Common Due-Date and Priority Scheduling

by

Jeffrey E. Diamond, B.Sc. M.Sc.

A Thesis
Submitted to the Faculty of Graduate Studies
In partial fulfillment of the requirements
for the degree
Master of Sciences

Department of Actuarial and Management Sciences
University of Manitoba
Winnipeg, Manitoba

ISBN 0-315-77993-4

COMMON DUE-DATE AND PRIORITY SCHEDULING

BY

JEFFREY E. DIAMOND

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCES

Abstract

In part one we consider the scheduling of jobs onto single or parallel machines to minimize the sum or weighted sum of a combination of performance measures including job earliness and tardiness, due-date and idle-time.

Chapter 1 provides a review of research in this area. Chapter 2 proposes a simulated annealing/neural computing approach as a heuristic for minimizing the weighted sum of absolute lateness of jobs on a single machine. The performance of the algorithm proposed is found (through some computational experience) to be comparable to the better methods proposed so far when the processing times are uniformly distributed.

Chapter 3 provides approximate solutions for the problem of scheduling jobs on parallel machines to minimize the weighted sum of job earliness, tardiness, due-date and idle-time. The approximate solutions are shown to be asymptotically optimal (in non-degenerate cases) as the number of jobs tends to infinity.

Part two considers the scheduling of jobs which have priority assigned. In chapter 4 , we provide a generalized SPT rule which hierarchically minimizes the flowtimes of the priority classes.

Chapter 5 provides a polynomial dynamic programming algorithm for minimizing the total flowtime of a two-priority-class set of jobs subject to the constraint that higher priority jobs must precede lower priority jobs if they are processed on the same machine.

Acknowledgements

I would like to acknowledge my advisor, Professor T.C.E. Cheng for his help in guiding this research. I thank my wife and my family for giving me support in the difficult times.

This Manuscript is dedicated to the future and to the child on its way.

This research was supported in part by the Natural Sciences and Engineering Research Council.

Table of Contents

Figures

Tables

Introduction

Recently, much attention has been given to problems involving the scheduling of jobs about a common due-date where penalty is imposed for both early and late completion of jobs. In part one of this thesis we review this literature and propose approximate or heuristic solutions for two such problems: The first problem (chapter 2) involves a single machine while the second problem (chapter 3) involves parallel machines.

Part two concerns the scheduling of jobs on parallel machines to minimize flowtime when priorities are assigned to each job. We provide solutions for two problems. The first problem (chapter 4) requires hierarchical minimization of the flowtimes of an arbitrary number of job priority classes. The second problem (chapter 5) requires minimization of total flowtime of a two-priority-class set of jobs subject to the constraint that higher priority jobs must precede lower priority jobs when they are scheduled on the same machine.

Part One:
Common Due-Date Scheduling

Chapter 1

Common Due Date Scheduling Review

1.1 Introduction

In a job shop production system, each job is assigned a due-date for delivery before it is processed. A variety of methods exist for assigning these due-dates. Cheng and Gupta (1989) provide a review of scheduling research involving due-date determination decisions. They classify due-date determination decisions into two categories: Exogenous and Endogenous procedures.

Endogenous procedures are those in which the scheduler determines the due-date for each job based on a variety of factors such as job characteristics and the status of other jobs in the facility.

Exogenous due-dates are those in which the due-date is determined by some agency external to and independent of the job shop itself. Cheng and Gupta find that two exogenous methods of due-date determination are found in the literature: Constant (CON) and Random (RAN) flow allowance. These two rules can be expressed as follows:

CON: $d_i = r_i + k$

RAN: $d_i = r_i + e_i$.

where k is a constant, d_i and r_i represent the due-date and ready time (respectively) of the job i and e_i is a random number. If all ready times are zero, i.e. all jobs are ready for

processing at time zero then we have, for the CON rule:

$$d_i = k.$$

Thus , in this case, all jobs are assigned a common due date k. We will address this type of problem here.

Recently, much attention has been given to scheduling problems in which costs or penalties are incurred by early completion of jobs as well as late. Costs due to early completion could, for example, arise from product deterioration or inventory holding costs. Consideration of these types of costs in scheduling is consistent with the basic premises of JIT inventory control systems which place a great deal of emphasis on reducing in-process inventories and related costs. Baker and Scudder (1990) provide a comprehensive review of machine scheduling with costs for both early and late completion of jobs. We will consider only problems in which the objective is to minimize the absolute lateness (or weighted absolute lateness) though there are other objectives which have been considered.

We divide the common due-date problems into two classes.

(a) problems in which the common due-date is fixed (e.g. Bagchi et al, 1986; Szwarc, 1989),

(b) problems in which the due-date is either a decision variable or fixed at a value large enough so that it does not constrain the scheduling of jobs before the due-date (e.g. Kanet, 1981; Cheng, 1987, 1991).

Problems in the first class tend to be NP-hard whereas many problems in the second class are solvable in polynomial time (see, for example, the survey papers by Cheng and

Gupta, 1989 and Baker and Scudder, 1990). Also, the first class clearly involves exogenous determination of the due-date while the second class involves endogenous due-date determination. It could be argued that the case in which the due-date is fixed at a very large value falls into the exogenous class since the due-date is determined by some outside agency. However, we place it in the other class because the value of the due-date does not affect the scheduling of the jobs.

We also divide the problems into two groups depending upon whether they are set in a single-machine or multi-machine facility. For the multi-machine case , only parallel identical machines have been considered so far and so we will limit our discussion to these.

1.2 Single Machine, Unrestrictive Due-Date

Kanet (1981) introduced the problem of scheduling jobs on a single machine to minimize the absolute deviation of job completion times from a common due date. Kanet used the objective function

$$f(\sigma) = \frac{1}{n}\sum_{j=1}^{n}\left|C_j(\sigma) - d\right| \tag{1.1}$$

where

n= the number of jobs to be scheduled

σ= a sequence of the n jobs

$C_j(\sigma)$= the completion time of the jth job

d= the common due date

He further made the assumption that the due date was large enough:

$$d \geq MS = \sum_{j=1}^{n} t_j \tag{1.2}$$

(where t_j= processing time of the jth job) so that it did not constrain the scheduling of the jobs. Under this assumption. He provided an algorithm for finding an optimal solution in polynomial (in n) time. Bagchi et al (1986) noticed that Kanet's solution remained valid under the weaker assumption that $d \geq d^0$ where (if we assume $t_1 \geq t_2 \geq \ldots \geq t_n$)

$$d^0 = \begin{cases} t_1 + t_3 + \ldots + t_n & \text{if n is odd} \\ t_2 + t_4 + \ldots + t_n & \text{if n is even} \end{cases} \tag{1.3}$$

Kanet's solution can be implemented by labeling the jobs in non-increasing order of processing times ($t_1 \geq t_2 \geq \ldots \geq t_n$), scheduling the longest job first, and assigning the jobs 2j and 2j+1 to the j+1'st and n-j+1'st positions (not necessarily respectively) for $j=1,2,\ldots,\frac{n-1}{2}$ (add a job (n+1) with $t_{n+1} = 0$ if n-1 is odd). The optimality conditions for Kanet's problem and the availability of a large number ($\geq 2^{\frac{n-1}{2}}$) of optimal solutions are discussed by Hall (1986) and Bagchi et al (1986). The large number of solutions stems from the fact that the job 2j can be scheduled either in the j+1'st position of the n-j+1'st position for each job $j=1,2,\ldots,\frac{n-1}{2}$. Emmons (1987) has considered secondary criteria for choosing among these optimal solutions in the general case of parallel machines.

Bector et al (1988) related the problem to a generalized linear goal program from which some basic results were proved using elementary properties of linear equations and a linear goal programming problem. Using these results. and the idea of sensitivity analysis in linear goal programming, they developed a 2-exchange type algorithm to solve the problem. The algorithm proceeds by exchanging the positions of complementary pairs of jobs when this improves the value of the objective . Complementary pairs are pairs of jobs, one scheduled early and one late, for which exchange of their positions does not disturb the V-shape property (early jobs in non-increasing order of processing times (LPT) and late jobs in non-decreasing order of processing times (SPT)) of the sequence. It can be shown that this V-shape property is preserved if and only if the two jobs are consecutive in processing time (i.e. i and j are consecutive in processing time if $\{k \mid t_i < t_k < t_j \quad \text{or} \quad t_j < t_k < t_i\} = \varphi$).

This fact establishes the connection between this algorithm and Kanet's. Kanet's algorithm can be executed by listing the job positions in non-decreasing order of the position labels

$$\lambda_j = \begin{cases} j-1 & j \le \left\lceil \frac{n-1}{2} \right\rceil \\ n-j+1 & j > \left\lceil \frac{n-1}{2} \right\rceil \end{cases} \tag{1.4}$$

and permuting the jobs among these positions until the jobs are in non-increasing order of processing times . Bector's algorithm can be executed by first listing the jobs in non-increasing order of processing times and the job positions in increasing order, and then permuting the job positions, by interchanging adjacent ones whose position labels are out of non-decreasing order, until the position labels are all in non-decreasing order.

Panwalkar et al (1982) considered a more general problem where the penalty for earliness and tardiness are different , the due date is a decision variable and large due dates are penalized. With each job sequence σ and due-date d is associated the penalty

$$f(\sigma,d) = \sum_{j=1}^{n} P_1 d + P_2 E_j + P_3 T_j \tag{1.5}$$

where P_1, P_2 and P_3 are non-negative constants and $E_j = \max\{0, d - C_j\}$ and $T_j = \max\{0, C_j - d\}$ are the earliness and tardiness (respectively) of the job j. They show that there exists an optimal common due date which coincides with the completion time of the K'th job where K is the smallest integer greater than or equal to $n(P_3 - P_1)/(P_2 + P_3)$. They also showed that the sequence that optimizes the objective function must be V-shaped

and provided a polynomially bound algorithm for solving the sequencing problem.

Bector (1988,1989) considered a generalization of this problem in which each job position has an arbitrarily assigned weight. He generalized Panwalkar's result concerning the due date (1988) to apply to this case and showed (1989) that the optimal sequence was still V-shaped. In both papers, a generalized linear goal programming approach was used.

Various authors have considered a more difficult problem which is similar to the one considered in Bector (1988,1989) in that the objective function is a weighted sum of deviations of completion times from a common due date. The weights, however, are associated not with positions in the sequence but with the jobs themselves. The due date in this problem , as in Panwalker (1982) and Bector (1988,1989), is a decision variable. This is essentially equivalent to constraining the due date to be very large (as in Kanet (1981) if processing of the jobs is allowed to begin any time after time t=0). Cheng (1985) formulated the due date determination part of the problem (determining the optimal common due date for a given job sequence) as an LP problem and derived the following result via solution of the LP dual problem: The optimal due date d* is given by

$$d^* = C_{[r]}$$

where

$$\sum_{i=1}^{r-1} w_{[i]} \leq \frac{1}{2} \tag{1.6}$$

and

$$\sum_{i=1}^{r} w_{[i]} \geq \frac{1}{2} \tag{1.7}$$

Here [i] denotes the job in the ith position of the sequence and w_i denotes the weight associated with job i. This result is essentially the same as that given in Bector (1988,1989) except that the weights are associated with the jobs instead of with the job positions in the sequence. Cheng (1987) provided an algorithm for finding the optimal sequence by complete enumeration of sequences which satisfy (1.6) and (1.7) as well as a weighted V-shape property (the quantities $\frac{t_i}{w_i}$ are in a V shape) which Cheng has shown to be possessed by at least one optimal sequence. The algorithm has complexity $O(n^2 2^n)$ and is efficient for small to medium values of n. In a subsequent paper Cheng(1990) improves upon this by providing an algorithm of complexity $O(n^{\frac{1}{2}} 2^n)$. This is a partial search algorithm which eliminates some schedules implicitly by applying conditions (6) and (7) at an earlier stage.

Hall and Posner (1989) prove that the problem is NP hard and provide a pseudo-polynomial dynamic programming solution of complexity $O(n\sum_j t_j)$. The basic idea behind the algorithm is as follows. If we consider only schedules which possess the V-shape property, then a schedule is completely determined (except where some jobs have identical values of $\frac{t_i}{w_i}$) by specifying whether each job is scheduled early or late. Also the contribution to the objective function from scheduling a job k early is just w_k multiplied by the sum of the processing times of the jobs scheduled after k but completed early. A similar result holds for scheduling jobs late. Thus, since this contribution depends only on the sum of the processing times of these jobs and not upon which jobs in particular are scheduled, it is possible to formulate an efficient pseudo-polynomial dynamic programming algorithm to solve the problem. The recursion relation is:

$$f_k(e) = \min\left\{ w_k e + f_{k+1}(e + t_k), w_k(\sum_{j=1}^{k} t_j - e) + f_{k+1}(e) \right\} \tag{1.8}$$

where $f_k(e)$= the minimum cost to schedule jobs k,k+1,....,n given that, before job k is scheduled, the sum of the processing times of the early jobs is e. For the case where the weights are bounded by a polynomial function of the number of jobs, a fully polynomial approximation scheme is given. In this scheme the processing times are rounded off to integral multiples of a basic quantum so that the sums of processing times e are then also restricted to multiples of this quantum, thus reducing the number of possibilities (for e) to be considered in the dynamic programming algorithm. The relative error due to this rounding off is proportional to the size of the quantum. Some polynomially solvable cases are also described.

Cheng and Kahlbacher (1989) have provided a similar algorithm in which the direction of the dynamic programming algorithm is reversed with respect to Hall and Posner's. The recursion relation is given by:

$$f'_k(e) = \min\left\{ w_k|t_k - e| + f'_{k-1}(e - t_k), w_k(\sum_{j=1}^{k} t_k - e) + f_{k-1}(e) \right\} \tag{1.9}$$

where $f'_k(e)$= the minimum cost of scheduling the job set {1,2,....,k} about the due date e. They also provide a polynomial approximation scheme by rounding off the processing times.

De et al (1990) provide a rather complete survey of the problem and various solution algorithms. They outline three possible approaches to the problem. The first is a 0-1 integer quadratic programming formulation where the 0-1 variables are given by:

$$x_j = \begin{cases} 0 & \text{if j is scheduled late} \\ 1 & \text{otherwise} \end{cases} \tag{1.10}$$

and the objective function is expressed in terms of these variables. The authors suggest a local search heuristic which consists of changing the value of one variable at a time $(x_j \rightarrow 1 - x_j)$ until a local minimum is found. This formulation and heuristic is very closely related to the method we shall propose here.

The second method outlined consists of two possible approaches of a branch and bound algorithm. The first, RBB_A schedules the inside jobs (i.e. those with smaller values of $\frac{t_j}{w_j}$ which are scheduled further from, the beginning or end of the sequence) first whereas the second approach RBB_P schedules the outside jobs (those with larger values of $\frac{t_j}{w_j}$ which are scheduled closer to the beginning or end of the sequence) first. The authors find, after solving some test problems that the first approach tends to work better for larger values of n and they conjecture that this is because the bounds used in this approach are tighter than those used in the second approach.

The third method outlined consists of two dynamic programming approaches. One approach is identical to that proposed by Cheng and Kahlbacher. The other differs from both those of Cheng and Kahlbacher and Hall and Posner in that the recursion relation depends on the sum of the weights (w_j) of the jobs previously scheduled early instead of the sum of the processing times of those jobs. In a number of test problems the dynamic programming approaches were found to consistently outperform the other exact algorithms. The authors point out, however, that the approach may not be practical (in terms of time and space requirements) for problems involving disagreeable values of the parameters (w's and t's). The local search heuristic consistently produced close-to-optimal results (within 10%) in negligible time.

In chapter 2 we adopt a simulated annealing approach as a heuristic for this problem. We formulate the problem as minimization of the consensus function of a Boltzmann machine . Following the approach in Aarts, we defines a polynomial ($O(n^2)$) algorithm which finds near -optimal solutions via Monte Carlo computer simulation of an analogous physical process. This algorithm is guaranteed, under certain conditions which are satisfied by this problem, to converge to the optimal solution. However, since this convergence requires an infinite amount of time in general, a finite-time approximation must be used. On the basis of some computational experiments we conclude that the performance of the annealing algorithm is comparable to that of some of the more effective methods proposed so far when the processing times are uniformly distributed.

1.3 Single Machine, Restrictive Due-Date

Szwarc (1988) considered Kanet's problem where the due date is small enough to constrain the scheduling decision. He derived some necessary conditions for optimality and developed a branch and bound procedure for solving the problem.

Hall and Posner (1989) showed that the problem is NP-complete and provided a pseudo -polynomial dynamic programming algorithm of complexity $O(n\sum_j t_j)$ to solve the problem. Liman and Lee (in submission) provide a simple heuristic for the problem which has a complexity of O(nlogn) and a tight error bound of 50%.

Garey et al (1988) have considered a problem where the jobs all have different due dates and the objective is to minimize the sum of the absolute deviations of the job completion times from their respective due dates. They showed that the problem is NP-complete and gave an efficient algorithm for finding minimum cost schedules whenever the

jobs all have the same length or are required to be processed in a given sequence. Ahmed (1989) developed a very good 2-exchange local search heuristic for solving this problem. In this procedure, the positions of two jobs are interchanged whenever this leads to an improvement in the objective function. The procedure halts when no such pair of jobs can be found. In general, this procedure does not find an optimal solution for the problem but it finds a solution which is locally optimal with respect to the neighborhood structure defined by the exchange procedure. Sundaragahvan and Ahmed (1984) also provided a heuristic for the single machine, reatrictive due-date case.

1.4 Parallel Machines, Unrestrictive Due-Date

Sundararaghavan and Ahmed(1984) extend Kanet's result to the case of parallel identical machines and provide a constructive algorithm for solving this problem. Hall (1986) provides an algorithm for this problem which takes into account the large number of optimal solutions available and widens the applicability of the model by providing the scheduler with many ways to achieve an optimal solution. Emmons (1987) considers a generalization of this problem in which the earliness and tardiness penalties are weighted differently. He takes advantage of the large number of optimal solutions available by considering the secondary criteria of minimizing makespan and machine occupancy.

The generalization of Panwalkar et al's problem to the case of multiple, parallel and identical machines has been considered by Cheng (1989). He shows that Panwalkar's result concerning the optimal due-date for a given sequence can be generalized so that it will apply to this case and he provides a heuristic for finding approximate solutions to the problem. He shows, in a later paper (Cheng and Kahlbacher, 1990), that the parallel-

machine version of the problem is NP-hard. Cheng's formulation of the problem also includes the constraint that all machines begin processing at time $t = 0$ so that there is no idle time on any machine. This constraint is, of course, redundant in the single-machine case since it is clearly not advantageous to insert idle time before the processing of the first job. It can also be shown that it is never advantageous to insert idle time between jobs in the multiple-machine case. So it is only necessary to consider sequences without inserted idle time between jobs.

De et al (1991) have observed that better solutions can be obtained for this problem if Cheng's zero start time constraint is relaxed. Although their approach is applicable in the case where no cost is incurred by introducing idle time before the start of processing, there are cases in which such costs due to machine idleness may arise. In chapter 3 we generalize the problem to include both Cheng's and De et al's versions of the problem by relaxing the zero start time constraint and replacing it with a cost which is linear in the total idle time. The zero start time version and, because of no inserted idle time between jobs, the zero idle time version due to Cheng is recovered when the idle time cost is sufficiently large, while the relaxed zero start (idle) time version due to De et al is recovered when the idle time cost is set to zero.

We first show that the problem is NP-hard. We then construct approximate solutions by employing the optimal solution of a polynomially solvable problem which is obtained by allowing the idle time penalty to be negative, thus rewarding idle time. We provide upper bounds on the relative error incurred by the approximation and show that, if the cost associated with the due-date is non-zero, the approximation is asymptotically optimal as the number of jobs grows large.

1.5 Parallel Machines, Restrictive Due-Date

The parallel machine version of Kanet's problem in which the due date is not required to be large is considered by Sundararaghavan and Ahmed (1984). They suggest a heuristic and provide some computational experience. The heuristic seems to perform well for the instances of the problem considered.

1.6 Conclusion

The development and current state of literature regarding common due date assignment and job scheduling on single or parallel machines to minimize absolute lateness (or weighted absolute lateness) has been reviewed. There is also a great deal of literature available which considers similar problems such as minimizing the flowtime variance on single or parallel machines. The current interest in JIT inventory control systems will most likely continue to motivate research in this area.

Chapter 2

Simulated Annealing of a Single Machine Problem (Unrestrictive Due-Date Case)

2.1 Introduction

In this chapter we propose a polynomial simulated annealing algorithm as a heuristic solution for the single machine scheduling problem in which weights are assigned to each job. This is the generalization of Kanet's problem (see chapter 1) which has been considered by Cheng (1985,1987,1989), Hall and Posner (1989), Cheng and Kahlbacher (1989) and De et al (1990). Recall (from chapter 1) that the methods considered so far include enumeration (Cheng) , Dynamic programming (Cheng & Kahlbacher, Hall & Posner and De et al), Branch and bound,quadratic 0-1 programming and a local search heuristic derived from the 0-1 programming formulation (De et al).

We formulate the problem in terms of a neural computing network model known as a Boltzmann machine. Using this model , we simulate a stochastic process in which the Boltzmann machine moves gradually toward an optimal configuration. We use the polynomial annealing schedule given in Aarts to control this process.

On the basis of some computational experiments we conclude that the performance of the annealing algorithm is comparable to that of some of the more effective methods proposed so far when the processing times are uniformly distributed.

2.2 Problem Formulation

Let N={1,2,...,n} represent a set of n jobs to be processed on a single processor . With each job $j \in N$ is associated a processing time t_j and a weight w_j The more "important" jobs are given larger weights and both the processing times and the weights are normalized so that they sum to 1. We assume that all jobs are available at time t=0 and that production starts at time t=0. No preemption is allowed. A schedule σ can be described by specifying the completion times C_j ;j=1,2,...n for all jobs in such a way that the jobs do not overlap. i.e.

$$C_{j'} \leq C_j - t_j \quad \text{or} \quad C_{j'} \geq C_j + t_{j'} \tag{2.1}$$

for all pairs $(j, j') \in N \times N$. Under these assumptions we wish to find a schedule σ^* and due date d* which minimize the objective function :

$$f(\sigma, d) = \sum_{j=1}^{n} w_j \left| C_j - d \right| \tag{2.2}$$

We will refer to this problem as P1.

2.3 Previous Results

The proof of the following results can be found in Cheng and Kahlbacher (1989).

Lemma 1:: It is not useful to insert idle times between consecutive jobs in the schedule.

Because of *Lemma 1* a schedule σ can be specified by giving the sequence of jobs (which we will also denote by σ) in the schedule.

Lemma 2 : For a given job sequence σ an optimal common due date d* coincides with the completion time of the rth job in σ where r satisfies

$$\sum_{j=1}^{r-1} w_j \leq \frac{1}{2} \leq \sum_{j=1}^{r} w_j \tag{2.3}$$

Lemma 3 : There exists an optimal sequence in which jobs scheduled early (j is considered early if $C_j \leq d$) are processed in non-increasing order of the quantities $\frac{t_j}{w_j}$ whereas jobs scheduled late ($C_j > d$) are processed in non- decreasing order of the quantities $\frac{t_j}{w_j}$.

2.4 0-1 Formulation

We formulate P1 as follows:

$$\text{Let } x_i = \begin{cases} 1 & \text{if job i is scheduled late} \\ 0 & \text{otherwise} \end{cases}$$

From the three lemmas in the previous section , it is clear that knowing whether each job is scheduled erly of late is inough to determine a schedule. Specifically, we have:

$$f(x) = \sum_{\substack{i,j \in N \\ i<j}} \min\left\{\frac{t_j}{w_j}, \frac{t_i}{w_i}\right\} w_i w_j [x_i x_j + (1-x_i)(1-x_j)] + \sum_{i \in N} w_i t_i x_i . \qquad (2.4)$$

Let

$$s'_{ij} = \min\left\{\frac{t_j}{w_j}, \frac{t_i}{w_i}\right\} w_i w_j . \qquad (2.5)$$

Then

$$f(x) = \sum_{\substack{i<j \\ i,j \in N}} s'_{ij} + 2 \sum_{\substack{i<j \\ i,j \in N}} s'_{ij} x_i x_j - \left(\sum_{\substack{i<j \\ i,j \in N}} s'_{ij} x_i + \sum_{\substack{i<j \\ i,j \in N}} s'_{ij} x_j \right) + \sum_{i=1}^{n} s'_{ii} x_i \qquad (2.6)$$

Since $\sum_{\substack{i<j \\ i,j \in N}} s'_{ij} x_j = \sum_{\substack{i>j \\ i,j \in N}} s'_{ij} x_i$ we have:

$$f(x) = \sum_{\substack{i<j \\ i,j\in N}} s'_{ij} + 2\sum_{\substack{i<j \\ i,j\in N}} s'_{ij} x_i x_j + \sum_{i=1}^{n}\left(s'_{ii} - \sum_{\substack{i\neq j \\ i,j\in N}} s'_{ij} \right) x_i$$

$$= \sum_{\substack{i<j \\ i,j\in N}} s'_{ij} + 2\sum_{\substack{i<j \\ i,j\in N}} s'_{ij} x_i x_j + \sum_{i=1}^{n}\left(2s'_{ii} - \left(s'_{ii} + \sum_{\substack{i\neq j \\ i,j\in N}} s'_{ij} \right) \right) x_i$$

$$= \sum_{\substack{i<j \\ i,j\in N}} s'_{ij} + 2\sum_{\substack{i<j \\ i,j\in N}} s'_{ij} x_i x_j + \sum_{i=1}^{n}\left(2s'_{ii} - \sum_{j=1}^{n} s'_{ij} \right) x_i .$$

Since $i = j \Rightarrow x_i = x_i x_j$, we can write:

$$f(x) = \sum_{\substack{i<j \\ i,j\in N}} s'_{ij} + 2\sum_{\substack{i<j \\ i,j\in N}} s'_{ij} x_i x_j + \sum_{i=1}^{n}\left(2s'_{ii} - \sum_{j=1}^{n} s'_{ij} \right) x_i x_j$$

$$= \sum_{\substack{i<j \\ i,j\in N}} s'_{ij} + \sum_{\substack{i\leq j \\ i,j\in N}} s_{ij} x_i x_j \qquad (2.7)$$

where

$$s_{ij} = \begin{cases} 2s'_{ij} & \text{if } i \neq j \\ 2s'_{ii} - \sum_{k=1}^{n} s'_{ik} & \text{if } i = j \end{cases} \quad . \tag{2.8}$$

Thus, since $\sum_{\substack{i<j \\ i,j \in N}} s'_{ij}$ is independent of the x_i's, we can write P1 as:

$$\text{minimize} \quad \sum_{\substack{i \leq j \\ i,j \in N}} s_{ij} x_i x_j$$

$$\text{subject to:} \quad x_i = 0 \text{ or } 1, \qquad i=1,2,\ldots,n$$

where the s_{ij}'s are calculated from (2.5) and (2.8).

2.5 Boltzmann Machine Formulation

The problem P1 can be formulated in terms of minimizing the consensus function of a Boltzmann machine . A Boltzmann machine can be viewed as a network consisting of a number of 2-state units which are connected in some way. The network is represented by an undirected graph G=(U,C) where U denotes the set of units and C is a set of unordered pairs of elements of U. An element of C is called a connection. With each connection $\{u,v\} \in C$ is associated a connection strength s_{uv} . A unit $u \in U$ can be in one of two states: on or off. A connection is considered activated if both units in the connection are on. A configuration k of a Boltzmann machine is given by a global state of the machine and is defined by a sequence or vector of length $|U|$ whose uth component is given by:

$$k_u = \begin{cases} 1 & \text{if u is on} \\ 0 & \text{if u is off} \end{cases} \tag{2.9}$$

The configuration space S is given by the set of $2^{|U|}$ possible configurations. The consensus function $f:S \rightarrow \Re$ assigns to each configuration k a real number f(k) given by the sum of the connection strengths of all of the activated connections. i.e.

$$f(k) = \sum_{\{u,v\} \in C} s_{uv} k_u k_v \tag{2.10}$$

The problem P1 can be formulated as minimization of the consensus function of a Boltzmann machine with n units, one for each job in P1, where units corresponding to early jobs are off and those corresponding to late jobs are on. The graph G, in this case, is a complete graph. From the three Lemmas in the previous section it is clear that a configuration of the machine uniquely defines a schedule for P1 since we need only know which jobs are early and which are late. Let x denote a configuration of the machine and let x_j denote the state (0 or 1) of the unit corresponding to the job j. Then P1 is equivalent to finding a configuration which minimizes the consensus function of a Boltzmann machine with connection strengths s_{ij} $i, j \in N$. In order to reach a near optimal configuration a state transition mechanism is introduced which allows the units to adjust their states to those of their neighbors. Unit i is a neighbor of unit j if there is a connection $\{i, j\} \in C$ between them. Recall that, for P1, the graph of the Boltzmann Machine is complete and so every unit is a neighbor to every other unit in this case. The adjustment is determined by a

stochastic function of the states of the neighbors and the corresponding connection strengths. The process used here is a special case of the method of simulated annealing (SA).

2.6 Simulated Annealing of the Boltzmann Machine

Annealing is a thermal process for obtaining low energy states of a solid in a heat bath. The process consists of first increasing the temperature of the heat bath until the solid melts, and then gradually lowering the temperature until the particles in the solid arrange themselves in the lowest energy or ground state of the solid. In contrast to quenching, where the temperature of the bath is rapidly reduced, annealing gives the particles time to come to thermal equilibrium at each temperature so that they end up in a stable state corresponding to a global energy minimum. In quenching, the particles do not have time to come to thermal equilibrium and so end up in a meta-stable state corresponding to a local energy minimum.

In thermal equilibrium at temperature T the probability that a system is in a state x with energy E(x) is given by the Boltzmann distribution:

$$P_T(x) = \frac{\exp\left(-\frac{E(x)}{k_B T}\right)}{\sum_{y \in S} \exp\left(-\frac{E(y)}{k_B T}\right)} \tag{2.11}$$

where S is the set of all possible states of the system and k_B is the Boltzmann constant. It

can be shown (see Aarts and Korst (1981)) that:

$$\lim_{T \to 0} P_T(x) = \begin{cases} 1 & \text{if } x \in S_{min} \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

where S_{min} is the set of all states with minimum energy. Thus if the temperature is lowered gradually enough so that the particles in the solid are always in thermal equilibrium (or very close to it) we would expect that, eventually, all the particles would be in a ground state (a member of S_{min}). In simulated annealing this process is simulated on a computer via Monte Carlo techniques. In optimization applications the objective function which is to be minimized replaces the energy of the physical system and, at each temperature, thermal equilibrium is simulated by constructing a Markov chain whose stationary distribution is the Boltzmann distribution at the appropriate temperature.

In a Boltzmann machine we say that two states or configurations x and y are neighboring if they differ on only one unit. i.e.

$$\left|\{u \in U | x_u \neq y_u\}\right| = 1 \tag{2.13}$$

For any configuration k we will denote the set of all neighboring configurations by S_k. Thermal equilibrium is reached by simulating a Markov chain with transition probabilities given by:

$$P_{xy}(T) = \begin{cases} G_{xy} A_{xy}(T) & \text{if } x \neq y \\ 1 - \sum_{z \neq x} P_{xz}(T) & \text{if } x = y \end{cases} \tag{2.14}$$

where

$P_{xy}(T)$ =the probability of the system making a transition from state x to state y at temperature T

$$G_{xy} = \begin{cases} \frac{1}{|S_x|} & \text{if } y \in S_x \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

$$A_{xy}(T) = \exp\left[-\frac{(f(y)-f(x))^+}{T}\right] \tag{2.16}$$

and for all $a \in \Re$

$$a^+ = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

The G_{xy}'s are called generation probabilities and the A_{xy}'s are the acceptance probabilities. The Markov chain is simulated by proposing a transition from the current configuration to a neighboring configuration, generating a random number R from a uniform distribution on [0,1] and executing the transition if $R \leq A_{xy}(T)$. This is done a large number of times so that after a time the probability of finding the system in any given configuration approaches the probability, under the stationary distribution associated with the transition matrix $P_{xy}(T)$, of finding the system in that state. The stationary distribution for this Markov chain is (see Aarts for proof) a Boltzmann distribution at temperature T.

Let x be a configuration of the boltzmann machine representing our scheduling problem and let f(x) denote the value of the consensus function of the machine in that configuration..

For our problem we set :

(a) $$G_{xy} = \frac{1}{n} \tag{2.18}$$

(b) Let x^u denote the configuration obtained from x by changing the state of the unit u. Then

$$x_v^u = \begin{cases} x_v & \text{if } v \neq u \\ 1 - x_v & \text{if } v = u \end{cases} \tag{2.19}$$

Let x and y be neighboring configurations. Then $y = x^u$ for some $u \in U$ and we have, as in Aarts:

$$f(y) - f(x) = f(x^u) - f(x)$$

$$= \left[x_u^u \sum_{\{u,v\} \in C_u} s_{uv} x_v^u + (x_u^u)^2 s_{uu} \right] - \left[x_u \sum_{\{u,v\} \in C_u} s_{uv} x_v + x_u^2 s_{uu} \right] \tag{2.20}$$

where $C_u = \{\{u,v\} \in C | v \in U\}$. Using (28) we get:

$$f(x^u) - f(x) = (1 - 2x_u)\left[\sum_{\{u,v\}\in C_u} s_{uv}x_v + s_{uu}\right] \tag{2.21}$$

Thus the effect on the consensus function resulting from a transition to a neighboring configuration and hence also the acceptance probability for the transition is completely determined by the states of the neighbors of the unit that is switched. Consequently each unit can evaluate its transition probability locally.

We now have the tools to simulate the Markov chain at any given temperature.

2.7 Asymptotic Convergence

The SA process consists of a number of homogeneous regular Markov chains (one at each value of the decreasing temperature) which combine to form an inhomogeneous chain which converges in distribution to the probability vector q* with components

$$q_y^* = \begin{cases} \dfrac{1}{|S_{min}|} & \text{if } x \in S_{min} \\ 0 & \text{otherwise} \end{cases}. \tag{2.22}$$

This convergence is guaranteed (see *Theorem 3.6* in Aarts) provided three conditions are met. Let T_k be the temperature of the kth homogeneous regular Markov chain and let L be the length (number of transitions) of each of the chains. Then the following three conditions are sufficient for convergence.

(C1) The transition matrix for the kth Markov chain is given by $P_{xy}(T_k)$ as defined in (2.14)

(C2) Each possible state $x \in S$ can be reached from any other state $y \in S$ through a finite number of transitions for which the generation probabilities are all non-zero.

(C3) $$T_k \geq \frac{(L+1)\Delta}{\log(k+2)} \qquad k = 0,1,\ldots\ldots\ldots \tag{2.23}$$

where

$$\Delta = \max_{\substack{x,y \in S \\ y \in S_x}} \{f(y) - f(x)\} \tag{2.24}$$

and L is chosen such that any state $x \in S$ can be reached from some state $y \in S_{min}$ in L transitions or less.

We are guaranteed to eventually converge to the desired distribution provided these conditions are met. However, in general, this may take an infinite number of steps and so we require a finite time approximation for this scheme. We now describe an annealing schedule (a schedule for the variation of the temperature T; i.e. specification of the sequence $T_1, T_2, \ldots\ldots T_k, \ldots\ldots$) due to Aarts which can be performed in polynomial (in $O(n^2)$) time.

2.8 Polynomial Annealing Schedule

The following annealing schedule is proposed in Aarts and Korst (1981) as a general purpose simulated annealing schedule. We will use this schedule for the simulated annealing of the Boltzmann Machine which represents our scheduling problem.

(1) Initial Value of the Temperature

The initial value of the temperature is chosen so that nearly all proposed transitions are accepted. Suppose that a sequence of trials is generated in which m_1 of the proposed transitions correspond to a decrease in the objective function and m_2 correspond to an increase. Let $\overline{\Delta f^{(+)}}$ denote the average cost increase for the m_2 cost increasing transitions. We can approximate the number of accepted cost-increasing transitions by

$$m_2 \exp\left(-\frac{\overline{\Delta f^{(+)}}}{T}\right).$$

The fraction of accepted transitions is then approximated by:

$$\chi(T) \approx \frac{m_1 + m_2 \exp\left(-\frac{\overline{\Delta f^{(+)}}}{T}\right)}{m_1 + m_2} \tag{2.25}$$

Thus we have:

$$T \approx \frac{\overline{\Delta f^{(+)}}}{\ln\left(\frac{m_2}{m_2\chi - m_1(1-\chi)}\right)}. \tag{2.26}$$

The initial temperature is calculated in the following way.

<u>step 1:</u> Perform a sequence of m trials with T=0 to determine initial values of m_1, m_2 and $\Delta f^{(+)}$.

<u>step 2:</u> Define an acceptance ratio χ_0 ($\chi_0 \approx .95$ usually) and evaluate T in (2.26) by setting $\chi_0 = \chi_0$ and substituting m_1, m_2 and $\Delta f^{(+)}$ from above.

<u>step 3:</u> Determine new values of m_1, m_2 and $\Delta f^{(+)}$ by executing another series of trials at the new temperature.

<u>step 4:</u> Update the temperature using (35) and the results from the last series of trials and go to step 3.

This algorithm continues until the value of T converges.

(2) <u>Temperature decrement</u>

We decrement the value of the temperature, as in Aarts, according to the relation:

$$T_{k+1} = \frac{T_k}{1 + \frac{T_k \ln(1+\delta)}{3\sigma_k}} \tag{2.27}$$

where

δ = a small positive number. We use δ=0.1 in accordance with common practice.

σ_k = variance of the objective function values of states visited in a Markov chain at temperature T_k.

This guarantees that (see Aarts *Theorem 4.1*)

$$\frac{1}{\delta} < \frac{q_x(T_k)}{q_x(T_{k+1})} < 1+\delta \qquad \forall x \in S, \;\; k = 0,1,.... \tag{2.28}$$

where $q_x(T_k)$ represents the probability, under the stationary distribution of the Markov chain at temperature T_k, of the system being in the state x. Thus the stationary distribution changes gradually so that the system, if it begins in thermal equilibrium, will remain very close to equilibrium throughout the annealing process.

(3) <u>Final Value of the Temperature</u>

Let $\langle f \rangle_T$ denote the expectation value of the objective function for the stationary distribution of a Markov chain at temperature T. (This notation is common in Statistical Mechanics Literature.) Also let

$$\langle f\rangle_\infty = \lim_{T\to\infty} \langle f\rangle_T. \tag{2.29}$$

For small T we can use the linear approximation

$$\langle f\rangle_T - \langle f\rangle_0 \cong T\frac{\partial\langle f\rangle_T}{\partial T}. \tag{2.30}$$

Hence we may reliably terminate the algorithm if for some k we have

$$\frac{T_k}{\langle f\rangle_\infty}\frac{\partial\langle f\rangle_T}{\partial T}\bigg|_{T=T_k} < \varepsilon_s \tag{2.31}$$

where ε_s is a small positive number referred to as the stop parameter. We used $\varepsilon_s = 10^{-5}$ in accordance with common practice. In practice $\langle f\rangle_T$ is approximated by an average over states visited during simulation of a Markov chain at temperature T. $\langle f\rangle_\infty$ is approximated by $\langle f\rangle_{T_0}$ where T_0 is the initial temperature. Also in practice we terminate the algorithm if no transitions are accepted at some temperature T_k since in this case $\sigma_k = 0$ so that $T_{k+1} = 0$ by (36).

(4) <u>Length of the Markov Chains</u>

We let L=|U| (|U| = |N| = n for our problem). This guarantees that condition (C2) is satisfied. Let K be the first integer for which the stop criterion (40) is satisfied. Then it can be shown (Aarts *Theorem 4.3*) that

$$K = O(\ln|S|) = O(n) \tag{2.32}$$

It can also be shown that the number of steps required to simulate one of the homogeneous Markov chains at a single temperature is O(n).Thus the simulated annealing algorithm with the above annealing schedule has complexity $O(n^2)$.

2.9 Numerical Results

We solved 10 problems for each value of n=20,50,100 using:

(a) Local search Heuristic (De et al)

(b) Simulated annealing of the Boltzmann Machine model of the problem

(c) Approximate Dynamic Programming algorithm (Cheng and Kahlbacher).

The results are recorded in Table 2.1. In the approximate Dynamic Programming algorithm, each processing time t_j is approximated by :

$$t_j^Q = Q\left\lfloor \frac{t_j}{Q} \right\rfloor. \tag{2.33}$$

Since the processing times are uniformly distributed and normalized so that they sum to 1, the average processing time is always given by:

$$\bar{t} = \frac{1}{n}. \tag{2.34}$$

For the cases n=20 and n=50 we ran the approximate DP algorithm at $Q=.5\bar{t}, .2\bar{t}$ and $.01\bar{t}$. For the case n=100 we ran the algorithm at $Q=.5\bar{t}, .2\bar{t}$ and $.06\bar{t}$. We chose the smallest value of Q for the case n=100 larger than the smallest value used for n=20 or n=50 because we had insufficient memory to run n=100 at $Q=.01\bar{t}$. In each case we recorded an upper bound for the relative error according to:

$$\varepsilon = \frac{f - lb}{lb} \tag{2.35}$$

where f is the value of the objective function for the schedule obtained by the algorithm used and lb is the lower bound for that instance of the problem. This lower bound is given by the minimum value of the objective function returned from the DP algorithm for the instance of the problem with the approximate processing times t_j^Q, where $Q=.01\bar{t}$, for n=20 and n=50, and $Q=.06\bar{t}$ for n=100 (i.e. the smallest value of Q used for a given value of n). Table 1 lists the minimum, maximum and average value of ε for each algorithm and for each value of n as well as the average computation time. We also include the maximum number of cells required for the array which represents $f_k(e)$ in the DP algorithms. We find that for n=50 and for n=100 both the average accuracy and computation time of the SA algorithm fall somewhere in between those of the approximate DP algorithms at $Q=.5\bar{t}$ and $.2\bar{t}$. For n=20, the accuracy of SA also falls between that of these two approximate DP algorithms but the computation time is somewhat better for both of the DP algorithms. Thus the SA algorithm seems to be quite comparable, in terms of accuracy and computation

time to these two algorithms. The local search heuristic is less accurate than all of these but is also the fastest. The DP algorithm for the smallest values of Q do somewhat better than all of these, but at great costs in terms of memory and computation time required.

The upper bound for the absolute error incurred by the DP algorithm is given by (see Cheng and Kahlbacher)

$$\Delta \leq nQ = \frac{Q}{\bar{t}}. \tag{2.36}$$

The actual magnitude of the minimum value of the objective function is between .1 and .15 for all cases. Thus Δ corresponds to a relative error of roughly 500% for $Q=.5\bar{t}$ and 200% for $Q=.2\bar{t}$. The posterior upper bounds (ε's) for these algorithms are less than 8.1% for all cases considered. Thus it is clear that the actual performance of these algorithms is much better than the a priori upper bounds suggest. In fact it seems somewhat remarkable that the actual error is less than 8.1% when the processing times are truncated to integer multiples of a quantum (Q) which is half the average processing times of the jobs.

2.10 Conclusion

We have introduced a heuristic algorithm for solving an NP hard problem based on the theory of simulated annealing. The performance of the algorithm was found to be comparable to that of the approximate Dynamic Programming algorithm for comparable computation times. The DP algorithm has the advantage that much more accurate solutions

can be found, however this also requires a very large increase in the amount of memory and computation time used. The simulated annealing algorithm has the advantage that it requires only that we be able to evaluate the objective function for any particular solution. Because of this, it is a very general method, and can accommodate changes to the model easily by simply changing the way we evaluate the objective function.. It also requires very little memory , but it is limited in it's accuracy since, unlike the approximate DP algorithm, its error cannot be controlled by adjusting the value of a parameter.

Chapter 3

Approximate Solution for the Parallel Machine Problem

3.1 Introduction

In this chapter we propose an approximate solution for the parallel machine generalization of Panwalkar's problem discussed in chapter 1. This problem has been considered by Cheng (1989), Cheng & Kahlbacher (1990) and De et al (1991) (Chapter 1 includes a discussion of these results). Cheng introduced the constraint that all machines must begin processing at time zero and provided a heuristic for solving the problem. De et al have criticized the imposition of this constraint because better solutions can be obtained when it is relaxed, We will bridge the gap between these two views by introducing an idle time penalty which , if very large, recovers Cheng's version and which, if set to zero, recovers De et al's version.

We will show that the problem is NP-hard and provide an algorithm for obtaining approximate solutions which are asymptotically optimal as the number of jobs tends to infinity.

3.2 Problem Formulation

Let $J = \{1, 2, ..., n\}$ be a set of n independent jobs to be scheduled on a set $M =$

$\{1, 2, ..., m\}$ of m identical machines. Let t_j denote the processing time (on any of the m machines) required by job $j \in J$. Also let the jobs in J be indexed in non-increasing order of processing times (i.e. $t_1 \geq t_2 \geq ... \geq t_n$). All jobs are available at the same time for processing and are assigned a common due-date $d \geq 0$. We define a schedule to be any function of the form σ: $\{1, 2, ..., n\} \rightarrow \Re \times \{1, 2, ..., m\}$, i.e.

$$\sigma(j) = (s_j, m_j), \qquad \text{for all } j \in J. \tag{3.1}$$

Thus σ assigns to each job $j \in J$ a start time s_j and a machine m_j. The schedule is feasible if the following conditions are satisfied:

(a) $s_j \geq 0 \quad j = 1, 2, ..., n,$

(b) $s_{j'} + t_{j'} \leq s_j$ or $s_{j'} \geq s_j + t_j$ for all pairs of (j, j') with $m_j = m_{j'}$. (3.2)

Let $C_j = s_j + t_j$ denote the completion time of job j. A job is considered early if it is finished on or before the due-date d, and late otherwise. Let J_i^E and J_i^T denote respectively the sets of jobs which are scheduled early and late on machine i, i = 1, 2, ..., m, and let $J^E = \bigcup_{i \in M} J_i^E$ and $J^T = \bigcup_{i \in M} J_i^T$. For a given sequence σ, let b_i denote the earliest time at which machine i is occupied. Since it is again easy to see that it is not beneficial to insert idle time between jobs, the total idle time in σ is given by

$$I = \sum_{i=1}^{m} b_i . \tag{3.3}$$

Let Π be the set of all feasible schedules. Then our objective is to find an optimal combination of due-date d^* and schedule $\sigma^* \in \Pi$ which minimizes the objective function

$$f(\sigma,d) = P_I I + \sum_{j=1}^{n}\left\{P_d d + P_E E_j + P_T T_j\right\}$$

$$= P_I \sum_{i=1}^{m} b_i + \sum_{j=1}^{n}\left\{P_d d + P_E E_j + P_T T_j\right\} \qquad (3.4)$$

subject to the constraint

$$\min_{i\in M}\{b_i\} = \min_{j\in J}\{s_j\} = 0 \qquad (3.5)$$

where P_I, P_d, P_E and $P_T \geq 0$ are constants representing the unit penalties for idle time, due-date, earliness and tardiness respectively. Note that constraint (3.5) is redundant if $P_I > 0$. We will refer to this problem as P2.

3.3 NP-Hardness of P2

We now show that problem P2 is NP-hard. We shall show that the recognition version of P2 is NP-complete by a reduction from the minimum makespan problem $P||C_{max}$ which has been known to be NP-complete (Garey and Johnson, 1978). The recognition form of $P||C_{max}$ can be stated as follows.

<u>Minimum Makespan Problem $P||C_{max}$</u>

Given a set of n jobs $\{1, 2, ..., n\}$ with processing times $\{t_1, t_2, ..., t_n\}$, a set of

m machines $\{1, 2, ..., m\}$ and a real number C, can we assign each job j a start time $s_j \in \Re$ and a machine $m_j \in 1, 2, ..., m\}$ such that

(a) $s_j \geq 0 \quad j = 1, 2, ..., n,$

(b) $s_{j'} + t_{j'} \leq s_j$ or $s_{j'} \geq s_j + t_j$ for all pairs of (j, j') with $m_j = m_{j'}$, (3.6)

(c) $\max_{j \in J}\{s_j + t_j\} \leq C?$

Proposition 1: Problem P2 is NP-hard.

Proof:

Given an instance of $P||C_{max}$ with n jobs having processing times $\{t_1, t_2, ..., t_n\}$, m machines and the real number C, we construct an instance of the problem P1 which as the same jobs, machines and processing times parameters while the parameters P_I, P_d, P_E and P_T are are given as $P_I = P_E = P_T = 0$ and $P_T > nP_d > 0$. We show that there exists a feasible schedule for the instance of $P||C_{max}$ satisfying condition (6) above if and only if our constructed instance of the problem P2 has a solution (σ,d) satisfying $f(\sigma,d) \leq nP_dC$.

Let $f(\sigma,d) \leq nP_dC$. Then $(\sigma^*,d^*) \leq nP_dC$. Let $\Delta \in \Re$ be such that $0 < \Delta < \min_{j \in J^T}\{T_j\}\Big|_{(\sigma^*,d^*)}$. Then

$$0 \leq f(\sigma^*,d^*+\Delta) - f(\sigma^*,d^*) = \Delta\left(nP_d - \left|J^T\right|P_T\right). \qquad (3.7)$$

Thus,

$$\left|J^T\right| \le \frac{nP_d}{P_T} < 1 \tag{3.8}$$

so that $\left|J^T\right| = 0$ and all jobs are completed early in (σ^*,d^*). Thus, $d^* \ge \max_{j\in J}\{C_j\}\big|_{\sigma^*}$. But the optimality of d^* requires that $d^* \le \max_{j\in J}\{C_j\}\big|_{\sigma^*}$ since $P_d > 0$. It follows that $d^* = \max_{j\in J}\{C_j\}\big|_{\sigma^*}$ and

$$f(\sigma^*,d^*) = nP_d \max_{j\in J}\{C_j\}\big|_{\sigma^*} \le nP_d C. \tag{3.9}$$

Thus σ^* defines a feasible schedule for the $P||C_{max}$ problem with

$$\max_{j\in J}\{C_j\} = \max_{j\in J}\{s_j + t_j\} \le C. \tag{3.10}$$

Conversely, given any schedule σ for $P||C_{max}$ with $\max_{j\in J}\{s_j + t_j\} \le C$, we have the corresponding schedule-due-date combination $(\sigma, \max_{j\in J}\{s_j + t_j\}\big|_{\sigma})$ which satisfies

$$f(\sigma, \max_{j\in J}\{s_j + t_j\}\big|_{\sigma}) = nP_d \max_{j\in J}\{C_j\}\big|_{\sigma} \le nP_d C. \qquad \bullet\bullet$$

3.4 Negative Idle Time Penalty: A Polynomially Solvable Case

We first consider a special case of problem P2 in which $P_I = -\frac{nP_d}{m}$. This problem is not very interesting in itself but provides a useful tool for constructing approximate solutions for the case $P_I \geq 0$. Let $f^-(\sigma,d)$ denote the objective function obtained from $f(\sigma,d)$ by setting $P_I = -\frac{nP_d}{m}$. Then from (3.4):

$$f^-(\sigma,d) = \frac{nP_d}{m}\sum_{i=1}^{m}(d - b_i) + \sum_{j=1}^{n}\left\{P_E E_j + P_T T_j\right\}. \tag{3.11}$$

The following proposition is proved in the appendix.

Proposition 2: There exists an optimal schedule-due-date combination (σ^-, d^-) with respect to $f^-(\sigma,d)$ in which condition (C^-): *The due-date coincides with the completion time of one job on each machine* is satisfied.

If (σ,d) satisfies condition (C^-) in Proposition 2, then

$$f^-(\sigma,d) = \frac{nP_d}{m}\sum_{j\in J^E} t_j + \sum_{j=1}^{n}\left\{P_E E_j + P_T T_j\right\}. \tag{3.12}$$

For any given (σ,d), let the triple (i, k, 0) denote the position of the kth job scheduled early on machine i, and let (i, k, 1) denote the position of the *kth last* job scheduled tardy on machine i, for every $i \in M$. Let $t_{[i,k,\rho]}$ denote the processing time of the job assigned to the position (i, k, ρ). If no job is assigned to the position (i, k, ρ), then set $t_{[i,k,\rho]} = 0$. If (σ,d) satisfies (C^-) in Proposition 2, then the start time for the job

assigned to the position (i, k, ρ) will be

$$s_{[i,k,\rho]} = \left[d - \sum_{k'=k}^{n} t_{[i,k',\rho]} \right](1-\rho) + \left[d + \sum_{k'=k+1}^{n} t_{[i,k',\rho]} \right]\rho \tag{3.13}$$

In addition, b_i will be given by

$$b_i = s_{[i,1,0]} = d - \sum_{k=1}^{n} t_{[i,k,0]} \tag{3.14}$$

and condition (5) yields

$$d = \max_{i \in M} \left\{ \sum_{k=1}^{n} t_{[i,k,0]} \right\}. \tag{3.15}$$

Then we have

$$f^{-}(\sigma,d) = \frac{nP_d}{m} \sum_{i=1}^{m} \sum_{k=1}^{n} t_{[i,k,0]} + \sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{\rho=0}^{1} \left\{ P_E(1-\rho) \sum_{k'=k+1}^{n} t_{[i,k',\rho]} + P_T\rho \sum_{k'=k}^{n} t_{[i,k',\rho]} \right\}$$

$$= \sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{\rho=0}^{1} \lambda_{ik\rho} t_{[i,k,\rho]} \tag{3.16}$$

where

$$\lambda_{ik\rho} = \left[\frac{nP_d}{m} + (k-1)P_E \right](1-\rho) + kP_T\rho. \tag{3.17}$$

We will refer to the $\lambda_{ik\rho}$'s as 'position labels'. Since the sum of the pairwise products of two sets of real numbers is minimized by arranging one set in non-decreasing order and the other set in non-increasing order, we can minimize $f^{-}(\sigma,d)$ by assigning the longest job to the position with the smallest label, the second longest job to the position with the second smallest label, and so on. This yields the optimal sequence σ^{-}; the optimal due-date d^{-} with respect to $f^{-}(\sigma,d)$ can then be obtained from (3.15). Note that this does not determine a unique schedule since $\lambda_{ik\rho} = \lambda_{i'k\rho}$ for every pair of machines $(i, i') \in M \times M$. This algorithm, which we will refer to as algorithm_1, can be accomplished in O(nlogn) time. The following two properties of the resulting schedule will be useful in later analysis and so we state and prove them here. Let $q = m\left\lfloor \frac{nP_d}{mP_T} \right\rfloor$, where $\lfloor x \rfloor$ ($\lceil x \rceil$) denotes the greatest (smallest) integer no larger (smaller) than $x \in \Re$.

Proposition 3: Under (σ^{-}, d^{-}),

$$b_i \le t_{q+1}, \qquad \text{for all } i \in M. \tag{3.18}$$

Proof:

Note that

$$\lambda_{ik\rho} = \lambda_{i'k\rho} \le \lambda_{i'(k+1)\rho} \qquad i, i' = 1, 2, ..., m;\ k = 1, 2, ..., n;\ \rho = 0, 1. \tag{3.19}$$

Thus, for the optimal (σ^{-}, d^{-}), we must have

$$t_{[i,k,\rho]} \ge t_{[i',k+1,\rho]} \qquad i, i' = 1, 2, ..., m;\ k = 1, 2, ..., n;\ \rho = 0, 1.$$

(3.20)

Let $i' \in M$ be such that $b_{i'} = 0$ (recall constraint (5)). Then from (14):

$$d^- = \sum_{k=1}^{n} t_{[i',k,0]}$$

and, for all $i \in M$,

$$b_i = d^- - \sum_{k=1}^{n} t_{[i',k,0]}$$

$$= \sum_{k=1}^{n} \left\{ t_{[i',k,0]} - t_{[i,k,0]} \right\}$$

$$\leq \sum_{k=1}^{n} \left\{ t_{[i',k,0]} - t_{[i',k+1,0]} \right\}$$

$$= t_{[i',1,0]}. \qquad (3.21)$$

Now, we see from (17) that, for all $i \in M$,

$$\lambda_{i10} = \frac{nP_d}{m} \geq P_T \left\lfloor \frac{nP_d}{mP_T} \right\rfloor = \lambda_{i'' \frac{q}{m} 1}, \qquad \text{for all } i'' \in M. \qquad (3.22)$$

Also, for all $i \in M$,

$$\lambda_{ik1} \le \lambda_{i''\frac{q}{m}1}, \qquad \text{for all } i'' \in M \text{ and } k \le \frac{q}{m}. \tag{3.23}$$

Thus, we can choose

$$t_{[i'',1,0]} \le t_{[i,k,1]}, \qquad \text{if } i'' \in M \text{ and } k \le \frac{q}{m}. \tag{3.24}$$

There are q such pairs of (i", k) (with $\left\lfloor \frac{nP_d}{mP_T} \right\rfloor$ for each machine i") which satisfy the condition in (24). Since jobs are indexed in non-increasing order of processing times, we must have

$$b_i \le t_{[i',1,0]} \le t_{q+1}. \qquad \bullet\bullet$$

Proposition 4:

$$f^-(\sigma^-, d^-) \ge P_T \sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j + \frac{nP_d}{m} \sum_{j=q+1}^{n} t_j + \min\{P_E, P_T\} \sum_{j=1}^{n-q-2m} \left\lceil \frac{j}{m} \right\rceil t_{q+2m+j}. \tag{3.25}$$

Proof:

Since $\lambda_{ik0} \geq \lambda_{i10} \geq \lambda_{i\frac{q}{m}1}$ for i = 1, 2, ..., m and k = 1, 2, ..., n, we can, in σ^-, assign the first q jobs to the positions $\{(i, k, 1) : i = 1, 2, ..., m \text{ and } 1 \leq k \leq \frac{q}{m}\}$ according to algorithm_1. The contribution of these jobs to the penalty function is then given by $P_T \sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j$.

We can now write

$$f^-(\sigma^-, d^-) = P_T \sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j + \sum_{i=1}^{m} \sum_{\rho=0}^{1} \sum_{k=\frac{q}{m}\rho+1}^{n} \lambda_{ik\rho} t_{[i,k,\rho]}$$

$$= P_T \sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j + \sum_{i=1}^{m} \sum_{\rho=0}^{1} \sum_{k'=1}^{n} \lambda_{i(k'+\frac{q}{m}\rho)\rho} t_{\left[i,k'+\frac{q}{m}\rho,\rho\right]} \tag{3.26}$$

where

$$\lambda_{i(k'+\frac{q}{m}\rho)\rho} = \left[\frac{nP_d}{m} + P_E(k'-1)\right](1-\rho) + P_T\left(k'+\frac{q}{m}\right)\rho$$

$$\geq \left[\frac{nP_d}{m} + P_E(k'-1)\right](1-\rho) + P_T\left(k'+\frac{nP_d}{mP_T} - 1\right)\rho$$

$$= \frac{nP_d}{m} + (k'-1)\left[P_E(1-\rho) + P_T\rho\right]$$

$$\geq \frac{nP_d}{m} + (k'-1)\min\{P_E, P_T\}. \tag{3.27}$$

Then

$$f^{-}(\sigma^{-},d^{-}) \geq P_T \sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j + \sum_{i=1}^{m} \sum_{\rho=0}^{1} \sum_{k'=1}^{n} \left(\frac{nP_d}{m} + \min\{P_E, P_T\}(k'-1) \right) t_{\left[i,k'+\frac{q}{m}\rho,\rho \right]}. \tag{3.28}$$

This function is minimized by matching the jobs q+1, q+2, ..., n (listed in non-increasing order of processing times) to the positions $\{(i, k'+\left(\frac{q}{m}\right)\rho, \rho)$: k' = 1, 2, ..., n; i $\in$ M; ρ = 1, 0} listed in non-decreasing order of k'. If we do this, then job q+j, j = 1, 2, ..., 2m, will be assigned to a position with weight $\frac{nP_d}{m}$ and job q+2m+j, j = 1, 2, ..., n-q-2m, will be assigned to a position with weight $\left\{\frac{nP_d}{m} + \min\{P_E, P_T\}\left\lceil \frac{j}{2m} \right\rceil\right\}$. Thus we have

$$f^{-}(\sigma^{-},d^{-}) \geq P_T \sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j + \frac{nP_d}{m} \sum_{j=q+1}^{n} t_j + \min\{P_E, P_T\} \sum_{j=1}^{n-q-2m} \left\lceil \frac{j}{m} \right\rceil t_{q+2m+j}.$$

••

3.5 Non-Negative Idle Time Penalty

(i) Positive Due-Date Penalty

We now consider the case $P_I \geq 0$. This, of course, is the more interesting case since it can be more easily applied to a realistic situation. The problem is similar to that considered by Cheng (1989) except that here, the 'zero idle time' constraint is relaxed and a penalty proportional to the idle time is imposed instead. Although the case $P_I = -\frac{nP_d}{m}$ is polynomially solvable, the case $P_I \geq 0$ is NP-hard; however, the solution of the former case provides a good approximation for the solution of the latter one. From (3.4) and (3.11), we see that, for any $(\sigma,d) \in \Pi \times \Re$,

$$f(\sigma,d) = f^{-}(\sigma,d) + \left(P_I + \frac{nP_d}{m}\right)\sum_{i=1}^{m} b_i . \tag{3.29}$$

Thus, constraint (3.5) guarantees that

$$f^{-}(\sigma,d) \leq f(\sigma,d). \tag{3.30}$$

Since (σ^*,d^*) is optimal with respect to $f(\sigma,d)$, we have

$$f^{-}(\sigma^{-},d^{-}) \leq f^{-}(\sigma^*,d^*) \leq f(\sigma^*,d^*) \leq f(\sigma^{-},d^{-}). \tag{3.31}$$

It follows that

$$\frac{f(\sigma^-,d^-)-f(\sigma^*,d^*)}{f(\sigma^*,d^*)} \le \frac{f(\sigma^-,d^-)-f^-(\sigma^-,d^-)}{f^-(\sigma^-,d^-)} = \frac{\left(P_I+\frac{nP_d}{m}\right)\sum_{i=1}^{m} b_i}{f^-(\sigma^-,d^-)}. \quad (3.32)$$

Now applying (5), Proposition 3 and Proposition 4 yields the following proposition:

Proposition 5:

$$\frac{f(\sigma^-,d^-)-f(\sigma^*,d^*)}{f(\sigma^*,d^*)} \le$$

$$\frac{(m-1)\left(P_I+\frac{nP_d}{m}\right)t_{q+1}}{P_T\sum_{j=1}^{q}\left\lceil\frac{j}{m}\right\rceil t_j+\frac{nP_d}{m}\sum_{j=q+1}^{n}t_j+\min\{P_E,P_T\}\sum_{j=1}^{n-q-2m}\left\lceil\frac{j}{2m}\right\rceil t_{q+2m+j}}. \quad (3.33)$$

We also have the following corollary:

Corollary 6:

If $0 \le P_d \le P_T$, then

$$\frac{f(\sigma^-,d^-)-f(\sigma^*,d^*)}{f(\sigma^*,d^*)} \le \frac{2(m-1)\left(1+\frac{mP_I}{nP_d}\right)}{n\left(\frac{P_d}{P_T}\right)^2}. \quad (3.34)$$

Proof:

First note that $q \le n$.

Case 1: If $q = n$, then no jobs are scheduled early in (σ^-, d^-) and so $f^-(\sigma^-, d^-) = f(\sigma^-, d^-)$. From (31), $f(\sigma^-, d^-) = f(\sigma^*, d^*)$, thus (34) holds.

Case 2: If $q < n$, we have, from Proposition 4

$$f^-(\sigma^-, d^-) \ge P_d \left[\sum_{j=1}^{q} \left\lceil \frac{j}{m} \right\rceil t_j + \frac{n}{m} \sum_{j=q+1}^{n} t_j \right]$$

$$\ge \frac{P_d}{m} \sum_{j=1}^{n} j t_j$$

$$\ge \frac{P_d t_{q+1}}{m} \sum_{j=1}^{q+1} j$$

$$= \frac{P_d t_{q+1} (q+1)(q+2)}{2m}$$

$$\ge \frac{n^2 P_d}{2m} \left(\frac{P_d}{P_T} \right)^2 t_{q+1}.$$

Inequality (34) follows by replacing the denominator in Proposition 5 with the last expression above. ••

Thus if we approximate the optimal solution for the case $P_I \geq 0$ with the optimal solution for the case $P_I = -\frac{nP_d}{m}$, the relative error is $O\left(\frac{1}{n}\right)$ as n approaches ∞. So our approximation is asymptotically optimal. Note also that there is no error in the case m = 1. This should be expected since the problem is then reduced to the one solved by Panwalkar et al (1982). The condition $P_d \leq P_T$ is natural here, since in the case of $P_d > P_T$ the optimal solution must have $d^* = 0$ (see appendix for proof) and so coincides with the solution of the minimum average completion time problem $P||\overline{C}$, which can be solved in polynomial time by applying the generalized SPT rule of Conway et al (1967). The case $P_d = 0$ is summarized below.

<u>(ii) Zero Due-Date Penalty</u>

If $P_d = 0$, then the bound in Corollary 6 is no longer useful. The problem is still NP-hard (see appendix for proof) and so it is useful to consider approximate solutions. Let (σ^0, d^0) denote the optimal solution for the case $P_d = 0$ and let $f^0(\sigma, d)$ denote the corresponding objective function. Then applying Proposition 5 yields

Corollary 7:

$$\frac{f^0(\sigma^-, d^-) - f^0(\sigma^0, d^0)}{f^0(\sigma^0, d^0)} \leq \frac{4m(m-1)}{(n-2m)(n-2m+1)}\left[\frac{P_I}{\min\{P_E, P_T\}}\right]\left(\frac{t_1}{t_n}\right)$$

where (σ^-, d^-), in the present case, refers to the schedule-due-date obtained by applying algorithm_1 with $P_d = 0$.

Proof:

Retain only the last term in the lower bound for $f^{-}(\sigma^{-},d^{-}) \geq f^{0}(\sigma^{0},d^{0})$ and use the relations:

(a) $q = 0,$

(b) $$\sum_{j=1}^{n-2m} \left\lceil \frac{j}{2m} \right\rceil t_{2m+j} \geq t_n \sum_{j=1}^{n-2m} \frac{j}{2m} = \frac{(n-2m)(n-2m+1)t_n}{4m}. \qquad \bullet\bullet$$

Thus the relative error is $O\left(\frac{t_1}{n^2 t_n}\right)$ as n approaches ∞. We show in the appendix that if the processing times are uniformly and independently distributed over the interval [0, 1], then

$$\lim_{n\to\infty} \Pr\left[\frac{t_1}{n^2 t_n} > n^{\alpha-1}\right] = 0, \qquad \text{if } \alpha > 0. \qquad (3.35)$$

It follows that our approximation is asymptotically optimal in probability.

3.6 Zero Idle Time Constraint

This is the problem considered by Cheng (1989). We can recover this version of the problem by setting $P_I > nP_E$ in P1. This is because the optimal solution for the case $P_I > nP_E$ must have zero idle time (see appendix for proof). When we consider approximate solutions, however, the two problems must be treated differently since the approximate solution (σ^-, d^-) to P1 may not satisfy the zero idle time constraint. We can, however, alter (σ^-, d^-) slightly so as to make it satisfy the constraint and still retain a relative error of $O\left(\frac{1}{n}\right)$.

Let $(\sigma^\infty, d^\infty)$ denote the optimal solution subject to the zero idle time constraint. Given any schedule $\sigma \in \Pi$, let σ_f be the sequence obtained from σ by shifting the start time of each job on machine i backward in time by an amount $b_i|_\sigma$ for every machine $i \in M$, so that all machines begin processing at time $t = 0$ under σ_f. Then we have the following proposition.

Proposition 8:

$$f(\sigma_f^-, d^-) - f(\sigma^\infty, d^\infty) \le (m-1)(P_E + P_T)\left[1 + \min\left\{\frac{t_{q+1}}{t_n}, \left\lceil \frac{n}{m} \right\rceil\right\}\right] t_{q+1}. \quad (3.36)$$

Proof:

Let $n_i(b_i)$ be the number of jobs on machine i which are tardy under (σ^-, d^-) but early under (σ_f^-, d^-). Then

$$f^{-}(\sigma_f^{-},d^{-}) - f^{-}(\sigma^{-},d^{-}) =$$

$$\sum_{i=1}^{m}\left\{\sum_{k=1}^{n_i(b_i)}\left[\frac{nP_d}{m}+P_E\left(\left|J_i^E\right|+k-1\right)-P_T\left(\left|J_i^T\right|-k+1\right)\right]t_{\left[i,\left|J_i^T\right|-k+1,1\right]}\right.$$

$$+$$

$$\left.\left[\frac{nP_d}{m}+P_E\left(\left|J_i^E\right|+n_i(b_i)\right)-P_T\left(\left|J_i^T\right|-n_i(b_i)\right)\right]\left[b_i-\sum_{k=1}^{n_i(b_i)}t_{\left[i,\left|J_i^T\right|-k+1,1\right]}\right]\right\}$$

$$\leq \sum_{i=1}^{m}\left[\frac{nP_d}{m}+P_E\left(\left|J_i^E\right|+n_i(b_i)\right)-P_T\left(\left|J_i^T\right|-n_i(b_i)\right)\right]b_i$$

$$= \sum_{i=1}^{m}\left\{\left[\frac{nP_d}{m}+P_E\left(\left|J_i^E\right|-1\right)-P_T\left(\left|J_i^T\right|+1\right)\right]b_i+(P_E+P_T)(n_i(b_i)+1)b_i\right\}.$$

But the optimality of σ^{-} with respect to $f^{-}(\sigma,d)$ implies that

$$\left.\frac{nP_d}{m}+P_E\left(\left|J_i^E\right|-1\right)-P_T\left(\left|J_i^T\right|+1\right)\right|_{(\sigma^{-},d^{-})} \leq 0, \qquad \text{for all } i \in M, \quad (3.37)$$

since otherwise reassigning the last early job on machine i to the first tardy position on i would decrease the objective function value. Thus

$$f^{-}(\sigma_f^{-},d^{-}) - f^{-}(\sigma^{-},d^{-}) \leq (P_E+P_T)\sum_{i=1}^{m}(n_i(b_i)+1)b_i. \qquad (3.38)$$

But $b_i \le t_{q+1}$ (Proposition 3) and thus

$$n_i(b_i) \le \frac{b_i}{t_n} \le \frac{t_{q+1}}{t_n}. \tag{3.39}$$

Also, $n_i(b_i) \le \left|J_i^T\right| \le \left\lceil \frac{n}{m} \right\rceil$ and $b_i = 0$ for some $i \in M$. Thus we have

$$f^-(\sigma_f^-, d^-) - f^-(\sigma^-, d^-) \le (m-1)(P_E + P_T)\left[1 + \min\left\{\frac{t_{q+1}}{t_n}, \left\lceil \frac{n}{m} \right\rceil\right\}\right] t_{q+1}. \tag{3.40}$$

Also, from (29), with zero idle time,

$$f^-(\sigma_f^-, d^-) = f(\sigma_f^-, d^-) \ge f(\sigma^\infty, d^\infty) \ge f^-(\sigma^\infty, d^\infty) \ge f^-(\sigma^-, d^-) \tag{3.41}$$

and so

$$f^-(\sigma_f^-, d^-) - f(\sigma^\infty, d^\infty) \le f^-(\sigma_f^-, d^-) - f^-(\sigma^-, d^-)$$

$$\le (m-1)(P_E + P_T)\left[1 + \min\left\{\frac{t_{q+1}}{t_n}, \left\lceil \frac{n}{m} \right\rceil\right\}\right] t_{q+1}. \quad \bullet\bullet$$

We also have the following corollary.

Corollary 9:

$$\frac{f(\sigma_f^-,d^-)-f(\sigma^\infty,d^\infty)}{f(\sigma^\infty,d^\infty)} \le \frac{(P_E+P_T)\left(1+\frac{2m}{n}\right)\frac{2(m-1)}{n}}{P_d\left(\frac{P_d}{P_T}\right)^2}.$$

Thus the relative error incurred in using (σ_f^-,d^-) as an approximation for (σ^∞,d^∞) is of $O\left(\frac{1}{n}\right)$ so that our approximation is asymptotically optimal as n approaches ∞ provided that $P_d > 0$.

3.7 Zero Due-Date Penalty with Zero Idle Time Constraint

The case $P_d = 0$ presents some difficulty again since the bound in Corollary 9 can no longer be applied. Applying Proposition 4 and Proposition 8 yields

Proposition 10:

$$\frac{f^0(\sigma_f^-,d^-)-f^0(\sigma^0,d^0)}{f^0(\sigma^0,d^0)} \le \frac{4(m-1)(n+2m)}{(n-2m)(n-2m+1)}\left[\frac{P_E+P_T}{\min\{P_E,P_T\}}\right]\left(\frac{t_1}{t_n}\right)$$

where (σ^-,d^-), in the present case, refers to the schedule-due-date obtained by applying algorithm_1 with $P_d = 0$.

Proof:

The proof is similar to that of Corollary 7. ••

Thus the relative error is $O\left(\frac{t_1}{nt_n}\right)$ as n approaches ∞. If the processing times are uniformly and independently distributed over the interval [0, 1], then

$$\lim_{n\to\infty} \Pr\left[\frac{t_1}{nt_n} > n^{\alpha}\right] = 0, \qquad \text{if } \alpha > 0. \tag{3.42}$$

Thus, even though we may not have asymptotic optimality in this case, we can reasonably expect that the relative error will remain finite as n approaches ∞.

3.8 Computational Experiences

Tables 3.1 - 3.4 show the results of several test runs of algorithm_1 with $P_I = P_T = P_E = 1$ for the four cases below. Note that only the relative magnitude of these parameters is significant so that we can set the largest of them equal to 1. Setting $P_T = P_E$ discourages early and tardy jobs equally, though other choices for these parameters may be desirable depending on the application. P_d is bounded between 0 and 1 since values outside this range yield degenerate problems as discussed below. Thus we set its value in the middle of this range for our tests. Setting P_I at a higher value would discourage idle time more. Our results will all hold provided P_I does not grow with n or grows slower than linearly with n. We set:

(I) $P_d = 0.5$ without zero idle time constraint,

(II) $P_d = 0.5$ with zero idle time constraint,

(III) $P_d = 0$ without zero idle time constraint,

(IV) $P_d = 0$ with zero idle time constraint.

All instances are generated with jobs having processing times uniformly distributed in the interval (0, 1). We list the parameters m and $\frac{n}{m}$ for each instance as well as the *a priori* upper bounds (from Corollaries 6, 7 and 9 and Proposition 10) and the following *posterior* upper bounds on the relative error:

1) $\frac{f(\sigma^-,d^-)-f^-(\sigma^-,d^-)}{f^-(\sigma^-,d^-)}$, for cases without zero idle time constraint (i.e.cases (I) and (III)),

2) $\frac{f(\sigma_f^-,d^-)-f^-(\sigma^-,d^-)}{f^-(\sigma^-,d^-)}$, for cases with zero idle time constraint (i.e. cases (II) and (IV)).

Tables 3.5 -3. 8 list the ratios of the *a priori* upper bound to the *posterior* upper bound for the four cases above. In case (I) the ratios range between 11 and 32 with a mean of 20. For case (II) they range between 18 and 79 with a mean of 59. Although the relative errors for the case of uniformly distributed processing times do not come close to the *a priori* bounds, the ratio of the two bounds appears to be relatively stable and there seems to be no discernible pattern in the variation of these ratios with n and m. Thus the *a priori* upper bound can provide a reasonable rough estimate of the error if this ratio is taken into account.

The ratios for cases (III) and (IV) (i.e. $P_d = 0$), however, are very large and fluctuate wildly. Hence, the *a priori* bounds for these two cases cannot provide any reasonable estimate of the actual error. However, the *posterior* bounds are reasonably small, especially in case (III) in which the error is less than 3 per cent for $\frac{n}{m} \geq 10$. Even in case (IV) the *posterior* bounds decrease steadily as $\frac{n}{m}$ increases. But this is not predicted explicitly in the *a priori* bound. The large error with the *a priori* bounds seems to be caused by the factor $\frac{t_1}{t_n}$ which can be extremely large. If $P_d = 0$, the longest 2m jobs do not contribute to the lower bound in Proposition 4 whereas they do contribute to the upper bound in Proposition 8. Thus the relative error bound, which is the ratio of these two, can be very large if these 2m longest jobs are extremely long compared to the other n - 2m shorter jobs. Thus, even though the *a priori* bounds do not indicate so, the computational results show that our algorithm provides a good heuristic for the case $P_d = 0$ with the processing times uniformly distributed and, probably, in most cases in which the longest 2m jobs are not extremely long compared with the n - 2m shorter jobs.

3.9 Conclusion

We have described an approximate solution for an NP-hard problem concerning the assignment of a common due-date and job scheduling on parallel machines. Our basic idea was that the difficulty of the problem was due to the dependence of the objective function on the due-date, which can be viewed as the maximum completion time of the early jobs. The problem of minimizing the maximum completion time on parallel machines is known to be NP-hard. Setting $P_I = -\frac{nP_d}{m}$ has the effect of approximating the maximum completion

time of the early jobs, which corresponds to the maximum of the 'early workload' on each machine, by the average 'early workload' of the machines. This approach leads naturally to *a priori* relative error bounds which are easily calculated and which appear to be reasonably small for large enough values of $\frac{n}{m}$. Our approximation was shown to be asymptotically optimal when the due-date cost is non-zero. When the due date cost is zero and no zero idle time constraint is imposed the approximation was shown to be asymptotically optimal in probability for uniformly distributed processing times. Our test cases indicate that the approximation provides a good heuristic for problems with uniform processing times in all of the four cases considered.

3.10 Appendix

(i) Proof of Proposition 2:

Let (σ,d) be optimal with respect to $f^{-}(\sigma,d)$ and suppose that d does not coincide with the completion time of any job on machine i. Let σ' be the schedule derived from σ by shifting each job on machine i forward in time by an amount Δ and let $d' = d$. Let C_j and C'_j denote the completion times of job j under (σ,d) and (σ',d) respectively. So

$$C'_j = C_j + \Delta, \qquad \text{for all } j \in J_i^E \cup J_i^T.$$

Then we have

$$f^{-}(\sigma',d) - f^{-}(\sigma,d) = \Delta\left[\frac{-nP_d}{m} + \left|J_i^T\right|P_T - \left|J_i^E\right|P_E\right],$$

provided that

$$d - \min_{j \in J_i^T}\{C_j\} \le \Delta \le d - \max_{j \in J_i^E}\{C_j\}. \qquad \text{(A1)}$$

Since $f^{-}(\sigma,d)$ is optimal, we must have

$$\Delta\left[\frac{-nP_d}{m} + \left|J_i^T\right|P_T - \left|J_i^E\right|P_E\right] \ge 0. \qquad \text{(A2)}$$

But since d does not coincide with the completion time of any job on i, Δ can, according to (A1), take on both positive and negative values. Thus

$$\Delta < 0 \Rightarrow \left[\frac{-nP_d}{m} + \left|J_i^T\right|P_T - \left|J_i^E\right|P_E\right] \le 0$$

$$\Delta > 0 \Rightarrow \left[\frac{-nP_d}{m} + \left|J_i^T\right|P_T - \left|J_i^E\right|P_E\right] \ge 0.$$

Therefore,

$$\left[\frac{-nP_d}{m} + \left|J_i^T\right|P_T - \left|J_i^E\right|P_E\right] = 0$$

and so

$$f^{-}(\sigma',d) = f^{-}(\sigma,d)$$

and $f^{-}(\sigma',d)$ is optimal provided that Δ satisfies (A1). But then we can take $\Delta = d - \max_{j \in J_i^E}\{C_j\}$, in which case the due-date d will coincide with the completion time $\max_{j \in J_i^E}\{C'_j\}$ in σ'. By a repeated application of the above argument to each machine $i \in M$, we can construct an optimal solution in which the due-date coincides with the completion time of one job on each machine.

••

<u>(ii) Proof of the claim</u>

If $P_d > P_T$, then $d^* = 0$.

Proof:

Let $P_d > P_T$ and suppose that $d^* > 0$. Let $\Delta \in \Re$ be such that

$$0 < \Delta < d^* - \max_{j \in J^E}\{C_j\}.$$

Also let $k = \left|\left\{j : C_j = d^*\right\}\right|$. Then

$$0 \le f(\sigma^*,d^*-\Delta) - f(\sigma^*,d^*) = \Delta\left[P_T\left(\left|J^T\right| + k\right) - P_E\left(\left|J^E\right| - k\right) - nP_d\right]$$

$$\le \Delta n\left(P_T - P_d\right) < 0.$$

Thus we have a contradiction. ••

(iii) Proof of the claim

$$\lim_{n\to\infty} \Pr\left[\frac{t_1}{n^2 t_n} > n^{\alpha-1}\right] = 0, \qquad \alpha > 0.$$

Let $f(t_1,t_n)$ be the joint density function of t_1 and t_n, both of which are uniformly distributed over the interval [0, 1]. Then

$$f(t_1,t_n) = n(n-1)(t_1 - t_n)^{n-2}.$$

Let $y_1 = \frac{t_n}{t_1}$, $y_2 = t_1$. Then

$$f(y_1,y_n) = n(n-1)y_2^{n-1}(1-y_1)^{n-2}.$$

Hence, the marginal density of y_1 is given by

$$f(y_1) = (n-1)(1-y_1)^{n-2}$$

and

$$\Pr\left[\frac{t_1}{n^2 t_n} > n^{\alpha-1}\right] = \Pr\left[\frac{t_n}{t_1} < n^{-(1+\alpha)}\right] = 1-\left(1-n^{-(1+\alpha)}\right)^{n-1} \xrightarrow{n\to\infty} 0. \bullet\bullet$$

(iv) Proof of the claim

If $P_I > nP_E$, then $I\big|_{\sigma^*} = 0$.

Proof:

Consider the optimal sequence σ^* and suppose that $I > 0$ in this sequence. Then we must have $b_i > 0$ for some $i \in M$. Let $\Delta \in \Re$ be such that

$$0 < \Delta < \min\left\{b_i, \left(T_j : j \in J_i^T\right)\right\}.$$

Let σ' denote the schedule obtained from σ^* by shifting each job on machine i backward in time by an amount Δ. Then

$$0 \le f(\sigma',d^*) - f(\sigma^*,d^*) = \Delta\left[P_E\left|J_i^E\right| - P_I - P_T\left|J_i^T\right|\right] \le \Delta(nP_E - P_I) < 0$$

and we have a contradiction. ••

<u>(v) Proof of the claim</u>

The problem P1 with $P_d = 0$ is NP-hard.

Proof:

We prove The NP-completeness of the recognition version of P1 with $P_d = 0$ by a reduction from the Even-Odd-Partition (EOP) problem, which has been shown to be NP-complete by Garey et al (1988).

<u>Even Odd Partition (EOP) Problem</u>

Given a set of 2k integers $S = \{s_1, s_2, ..., s_{2k}\}$ where $s_1 \geq s_2 \geq ... \geq s_{2k}$, does there exist a subset A of S consisting of exactly k integers from S such that $\sum_{s \in A} s = \sum_{t \in S-A} t$ and A contains exactly one integer from each set $\{s_{2i-1}, s_{2i}\}$, i = 1, 2, ..., k?

Given an instance of the EOP problem, we construct an instance of P1 with $P_d = 0$, $P_T > nP_E$, m = 2, n = 2k and $t_i = s_i$, i = 1, 2, ..., 2k. Then P1 has a solution (σ,d) satisfying

$$f(\sigma,d) \leq P_E \sum_{j=1}^{n} \left\lfloor \frac{j}{2} \right\rfloor t_j \tag{A3}$$

if and only if the EOP problem has a solution.

The proof consists of three parts.

(1) The optimal solution (σ^*,d^*) to P1 has no tardy jobs.

Suppose to the contrary that some jobs on machine i are tardy under (σ^*,d^*). Let $\Delta = \min_{j \in J_i^T} \{T_j\}$ and let σ' be the schedule obtained from σ^* by shifting all jobs on machine i backward in time by an amount Δ. Then

$$0 \leq f(\sigma',d^*) - f(\sigma^*,d^*) = \Delta\left[P_E\left|J_i^E\right| - P_I - P_T\left|J_i^T\right|\right] \leq \Delta(nP_E - P_I) < 0$$

and so we have a contradiction.

(2) If all jobs are early in (σ,d), then $f(\sigma,d) \geq P_E \sum_{j=1}^{n} \left\lfloor \frac{j}{2} \right\rfloor t_j$ and the equality holds only if $I = 0$ and the jobs $2i - 1$ and $2i$ are both scheduled ith on their respective machines for some ordering of the jobs $j = 1, 2, \ldots, n$. (There may be more than one order which satisfies $t_1 \geq t_2 \geq \ldots \geq t_n$).

Since all jobs are early, we have

$$f(\sigma,d) = P_I I + P_E \sum_{i=1}^{2} \sum_{k=1}^{n} (k-1) t_{[i,k,0]}.$$

The second term is minimized by assigning the two longest jobs to the positions with $k = 1$, the next two longest jobs to the positions with $k = 2$, and so on. The resulting minimum value of the second term is given by $P_E \sum_{j=1}^{n} \left\lfloor \frac{j}{2} \right\rfloor t_j$. Clearly, if this is the total penalty for some sequence, we must have $I = 0$ and the sequence must be the one described above which minimizes the second term.

(3) Suppose that (σ,d) satisfies (A3). Then (σ^*,d^*) must also satisfy (A3). But (σ^*,d^*) has no tardy jobs and so must be the schedule described in (2) above. Thus $I|_{\sigma^*} = 0$. But then $b_1 = b_2 = 0$ and $\sum_{k=1}^{n} t_{[1,k,0]} = d^* = \sum_{k=1}^{n} t_{[2,k,0]}$. Thus we have a solution to EOP given by $A = \{s_j\colon j \text{ is assigned to machine 1 in } (\sigma^*,d^*)\}$.

Conversely, given any partition A of EOP, we can find a sequence σ^* satisfying (A3) by assigning the jobs $\{j\colon s_j \in A\}$ to machine 1 and the jobs $\{j\colon s_j \in S - A\}$ to machine 2, in non-increasing order of processing times, and setting $d^* = \sum_{s_j \in A} t_j$. ••

Part Two:
Priority Scheduling

Chapter 4

Hierarchical Minimization of Flow Times

4.1 Introduction

Research into scheduling problems involving multiple criteria has begun only recently. Dileepan and Sen (1988) have provided a review of scheduling problems involving two criteria and a single machine. They divide the problems considered so far into two groups: Class-1 problems and Class-2 problems. Class-2 problems can be stated as follows:

$$\text{minimize } Z(\sigma) = f(C_1(\sigma), C_2(\sigma))$$

$$\text{subject to } \sigma \in \Omega = \{\text{set of all possible schedules}\}$$

where $C_1(\sigma)$ and $C_2(\sigma)$ are performance measures such as flowtime, tardiness/Earliness, set-up cost, etc. The approach here is to combine the performance measures into a single analytical expression which is to be minimized. The problem considered in the previous chapter is similar to this type of problem. It involves four performance criteria: earliness, tardiness, due-date, and idle time and it seeks to minimize a linear combination of these.

Class-1 problems are more closely related to the problem we consider here and can be stated as follows:

$$\text{minimize } Z(\sigma) = f(C_1(\sigma))$$

$$\text{subject to } \sigma \in \Omega = \{\text{set of all possible schedules satisfying } g(C_2(\sigma)) = c\}$$

That is, one of the two criteria acts as the objective function and the other acts as a constraint. Smith (1956) provided the earliest pioneering work by considering the problem with mean flowtime as $f(C_1(\sigma))$, maximum tardiness as $g(C_2(\sigma))$ and zero as the value of c. Emmons (1975) considered the problem with total flowtime as

$f(C_1(\sigma))$, $\max_i\{w_i T_i(\sigma)\}$ (where w_i= weight associated with job i and $T_i(\sigma)$= tardiness of the job i) as $g(C_2(\sigma))$ and the following value for c:

$$c = \min_{\sigma\in\Omega}\left\{\max_i\{w_i T_i(\sigma)\}\right\}$$

This approach imposes a hierarchical structure onto the problem. We first must choose schedules which minimize $\max_i\{w_i T_i(\sigma)\}$ and then choose, from among those, a schedule with the smallest flowtime. Emmons (1987) considers a parallel machine scheduling problem in the same vein. He uses the sum of absolute deviation of completion times from a common due-date as the primary criterion:

$$g(C_2(\sigma)) = \sum_{j=1}^{n}\left|c_j - d\right|$$

where c_j= completion time of the job j and d=due-date. He uses the criteria of minimum makespan and machine occupancy for the secondary criterion $f(C_1(\sigma))$.

This hierarchical structure is very similar to that encountered in linear goal programming problems. In that context, we have a multiple criteria hierarchical objective. Given a feasible set $S \subseteq \Re^n$ and a set $f_1, f_2, \ldots, f_k$ of k linear objective functions ($f_j: S \rightarrow \Re$, $j = 1,2,\ldots,k$) the objective is first to choose schedules which minimize f_1,then to choose from among those some which minimize f_2, and so on.

The problem which we consider here is a hierarchical multiple criteria scheduling problem on parallel machines. The jobs are divided into p priority classes, and the total flowtime of the class with the kth highest priority is the kth highest (kth most primary) objective criterion.

This type of problem could arise in any environment (job shop scheduling, computer job scheduling etc.) where a priority is assigned to each job. The interpretation here is that jobs in a higher priority class are much more"important" than jobs in lower priority classes since we can make no improvement in the scheduling of the lower priority jobs if it compromises (even minutely) the scheduling of the higher priority jobs. In the following chapter we will explore a somewhat less stringent interpretation of job priority.

If only one priority class exists, the problem reduces to that of minimizing total flowtime on parallel machines. That problem is solved by the generalized SPT rule of Conway et al (1967) which consists of scheduling the jobs in non-decreasing order of

processing times and assigning each job to the earliest available machine.

The algorithm which solves the multi-priority problem is a further generalized SPT rule where the jobs are ordered by priority first and then by processing time.

4.2 Problem Formulation

Consider a set J= { 1,2,....n) of n jobs to be scheduled on a set M= { 1,2,...,m } of m machines . Suppose that each job in J is assigned a priority $P(j) \in \{1,2,\ldots,p\}$ via the function $P: J \rightarrow \{1,2,\ldots,p\}$. We assume that the most "important" jobs are assigned smaller values of P. Thus the highest priority or most important jobs are those in the priority class

$$\Pi_1 = \{j \in J | P(j) = 1\} . \tag{4.1}$$

The function P divides the jobs into the p priority classes $\Pi_1, \Pi_2, \ldots, \Pi_p$ where

$$\Pi_k = \{j \in J | P(j) = k\} \tag{4.2}$$

With each job $j \in J$ is also associated a processing time $t_j > 0$. We define a schedule of the jobs in J to be any function of the form $\sigma: J \rightarrow M \times \Re$ where

$$\sigma(j) = (m_j, s_j) \tag{4.3}$$

and

(a) $s_j \geq 0$

(b) $\quad s_{j'} + t_{j'} \le s_j \quad \text{or} \quad s_{j'} \ge s_j + t_j \qquad \text{if } m_j = m_{j'}$ (4.4)

m_j represents the machine on which the job j is to be processed and s_j represents the start time of that job. Condition (a) requires that all jobs begin processing after time zero and (b) ensures that no two jobs are processed simultaneously on a single machine. Let $c_j = s_j + t_j$ denote the completion time of the job j. We will denote the total completion time (under σ) of jobs in the priority class Π_k by:

$$f_k(\sigma) = \sum_{j \in \Pi_k} c_j . \tag{4.5}$$

Let S be any set and let $f: S \rightarrow \Re$ be any real function defined on S. Then we define the restriction operator R_f according to

$$R_f S \equiv \{s \in S | f(s) \le f(s') \quad \forall s' \in S\} \tag{4.6}$$

Thus R_f restricts S to the subset of S on which the function f is a minimum. We adopt the usual convention that in the case of products of such operators the rightmost operator acts first on the set. These restriction operators can be used to effectively express problems with multi-criteria objectives, especially if the criteria are hierarchical. For example, a linear goal programming problem with ℓ linear objective functions $f_1, f_2, \ldots f_\ell$ (where f_1 has the highest priority) and a feasible set S can be expressed as the problem of finding any member of the set $R_{f_\ell} R_{f_{\ell-1}} \ldots R_{f_1} S$ (note that this set is empty only if S is empty).

Let S denote the set of all possible schedules for the jobs in J. Then our problem, which we will denote by P3 is:

P3: Find a member of the set

$$O = R_{f_p} R_{f_{p-1}} \ldots R_{f_1} S \qquad \bullet\bullet$$

Thus the problem is first to choose the schedules which minimize the total completion time of the highest priority jobs, then to choose among these by applying the total completion time of the 2nd highest priority jobs as a secondary criterion, and so on. Any schedule $\sigma \in O$ will be called an optimal schedule. Let $m_j(\sigma)$ and $s_j(\sigma)$ denote the machine and start time (respectively) of the job j under the schedule σ. We now prove some fundamental results which will lead to the solution of PI.

4.3 Results

We first prove that for an optimal schedule, the total completion time of any priority class cannot be decreased without increasing the total completion time of some higher priority class.

Lemma 1: If there exist $k \in \{1,2,\ldots p\}$ and $\sigma, \sigma' \in S$ such that:

(a) $\quad f_{k'}(\sigma') \leq f_{k'}(\sigma) \quad \forall k' < k$

and

(b) $f_k(\sigma') < f_k(\sigma)$

then σ is not optimal.

Proof:

Let σ be optimal. Note that

(i) $f_1(\sigma') \leq f_1(\sigma) \Rightarrow \sigma' \in R_{f_1}S$

(ii) $\sigma' \in R_{f_{\ell-1}} \ldots R_{f_1}S$ and $f_\ell(\sigma') \leq f_\ell(\sigma) \Rightarrow \sigma' \in R_{f_\ell} \ldots R_{f_1}S$

Therefore, using (a) and the principle of induction:

$\sigma' \in R_{f_{k-1}} \ldots R_{f_1}S.$

Since σ is optimal, $\sigma \in R_{f_k} \ldots R_{f_1}S$. Therefore $\sigma \in R_{f_{k-1}} \ldots R_{f_1}S$ and $f_k(\sigma) \leq f_k(\sigma')$ $\forall \sigma' \in R_{f_{k-1}} \ldots R_{f_1}S$. This contradicts (b). ••

Lemma2: No optimal schedule has idle time inserted between jobs.

Proof:

Let σ be optimal and suppose that, under σ, there is an idle time of A inserted between jobs i and j which are on the same machine $m_i(\sigma) = m_j(\sigma)$. Assume, without loss of generality, that $s_i(\sigma) < s_j(\sigma)$.

Let σ' be the schedule obtained from σ by shifting backward in time by the amount

Δ all jobs which follow j on $m_j(\sigma)$. Then:

$$c_l(\sigma') = \begin{cases} c_l(\sigma) - \Delta & \text{if } m_l(\sigma) = m_j(\sigma) \text{ and } s_l(\sigma) \geq s_j(\sigma) \\ c_l(\sigma) & \text{otherwise} \end{cases} \tag{4.8}$$

so that $c_l(\sigma') \leq c_l(\sigma) \quad \forall l \in J$ and $c_j(\sigma') < c_j(\sigma)$. Thus

$$f_k(\sigma') \leq f_k(\sigma) \quad \forall k < P(j) \tag{4.9}$$

and

$$f_{P(j)}(\sigma') < f_{P(j)}(\sigma) \tag{10}$$

Therefore, by *Lemma 1*, σ is not optimal. ••

The following two lemmas establish that higher priority jobs are scheduled earlier than lower priority jobs in an optimal schedule.

Lemma 3: If σ is optimal, then

$$P(i) < P(j) \Rightarrow s_i(\sigma) \leq s_j(\sigma)$$

if $m_i(\sigma) = m_j(\sigma)$. i.e. If two jobs are scheduled on the same machine under an optimal schedule then the higher priority job must be scheduled first.

Proof:

Let σ be optimal and suppose that $m_i(\sigma) = m_j(\sigma)$, $P(i) < P(j)$and $s_i(\sigma) > s_j(\sigma)$. Let $\ell \in J$be the earliest job scheduled after j on $m_j(\sigma)$ which has a higher priority than P(j).i.e.

$$S_\ell(\sigma) = \min_{\substack{m_k(\sigma)=m_j(\sigma) \\ P(k)<P(j) \\ s_k(\sigma)>s_j(\sigma)}} \{s_k(\sigma)\} \tag{4.11}$$

Let σ' be the schedule obtained from σ by rescheduling ℓ to the position immediately before j.Then

$$s_k(\sigma') = \begin{cases} s_j(\sigma) & \text{if } k = \ell \\ s_k(\sigma) + t_\ell & \text{if } m_k(\sigma) = m_j(\sigma) \text{ and } s_j(\sigma) \le s_k(\sigma) < s_\ell(\sigma). \\ s_k(\sigma) & \text{otherwise} \end{cases} \tag{4.12}$$

By definition of ℓ we have:

$$P(j)>P(\ell)$$

and

$$s_j(\sigma) \le s_k(\sigma) < s_\ell(\sigma) \Rightarrow P(k) \ge P(j) \Rightarrow P(k) \ge P(\ell).$$

Therefore, according to (4.12)

$$s_k(\sigma') = s_k(\sigma) \quad \forall k \ni P(k) < P(\ell).$$

Thus,

$$f_k(\sigma') = f_k(\sigma) \quad \forall k < P(\ell) \tag{4.13}$$

and since $s_l(\sigma') = s_l(\sigma) - t_j < s_l(\sigma)$, we have

$$f_{P(\ell)}(\sigma') < f_{P(\ell)}(\sigma) \tag{4.14}$$

Thus, by *Lemma 1,* σ is not optimal. ••

Lemma 4: If σ is optimal and $i, j \in J$ then

$P(i) < P(j) \Rightarrow s_i(\sigma) \leq s_j(\sigma)$.

(note this does not require $m_i(\sigma) = m_j(\sigma)$ as in *Lemma 3)*

Proof:

Let σ be optimal and suppose that P(i)<P(j). Let σ' be the schedule obtained from σ by reassigning i to the position of j , leaving the jobs which follow i as they are, and shifting j and all jobs which follow it forward by an amount t_i.i.e.:

$$s_k(\sigma') = \begin{cases} s_j(\sigma) & \text{if } k = i \\ s_k(\sigma) + t_i & \text{if } m_k(\sigma) = m_j(\sigma) \text{ and } s_k(\sigma) \geq s_j(\sigma) \\ s_k(\sigma) & \text{otherwise} \end{cases} \tag{4.15}$$

The jobs which follow j must (by the converse of *Lemma 3)* have the same or lower priority than j and thus must have a strictly lower priority that P(i). Thus the job i is the only job with priority equal to or higher than P(i) which has a different start time under σ' than it does under σ. Therefore

(1) $$f_k(\sigma') = f_k(\sigma) \quad \text{if } k < P(i) \tag{4.16}$$

and

(ii) $$f_{P(i)}(\sigma') - f_{P(i)}(\sigma) = s_i(\sigma') - s_i(\sigma) = s_j(\sigma) - s_i(\sigma). \tag{4.17}$$

But the optimality of σ, together with *Lemma 1*, require that

$$f_{P(i)}(\sigma') - f_{P(i)}(\sigma) \geq 0 \tag{4.18}$$

and thus

$$s_j(\sigma) \geq s_i(\sigma) \qquad \bullet\bullet$$

We now define two swapping operations which we will need to use in the upcoming proofs.

Definition 1: Given any schedule σ and two jobs $i, j \in J$, we define σ_{ij} to be the schedule obtained from σ by interchanging the positions of the two jobs and shifting the jobs on $m_i(\sigma)$ and $m_j(\sigma)$ which follow i and j forward or back as necessary, to eliminate idle time

or overlap. i.e.

$$m_k(\sigma') = \begin{cases} m_j(\sigma) & \text{if } k = i \\ m_i(\sigma) & \text{if } k = j \\ m_k(\sigma) & \text{otherwise} \end{cases} \tag{4.19}$$

and

$$s_k(\sigma') = \begin{cases} s_j(\sigma) & \text{if } k = i \\ s_i(\sigma) & \text{if } k = j \\ s_k(\sigma) + t_j - t_i & \text{if } m_k(\sigma) = m_i(\sigma) \text{ and } s_k(\sigma) > s_i(\sigma) \\ s_k(\sigma) + t_i - t_j & \text{if } m_k(\sigma) = m_j(\sigma) \text{ and } s_k(\sigma) > s_j(\sigma) \\ s_k(\sigma) & \text{otherwise} \end{cases} \tag{4.20}$$

••

Definition 2: Given any schedule σ and two jobs $i, j \in J$, we define σ'_{ij} to be the schedule obtained from σ by the following operation:

<u>step 1:</u> Remove i and all jobs scheduled after i from $m_i(\sigma)$ and remove j and all jobs scheduled after j from $m_j(\sigma)$.

<u>step 2</u>: Reschedule the jobs removed from $m_i(\sigma)$ onto $m_j(\sigma)$, beginning at time $s_j(\sigma)$, and preserving their order.

<u>step 3</u>: Reschedule the jobs removed from $m_j(\sigma)$ onto $m_i(\sigma)$, beginning at time $s_i(\sigma)$, and preserving their order.

Then we have

$$m_k(\sigma'_{ij}) = \begin{cases} m_j(\sigma) & \text{if } m_k(\sigma) = m_i(\sigma) \text{ and } s_k(\sigma) \geq s_i(\sigma) \\ m_i(\sigma) & \text{if } m_k(\sigma) = m_j(\sigma) \text{ and } s_k(\sigma) \geq s_j(\sigma) \\ m_k(\sigma) & \text{otherwise} \end{cases} \tag{4.21}$$

and

$$s_k(\sigma'_{ij}) = \begin{cases} s_k(\sigma) + s_j(\sigma) - s_i(\sigma) & \text{if } m_k(\sigma) = m_i(\sigma) \text{ and } s_k(\sigma) \geq s_i(\sigma) \\ s_k(\sigma) + s_i(\sigma) - s_j(\sigma) & \text{if } m_k(\sigma) = m_j(\sigma) \text{ and } s_k(\sigma) \geq s_j(\sigma) \\ s_k(\sigma) & \text{otherwise} \end{cases} \tag{4.22}$$

••

The following proposition establishes that if, under an optimal schedule, some job is scheduled later than another job with the same priority and a smaller processing time, then the positions of the two jobs can be interchanged without disturbing the optimality of the schedule. Note that, although this result seems obvious, it is not trivial since the effect of the interchange depends on the number of jobs in each priority class which follow the jobs which are to be interchanged.

Proposition 1: If σ is optimal and P(i)=P(j)=k' then

$$(s_i(\sigma) - s_j(\sigma))(t_i - t_j) < 0 \Rightarrow f_k(\sigma_{ij}) = f_k(\sigma)$$

Proof:

Let σ be optimal and P(i)=P(j)=k'.

(i) Clearly $f_k(\sigma_{ij}) = f_k(\sigma)$ if k<k' regardless of whether $(t_i - t_j) < 0$ or $(t_i - t_j) \geq 0$ since (by *Lemma 4*) all jobs with priority higher than k' are scheduled before both i and j and are thus unaffected by the interchange.

(ii) Assume, without loss of generality, that $s_i(\sigma) - s_j(\sigma) > 0$ and $t_i - t_j < 0$. Let n_i^k and n_j^k denote, respectively, the number of jobs (including i and j if k=k') with priority k scheduled after the start of i on $m_i(\sigma)$ and after the start of j on $m_j(\sigma)$.

Then

$$f_k(\sigma_{ij}) - f_k(\sigma) = (t_i - t_j)(n_j^k - n_i^k) \quad \text{for } k \geq k' \tag{4.23}$$

Suppose that

$$f_k(\sigma_{ij}) - f_k(\sigma) \neq 0$$

for some $k \geq k'$. Let k" be the highest priority for which this occurs. Then we must have:

$$f_{k''}(\sigma_{ij}) - f_{k''}(\sigma) > 0 \tag{4.24}$$

since otherwise the optimality of σ would be contradicted. Then

$$n_j^{k''} - n_i^{k''} < 0 \tag{4.25}$$

and

$$n_j^k = n_i^k \quad \text{for } k' \le k < k'' \tag{4.26}$$

Clearly (25) implies that i and j are scheduled on different machines since otherwise $s_i(\sigma) - s_j(\sigma) > 0$ would require that $n_j^{k''} - n_i^{k''} \ge 0$. Now consider σ'_{ij}. We have

$$f_k(\sigma'_{ij}) - f_k(\sigma) = (s_i(\sigma) - s_j(\sigma))(n_j^k - n_i^k) \quad \text{for } k > k'. \tag{4.27}$$

Now, using (25) and (26) we get:

$$f_k(\sigma'_{ij}) = f_k(\sigma) \quad \text{if } k' \le k < k'' \tag{4.28}$$

and

$$f_{k''}(\sigma'_{ij}) < f_{k''}(\sigma). \tag{4.29}$$

From an argument similar to that in (1), we can show that

$$f_k(\sigma'_{ij}) = f_k(\sigma) \quad \text{if } k < k'. \tag{4.30}$$

Thus, by *Lemma 1*, σ cannot be optimal. ••

We now use *Proposition 1* to define a constructive algorithm for solving the problem. First, we define the list index λ_i for each job i as follows:

$$\lambda_i = \left|\left\{ j \in J \middle| [P(j) < P(i)] \text{ or } [P(j) = P(i) \text{ and } t_j < t_i] \right\}\right|. \tag{4.31}$$

Thus if we list the jobs in non-decreasing order of their list indices, the priority classes will appear in order of priority, with highest priority first, and the jobs within each priority class will be listed in non-decreasing order of their processing times. The algorithm which solves the problem is as follows:

Algorithm AI

step 1: List all jobs in non-decreasing order of their list indices.

step 2: Remove the first unscheduled job from the list and schedule it on the first available machine.

step 3: If there are still jobs on the list go to step 2.
Otherwise stop.

We now prove that AI yields an optimal schedule. We can combine *Lemma 4* and *Proposition I* to get the following proposition:

Proposition 2: If σ is optimal and

(a) $s_i(\sigma) < s_j(\sigma)$

(b) $\lambda_i \geq \lambda_j$

then σ_{ij} is also optimal.

Proof:

(i) if $\lambda_i = \lambda_j$ then P(i)=P(j) and $t_i = t_j$ so that

$$f_k(\sigma_{ij}) = f_k(\sigma) \quad \forall k \in \{1,2,\ldots p\} \tag{4.32}$$

(ii) Suppose that $\lambda_i > \lambda_j$. Then P(i)>P(j), or P(i)=P(j) and $t_i < t_j$. But if P(i)>P(j) then $s_i(\sigma) \geq s_j(\sigma)$ by *Lemma 4* and (a) is contradicted. If P(i)=P(j) and $t_i < t_j$ then the result follows from *Proposition 1* ••

Lemma 5: If σ is optimal and

$$s_i(\sigma) = s_j(\sigma)$$

then σ'_{ij} is also optimal.

Proof:

If $s_i(\sigma) = s_j(\sigma)$ then $s_\ell(\sigma'_{ij}) = s_\ell(\sigma) \quad \forall \ell \in J$. Therefore

$$f_k(\sigma'_{ij}) = f_k(\sigma) \quad \forall k \in \{1,2,\ldots,p\}$$ ••

The algorithm A1 does not yield a unique solution for the following reasons:

(1) Step 1 can be accomplished in many different ways since if a group of jobs all have the same list index, then their order in the list is arbitrary.

(2) In step 2, if more than one machine becomes the first available at the same time then the next unscheduled job could be loaded onto any one of these machines.

We adopt the convention of always assigning the next unscheduled job to the first available machine with the smallest index. Since the assignment of indices to the machines is arbitrary, this still leaves the possibility for multiple solutions, but for a given job list (from step 1) and a given indexing of machines, the solution will be unique. We now define the algorithm A2 which begins with an optimal solution and manipulates it in such a way that the resulting schedule, by virtue of *Proposition 2 and Lemma 5,* is also optimal. We then show that the resulting schedule is exactly the same as that which results from applying the algorithm A1 so that that schedule must also be optimal.

Algorithm A2

step 1: Begin with any optimal sequence σ.

step 2: List all jobs in non-decreasing order of their list indices.

step 3: Let σ^c denote the current schedule.

If there exist,under σ^c,a pair of jobs, i and j, such that j occurs before i on the list and $s_i(\sigma^c) < s_j(\sigma^c)$ then replace σ^c with σ^c_{ij} and repeat step 3.

Otherwise continue on to step 4

step 4: If there exist, under the current schedule σ^c, two jobs, i and j, such that the following three conditions are met:

(a) j occurs before i in the list.

(b) $s_i(\sigma^c) = s_j(\sigma^c)$

(c) $m_i(\sigma^c) < m_j(\sigma^c)$

then replace σ^c with $(\sigma^c)'_{ij}$ where i and j are the earliest occurrence of such a pair, and repeat step 4.

Otherwise go to step 5

<u>step 5</u>: If any job i is scheduled last on it's machine and another machine with a lower index is available at time $s_i(\sigma^c)$, then reschedule i as the last job on that machine and repeat step 5..

Otherwise stop.

We now prove that A1 and A2 yield the same solution.

Proposition 3: The algorithms AI and A2 yield the same result if the indexing of the machines and the listing of the jobs (in step 1 of A1 and step 2 of A2) are the same.

Proof:

Let σ_1^k and σ_2^k represent the schedules obtained by applying algorithms A1 and A2 (respectively) to the first k jobs in the list.

(i) Clearly $\sigma_1^1 = \sigma_2^1$ since in both cases the first job on the list must be scheduled first on machine 1.

(ii) Assume $\sigma_1^k = \sigma_2^k$ for some k<n. Under σ_2^{k+1} the k+ 1'st job in the list must be scheduled on the lowest indexed first available machine (under $\sigma_1^k = \sigma_2^k$). This is because if the job were to be scheduled at a later time then σ_2^{k+1} would not be optimal whereas A2 always yields an optimal sequence. If the job was scheduled to begin at the same time on a machine with a higher index, then A2 would not have stopped at σ_2^{k+1}. This, however, is exactly the rule for scheduling the k+l'st job according to Al. Thus

$$\sigma_1^k = \sigma_2^k \Rightarrow \sigma_1^{k+1} = \sigma_2^{k+1} \quad \forall k < n.$$

(iii) Therefore, by induction,

$$\sigma_1^n = \sigma_2^n \tag{4.33}$$

and so AI and A2 yield the same result. ••

Proposition 3 and the optimality of σ_2^n immediately yield the result:

Proposition 4: The algorithm AI yields an optimal solution . ••

4.4 Complexity

A1 belongs to a class of algorithms known as "list scheduling" algorithms. In these

algorithms an ordered list of the jobs is created and at each stage the first unscheduled job in the list is loaded onto the first available machine. The order of the jobs in the list determines the schedule. Various list orders (eg. random, LPT, SPT) have been considered for a variety of different problems. These type of algorithms are generally polynomially bounded.

In A1, arranging the jobs in order of their list indices can be done in O(nlogn) time. The construction of the schedule, once the list is arranged, can be done in O(nm) time. Thus the complexity of A1 is O(mnlogn).

4.5 Example

We consider an example with two machines and six jobs. The priorities for the jobs are given by

$$\mathbf{P} = (P(1),P(2),\ldots P(6)) = (1,1,2,2,3,3)$$

and the processing times are given by

$$\mathbf{t} = (t_1,t_2,\ldots t_6)=(4,6,3,5,3,3).$$

The jobs are already in non-decreasing order of their list indices

$(\lambda_1,\lambda_2,\ldots\lambda_6)=(0,1,2,3,4,4)$. Figure 4.1 is a Gantt diagram of the optimal schedule obtained by applying A2. Figure 4.2 shows three Gantt diagrams which correspond to

various stages of A2. The flowtime for all schedules depicted are given by

$$\mathbf{f} = (f_1, f_2, f_3) = (10,18,24).$$

Note that , as expected, A2 yields the same result as A1. It is clear that job 1 and job 2 must be scheduled on different machines since otherwise f_1 would not be optimal. If jobs 2 and 3 are scheduled on different machines then it is clear from inspection of the Gantt diagrams in Figure 4.2 that jobs 5 and 6 must also be scheduled on different machines (as depicted in the figure) and so Figure 4.2 covers all possibilities (in that case) which may be optimal. If jobs 3 and 4 are scheduled on the same machine then we get the schedules depicted in Figure 4.3. It is clear from inspection of these diagrams that jobs 5 and 6 must be scheduled on the same machine in this case if we are to have an optimal solution. The flowtimes for these two schedules are given by

$$\mathbf{f} = (f_1, f_2, f_3) = (10,19,21)$$

and

$$\mathbf{f} = (f_1, f_2, f_3) = (10,23,17).$$

Clearly these are both sub-optimal schedules.

4.6 Conclusion

We have considered the problem of hierarchical minimization of total flowtimes of

priority classes of jobs scheduled on parallel machines. We have provided a list scheduling algorithm of complexity O(mnlogn) to solve the problem. The algorithm is a generalized SPT rule.

Chapter 5

Minimum Flowtime
With Priority Constraint

5.1 Introduction

In the previous chapter we considered a problem involving the scheduling of jobs with different priorities. There, our interperetation of priority was quite stringent, in that we could not make any improvement in the scheduling of lower priority jobs if it compromised (even minutely) the quality of the scheduling of the higher priority jobs. In this chapter, we present a different, less stringent view of the job priorities. We consider the objective of minimizing the total flowtime subject to the constraint that a lower priority job can never precede a higher priority job on the same machine.

This type of problem is likely to occur where there are technological constraints or set-up costs involved. The priority classes may actually divide the jobs into groups which have different processing requirements. There may be a set-up cost involved in adapting a machine from a state where it is ready to process jobs from one group to a state where it is ready to process jobs from a second group. If there are two groups (we consider only the problem with two priority classes) then the constraint we impose guarantees that the set-up cost is incurred only once for each machine.

Similarly, technological requirements may prohibit one type of job from being processed after another type of job on the same machine. For example, in paint production, light color paints must precede darker color paints.

A great deal of research (see the reviews: Coffman, Cheng(1990) and Lawler(1989)) dealing with precedence constraint has been reported so far. However, all of this deals with precedence relations which can be represented by graphs which are trees. The precedence graph for the problem we consider here is a complete directed bipartite graph and is therefore not a tree. This analysis can therefore not be applied here.

Gupta(1984) and Potts (1991) consider the problem of scheduling jobs which fall into two different classes on a single machine. Setup time is necessary between jobs of a different class and the objective is to minimize the sum of the completion times. This problem is similar to ours in that, if the set up time between jobs of different classes is large enough, then any optimal schedule must involve at most one such set-up on each machine. Our objective is to minimize the sum of completion times subject to the constraint that only one setup is allowed on each machine. Thus our objective could well be applied to the cases which Potts (1991) has cited if multiple processors are available (Our problem is trivial in the single machine case.) and one of the two set up costs is very large. These applications include steel pipe manifacturing (Ahn and Hyun (1990), computer scheduling (Monma and Potts (1989)) and operation of multiple processors by a single operator (Sahney (1972)).

We formulate our problem as follows.

5.2 Problem Formulation

Consider a set J={ 1,2,...,n} of n jobs to be scheduled on a set M={1,2,...,m} of m parallel machines. Each job $j \in J$ is assigned a priority $P(j) \in (1,2)$ and a processing time $t_j \geq 0$. We will refer to jobs j with priority P(j)=1 as "p-jobs" and to jobs with P(j)=2 as "q-jobs". Let n_p and n_q denote the number (respectively) of p-jobs and q-jobs. We assume that the jobs are indexed such that:

(a) $P(1) = P(2) = \ldots = P(n_p) = 1$

(b) $P(n_p + 1) = P(n_p + 2) = \ldots = P(n) = 2$

(c) $t_1 \geq t_2 \geq \ldots \geq t_{n_p}$

(d) $t_{n_p+1} \geq t_{n_p+2} \geq \ldots \geq t_n$ (5.1)

We define a schedule of the jobs in J to be any function of the form $\sigma: J \rightarrow M \times \Re$ where

$$\sigma(j) = (m_j, s_j) \tag{5.2}$$

and

(a) $s_j \geq 0$

(b) $s_{j'} + t_{j'} \leq s_j$ or $s_{j'} \geq s_j + t_j$ if $m_j = m_{j'}$ (5.3)

m_j represents the machine on which the job j is to be processed and s_j represents the start time of that job. Condition (a) requires that all jobs begin processing after time zero and (b) ensures that no two jobs are processed simultaneously on a single machine. Let $c_j = s_j + t_j$ denote the completion time of the job j . We will require the following definition:

Definition 1: A schedule σ is said to be feasible if no p-job is scheduled after a q-job on the same machine .i.e. if

$$m_i(\sigma) = m_j(\sigma) \Rightarrow (P(i) - P(j))(s_i(\sigma) - s_j(\sigma)) \geq 0 \qquad \bullet\bullet$$

We denote the sum of completion times (under σ) of the jobs in J by

$$f(\sigma) = \sum_{j \in J} c_j(\sigma) \tag{5.4}$$

Then the problem we consider is:

P4: Find a schedule σ such that :

$$f(\sigma) \leq f(\sigma') \text{ for all feasible schedules } \sigma' \qquad \bullet\bullet$$

5.3 Results

Any schedule which satisfies the above condition will be called "optimal". It is easy to see that it is not beneficial to insert idle time between jobs. Then, if we restrict our attention to schedules without inter-job idle time, we can describe a schedule by specifying the sequence of jobs on each machine. Let (i,k) denote the position of the job scheduled kth last on machine i and let [i,k] denote the job scheduled in position (i,k). Also let $t_{[i,k]}$ and $p_{[i,k]}$ denote the processing time and priority of the job in position (i,k). If no job is assigned to (i,k) then set $t_{[i,k]} = p_{[i,k]} = 0$. Then the objective function can be expressed as

$$f(\sigma) = \sum_{i=1}^{m}\sum_{k=1}^{n} k t_{[i,k]}\Big|_{\sigma} = \sum_{k=1}^{n} k \sum_{i=1}^{m} t_{[i,k]}\Big|_{\sigma} \tag{5.5}$$

and the schedule is feasible if

$$p_{[i,1]} \geq p_{[i,2]} \geq \ldots \geq p_{[i,n]} \quad \forall i \in M. \tag{5.6}$$

We will refer to the set $\{(i,k)|k' = k\}$ as "column k". Since, for any given σ, the contribution to $f(\sigma)$ from column k depends only upon the sum of processing times of fobs assigned to the column, we can permute the jobs in column k amongst the positions in the

column without altering the value of $f(\sigma)$. This motivates the following definition.

Definition 1: A schedule σ is said to be "separated" if , under σ:

$$p_{[1,k]} \geq p_{[2,k]} \geq \ldots \geq p_{[m,k]} \quad \forall k \leq n. \tag{5.7}$$

In a separated schedule the q-jobs are assigned to the lowest indexed machines in each column, the p-jobs are assigned to higher indexed machines and only positions on the highest indexed machines are empty. Figure 5.1 shows a diagram of a separated schedule. Let n_k^q denote the number of q-jobs scheduled in column k. We will call any sequence of the form $\{n_k^q\}$, where each element n_k^q (k=1,2,...n) is defined as above, a "q-sequence". Then we have the following:

Proposition 1: A separated schedule is feasible if and only if $\{n_k^q\}$ is a nonincreasing sequence.

Proof:

(i) Let σ be feasible. Then each q job in column k+l must be followed by another q job in column k (for k=1,2 .... n- 1). Therefore

$$n_{k+1}^q \leq n_k^q \quad \forall k \leq n-1. \tag{5.8}$$

(ii) Suppose that σ is not feasible and let (i,k) be the position of a p job which directly follows a q job. Then there is a q job in (i,k+ 1) so that, since σ is separated we must have

$$n^q_{k+1} \geq i \ . \tag{5.9}$$

Also, since a p job is scheduled in (i,k), and σ is separated we must have

$$n^q_k < i. \tag{5.10}$$

Therefore $n^q_{k+1} > n^q_k$ and so $\{ n^q_k \}$ is not a non-increasing sequence. ••

Note that part (i) of the proof above did not require that the schedule σ be separated. Thus we have the following corollary.

Corollary 1: If σ is feasible then $\{ n^q_k \}$ is a non-increasing sequence. ••

Thus, given any feasible schedule, if we permute the jobs in each column until the schedule becomes separated, the resulting schedule will be feasible, since the sequence $\{ n^q_k \}$ is unchanged. Also, since "separating" a schedule in this manner does not alter the objective function value, the separated version of any optimal schedule must also be optimal. Thus we can restrict our search for an optimal schedule to the set of all separated

schedules. We define a "feasible q-sequence" as follows.

Definition 3: A feasible q-sequence is any sequence $\{ n_k^q \}$ of n integers which satisfies:

(a) $\quad 0 \le n_k^q \le m \quad \forall k \le n$

and

(b) $\quad n_{k+1}^q \le n_k^q \quad \forall k \le n-1$ ••

Let $S(\{n_k^q\})$ denote the set of all separated schedules whose "q-sequences" are given by $\{ n_k^q \}$ and let Q^f denote the set of all feasible q-sequences. Also let $S(Q^f)$ denote the set of all separated schedules which have feasible q-sequences. i.e.

$$S(Q^f) = \bigcup_{\{n_k^q\} \in Q^f} S(\{n_k^q\}). \tag{5.11}$$

Then our restricted problem , which shares a solution with P4 is :

P5: Find a schedule $\sigma \in S(Q^f)$ such that

$$f(\sigma) = \min_{\sigma' \in S(Q^f)} \{f(\sigma')\} = \min_{\{n_k^q\} \in Q^f} \left\{ \min_{\sigma' \in S(\{n_k^q\})} \{f(\sigma')\} \right\} \tag{5.12}$$

••

We are also interested in the following subproblem:

P6: Given a q-sequence $\{n_k^q\} \in Q^f$, find a schedule $\sigma \in S(\{n_k^q\})$ such that

$$\sigma = \min_{\sigma' \in S(\{n_k^q\})} \{f(\sigma')\} \tag{5.13}$$

••

If $\sigma \in S(\{n_k^q\})$ then we can rewrite $f(\sigma)$ as

$$f(\sigma) = f_1(\sigma) + f_2(\sigma) \tag{5.14}$$

where

$$f_1(\sigma) = \sum_{k=1}^{n} \sum_{i=1}^{n_k^q} k t_{[i,k]} \tag{5.15}$$

and

$$f_2(\sigma) = \sum_{k=1}^{n} \sum_{i=n_k^q+1}^{\min\{m,n_p\}} k t_{[i,k]}. \tag{5.16}$$

Since both $f_1(\sigma)$ and $f_2(\sigma)$ are sums of pairwise products of two sets of real numbers, we can minimize either by arranging one set (the k's) in non-decreasing order and the other set ($t_{[i,k]}$'s) in non-increasing order, and matching corresponding pairs. This will yield the schedule described by:

$$[i,k] = \begin{cases} n_p + \sum_{j=1}^{k-1} n_j^q + i & \text{if } i \le n_k^q \\ m(k-1) - \sum_{j=1}^{k} n_j^q + i & \text{if } i > n_k^q \text{ and } m(k-1) - \sum_{j=1}^{k} n_j^q + i \le n_p \\ \phi & \text{otherwise} \end{cases} \quad (5.17)$$

We will denote the schedule described above by $\sigma^*\{n_k^q\}$. Thus $\sigma^*\{n_k^q\}$ is always a solution to P6 for any given $\{n_k^q\} \in Q^f$ and so the following problem P7 shares a solution with P5 , and thus P4.

P7: Find a sequence $\{n_k^q\} \in Q^f$ and the corresponding schedule $\sigma^*\{n_k^q\}$ such that :

$$f(\sigma^*\{n_k^q\}) = \min_{\{n'^q_k\} \in Q^f} \{f(\sigma^*\{n'^q_k\})\}. \qquad \bullet\bullet$$

Thus it remains only to find the optimal q-sequence, We do this via a polynomial dynamic programming algorithm.

5.4 Dynamic Programming Algorithm

(i) Let g(k,q,x) denote the contribution to $f(\sigma^*\{n_k^q\})$ from the q-jobs scheduled in column k if q q-jobs in total are scheduled in columns 1 through k. i.e.

$$q = \sum_{j=1}^{k} n_k^q \qquad (5.18)$$

Then

$$g(k,q,x) = k\left(\sum_{j=n_p+q-n_k^q+1}^{n_p+q} t_j\right) \tag{5.19}$$

(ii) Let h(k,q,x) denote the contribution to $f(\sigma^*\{n_k^q\})$ of the p-jobs scheduled in column k if x q-jobs are scheduled in column k and q q-jobs in total are scheduled in columns 1 through k. Then:

$$h(k,q,x) = k\left(\sum_{m(k-1)-q+x+1}^{\min\{n_p,mk-q\}} t_j\right). \tag{5.20}$$

Let $\ell_k = n_{k+1}^q$ and let $u_k(q) = \min\left\{m, \left\lfloor \frac{q}{k} \right\rfloor\right\}$. Then in a feasible schedule with q q-jobs in total assigned to columns 1 through k, we must have

$$\ell_k \le n_k^q \le u_k(q) \tag{5.21}$$

The upper bound $\left\lfloor \frac{q}{k} \right\rfloor$ arises because if more than $\left\lfloor \frac{q}{k} \right\rfloor$ q-jobs are assigned to column k then some column k'<k must have less q-jobs than column k since there are not enough q-jobs to assign $\left\lfloor \frac{q}{k} \right\rfloor$+1 of them to each column in $\{1,2,\ldots,k\}$.

Let f(k,q,l) denote the minimum value of the objective function for schedules of the form $\sigma^*\{n_k^q\}$ (where $\{n_k^q\} \in Q^f$) which have q q-jobs in total scheduled in columns 1 through k and at least l q-jobs in column k. Then we have the recurrence relation:

$$f(k,q,\ell) = \min_{\ell \le x \le u_k(q)} \{g(k,q,x) + h(k,q,x) + f(k-1,q-x,x)\} \tag{5.22}$$

for $1 \le k \le n$, $0 \le q \le \min\{n_q, mk\}$ and $0 \le \ell \le u_k(q)$

.

The boundary condition is given by:

$$f(0,q,l) = \begin{cases} 0 & \text{if } q = 0 \text{ and } 1 \le \ell \le m \\ \infty & \text{otherwise} \end{cases}.$$

The optimal objective function value is given by

$$f_{min} = f(n, n_q, 0) \tag{5.23}$$

Since ℓ represents only the lower bound for values over which a minimum is taken we can separate (5.22) into two simpler recurrence relations:

(a) $$f(k,q,u_k(q)) = g(k,q,u_k(q)) + h(k,q,u_k(q)) + f(k-1, q-u_k(q), u_k(q)) \tag{5.24}$$

for $1 \le k \le n$ and $0 \le q \le \min\{n_q, mk\}$

and

(b) $$f(k,q,\ell) = \min\{f(k,q,\ell+1), g(k,q,\ell) + h(k,q,\ell) + f(k-1, q-\ell, \ell)\} \tag{5.25}$$

for $0 \le \ell \le u_k(q)$, $1 \le k \le n$ and $0 \le q \le \min\{n_q, mk\}$.

Thus we have a dynamic programming algorithm where the n stages correspond to the columns k (first argument of f) and the state at each stage is given by the last two arguments (q,ℓ) of f. We begin at stage 0, by evaluating $f(0,q,\ell)$ for each possible state $\{(q,\ell) | 0 \le q \le n_q \text{ and } 0 \le \ell \le m\}$. At each successive stage we start with the state $(q,\ell)=(0,0)$. Each time we increase q to q+1 , we use (b) to evaluate $f(k,q+1,\ell)$ for each value of ℓ, beginning with $u_k(q)$, and decreasing 1 unit at a time until we reach the state (q+1,0).

In backtracking to find the solution we start with the combination $(k,q)=(n,n_q)$. At each stage k, if we are currently at the combination (k,q) , we find the smallest value of ℓ, say ℓ', for which the second argument was chosen as minimum in applying recurrence relation (b). If the second argument is never chosen

for some combination (k,q) then we set $\ell' = u_k(q)$. Then we set $n_k^q = \ell'$ and move to the combination (k-1,q-ℓ'). When we are finished, we will have the optimal q-sequence $\{n_k^q\}$ and we can use (17) to find the optimal schedule $\sigma^*\{n_k^q\}$.

5.5 Complexity

Applying the recursion relation (26) always involves taking a minimum of only two values. Thus the complexity of the dynamic programing algorithm is, if all the values of g(k,q,x) and h(k,q,x) are already computed, proportional to the number of cells in the array represented by $f(k,q,\ell)$. The maximum number of cells in this array is mn^2 since $k \le n$, $q \le n_q \le n$ and $\ell \le m$. The calculation of each value of g(k,q,x) or h(k,q,x) requires summing at most m terms. Each of the corresponding arrays has at most mn^2 cells (since $k \le n$, $q \le n_q \le n$ and $x \le m$). Thus the number of steps required to calculate these two arrays is at worst $O(m^2n^2)$. However, if g(k,q,x) and h(k,q,x) are calculated recursively, we can calculate all of the values for these arrays in $O(mn^2)$ time. Thus the complexity of the entire algorithm is $O(mn^2)$. Since we can assume m<n (otherwise the problem is trivial) this is clearly a polynomial algorithm.

5.6 Example

We now illustrate the above algorithm via a simple example. Consider an instance of the problem with two p-jobs and two q-jobs. The p-jobs have processing times 1 and 3 and the q-jobs have processing times 2 and 4. Thus:

$$(P(1),P(2),P(3),P(4))=(1,1,2,2)$$

and

$$(t_1,t_2,t_3,t_4) = (3,1,4,2).$$

The only feasible q-sequences are

$(n_1^q, n_2^q, n_3^q, n_4^q) = (2,0,0,0)$

and

$(n_1^q, n_2^q, n_3^q, n_4^q) = (1,1,0,0)$.

The corresponding schedules are illustrated in figure 5.1. The sequences (1,1,0,0) and (2,0,0,0) correspond to objective function values of 13 and 14 respectively. Thus (1,1,0,0) is the optimal q-sequence. Table 5.1 contains the values of g(k,q,x) and h(k,q,x) for relevant values of k,q and x. Table 5.2 contains the values of $u_k(q)$ for relevant values of k and q. The boundary condition is applied to yield the following table for $f(0,q,\ell)$:

$f(0,q,\ell)$

		ℓ		
		0	1	2
	0:	∞	0	0
q	1:	∞	∞	∞
	2:	∞	∞	∞

The following calculations are performed in the order in which they occur here. The value of each expression is written above the expression and a "*" precedes the expressions which correspond to the smallest value of ℓ for which the second argument is chosen as minimum.

k=1

<u>q=0</u>

$$f(1,0,0)=\overset{0}{g(1,0,0)}+\overset{4}{h(1,0,0)}+\overset{\infty}{f(0,1,0)}=\infty$$

<u>q=1</u>

$$*f(1,1,1)=\overset{4}{g(1,1,1)}+\overset{3}{h(1,1,1,)}+\overset{0}{f(0,0,1)}=7$$

$$f(1,1,1)=\min\{\overset{7}{f(1,1,1)},\overset{0}{g(1,1,0)}+\overset{3}{h(1,1,0,)}+\overset{\infty}{f(0,1,0)}\}=7$$

<u>q=2</u>

$$f(1,2,2)=\overset{6}{g(1,2,2)}+\overset{0}{h(1,2,2)}+\overset{0}{f(0,0,2)}=6$$

$$f(1,2,1)=\min\{\overset{6}{f\{1,2,2)},\overset{2}{g(1,2,1)}+\overset{0}{h(1,2,1)}+\overset{\infty}{f(0,1,1)}\}=6$$

$$f(1,2,0)=\min\{\overset{6}{f\{1,2,1)},\overset{0}{g(1,2,0)}+\overset{0}{h(1,2,0)}+\overset{\infty}{f(0,2,0)}\}=6$$

k=2

<u>q=0</u>

$$f(2,0,0)=\overset{0}{g(2,0,0)}+\overset{0}{h(2,0,0)}+\overset{\infty}{f(1,0,0)}=\infty$$

<u>q=1</u>

$$f(2,1,0)=\overset{0}{g(2,1,0)}+\overset{2}{h(2,1,0)}+\overset{7}{f(1,1,0)}=7$$

<u>q=2</u>

$$*f(2,2,1)=\overset{4}{g(2,2,1)}+\overset{2}{h(2,2,1)}+\overset{7}{f(1,1,1)}=13$$

$$f(2,2,0)=\min\{\overset{13}{f\{2,2,1)},\overset{0}{g(2,2,0)}+\overset{8}{h(2,2,0)}+\overset{6}{f(1,2,0)}\}=13$$

k=3

$$f(3,0,0)=\overset{0}{g(3,0,0)}+\overset{0}{h(3,0,0)}+\overset{\infty}{f(2,0,0)}=\infty$$

$$f(3,1,0)=\overset{0}{g(3,1,0)}+\overset{0}{h(3,1,0)}+\overset{7}{f(2,1,0)}=7$$

$$*f(3,2,0)=\overset{0}{g(3,2,0)}+\overset{0}{h(3,2,0)}+\overset{13}{f(2,2,0)}=13$$

k=4

$$f(4,0,0)=\overset{0}{g(4,0,0)}+\overset{0}{h(4,0,0)}+\overset{\infty}{f(3,0,0)}=\infty$$

$$f(4,1,0)=\overset{0}{g(4,1,0)}+\overset{0}{h(4,1,0)}+\overset{7}{f(3,1,0)}=7$$

$$*f(4,2,0)=\overset{0}{g(4,2,0)}+\overset{0}{h(4,2,0)}+\overset{13}{f(3,2,0)}=13$$

The following diagram illustrates the backtracking procedure.

$$(n,n_q)=(4,2)\xrightarrow{\text{set } n_4=0}(3,2)\xrightarrow{\text{set } n_3=0}(2,2)\xrightarrow{\text{set } n_2=1}(1,1)\xrightarrow{\text{set } n_1=1}(0,0)$$

Thus the optimal q-sequence is (1,1,0,0).

5.7 Conclusion

We have provided a poynomial dynamic programming algorithm for minimizing total flowtime on parallel machines subject to a 2-class job priority precedence constraint. The next step, for further research, is to generalize this result to an arbitrary number of priority classes.

References

E.H.L. Aarts and J. Korst (1981), *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*, Chichester,New York

M. U. Ahmed and P. S. Sundararaghavan (1989) Minimizing the sum of absolute deviation of job completion times from their due-dates in a single machine scheduling problem. *TIMS S.E. chapter meeting*,38-40

B.H. Ahn and J.H. Hyun (1990) Single facility multi-class job scheduling. *Computers Ops. Res.*, **17**, 265-272

U. Bagchi, R.S, Sullivan and Y.L. Chang (1986) Minimizing mean absolute deviation of completion times about a common due-date. *Naval Research Logistics Quarterly*, **33**, 227-240.

K.R. Baker and G.D. Scudder (1990) Sequencing with earliness and tardiness penalties: a review. *Operations Research*, **38**, 22-36.

C. R. Bector ,Y. P. Gupta and M.C. Gupta (1988) Determination of an optimal common due date and optimal sequence in a single machine job shop. *Int. J. Prod Res.,**26***,613-628

C. R. Bector ,Y. P. Gupta and M.C. Gupta (1988) V-shape property of optimal sequence of jobs about a Common due date on a singe machine. *Comp. Op. Res.,**16***,583-588

T. C. E. Cheng (1985) A duality approach to due date determination *Engng. Optim.*, **9**, 127-130

T.C.E. Cheng (1987) An algorithm for the CON due-date determination and sequencing problem. *Computers and Operations Research*, **14**, 537-542.

T.C.E. Cheng (1989) An algorithmic approach to a common due date scheduling problem in single machine systems, Working Paper, Dept. of Act. and Mgt. Sci.,University of Manitoba.

T.C.E. Cheng (1989) A Heuristic for Common Due-Date Assignment and Job Scheduling on Parallel Machines. *Journal of the Operational Research Society*, **40**, 1129-1135

T.C.E. Cheng and M.C. Gupta (1989) Survey of scheduling research involving due-date determination decisions. *European Journal of Operational Research*, **38**, 156-166.

T.C.E.Cheng (1990) A note on a partial search algorithm for the single-machine optimal common due-date assignment and sequencing problem, Computers and Operations Research, **17**, 321-324

T.C.E. Cheng and C.C.S.Sin (1990) A state-of-the-art review of parallel-machine scheduling research. *E.J.O.R.*,**47**, 271-292

T.C.E. Cheng and H.G. Kahlbacher (1990) The parallel-machine common due-date assignment and scheduling problem is NP-hard. In submission.

T.C.E. Cheng (1991) Optimal assignment of total-work-content due-dates and sequencing in a single-machine shop. *Journal of the Operational Research Society*, **42**, 177-181

E.G. Coffman, Jr. *Computer and Job-Shop Scheduling Theory*. Wiley & Sons, New York

R.W. Conway, W.L. Maxwell, and L.W. Miller (1967) *Theory of Scheduling*. Addison-Wesley, Reading, Mass.

P. De, J.B. Ghosh and C.F. Wells (1990) Con due date determination and sequencing. *Comp. Opns. Res.*, **17**, 333-342

P. De, J.B. Ghosh and C.F. Wells (1991) On the multiple machine extension to a common due-date assignment and scheduling problem. *J. Opl Res. Soc*, **42**, 419-422

P. Dileepan and T. Sen (1988) Bicriterion Static Scheduling research for a single machine.

*Omega, **16,*** 53-59.

H. Emmons (1987) Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics Quarterly*, **34**, 803-810.

M.R. Garey and D.S. Johnson (1979) *Computers and Intractability: A Guide to the T heory of NP-Completeness*, W.H. Freeman and Co., New York.

M.R. Garey, R.T. Tarjan and G.T. Wilfong (1988) One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, **13**, 330-348.

J.N.D. Gupta (1984) Optimal schedules for single facility with two job classes.*Computers Ops. Res.*, **11**, 409-413

N.G. Hall (1986) Single- and multiple-processor models for minimizing completion variances. *Naval Research Logistics Quarterly*, **33**, 49-54.

N.G. Hall (1989) and M.E. Posner Weighted deviation of completion times about a common due date, Working paper No. WPS 89-15, College of Business, Ohio State University, Columbus, Ohio.

J.J. Kanet (1981) Minimizing the average deviation of job completion times about a common due-date. *Naval Research Logistics Quarterly*, **28**, 643-651.

E.L. Lawler (1989) Sequencing and scheduling algorithms and complexity. *Repot BS-R8909 June*, Dept. of Operations Research,Statistics and System theory,Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

S. D. Liman and C. Y. Lee (in submission) Error bound of a heuristic for the common due date scheduling problem.

C.L. Monma and C.N. Potts (1989) On the complexity of scheduling with batch set-up times. *Ops. Res.*, **37**, 798-804.

S.S. Panwalkar, M.L. Smith and A. Seidmann (1982) Common due-date assignment to

minimize total penalty for the one machine scheduling problem. *Operations R esearch*, **30**, 391-399.

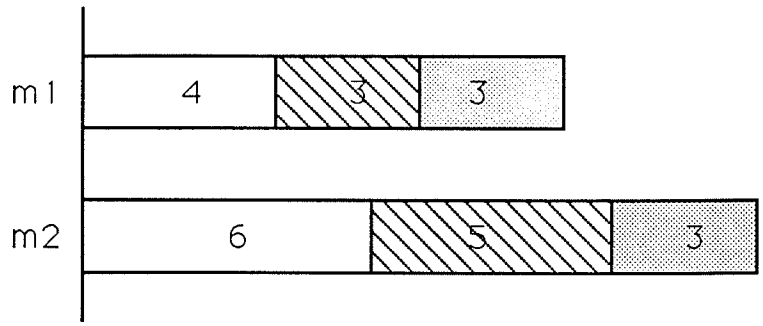
C.N. Potts (1991) Scheduling two job classes on a single machine. *Computers Ops. Res.*, **18**, 411-415.

V.K. Sahney (1972) Single server, two machine sequencing with switching times. *Ops. Res.*, **20**, 26-36

P.S. Sundararaghavan and M.U. Ahmed (1984) Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. *Naval Research Logistics Quarterly*, **31**, 325-333.

W. Szwarc (1989) Single-machine scheduling to minimize absolute deviation of completion times from a common due-date. *Naval Research Logistics Quarterly*, **36**, 663-673.

Figure 4.1
Schedule resulting from A1



Flowtimes

f1=10
f2=18
f3=24

Legend:

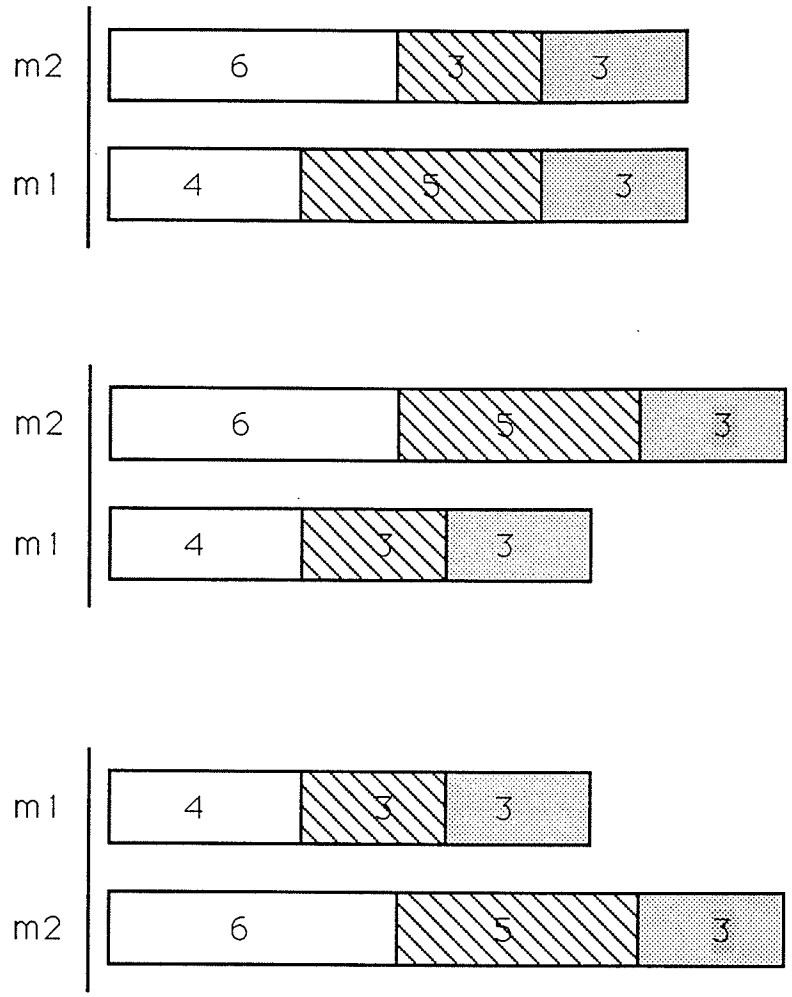
Priority=1

Priority=2

Priority=3

Figure 4.2

Sequence of schedules from A2



Flowtimes

<u>All schedules:</u>

f1=10
f2=18
f3=24

<u>Legend</u>:

Priority=1

Priority=2

Priority=3

Figure 4.3
Alternate schedules

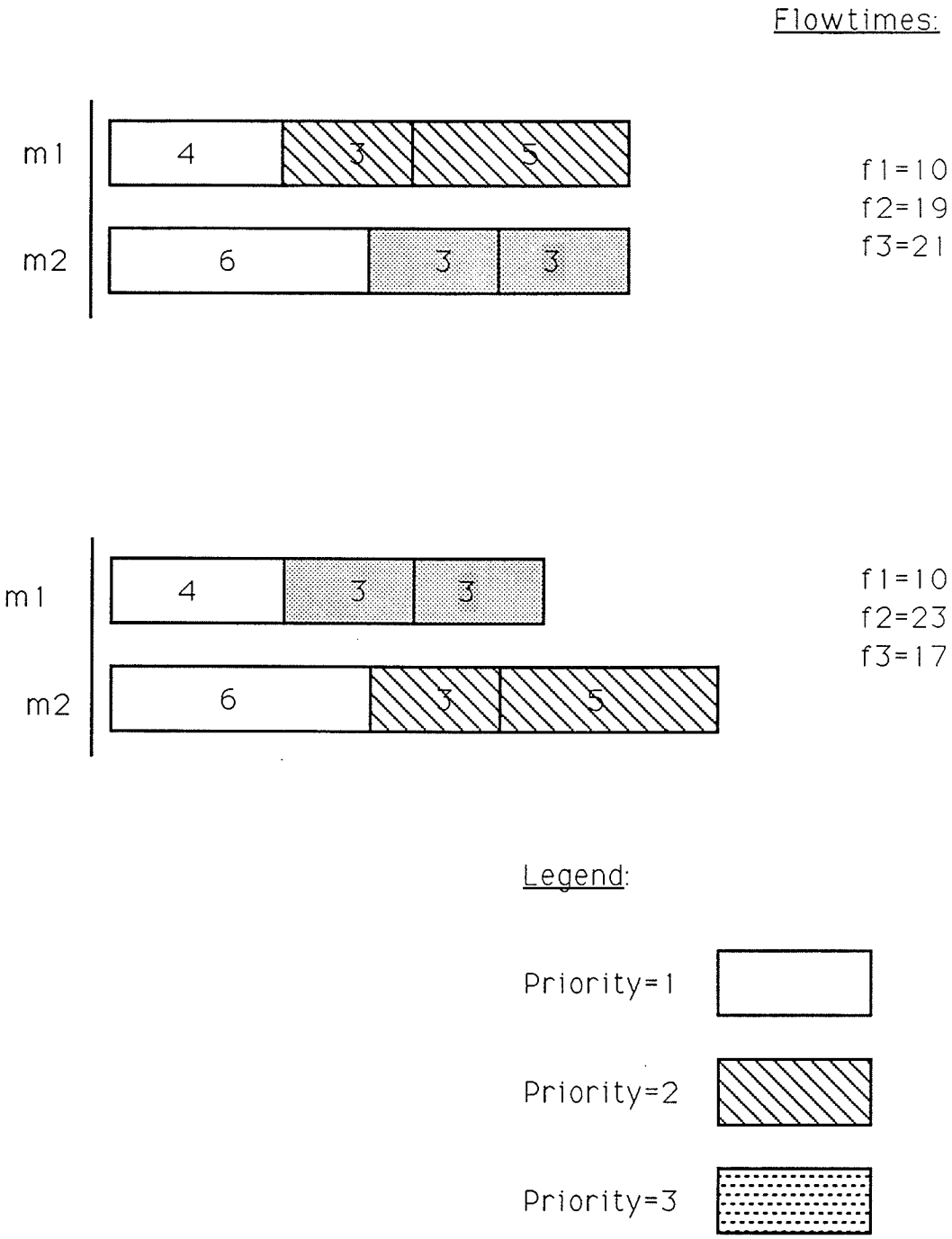


Figure 5.1:Separated Schedule

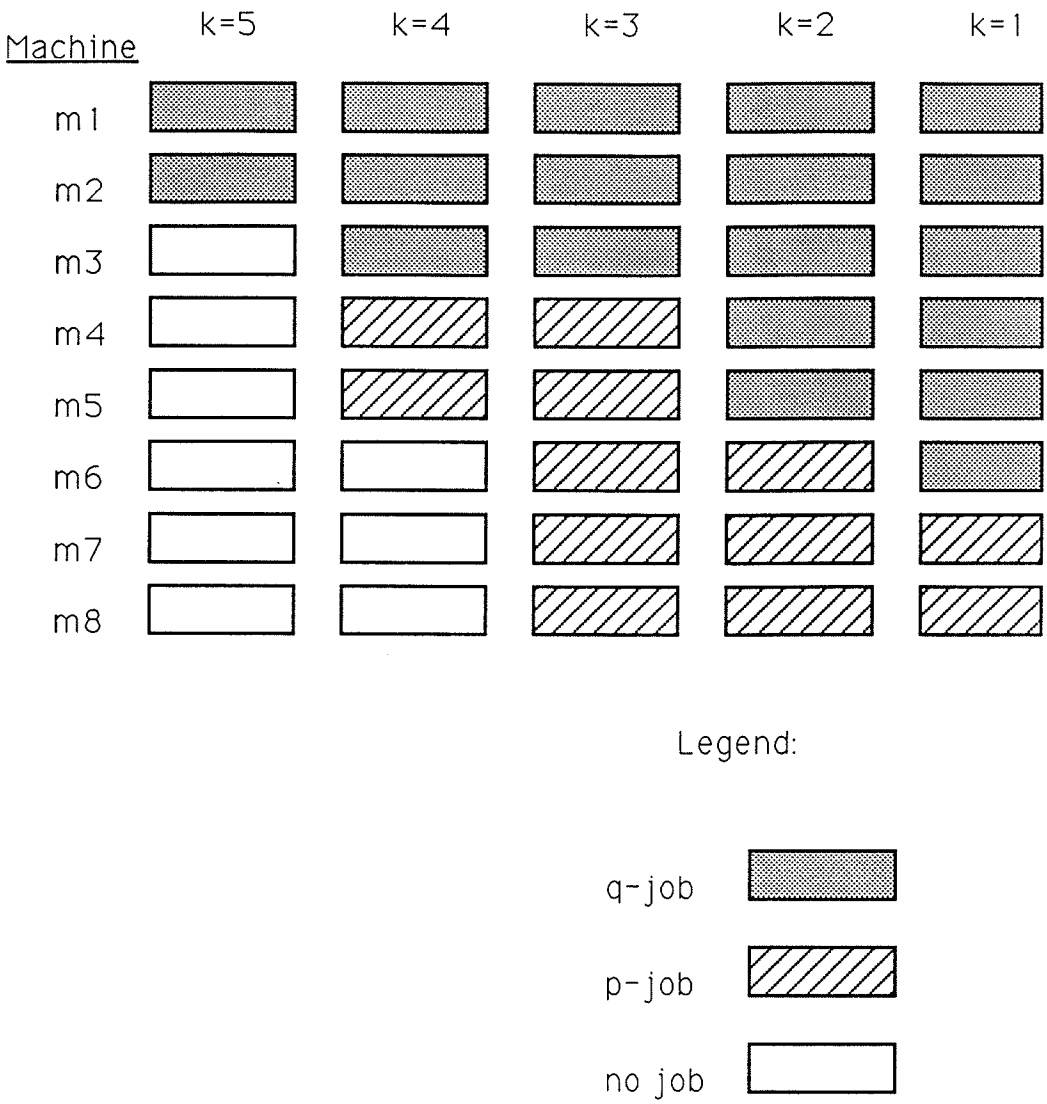


Table 2.1: Error bounds, computation time and memory

			local search	SA	DP:Q=.5t	DP:Q=.2t	DP:Q=.06t	DP:Q=.01t
n=20	rel. error bound	min	0.04	0.001	0.017	0.008		0.006
		max	0.144	0.024	0.044	0.026		0.009
		avg.	0.071	0.016	0.03	0.014		0.007
		avg. time	0.05	0.23	0.08	0.13		1.62
		memory			800	2000		40000
n=50	rel. error bound	min	0.029	0.009	0.018	0.008		0.007
		max	0.06	0.025	0.035	0.013		0.009
		avg.	0.036	0.017	0.029	0.011		0.0083
		avg. time	0.29	0.61	0.47	0.74		9.46
		memory			5000	12500		250000
n=100	rel. error bound	min	0.015	0.049	0.064	0.044	0.044	
		max	0.111	0.061	0.081	0.056	0.055	
		avg.	0.06	0.055	0.071	0.51	0.049	
		avg. time	1.14	2.18	1.82	2.9	6.86	
		memory			20000	50000	186667	1000000

a priori error bound
posterior error bound

Table 3.1: Relative Error bounds for the case Pd=0.5 and without zero idle time constraint

n/m

m	2	5	10	15	20	50	100
2	4	1.12	0.48	0.302	0.22	0.083	0.041
	0.355	0.062	0.014	0.014	0.01	0.005	0.002
5	6.39	1.79	0.768	0.484	0.352	0.133	0.066
	0.351	0.106	0.044	0.026	0.011	0.006	0.003
10	7.2	2.02	0.864	0.544	0.396	0.15	0.073
	0.366	0.138	0.043	0.021	0.018	0.007	0.004
15	7.47	2.09	0.896	0.564	0.411	0.155	0.076
	0.42	0.126	0.043	0.027	0.025	0.007	0.004
20	7.6	2.13	0.912	0.574	0.418	0.158	-
	0.479	0.163	0.031	0.026	0.016	0.009	-

Table 3.2: Relative error bounds for the case Pd = 0.5 and with zero idle time constraint

n/m

m	2	5	10	15	20	50	100
2	16	3.2	1.6	1.07	0.8	0.32	0.16
	0.355	0.08	0.019	0.019	0.014	0.007	0.003
5	6.4	5.12	2.56	1.71	1.28	0.512	0.256
	0.35	0.136	0.058	0.036	0.014	0.009	0.005
10	28.8	5.76	2.88	1.92	1.44	0.576	0.288
	0.366	0.177	0.057	0.029	0.025	0.011	0.005
15	29.9	5.97	2.99	1.49	1.49	0.597	0.299
	0.42	0.162	0.057	0.036	0.035	0.011	0.005
20	30.4	6.08	3.04	2.03	1.52	0.608	-
	0.479	0.21	0.068	0.035	0.021	0.012	-

a priori error bound
posterior error bound

Table 3.3: Relative error bounds for the case Pd = 0 and without zero idle time constraint.

n/m

m	3	5	10	15	20	50	100
2	48.1	13.3	1.66	0.614	0.328	0.17	0.247
	0.315	0.043	0.01	0.007	0.004	0.001	0
5	200	7.62	1.49	22.7	11.8	1.66	0.331
	0.433	0.139	0.025	0.011	0.005	0.001	0
10	116	80.2	67	6.06	3.18	1.56	2.34
	0.462	0.137	0.027	0.013	0.006	0.001	0
15	769	499	16.75	11.6	6.05	10.1	0.21
	0.376	0.146	0.028	0.015	0.068	0.001	0
20	4707	121	31.25	22.5	7.34	0.616	-
	0.582	0.153	0.03	0.013	0.006	0.001	-

Table 3.4: Relative error bounds for the case Pd = 0 and with zero idle time constraint.

n/m

m	3	5	10	15	20	50	100
2	160	133	15.2	19.3	13.7	17.3	49.9
	0.629	0.13	0.048	0.054	0.037	0.014	0.007
5	1331	934	34	749	507	171	67.3
	0.865	0.418	0.124	0.086	0.053	0.028	0.012
10	950	1051	1569	202	138	162	476
	0.924	0.411	0.132	0.108	0.062	0.027	0.014
15	6736	6683	393	390	264	1047	42.8
	0.751	0.439	0.138	0.116	0.083	0.028	0.014
20	4259	1649	740	758	3210	64	-
	1.16	0.46	0.153	0.105	0.062	0.027	-

Table 3.5: Ratios of a priori to posterior error bounds
for the case Pd = 0.5 and without zero idle time constraint.

n/m

m	2	50	10	15	20	50	100
2	11	18	34	22	22	17	21
5	18	17	17	19	32	17	22
10	20	15	20	26	22	21	18
15	18	17	21	21	16	22	19
20	16	13	18	22	26	18	

Table 3.6: Ratios of a priori to posterior error bounds
for the case Pd = 0.5 and with zero idle time constraint.

n/m

m	2	50	10	15	20	50	100
2	40	45	89	56	60	46	53
5	38	18	44	44	91	57	51
10	33	79	51	66	58	52	58
15	37	71	52	55	43	54	60
20	30	64	45	58	72	60	-

Table 3.7: Ratios of a priori to posterior error bounds
for the case Pd = 0 and without zero idle time constraint.

n/m

m	2	5	10	15	20	50	100
2	153	309	166	88	82	170000	2470
5	462	55	60	2063	2360	1660	1655
10	251	584	2481	466	530	1560	11700
15	2045	3418	598	778	756	10100	1050
20	8087	791	1008	1730	12233	616	-

Table 3.8: Ratios of a priori to posterior error bounds
for the case Pd = 0 and with zero idle time constraint.

n/m

m	2	5	10	15	20	50	100
2	255	1023	317	402	370	1237	7128
5	1539	223	274	8709	9579	6107	3608
10	1028	2587	11866	1870	2226	6000	34000
15	8969	15227	2662	3362	3180	37392	3057
20	36716	3565	4836	7219	51774	2370	-

Table 5.1: g(k,q,x) and h(k,q,x)

g(k,q,x)

			x		
			0	1	2
k=1	q	0	0	1	4
		1	0	4	5
		2	0	2	6

			x		
			0	1	2
k=2	q	0	0	2	8
		1	0	8	10
		2	0	4	12

			x		
			0	1	2
k=3	q	0	0	3	12
		1	0	12	15
		2	0	6	18

			x		
			0	1	2
k=4	q	0	0	4	16
		1	0	16	20
		2	0	8	24

h(k,q,x)

			x		
		0	1	2	
k=1	q	0	4	1	0
		1	3	3	0
		2	0	0	0

			x		
			0	1	2
k=2	q	0	0	0	0
		1	2	0	0
		2	8	2	0

			x		
			0	1	2
k=3	q	0	0	0	0
		1	0	0	0
		2	0	0	0

			x		
			0	1	2
k=4	q	0	0	0	0
		1	0	0	0
		2	0	0	0

Table 5.2: u(q,k)

u(k,q)

		q		
		0	1	2
	1	0	1	2
k	2	0	0	1
	3	0	0	0
	4	0	0	0