exchanged such that job i belongs to machine 1 and job j belongs to machine 2 , $p_i = p_j$, $d_i < d_k$ and $d_k < d_j < d_l$.

Strategy b (Refer to figure 10)

This would require either the transfer of a single job i from M1 to M2 such that $d_i < d_k$ (interchange of type 1) or the interchange of job i on Machine 1 with job j on Machine 2 such that $p_i > p_j$ and $d_i < d_k$ and $d_j < d_k$ (interchange of type 2). As noted in case a, this procedure can be generalized to the interchange of a group of jobs A on Machine 1 with a group of jobs B on Machine 2 such that for every job $\alpha \in A$ and $\beta \in B$, $d_\alpha < d_k$ and $d_\beta < d_k$.

However, the algorithm considers only the interchange of a single job i on machine 1 with a single job j on machine 2.

Again, the restriction of the final values of T_k and T_l to the closed interval $[T_l, T_k]$ would require that

$p_i \leq T_k - T_l$ in case of interchange of type 1

$p_i - p_j \leq T_k - T_l$ in case of exchange of type 2

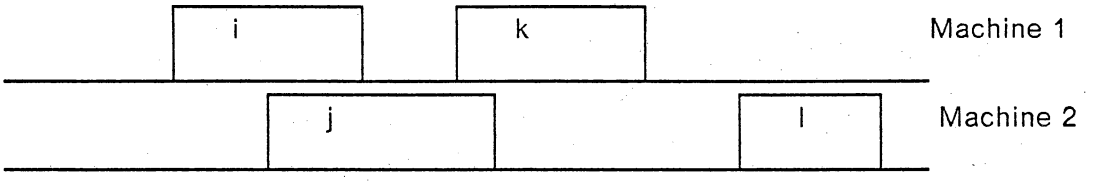


Figure 10. Case b - Strategy b

Strategy c (Refer to figure 11)

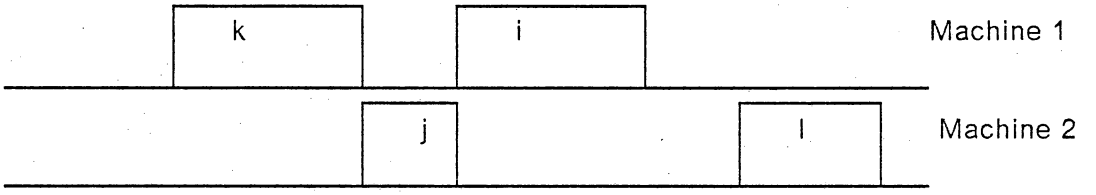


Figure 11. Case b - Strategy c

This would require either the transfer of a single job i from M1 to M2 such that $d_k \leq d_i \leq d_l$ (interchange of type 1) or the interchange of job i on Machine 1 with job j on Machine 2 such that $p_i > p_j$ and $d_k < d_i < d_l$ and $d_k < d_j < d_l$. As noted in the use of Strategy b, this procedure can be generalized to the interchange of a group of jobs A on Machine 1 with a group of jobs B on Machine 2 such that for every job α and β, $d_\alpha < d_k$ and $d_\beta < d_k$.

However, by considering only single element sets A and B, and calling these jobs to be i and j as shown in figure 11 to restrict the final values of T_k and T_l to the closed interval $[T_l, T_k]$ requires that

$p_i \leq T_k - T_l$ in case of interchange of type 1

$p_i - p_j \leq T_k - T_l$ in case of exchange of type 2

3.5 Properties of an optimal solution

While the jobs are being interchanged between machines 1 and 2, an optimal solution is obtained if

- No interchange of jobs between the two machines would lead to a value of Tmax less than $\min(T_k, T_l)$ and
- No interchange of jobs between the two machines would help us to constrict the closed interval $[T_k, T_l]$.

3.6 Tmax minimization algorithm

Based on the analysis presented above and the required properties of an optimal solution, an algorithm to minimize Tmax is presented next. The algorithm is presented in two parts, namely, a)Phase I, and b) Phase II. Phase 1 deals with the verification of the completion times and processing times of the jobs in the current schedule (we start by arranging the jobs in EDD sequence on the two machines in that order giving each job the earliest possible start), and performing a reallocation (by means of exchange of jobs between the machines) of jobs to determine the location of the range over which the minimal value of Tmax lies. At the end of Phase 1, Max(Tmax1 , Tmax2) sets an upper bound, and Min(Tmax1 , Tmax2) sets a lower bound on Tmax. (*Tmax*1 refers to the maximum tardiness value on machine 1 and *Tmax*2 refers to the maximum tardiness value on machine 2.)

In Phase II, the schedule obtained at the end of Phase 1 is modified,in order to constrict the range obtained in Phase 1. After every reduction, the algorithm directs the process to Phase 1 in order to relocate the range within the previous range, The process terminates when it is no longer possible to constrict the range obtained in Phase 1.

3.6.1 Algorithm to Minimize Tmax

A flow chart of this algorithm is given in Appendix A.

Initialization

In steps 1 and 2 of initialization, we arrange the jobs in an EDD sequence on the two machines giving each job an earliest possible start. This gives a good start sequence because the load is evenly distributed between the two machines and hence the values of Tmax1 and Tmax2 in this sequence would be close to the optimal values.

1. Arrange all the jobs in EDD sequence and if $d_i = d_j$ and $p_i > p_j$ then i .b. j.

2. Arrange the jobs in that sequence on the two machines such that each job gets the earliest possible start.

3. Get the maximum tardiness values on the two machines. Let T_k and T_l correspond to the two values such that k .b. l . (Designate the machine to which job k belongs to as Machine 1 and the other machine as Machine 2.)

   Phase I (Range relocation Phase)

This Phase can also be referred to as the range location phase. We try to get the Tmax values on the two machines such that these values are less than the minimum of the two Tmax values on the two machines in the current assignment.This trial is made in two parts. First, we try this by the interchange of jobs other than the ones that have the Tmax values in the current sequence. If this is not possible, then we try the interchange of jobs which have the Tmax values. This is done in step 9.

In steps 5a - 5g we test for the necessary conditions 3.3.7 to 3.3.10 and 3.3.12 to 3.3.14 developed in section 3.3.1 for a successful interchange of jobs between the two machines. If it is found that the interchange of single jobs would not lead to range relocation, then the relaxed conditions to be described subsequently are tested to verify the possibility of a multiple job interchange. The multiple job interchange problems are equivalent to 0-1 knapsack problems. The case of switching the Tmax job on machine 1 (i.e. job k) and the Tmax job on machine 2 (i.e. job l) (discussed in 3.3.3) is considered in step 9.

4. Is there a job 'j' on Machine 2 such that k .b. j .b. l ? If not, go to step 9.

5. If $T_k > T_l$, continue. otherwise go to step 8.

   a. Is there a job 'i' on Machine 1 such that $p_i \geq T_k - T_l + 1$ and i .b. k. If not, go to (h) to obtain multiple jobs i on M1 which satisfy these conditions; (note that a single job i from M1 which increases the tardiness value of a job on machine 2 beyond the Tmax value should not be considered to switch.) Otherwise continue.

b. Determine min p_i such that i belongs to machine 1, $p_i \geq T_k - T_l + 1$ and i .b. k. Call it p_{imin} .

c. Is there a job j on machine 2 such that $p_j > p_{imin}$ and k .b. j .b. l? If not, we cannot locate a job j which satisfies these conditions. Therefore try multiple job j on machine 2, go to 5(n); otherwise continue.

d. Find min $p_j$such that j belongs to machine 2, $p_j > p_{imin}$ and k .b. j .b. l. Call it job j.

e. For every job 'p' on machine 2 such that i .b. p .b. j, Is $L_p \leq T_l - (p_{imin} + 1)$? (L_p denotes the lateness of job p). If not, try multiple jobs from machine 2. To check if multiple job switch could be successful, first test the relaxed conditions. Go to step 6; otherwise continue.

f. For every job 'm' on machine 1 such that j .b. m, Is $L_m \leq T_l - (p_j - p_{imin}) - 1$? If not, try multiple jobs. To check if the multiple job switch could be successful, first test if $L_m \leq T_l - 2$ in Step 7; otherwise continue. Note that the condition $L_m \leq t_l - 2$ is a relaxed version of the above condition as $p_j - p_{imin} \geq 1$. We call this Relax 2 condition.

g. (Come here with a job i on machine 1 and job j on machine 2 by switching which we could potentially relocate the range for Tmax.) Assign job i to Machine 2 and job j to Machine 1.Determine the new values for T_k and T_l (Call them T_{knew} and T_{lnew}). If $\max(T_{knew}, T_{lnew}) \geq T_l$, do not switch jobs i and j, that is reassign job i to machine 1 and job j to machine 2 and go to step 5(a).If switch is successful, go to step 3.

h. (Come here to get multiple jobs i. First check the feasibility of obtaining such a set in step 5 (h) and if so obtain the multiple job set i in steps 5(i), 5(j),5(k) by solving a knapsack problem stated below in step j. If such a set is not possible, exit to Step 9.) Is there a set of jobs A consisting of jobs α' such that α' belongs to machine 1, $\sum p_{\alpha'} \geq T_k - T_l + 1$ α' .b. k for every α' belongs to machine 1. If not, go to step 9. Otherwise continue.

i. Set IPAST = 0, V = $T_k - T_l + 1$.

j. Determine min $\sum p_{\alpha'} x_{\alpha'}$ such that $\sum p_{\alpha'} x_{\alpha'} \geq V$ and $\alpha' \in A$, where A denotes the set of jobs on Machine 1 before job k, and the solution should have $x_{\alpha'} = 1$ for at least one job after IPAST. (This is the problem referred to above as the knapsack problem.) While solving the problem if we have alternate optima, we choose the solution for which $x_{\alpha'} = 1$ and $d_{\alpha'}$ maximum for the last job α' in A. Call this job α with maximum d_α as llast. Set $p_{imin} = \sum p_{\alpha'} x_{\alpha'}$. Continue.

k. For every job p belonging to Machine 2, such that llast .b. p .b. k , Is $L_p \leq T_l - (p_{imin} + 1)$? If not, job p will have its tardiness value larger than that of T_l and this outcome will not relocate the range. Therefore, search for another set A. Set IPAST = llast, and the r.h.s of the knapsack constraint to be V = V + 1 because all the solutions with the previous r.h.s value of V have been considered. Go to 5(j).

l. (Come here with a set A of jobs i on machine 1 and check if the switch is going to be successful.) Verify for every job 'p' which belongs Machine 2, p .b. k and whose completion time would increase by an amount δ (δ could

be different for different jobs because of multiple job switch), if $L_p \leq T_l - \delta - 1$ .If not, set IPAST = Ilast, Go to 5(j). Otherwise continue.

m. Set $p_{imin} = \min \sum p_{\alpha'} x_{\alpha'}$ (as obtained in 5(j)). Denote the last job with $x_{\alpha'} = 1$ as i. Go to 5(c).

n. Determine $\min \sum p_\alpha$ such that $\sum_{\alpha \in A} p_\alpha x_\alpha > RHS$ (initially, RHS = p_{imin}) for jobs α such that α belongs to machine 2 and k .b. α .b. l. Obtain all the alternate optima. Start with solution 1 (i.e. the one was obtained first). Continue.

o. For the current solution being considered, denote the first job α with $x_\alpha = 1$ by j^* (Note that this solution is a collection of jobs from machine 2 for which $x_\alpha = 1$.)

p. For every job 'p' such that p belongs to machine 2 and i .b. p .b. j^*, Is $L_p < T_l - p_{imin}$? If yes, go to 5(q).

If not,

Is there another alternate optimum?

If not, set RHS = RHS + 1. Go to 5(n).

Otherwise, Go to 5(o).

q. Assign the set of jobs with $x_\alpha = 1$ in the current optimal solution to machine 1 and job i (or set of jobs i) to machine 2. Determine the new values of T_k and T_l (Call them as T_{knew} and T_{lnew}.) Is $\max(T_{knew}, T_{lnew}) < T_l$?

- Before we analyze response to this condition, we first make the following observation. This condition may be violated under three different situations.

  - Situation 1 : When $T_{knew} \geq T_l$ but $T_{lnew} < T_l$.

  - Situation 2 : When $T_{lnew} \geq T_l$ but $T_{knew} < T_l$.

  - Situation 3 : When $T_{knew} \geq T_l$ and $T_{lnew} \geq T_l$.

  We note that in situations 1 and 2 it is possible that this condition may still be satisfied if we select different sets of jobs from machine 1 or machine 2 to be switched between machines. We describe the two situations as follows.

  Situation 1 If for any job p on machine 1, the value of T_p obtained after the interchange is such that $T_p > T_l$, then this indicates that $L_p + (p_j - p_i) \geq T_l$ for that job p, where L_p denotes the value of lateness before the interchange. However, since for the given job i, $p_j - p_i$ has been selected to be the minimum possible value, there is a possibility that the value of T_p doesn't exceed T_l if an alternate value of p_i is tried so that $p_i - p_j$ decreases.

  Situation 2 Note that, in case of single job switch we verify in step 5(e) the lateness condition for all jobs 'p' on machine 2 such that i .b. p .b. j However, in case of multiple job switch, this condition is verified in step 5(p) only for jobs p on machine 2 such that $i.b.p.b.j^*$ (where j^* is the first job in the set of jobs j from machine 2 that is being transferred.) .

Therefore, this condition could be violated for jobs on machine 2 after j^* . Hence, there is a possibility that this condition is not violated for any job on machine 2 if an alternate set of jobs on machine 2 is being transferred from machine 2 to machine 1.

- Note that in Situation 3, it is not possible to find another set of jobs on machine 1 and machine 2 because this condition is violated for p_{imin} and $\min_j(p_j - p_{imin})$. For any other set of jobs at least one of these values will increase thereby violating the condition once again.

We now analyze the response to the question : Is $\max(T_{knew}, T_{lnew}) < T_l$? If yes, continue to step 5(r). If not, determine if it is Situation 1, Situation 2 or Situation 3. If it is Situation 3, reassign the jobs to positions as before and go to step 9. If it is Situation 1, Check if there is another job i such that i belongs to Machine 1, i .b. k and $p_i > p_{imin}$. If not, go to step 5(h) to obtain multiple jobs i from machine 1 to switch with jobs from machine 2 to relocate the range. Otherwise, set $p_{imin} = \min(p_i)$ (such that $p_i > p_{imin}$).Go to step 5(n) to obtain multiple jobs j on machine 2 with respect to the new job i. If it is Situation 2, check if there is another optimal solution to the knapsack problem of step 5(n). If not, go to step 5(n) to get the next optimal solution. Otherwise, get the alternate optimum and Go to step 5(o). (This is done to check conditions which may lead to switching of jobs to relocate the range. (in step 5(q)).)

r. Get the new values for T_k and T_l. Go to step 3 to further relocate the range.

- Note : In the following steps 6 and 7 we verify Relax1 condition (see discussion below condition 3.3.14), and a relaxed form of the condition tested in step 5(f) (called Relax 2 condition). These form a set of conditions with respect to the lateness values of the jobs whose lateness would be affected in the process of multiple job interchange.

6. (Testing of Relaxed condition (see discussion below condition 3.3.14))

   a. Let s be a job belonging to machine 2 such that $d_s \leq d_j$ for all jobs j such that k .b. j .b. l and k .b. s .b. l.

   b. For every job p belonging to machine 2 such that i .b. p .b. s, Is $L_p \leq T_l - (p_{imin} + 1)$? (Refer to equation 3.2.14) If not, multiple job switch will not be successful, and go to step 9 to consider switching jobs k and l. Otherwise, go to 5(n).

7. (Testing Relax 2 condition (relaxed version of condition tested in step 5(f)))

   a. For every job m belonging to machine 1 such that l .b. m, Is $L_m \leq T_l - 2$? If not, multiple job switch will not be successful, and go to step 9 to consider switching jobs k and l. Otherwise, go to 5(n).

8. (Execute this step to relocate the range when $T_k < T_l$.) For reference, see figure 12.

   - Note that in order to relocate the range and thereby have both T_{knew} and T_{lnew} less than T_k, we need to decrease T_k at least by a unit value and T_l at least

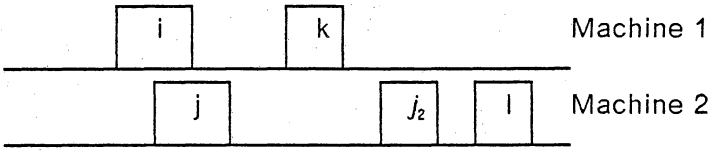


Figure 12. Phase I - when $T_k < T_l$

by an amount $T_l - T_k + 1$. Since we would like to decrease T_k just by the least amount possible, this can be done in two ways as follows.

(1) By shifting job i from machine 1, such that i .b. k where $p_i = 1$. (This is a single job shift.) In this case we designate $p_{imin} = p_i$.

(2) By moving job i such that i .b. k from machine 1 to machine 2, and job j such that j .b. k from machine 2 to machine 1 so that $p_i - p_j$ has the minimum possible value. In this case, we designate $p_{imin} = p_i - p_j$. Note that this is a special case of a multiple job interchange procedure in which we can interchange a set of jobs A on machine 1 with a set of jobs B on machine 2 where the value of $\sum_{\alpha \in A} p_\alpha - \sum_{\beta \in B} p_\beta$ is kept at a minimum possible value. As in the other situation of this type, we consider only a single job switch for the sake of simplicity. Nevertheless, it makes the procedure a heuristic one.

- Now, to decrease the value of T_l by at least $T_l - T_k + 1$, we need to move either a single or multiple jobs j_2 from machine 2 such that $\sum p_{j2} \geq p_{imin} + (T_l - T_k + 1)$

a. Set the following $LB_i = 0$ $UB_i = k$ $LB_j = 0$ $UB_j = k$ $LB_{j2} = k$ $UB_{j2} = l$.

b. (Determine job i) Is there a job i belonging to Machine 1 such that LB_i .b. i .b. UB_i and $p_i = 1$? If so, set $p_{imin} = p_i$ and Go to step 8(d); otherwise continue.

c. (Determine jobs i (from machine 1) and j (from machine 2), if a single job i is not determined in step 8(b). Get a job i belonging to Machine 1 and a job j belonging to machine 2 such that LB_i .b. i .b. UB_i. and LB_j .b. j .b. UB_j such that $p_{imin} = p_i - p_j$ is minimized and $(p_j < p_i)$. Set $p_{imin} = p_i - p_j$. If k = 1,Go to step 9 (because no such i and j can be found); otherwise continue.

d. (Determine job j_2) Is there a job j_2 belonging to Machine 2 such that LB_{j2} .b. j2 .b. UB_{j2} and $p_{j2} > p_{imin} + (T_l - T_k + 1)$. If not, go to step 8(m); to get multiple jobs j_2 on machine 2; Otherwise, find a job j_2 belonging to Machine 2 such that p_{j2} is minimal and $p_{j2} > p_{imin} + (T_l - T_k + 1)$ If no such job j_2 is found, go to step 8 (m); otherwise continue.

e. (Verify if switch would be successful) For every job m belonging to Machine 1 such that j2 .b. m, Is $L_m \leq T_k - (p_{j2} - p_{imin})$? If not, get the latest job 'e' which violates this condition and set $LB_{j2} = e$ and go to step 8 (d) (to check for another j_2); otherwise continue.

f. (Perform the switch) If $p_i = 1$, then assign job i to Machine 2 and job j2 to Machine 1. Otherwise, assign job i to Machine 2 and jobs j and j2 to Machine 1.. Determine the new values for T_k and T_l. Call them T_{knew} and T_{lnew} respectively. If $\max(T_{knew}, T_{lnew}) < T_k$, go to step 3 to further relocate the range; otherwise continue.

g. Determine a set of jobs E such that for $e \in E$, $T_e \geq T_k$.

In steps h, i and j, we reset the values of LB_i , UB_i , LB_j , UB_j , LB_{j2} , UB_{j2} , depending upon the positions of the jobs in the set E.

We do this in order to select other jobs i,j and j_2 which could potentially relocate the range. Three situations can occur depending upon if

1) If $p_i \neq 1$ and i .b. j.

2) If $p_i \neq 1$ and j .b. i.

3) If $p_i = 1$

Each of these are analyzed respectively in steps h, i and j.

h. ($p_i \neq 1$ and i .b. j.) Does there exist a job $e \in E$ such that e belongs to machine 2 and j .b. e .b. k ? If so, set $LB_i = e$. (If there is more than one such job, choose the one that is closest to k.)

Does there exist a job $e \in E$ such that e belongs to machine 2 and k .b. e .b. j2? If so, set $UB_{j2} = e$. (if there is more than one such job 'e', choose the one that is closest to k.)

Does there exist a job $e \in E$ such that e belongs to machine 2 and i .b. e .b. j? If so, set $UB_j = e$. If there is more than one such job, choose the one that is positioned closest to i. Reposition the jobs and go to step 8(c).

i. ($p_i \neq 1$ and j .b. i).

Does there exist a job $e \in E$ such that e belongs to machine 1 and j .b. e .b. i? If so, set $LB_j = e$ (If more than one such job 'e' choose the one that is closest to k.)

Does there exist a job $e \in E$ such that e belongs to machine 2 and i .b. e .b. k? If so, set $LB_i = e$. (If there is more than one such job e, choose the one that is closest to k.)

Does there exist a job $e \in E$ such that e belongs to machine 2 and k .b. e .b. j^2? If so, set $UB_{j2} = e$. If more than one such job e choose the one that is closest to k.)

Reposition the jobs and go to step 8(c).

j. $(p_i = 1)$

Does there exist a job 'e' belonging to Machine 2 such that i .b. e .b. k? If so, set $LB_i = e$.(If more than one such job 'e' exists choose the one that is closest to k.)

Does there exist a job 'e' belonging to machine 2 such that k .b. e .b. j2? If so, set $Ub_{j2} = e$. (If there is more than one such job 'e', choose the one that closest to k.)

Reposition the jobs and Go to step 8(b).

- In the following steps k - n we determine the multiple jobs j_2 on machine 2.

k. Need to determine a set of jobs A so as to $\min \sum_{\alpha \in A} p_\alpha x_\alpha$ such that $\sum_{\alpha \in A} p_\alpha x_\alpha \geq p_{imin} + (T_l - T_k + 1)$ and LB_{j2} .b. α .b. UB_{j2}. Set $RHS = p_{imin} + (T_l - T_k + 1)$. Continue.

l. (Check feasibility of determining a set j_2) Determine $\sum p_\alpha$ for all jobs α such that LB_{j2} .b. α .b. UB_{j2} and α belonging to machine 2. Is $\sum p_\alpha \geq RHS$? If not, go to step 9; otherwise continue.

m. (Determine a set of j_2) Determine $\text{Min}\sum p_\alpha x_\alpha$ ($x_\alpha =$ 0 or 1) such that α belongs to machine 2 and LB_{j2} .b. α .b. UB_{j2}, and $\sum p_\alpha x_\alpha \geq RHS$. Get all the alternate optima.

n. Amongst the set of alternate optima A_t, choose the one with $\min_t \beta_t$ where $\beta_t = \max_{\alpha \in A_t}[\alpha: x_\alpha = 1]$. Call this set of jobs as A. Designate $j^* = \min_{\alpha \in A}[\alpha: x_\alpha = 1]$ $j^{**} = \max_{\alpha \in A}[\alpha: x_\alpha = 1]$

o. If $p_i \neq 1$, then assign all jobs α with $x_\alpha = 1$ to machine 1, job i to machine 2 and job j to machine 1.

If $p_i = 1$, then assign all the jobs α with $x_\alpha = 1$ to machine 1 and job i to machine 2.

p. Determine new values of $T_k$and T_l. Call them T_{knew} and T_{lnew} respectively. If $\max[T_{knew}, T_{lnew}] < T_k$, go to step 3. Otherwise continue.

q. Enlist all jobs 'e' which violate this condition. Go to step 8(h).

9. (Interchange of jobs with Tmax values)

- This step involves the interchange of jobs 'k' and 'l' i.e. the jobs with Tmax values on the two machines. The exchange procedure used is an approximation. The interchange of job 'm' (which is a job before job 'k') theoretically,

could be the interchange of a set of jobs on machine 1 with a set of jobs on machine 2 such that the difference in the sum of their processing times is less than the processing time of job 'k'. This is again indicative of the heuristic nature of the algorithm.

a. Set $LB_m = 0$.

b. Is there a job 'm' belonging to Machine 2 such that LB_m .b. m .b. k and $p_m < p_k$? If not, the range cannot be relocated thus go to step 10 (Phase II) for range reduction; otherwise continue.

c. Get the job 'm' such that p_m is minimum. Shift job m to machine 1, job k to machine 2 and job l to machine 1.

d. Get the values of Tmax1 and Tmax2 after the interchange. Is max(Tmax1,Tmax2) < min (T_k , T_l).? If not, replace the jobs shifted at their original positions. Set LB_m = m. Go to step 9 (b). Otherwise, Go to step-3 to try to relocate the range once again.

10. Phase II (Range Reduction Phase)

This is a range reduction phase. In this phase, a reduction of the gap between the Tmax values on the two machines is tried. This range reduction scheme employs all the three strategies described in section 3.4.

a. Is $T_k = T_l$? If so, Go to step 11. (Range reduction is not possible, optimal solution is obtained.)

b. If $T_k < T_l$ continue.

c. If $T_k > T_l$, go to 10(l).

- Strategy A - Single job Switch

  - Set Tag(i) = 0 for all i. This flag keeps track if a job has been tried for exchange or not. For job i, Tag(i) = 1 if job i has been tried for exchange.

d. Is there a job 'i' belonging to machine 2 such that $p_i < T_l - T_k$ and k .b. i .b. l? If so, choose i such that p_i is minimal and Go to 10(f). Otherwise continue.

- Strategy B - Single job Switch

e. Is there a job i belonging to machine 2 such that $p_i < T_l - T_k$, $Tag_i \neq 1$ (that is the transfer of job i shouldn't have been tried before.) and i .b. l? If so, choose i such that p_i is minimal and continue. Otherwise, go to step 10(h) to try multiple job switch.

f. Assign job i to machine 1. Determine the tardiness of job i when assigned to machine 1. Continue.

g. Is $T_i \geq T_l$? (Note that no job p on machine 1 such that i .b. p would reach a tardiness value greater than or equal to T_l .) If so, reposition job i. Set TAG (i) = 1. Go to step 10(d). Otherwise, the switch is successful. Go to Step 3 to further relocate the range.

- Strategy B - Multiple job switch

- Set Tag(i,j) = 0 for all jobs i and j. This flag keeps track if a set of jobs has been tried for exchange or not. For job (i,j), Tag(i,j) = 1 if the set of jobs i and j has been tried for exchange.

h. Is there a job i belonging to machine 1 and job j belonging to machine 2 such that $Tag(i,j) \neq 1$ (that is, the transfer of this set should not have been considered before.), i .b. k , j .b. k and $p_i < p_j < p_i + (T_l - T_k + 1)$? If not, Go to step 11 (OPTIMUM). (Note that interchanges of more than two jobs is not tried.) If yes, enlist all such pairs.

i. Select a pair (i,j). Two cases may occur, namely, i .b. j or j .b. i. If i .b. j, check if for every job m such that m belongs to machine 2 and i .b. m .b. j. Is $L_m + p_i \leq T_l - 1$? If not, set Tag(i,j) = 1, go to step 10(h). If yes, go to step 10(j) to switch the jobs. If j .b. i, then for every job m such that m belongs to machine 1 and j .b. m .b. i, is $L_m + p_j \leq T_l - 1$? If not, set Tag(i,j) = 1. go to 10(h). If yes, go to step 10(j) to switch the jobs.

j. Assign job i to machine 2 and job j to machine 1. Determine T_i and T_j after the assignment.

k. Is T_{inew} or $T_{jnew} \geq T_l$? If so, set Tag(i,j) = 1, (that is, these jobs are not suitable for transfer from one machine to another), reassign i and j to their original positions and go to step 10(h). Otherwise, go to step 3 to further relocate the range.

- Strategy B - Single job switch

- Set Tag(i) = 0 for all jobs i.

l. (Come here with $T_k > T_l$) Is there a job i belonging to machine 1 such that i .b. k and $p_i \leq T_k - T_l$? If not, go to step 10 (n) and try multiple job switch. If yes, continue.

m. Assign this job to machine 2 and determine the tardiness value for job i (after the assignment).

n. Is $T_i \leq T_k$? If not, reassign i to machine 1 set Tag(i) = 1 and go to step 10(l). Otherwise, go to step 3 to relocate the range further.

- Strategy B - Multiple job switch

  - Set Tag(i,j) = 0 for all pairs of jobs i and j.

o. Is there a job i belonging to machine 1 and a job j belonging to machine 2 such that i .b. k , j .b. k , $Tag(i,j) \neq 1$, $p_j < p_i < p_j + (T_k - T_l + 1)$? If not, go to 10(t) and try strategy C. If yes, enlist all such combinations.

p. Choose the pair (i,j) such that p_j is minimal and p_i is minimal for the given p_i . Call them p_{jmin} and p_{imin} respectively. Two cases may occur, that is i .b. j or j .b. i.

q. If i .b. j, then for every job m belonging to machine 2 and i .b. m .b. j, is $L_m + p_{imin} \leq T_l - 1$. . If not, set $Tag(i,j) = 1$ and go to 10(o). If yes, go to step 10(r). Otherwise, j .b. i. Then, for every job m belonging to machine 1 Is

$L_m + p_{jmin} \leq T_k - 1$. . If not, set $Tag(i,j) = 1$ and go to step 10(o). If yes, go to step 10(r).

r. Assign job i to machine 1 and job j to machine 2. Determine the values of T_i and T_j. Continue.

s. Is $T_i \geq T_k$ or $T_j \geq T_k$? If so, set Tag(i,j) = 1 Go to step 10(o). Otherwise, go to step 3 to relocate the range.

- Strategy C - Single job switch

  - Set Tag(i) = 0 for all jobs i.

t. Is there a job 'i' belonging to machine 1 such that k .b. i .b. l, $Tag(i) \neq 1$ and $p_i \leq T_k - T_l$. If not, Go to 10 (v) to try multiple job switch. If yes, continue.

u. Move job 'i' to machine 2. Compute T_{inew}. If $T_{inew} < T_k$, switch is successful and go to step 3 to relocate the range. Otherwise, set Tag(i) = 1 and go to step 10(t).

- Strategy C - Multiple job switch

  - Set Tag(i,j) = 0 for all pairs of jobs i and j.

v. Is there a job 'j' belonging to machine 2 and job 'i' belonging to machine 1 such that k .b. j .b. l, k .b. i .b. l, $Tag(i,j) \neq 1$ (i.e. the exchange of these two jobs has not have been tried once and failed), $p_i > p_j$ and $p_i - p_j \leq T_k - T_l$. If

not, go to step 11 (OPTIMUM). (Note that, we do not try switching more than two jobs.) Otherwise, continue.

w. Move job i to machine 2 and job j to machine 1. Now, note that there can be two cases, that is i .b. j or j .b. i. If i .b. j, then for every job m belonging to machine 2 and i .b. m .b. j, is $L_m + p_{imin} \leq T_k - 1$. ? If not, set $Tag(i,j) = 1$ and go to step 10(v). If yes, go to step 10(x). Otherwise, j .b. i. Then, for every job m belonging to machine 1 Is $L_m + p_{jmin} \leq T_k - 1$. . If not, set $Tag(i,j) = 1$ and go to step 10(v). If yes, go to step 10(w). If yes, continue.

x. Assign job i to machine 1 and job j to machine 2. Determine the values of T_{inew} and T_{jnew}. Is $T_{inew} \geq T_k$ or $T_{jnew} \geq T_k$? If so, set Tag(i,j) = 1 Go to 10(v). Otherwise, go to step 3 to relocate the range.

11. OPTIMUM Determine the T_k and T_l values for the current schedule. Optimum value of Tmax = $\max(T_k, T_l)$

STOP

3.6.2 Minimization of the secondary criterion

The problem of minimizing the number of tardy jobs subject to a level of maximum tardiness forms a more complicated problem in the two machine case because of the possibility of having various assignments of the jobs to the two machines.

The approach suggested in this research is a branch and bound method which is based along the same lines as the branch and bound method used in the single machine case. It proceeds by scheduling the jobs backward on both the machines. To implement this we need to fix the makespan on both the machines and then make the job allocations. However, none of the dominance rules applicable to the single machine case is applicable to the two machine case as will be explained later.

3.6.3 Some analysis of the Makespan values on the two machines

Let Tmax1 and Tmax2 be the Tmax values obtained on machine 1 and machine 2 respectively, in the Tmax problem. Therefore, the optimal Tmax value for the two machine problem is Max (Tmax1, Tmax2) . Note that the sum of the makespan values of any schedule on the two machines equals the sum of the processing times of the jobs and is therefore a constant. However, if we let C_1 and C_2 be the makespan value on machines 1 and 2 respectively, then C_1 and C_2 could vary depending upon the assignment of jobs to the individual machines. Without loss of generality, let $C_1 \geq C_2$. For every value of C_1 there corresponds a definite value of C_2.

We make the following observation. If d_{max} denotes the maximum value of the due dates, then the maximum possible value of C_1 is $d_{max} + T_{max}$, that does not violate the primary criterion. Also, in order that the optimal value of Tmax is maintained, for any job to end at time 't', we require that $t \leq d_i + Tmax$. Hence, not every combination of C_1 and C_2 will be feasible to the primary criterion. That is, if the jobs are scheduled backwards starting from C_1 and C_2 , we may come across a value of 't' at which no job is eligible to be scheduled with respect to the primary criterion. Also, since the cri-

teria to be optimized form regular measures, we can't have gaps between schedules. Hence, such schedules cannot be optimal (referred to also as infeasible schedules). However, we know that there does exist at least one feasible set of C_1 and C_2 values (in particular, the makespan values corresponding to the schedule that optimizes the primary criterion) that gives a feasible set of makespan values.

In the algorithm presented, we begin by examining various possible assignments beginning with a makespan value $C_1 = d_{max} + Tmax$ on machine 1 and the corresponding value of C_2 on machine 2. As soon as we find that a certain value of C_1 is not feasible, we proceed to the next possible value of the makespan. The procedure is continued till all the possible values of makespan are covered.

3.6.4 Algorithm to minimize number of tardy jobs

Based on the above discussion of makespan values on the two machines and the single machine scheduling algorithm to minimize the tardy jobs, we can outline the steps of the algorithm for the two machine problem.

Notations used

- NT_i = Number of Tardy jobs obtained so far in any branch i.
- UB = the upper bound value on the number of tardy jobs.
- Tminus1 and Tminus2 - Arrays used to store the values of time obtained after scheduling the eligible jobs (one set of values for each active branch).

Step 1 Initialize the values of C_1 and C_2 to be the makespan values as corresponding to the optimal solution obtained for the Tmax criterion. Without loss of generality, let $C_1 \geq C_2$ (that is, the machine with the larger makespan is designated as machine 1.) Set the value of Tmax to be equal to the optimal value obtained for the Tmax problem. Set UB = upper bound value of the number of tardy jobs. (Discussion on upper bound is presented in section 3.6.6)

Step 2 Initialize NT = 0. Set $C = \max(C_1, C_2) = C_1$. Set T = C.

Step 3 Determine the set of jobs which when scheduled to end at time T would not have a tardiness value greater than Tmax. Note, this would also include the jobs that would be early if scheduled to end at that position. From this set,remove the jobs that have been already scheduled (either on Machine 1 or on Machine 2 for the branch under consideration). If no job is eligible, fathom that branch. Go to step 5.

Step 4 For each eligible job i, branch out by scheduling job i to end at time T on the machine from which the value of T was obtained. Determine the corresponding starting time of the job on machine 1 or 2 wherever scheduled. that is T_1 and T_2, obtained after scheduling any eligible job i (for each branch). If for any branch $T_1 < 0$ or $T_2 < 0$, fathom that branch (because this would be an infeasible assignment). Also, fathom a branch if $\max(T_1, T_2) = 0$. This indicates the attainment of a feasible schedule. If $T \leq d_i$, then set NT value for this branch equal to the NT value of the parent branch. Otherwise, NT value of this branch = NT value for the parent branch + 1 (If $NT \geq UB$, then fathom this node). Store the values of T_1 and T_2 corresponding to each branch in the arrays Tminus1 and Tminus2 respectively. (Note, we would have a set (T_1, T_2) for every branch not fathomed and formed by scheduling a job with a different processing time.) Also store the corresponding partial schedules on the two ma-

chines corresponding to that branch. If there are two jobs with the same processing time eligible to be scheduled at this stage, store the schedule corresponding to the job that would not be tardy at this position, otherwise store the schedule corresponding to any one of them. For a branch for which Max (T_1, T_2) determine its NT value and set UB = Min (UB, NT).

Step 5 From the arrays Tminus1 and Tminus2, set $T = \max_{i \in AN}\{ \max[Tminus1(i), Tminus2(i)]\}$, where AN refers to the set of active nodes. If $AN = \phi$, then go to Step 6. Let k be the position in the arrays corresponding to T. Set $T_1 = Tminus1(k)$ and $T_2 = Tminus2(k)$ and update Tminus1 and Tminus2 arrays by deleting these T_1 and T_2 values. Go to Step 3.

Step 6 If $C_1 - 1 \geq C_2 + 1$, Set $C_1 = C_1 - 1$, $C_2 = C_2 + 1$ and Go to step 2. Otherwise, go to Step 7.

Step 7 The current value of the upper bound (UB) gives the optimal value of NT. Stop.

3.6.5 Determination of Lower Bound

An effective lower bound value for each node can be obtained by scheduling the remaining jobs on a single machine and minimizing the number of tardy jobs in this set, for a given value of Tmax. To get the equivalent single machine problem, the equivalent due dates used on a single machine need to be twice the actual due date. Also, the minimum value of Tmax for the single machine problem can be obtained from the EDD sequence.

3.6.6 How to set a tighter upper bound value?

Obviously, the upper bound value as suggested in the algorithm is not very tight. A tighter upper bound can be obtained by minimizing the number of tardy jobs on the two machines individually, for the assignment of the jobs as obtained in the minimization of Tmax.

Also, note that in the algorithm suggested for NT minimization subject to a fixed value of Tmax, no dominance rule has been used. From the algorithm used to minimize NT subject to the optimal value of Tmax we note that while applying the dominance rules, we set up a matrix AFT, such that, if the dominance rules indicate that there exists an optimal schedule where job i precedes job j, then the entry AFT(i,j) is set equal to 1 thereby scheduling only job j when both job i and job j are eligible to end at some time 't'. i.e. we do not branch out scheduling job i to end at time 't' in one branch and job 'j' to end at time 't' in another. However, in the two machine case, the branch where job j is scheduled to end at time 't' may not lead to any feasible assignment thereby implying the non applicability of the dominance rules of the single machine problem to the two machine problem. However, in the calculation of the upper bound value suggested, the assignment of jobs is known for each machine and hence while solving the NT minimization as a single machine problem, all the dominance conditions can be used which would speed up the bound calculation.

Although, this upper bound value would decrease the computational efforts involved in the NT minimization for the two machine problem, it would involve the solution of two single machine problems. Of course, this may be desirable to avoid excessive branching in the two machine problem.

Chapter 4

Computations and Results

The algorithms suggested in Chapter 3 were coded in FORTRAN-77 and the problems generated were run on IBM 3090.

4.1 Results for the Algorithm to minimize Tmax

For the algorithm to minimize Tmax, problems were generated using the due date procedure similar to the one suggested by Emmons. In this procedure, the processing times are selected as uniformly distributed integer values between 1 and u (where u is an integer). Problems were generated for various values of u. The due date for each job is determined by adding a number randomly selected from a uniform distribution of integers between 0 and n, where 'n' represents the number of jobs.

Problem number	Problem size (n)	Average Execution time (sec)
1	50	0.13
2	100	0.34
3	150	0.70
4	200	1.20
5	250	1.86
6	300	2.61
7	350	3.65
8	400	4.57
9	450	5.75
10	500	7.08

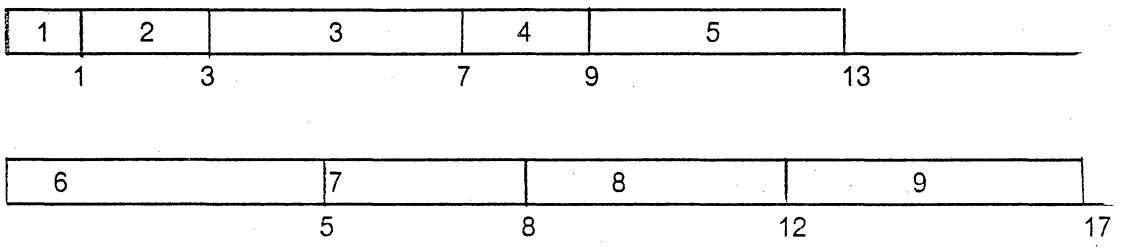
Table 3. Average Execution time vs problem size

$d_i = p_i + U(0,n)$

The results obtained regarding the variation of execution time .vs. the problem size are presented in Table 3. From the table we can infer that the time required for execution does not increase linearly with the size of the problem.

Another observation regarding the working of the algorithm shows that in all the cases the exchanges tried in Phase 1 of the algorithm were not successful. All the job interchanges were carried out in Phase 2. Even though Phase I does not seem to be applicable often, yet it does not mean that it is not required. The following example demonstrates the need of Phase 1 of the algorithm.

Example



$T_k = 2$

$T_l = 1$

Figure 13. Tmax values for an initial assignment

The processing times p_i and the due dates d_i of jobs 1 to 9 are given as follows.

job #	p_i	d_i
1	1	3
2	2	5
3	4	6
4	2	10
5	4	11
6	5	12
7	3	14
8	4	15
9	5	16

Now, for some assignment of jobs to the two machines, arranging these jobs on the two machines, (maintaining EDD sequence on the two machines individually.), we have the completion times and Tmax values as shown in figure 13. Note that the range located now is [1,2]. Now, switching job 2 from machine 1 to machine 2 and job

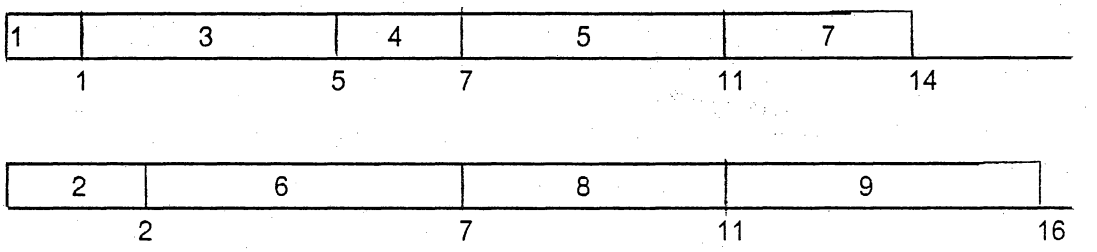


Figure 14. Arrangement of jobs after the Phase I switch.

7 from machine 2 to machine 1, the jobs would be arranged as shown in figure 14. Note that the Tmax value now obtained equals zero. $0 \notin [1,2]$. Hence, we have relocated the range as a result of Phase I operation. $T_{knew} = 0$ $T_{lnew} = 0$

Table 4 shows the variation of the number of iterations required to solve the problem with different values of u (for a constant value of n = 1000). Each iteration here refers to the successful execution of the range reduction step (Phase II). So, in other words the number of iterations also indicate the number of times range reduction is accomplished. We observe that the number of iterations required to solve the problem increases with 'u' (i.e. the greater the variation in job processing times, the greater is the number of iterations required in the knapsacks involving $p_i - p_j$) .

Theoretically, the problem is NP-complete (Rinnooy Kan [13],). However, in the heuristic procedure used, the computational time does not seem to grow exponentially with the size of the problem. This is clear upon the examination of table 3 (also figure 15).

In almost all the cases considered where $p_i = U(1,10)$, an optimal solution was obtained. (Any solution obtained from the algorithm can be claimed to be an optimal

Range of processing times (1,U)	Average # of iterations for problems of size n=1000
(1,10)	2
(1,250)	7
(1,1000)	11
(1,2000)	13
(1,3000)	17
(1,5000)	30

Table 4. Number of iterations vs range of processing times

solution, if the values of T_k and T_l differ by a maximal value of 1 and there is no job 'j' on machine 2 such that k .b. j .b. l ensuring that T_k and T_l form the upper and lower bound values for Tmax.). Table 5 summarizes these results for problems of various sizes. These results are based on five different observations for each problem size.

Further, to study the behavior of the algorithm, problems were run for different types of due dates and processing times. The problem that showed the worst kind of computational complexity had $p_i = 1$ for most of the jobs in between jobs k and l. and due dates for jobs until k, loose due dates (achived by setting the processing times extremely small) for jobs 'j' such that k .b. j .b. l, and a long processing time for job l delaying it by a large amount.

n = 10		n = 50		n = 100		n = 500		n = 1000	
l-k	$T_l - T_k$	l-k	$T_l - T_k$	l-k	$T_l - T_k$	l-k	$T_l - T_k$	l-k	$T_l - T_k$
1	0	1	0	1	0	1	0	1	0
1	1	1	0	1	1	1	1	1	1
1	0	1	0	1	1	1	1	1	0
1	0	1	0	1	0	1	1	1	1
2	1	1	0	1	0	1	0	1	0

Table 5. Difference between the locaions of the Tmax jobs on machine 1 and 2,(namely, jobs k an l), and their tardiness values

It can be noted that starting with an EDD sequence and assigning jobs in this sequence to the two machines in that order so that each job gets the earliest possible start, tends to reduce the number of jobs between k and l to a minimum level and hence in the average case, the suggested algorithm is quite efficient. However, starting with a sequence other than the one suggested in the algorithm, it is observed that there are quite a few successful interchanges carried out in Phase I of the algorithm, that are necessary to set the upper and lower bound values for the minimum value of Tmax. Hence, the time required to arrive at the solution value is relatively higher in these cases.

As an approximation to this procedure suggested to minimize Tmax, we can set the heuristic which implements only Phase II of the algorithm i.e. where, $\max(T_k, T_l)$ sets an upper bound on Tmax and $\min(T_k, T_l)$ sets a lower bound on Tmax.

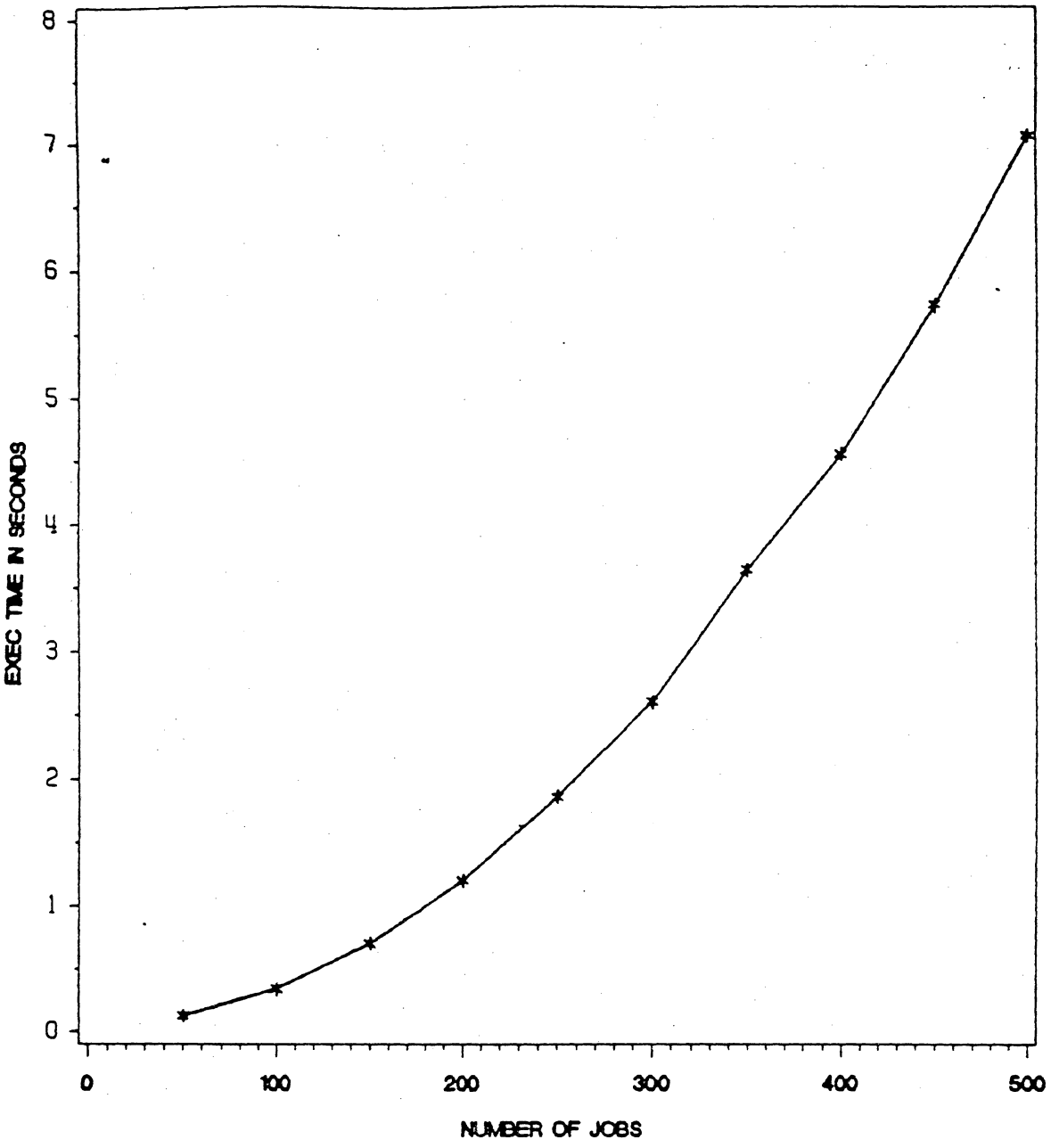


Figure 21. Variation of Execution time for Tmax problem

Such a procedure may not guarantee an optimal solution. However, using the EDD sequence as a start before proceeding with the algorithm, we can say that an optimal solution is generated in most of the cases (based on the computational experience.).

4.2 Results for the Algorithm to Minimize the Secondary Criterion.

The implementation of this algorithm requires the results from the algorithm for the minimization of Tmax. Hence, the execution times obtained, indicate only the amount of computational time required to minimize the secondary criterion given a particular value of the primary criterion.

Computational experience indicates that the high storage requirements for this algorithm restrict its application to problems of larger size, that is for $n > 100$. To study the behavior of the algorithm for the problems generated, the processing times were chosen such that $p_i = U(1, 10)$. The due dates were chosen in the same manner as for the problem to minimize the value of Tmax. A lower bound value was calculated for each node by minimizing the NT value for the remaining jobs when they are arranged on a single machine. In this case, (that is, when the remaining jobs are arranged on a single machine) the due date dd_i for every job i was set equal to 2 * d_i, where d_i is the due date of the job. The nodes with lower bound values greater than the current upper bound were fathomed.

Problem size (n)	Average Time (secs)	Minimum Time (secs)	Maximum Time (secs)	T' (secs)
10	0.36	0.01	0.69	- -
20	0.52	0.04	2.26	- -
30	1.28	0.09	4.20	- -
40	3.49	0.18	8.35	- -
50	13.06	0.33	63.57	- -
75	> 600	0.56	> 600	0.85
100	> 600	> 600	> 600	2.81

Table 6. Execution time vs problem size - NT minimization algorithm

Table 6 shows the variation of execution time vs n. The observations presented in the table are based on the results for five different problems. Columns 3 to 5 report CPU time in seconds required to obtain optimal solutions. Also, we show CPU time required to reach a stage when the least lower bound obtained does not differ from the upper bound value by more than 1. This is designated by T', also depicted in the last column. Note that T' is relatively very small for the problems tested, while the CPU time to obtain an optimal solution took more than 600 seconds for all the problems of size $n = 100$ and for some problems of size $n = 75$.

The variation in computation time (min, max and average) vs problem size is shown in the graph on figure 16 for the algorithm to obtain optimal solution for problems of size upto $n = 50$.

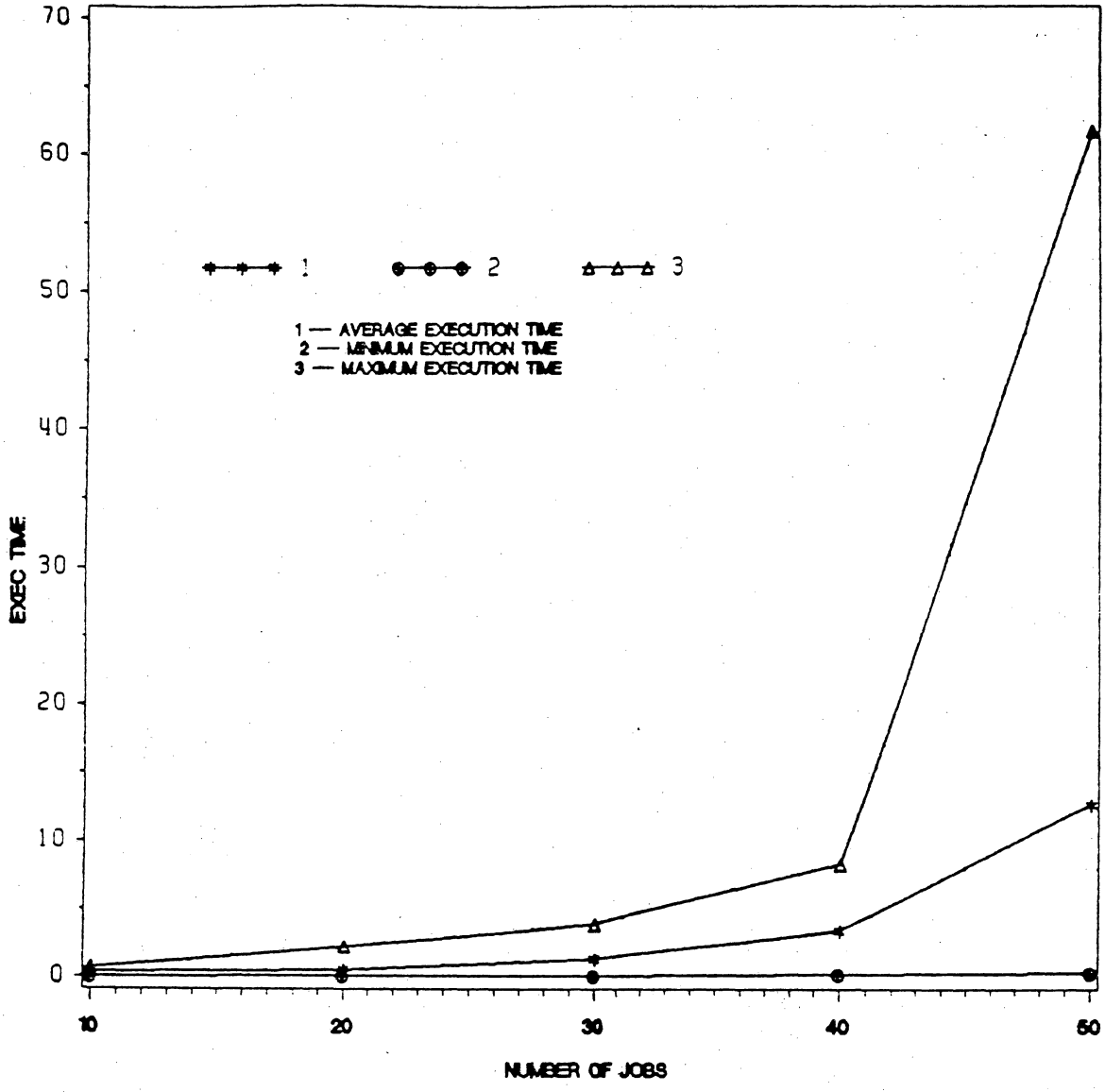


Figure 16. Computational time for NT minimization- two machine case

4.2.1 Comments regarding the upper bound value

The upper bound value used in the analysis was the number of tardy jobs in the sequence that resulted after the minimization of Tmax. The results obtained for the minimum number of tardy jobs were compared with the upper bound values obtained by using the single machine NT minimization algorithm separately on the two machines for the assignment of jobs as obtained at the end of the Tmax minimization algorithm. For all the comparisons made (upto problem size 30), it was observed that the minimum NT value obtained by using the two machine algorithm was the same as that obtained for the upper bound. This is certainly indicative of the fact that the upper bound being calculated is pretty tight.

Also, since the application of the two machine algorithm would be restrictive due to high storage requirements, the schedule corresponding to the upper bound value would provide a good solution for problems of larger sizes. Also, this would allow the user to obtain heuristic solutions for problems of size upto 1000 jobs and also would allow the user to exploit the dominance conditions applicable to the single machine case.

Chapter 5

Conclusion and Future Work

This research was aimed at the development of efficient algorithms to solve the bicriteria problem with Tmax (maximum tardiness) as the primary criterion and NT (number of tardy jobs) as the secondary criterio in the single and two machine cases. Although an exact algorithm is well known for the single machine Tmax problem, the NP complete nature of the Tmax problem in the two machine case restricted the two machine algorithm to minimize Tmax to be a heuristic one. However, results obtained with the proposed procedure indicate the attainment of optimality in most cases. The idea on which the algorithm is based would be difficult to be extended to a multimachine case.

The branch and bound algorithm used to optimize NT in the two machine problem proves to be fairly quick in obtaining the solution close to the optimal value. The algorithm was successfully tested on problems of size upto 100 jobs, which is a fairly large number.

Through out this work the processing time of the jobs on the machines was assumed to be deterministic. However, in practice this may not be the case. A natural extension of this work would be in the area of bicriteria optimization with probabilistic processing times following exponential, gamma or other distribution.

As indicated in the literature survey, very little effort has gone into the area of bicriteria optimization for the multimachine case. Future work would be aimed at the development of efficient algorithms (exact as well as heuristic) for other pairs of secondary and primary criteria. Some of them would be weighted number of tardy jobs and Tmax, total completion time and Tmax, total tardiness and Tmax, weighted completion times and Tmax, Tmax and NT, and NT and total completion time.

Since complexity considerations would restrict the application of exact algorithms to problems of larger size, it would be more appropriate for future research to aim at obtaining optimal or nearly optimal assignment of jobs to machines so that the sequencing on each machine could then be done independently. Perhaps, a lower bound to the optimal values in these cases could be obtained by considering the optimal value in a parallel processing environment.

Another interesting area for future work would be bicriteria optimization on two non uniform machines. This would be a problem of managerial interest when the allocation of jobs has to be done on two machines, one of them old and the other one new. This occurs typically in industry when the new as well as the old technologies are being used simultaneously.

Bibliography

1. Burns R.N. "Scheduling to minimize the Weighted Sum of Completion Times with Secondary Criteria" *Naval Research Logistics Quarterly*, Vol. 23 No.1 (1976), pp 125-129

2. Bulfin Robert. "Models for scheduling on a single machine with dual criteria." Presented at the ORSA/TIMS conference at St.Louis. May 87.

3. Bansal S.P : "Single Machine Scheduling to Minimize Weighted Sum of Completion Times with Secondary Criteria- A branch and Bound Approach", *European Journal of Operations Research* vol. 23, pp 177-181

4. Chand S. and Schneeberger Hans : "A Note on the Single Machine Scheduling Problem With minimum Weighted Completion Times and Maximum Allowable Tardiness" *Naval Research Logistics Quarterly* , Vol. 33 No.2 (1986),

5. Cottingham B. : Job Flowtime versus Number of Jobs Tardy : Algorithm and Heuristic Approaches, Unpublished M.S. Thesis, UCLA, Graduate School of Management 1983

6. Emmons,H : "A note on the Scheduling Problem with Dual Criteria" *Naval Research Logistics Quarterly*, Vol. 22 No.3 (1975), pp 615-616

7. Emmons, H.,"One Machine Sequencing to Minimize the weighted sum of completion times with secondary criteria" *Naval Research Logistics Quarterly*, Vol. 22 No.1 (1975), pp 585-592

8. Heck H. and Roberts,S. : "A note on the Extensions of a Result on a Scheduling with secondary Criteria" *Naval Research Logistics Quarterly*, Vol. 19 No.2 (1974), pp 403 -405

9. Miyazaki.S : "One Machine Scheduling Problem with Dual Criteria" *Journal of Operations research in Japan* , Vol 24. No.1 (1981), pp 37-50

10. Moore.J.M, :"An n job one machine sequencing algorithm for minimizing the number of late jobs" *Management Science* Vol. 15, (1968), pp 102-109

11. Nelson R.T., R.K.Sarin, and R.L.Daniels, : "Scheduling with Multiple Performance Measures : The One Machine Case" *Management Science* Vol.32, pp 464-480, 1986.

12. Posner,Marc.E, : "Minimizing weighted completion times with deadlines ", *Operations Research* , Vol.33 No.3 (1985), pp 562-574

13. Rinooy Kan, A.H.G, Machine Scheduling Problems : Classification, Complexity and Computations , Martinus Nijhoff The Hague, 1976

14. Smith,W.E : "Various Optimizers for Single Stage Production" *Naval Research Logistics Quarterly*, Vol. 3 No.1 (1956), pp 59 -66

15. Sen,T and Gupta.S.K, "A Branch and Bound Procedure to Solve Bicriteria Scheduling Problems", *IIE Transactions* Vol. 15, pp 84-88,1983

16. Shantikumar,J.G.,"Scheduling n jobs on one Machine to Minimize the Maximum Tardiness with Minimum Number Tardy", *Computers and Operations Research* , Vol. 7, pp 251-259, 1980

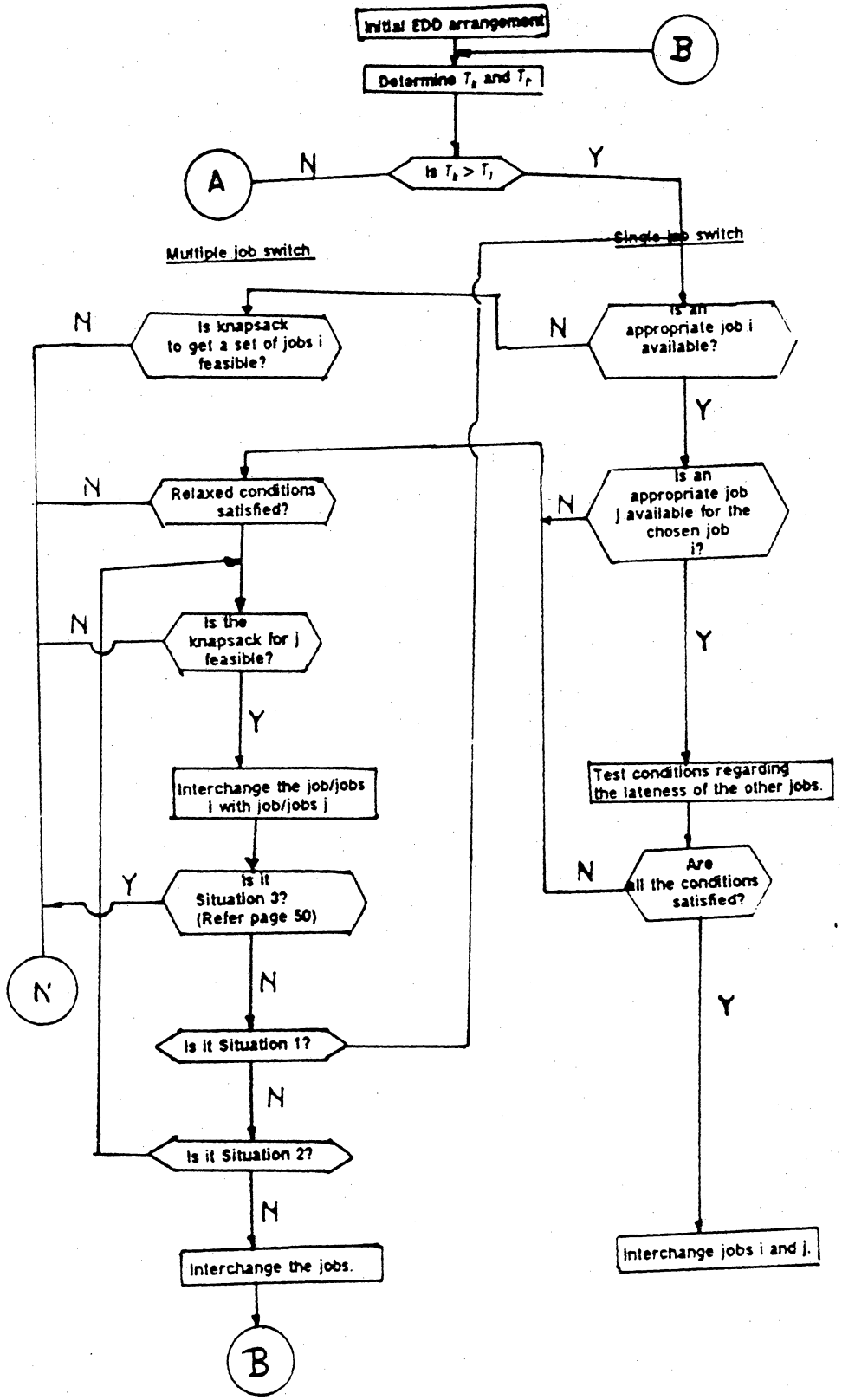
17. Van Wassenhove L.W and V.R. Baker : "A Bicriteria Approach to Time/Cost Trade-Offs in Sequencing ", *European Journal of Operations Research* vol. 11, pp 42-54, 1982

18. Van Wassenhove L.N and F.Gelders : "Solving a Bicriteria Scheduling Problem", *European Journal of Operations Research* vol. 4, pp 42-48, 1980

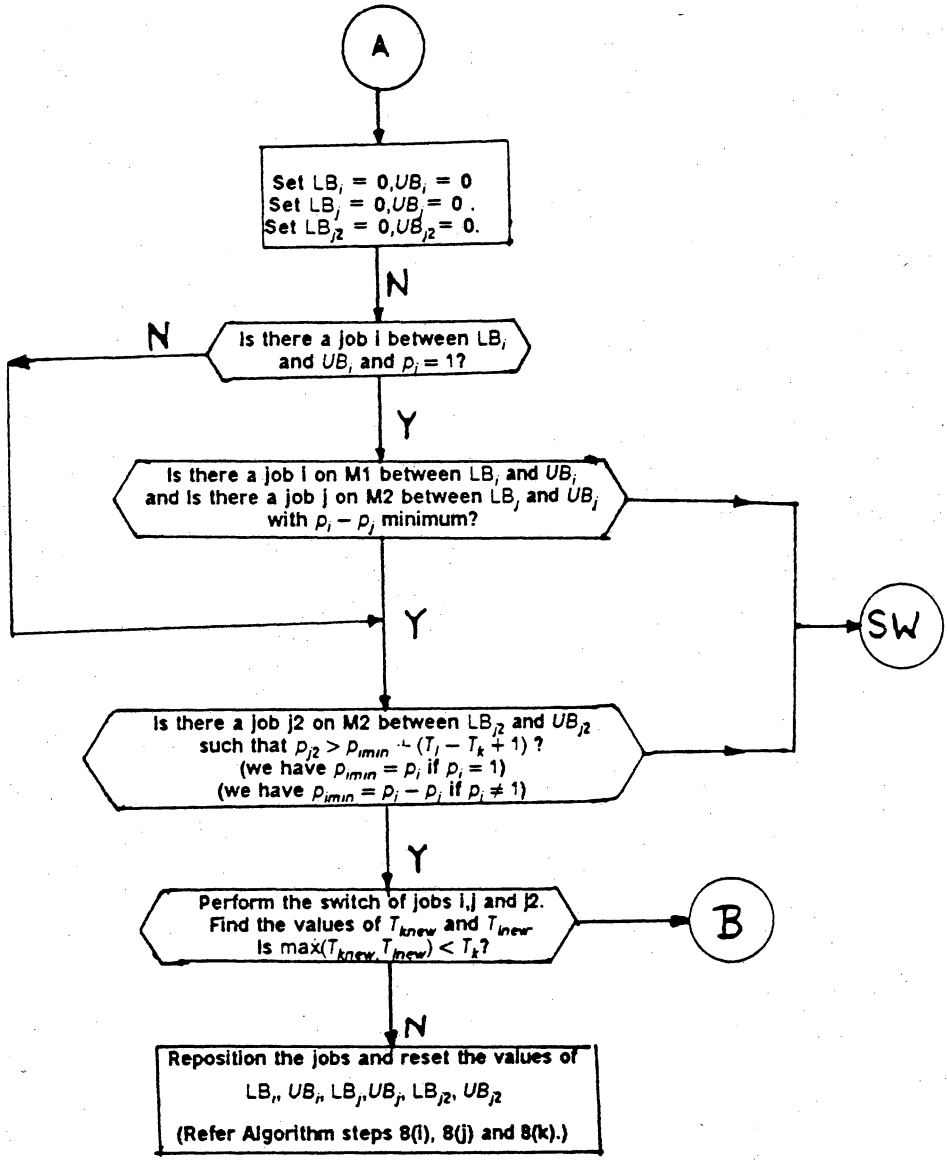
19. Van Wassenhove L.N and C.N. Potts: "An algorithm for single machine sequencing with deadlines to minimize total weighted completion times", *European Journal of Operations Research* vol. 12 (1983) pp 379-387.

Appendix A

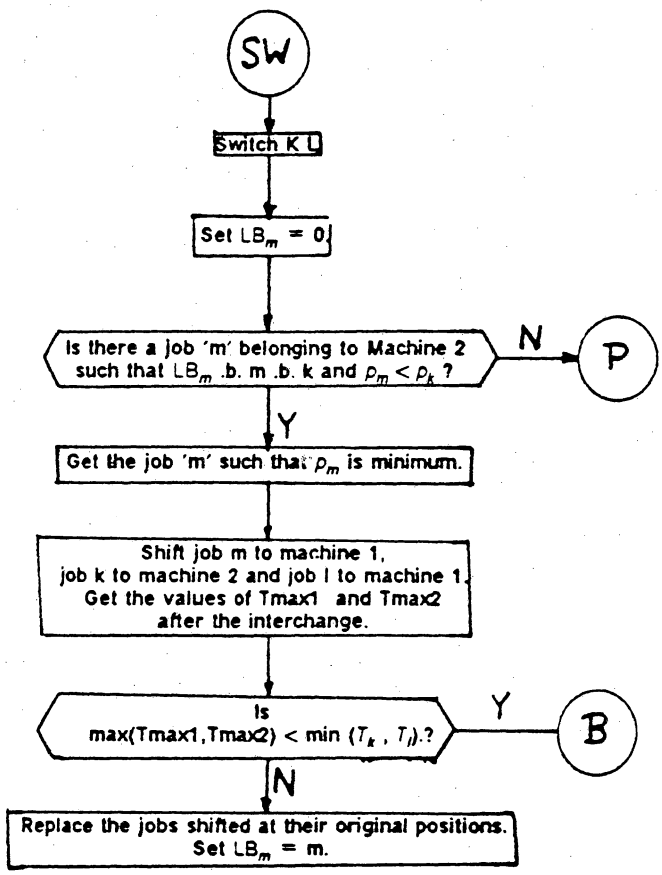
Flow Chart for Tmax Minimization Algorithm



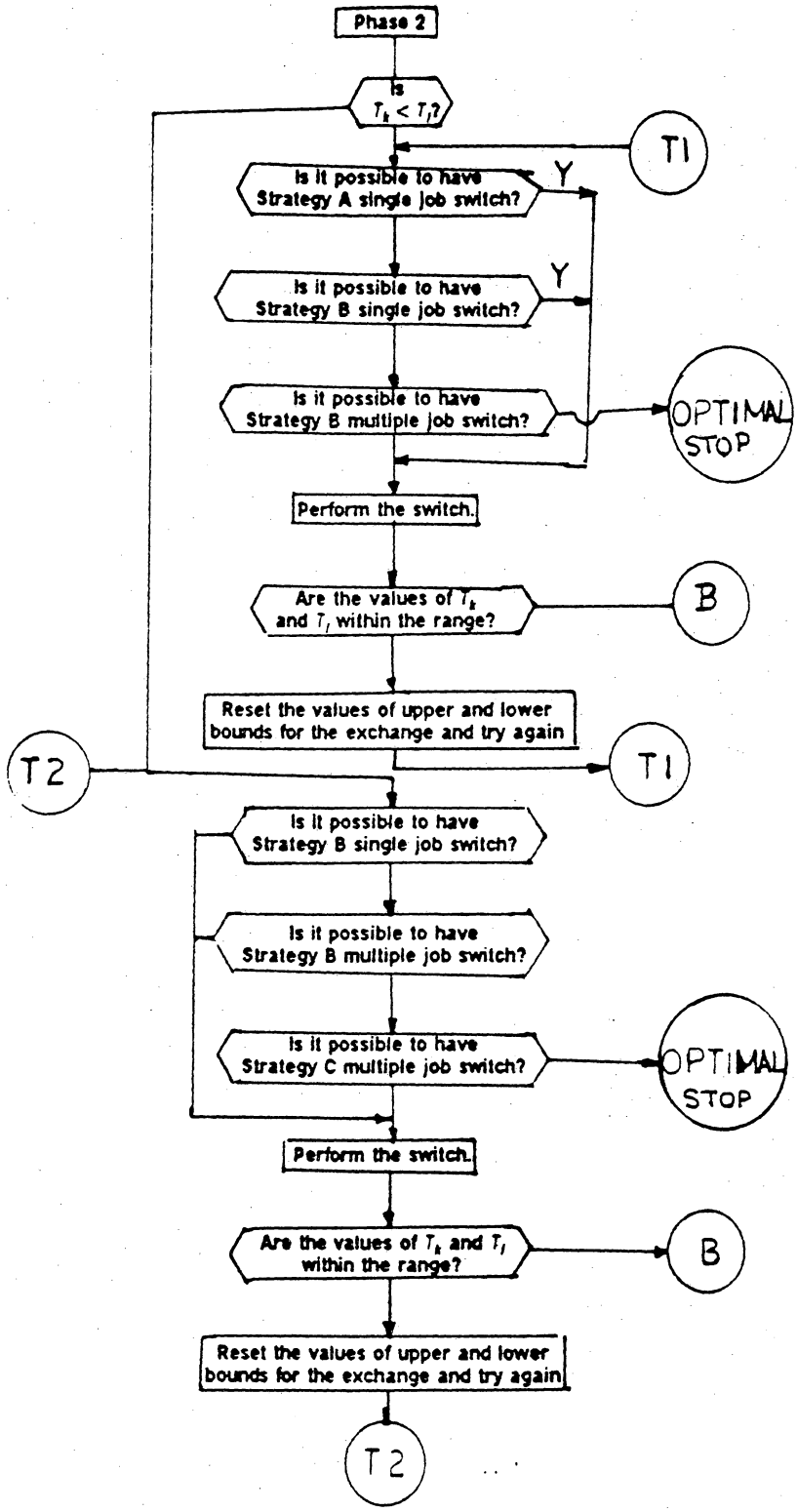
Flow Chart for Tmax Minimization Algorithm



Flow Chart for Tmax Minimization Algorithm



Flow Chart for Tmax Minimization Algorithm



Flow Chart for Tmax Minimization Algorithm

The vita has been removed from the scanned document

BICRITERIA OPTIMIZATION OF SCHEDULES ON ONE AND TWO MACHINES

by

Rema Hariharan

Subhash.C.Sarin, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

The practical applications of scheduling generally involve the optimization of more than one criterion. This thesis focusses on the bicriteria optimization problem of scheduling jobs on single and two machines. The optimization criteria that are considered are those of minimization of maximum tardiness and minimization of the total number of tardy jobs in the schedule. The former is considered as the primary criterion while the latter is considered as the secondary criterion. For the single machine problem, a search tree method is presented which is based on the implementation of some new dominance rules. Computational results presented show that the performance of this algorithm is better than that of an earlier work reported in the literature.

For the two machine problem, a heuristic algorithm is developed to minimize maximum tardiness. Computational results are presented regarding the performance of this heuristic. A search tree method is developed for the optimization of the secondary criterion. This search tree method is similar to that for the single machine problem except that it does not use the dominance rules that were developed for the single machine case. Computational experience is presented for this algorithm.