SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853–7501

TECHNICAL REPORT NO. 823

September 1988

A PRICE–DIRECTED APPROACH TO REAL–TIME SCHEDULING OF PRODUCTION OPERATIONS

by

R. Roundy, W. Maxwell, Y. Herer
S. Tayur and A. Getzler

This research was supported by NSF Grants ECS–8404641 and DMC–8451984, and also by AT&T Information Systems and DuPont Corporation.

Abstract

Most past approaches to job shop scheduling are either highly myopic, or they are unable to adapt effectively to the stream of unforeseen disruptions that characterizes almost all manufacturing systems. We propose a two-module scheduling system that is both robust and non-myopic. Periodically a Planning Module develops a global view of what is likely to happen in the facility over the next one to four weeks. It then passes information down to the Dispatching Module. The Dispatching Module is based on microprocessors that are located on the factory floor. Whenever a machine is about to become idle, a fast optimization algorithm is run on the closest microprocessor to decide which job should be done next. The data provided by the Planning Module allows this decision to be made in a non-myopic manner.

1 Introduction

We consider the problem of job shop scheduling. A typical job shop consists of several machines. Each machine is capable of performing a variety of operations, but machines can perform only one operation at a time. New jobs arrive at various times. Each job consists of a sequence of operations which are to be performed on different machines. We assume that the operations corresponding to a job must be performed in a predetermined sequence, and that each operation must be performed on a predetermined machine. When all of the prescribed operations on a job are completed, the job leaves the shop.

Associated with each job is a weight and a due date. Our task is to schedule the operations over time so as to minimize the total (weighted) tardiness. If a job is not completed by its due date, a penalty is imposed which is proportional to its weight and to the tardiness of the job.Thus, if a job has a weight w and is tardy by k days, the weighted tardiness penalty incurred is wk.

What we have described so far is a simplified version of a very common type of manufacturing system. However in the real world, many complications not addressed by the classical model are present and are highly significant. The most important of these are the variability of the processing times, machine breakdowns and other unforeseen disruptions.

Most past research on job shop scheduling can be divided into two categories: combinatorial optimization and dispatching rules. Much effort has been expended in the area of combinatorial optimization, but the success has been limited. Fast polynomial-time algorithms have been developed for a few simple problems, but most problems with direct real-world applications have been shown to be NP-Hard. Algorithms have been developed for solving some NP-Hard scheduling problems. These problems typically model systems with special structure, and the algorithms are limited to very small data sets (by industrial standards) because of high run times.

Combinatorial optimization algorithms produce pre-determined schedules that are laid out for the entire shop over a fixed planning horizon. Pre-computed, feasible schedules are valuable. They are a useful tool for supporting short-term planning decisions, such as short-term manpower planning and inventory management. They are also useful as a target and

as a yardstick by which actual performance can be evaluated.

However pre-computed schedules do not adequately provide support for the actual sequencing and scheduling decisions that need to be made. Pre-computed schedules are insufficiently robust. Unforeseen delays and disruptions make them infeasible almost as soon as they are created. It is not realistic to compute a new schedule every time a disruption occurs because the run times of the algorithms and the frequency of disruptions are both high, and because up-to-date information on the current status of the entire factory would have to be gathered after every disruption. Since frequent disruptions occur in virtually all manufacturing systems, a practical scheduling method should be flexible to accommodate them.

Research in the second category focuses on determining the effectiveness of various dispatching rules under different measures of performance. Dispatching rules always yield an implementable schedule, regardless of unforeseen delays and other problems that arise during the course of a workday. The computations required are very simple and very little information is required to make a decision. However, dispatching rules are myopic in both space and time *i.e.*, an individual machine has no information regarding the current needs of other machines or what is likely to happen in the near future. Hence decisions are made on very limited information and overall efficiency is sacrificed.

We believe that it is time to re-think what a scheduling system should accomplish. In our opinion a scheduling system should produce pre-computed, feasible schedules to support the short-term planning functions such as the ones decribed above. However it should also provide real-time decision support to the workers on the factory floor who ultimately make the actual sequencing decisions. It should base its recommendations on information reflecting what is probably going on elsewhere in the factory, and on what is likely to occur in the future.

We propose a two-module approach that seeks to address these needs. The Planning Module consists of a computation that attempts to get an accurate global view of what is likely to happen in the shop over the next one to four weeks. This module is run periodically on a mainframe, say once per day or once per week. It is also run when there is a major disruption. Its outputs are:

1. A detailed schedule for the next 1-4 weeks, used to support short term planning functions with regard to overtime, subcontracting, shipping schedules, raw material delivery, etc.

2. Information to be passed to the Dispatching Module in the form of costs associated with performing a given operation at a given time.

The Dispatching Module is based on microprocessors which are physically located on the factory floor. Whenever a machine is about to become idle, a fast optimization algorithm is run on the nearest of these microprocessors to determine which operation should be performed next. This algorithm uses up-to-the-minute local information, such as the current status of the machine and which operations are currently available for processing. It also uses the costs passed from the Planning Module, which enable it to anticipate the future consequences of the decision currently being made. The Dispatching Module is an on-line decision support system. The process of selecting the next job to be worked on is interactive. The algorithm may report several near optimal solutions, and it may evaluate suggestions made by the worker. The Dispatching Module also periodically provides information on the current status of the factory to the Planning Module.

The advantages of the two module approach are clear. All the decisions are made interactively. They are based on current local information and on data computed by the Planning Module which takes into account what is probably happening elsewhere in the factory and what is likely to happen in the future. Minor disruptions or delays do not pose a problem as the schedules produced are always implementable. If there is a major disruption, the Planning Module computation is rerun. Up-to-date information on the status of the factory is gathered by the microprocessors of the Dispatching Module and communicated to the mainframe. The mainframe then generates a new set of costs that take the nature of disruption into account. Thus, in this two-module approach the decisions taken are not myopic and they always produce an implementable schedule. In this way, the flexibility of dispatching rules can be retained without ignoring the goal of global efficiency in the job shop.

There clearly is a need for an approach which is both robust and non-myopic. We are aware of three other current research efforts that address this need. [Birge and Dempster 87] and [Bean *et al.* 87] consider systems

with multiple jobs, each of which requires only one operation. There are multiple machines and jobs have alternative routings. For example, it may be possible to process a given job on either machine A or machine B, and the processing time of the job may be machine dependent. They propose that a pre-planned schedule be computed, and that when disruptions force deviations from the pre-planned schedule, that a computation be performed to decide how to merge back into the pre-planned schedule at some future date. Two heuristics are presented.

[Gallego 88a] and [Gallego 88b] considers a version of the Economic Lot Scheduling Problem, in which several products with constant demands are produced on a single machine. The machine can produce only one product at a time, and there are setup costs and setup times. Gallego's approach is similar to Birge in that there is a pre-planned schedule that one would ideally like to follow, and when disruptions occur one computes an optimal transition from an arbitrary starting point to the pre-planned schedule. In Gallego's model the pre-planned schedule is cyclic and the time horizon is infinite.

[Morton *et al.* 86] and [Morton *et al.* 88] discuss an evolving cost based method of scheduling called Patriarch. Patriarch is designed to support planning decisions as well as scheduling decisions. The underlying philosophy in their approach has many similarities to this work, although the research was done independently. The basic idea behind their approach is that using a machine in any time period costs an amount that is dependent on the status of the factory. The goal is to schedule jobs on the machines based on the machine usage costs, the tardiness costs and interest on delayed scheduling costs. They calculate costs based on the following assumptions:

1. Material, labor, and machine costs are incurred only when the job has begun processing.

2. Extra direct holding costs are incurred if the job is done early.

3. The loss of interest on revenues received for late jobs is a real cost.

4. NPV is the only true objective function for a firm; other views taken are too simple, o r are trying to avoid the real problem of estimating the 'cost' of tardiness.

The effect of assumptions 1 and 2 is to create an incentive to defer production.

By contrast, our main goal is to provide a real-time support system for scheduling decisions. We view our system as supporting planning functions, such as manpower management and raw material acquisition, only in feedback mode. Therefore we make the following assumptions about costs.

1′. Material acquisition and manpower allocation have already taken place, and interest on the direct machine usage costs does not create a significant incentive to defer production.

2′. Extra direct holding costs which are incurred if a job is done early are small.

The impact of these assumptions is to remove the incentive to idle a machine when there is work it could be doing.

Another major difference in the approach of [Morton *et al.* 86] and [Morton *et al.* 88] and our approach is in the method used for computation of machine usage costs. They estimate machine usage costs be simulating the factory until a busy period ends, and using the schedule generated to determine the costs. The simulations are rerun until a satisfactory set of costs is obtained. Our approach, described in section 2 below, is considerably different. However, early on we experimented with an approach which is similar in spirit to theirs. This approach was tested on a subset of the data sets used in stage one of our computational tests. We compared these results with those obtained from using the proposed two module approach and the latter were found to be superior. The superiority was especially evident with longer time horizons and scattered due dates which are more representative of the real world. For this reason we abandoned this approach.

In this study, we have primarily concentrated on testing our two-module concept. The key questions that need to be considered are:

1. What data need to be passed to the lower-tier from the top-tier?

2. What algorithms should be used in the two modules?

3. How should we measure the effectiveness of the algorithms and of this approach?

The costs that we are passing down from the Planning Module to the Dispatching Module are the estimated cost of completing an operation at a given time. We have also decided what algorithms are to be used in the Planning and Dispatching Modules. The focus of our research is on validating the two module concept. Consequently we have not yet attempted to make our algorithms and data structures computationally efficient.

The paper is organized as follows: In section 2, our assumptions about the factory are formally stated and the Planning Module is discussed. The weighted tardiness problem is formulated as an integer programming problem (IP). We dualize selected constraints of the above mentioned IP. Any feasible solution to the Lagrangian relaxation of the IP is a valid lower bound for the IP. The lower bound is improved by using standard subgradient optimization procedures. The values of the Lagrangian dual variables have an intuitive meaning. They correspond to machine 'prices' -the cost of using a machine at a given time. These machine prices are used to compute the cost of completing an operation at a given time. These operation costs are passed down to the Dispatching Module. The Dispatching Module is discussed in section 3. Section 4 describes the computational tests performed and the extremely encouraging results obtained.

2 The Planning Module

2.1 The Job Shop

Our factory consists of M machines, each capable of performing a variety of operations, but each machine can perform only one operation at a time. Preemption is not allowed. There are n jobs in the shop, all available for immediate processing. Each job j consists of o_j operations which must be performed in a specified order: $1, 2 \ldots o_j$. Thus, the sequence is serial and fixed. Operation i of job j must beperformed on machine m_{ji}. The machines on which the operations are to be performed are pre-determined. The processing times are variable; we assume that the processing time for operation i of job j is a uniformly distributed random variable with mean p_{ji}. Each job j has a due date d_j. A penalty is imposed if a job is not completed by its due date. If job j is tardy by k days, the penalty imposed

is kw_j. Given a schedule, the sum total of this tardiness penalty over all jobs is called the total weighted tardiness of the schedule. Our task is to find a schedule that has a low value of the total weighted tardiness.

2.2 Formulation of the weighted tardiness problem

We now formulate the job shop scheduling problem as an integer programming problem. This formulation is the basis of the Planning Module, but it is not used in the Dispatching Module.

In this formulation we assume that the processing times are deterministic with a value equal to the expected processing time. However, in conducting our simulation of the overall system, we have allowed for variability in the processing times. Furthermore, we assume in this formulation that time is discrete, and that T is the last time period in the horizon. The objective is to minimize the total weighted tardiness.

The integer programming formulation (P) of the job shop scheduling problem given below is due to [Pritsker and Watters 68]. A more compact formulation is available [Pritsker *et al.* 69], but (P) has structure which can be exploited to our advantage. We will dualize some of the constraints of (P) to obtain a Lagrangian relaxation of (P). The Lagrangian relaxation of (P) can be decomposed into n independent sub-problems, one for each job. These sub-problems are the duals of minimum flow problems with special structure. This structure enables us to solve them in linear time using a special purpose algorithm.

Let X_{jit} be a variable such that $X_{jit} = 1$ if operation i of job j has started by time t, and $X_{jit} = 0$ otherwise. We formulate the integer programming problem with the following constraints:

1. Once started an operation remains started in all subsequent time periods.

$$X_{j,i,t+1} \geq X_{jit} \qquad 1 \leq j \leq n, 1 \leq i \leq o_j, 1 \leq t < T. \tag{1}$$

2. An operation cannot start until its predecessor is completed.

$$X_{j,i,t-p_{j,i}} \geq X_{j,i+1,t} \qquad 1 \leq j \leq n, 1 \leq i < o_j, p_{ji} \leq t \leq T. \tag{2}$$

3. At most one job can be done on a particular machine in a given time period.

$$\sum_{\{j,i:m_{ji}=m\}} (X_{jit} - X_{j,i,t-p_{ji}}) \leq 1 \begin{cases} \forall \text{ machines } m, \\ 1 \leq t \leq T. \end{cases} \tag{3}$$

Recall tha a cost of w_j is incurred for each time period after d_j during which job j has not been completed. Thus our objective function is the following:

$$\text{minimize} \quad \sum_j w_j \sum_{t > d_j - p_{j,o_j}} (1 - X_{j,o_j,t}).$$

Therefore, the problem of choosing an optimal schedule is

$$\text{(P)} \quad \text{maximize} \sum_j w_j \sum_{t > d_j - p_{j,o_j}} X_{j,o_j,t} - \sum_j w_j (T - d_j + p_{j,o_j})$$

such that (1), (2), and (3) hold, and $X_{jit} \in \{0,1\} \forall\, j,i,t$.

2.3 Dualization

Note that the objective function of (P) can be restated as

$$\text{maximize} \quad \sum_i \sum_j \sum_t (w_{jit} X_{jit}) - \sum_j w_j (T - d_j + p_{j,o_j})$$

where $w_{jit} = w_j$ if $i = o_j$ and $t > d_j - p_{j,o_j}$, and $w_{jit} = 0$ otherwise. Further, note that the second term is constant. Dualizing the machine capacity constraint (3) with Lagrangian multipliers λ_{mt}, we get:

$$\begin{aligned} (\text{P}_\lambda) \qquad \text{maximize} \qquad & \sum_j \sum_i \sum_t [(\lambda_{m_{ji},t+p_{j,i}} - \lambda_{m_{ji},t} + w_{jit}) X_{jit}] \\ & + \sum_j w_j (-T + d_j - p_{j,o_j}) + \sum_m \sum_t \lambda_{mt} \end{aligned}$$

such that

$$X_{jit} - X_{ji,t+1} \leq 0 \qquad \forall\, j,i,t, \tag{1}$$
$$X_{j,i+1,t} - X_{j,i,t-p_{ji}} \leq 0 \quad \forall\, j,i,t, \tag{2}$$
$$X_{jit} \in \{0,1\} \qquad \forall\, j,i,t. \tag{4}$$

Note that (P_λ) can be decomposed by job into n independent sub-problems, one for each job. The sub-problem corresponding to job j is $(P_{\lambda j})$. Dropping the job subscript j, $(P_{\lambda j})$ can be written as:

$$(P_{\lambda j}) \qquad \text{maximize} \sum_i \sum_t [(\lambda_{m_i,t+p_i} - \lambda_{m_i,t} + w_{it})X_{it}]$$

such that

$$X_{it} - X_{i,t+1} \leq 0 \qquad \forall\, i,t, \tag{1'}$$
$$X_{i+1,t} - X_{i,t-p_i} \leq 0 \quad \forall\, i,t, \tag{2'}$$
$$X_{it} \in \{0,1\} \qquad \forall\, i,t. \tag{4'}$$

Observing that the constraints in $(P_{\lambda j})$ are dual network flow constraints we can replace $(P_{\lambda j})$ by its linear relaxation. Therefore, (4′) can be replaced by:

$$X_{it} \leq 1 \qquad \forall\, i,t \tag{5}$$

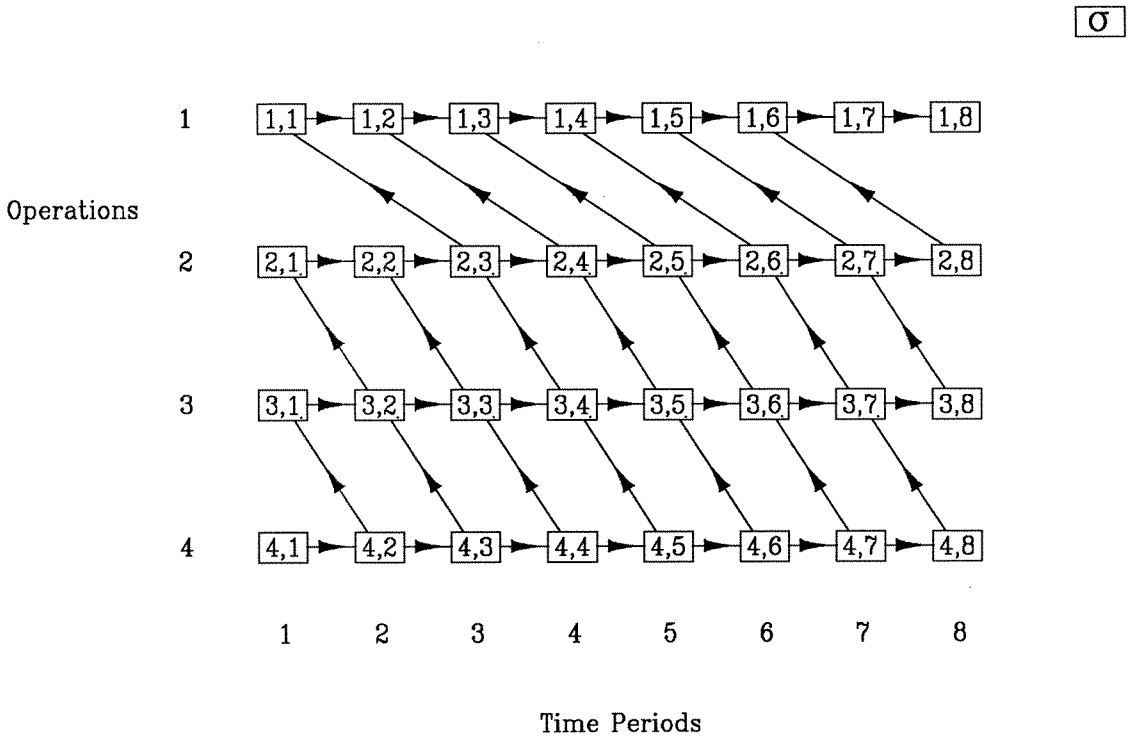
$$X_{it} \geq 0 \qquad \forall\, i,t \tag{6}$$

By repeated application of (1) and (2), we see that $X_{1T} \leq 1$ yields (5). Thus, we can replace (5) by:

$$X_{1T} \leq 1 \tag{7}$$

After replacing (4′) in $(P_{\lambda j})$ by (6) and (7), only one constraint of $(P_{\lambda j})$ has a non-zero right hand side. The dual to $(P_{\lambda j})$ is therefore a max-flow problem on a network. The topology of the network is illustrated in Figure 1. There is a sink node σ. There is also a node for each (i,t) pair. There are three sets of (directed) arcs. The first set of arcs connects (i,t) to $(i,t+1)$ for all i and all $t < T$. The second set of arcs connects (i,t) to $(i-1, t-p_{i-1})$ for all $i > 1$ and $t - p_{i-1} \geq 0$. The third set of arcs is not shown in Figure 1; it connects each of the nodes (i,t) to the sink node σ. This is a network with very special structure which enables us to develop a special purpose algorithm that solves the max-flow problem in linear time.

Figure 1: Network Topology.



A detailed formulation of the problem and the algorithm that solves it are presented in Appendix B.

Observe that for every set of Lagrangian multipliers λ_{mt}, the sum of the solutions to $(P_{\lambda j})$ can be used to obtain a lower bound on the optimal cost, or equivantly, an upper bound on the optimal profit in (P). We use standard subgradient optimization procedures to select values of λ_{mt} that improve the lower bound. We use a geometrically decreasing step size with ratio .975 and stop when the step size drops below machine precision. This procedure yields a constant number of iterations. This is probably not the most effecient search procedure, but as stated earlier we are not after an efficient implementation, but rather trying to test the performance of the two module approach. The lower bound itself is of secondary importance to us. In addition to the lower bound, two very useful pieces of information are obtained from the above computations. The first is the set of λ's. Recall that the lambda's are the Lagrangian variables that we used to dualize the machine capacity constraints. Therefore, intuitively they correspond to machine prices. Thus, λ_{mt} can be thought of as the price of using machine m in time period t. The second piece of information is obtained when we solve the maxflow problems. Recall that an optimal solution to a maxflow problem also yields a minimum cut. These cuts are useful. They correspond to the X variables of $(P_{\lambda j})$ and (P). For problem $(P_{\lambda j})$, the nodes that are to the right of (respectively, to the left of) the cut correspond to pairs (i,t) such that operation i of job j has (respectively, has not) started by time t.

2.4 Costs passed down to the Dispatching Module

Once a satisfactory set of machine prices λ is obtained, a cost function for completing a given operation at a given time is generated. These cost functions are passed down to the Dispatching Module. Whenever a machine becomes idle, a one-machine scheduling problem based on these functions is solved to decide which operation should be performed next.

The cost function is calculated as follows: let C_{jit} be the estimated cost of completing operation i of job j at time t. Recalling that o_j is the final operation of job j, we have

$$C_{j,o_j,t} = \sum_{t-p_{j,o_j} < s \leq t} \lambda_{m_{j,o_j},s} + w_j(t-d_j)^+ \tag{8}$$

where $A^+ = max\{A, 0\}$. The intuition behind (8) is as follows: The cost of completing the final operation at time t comes from two sources. The first term corresponds to the cost of using machine m_{j,o_j} for the time periods $t - p_{j,o_j} + 1$ through t. Recall from the previous section that λ_{mt} is an estimate of the value of time on machine m at time t. The second term is the tardiness penalty that is imposed if the job is tardy.

Recall that when we solved the maxflow problem corresponding to job j, we obtained a cut that partitioned the nodes of the network. The nodes to the right of the cut correspond to the pairs (i,t) such that operation i of job j has started by time t, i.e. $X_{jit}=1$. The node (i, z_{ji}) is the node corresponding to operation i of job j immediately to the right of the cut, i.e. $X_{j,i,z_{ji}}=1$ and $X_{jit}=0 \forall\ t < z_{ji}$. Intuitively, this means that according to the Planning Module operation i of job j is not likely to start before time z_{ji}. For operations $i < o_j$ of job j, we compute C_{jit} as follows:

$$C_{jit} = \sum_{t-p_{ji} < s \leq t} \lambda_{m_{ji},s} + C_{j,i+1,(max\{z_{j,i+1},t\}+p_{j,i+1})} \tag{9}$$

The cost of completing operation i of job j at time t comes from two sources. The first term is the machine usage cost. The logic behind the second term is as follows: completing operation i of job j at time t means that the earliest j is available for the next operation is at time t. Recall from our discussion above that operation $i+1$ of job j is not likely to start before time $z_{j,i+1}$. Thus, we estimate that the starting time for operation $i+1$ is max($z_{j,i+1}$,t), and that operation $i+1$ will complete at time max($z_{j,i+1}$,t)+$p_{j,i+1}$.

Since the machine prices λ are not monotone, C_{jit} may not be monotone in t. This can cause an operation to be delayed even when the operation is available for processing and the machine is idle. According to our modeling assumptions this is clearly not desirable. Therefore, we impose monotonicity on C_{jit}. This is done as follows:

$$C^*_{jit} = \max_{s \leq t} C_{jis}$$

Now C^*_{jit} is a non-decreasing step function of time. The values of $C^*_{jit} \forall\ j,i,t$ are passed down to the Dispatching module.

3 The Dispatching Module

3.1 Introduction

All the computations described above are performed on a mainframe computer and constitute the Planning Module. These computations are performed periodically (once a day, weekly) or when there is a major disruption on the factory floor. The costs computed in the Planning Module are passed down to the Dispatching Module. The Dispatching Module is based on microprocessors located on the factory floor. When a machine is about to finish its current operation, the operator at this machine has to decide which one of the jobs available at his machine is to be processed next. The operator provides local information regarding which jobs are available to choose from to the microprocessor. Algorithms based on the microprocessor use this local information and the costs passed down from the Planning Module to provide decision support for the worker in determining which job should be processed next on the machine. Thus, the decisions are made interactively and in real time. The schedule thus generated is always implementable because it uses up-to-the minute local information, and it is non-myopic in space and in time because it is guided by the costs passed down from the Planning Module.

The Dispatching Module solves a one-machine scheduling problem, using only the jobs that are currently available for processing. It recommends the first job in the sequence for immediate processing. If the worker suggests that a different job, job A, be done next, the Dispatching Module computes a schedule in which A is the first job, and compares the cost of this schedule to the cost of the schedule it computed. We now describe the one-machine scheduling problem that the Dispatching Module solves, and the algorithm used to solve it.

3.2 The One-Machine Scheduling problem

The scheduling problem solved by the Dispatching Module is the following. We are given a machine and k jobs that need to be processed on this machine. Only one job can be processed at a time, and all jobs have to be processed. Pre-emption is not allowed. Associated with each job

J_i are processing times $p_i \geq 0$ and a non-decreasing cost function $C_i(t)$ which represents the cost incurred if job J_i finishes at time t. We want to find a sequence of the jobs $J_{(1)}, J_{(2)} \ldots J_{(k)}$ that minimizes the total cost $C_T = \sum_1^k C_i(t_i)$, where t_i is the completion time of job i. In the context of our two-module approach to job shop scheduling the jobs to be processed are those currently available for processing at a given machine at a given point in time, and the cost functions $C_i(t)$ are the functions C^*_{jit} computed in section 2.4.

Several special cases of this problem have been extensively studied [Lawler *et al.* 82] and [Lenstra and Rinnooy Kan 84]. The best known of these is the weighted tardiness problem, in which $C_i(t) = w_i.max\{0, t - d_i\}$ [Lawler 71] showed that the weighted tardiness problem is unary NP-Hard. Numerous branch and bound algorithms, dynamic programming algorithms and heuristics have been proposed for this problem [Lawler *et al.* 82] and [Lenstra and Rinnooy Kan 84].

We propose a simple heuristic for solving this problem. It is similar in spirit to the heuristic of [Vepsalainen and Morton 87]. The *weight* of a job is a measure of its urgency. Let $0 < \rho < 1$ and $\bar{\rho} \equiv 1 - \rho$. Note that $C_i(s + p_i + 1) - C_i(s + p_i) \geq 0$ is the cost of delaying a job i scheduled to start at time s by one time unit. At time t, we estimate the weight of job i to be:

$$\begin{aligned} \tilde{C}_i(t) &\equiv \sum_{s=0}^{\infty} \bar{\rho}\rho^s (C_i(s + t + p_i + 1) - C_i(s + t + p_i)) \\ &= \sum_{s=0}^{\infty} \bar{\rho}^2 \rho^s C_i(s + t + p_i + 1) \end{aligned}$$

The parameter ρ determins how much weight we give to future information. Based on the formula for $\tilde{C}_i(t)$, the average look-ahead time is $\sum_{s=0}^{\infty} \bar{\rho}\rho^s = \frac{\rho}{\bar{\rho}}$. We suggest setting $\frac{\rho}{\bar{\rho}} = \alpha\bar{p} \Leftrightarrow \rho = \frac{\alpha\bar{p}}{1+\alpha\bar{p}}$ where $\bar{p}$ is the average processing time of the jobs to be processed. In our experiments and those of [Vepsalainen and Morton 87] $\alpha = 2.25$ worked well.

If we are trying to decide which of a given set of jobs to schedule next on a given machine at a given time t, we select the job j for which $\frac{\tilde{C}_j(t)}{p_j}$ is maximal. If we want a complete schedule for a number of jobs on a single machine, we obtain it by applying this rule repeatedly each time the machine becomes idle.

This approach does not always produce schedules which are locally optimal. We experimented with three approaches for solving the one-machine problem: The simple procedure described above, the simple procedure followed by a sequence of adjacent pairwise interchanges to achieve local optimality, and the optimal solution (computed by dynamic programming). In the context of our two-module approach to job shop scheduling, when there was randomness in the processing times, the first of these rules exhibited the best performance. Therefore it was chosen.

4 Computational Results

The performance of the one-machine heuristic and the two-module approach developed above was compared to a number of existing heuristics. The comparison was done by randomly generating a number of data sets. Each data set contains routing information of the jobs, expected processing times of the operations, and the due dates of each of the jobs. The data sets generated had different numbers of jobs, different numbers of machines, different due date characteristics (loose(L), tight(T), clustered(C), scattered(S)), and had different routing characteristics (flowshop(F), jobshop(J)). The number of operations per job was set equal to the number of machines and each job visited each machine exactly once. The due dates were selected from a uniform distribution with mean and variance dependent on the size and other characteristics of the data set. The weights w_j were all set equal to one. For all data sets, the time horizon was large enuogh so that all jobs were completed before time T. We used $\rho = \frac{\alpha\bar{p}}{1+\alpha\bar{p}}$ where $\alpha = 2.25$.

Table 1 shows the different data sets that were generated. Nine different sizes of data sets were considered. For each size, the data sets were divided into eight categories: LCF, TCF, LSF, TSF, LCJ, TCJ, TSJ, LSJ. The explanation is as follows: L and T stand for loose and tight due dates respectively; C and S stand for clustered and scattered due date respectively; and F and J stand for flow and job shop configurations respectively. Thus LSJ means that the data set represents a job shop with jobs that usually have loose and scattered due dates.

Our simulations can be divided into two categories: deterministic and

stochastic. For each data set, the planning module is executed first using routing information and expected processing times to generate the costs. These costs are used in both the deterministic and stochastic simulations. In deterministic simulations, the true processing times of the operations are assumed fixed at their expected values. In stochastic simulations true operation times are generated from an uniform distribution whose mean is the expected processing time of that operation. Four levels of variability were considered: level 1 = ±20%, level 2 = ±40%, level 3 = ± 60%, and level 4 = ±80% of the mean processing times.

The performance measures obtained above were compared against six existing heuristics: FIFO, least slack, least number of operations remaining, SPT, random ordering, and a rule based on [Vepsalainen and Morton 87]. This was done as follows: For each data set deterministic simulations were performed first, one using each of the six heuristics. The heuristic amongst the six above that yeilded the lowest weighted tardiness is called the 'best heuristic' for this data set. This 'best heuristic' is used in stochastic simulations and the tardiness compared to our two module approach. The reason behind this comparison procedure is bassed the rationale that the dispatching rule to be used on the shop floor is decided prior to the true occurance of the events. Therefore, although some heuristic might outperform the 'best heuristic' in some cases of stochastic simulations it is not valid to perform stochastic simulations at each variability level for each of the six heuristics and then compare the results with our two module approach.

Our simulation study was divided into two stages. In the first stage, only data sets of sizes 1-6 were considered. Six replications were performed for each combination of parameters. Thus, in this stage a total of ($6\times8\times6$ =) 288 data sets were generated and tested. For each of these data sets eighty-one simulations were performed: one deterministic simulation, and twenty stochastic simulations for each of the four levels of variability in processing times. In this stage our approach was compared with all six heuristics mentioned above and the 'best heuristic'. The results are shown in Tables 2 and 3. Because of space considerations only the best of the competing heuristics are shown.

In the second stage, investigations were focused on determining the performance of our approach as the length of the time horizon increases and the variability of the true processing time increases. Sizes 1,4,7,8,

and 9 were used. Six replications were generated for each combination of parameters, but only scattered due dates were used. This is because we felt that scattered due dates, especially in the problems with longer time horizons, were more representative of the real world. Thus a total of $5 \times 4 \times 6 = 120$ data sets were generated and tested.

For each of the 120 data sets used in stage two, eighty-one simulations were run: One deterministic simulation, and twenty replications for each of the four different levels of variability in the stochastic simulations. The results from the computational study are shown in tables 4 and 5. Because of space considerations only the best of the competing heuristics are shown.

Our two level approach performed very well in all categories and was significantly superior to all other heuristics. We briefly summarize the results.

1. As the number of machines and number of jobs increased, our two-level approach increased its superiority over the other heuristics. This is extremely encouraging as this indicates that our approach is less myopic and takes a global viewpoint. In the real world, the problems sizes are usually very large, and this makes our approach far more attractive to use.

2. Although our two level approach uniformly out-performed the other heuristics, it did particularly well when the due dates were loose and/or scattered. This is also encouraging because in practice the due dates are typically not clustered.

3. Our two level heuristic did equally well in both the flow shop and the job shop categories.

4. As the variability in true processing times increased, we observe that the relative superiority of our approach drops. However, even at $\pm 80\%$ variation, our approach was significantly better than the others. The decrease in relative performance can easily be explained: as the randomness increases the mean processing times used in the Planning and Scheduling Module becomes less reliable.

Size	Machines	Jobs
1	6	6
2	9	9
3	12	12
4	6	18
5	9	27
6	12	36
7	6	30
8	6	42
9	6	54

Table 1

Category	CMU	SPT	LSlack	Best Heur	Two-Module
	ratio	ratio	ratio	ratio	average
Total	1.24	1.23	1.38	1.15	340
Size1	1.03	1.03	1.26	1.06	59
Size2	1.07	1.08	1.31	1.06	94
Size3	1.12	1.13	1.26	1.07	182
Size4	1.22	1.21	1.43	1.14	257
Size5	1.32	1.30	1.45	1.20	439
Size6	1.25	1.25	1.38	1.16	1008
Loose	1.48	1.46	1.48	1.26	199
Tight	1.14	1.13	1.34	1.11	481
Clustered	1.14	1.12	1.46	1.11	351
Scattered	1.34	1.35	1.30	1.20	329
Flow	1.22	1.18	1.40	1.15	415
Job	1.26	1.30	1.34	1.15	265
Level0	1.35	1.34	1.47	1.20	283
Level1	1.28	1.27	1.38	1.16	311
Level2	1.24	1.23	1.37	1.15	333
Level3	1.20	1.19	1.36	1.14	366
Level4	1.16	1.16	1.35	1.12	407

Table 2
Ratio of Average Heuristic Cost to Average Two Module Cost, and Average Two-Module Cost

Category	CMU	SPT	LSlack	Best Heur
Total	24	25	15	31
Size1	42	42	23	44
Size2	38	37	22	43
Size3	27	29	21	38
Size4	15	17	11	25
Size5	11	14	8	19
Size6	9	10	5	14
Loose	19	21	19	29
Tight	28	29	11	32
Clustered	28	31	10	32
Scattered	19	19	20	29
Flow	21	24	9	27
Job	26	26	21	34
Level0	13	14	7	26
Level1	19	21	12	26
Level2	24	25	16	30
Level3	29	30	19	33
Level4	33	34	21	38

Table 3
Percentage of times Heuristic beat Two-Module Approach

Category	CMU	SPT	LSlack	Best Heur	Two-Module
	ratio	ratio	ratio	ratio	average
Total	1.93	1.89	1.43	1.37	422
Size1	1.02	1.03	1.16	1.04	613
Size4	1.28	1.29	1.32	1.16	257
Size7	2.29	2.34	1.53	1.49	230
Size8	2.11	2.02	1.52	1.43	555
Size9	1.97	1.92	1.41	1.39	1007
Loose	3.04	2.94	1.57	1.54	206
Tight	1.57	1.55	1.39	1.32	637
Flow	1.71	1.58	1.52	1.42	532
Job	2.31	2.41	1.27	1.28	312
Level0	2.27	2.21	1.52	1.45	340
Level1	2.10	2.04	1.44	1.40	376
Level2	1.96	1.91	1.42	1.37	412
Level3	1.81	1.78	1.41	1.35	461
Level4	1.68	1.65	1.41	1.33	521

Table 4
Ratio of Average Heuristic Cost to Average Two Module Cost, and Average Two-Module Cost

Category	CMU	SPT	LSlack	Best Heur
Total	11	11	17	22
Size1	44	43	29	48
Size4	10	11	19	26
Size7	0.7	1.2	16	16
Size8	0.4	1.0	11	11
Size9	0.2	0.6	10	10
Loose	9	8	22	25
Tight	13	15	12	19
Flow	10	11	8	14
Job	12	12	26	31
Level0	7	8	8	19
Level1	10	10	14	18
Level2	11	11	19	22
Level3	13	13	21	24
Level4	14	15	24	28

Table 5
Percentage of times Heuristic beats Two-Module Approach

5 Conclusions

Our goal in this research was to test our two-module concept. The computational results herein clearly indicate that the two-module approach gives much better results than existing dispatching rules. However, the data structures and the algorithms used in this exploratory research are too inefficient for immediate implementation. Our next step is to develop efficient data structures and algorithms.

6 Appendix: The Max Flow Algorithm

In this appendix we formulate the dual of ($P_{\lambda j}$) as a max flow problem. We observe that the network has a special structure. We give a fast alogorithm for solving it, prove that the algorithm produces an optimal solution, and analyze its running time.

The dual of $(P_{\lambda j})$ is

$(D_{\lambda j})$ maximize z_{1T}

such that

$$\begin{aligned} v_{it} - v_{i,t-1} + y_{it} - y_{i+1,t+p_i} - z_{it} &= \Delta_{it} \quad \forall\, i,t, \\ v_{it} &\geq 0 \quad \forall\, i,t, \\ y_{it} &\geq 0 \quad \forall\, i,t, \\ z_{it} &\geq 0 \quad \forall\, (i,t) \neq (1,T), \\ z_{1T} &\leq 0 \end{aligned}$$

where $\Delta_{it} = -\lambda_{m_i,t+p_i} + \lambda_{m_i,t} - w_{it}\ \forall\, i,t$. The variables v_{it}, y_{it}, z_{it}, and z_{1T} correspond respectively to constraints (1'), (2'), (6), and (7).

The constraints of $(D_{\lambda j})$ define a network G. There is a node in G corresponding to each operation - time period pair. They are designated $\{(i,t)\colon 1 \leq i \leq I,\ 1 \leq t \leq T\}$ where $I = o_j$. G also has a source node σ. The set of arcs is $\{(i,t) \rightarrow (i,t+1)\colon 1 \leq i \leq I,\ 1 \leq t < T\} \cup \{(i,t) \rightarrow (i-1,t-p_{i-1}) : 1 < i \leq I,\ p_{i-1} < t \leq T\} \cup \{\sigma \rightarrow (i,t)\colon 1 \leq i \leq I,\ 1 \leq t \leq T\}$. We denote the flow on arc $\sigma \rightarrow (i,t)$ by z_{it}. The flow along every arc is constrained to be nonnegative with the exception of arc $\sigma \rightarrow (1,T)$, whose flow is constrained to be nonpositive. Our objective is to maximize z_{1T}, or, equivalently, to minimize $-z_{1T}$. Δ_{it} is the net supply at node (i,t). A negative net supply is interpreted as a demand. (Figure 1).

Solving the max flow problem also yields the minimum cut. The relationship between the cuts and the start times of the operations ($\min\{t\colon x_{it} = 1\}$) was discussed in section 2.3. The minimal cut will satisfy (1) and (2) because the arcs $(i,t) \rightarrow (i-1,t-p_{i-1})$ and $(i,t) \rightarrow (i,t+1)$ have infinite capacity. Therefore the minimal cut in the network of Figure 1 will run from the upper left to the lower right.

We are now ready to give an algorithm that solves $(D_{\lambda j})$. It is convenient to re-index the time subscripts. We will use $(i, t - \sum_{k<i} p_i)$ to denote node (i,t). Therefore arcs go from node (i,s) to node $(i,s+1)$ and from node (i,s) to node $(i-1,s)$.

We refer to the arcs $(i,s) \rightarrow (i,s+1)$ as "time arcs" and to the arcs $(i,s) \rightarrow (i-1,s)$ as "operation arcs". The intuition behind the algorithm is as follows: the variables v_{is} represent the flow out of node (i,s) along

time arcs, and the variables y_{is} represent the flow out of node (i,s) along operation arcs. The variables z_{is}, $(i,s) \neq (1,T)$ can be thought of as the unsatisfied demand at node (i,s), while Δ_{is} is the initial net supply at node (i,s). $-z_{1T}$ is the excess supply that flows out of the network. Thus, the objective of the max flow alogorithm is to minimize the excess supply that flows out of the network by using the supplies at nodes (i,t), $\Delta_{it} > 0$ to meet demands at nodes (k,s), $\Delta_{ks} < 0$ to the greatest extent possible.

The following algorithm solves $(D_{\lambda j})$.

THE FLOW ALGORITHM

Step 1 (Initialize). Set $v_{it} = 0$, $y_{it} = 0$, $z_{it} = -\Delta_{it}$ $\forall$ i,t, and set $V = \{(i,t)\colon 1 \leq i \leq I,\ 1 \leq t \leq T\}$.

Step 2 (Find a node (i,t) with a supply). If $V = \emptyset$ then stop. Otherwise choose the $(i,t) \in V$ such that for all other $(k,s) \in V$, either $t < s$ or $t = s$ and $k > i$. Remove (i,t) from V. If $z_{it} \geq 0$ then go to Step 2. Otherwise set $q = i$ and go to Step 3.

Step 3 (Try to use the supply at (i,t) in $\{(q,t)$: i > q$\}$). Set $q = q - 1$. If $q < 1$ or if $q \geq 1$ and $v_{qt} > 0$, then go to Step 4. If $q \geq 1$ and $z_{qt} \leq 0$ then mark arc $(q+1,t) \rightarrow (q,t)$, and go to Step 3. Otherwise augument the flow around the simple cycle $\sigma \rightarrow (i,t) \rightarrow (i-1,t) \rightarrow \ldots \rightarrow (q,t) \rightarrow \sigma$ by $\delta = \min(-z_{it}, z_{qt}) > 0$ units, and mark arc $(q+1,t) \rightarrow (q,t)$. After the augmentation, if $z_{it} = 0$ then the supply at (i,t) is exhausted, so go to Step 2. If $z_{it} < 0$ then $z_{qt} = 0$. Go to Step 3.

Step 4 (Pass the excess supply at (i,t) to (i,t+1)). If $t < T$ then augument the flow around the simple cycle $\sigma \rightarrow (i,t) \rightarrow (i,t+1) \rightarrow \sigma$ by $-z_{it} > 0$ units, mark the arc $(i,t) \rightarrow (i,t+1)$, and go to Step 2. If $t = T$ then augument the flow around the simple cycle $\sigma \rightarrow (i,T) \rightarrow (i-1,T) \rightarrow \ldots \rightarrow (1,T) \rightarrow \sigma$ by $-z_{it} > 0$ units and mark the arcs $(q,T) \rightarrow (q-1,T)$, $1 < q \leq i$. Then go to Step 2.

Note that every arc not adjacent to σ that has positive flow is marked.

Lemma .1 *The flow computed by the algorithm is feasible.*

Proof. We must show that (1) is satisfied, that v_{it}, $y_{it} \geq 0$ for all i,t, that $z_{it} \geq 0$ for all $(i,t) \neq (1,T)$, and that $z_{1T} \leq 0$. Initially all of these

conditions except the last two are satisfied. By induction on the iterations of Step 3 and Step 4, it is easily shown that they are still satisfied when the algorithm terminates.

With regard to the sign of z_{it}, $(i,t) \neq (1,T)$ it is clear that $z_{it} \geq 0$ when the iteration of Steps 2 through 4 for (i,t) ends. It is also clear that after that time the value of z_{it} will change only if it is strictly positive, and that it cannot become negative. Thus $z_{it} \geq 0$ when the algorithm terminates. Since $\Delta_{1T} = \lambda_{m_1,T} \geq 0$, z_{1T} starts out nonpositive and becomes more and more negative as the algorithm progresses. QED.

Lemma .2 *If* $(k,t) \rightarrow (k,t+1)$ *is marked and* $q < k$ *then there is an* i, $q \leq i \leq k$ *such that* $(i,t) \rightarrow (i,t+1)$ *is marked and* $(j,t) \rightarrow (j-1,t)$ *is marked for all* $i \geq j > q$.

Proof. The result clearly holds if $k = 1$. Steps 3 and 4 imply that if it is true for all $k < m$ then it is true for m. QED.

Let G_t be the subnetwork of G with node set $\{(i,s)\colon\ 1 \leq i \leq I,\ 1 \leq s \leq t\}$ and arc set $\{(i,s) \rightarrow (i-1,s)\colon\ 1 < i \leq I,\ 1 \leq s \leq t\} \cup \{(i,s) \rightarrow (i,s+1)\colon\ 1 \leq i \leq I,\ 1 \leq s < t\}$.

Lemma .3 *If* $(k,s) \rightarrow (i,t)$ *is an unmarked arc in* G_t, *then* (k,s) *is not connected to any node in the set* $\{(n,t)\colon\ 1 \leq n \leq i\}$ *by a path consisting entirely of marked arcs in* G_t.

Proof. By the topology of G_t, (k,s) and (n,t) can only be connected by a path of marked arcs in G_t if there is a node (k,r) such that both $(k+1,r) \rightarrow (k,r)$ and $(k,r) \rightarrow (k,r+1)$ are marked. But by Steps 3 and 4, if $(k,r) \rightarrow (k,r+1)$ is marked then $(k+1,r) \rightarrow (k,r)$ will not be marked. QED.

Let N be the set of nodes that are connected to $(1,T)$ by a path of marked arcs, and let $\bar{N}$ be its complement. We will show that $(N,\bar{N})$ is a cutset of minimal capacity.

Lemma .4 *If* $(k,t) \in N$ *then* $(i,t) \in N$ *for all* $i < k$. *If in addition* $t < T$ *then* $(k,t+1) \in N$ *and* $(j,t) \rightarrow (j,t+1)$ *is marked for some* $j \geq k$.

Proof. The hypothesis holds for $t = T$ by Step 4. Assume it holds for $t+1$, and let $(k,t) \in N$ and $1 \leq n < k$. We will show that $(n,t) \in N$. If $(k,t) \rightarrow (k,t+1)$ is marked then $(k,t+1) \in N$. If $(k,t) \rightarrow (k,t+1)$ is unmarked, Lemma B.3 implies that the path of marked arcs that connects (k,t) to σ must pass along $(j,t) \rightarrow (j,t+1)$ for some $j > k$. By the induction hypothesis $(k,t+1)$ is in N.

If $(i,t) \rightarrow (i-1,t)$ is marked for all $n < i \leq k$ then $(n,t) \in N$. Otherwise Lemma B.2 implies that (n,t) is connected to $(i,t+1)$ by a path of marked arcs for some $n \leq i < k$. By the induction hypothesis $(i,t+1)$ is in N, so $(n,t) \in N$. QED.

Lemma .5 *If (i,t)* $\in N$ *then* $z_{it} = 0$.

Proof. Steps 3 and 4 imply that if $(k,t) \rightarrow (k,t+1)$ is marked then $z_{qt} = 0$ for all $q \leq k$. The result now follows from Lemma B.4. QED.

Theorem *The Flow Algorithm solves (P).*

Proof. Lemma B.4 implies that the only arc that leads out of N is $(1,T) \rightarrow \sigma$. Therefore the capacity of the cutset $(N,\bar{N})$ is $\Sigma_{(i,t)\in N} \Delta_{it}$. Lemma B.5 implies that all arcs connecting N and $\bar{N}$ have zero flow except z_{1T}, so $z_{1T} = \Sigma_{(i,t)\in N} \Delta_{it}$. QED.

Example: Let I = 2, T = 3,
$\Delta_{11} = 0, \Delta_{12} = 2, \Delta_{13} = 0,$
$\Delta_{21} = -2, \Delta_{22} = -3, \Delta_{23} = -1.$

Steps	Augmenting	Path Flow	Marked Arcs
3			(2,1)→(1,1)
4	σ →(2,1)→(2,2)→ σ	2	(2,1)→(2,2)
3	σ →(2,2)→(1,2)→ σ	2	(2,2)→(1,2)
4	σ →(2,2)→(2,3)→ σ	3	(2,2)→(2,3)
3			(2,3)→(1,3)
4	σ →(2,3)→(1,3)→ σ	4	(2,3)→(1,3)

$z_{1T} = -4$. $\bar{N} = \emptyset$.

Let $z^{j\lambda}$ be the optimal flow $-z_{1T}$ from the subproblem $(D_{\lambda j})$. We define

$$f(\lambda) = \sum_j z^{j\lambda} + \sum_j w_j(-T + d_j - p_{j,o_j}) + \sum_m \sum_t \lambda_{mt}.$$

For every λ, $f(\lambda)$ is an upper bound on the optimal solution to (P), or equivalently, $-f(\lambda)$ is a lower bound on the optimal cost. To get the best possible bound, we must solve (D)

$$\text{(D)} \qquad \begin{array}{r} \text{minimize } f(\lambda) \\ \text{subject to } \lambda \geq 0. \end{array}$$

(D) can be solved using standard subgradient optimization techniques. f is evaluated by solving (P_λ). A subgradient of f at λ is given by

$$\frac{\partial f(\lambda)}{\partial \lambda_{mt}} = \sum_j \sum_{m_{ji}} (X_{j,i,t-p_{ji}} - X_{jit}) + 1 \qquad \forall \quad 1 \leq m \leq M, 1 \leq t \leq T$$

where X solves (P_λ), and M is the number of machines in the shop.

References

[Bean *et al.* 87] Bean, J., J. Birge, J, Mittenthal and C. Noon, "Matchup Scheduling with Multiple Resources, Release Dates and Disruptions", Working Paper, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, July 1987.

[Birge and Dempster 87] Birge, J. and M. Dempster, "Optimality Conditions for Match-UP strategies in Stochastic Scheduling and Related Dynamic Stochastic Optimization Problems", Working Paper, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, June 1987.

[Gallego 88a] Gallego, G., "Linear Control Policies for Scheduling a Single Facility After An Initial Disruption", Tech. Report No.

770, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, January 1988.

[Gallego 88b] Gallego, G., "Produce-Up-To Policies for Scheduling a Single Facility After An Initial Disruption", Tech. Report No. 771, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, January 1988.

[Lawler 71] Lawler, E., "A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness", *Annals of Discrete Mathematics*, 1971, 331-342.

[Lawler *et al.* 82] Lawler, E., J. Lenstra and A. Rinnooy Kan, "Recent Developments in Deterministic Sequencing and Scheduling: A Survey" In M. Dempster *et al.* (eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Company, 1982

[Lenstra and Rinnooy Kan 84] Lenstra, J. and A. Rinnooy Kan, "Scheduling Theory Since 1981: An Annotated Bibliography" In M. O'hEigeartaigh, J Lenstra and A. Rinnooy Kan (Eds.) *Combinatorial Optimization: Annotated Bibliographies*, Wiley, Chichester, 1984.

[Lawler 71] Lawler, E., "A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness", *Annals of Discrete Mathematics*, 1971, 331-342.

[Morton *et al.* 86] Morton, T., S. Lawrence, S. Rajagopalan and S. Kekre, "MRP-STAR PATRIARCH's Planning Module", Working Paper, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, December 1986.

[Morton *et al.* 88] Morton, T., S. Lawrence, S. Rajagopalan, and S. Kekre, "SCHED-STAR A Price-Based Shop Scheduling Module", Working Paper, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, February 1988.

[Pritsker and Watters 68] Pritsker, A. and L. Watters, "A Zero-One Programming Approach to Scheduling with Limited Resources", The RAND Corporation, RM-5561-PR, January 1968.

[Pritsker *et al.* 69] Pritsker, A., L. Watters and P. Wolfe, "Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach", *Managment Science: Theory* 16 (1969) 1 (Sept.), 93-108

[Vepsalainen and Morton 87] Vepsalainen, A., T. Morton, "Priority Rules For Job Shops With Weighted Tardiness Costs", *Management Science*, 33 (1987) 8 (August), 1035-1047