Bi-criteria Scheduling Problems on Parallel Machines

by

Divya Prakash

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 System Engineering

APPROVED:

Dr. Subhash C. Sarin

Dr. H. D. Sherali

Dr. J. E. Kobza

May, 1997

Blacksburg, Virginia

Bi-criteria Scheduling Problems on Parallel Machines

by

Divya Prakash

Subhash C. Sarin, Chairman

Department of Industrial System Engineering

(ABSTRACT)

Mathematical programming has not been used extensively for the solution of scheduling problems. Moreover, the study of bicriteria problems on single and parallel machines is an open field for research. This thesis is aimed at developing algorithms to solve bicriteria problems more efficiently and in reasonable amount of time and with little compromise on the optimality of the solutions obtained.

Two classes of problems are considered. The first class consists of scheduling unit duration tasks on parallel machines. Various combinations of primary and secondary criteria are considered and optimal seeking algorithms of polynomial time complexity are developed. The second class of problems assume general processing time tasks. An algorithm is developed for the primary criterion of total tardiness and the secondary criterion of total flow time. This algorithm is based on the solution of the underlying mathematical program and makes use of dominance relationship among the jobs and fixing of variables. Experimental results are presented regarding its performance.

Acknowledgments

I would like to thank Dr. Subhash C. Sarin for guiding me throughout the thesis work. I am grateful for the numerous ideas and suggestions that he provided and for all his encouraging words in the moments of difficulty and frustration. I also thank Dr. Sherali and Dr. Kobza for serving on my committee and for their guidance and support. I thank Dr. Jacobson for sitting in my defense in place of Dr. Sherali. I also extend my gratitude to the staff of the Department of Industrial and Systems Engineering and the Simulation and Optimization Laboratory.

I thank my parents for their constant encouragement and motivation in my pursuit for higher studies and for supporting me at every step in life. Aradhana, thank you for being by my side as steadfastly as ever and for all the helpful ideas and valuable suggestions. I am also grateful to all my friends for their company and support. I would like to express my special thanks to Sravasti and my room mates, specially Manish and Rajesh, for being great friends. And finally, I thank my brothers, Sudhanshu and Navendu, whose love and affection has seen me through all the difficult times in my life.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In real life situations, decisions to be made are often constrained by specific requirements. More importantly, these constraints are typically conflicting in nature. The decision making process gets increasingly more complicated with increment in the number of constraints. Modeling and development of solution methodologies for these scenarios have been the challenge for operations researchers from the outset. A variety of algorithms and formulations have been developed for various classes of problems. Scheduling is one such class of problems.

Scheduling problems in real life applications generally involve optimization of more than one criterion. As stated earlier, these criteria are often conflicting in nature and are quite complex. This research addresses the bicriteria scheduling problems involving identical, parallel machines.

One finds just a few references in the literature that address multicriteria scheduling problems. Most of these are related to the single machine case. Optimizing

dual performance measures on parallel processors is fairly an open area for research as well.

Two approaches can be used to address the bicriteria problems.

1. Both the criteria are optimized simultaneously by using suitable weights for the criteria, and
2. the criteria are optimized sequentially by first optimizing the primary criterion and then the secondary criterion subject to the value obtained for the primary criterion.

In this research, we will use the second approach .

1.1 Problem Definition

We consider the parallel, machines bicriteria scheduling problem in which the objective is to schedule jobs on parallel, identical machines so as to minimize a primary and a secondary criteria. We address the problem in two parts . In the first part, we consider all the jobs having identical processing times. Various combinations of primary and secondary criteria are considered from among total tardiness, maximum tardiness, number of tardy jobs and weighted flow time. As a stepping stone towards the development of the algorithm for the stage 1 problems, we first consider the single criterion, parallel identical machine, unit processing time problems and develop algorithms for the above mentioned criteria. In the second part, we generalize the processing time and develop an algorithm for the primary criterion of total tardiness and secondary criterion of the total flow time.

A practical application of this problem would be to minimize the cost of production or production time given the penalty for delaying the product. Another application would be to allocate processors among similar application programs where the processors have similar capabilities and identical jobs require the same amount of time. Since the suggested procedures and formulation can be used to generate efficient solutions, they can be useful for the managerial decision making regarding the allocation of CPUs to various application programs.

1.2 Literature Review

A survey of literature has revealed little work reported on the bicriteria scheduling problems. Most of the work done in the bicriteria problems has been on the single machine bicriteria problems. Many researchers used branch and bound type of algorithms to solve the problem while some approaches have utilized dynamic programming algorithms. Use of mathematical programming as a tool to solve the scheduling problem has hardly attracted the interests of the researchers.

Chen and Bulfin[8] studied the single machine bicriteria problems for the case of unit processing time. They proposed algorithms to generate both the efficient (nondominated) schedules as well as the schedules that optimize the bicriteria problems in a sequential fashion. They also proposed an assignment model for these problems. Table 1.1 summarizes their work for hierarchical approach to solve the bicriteria problems.

Table 1.1: Summary of the results for the unit processing time, single machine , bicriteria problem (Chen and Bulfin[8])

Primary Criteria → ——— Secondary Criteria ↓	Flowtime	Weighted flowtime	Tmax	Total tardiness	Weighted Tardiness	Number of Tardy jobs	Weighted number of tardy jobs
Flowtime	-	WSPT	EDD	EDD	O Assignment Problem	Hodgson's Algorithm	Lawler's Algorithm
Weighted flowtime	WSPT	-	O Assignment problem	Algorithm	Algorithm	Algorithm	Algorithm
Tmax	EDD	Assignment problem	-	EDD	O Assignment problem	O Assignment problem	O Assignment problem
Total tardiness	EDD	Algorithm	EDD	-	Algorithm	Algorithm	O Assignment Problem
Weighted total tardiness	O Assignment Problem	O Assignment Problem	O Assignment Problem	-	-	O Assignment Problem	O Assignment Problem
Number of tardy jobs	Hodgson's Algorithm	O Assignment Problem	O Assignment Problem	O Assignment Problem	O Assignment Problem		O Assignment Problem
Weighted number of tardy jobs	Lawler's Algorithm	O Assignment Problem	O Assignment Problem	O Assignment Problem	O Assignment Problem		

It is clear from the Table 1.1, that many bicriteria problems for the relatively simple case of single machine unit processing time case can be solved rather easily. Prabuddha et.al.[13] enriched the work of Chen and Bulfin[8] by providing some clarifications on the simultaneous minimization of the criteria under consideration.

Chen and Bulfin[9] later studied the complexity of various single machine, bicriteria problems. These different problems are results of various combinations of different criteria. The results are largely based on the behavior of the single criterion problem on single machine. That is, if a problem involving the primary criterion *c1* is NP hard, then the bicriteria problem involving *c1* as primary criterion is NP hard as well.

Most of the problems in bicriteria scheduling literature involve minimization of flowtime or weighted flowtime subject to a Tmax value. Minimization of weighted completion time subject to minimal value of Tmax was first considered by Smith who proposed an algorithm of complexity O(nlogn). Smith's algorithm for optimizing weighted flowtime is as follows(Chand and Schneeberger[7]).

Smith's Algorithm: (optimizing weighted flowtime subject to Tmax)

Step 1. Arrange the jobs in using EDD rule. If the jobs are not on time then the problem does not have a feasible solution which implies that Tmax = 0.

Step 2. Find the feasible candidates for the last position, these jobs are the jobs having due dates more than or equal to the total processing time of all the jobs.

Step 3. Find the job with the largest p/w ratio from the feasible job set and schedule it in the last position.

Step 4. Reduce the set of jobs yet to be scheduled by one by removing the job just scheduled in step 3, go to step 2 until all the jobs have been scheduled by this method.

They analyzed Smith's algorithm and modified this heuristic for solving the problem of minimizing the weighted flow time constrained by Tmax. They proved that weighted flowtime problem (w_i, p_i, d_i) is equivalent to the problem ($w_i + qp_i$, p_i, d_i). Many combinations of these parameters are identified for which Smith's algorithm will produce optimal result.

In 1980, Wassenhove and Gelders[29] suggested a pseudo-polynomial algorithm for the problem involving flow time (holding cost) and maximum tardiness criteria. They used the modified due date concept and modified Smith's algorithm to solve the problem. Original Smith's backward scheduling rule is given below:

Smith's backward scheduling rule

Step 1. Let SUM = $\sum p_i$, k=N and S=Set of all the jobs, Here p denotes processing time of job i and N is the number of jobs.

Step 2. Find a job j such that $p_j \geq p_i$, D_j, $D_i \geq SUM$, $j, i \in S$; (break ties arbitrarily). Assign the job i to position k.

Step 3. Set SUM = SUM - p_j, and remove j from the set of jobs. Set S = S - {j} and k = k - 1. If k = 0, stop; else, go to step 2.

In the modified version of the above mentioned Smith's algorithm, the ties are not broken arbitrarily, instead the job with the largest due date is chosen. The proposed algorithm is:

Wassenhove and Gelder's Algorithm:

Step 1. Put $\Delta = \sum p_i$, $i = 1, ..., N$

Step 2. Modify the due date as; $D_i = d_i + \Delta$ *for all i.*

Step 3. Solve the problem using the modified Smith's rule. If a solution exists, then it is efficient; else, go to step 5.

Step 4. Compute Tmax and put $\Delta = Tmax - 1$. *Go to step 2.*

Step 5. Stop.

The motivation of using the modified due date is to find a schedule having the minimum flow time with no late job for the modified due date case. The validation of all the steps was proved by them in their work.

Nelson et al[22] developed a branch and bound technique to solve the problem for mean flow time, number of tardy jobs and maximum tardiness. The objective of their work was to develop trade off curves to obtain efficient schedules. They suggested some algorithms which make use of dominance relations developed for various combinations of these criterion. The bounds developed were based on the Moore's algorithm (for lower bound on number of tardy jobs), EDD rule for lower bound on maximum tardiness value and SPT rule that provides a lower bound on the value of flowtime and upper bounds on the value of second criterion.

The dominance relation for the criteria of flowtime and number of tardy jobs is as follows:

a. Suppose only one of the jobs i or j can be early that minimizes flowtime subject to number of tardy jobs. If $p_i \geq p_j$ *and* $p_i - p_j \geq d_i - d_j$, *then job i need not to be considered for inclusion in early set.*

Algorithm for the Mean flowtime and NT problem

Step 1. Get the lower bound (lb) and upper bound (ub) on NT by Moore's algorithm and the SPT rule respectively. They claimed that the complete tree will have (ub - lb + 1) levels.

Step 2. Find the jobs which are not tardy in the SPT schedule and enter those on node a = 1 of the tree as early set E1.

Step 3. Expand to new nodes.

Step 4. Use the dominance relation to limit the nodes.

Step 5. Evaluate other nodes and record the efficient schedule.

Step 6. Follow the procedure of identifying the early jobs at any node, say n and move them to a set En. Expand the nodes. Use the dominance rules to eliminate the nodes. This procedure is similar to followed from step 2 to step 5. It is repeated till the termination condition is reached

Algorithm for the Mean flowtime and NT problem

Step 1. Get upper and lower bounds on Tmax and number of tardy jobs by EDD sequence and Moore's algorithm, respectively.

Step 2. Determine the set of eligible jobs for the position under consideration (starting with the last location).

Step 3. Allocate a job to the current position, if it is not late, move to next position and return to step 2; else, go to step 4.

Step 4. If all eligible jobs are tardy then they must constitute a alternate solution. The nodes are also reduced by using the dominance rule which states that job i has priority over job j, if $p_i \geq p_j$, $d_i \geq d_j$ *and* $p_i - p_j \geq d_i - d_j$.

They proposed an algorithm similar to the Wassenhove and Gelder's algorithm for the flowtime and Tmax problem. It uses the concept of modified due dates but does not use the modified Smith's algorithm as used by Wassenhove and Gelder. It uses Smith's algorithm to optimize the solution with modified due dates.

Nelson et al [22] extended their work to optimizing these three criteria simultaneously.

Daniels[12] used preference information on an interactive basis to solve the problem of optimizing the multiple criteria. He suggested that, along with providing a preferred schedule, preference information can be exploited to improve the computational effort to solve the problem. Dominance rules have been used based on the preference of the performance measures to develop the solution methodology to solve the problem. He, in general, used the dominance rules and solution approach as it has been used in Nelson et al[22].

Recently, in 1995, Vairaktarakis and Chung Yee[27] proposed a branch and bound algorithm to minimize total tardiness subject to minimum number of tardy jobs. The basic steps involved in their approach make use of dominance properties and a branching rule to make the solution procedure more efficient. These properties are summarized below;

Dominance Property 1: Job k dominates job j if there exists an optimal schedule such that if $j \in Tset \Rightarrow k \in Tset$*, where Tset is set of tardy jobs.*

The above property implies that if job *j* is tardy then job *k* must also be tardy.

Dominance Property 2: Let j, k be the jobs such that $d_k \geq d_j$, $p_k \geq p_j$ *and* $p_k - p_j \geq d_k - d_j$ *then job k dominates job j.*

Dominance Property 3: If D is the due date of the first tardy job, when the jobs are arranged in EDD, then for any two jobs j and k with $D \geq d_j$ *and* $D \geq d_k$, *if* $p_k \geq p_j$ *and* $d_k \geq d_j$, *then k dominates j.*

They used Moore's algorithm and Sidney's algorithm to find the optimal value of the given number of tardy jobs and set of tardy jobs *Tset*. These jobs are then arranged backwardly to minimize the tardiness for this set of the jobs. The schedule obtained is thus used for completion time t_i of the jobs in set *Tset*. Then the lower bound on the value of tardiness is obtained as follows:

Lower bound: $\sum (t_i - M_i) \; \forall \; i \in Tset$, where M_i is decision point of the ith iteration.

The basic branch and bound approach involves the dominance properties and bounds developed. The basic steps are

Step 1. obtain the partial tardy and nontardy set of jobs.

Step 2. Get the lower bound on the partial tardy set by using the above mentioned expression.

Step 3. Use the largest job which is not declared as tardy or non tardy job to branch to the next level.

Lin [21] used a dynamic programming approach to optimize total tardiness and flow time of jobs on a single machine. He developed theoretical results regarding dominance among jobs and later incorporated them in the dynamic programming

formulation. Both the criteria are optimized simultaneously and consequently, Emmon's dominance rules for total tardiness are not applicable.

If we let S be the collection of all feasible schedules and $h^i(s)$ be the *i*th criterion of schedule s, s S, *i* = 1, ..., *p*, then $H(s) = (h^1(s), h^2(s), ...,h^p(s))$: $S \rightarrow R^p$ is the criteria vector; for example, $h^1(s)$ is total tardiness $h^2(s)$ is total flow time, etc.

After proving the separability, monotonicity and nondominance boundedness of the criteria vector function, the recursive equation developed by Lin[21] is given below:

Let *J* be a subset of the *n* jobs scheduled last in the sequence, and let *J'* be the complement of set *J*. At the *k*th stage,

$$|J| = k$$

Let E(J) denote the earliest start time of the jobs in J, i.e.,

$$E(J) = \Sigma\, p_i \qquad \text{where } i \in J'$$

Let H(i|*J*) denote the criteria vector for job *i*, where *i* is the first job of *J*, [tardiness, flow time]. Let Z(J|i) denote the set of nondominated schedule of job set J, with i as the first job. Let denote the vector operator (+ +). The recurrence relation is

Z(J|i) = {H(i|J) a|a N (J - {i})}; where N(J) is the non dominated subset of {Z(J|i)|i J}.

Azizogulu[1] used a branch and bound method to solve the total earliness and total tardiness problem for the single machine problem. They also developed dominance rules for use in the proposed procedure. The lower bounds are developed using shortest

processing time rule, longest processing time rule and earliest due date rules. Dileepan and Sen[14] suggested a branch and bound technique to solve the problem of optimizing total flowtime and sum squared of lateness for the single machine case.

There has not been much work reported in the case of parallel machine bicriteria problems. In 1977, Garey and Johnson[15] proposed an algorithm for the two independent identical processors, bicriteria scheduling problem. The criterion considered were the start times and the dead lines. This algorithm did not give the solutions to all the problems, that is, the jobs did not meet start times and dead lines for some problems. Later, Bratley et al[5] used the above two criterion as constraint to give an optimal completion time for a similar problem.

Tabucanon and Cenna[6] in 1991 suggested a method to optimize mean flow time and maximum tardiness using a goal programming approach i.e. by assigning weights to both the criteria. They generate efficient schedules based on Wassenhove and Gelders[29] algorithm to be used as the dispatching rules and used simulation as an approach to solve the problem.

Sarin and Hariharan[25] proposed a heuristic to optimize number of tardy jobs under the constraint of maximum tardiness. The basic steps involved in the optimization of primary criterion are as follows.;

Step 1. Arrange all the jobs in EDD.

Step 2. Assign the jobs to the machines at the earliest available position.

Step 3. Get the maximum tardiness on machine 1, Tmax1 and on machine 2, Tmax2.

Step 4. Use interchange of jobs to restrict the range or relocate the range. This interchange is based on various switch conditions that are reported in the paper. If no interchange is possible then, Tmax = max(Tmax1, Tmax2).

They employed the search tree approach along with the dominance rules to optimize the secondary criterion, .i.e., number of tardy jobs.

Cheng[13] developed a solution methodology for minimizing the flow time and missed due dates for the single machine problem. Cheng[12] also proposed a better solution procedure by using penalty functions for job completion times and total flow time. Ramesh and Cary[23] suggested scheduling approach for efficient job shop scheduling under the criteria of total number of tardy jobs, mean tardiness and average completion time for the purpose of there study. They considered to was stochastic data for the purpose of study of these problems.

Recently, Grabot et al[16] have used an approach based on fuzzy sets to suggest dispatching rules similar to SPT for multi-criteria problems.

From practical point of view, Raymond[4] used a branch and bound approach to solve the problem for steel plants that involves a single machine bicriteria (process based criterion) problem. Pickens and Hoff[23] used Fuzzy goal programming approach for forest harvest scheduling.

It is clear from the literature review that a limited amount of work has been done in the area of multi-criteria scheduling. The majority of the research work has been focused in the area of the single machine bicriteria problems. Parallel machine bicriteria problems have not gained much attention.

Literature survey has also revealed that techniques used to develop algorithms for these problems are based on:

Branch and bound, e.g., Nelson et.al.[22]

Dynamic Programming, e.g., Lin K. S.[21]

Algorithms and switch conditions, e.g., Sarin and Hariharan[25]

In general, mathematical programming based approaches have found limited applicability due to the combinatorial nature of the problem. The key features of any effective algorithm involve the use of dominance relationships and tight bound.

1.3 Thesis Outline

In the sequel, Chapter 2 presents the algorithms and procedures for the scheduling of jobs having unit processing time on parallel processors. Both single and multi-criteria problems are considered. Also, a formulation is developed for all possible combinations of primary and secondary criteria. In Chapter 3, a formulation for the general processing time problems is developed involving different criteria, and one combination of primary and secondary criteria is analyzed in detail. The criteria considered for this case are total tardiness as primary criterion and flow time as secondary criterion. A procedure is developed to solve the underlying mathematical program. It is applied to a number of problems to test its performance. Results regarding the CPU time required and the accuracy of the final solution obtained are presented in Chapter 4. Conclusions and suggestions for future work are presented in Chapter 5.

Chapter 2

Algorithms For Parallel Machines, Unit Processing Bicriteria Problems

In this chapter, we present algorithms for several parallel machines, unit processing time bicriteria problems. Mathematical formulations of these problems are first developed and they result in straightforward 0-1 integer programs. Consequently, they can be solved using a solver like CPLEX. However, solution times can become prohibitive for these problems with increase in their size. We develop algorithms for these problems that are of polynomial complexity. As a part of the proposed approach, first algorithms are developed for the parallel machines single criterion problems. This is followed by presentation of algorithms for the bicriteria problems. The various criteria that are studied include flow time, weighted flow time, total tardiness, maximum tardiness and number of tardy jobs. Various combinations of these criteria are considered as primary and secondary criteria.

Before proceeding with the analysis of each case, first, we make the following assumptions:

1. The jobs are available at time zero.
2. The jobs are independent of each other.
3. No preemption is allowed.
4. The machines are identical in all respects and are available all the time.

The following notation will be used throughout:

i	designate the job, where $i = 1,2,3, ..., N$
d_i	*due date* of *job i.*
t_i	*completion time* of *job i.*
T_i	*tardiness* of *job i* which is non negative lateness i.e. $max(t_i - d_i, 0)$
NT	number of the tardy jobs
Tmax	maximum job tardiness
N	total numbers of jobs to be scheduled,
j	location of job i on machine k, where $j = 1, 2, 3, ..., N$
N	total number of positions,
k	machine on which job i is assigned at position j,
M	total number of machines available,
w_i	weight of job i,
x_{ijk}	$= 1$; if job i is located at position j on machine k
	$= 0$; otherwise

Next, we present the mathematical formulations for both, single criterion problems and bicriteria problems.

2.1 Mathematical Formulation

Formulations for the single criterion problems are presented first. These are then extended to the bicriteria problems.

Criterion: Flow time

$$\min z = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} j x_{ijk}$$

subject to:

$$\sum_{j=1}^{N} \sum_{k=1}^{M} x_{ijk} = 1 \qquad i \qquad (2.1)$$

$$\sum_{i=1}^{N} x_{ijk} \leq 1 \qquad j, k \qquad (2.2)$$

x_{ijk} binary $\qquad i, j, k \qquad (2.3)$

As every job is assumed to have a unit processing time, and all the jobs are available at time zero, the location number multiplied by the location variable for each job gives the completion time or flow time for that job. The constraint set (2.1) captures the fact that each job has been allocated to one and only one location on any of the machines. Constraint set (2.2) permits allocation of at most one job to each location.

Criterion: Weighted Flow Time

For this case, the formulation is the same as above except that the objective function is multiplied by a weight w_i that is associated with job i. That is,

$$\min z = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} w_i x_{ijk}$$

subject to: constraint sets 2.1, 2.2, and 2.3 respectively.

Criterion: Total Tardiness

Tardiness is given by $max(0, C_i - d_i)$, where C_i and d_i are the completion time and due date of job i, respectively. This function, clearly, is a non-linear function. Let the tardiness be represented by T_i. Then, the formulation is as follows.

$$\min z = \sum_{i=1}^{N} T_i$$

subject to:

Constraint set 2.1, 2.2 and 2.3, and

$T_i \geq C_i - d_i$ i (2.4)

The constraint sets (2.1) and (2.2) are as described before. The constraint set (2.4) along with the non-negativity constraints ensures that the tardiness is obtained as per the non-linear maximum function described earlier. For optimizing the total weighted tardiness, the objective function needs to be changed to,

$$\min z = \sum_{i=1}^{N} w_i T_i$$

Criterion: Maximum Tardiness

Maximum tardiness is given by $Tmax$, where $Tmax = max(T_i)$, and T_i is obtained as described earlier in this section.

The formulation is given below:

Minimize: Tmax

subject to:
Constraint set 2.1, 2.2 and 2.3

$$Tmax \geq C_i - d_i \qquad i \qquad (2.5)$$

The constraint sets (2.1) to constraint set (2.3) are the same as before while the inclusion of constraint set (2.5), along with the non-negativity constraint, ensures that maximum tardiness is obtained as per the non-linear maximum function described.

Criterion: Number of Tardy Jobs (NT)

A job is considered to be late only if its tardiness is strictly positive. Let Y_i be a binary variable that depicts whether or not the job *i* is late. It assumes a value of *1* when the job *i* is late; and *0*, otherwise. The summation of Y_i's gives the total number of tardy jobs.

The formulation is

$$\min z = \sum_{i=1}^{N} Y_i$$

subject to:

Constraint set 2.1, 2.2, 2.3 and 2.4

$$Y_i \geq MT_i \qquad i \qquad (2.6)$$

$$Y_i \text{ binary} \qquad i \qquad (2.7)$$

where M is a very large number. Constraints (2.6 and 2.7) along with the given objective function, guides the value of Y_i to be 0 or 1.

2.2 Formulations For Bicriterion Problems

Formulations for the bicriteria problems are similar to that for the single criterion problems with additional constraints requiring that the optimal value of the primary objective is not violated.

Since these formulations are essentially a repetition of those developed in the previous section, they are not repeated here. There are two parts of the formulations.

Primary objective function
subject to:
    Primary problem constraints

Secondary objective function
subject to:
    Secondary problem constraints
    primary objective function value constraint
    primary problem constraints

Consequently, the problem is solved in two steps. First, we optimize the primary criterion followed by the optimization of the secondary criterion subject to the primary objective value.

The primary and secondary criteria problems can also be solved in one step by combining the two criteria into one objective function with appropriate weights associated with each as follows:

w1*Primary criterion + w2* Secondary criterion

subject to:

Primary criterion constraints

Secondary criterion constraints

Appropriate values of w1 and w2 can be used to achieve the primary and secondary objectives. To experiment with the above formulation, we used the following combinations of w1 (weight for the primary criterion), and w2 (weight for the secondary criterion), and solved the problems using CPLEX. We solved the above integer program for NT/Total Tardiness and Total Tardiness/NT. We obtained optimal solutions for all problems with the values of w1 and w2 given in Table 2.1. It indicates that lower value of w2 might have resulted in optimal solution.

Table 2.1: Different weights used for primary and secondary criterion

w1	w2
1	1000
1	5000
1	10000

However, the solutions of linear relaxations of the formulations of the primary and secondary problems as well as of the combined problem result in fractional solutions.

Hence, they need to be solved as integer programs but these become computationally prohibitive with increase in problem size. As a result, we analyze each problem situation individually and develop an efficient algorithm for its solution. First, the parallel machines, single criterion problems are addressed. This is followed by an analysis of the bicriteria problems. The main results are presented as theorems.

2.3 Parallel Machines, Unit Processing Time, Single Criterion Problems

In this section, we present results for various unit processing time, single criterion problems on parallel machines. The criteria that are considered include Tmax, total tardiness, number of tardy jobs (NT) and weighted flowtime. First, we present results for the criteria of Tmax and Total Tardiness.

Tmax and Total Tardiness

Intuition would dictate the use of the earliest due date (EDD) order to allocate the jobs to the first available machine to solve for these due date related criteria. In fact, we show that this procedure generates the optimal schedule for the Tmax and total tardiness criteria. For future reference, we designate this EDD based allocation procedure by *Algorithm A1*. It can formally be presented as follows:

Algorithm A1 (Minimization of Tmax and Total Tardiness)
Step 1: Arrange the jobs in the increasing orders of their due dates; break the ties arbitrarily.
Step 2: Allocate the next job in sequence to the next available position on any machine.

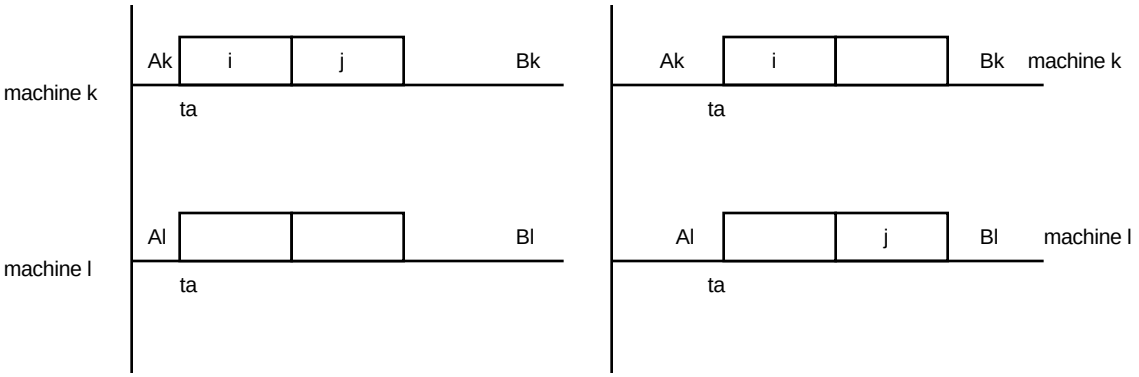
Theorem 2.1: Algorithm A1 minimizes:

a. Tmax, and

b. total tardiness.

Proof:

Two jobs *i* and *j* are called adjacent on parallel machines if they are located on adjacent positions as depicted in *Figure 2.1.* Note that, an adjacent pair of jobs may be located on the same machine, say *k*, or on different machines, say *k* and *l.* The jobs appearing on the machines before an adjacent pair are designated by *A* and those appearing after the pair are designated by *B* with an appropriate subscript for the corresponding machine.



Adjacent jobs on the same machine Adjacent jobs on different machines

Figure 2.1: Adjacent jobs on parallel machines

a. For the *Tmax* criterion

Designate by *S* the schedule obtained by using Algorithm *A1* and by S' the schedule in which adjacent jobs *i* and *j* are switched. Let t_a be the completion time of the job before job *i* on machine *k*.

In schedule S;

$T_i = \max(t_a + 1 - d_i, 0)$

$T_j = \max(t_a + 2 - d_j, 0)$; irrespective of the machine this job is on.

In schedule S';

$T_i' = \max(t_a + 2 - d_i, 0)$

$T_j' = \max(t_a + 1 - d_j, 0)$

Now, $T_i \le T_i'$ and $T_j \le T_i'$ as $d_i \le d_j$. Therefore, $\max(T_i', T_j') \ge \max(T_i, T_j)$ which implies that schedule *S* is better. QED.

b. For the *total tardiness* criterion,

$T_i + T_j = \max(t_a + 1 - d_i, 0) + \max(t_a + 2 - d_j, 0)$

$T_i' + T_j' = \max(t_a + 2 - d_i, 0) + \max(t_a + 1 - d_j, 0)$

Case I: $d_i < t_a + 1$

Then,

$T_i + T_j = t_a + 1 - d_i + \max(t_a + 2 - d_j, 0)$, and

$T_i' + T_j' = t_a + 2 - d_i + \max(t_a + 1 - d_j, 0)$

From which it clearly follows that, $T_i + T_j \le T_i' + T_j'$

Case II: $d_i > t_a + 2$

Then,

$T_i + T_j = 0$, and also $T_i' + T_j' = 0$.

Case III: $t_a + 1 \leq d_i \leq t_a + 2$

Then,

$T_i = 0$

$T_j = \max(t_a + 2 - d_j, 0)$

$T_i' = t_a + 2 - d_i$

$T_j' = \max(t_a + 1 - d_j, 0) = 0$ (as $d_j \geq d_i \geq t_a + 1$)

Therefore,

$T_i + T_j = \max(t_a + 2 - d_j, 0)$, and

$T_i' + T_j' = t_a + 2 - d_i$

Hence,

$T_i + T_j \leq T_i' + T_j'$, as $d_i \leq d_j$. QED.

Next, we consider the problem of minimizing the number of tardy jobs.

Number Of Tardy Jobs (NT)

We make the following observation.

Observation 2.1: If the jobs are allocated in the EDD order first to, say, machine *k* and then to machine *l*, and if a job *i'* allocated to machine *l* is tardy then a *job* i, *allocated* to machine *k* at an identical position to that of job *i'* on machine *l* , is also tardy (see Figure 2.2).

This follows from the fact that $d_i' \geq d_i$. As the processing times of all the jobs are equal, it indicates that, if a job is late on a machine and there are other jobs ending in the identical positions on earlier machines, then all these jobs on previous machines are also late.

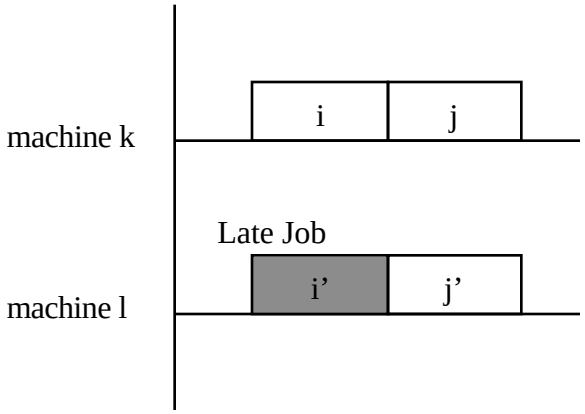


Figure 2.2: Relationship between the jobs located at identical locations on different machines

Moreover, if only one job is late then that job must appear on the first machine of the order in which the jobs are allocated to them. This is depicted in *Figure 2.3(a).*

An algorithm (called *Algorithm A2*) is presented next to minimize NT. Subsequently, we show that this algorithm generates the optimal value of NT.

Algorithm A2: (*Minimization of the number of tardy jobs*)

Step 1. Arrange all the jobs by Algorithm A1.

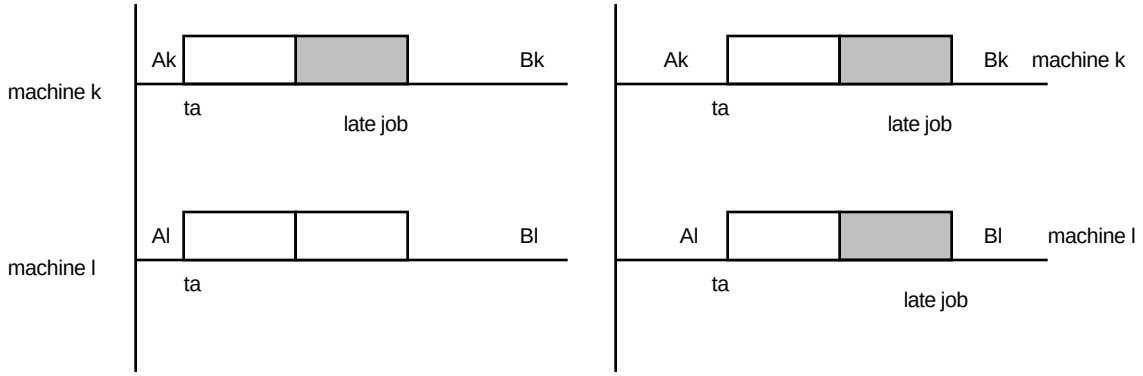
Step 2. Calculate the number of tardy jobs i.e. NT.

Step 3. If NT ≤ m and all the late jobs are located at the identical locations on all the machines (as shown in Figure 2.3(b) for m = 2) then stop; the schedule is optimal; else, go to step 4.

Step 4. Find the first late job and put it in the late set L. If no more late job exists, then stop; else go to step 5.

Step 5. Allocate all the jobs j, j∉ L, by Algorithm A1 and go to step 4.

Step 6. Arrange all the jobs of the late set in EDD order and reassign them one by one to the earliest available machine.



(a) NT = 1 (b) NT = 2

Figure 2.3: Schedules with NT = 1 and NT = 2 (and the late jobs appear at the same location number on different machines)

Theorem 2.2: Algorithm *A2* minimizes the total number of tardy jobs.

Proof: The proof is divided into two cases.

Case 1: $NT \leq m$, and all the late jobs appear at the identical location numbers on the machines (Figure 2.3(b))

Let there exist a better schedule S' which differs from the schedule obtained by *A2* in that a late job of schedule *S* occupies an earlier position. Then, some other job, with due date earlier or identical to that of the job that is moved, will occupy in schedule S' the original position of the late job that is moved. As the jobs are in EDD order, this job must also be late. In fact, by moving a late job to an earlier position, more than one job may become late. Hence, S' is not better than the schedule given by the Algorithm *A2*. Note that, the case of $NT = 1$ is covered in this part of the proof.

Case 2: *NT ≥ 2 and the late jobs do not appear on the same location on the machines*

This is a general case and is different from Case 1 in that the late jobs may appear in different locations. First, note that, if an EDD schedule results in two or more jobs late, then at least one job must be late in the optimum schedule. This can easily be shown by using arguments similar to those used in *case 1*.

Now, referring to Algorithm A2, in case NT ≥ *2*, the first late job, say *i,* is selected and put in the late set. Suppose this move is not optimal, and some other move can be made to obtain a better schedule. The other two possible moves are as follows:

In the first move, a job is moved to an earlier position. If the job, that is moved to a earlier position is currently not late, then, after the move, the new schedule either has no additional late jobs or one or more jobs may become late. On the other hand, if a late job is moved to an earlier position, then at least one job remains late in the new schedule as the initial schedule is in EDD order. Hence, irrespective of which job is moved to an earlier position, the new schedule is no better than before.

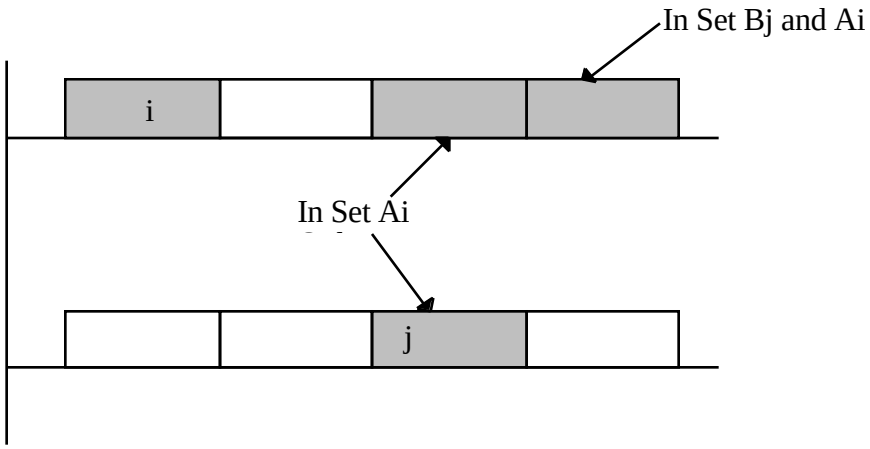


Figure 2.4: General location of tardy jobs

In the second move, a job *k*, other than the first late job, is moved to a later position. If job k is not late in the current schedule and remains early in the new position, then this move does not change NT as the jobs are in EDD order. If job k becomes late when it is delayed then, since all the processing times are the same, the same improvement can also be made by moving the first late job to the late set. Next, consider the situation in which a late job, other than the first late job, is moved to the late set. Let this job be *j*, while the first late job be *i*. Let D_i be the set of late jobs after job *i* as obtained by Algorithm A2 and accordingly D_j be the set of late jobs after job *j*. Note that $D_j \subset D_i$. Removing job j from the schedule and moving it to the late set affects the tardiness values of the jobs in D_j, since the remaining jobs are rearranged in *EDD* order according to *step 6*. On the other hand, a similar movement of job *i* to the late set affects the tardinesses of the jobs in D_i and thereby those in D_j. Thus, any improvement that can be obtained by delaying job *j* can also be obtained by delaying job *i*.

This covers all the cases. Hence, delaying the first late job *i* to the late set is at least as good as delaying any other job. QED.

Next, we consider the weighted flow time criteria. Weighted flowtime is defined to be the sum of the weighted completion times of all the jobs.

Weighted Flowtime

Theorem 2.3: Weighted Smallest Processing Time (WSPT) schedule, in which the jobs are allocated to the earliest available location on the machines in WSPT order, minimizes the weighted flow time.

Proof:

As the processing time of all the jobs is the same, arrange the jobs in the decreasing order of their weights. Refer to Figure 2.1.

Let the schedule obtained by using the WSPT rule be denoted by *S*, and *S'* be the schedule in which the jobs *i* and *j* are switched.

Let $F_i(S)$ and $F_j(S)$ be the completion times of jobs *i* and *j* in schedule *S*, respectively, and similarly $F_i'(S)$ and $F_j'(S)$ be the completion times of jobs *i* and *j* in schedule *S'*.

Then

$$F_i(S) = t_a + 1$$

$$F_j(S) = t_a + 2$$

and

$$F_i'(S) = t_a + 2$$

$$F_j'(S) = t_a + 1$$

Weighted flow time contributions of these two jobs in both the schedules are:

$$C(S) = w_iF_i(S) + w_jF_j(S)$$
$$= w_it_a + w_i + w_jt_a + 2w_j$$
$$= (w_i + w_j)t_a + w_i + 2w_j \quad (2.8)$$

$$C(S') = w_iF_i(S') + w_jF_j(S')$$
$$= w_it_a + 2w_i + w_jt_a + 1w_j$$
$$= (w_i + w_j)t_a + 2w_i + w_j \quad (2.9)$$

From (2.8) and (2.9), it is evident that $C(S') \geq C(S)$ because $w_i \geq w_j$

Therefore, the WSPT minimizes the weighted flow time. QED.

2.4 Bicriteria Problems

Next, we consider the bicriteria problems. Various combinations of the criteria are considered and analyzed as primary and secondary criteria. The approach used is similar to the sequential approach of goal programming. The primary criterion is optimized first, and then the secondary criterion is optimized constrained by the objective function value of the primary criterion. This is a two step process of which the first step is already discussed in Section 2.1. We designate the primary criterion as c1, the secondary criterion as c2, and the bicriteria objective function as c2/c1.

Total Tardiness/Tmax

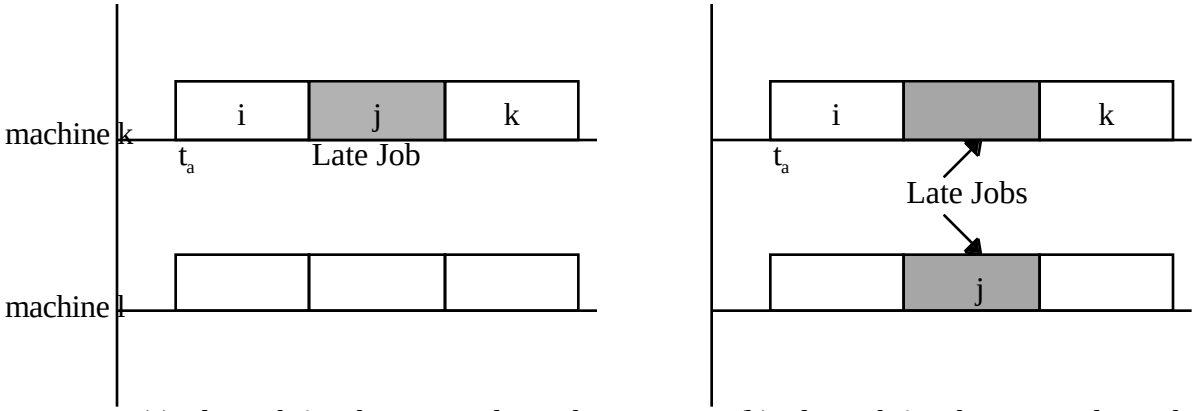
Theorem 2.4: *Algorithm A1* optimizes the Total tardiness/Tmax as well as the Tmax/Total tardiness bicriteria problems.

Proof:
This follows from the fact that Algorithm A1 optimizes both the Tmax and total tardiness criteria simultaneously (by Theorem 2.1). QED.

Next, we make an observation that is used in the subsequent discussion.

Observation 2.2: If the jobs are allocated to the machines in the EDD order and if the immediately preceding job of any late job, say *j*, is not late, then the due date of job *j* is not less than its start time and is strictly less than its completion time in the given schedule.

Proof:
Let *i* be the job preceding job *j*. Job i can be on the same machine (see *Figure 2.5(a)*) or it can be on a different machine (see *Figure 2.5(b)*). As the jobs are arranged in EDD, d_i £ d_j.
Note that, $d_i \geq t_a + 1$ as job *i* is not late, irrespective of the machine it is on.
Hence, $t_a + 1 \leq d_j < t_a + 2$, because, if $d_j \geq t_a + 2$, then the job *j* can not be late.



(a) a late job *j* with its preceding job on the same machine

(b) a late job *j* with its preceding job on different machine

Figure 2.5: The locations of the late job j and its preceding job i

NT/Tmax

First, we present Algorithm A3 for the NT/Tmax problem. Subsequently, we prove that this algorithm results in an optimal schedule.

Algorithm A3: (optimization of the number of tardy jobs given Tmax)

Step 1. Allocate the jobs using Algorithm A1. Let F be the set of jobs that are fixed at specified locations. Initially, set F = ∅.

Step 2. Determine Tmax. If Tmax ≤ 1, stop; else, go to step 3.

Step 3. Pick the first late job i, i ∉ F. If none exists, then stop; else go to step 4.

Step 4. Put job i in a location that is as late as possible without violating Tmax. Job i is now said to be fixed at that location. Break a tie using EDD with other fixed jobs. Update set F, F = F + {i}. Arrange the remaining jobs using A1 and go to step 3.

Theorem 2.5: Algorithm *A3* optimizes the NT/Tmax bicriteria problem.

Proof:

The proof is divided into two cases. Case I justifies step 2 of the algorithm while case II proves the validity of steps 3 and 4. Let the schedule obtained by *Algorithm A3* be denoted by *S*. Let *S'* be a schedule which is better than *S*. Let *Tmax* be the maximum tardiness of schedule *S* and *T'max* be the maximum tardiness of schedule *S'*.

Case I: Tmax ≤ 1

If schedule *S'* is better than *S*, then at least one of the tardy jobs of S must be early in S′. Let this job be *i*. That is, job *i* has tardiness of T_i in *S and 0* in *S'*.

Note that, 0 < Tmax ≤ 1 (by assumption). Let *j* be a job which replaces job *i* in its original position in schedule S. Job *j* must appear ahead of job *i* in schedule *S* to make T_i in S′ zero. That is, $d_j \leq d_i$. Moreover,

$$T'_j = \max(t_i - d_j, 0) > T_j = \max(t_j - d_j, 0); \text{ as } t_i > t_j.$$

Also,

$$T'_j \geq T_i = \max(t_i - d_i, 0); \text{ as } d_j \leq d_i.$$

Consequently, if $d_j \leq t_j$, then $T'_j > Tmax$, and if $d_j > t_j$, then job j becomes late, thereby, keeping NT the same. Both of these cases result in a schedule S′ that is no better than S.

Case II: Tmax > 1

First, consider the case when only one job is late and it causes Tmax > 1, or if more than one job is late then all these jobs are located at the same position number on all the machines. In this case, according to Algorithm A2, NT cannot be improved. Tmax may worsen depending upon where the late job is moved to from its current location in schedule S. Thus, the resultant schedule S′ will be no better than S.

Next, consider the case when more than one job is late and jobs are not at the same location number on all the machines. According to step 4, the first late job, say *i,* is selected and fixed at the furthest possible position, say *p,* without violating Tmax. Suppose, this move is not optimal and some other move can be made to obtain a better schedule. The new move can not be made to a position that violates the Tmax value. Hence, there are three possible cases.

In case 1, the first late job is moved to an earlier position. In that case, some job *j,* with an earlier due date and that is not a late job, occupies the original position of job *i.* Consequently, job *j* (and possibly some more jobs) become late, thereby resulting in a no better schedule. In the same vain, if any other job (whether late or early) is moved to an earlier position, then the resulting schedule cannot be better than the original schedule.

In case 2, a job *k* that is not late is delayed as much as possible to a location (designated as *l*) without violating Tmax. If no late job exists until location *l* in S′, then this move results in no better schedule. However, if there exists at least a job until location *l* that is late and is located in position *l* or earlier, the same improvement can also be obtained by moving the first of these late jobs to its *l* location since all the processing times are the same. This brings us to case 3.

In case 3, a late job other than the first late job is moved to its location *l.* Then, as in case 2, the same improvement can also be obtained by delaying the first late job as all the jobs (including those following any other late job that are before location *l*)move ahead by one unit. Q.E.D.

Weighted flow time/Tmax

An algorithm (Algorithm designated A4) for this case is presented next. Subsequently, we prove that it results in an optimal schedule.

Algorithm A4: (optimization of weighted flowtime given Tmax)

Step 1: Calculate the Tmax value using Algorithm A1.

Step 2: Arrange all the jobs in WSPT order and assign these jobs to the earliest available machine. Let F be a set of jobs fixed at specified locations. Initially set $F = \varnothing$.

Step 3: Pick the first late job $i \notin F$ *that violates the primary criterion. If none exists, then stop; else, go to step 4.*

Step 4: Advance the job and place it at the latest possible location without violating the Tmax value. If a job has already been assigned to that location, then use WSPT rule to break the tie. Set $F = F + \{i\}$. *Consider the jobs that are not fixed, and go to step 2.*

Theorem 2.7: Algorithm *A4* optimizes weighted flowtime/ Tmax.

Proof:

Let the schedule given by Algorithm *A4* be designated by *S.* Consider any two jobs *i* and *j* in this schedule. Let there exist a better schedule than *S* in which these two jobs are switched. Designate this schedule as *S'*. There are three possible cases:

Case I: (Jobs *i* and *j* F)

Jobs are arranged in WSPT. According to Theorem 2.3, the interchange of these jobs will not result in a better schedule. Hence, schedule *S* is at least as good as *S'*.

Case II: (Job *i* F and job *j* F)

Let T_i and T_j be the tardinesses of the jobs *i* and *j* in schedule S. Let C_j be the completion time of job *j* in schedule *S*. Also, note that $T_i \leq$ Tmax.

If the location of job *i* is earlier than that of job *j*, then after the interchange $T_i = C_j - d_i >$ Tmax (else step 4 of Algorithm A4 would have put this job in this location). It leads to the violation of the primary criterion or an increment of the weighted flow time value.

If the location of job *i* is after that of job *j*, then job j must have been ahead of job i in the WSPT sequence and switching of those two jobs will not improve weighted flow time.

Case III: (Jobs *i* and *j* F)

If job *i* occurs before job *j*, and *i* can be switched with *j* without violating primary criterion, then the resulting schedule will not be a better schedule as these jobs are in WSPT order (in accordance with step 4 of the algorithm); else jobs *i* and *j* can not be switched without violating the primary criterion. Hence schedule *S* is at least as good as *S'*. QED.

NT/Total Tardiness

An algorithm for this case is presented as Algorithm A5. The optimality of this algorithm is proved subsequently.

Algorithm A5: (optimization of number of tardy jobs/Total Tardiness)

Step 1: Arrange all the jobs using Algorithm A1. Let L be the set of late jobs in the current schedule. Initialize a set of jobs, C = ∅. It contains the jobs that cannot be switched.

Step 2: Calculate T_i, $\forall\ i \in L$. *If* $T_i < 1$, $\forall\ i \in L$, *then stop; else, go to Step 3.*

Step 3: Select the first late job i ∈ L and i ∉ C. If none exists then stop; else go to step 4.

Step 4: Check if $t_i = d_j$ *for some late job j ∈ L. If so, then exchange jobs i and j; set L = L - {j}. Else set C = C + {i}. Go to step 3, in any case.*

Theorem 2.6: Algorithm *A5* optimizes the NT/Total tardiness bicriteria function.

Proof:

To prove the optimality of Algorithm A5, we need to prove the optimality of steps 2, 3, and 4. As is shown earlier, Algorithm *A1* optimizes Total tardiness. Note that, the completion time of any job *i* is the location number of that job in the schedule. Hence, the completion time of a job can be determined by its new location after it is switched with another job.

Optimality Of Step 2:

By assumption, $T_i < 1$ i L. If there exists a better schedule, then that schedule can be obtained by moving jobs from the EDD schedule one at a time. We show that movement of any job, from EDD schedule, does not lead to a better schedule.

First note that, if a job *i* is moved to an earlier position from its position in the EDD schedule, then it delays all the intermediate jobs by at least one unit each. Hence, irrespective of whether job *i* is late or early in the EDD schedule, the values of NT and total tardiness at best remain the same and could possibly worsen from their values in the EDD schedule. Next, consider delaying a job *i.* If the job *i* is late in the EDD schedule, then every unit of time that it is delayed, its tardiness increases by a unit, while the tardiness of already late job *j* improves only by the amount T_j (< 1, by assumption). Hence, the net effect is an increment in total tardiness. However, if the job *i* is not late in the EDD schedule, then two situations may arise. If the job *i* is delayed to a position

where it is still early, then both the total tardiness and NT value remain the same. However, if it is moved to a position where it becomes late, for every unit increment in tardiness, the reduction in tardiness of already late job *j* is T_j (< 1, by assumption). Hence, the net change is an increment in total tardiness.

Optimality of Steps 3 and 4:

We prove the optimality of steps 3 and 4 by showing that the condition mentioned in these steps are the only conditions under which an improved schedule can be obtained.

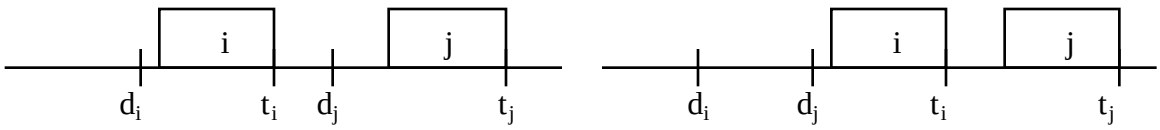
First note that, at these steps, $T_i \geq 1$ for all *i* L with strict inequality holding for at least one *i.* Also, switching either the early jobs or jobs with the same due date will not result in a better schedule. In fact, it may make the number of late jobs and total tardiness worse. Thus, consider switching a late job with a job that is either late or early.

Pick any two jobs i and j from the EDD schedule. Let job j be the late job. Consider the following cases:

Case I(a): *Job i is late and* $d_i < d_j$

Now $t_i = t_j$ or $t_i < t_j$. If $t_i = t_j$ then switching them will not improve the solution. If $t_i < t_j$ then the contribution to tardiness T before and T′ after the exchange, respectively, are:

$T = t_i - d_i + t_j - d_j$, $T' = \max(0, t_i - d_j) + t_j - d_i$



(a) (b)

Figure 2.6: Different EDD schedules when both jobs under consideration are late

Then, in case $t_i < d_j$ (see Figure 2.6 (a)), switching job i and job j will worsen the primary criterion. In case $t_i > d_j$, then switching jobs i and j does not change the total tardiness and NT values. Hence, the only case in which the primary criterion is not violated and NT improves is, if $t_i = d_j$.

Case I(b): *Job i is not late and $d_i < d_j$*

In this case, $T = t_j - d_j$, $T' = t_j - d_i$

Clearly, $T < T'$. Hence, it violates the primary criterion.

Case I(c): *Job i is not late and $d_i > d_j$*

$T = t_j - d_j$, $T' = t_i - d_j$

Once again, $T < T'$. Hence, it violates the primary criterion.

Case I(d): Analysis for the case when job *i* is late and $d_i > d_j$ is exactly the same as above, and hence, is not repeated.

Consequently, we have shown that a switch among any two jobs will worsen the EDD schedule except that made under the exchange condition stated in *Algorithm A5.* Q.E.D.

The following result follows from the above.

Corollary 2.1: Given a set of jobs initially arranged in EDD order, a late job need to be considered for being exchanged only with another late job or a job having the same due date to potentially improve the value of a secondary criterion, given the primary criterion of minimizing total tardiness.

Weighted flow time/Total tardiness

First, we propose an algorithm for this bicriteria problem and then prove its optimality:

Algorithm A6: (optimization of weighted flowtime/Total Tardiness)
Step 1: Arrange all the jobs in the EDD order with the ties broken according to the WSPT rule. Assign the jobs in this order to the earliest available machine. Let C denote a set of jobs. Initially , C = ∅. Let L be the set of all late jobs in the current schedule.
Step 2: Pick the first late job i∉ C. If none exists, then go to step 4; else, go to step 3.
Step 3: Check with the late jobs after job i. If there exists a j ∈ L, j ≠ i such that $t_i \geq d_j$ *and* $w_i < w_j$*, then exchange job i with that job j which has the maximum weight amongst all eligible jobs; update L, if necessary; else, set C = C + {i} and go to step 2 anyway.*
Step 4: Pick the first job i that is not late and i ∉ C. If none exists, then stop; else, go to step 5.
Step 5: Check with all other early jobs j after i in the schedule for $t_i \leq d_j$ *and* $t_j \leq d_i$ *and* $w_i < w_j$*. If it is true then, exchange the job i with the eligible job j that has the maximum weight amongst all eligible jobs; else, set C = C + {i}. Go to step 4 anyway.*

Theorem 2.7: Algorithm *A6* optimizes weighted flowtime/ total tardiness.

Proof:

The proof is divided into following cases:

Case I: *job j and i* $\in L$

This case corresponds to Step 3 of the algorithm. It follows directly from Corollary 2.1. If the conditions $t_i \geq d_j$, $w_i > w_j$ for *i, j* *L*, with *j* appearing after *i* in the schedule, are violated, then the primary criterion will be violated.

Case II: *jobs are early*

This corresponds to step 5 of the algorithm. This follows from the fact that the early jobs after being switched with each other should remain early because, otherwise, the tardiness of the system will increase thereby violating the primary criterion. The conditions specified in step 5 maintain feasibility.

The algorithm scans through all the late jobs first since some of the late jobs may become early while switching; then it scans through all the early jobs. There are finite number of jobs that will satisfy the given condition, and then number will keep decreasing due to increment of set C. Hence, the algorithm will stop whenever no more switches are possible. QED.

Consider next the primary criterion of NT.

Tmax or Total Tardiness/NT

The following algorithm is proposed

Algorithm A7: (*Minimization of Tmax or Total tardiness/NT*)

Step 1. Let L be the set of late jobs obtained by Algorithm A2. Arrange these jobs in EDD order. Designate the remaining jobs as E. The jobs in set E are early.

Step 2. Allocate the jobs in E, using Algorithm A1.

Step 3. Let Fj be the set of the followers of a job j on all the machines, with job j being inclusive in set Fj. Pick the first job i from the set L. If none exists, then stop; else, find the earliest job j in the schedule such that, $t_k + 1 \leq d_k \; \forall \; k \in Fj$. *If none exists, then go to step 5; else go to step 4.*

Step 4. Insert job i in the current location of job j in the schedule. Set L = L - {i}; go to step 3. If L is empty, then stop.

Step 5. Assign the jobs in set L to the machine after the jobs which have already been allocated, and stop.

Theorem 2.6: *Algorithm A7* optimizes:

(a) Tmax /NT and,

(b) Total Tardiness/NT.

Proof:

It is clear from Algorithm A2 that the jobs are in the late set have the largest possible due dates as the jobs that constitute the late set are the first late job at different iteration of Algorithm A2. Now we prove the optimality of the algorithm by showing the optimality of step 3. Let there be a schedule which is better than the schedule given by Algorithm A7 and the condition in step 3 is not satisfied. Designate such a schedule by S′. That is in S′ a job i L is inserted before job j such that for some job k Fj and $k \neq i$, $t_k + 1 > d_k$. If i is late in S′, then the primary criterion is violated. If i is early in S′ then some job m where $d_m \leq d_i$, hence improvement in Tmax or Total tardiness can be made by exchanging jobs m and i. Hence schedule S is at least as good as S′ or better. QED.

c2/weighted flow time

Lemma 2.8: If all the jobs have distinct weights, then no improvement can be made to the values of any secondary criterion.

Proof:

Let S be the schedule given by WSPT rule. Let t_i be the completion time of job *i* and t_j be the completion time of job *j*. Let S_i and S_j be the secondary criterion values of jobs *i* and *j*. Let *S'* be the schedule in which *i* and *j* are switched.

If $t_i < t_j$ then, $t'_i = t_j$ and $t_i = t'_j$. Hence, $w_i t_i + w_j t_j < w_i t'_i + w_j t'_j$, which violates primary criterion as the contribution of all other jobs remains the same. Therefore it implies that no exchange is possible which will strictly improve the value of the secondary criterion. The only possibility to improve the value of a secondary criterion is by rearranging the jobs having the same weights. QED.

Algorithm A8: (Minimization of secondary criterion/weighted flowtime)

Step 1. Arrange all the jobs in WSPT order.

Step 2. Arrange all the jobs with the same weights using an Algorithm that optimizes the given secondary criterion.

Step 3. Assign the jobs one by one to the earliest available machine.

Theorem 2.9: Algorithm A8 optimizes secondary criterion/weighted flowtime.

Proof:

The proof essentially follows from Lemma 2.8 as the jobs with different w_i's cannot be switched short of violating the primary criterion, and the jobs with the same w_i are rearranged to optimize the secondary criterion.

The summary of the results is given in Table 2.2. As can be seen, all the proposed algorithms are of polynomial time complexity. The detailed complexity analysis of the algorithms is presented in Appendix A.

Table 2.2: Summary of results for the bicriteria problems

Secondary Criterion → Primary Criterion↓	Flowtime	Tmax	Total tardiness	Number of tardy jobs	Weighted Flowtime
Flowtime	-	EDD $O(n^2)$ Algorithm A1	EDD $O(n^2)$ Algorithm A1	Algorithm A2 $O(n^3)$	WSPT $O(n^2)$
Tmax	EDD $O(n^2)$ Algorithm A1	-	EDD $O(n^2)$ Algorithm A1	Algorithm A3 $O(n^3)$	Algorithm A4 $O(n^3)$
Total tardiness	EDD $O(n^2)$ Algorithm A1	EDD $O(n^2)$ Algorithm A1	-	Algorithm A5 $O(n^3)$	Algorithm A6 $O(n^3)$
Number of Tardy jobs	Algorithm A2 $O(n^3)$	Algorithm A7 $O(n^3)$	Algorithm A7 $O(n^3)$	-	Open
Weighted flowtime	WSPT $O(n^2)$	Algorithm A8 $O(n^2)$	Algorithm A8 $O(n^2)$	Algorithm A8 $O(n^3)$	-

2.5 Single Machine Bicriteria Problems: Some Comments

The switch conditions and the algorithms developed for multi-machine case in this chapter are applicable to the case of single machine bicriteria problems as well. Problems of this class were first addressed by Chen and Bulfin[8]. As it turns out, for few combinations of primary and secondary criteria, specifically Weighted flow time/Total Tardiness and NT/Total Tardiness, our algorithms give better results than the results

obtained by Chen and Bulfin's[8] approach. Designate the approach of Chen and Bulfin as CB approach.

We present an example to corroborate this fact. Before going to the example, we present the CB approach.

CB Approach: (optimization of NT or weighted flow time/total tardiness)

Step1: Arrange all the jobs in EDD.

Step2: Arrange all the jobs with same due date in:

(a) WSPT order for weighted flowtime or,

(b) Hodgson's rule for number of tardy jobs.

Step 3: Assign the jobs one by one to the machine.

Example

The data for the example problem is tabulated in Table 2.3 for the single machine and four job bicriteria problem. The bicriteria under consideration are:

(a) weighted flow time/total tardiness

(b) NT/total tardiness

Table 2.3: Data for example problem

Job Number (i)	due dates (d_i)	weights (w_i)
1	2	4
2	2	3
3	2.5	4
4	3	7

Results obtained by using the CB approach for both bicriteria problems are tabulated in Table 2.4.

Table 2.4: Results by CB Approach

i	d_i	C_i	T_i	w_i	w_iC_i	Status Tardy or Not
1	2	1	0	4	4	n
2	2	2	0	3	6	n
3	2.5	3	0.5	4	12	y
4	3	4	1	7	28	y

The results obtained by our algorithm for the corresponding data set and bicriteria problems are listed in Table 2.5. It can be seen that there is a possibility of switch between jobs 3 and 4 that doesn't violate the primary criterion and does better on the secondary criterion for both weighted flow time and NT.

Table 2.5: Results by our Approach

i	d_i	C_i	T_i	w_i	w_iC_i	Status Tardy or Not
1	2	1	0	4	4	n
2	2	2	0	3	6	n
3	2.5	4	1.5	4	16	y
4	3	3	0	7	21	n

The values of the primary and secondary criteria are summarized in Table 2.6.

Table 2.6: Comparison of Results

Criteria	Value by CB approach	Value by our approach
Primary: (Total Tardiness)	1.5	1.5
Secondary: Weighted Flowtime	50	47
Secondary: Number of Tardy Jobs	2	1

It is clear from the comparison presented in Table 2.6 that there is a scope of improvement in the objective function value given by the CB approach because of the switching of the jobs that do not have the same due dates.

Chapter 3

Development of a Formulation and Algorithm for the Generalized Case

Most of the real-life problems involve different processing times even though the jobs may be similar in nature. With this in view, next, we generalize the job processing times and develop a procedure to solve the parallel machine bicriteria problems. The proposed algorithmic scheme is general in some sense and can exploit the nature of the criteria involved. It exploits the dominance amongst the jobs as well. Thus, it can be used for all the criteria for which the dominance rules are available.

First, this chapter delineates the complexity of the problems of this type. This is followed by the development of a formulation for the bicriteria problem involving total tardiness and flowtime, and presentation of an algorithm for its solution. The algorithm is based on the availability of an integer programming solver, which could be tailor-made for the problems of this class.

3.1 Complexity Analysis of the Parallel Machines Bicriteria Problems

The parallel machine bicriteria problems can be analyzed and categorized for their complexity and the effort required to solve them. Let c1 be the primary criterion and c2 be the secondary criterion. As indicated in the previous chapter, the bicriteria function is represented as c2/c1.

The following assumptions are valid unless specified otherwise:

1. The jobs are available at time zero.
2. The jobs are independent of each other.
3. No preemption is allowed.
4. The machines are identical in all respects and are available at all the time.
5. No machine can handle more than one job at a time.

Under these assumptions, we can state the complexity of several parallel machine bicriteria problems. The results stated by Chen and Bulfin[8] motivated the proof of the following theorem

Theorem 1: If problem of single criterion, c1, is NP hard, the scheduling problem of optimizing c2/c1 will also be NP hard.

Proof: Proof by contradiction:
Let there be a polynomial algorithm which can solve the problem of optimizing c2/c1. This implies that c1 can also be optimized in polynomial time. This is a contradiction, since c1 is NP hard. QED

Based on this result, we can state that all the problems of optimizing c2/c1 with c1 as:

1. Total tardiness,
2. Tmax,
3. number of tardy jobs,
4. weighted flowtime,
5. weighted tardiness.

are NP hard, where c2 can be any regular criterion other than the c1 for individual sets of problems. The results are summarized in *Table 3.1*. In this table, Nph stands for NP hard, and O stands for Open. Note that the problems that are open correspond to the flowtime as the primary criterion.

Table 3.1: Computation complexity of c1/c2

Secondary criterion → / Primary criterion ↓	flowtime	weighted flowtime	Tmax	Total tardiness	Weighted tardiness	number of tardy jobs
flowtime	-	-	O	O	O	O
weighted flowtime	-	-	NPh	NPh	NPh	NPh
Tmax	NPh	NPh	-	NPh	NPh	NPh
Total tardiness	NPh	NPh	NPh	-	-	NPh
Weighted tardiness	NPh	NPh	NPh	-	-	NPh
number of tardy jobs	NPh	NPh	NPh	NPh	NPh	-

3.2 Formulation of The Parallel Machines Bicriteria Problem

Development of mathematical models for the unit (or identical) processing time problem in Chapter 2 was based on the fact that the location number, in which a job is located, is its completion time. Consequently, in the formulation, it was not necessary to keep track of the jobs prior to the assigned job to get its completion time. For example, if *job 1* is located in location k, then its completion time is simply k. This is not true for the general processing time case. The general model developed can be used for single machine

scheduling problems as well, although the complexity analysis would be different in that case. The basic concept involved in model development is to capture the relative positions of various jobs to facilitate calculations of the completion times and other performance measures. We present the model formulation next. The following assumptions are made.

1. All the machines are identical.
2. Every machine has the same number of available locations.
3. The maximum number of locations on each machine is equal to the number of jobs to be allocated.

3.3 Definition of Variables

The variables used to formulate the problem are location variables and those needed to calculate performance measures. The location variables indicate the machine and location number in which a job is located. These locations can be of variable length. These also indicate the relative positions of the jobs, i.e., which job is earlier. To that end, let

N be the total number of jobs to be assigned,

M be the total number of machines available,

L be the number of locations, equal to the number of jobs, on each machine,

i represent the job number, where i = 1, 2, 3,..., N

j represent the job number, where j = 1, 2, 3,..., N

k represent the machine number, where k = 1, 2, 3,..., M

l represent the location number, where l = 1, 2, 3,..., L or N

X_{ilk} = 1, if the job i is located in location l on machine k;

= 0 otherwise,

Y_{ijk} = 1, if job i is before job j on machine k;

= 0, otherwise,

C_j represent the completion time of job j,

T_j represent the tardiness of job j.

A formulation of the problem is presented in the next two sections. The first section provides a formulation for the primary criterion while the subsequent section contains a formulation for the secondary criterion. The primary criterion considered is total tardiness while the secondary criterion is flowtime.

3.4 Formulation for the Primary Criterion of Total Tardiness

Total tardiness is one of the difficult performance measures to be optimized. As described in the previous chapter, the tardiness function is nonlinear but it can be linearized.

Formulation I:

minimize:

$$\sum_{j=1}^{N} T_j$$

subject to:

$$\sum_{l=1}^{L} \sum_{k=1}^{M} X_{ilk} = 1 \qquad i,\ i = 1,2,3,...,N \qquad (3.1)$$

$$\sum_{i=1}^{N} X_{ilk} \leq 1 \qquad l, k,\ l = 1,2,3,...,N \quad k = 1,2,3,...,M \qquad (3.2)$$

$$\sum_{l'=l+1}^{M} X_{jl'k} + X_{ilk} - Y_{ijk} \leq 1 \qquad l, k, i, j, j \neq i, l \neq L \quad (3.3)$$

$$C_j - \sum_{k=1}^{M} \sum_{i=1}^{N} Y_{ijk} \cdot p_i = p_j \qquad j, j = 1,2,3,...,N \quad (3.4)$$

$$T_j \geq C_j - d_j \qquad j, j = 1,2,3,...,N \quad (3.5)$$

$$C_j \geq 0 \qquad j, j = 1,2,3,...,N \quad (3.6)$$

$$T_j \geq 0 \qquad j, j = 1,2,3,...,N \quad (3.7)$$

X_{ilk} and Y_{ijk} are binary (3.8)

where, p_j is the processing time of job *j* and d_j is the due date of job *j*.

The constraint set (3.1) represents the job location constraint and ensures that each job is allocated to a position. The constraint set (3.2) ensures that *every location* on each processor *cannot have more than one job.* The constraint set (3.3) ascertains consistency of the relative positions of the jobs by linking the X_{ilk} variables with the Y_{ijk} variables. That is, if a job *i* is located in, say, location *l* on machine *k,* and if job *i* is before job *j* on the same machine, then job *j* can only be allocated to a location after location *l* on that machine. The completion time of a job is the summation of the processing times of the jobs located prior to that job plus its processing time. This is expressed by using Y_{ijk} variables in constraint set (3.4). Tardiness is non-negative lateness i.e. for any job *j,* $T_j = max(0, C_j - d_j)$. Constraint set (3.5) along with the non-negativity constraints, 3.6, and 3.7, determine the tardiness value for each task and linearize T_j.

One of the keys to solving the integer program efficiently is a tight formulation. The above formulation is not very tight as the linear relaxation of this formulation results

in most of Y_{ijk}*'s* being fractional. Thus, this formulation was enriched by incorporating additional information on flowtime. It is well known that the SPT ordering of the jobs and their allocation one by one to the earliest available machine optimizes flowtime under the assumptions considered. Consequently, the flowtime value obtained by the SPT based schedule can be used as a lower bound on flowtime for any other criterion.

Also, any feasible sequence will provide an upper bound on the value of the criterion under consideration. We use EDD ordering to get an upper bound on the total tardiness value. This upper bound will limit the search space. Consequently, formulation-I can be modified with this information to obtain formulation II.

Formulation II:

minimize:

$$\sum_{j=1}^{N} T_j$$

subject to:

Constraint set (3.1) through (3.8) of the formulation 1 and

$$\sum_{j=1}^{N} C_j \geq \text{lbf} \qquad (3.9)$$

$$\sum_{j=1}^{N} T_j \leq \text{ubt} \qquad (3.10)$$

where, *lbf* is lower bound on flowtime and *ubt* is an upper bound on tardiness value.

Constraint sets (3.1) to (3.8) are the same as before. Constraint (3.9) is the lower bound constraint corresponding to flowtime and constraint (3.10) is an upper bound constraint.

3.5 Formulation for the Secondary Criteria of Flowtime

Once the primary criterion is optimized, we have to optimize the secondary criterion. The formulation for the secondary criterion has Constraints 3.1 through 3.9 of the primary criterion formulation. The upper bound constraint on the primary criterion value is replaced with the equality constraint since the primary criterion is not to be violated. Furthermore, the solution for primary criterion gives a flowtime value which can be used as an upper bound on the value of secondary criterion.

minimize:

$$\sum_{j=1}^{N} C_j$$

subject to:
Constraint set (3.1) through (3.9) of formulation II

$$\sum_{j=1}^{N} T_j = z \qquad (3.11)$$

$$\sum_{j=1}^{N} C_j \leq \text{ubf} \qquad (3.12)$$

where, *z* is the primary criterion value, and *ubf* is a feasible flowtime value.

The explanation of all the constraint sets is the same as before except for the constraint (3.11) which represents the primary criterion value. In addition, constraint (3.12) specifies an upper bound on the secondary criterion value.

3.6 Development of an Algorithm

The main issue while solving an integer program is the complexity of computations. The more difficult the problem, the more time it requires to solve it. Some possible ways to reduce the computation time is to judiciously fix the variables, develop strong cutting planes and/or use tight bounds. Some inherent characteristics of the problem on hand can be used to develop one or more of these.

The main focus of our algorithm is to narrow down the solution space and still obtain almost optimal solution. Thus, the formulation is preprocessed by exploiting the structure of the problem. Once preprocessed, the algorithm uses the regular CPLEX Mixed Integer Programming solver to solve the problem. The following sections present the development of such an algorithm.

3.7 Motivation for Developing the Algorithm

The basic algorithm involves the following four steps:

Algorithm A10: (Basic steps of the algorithm)
Step 1: Use dominance relationships among the jobs to fix the variables.
Step 2: Solve the problem as a linear program.
Step 3: Check if new variables can be fixed. If yes, go to step 2; else, go to step 4.
Step 4: Solve the resulting problem using integer programming solver.

This algorithm essentially fixes the values of some variables at the upper or lower bounds. This is explained next. The performance of this scheme will be demonstated in chapter 4.

Dominance Rule

For the tardiness criterion, Emmons[30] developed dominance rules among the jobs for the single machine problem. The dominance relationships so generated can be used as such for the parallel machine case. However, to make them more effective, we propose a modification.

First, if we let;

1. A_i be the set of jobs that have been shown to follow job i in an optimal sequence for single machine
2. A'_i be the complement of set A_i
3. B_i be the set of jobs that have been shown to precede job i in an optimal sequence for single machine.

and also let, *i b j* represent the following relation between two jobs *i* and *j*:
i b j if (1.) $p_i < p_j$ or (2.) $p_i = p_j$ and $d_i \leq d_j$.

Then, the dominance relationships among the jobs as developed by Emmons[30] can be summarized as follows.

Property 3.1:

In the tardiness problem, if i b j and

$$d_i \leq \max \left\{ d_j,\ t_j + \sum_{k \in B_j} t_k \right\}$$

then there exists an optimal schedule in which $i \in B_j$ and $j \in A_i$.

Property 3.2

In the tardiness problem, if i b j and

1. $d_i > \max\left\{d_j,\ t_j + \sum_{k \in B_j} t_k\right\}$ and

2. $d_i + t_i \geq \sum_{k \in A_j} t_k$

then there exists an optimal sequence in which $j \in B_i$ and $i \in A_j$.

The dominance relationships among the jobs generated by the above properties can be summarized in a precedence matrix. This matrix provides information for the relative positioning of the jobs that is represented in the formulation by the variables Y_{ijk}. Thus, the information in the matrix can be used to fix the values of some of these variables. Consequently, some of the X_{ilk} variables can also be fixed. For instance, if a job is ahead of all the jobs; then it is fixed in the first location on one of the machines and the X_{ilk} and Y_{ijk} variables are adjusted accordingly.

The above dominance is developed with reference to a single machine. However, for the parallel machine problem, there are several locations available at each position. Consequently, to fill these positions the same procedures can be applied repeatedly by deleting dominating jobs from further consideration. As dominating jobs are deleted from further consideration, dominance relationship among the remaining jobs can change. A procedure to determine dominance among jobs that is based on this concept is presented next. The term, *the most dominant job,* that is used in this procedure, is defined to be a job which precedes the maximum number of jobs as a result of the use of Emmon's dominance rule.

Modified Dominance Procedure:

Step 1: Let the given set of jobs be denoted by O. Determine the precedence among the jobs in O using the Emmon's[] dominance rule.

Step 2: Determine the most dominant job. Remove it from O.

Step 3: *Store the relationship of the most dominant job with the other jobs from the precedence matrix in a new matrix, say M. If |O| containing the remaining jobs is 1, then stop; else, go to step 4.*

Step 4: *Use the Emmons dominance rule on the reduced set of jobs O and go to step 2.*

In case there is a tie in determining the most dominant job, select the job with the lowest processing time. If the processing times are also the same, then select the job with the lowest index.

Examples 1 and 2

Next, we present two examples to illustrate the use of the modified dominance procedure and difference in the generation of dominance among the jobs compared to that generated by the Emmon's rules. The first example demonstrates the stepwise development of matrix M. The second example is taken from Baker[2] and demonstrates how dominance among the jobs changes as Emmon's rules are applied repeatedly to reduced sets of jobs. Data for the first example is given in Table 3.2.

Table 3.2: Data for example 1

Job No.	Processing time	Due date
1	2	6
2	3	4
3	4	6
4	6	10
5	8	11

After step 1 of the algorithm, the Emmon's dominance rule gives matrix D. The most dominant job in D is job 2. The row corresponding to job 2 is thus added to matrix M as shown.

$$D = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Job 2 is now removed, and Emmon's dominance is applied on the remaining jobs. From the new matrix D, the most dominant job is job 1. The row corresponding to job 2 is now added to matrix M as shown.

$$D = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad M = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Following this process, the final matrix M is as shown.

$$M = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The Y_{ijk}'s can now be fixed as follows. A "1" in row *i* and column *j* of the matrix *M* indicates that if job *i* and *j* are on the same machine, then job i will be before job j; consequently, Y_{jik} will be fixed at 0 i.e. its lower bound. In the above case, job 2 is ahead of all the jobs and hence it can be fixed at the first position of one of the machines.

Next, we show the effect that removing of jobs can have on dominance. Table 3.3 displays the data used for this example and is taken from Baker [2].

Table 3.3: Data for example 2 (from Baker[2])

Job No.	Processing time	Due date
1	32	216
2	75	53
3	80	681
4	86	288
5	93	66
6	107	33
7	109	445
8	115	716
9	136	424
10	144	424

Complete calculations are not shown, but the jobs are arranged below in accordance with the resulting precedence among them due to dominance generated by both

the procedures. Let E be the sequence obtained by Emmon's dominance and T be the sequence by modified dominance.

E = 2, 5, 1, 4, 6, 7, ?, ?, ?, 10.

The jobs in places designated as "?" cannot be fixed as a result of the dominance between jobs and are determined by some other procedure.

T = 2, 5, 6, 1, 4, 9, 10, 7, 3, 8.

In the modified procedure, the most dominant job is removed from further consideration and T gives the sequence in which these are removed. It is evident from the sequences in E and T that the dominance amongst the jobs changes as soon as jobs 2 and 5 are removed.

The above rules are developed for the primary criterion. For the secondary criterion, the following dominance rule, designated as D1, can be used.

Dominance Rule D1: (for flowtime as secondary criterion)

if($d_i = d_j$ *and* $p_i \leq p_j$) *then i b j*

This follows from the fact that if the due dates of the jobs are the same, then arranging them in SPT on any machine will result in lower values of individual tardinesses and lower flowtime values.

Next, we discuss fixing of variables to reduce computation time.

Fixing of variables using LP relaxation

Another way of reducing the computation time of integer program is to effectively fix values of some variables. Fixing of variable values reduces the search area considerably. The solution of the LP relaxation of formulation II indicates how a job is processed by various machines. Let the fraction of job *i* on machine *k* be denoted by f_{ik}. This job can then be fixed on one of the machines based on the value of f_{ik}. We can set a threshold level (e.g.; $f_{ik} > 0.5$) level such that if f_{ik} is greater than or equal to this level on a machine *k*, then job *i* is fixed on that machine. In our case, for a given machine k and a given job i, if $\Sigma x_{ilk} \geq f_{ik}$ then job i is fixed on machine k. The x_{ilk} and y_{ijk} values for other machines are set to their lower bounds. We used $f_{ik} = 1$.

The effect of fixing the values of variables to their upper or lower bound values can be demonstrated as follows.

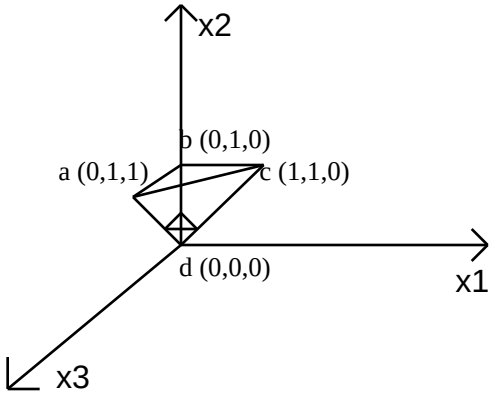


Figure 3.1: Original feasible region

Consider the original feasible region indicated in Figure 3.1. It has six feasible corner points three of which are integers. If a variable, say x_1, is fixed to zero, then the resultant solution space is indicated in Figure 3.2. Now, it has four feasible corner points,

two of them being integers. The search space is also restricted to one plane polyhedron instead of a polyhedron consisting of five planes. As a result, the effort required to solve the problem reduces considerably.

Note that the procedure of fixing the values of variables is heuristic in nature and doesn't guarantee optimality of the resulting solution.

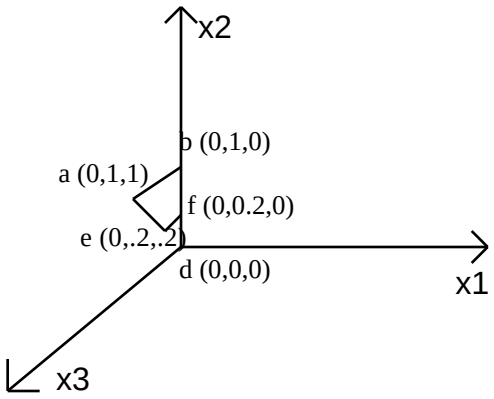


Figure 3.2: Reduced feasible region

Reduction in the number of locations

We present a procedure to reduce the possible number of locations on different machines, by way of analyzing their processing times, and subsequently, prove that it does not eliminate the optimal solution.

Let,

S be the set of all the jobs,

M_i be the makespan on machine i,

L_{max} be the maximum among the number of locations assigned on any machine

L_i be the number of locations assigned on machine i

N_i be the number of jobs on machine *i.*

N_{ij} be the number of jobs on machine *i* whose completion time is later than that of the completion time of the last job on machine *j*

Procedure P3.1: (for reducing the number of locations)

Step 1. Set $N_i = 0$, $M_i = 0$ *and* $L_i = 0 \; \forall \; i$, $i=1,2,3,...,N$

Step 2. Calculate Sum = Σp_i. *Arrange all the jobs in the decreasing order of their processing time (i.e. Using LPT rule) on machine 1. For machine 1, let* $M_1 = Sum$ *and* $L_1 = N$.

Step 3. If $M_1 - max(p_k, k \in S) \leq max(M_i, where\ i \neq 1) + max(p_k, k \in S))$ *then, set* $L_{max} = min(L_1, N - M + 1)$, *stop; else, go to Step 4.*

Step 4. Pick the first job j from machine 1 and assign it to the first available machine i, $i \neq 1$. *Break the ties amongst the machines arbitrarily. Set* $M_1 = M_1 - p_j$, $L_1 = L_1 - 1$, $M_i = M_i + p_j$, $L_i = L_i + 1$ *and* $S = S - \{j\}$. *Go to step 3.*

Theorem 3.1: Procedure P3.1 does not eliminate the optimal solution.

Proof:

Consider two machines *i* and *j.*, then for an optimal schedule $N_{ij} \leq 1$. Note that, N_{ij} is not the difference of N_i and N_j.

First we show that Machine 1 has the largest number of jobs at the end of procedure P 3.1. Let i be the job on machine 1 and j be the job on any other machine then by steps 2 and $p_i \leq p_j$ (3.13)

as the jobs are arranged in LPT order on machine 1 and the first of these jobs is always moved to another machine Also,

$$M_1 > M_k \qquad (3.14)$$

as condition in step 3 ensures that we stop before the makespan of a machine k, other than machine 1, becomes greater than or equal to M_1. We also know that if the number of jobs is less than the number of machines then, the problem is trivial; else, each machine would have at least one job. Then, N - M + 1 is an upper bound on the value of the number of locations. Hence,

$$L_{max} = \min(L_1, N - M + 1) \qquad (3.15)$$

Based on 3.13, 3.14 and 3.15, we can easily conclude that $L_1 \geq L_i$. In the worst case $p_i = p_j$ and $M_1 = M_k$, then $L_1 = L_k$.

Also note that, at the termination of the algorithm, there are two possibilities:

Possibility I: L1 > Lmax

It means that there exists at least one machine without the job (refer to 3.15). We know that in case of N jobs and M machine case there should be at least one job per machine i.e. at most any machine would have N - M + 1 locations available. Therefore it is an upper bound on Lmax.

Possibility II: L1 = Lmax

According to the algorithm, the stopping condition is $M_1 - max(p_k, k \in S) \leq max(M_i) + max(p_k, k \in S))$. It makes sure that $N_{1j} \geq 1$.

Let OL_{max} be the maximum number of jobs on any machine in an optimal schedule for any regular criterion and OM_{max} be the makespan on that machine. Let this machine be i. Now, we show that there exists an optimal schedule for a regular criterion such that:

Lmax >= OLmax

There are two possible cases:

Case I: $OL_{max} \leq L_{max}$.

It is a trivial case.

Case II: $OL_{max} > L_{max}$

It is shown already that with L_{max} position occupied with the jobs of smallest processing time on machine 1 and $N_{1j} \geq 1$. If $OL_{max} > L_{max}$, then there exists a machine j such that $N_{ij} > 1$. We can either improve the criterion by moving last job from machine i to machine j or do at least good as the existing schedule as it is going to improve the completion time of the job that has been moved. Also for $OL_{max} > L_{max}$, we can easily conclude $OM_{max} > M_1$. Hence, we can keep on moving the jobs from machine i to any machine j till the time $N_{ij} \leq 1$. QED.

Example 3

This example illustrates reduction in the number of locations as a result of Procedure P3.1. Data used in this example is given in Table 3.2. The number of machines available is *two.* The iterations are depicted in Table 3.3.

Table 3.4: Iterations of Procedure P3.1

Iteration No	Locations on Machine 1	Locations on Machine 2	Time on machine 1	Time on machine 2
1	5	0	23	0
2	4	1	15	8
3	3	2	9	14
4	3	2	9	14

As can be seen from the information obtained in Table 3.3 by implementing Procedure P3.1, the number of locations on each machine have reduced. The extent of this reduction is evident from the fact that the number of X_{ilk} variables reduce from 50 TO as a result Procedure *P3.1*.

3.8 The Algorithm

The proposed algorithm incorporates the features mentioned above. As will be demonstrated later, these features significantly reduce the computational time.

The proposed algorithm is divided into two parts. The first part solves the problem for the primary criterion. The value of the primary criterion is then incorporated as a constraint in the second part to solve the problem for the secondary criterion.

Algorithm A10: (Minimization of primary criterion with dominance only)

Part I:

Step 1: Obtain a lower bound on flowtime by using the SPT rule and add this as a constraint on the flowtime value in the formulation.

Step 2: Obtain an upper bound on the tardiness value by using the EDD rule and incorporate this as a constraint on the tardiness value in the formulation.

Step 3: Apply Emmon's dominance rule or the modified dominance rule to determine relationships amongst the jobs and fix the values of Y_{ijk} variables to their upper or lower bounds.

Step 4: Solve the problem as a relaxed linear program.

Step 5: If the LP value obtained is equal to the upper bound on the tardiness value; then exit part I and go to Part II; else go to step 6.

Step 6: Solve the problem as an integer program, and obtain the value of the primary criterion and secondary criterion. Proceed to the Part II.

Part II

Step 1: Update the formulation for the secondary criterion in accordance with the primary criterion and on upper bound on the secondary criterion value.

Step 2: Use dominance rule for this criterion and fix values of some variables.

Step 3: Solve the resulting formulation as a linear program.

Step 4: If the LP objective value is the same as the upper bound value on tardiness from step 1 then stop, the optimal solution is obtained; else, go to step 5.

Step 5: Solve the resulting problem as an integer program.

A slight modification can be made in Parts I and II of the algorithm to fix variable values on the basis of the information obtained from the LP solution. Let the new algorithm be designated as *A11*.

Algorithm A11: (Minimization of primary criterion with dominance and fixing)

Part I:

Steps 1 through 4 are the same as in Algorithm A10.

Step 4a: Check if the sum of the location variables of a job on some machine is greater than a predetermined value. If so, fix that job on the corresponding machine; otherwise, no location is specified for that job. If a job is assigned to a machine, then the position variables of that job corresponding to the other machines are set to zero. If the number of jobs fixed is not the same as in the previous run, then go to step 4; else, go to step 5.

Rest of the steps are the same as in Algorithm A10.

Part II

Step 1: Update the formulation for secondary criterion for the primary criterion and upper bound on the secondary criterion.

Step 2: Use dominance rule for this criterion and modify the bounds accordingly.

Step 3: Solve the resulting formulation as linear program.

Step 4: If the LP solution value is the same as the upper bound value on tardiness then stop, the optimal is obtained; else go to step 5.

Step 5: Check if the sum of the location variables of a job on some machine is greater than a predetermined value. If so, fix that job on the corresponding machine; otherwise, no location is specified for that job. If a job is assigned to a machine, then the position variables of that job corresponding to the other machines are set to zero. If the number of jobs fixed is not the same as in the previous run, then go to step 3; else, go to step 6.

Step 6: Solve the resulting problem as an integer program and obtain the solution.

Different variations of the proposed algorithm are possible depending on the mix of dominance among jobs, fixing of variables and reduction in the number of locations used. These variations are listed below and are investigated in the next chapter.

V1 : Emmons Dominance

V2 : Emmons dominance and fixing of the values of variables after LP relaxation

V3 : Modified Dominance

V4 : V3 and fixing of the variables after LP relaxation

V5 : V1 and Procedure P3.1

V6 : V3 and Procedure P3.1

V7 : V2 and Procedure P3.1

V8 : V4 and Procedure P3.1

Chapter 4

Computations and Results

The proposed algorithm was coded in ANSI "C" language, and was run on a Sun-Solaris Sparc-10 work-station. The results are reported for different variations of the algorithm for the primary and secondary criteria.

Different cases were generated using a random number generator according to a procedure similar to that of Emmons[]. The processing times were uniformly distributed between 1 and K, where K is an integer. Without loss of generality, all the values of processing times were generated as integers. The due dates of the jobs were obtained by adding a uniform integer variable defined over the number of jobs to the processing times, i.e., $d_i = p_i + U(0, \text{number of jobs})$. The processing times and due dates for different problems were generated according to the following relationships:

$$p_i = 1 + (K - 1)*U(0, 1)$$

$$d_i = p_i + \text{integer}(U(0, \text{Number of jobs}))$$

The details on the number of jobs and the number of machines is given in Table 4.1.

Table 4.1: Case configurations

No.	Case Numbers	No. of Machines/no. of jobs
1	1 - 11	2/5
2	12	2/7
3	13	2/10
4	14	3/10

4.1 Results for the Primary Criterion

The problems were run with the proposed algorithm using *CPLEX MIP (mixed integer programming solver).* The computation time taken to reach the solution and the objective function value obtained by the MIP solver for different variations of the algorithm under different cases considered are tabulated in *Table 4.2.*

Table 4.2: Objective values obtained by different variations of the algorithm

	Objective function values obtained with different variations of the proposed Algorithm								
Case #	V1	V2	V3	V4	V5	V6	V7	V8	Integer Program
1	11	11	11	11	11	11	11	11	11
2	14	14	13	14	13	14	13	14	13
3	10	10	10	11	10	11	10	11	10
4	9	9	9	9	9	9	9	9	9
5	10	10	10	12	10	10	10	10	10
6	9	9	9	9	9	9	9	9	9
7	10	10	10	10	10	10	10	10	10
8	8	8	8	8	8	8	8	8	8
9	4	4	4	4	4	4	4	4	4
10	8	8	8	10	8	8	8	8	8
11	11	12	11	11	11	11	11	11	11

ÜV1 : Emmons Dominance, V2 : Emmons dominance and fixing of the values of variables after LP relaxation, V3 : Modified Dominance, V4 : V3 and fixing of the variables after LP relaxation, V5 : V1 and Procedure P3.1, V6 : V3 and Procedure P3.1, V7 : V2 and Procedure P3.1, V8 : V4 and Procedure P3.1

As can be seen from the results tabulated in Table 4.2, different variations of the algorithm seem to perform fairly well when compared to the optimal solution value obtained using the IP solver on the original problem and reported in the last column of Table 4.2. Deviation from the optimal objective function value ranges between 0 and 2. There are a maximum of two instances for which variations V1, V2, V4, V6 and V8 deviate from the optimal solution

The CPU time in seconds taken by the different variations of the proposed algorithm are presented in Table 4.3. Note that all the variations of the proposed algorithm take much less time than that taken by the IP solver. In particular, variation V5 to V8 supersede in performance among the different variations tried.

Table 4.3: Time taken in seconds by different variations of the algorithm

	Computation time in seconds taken by different variations of the proposed Algorithm.								
Case #	V1	V2	V3	V4	V5	V6	V7	V8	IP
1	0.96	0.45	1.57	2.04	0.11	0.11	0.10	0.05	239.5
2	123.1	91.1	14.2	16.3	0.61	0.27	0.51	0.27	304.1
3	71.4	61.1	40.1	20.4	0.93	0.89	0.53	0.46	100.0
4	75.1	15.0	56.1	3.76	0.54	0.33	0.34	0.28	149.6
5	56.67	21.6	19.5	0.84	0.78	0.21	0.58	0.20	225.0
6	59.0	51.8	42.00	5.68	0.68	0.61	0.48	0.31	221.0
7	67.90	70.0	43.21	21.17	0.78	0.36	0.38	0.30	336.0
8	73.3	86.74	69.45	24.12	1.11	0.42	0.91	0.32	148.0
9	93.37	87.3	43.24	16.34	0.94	0.28	0.47	0.18	242.0
10	32.48	43.25	26.56	7.01	0.79	0.63	0.59	0.33	131.6
11	92.85	21.53	27.69	10.59	0.53	0.58	0.33	0.28	240.5

ÜV1 : Emmons Dominance, V2 : Emmons dominance and fixing of the values of variables after LP relaxation, V3 : Modified Dominance, V4 : V3 and fixing of the variables after LP relaxation, V5 : V1 and Procedure P3.1, V6 : V3 and Procedure P3.1, V7 : V2 and Procedure P3.1, V8 : V4 and Procedure P3.1

Variations V5 to V8 perform well both in terms of computation time and the quality of solutions obtained. To test their performance further, they were applied to cases 12-14 presented in Table 4.4. Since the optimal results for these cases are not known, no claim on optimality can be made, but based on their performance on cases 1-11, we believe them to be close to optimal. The cpu time for these cases are tabulated in Table 4.4

Table 4.4: Time taken for bigger size problems

	Computation time in seconds taken by different variations of the proposed Algorithm.		
Case #	V5	V6	V8
12	122.04	66.1	42.1
13	7200.8	2400.1	1329
14	>7200	2500.1	1700

One of the main reasons for the better performance of the proposed algorithm is the reduction in search space by judicious fixing of the variables as a result of dominance rules and information available from the LP relaxation. As described in the previous chapter, the possible impact of fixing variables can reduce the search space and thus the time taken to find the optimal solution. The fixing of the variables also seem to help in bridging the LP, IP gap. This is supported by the observations presented in Table 4.5. The cases considered here are different from those in Tables 4.2 and 4.3. All of these correspond to the 2 machine and 5 job problems. Note that the LP-IP gap decreases considerably as a result of dominance and reduction in the dimensionality of the problem as a result of setting the variables to their upper or lower bounds.

Table 4.5: IP and LP value gap for different scenarios

Case No.	Gap with no lower bound on Flow time and no Dominance	Gap with no dominance and no lower bound on Flow time	Gap with lower bound constraint and Dominance
1	11	2	0
2	14	5	3
3	6	2	0
4	13	4	2
5	11	2	1
6	10	4	4

Next, we present the results for the secondary criterion for the cases presented in Table 4.1. The computation time taken by different variations of the algorithm, namely, without dominance and fixing of the variables(W1), with dominance(W2) and reduction in locations(W3) is presented in Table 4.6. The computation times taken by the IP solver are also depicted.

Table 4.6: Computation time taken for optimizing the secondary criterion

Case #	Without dominance and fixing (W1)	With dominance (W2)	With dominance and reduction in locations (W3)	IP solver
1	7	0.61	0.15	6.19
2	0.65	1.15	0.36	0.7
3	84.86	2.10	1.71	280.79
4	10.1	0.43	0.43	0.58
5	110.1	2.57	0.98	322.36
6	38.1	0.31	1.2	3.21
7	1.0	0.33	0.48	5.38
8	2.18	0.49	0.62	3.56
9	2.62	0.38	0.47	0.39
10	32.1	0.24	0.61	0.64
11	14.1	0.68	0.96	70.11

Note that W2 and W3 are variations of Part II of algorithm A10. These are similar to variations V2 and V8 for the primary criterion. Variation W2 exploits just the dominance while W3 exploits fixing of variable values, reduction in number of open locations and dominance.

In all these cases and with all the variations, optimal solution was obtained as verified by comparing it with the solution obtained by the IP solver. The optimal values are not reported. It can be seen that the maximum computation time taken by W3 is only 1.71 seconds while the maximum computation time taken by the IP solver is 322.36 sec. Also, the minimum time taken by W3 is 0.15 seconds, while the IP solver requires 0.39 seconds. Overall, variation W3 seems to perform the best.

By comparing the results obtained for both the primary as well as the secondary criteria, the combination of variation V8 or V7 for the primary criterion and W3 for the secondary criterion give the best results in terms of quality of solution and cpu time.

Chapter 5

Conclusion and Future Work

Mathematical programming has not been used extensively for the solution of scheduling problems. This research is aimed at developing algorithms to solve some bicriteria problems more efficiently and in reasonable amount of time and with little compromise on the optimality of the solution. The criteria considered in this research are total tardiness and flowtime as primary criterion and secondary criterion, respectively, along with a class of unit processing bicriteria scheduling problems.

The bicriteria problem with the tardiness and flowtime criteria as primary and secondary criteria, respectively, for any number of machines is NP hard because the total tardiness problem for the single machine or parallel machines is NP hard. For this reason heuristic based procedure has been proposed. However, for the limited experimentation conducted, the optimal results were obtained with the proposed procedure in almost all the cases tried.

The developed algorithm can be generalized to any combination of different criteria. The only condition is the availability of some kind of effective dominance rule, e.g., if the primary criterion is flowtime, then the SPT rule can be used as the dominance rule or if the primary criterion is Tmax, then the EDD rule can be used as the dominance rule. The use of dominance rule for the secondary criterion can help in improving the performance of the algorithm. One such property is used in the proposed procedure and its effect is reflected in the results.

As referred to in the literature review, the bicriteria optimization for the multi-machine case has not been addressed much in the literature. The work presented in this thesis is the first attempt in addressing unit processing time, bicriteria, parallel machines problems. However, future work can address the criteria that are not addressed in this work such as weighted tardiness and weighted number of tardy jobs and the combinations thereof. This work forms a stepping stone for addressing the general processing time, bicriteria, parallel machine problems.

Throughout this work, processing times and due dates were assumed to be deterministic. A natural extension would be to use probabilistic processing times following a certain distribution. It might be interesting to see how well do the dominance rules hold in this scenario. Another extension would be to consider nonuniform machines.

It is also felt that investigation into the development of efficient integer programming based algorithms would be a worthwhile task.

Appendix A: Complexity Analysis of the Proposed Algorithms

Algorithm A1: It is a simple sorting routine of complexity $O(n^2)$.

Algorithm A2:

Step 1: $O(n^2)$

Step 2: $O(n)$

Step 3: $O(n)$

Step 5 will be repeated for every execution of step 4.

For each execution of step 4

step 5

End For

Complexity of this part $O(n^3)$.

Step 6: Complexity of this part at worst could be $O(n^2)$.

Overall complexity of algorithm A2 is $O(n^3)$.

Theorem 2.3: WSPT rule. Complexity of this part at worst could be $O(n^2)$.

Algorithm A3: $O(n^3)$

Step 1: $O(n^2)$.

Step 2: $O(n)$

Step 3 & Step 4: $O(n^3)$

For each job in step 3

Search for the location. O(n),

Arranging the jobs using A1 $O(n^2)$.

End For;

Algorithm A4: $O(n^3)$

Step 1: $O(n^2)$.

Step 2: $O(n^2)$

Step 3 & Step 4: $O(n^3)$

For each job in step 3

Search for the location. O(n),

Arranging all the jobs using WSPT $O(n^2)$.

End For;

Algorithm A5: $O(n^2)$

Step 1: $O(n^2)$.

Step 2: O(n)

Step 3 & Step 4: $O(n^2)$

For each job in step 3

Search for the job that satisfies the condition. O(n).

End For;

Algorithm A6: $O(n^2)$

Step 1: $O(n^2)$.

Step 2 & Step 3: $O(n^2)$

```
While (no more switches)
    For each job in step 3
        Search for the job that satisfies the condition.  O(n).
        If switch break;
    End For;
    if switch start with the new job
    else next job
end while
```

Step 4 & Step 5: $O(n^2)$

```
While (no more switches)
    if switch start with the new job
    else next job
    For each job in step 3
        Search for the job that satisfies the condition.  O(n).
        If switch break;
    End For;
end while
```

Algorithm A7: $O(n^3)$

Step 1: $O(n^2)$.

Step 2: $O(n^2)$.

Step 3 & Step 4: $O(n^3)$

```
For each i (LATE JOB)
    for each job j that follows i
        for each job after j
            check condition
        end for;
        If condition satisfied then
            fix the late job
        else
            break;
        end if;
    end for;
END FOR;
```

Step 5: $O(n^2)$

FOR all secondary criterion/WSPT rule. Complexity would be of the secondary criterion.

References

1. Azizoglu, Meral; Kondacki, Suna, Omer "Bicriteria scheduling problem involving total tardiness and total earliness penalties" *International Journal of Production Econonmics* v23 n1-3 (1991)

2. Baker K.R. *Introduction to Sequencing and Scheduling* (1974) Wiley New York

3. Bazaara S. M.; Jarvis J.J.; Sherali D. Hanif *Linear Programming and Network Flows* (1990) Wiley New York

4. Bausch Raymond "Multicriteria scheduling tool using a branch and bound algorithm" *European Journal of Operational Research* v61 n1-2 (1992) p213-218

5. Bratley, Paul; Florian, Michael and Robillard, Piere "Scheduling with the Earliest Start and Due Date Constraints on Multiple Machines" *Naval Research Logistics Quarterly* v22 n1 p165-173

6. Cenna, Ahmed Abu; Tabucanon, Mario T. "Bicriteria Scheduling Problem in a Job Shop with Parallel Processors" *International Journal of Production Economics* v25 n1-3 (1991) p95-101

7. Chand S.; Schneeberger "A note on Single Machine Scheduling Problem with Minimum Weighted Completion Time and Maximum Allowable Tardiness" *Naval Research Logisitic Quarterly* v33 n3 (1986) p551-557

8. Chen, Chuen-Lung; Bulfin, Robert L. "Complexity of single machine, multi-criteria scheduling problems" *European Journal of Operational Research* v70 (1993) p115-125

9. Cheng, T.C.E. "Improve solution procedure for n/1/max{γ(C)} → ΣC scheduling problem" *Journal of Operations Research Society* v42 n5 (1991) p413-417

10. Cheng, T.C.E. "Minimizing the flow time and missed due dates in a single machine sequencing" *Mathematical & Computer Modelling* v13 n5 (1990) p71-77

11. Daniels, Richard L."Incorporating preference information into multi-objective scheduling" *European Journal of Operational Research* v77 (1994) p272-286

12. De Prabudha; Ghosh, Jay B.; Wells, Charles E. "Some clarifications on the bicriteria scheduling of unit execution time jobs on a single machine" *Computer and Operations Research* v18 n8 (1991) p717-720

13. Dileepan P.; Sen T. "Bicriteria job shop scheduling with flowtime and sum square of lateness" *Engineering Cost and Production Economics* v21 n3 p295-299

14. Garey, M.R.; Johnson, D.S. "Two processor scheduling with start times and dead lines" *SIAM Journal of Computing* v6 (1977) p416-426

15. Grabot, B.; Geneste, L. "Dispatching rules in scheduling: a fuzzy approach" *Internatinal Journal Production Research* v32 n4 (1994) p903-915

16. Gupta, Yash P.; Evans, Gerald W; Gupta, Mahesh C. "Review of approaches to FMS scheduling problem" *International Journal of Production Economics* v22 n1 (1991) p13-31

17. Hariharan, R. "Bicriteria optimization of schedules on one and two machines" *Master's Thesis Virginia Polytechnic Institute and State University* (1988)

18. Ignizia, James P. "Generalized goal programming approach to the minimal interference multicriteria N/1 scheduling problem" I I E Transactions v16 n4 (1984)

19. Kernighan, B.W.; Ritchie, D.M.; *The C Programming Language* Prentice Hall International Inc. New Jersey

20. Lin K.S. "Hybrid algorithm for sequencing with bicriteria" *Journal of Optimization Theory and Applications* v39 (1983) p105-124

21. Nelson, R.T.; Sarin, R.K.; Daniels, R.L. "Scheduling with multiple performance measures: the one-machine case" *Management Science* v32 (1986) p464-479

22. Pickens, J.B.; Hoff, J.G. "Fuzzy goal programming in forestry, an application with special solution" *Fuzzy Sets and Systems* v39 n3 (1991) p239-246

23. Ramesh R.; Cary, J.M. "Multicriteria job shop scheduling" *Computers and Industrial Engineering* v17 (1989) p597-602

24. Sarin S.C.; Hariharan R. "A two machine bicriteria scheduling problem" *Virginia Polytechnic Institute and State University*

25. Sen T.; Raizhel, F.M.E.; Dileepan P. "Branch and bound approach to the bicriteria scheduling problem involving total flow time and range of lateness" *Management Science* v23 (1977) p1016-1019

26. Valadares, Tavares L. "Multicriteria scheduling of a railway renewal program" *European Journal of Operational Research* v25 n3 p395-405

27. Wassenhove, V.L.N.; Gelders, L.F. "Solving bicriterion scheduling problem" *European Journal of Operations Research* v4 (1980) p42-48

30. Emmons, H. " One-machine sequencing to Minimize Certain Functions of Job Tardiness," *Operations Research*, Vol. 17, No. 4 (July - August, 1969).

31. Moore, J. M. " Sequencing *n* Jobs on One M achine to Minimize the Number of tardy Jobs," *Management Science*, Vol. 17, No. 1 (September, 1968).

32. Sidney, J. B. " A Comment on a paper of Maxwell," *Management Science*, Vol. 18, No. 11 (July, 1972).

33. Smith, W. E. " Various Optimizers for Single Stage Production," *Naval Research Logistics Quartly*, Vol. 3, No. 1 (March, 1956).

VITA

Divya Prakash was born in New Delhi, India on February 12, 1967. He earned his bachelor's degree in Electrical Engineering from Delhi College of Engineering, University of Delhi, Delhi, India in July 1988. He joined Virginia Tech in Spring'1993 to pursue a Master's Degree. His career interest include Mathematical and Simulation Modeling and Analysis of real life and theoretical problems. Divya is working with Worthington Steel in Columbus, OH as Scheduling/Manufacturing Systems Engineer since August' 1995.

Divya Prakash