JOB SHOP SCHEDULING WITH DEADLINES

Egon Balas, Carnegie Mellon University, Pittsburgh, PA, Usa
Giuseppe Lancia, Carnegie Mellon University, Pittsburgh, PA, Usa
Paolo Serafini, University of Udine, Italy
Alkiviadis Vazacopolous, Fairleigh Dickinson University, Teaneck N.J.

Abstract: In this paper we deal with a variant of the Job Shop Scheduling Problem. We consider the addition of release dates and deadlines to be met by all jobs. The objective is makespan minimization if there are no tardy jobs, and tardiness minimization otherwise. The problem is approached by using a Shifting Bottleneck strategy. The presence of deadlines motivates an iterative use of a particular one machine problem which is solved optimally. The overall procedure is heuristic and exhibits a good trade-off between computing time and solution quality.

1. INTRODUCTION

In this paper we deal with a variant of the Job Shop Scheduling (JSS) Problem. The proposed model has all usual features of the well known Job Shop model with the addition of release dates and deadlines which must be met by all jobs. The objective is still the makespan minimization. We refer to this model simply as the Job Shop Scheduling Problem with Deadlines (JSSD).

There are practical reasons for introducing deadlines. The first reason is that the Job Shop model by itself does not capture an essential aspect of a production department. In practice there is never a single production run but new jobs arrive periodically and any schedule must be revised accordingly. Of course the Job Shop model can be reapplied each time new jobs become available, but there are some drawbacks to this approach. Most often in practice old jobs are to be completed within previously established deadlines and new jobs are to be simply completed as soon as possible. Finding the best possible makespan for the new jobs provides an indication for the setting of their deadlines in case they are needed either for later schedule computations or to contract delivery dates with the customer. Clearly there is the need to add deadlines to the usual Job Shop framework in order to cope with this production model.

As a second reason, there may be occasions when some machines are unavailable. This can be due to either machine breakdown or maintenance but also to the fact that a machine is already assigned to a higher priority job during a specific time interval. In other words the department manager could judge some jobs of higher priority and schedule them first. Then the other jobs would be scheduled but provision should be made that the machines are no longer always available. This can be accomplished by considering all operations of the higher priority jobs as dummy jobs with release dates and deadlines corresponding to their starting and completion times.

In general, release dates and deadlines constitute a useful tool for controlling the final schedule. It is often the case that a proposed schedule has to be revised due to factors outside the model. By changing a posteriori some release dates and/or deadlines it is often possible to modify the solution in the direction of a more desirable schedule. In spite of this fact and probably because of the difficulty of the problem, to the best of our knowledge the JSSD has never been investigated in the literature.

Since the Job Shop Problem is NP-hard and it is a special case of the JSSD, the latter problem is NP-hard as well.

In this paper we develop a heuristic procedure for the JSSD based on the Shifting Bottleneck approach of Adams, Balas and Zawack (1988). This approach iteratively identifies a bottleneck machine and sequences it optimally while holding fixed the job sequence on the machines already processed and ignoring the remaining machines. This is done by solving a one machine problem with release times and due dates. Although this problem is itself strongly NP-complete, it can nevertheless be solved efficiently in practice by a clever branch-and-bound procedure due to Carlier (1982). Once all the machines have been sequenced, each machine in turn is freed up and resequenced cyclically.

Recently two important improvements have been brought to this approach, both of which can be adapted to JSSD. The first one addresses the following issue. Sequencing a given machine may impose conditions on the sequence on some other machine, of the type that job i has to precede job j by at least a specified time lapse. We call these conditions delayed precedence constraints (DPC). Taking into account these DPC requires a major modification of Carlier's algorithm. Such a modified algorithm for solving the one-machine problem with DPC was developed by Balas, Lenstra and Vazacopoulos (1995) and shown to significantly improve the performance of the Shifting Bottleneck Procedure.

The second improvement consists of combining the Shifting Bottleneck approach, which can be viewed as optimization over the neighborhood defined by arbitrary changes in the sequence of any single machine, with optimization over the neighborhood defined by interchanging certain pairs of jobs anywhere in the overall job sequence. This is achieved by using the Shifting Bottleneck approach as a general framework and solving a sequence of one-machine problems with DPC by the above procedure, but replacing the cyclic reoptimization step by a guided local search procedure based on pairwise interchange of jobs. This approach described in Balas and Vazacopoulos (1994), has brought significant additional improvements.

Both of the above modifications of the Shifting Bottleneck approach are incorporated in our procedure for the JSSD problem. In particular, we developed an algorithm for solving to optimality the one-machine problem with DPC in the presence of job deadlines. This exact algorithm constitutes the backbone of our overall heuristic procedure for the JSSD problem. We also adapt the guided loal search to the presence of deadlines.

An important special case of the one-machine problem with deadlines has been investigated by Leon and Wu (1992): namely, the one-machine problem (without DPC) with unavailability over certain time intervals. This problem can be embedded into the framework of release dates and deadlines in the way outlined above. Our algorithmic approach is different from that of Leon and Wu (1992) and exhibits better computational performance.

In order to test our procedure for the JSSD we consider the continuous production model described at the beginning. First we apply the procedure to a set of data from a factory whose production environment

closely resembles a job-shop model. Second we apply the procedure to data derived by a famous benchmark instance. The results show that we have developed a useful tool for dealing with the problem at hand. In particular, we seem to have achieved a good trade-off between computing time and solution quality.

We have organized the paper as follows: in Section 2 we provide a formal description of the JSSD. In Section 3 we introduce and characterize an auxiliary problem which is the main tool in dealing with deadlines. In Section 4 a high level description of the Shifting Bottleneck Procedure is outlined; its basic building blocks are a particular one machine problem with deadlines and a local search procedure for the auxiliary problem. Sections 5 and 6 are devoted to the presentation of these two blocks respectively. Section 7 reports on the computational results for the one machine problem with deadlines and Section 8 reports on the computational results for the general JSSD procedure.

2. PROBLEM DESCRIPTION

We define the Job Shop Scheduling Problem with Deadlines as follows: a set $J = \{1, \ldots, |J|\}$ of jobs have to be processed on set M of machines within the minimum possible time, subject to the constraints that (i) the sequence of machines for each job is prescribed, (ii) each machine can process at most one job at a time, and (iii) jobs must start after given release dates and be completed before given deadlines. The processing of a job on a machine is called an operation, and its duration is a given constant.

We denote by

– $N = \{0, 1, \ldots, n\}$ the set of operations, with 0 and n two additional dummy operations used to identify the start and the end of the job processing;

– $\alpha(j)$ and $\omega(j)$ the first and the last operation respectively of job $j \in J$;

– A the set of ordered pairs of operations constrained by precedence relations, including $(0, \alpha(j))$ and $(\omega(j), n)$ for all $j \in J$;

– E_k the set of pairs of operations to be processed on machine k; let $E := \bigcup_k E_k$;

– p_i the duration or processing time for operation $i \in N$;

– r_j the release date of job $j \in J$;

– d_j the deadline of job $j \in J$;

– $d_{\max} := \max_j d_j$.

The variables to be determined are the operation starting times t_i. The set of t_i is called a schedule and t_n is called its makespan. The problem can be formally stated as

Problem JSSD

$$
\begin{aligned}
\widehat{t}_n := \min \quad & t_n & & & & \\
\text{s.t.} \quad & & & & & \\
& t_0 & = 0 & & & (1)\\
& t_j - t_i & \geq p_i & (i,j) \in A & & (2)\\
t_j - t_i \geq p_i \quad \vee \quad & t_i - t_j \geq p_j & & (i,j) \in E_k & k \in M & (3)\\
& t_{\alpha(j)} & \geq r_j & j \in J & & (4)\\
& t_{\omega(j)} + p_{\omega(j)} & \leq d_j & j \in J & & (5)
\end{aligned}
$$

By dropping in Problem JSSD the constraints (5) and (4), we obtain a standard Job Shop Problem. We remark that the constraints (4) can be embedded into a standard Job Shop Problem in a straightforward way by simply adding dummy operations of duration r_j. Therefore we shall assume that release dates have been already taken care of in this way. It is the presence of deadlines which makes the problem different from a standard JSS and more difficult.

We may represent the problem on a disjunctive graph $G = (N, A, E)$ with node set N, directed arc set A and undirected edge set E. The edges in E are orientable and are therefore called disjunctive whereas the arcs in A are called conjunctive. The length of an arc $(i, j) \in A$ is p_i, whereas the length of an edge $\{i, j\} \in E$ is either p_i or p_j depending on its orientation (if we choose (i, j) then it is p_i, otherwise it is p_j). Each machine k corresponds to a set N_k of nodes (operations) and a set E_k of edges which form a disjunctive clique.

Let $D = (N, A)$ denote the directed graph obtained from G by removing all the disjunctive edges. A machine selection S_k is a set of arcs obtained by orienting each edge in E_k. If S_k is acyclic then it induces a total ordering on the operations on machine k.

3. THE AUXILIARY JOB SHOP PROBLEMS

Let M' be a subset of machines. We define a relaxation RJSSD(M') of problem JSSD by imposing the constraints (3) only for the subset M'. We denote the optimal makespan of such a relaxed problem by $\widehat{t}_n(M')$. A selection S over M' is the union of machine selections S_k, for $k \in M'$. The selection is partial if M' is a proper subset of M, otherwise it is complete. A selection S gives rise to the directed graph $D_S = (N, A \cup S)$. A selection is acyclic if the digraph D_S is acyclic. Every acyclic selection S defines a family of schedules feasible for (1), (2), (4) and (3) restricted to M', but not necessarily for (5), and every such schedule induces an acyclic selection over the same machines. The minimum makespan over the schedules induced by S is equal to the length of a longest path in D_S. Let us denote this value by $t_n(S)$. An acyclic selection is feasible if (5) is also satisfied for at least one schedule of the family associated with S. Thus problem RJSSD(M') corresponds to finding an acyclic selection S over M' that is feasible and minimizes the length of a longest path in the directed graph D_S, that is $\widehat{t}_n(M') = \min_S t_n(S)$.

To any problem RJSSD(M') and nonnegative number τ, we associate a standard JSS problem (without deadlines) by appending to each job j a (last) dummy operation labeled $\omega'(j) := n + j$, whose processing time is $p_{\omega'(j)} = \max\{0; \tau - d_j\}$. We call this auxiliary problem PQ(M', τ). Note that in the new disjunctive graph, for each job j the dummy operation $\omega'(j)$ is preceded by operation $\omega(j)$ and followed by operation n, so that the makespan of the auxiliary problem is still given by the starting time of operation n A family of auxiliary JSS problems corresponding to different values of τ is used to solve problem RJSSD(M') as described in the next section. In Figure 1-a an example is provided of a JSS with 3 jobs, 7 operations ($n = 8$) and $d_1 = 22$, $d_2 = 14$ and $d_3 = 17$. The auxiliary problem for $\tau = 20$ is shown in Figure 1-b with the duration of dummy operation i written on arc (i, n)

By $T(\tau) = (T_0(\tau), \ldots, T_n(\tau), T_{n+1}(\tau), \ldots, T_{n+|J|}(\tau))$ we denote a schedule (solution) of the problem PQ(M', τ) with $T_n(\tau)$ being the corresponding makespan. By $\widehat{T}(\tau)$ and $\widehat{T}_n(\tau)$ we indicate the optimal schedule and optimal makespan of PQ(M', τ).

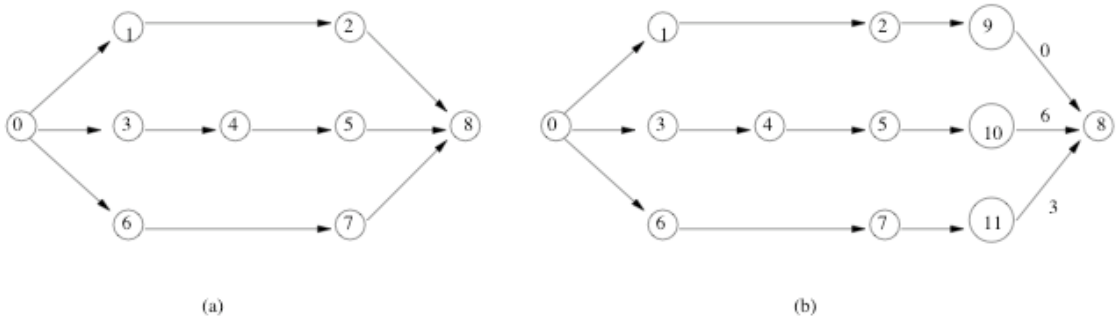

Figure 1

Given a schedule t for RJSSD(M'), for any τ we define the schedule $T(\tau)$ for PQ(M', τ), *associated with* t, as $T_i(\tau) := t_i$ for $i \in N \setminus n$, $T_{\omega'(j)}(\tau) := t_{\omega(j)} + p_{\omega(j)}$ for $j \in J$, and $T_n(\tau) := \max_{j \in J}\{t_{\omega(j)} + p_{\omega(j)} + \max\{0; \tau - d_j\}\}$. Conversely, given a schedule $T(\tau)$ for PQ(M', τ), we define the associated schedule t for RJSSD(M') as $t_i := T_i(\tau)$ for $i \in N \setminus n$, and $t_n := \max_{j \in J}\{T_{\omega(j)} + p_{\omega(j)}\}$

The relationship between problem RJSSD(M') and the family of auxiliary problems is characterized by the following Propositions. For the sake of simplicity we occasionally omit the dependence of $\widehat{T}_i(\tau)$ on τ and simply write $\widehat{T}_i$ if there is no risk of ambiguity. Similarly, when there is no risk of confusion, we will omit the suffix (M') from RJSSD(M').

Proposition 1: *If $\widehat{T}_n(\tau) \leq \tau$ then the schedule t associated with $\widehat{T}$ is feasible for problem RJSSD, i.e. $t_{\omega(j)} + p_{\omega(j)} \leq d_j$, for all $j \in J$.*

Proof: For all $j \in J$ we have

$$\tau \geq \widehat{T}_n(\tau) \geq \widehat{T}_{\omega'(j)} + p_{\omega'(j)} \geq \widehat{T}_{\omega(j)} + p_{\omega(j)} + p_{\omega'(j)} \geq \widehat{T}_{\omega(j)} + p_{\omega(j)} + \tau - d_j$$

Hence $\widehat{T}_{\omega(j)} + p_{\omega(j)} = t_{\omega(j)} + p_{\omega(j)} \leq d_j$. ∎

Proposition 2: *If $\widehat{T}_n(\tau) > \tau$ then all feasible schedules for RJSSD have makespan t_n at least equal to $\widehat{T}_n(\tau)$.*

Proof: Let t denote a feasible schedule for RJSSD with makespan t_n, and let $T(\tau)$ be the associated schedule for PQ(M', τ). We have

$$\widehat{T}_n(\tau) = \max_{j \in J} \widehat{T}_{\omega'(j)} + p_{\omega'(j)} = \max_{j \in J} \widehat{T}_{\omega(j)} + p_{\omega(j)} + p_{\omega'(j)} \leq \max_{j \in J} T_{\omega(j)} + p_{\omega(j)} + p_{\omega'(j)} =$$

$$\max_{j \in J} \max\left\{t_{\omega(j)} + p_{\omega(j)} \; ; \; t_{\omega(j)} + p_{\omega(j)} + \tau - d_j\right\} \leq \max\{t_n \; ; \; \tau\} = t_n$$

The first inequality holds by the optimality of the schedule $\widehat{T}(\tau)$ and the last equation holds because of the hypothesis $\widehat{T}_n(\tau) > \tau$: the maximum cannot be attained for τ because that would imply $\tau < \tau$. ∎

From Proposition 2 we can deduce

Proposition 3: *If $\widehat{T}_n(\tau) > \tau$ and the schedule t associated with $\widehat{T}(\tau)$ is feasibile for RJSSD, then t is also optimal.*

Proof: Let $\widehat{t}$ be an optimal schedule for RJSSD. We have $t_n := \max_{j\in J}\{\widehat{T}_{\omega(j)} + p_{\omega(j)}\} = \max_{j\in J} \widehat{T}_{\omega'(j)} \leq \widehat{T}_n(\tau) \leq \widehat{t}_n$, where the last inequality follows from Proposition 2. Hence $t_n = \widehat{t}_n$ and t is optimal. ∎

We may also derive an infeasibility condition as

Proposition 4: *$\tau \leq d_{\max}$ and $\widehat{T}_n(\tau) > d_{\max}$ imply infeasibility of Problem RJSSD.*

Proof: Feasible solutions of RJSSD have makespan not larger than $d_{\max}$. But by Proposition 2, feasible solutions have makespan not smaller than $\widehat{T}_n(\tau) > d_{\max}$. Thus there can be no feasible solutions under the conditions of the proposition. ∎

The above results could be used to solve a Problem RJSSD through a sequence of Problems PQ(M', τ) by adopting for instance a binary search strategy. In fact, given a guess τ of the optimal makespan for Problem RJSSD, and the corresponding makespan $\widehat{T}_n(\tau)$, Proposition 1 and 2 provide a restricted range of values for the next guess.

However, such an approach requires the exact solution of each auxiliary problem and this is not practical. If we are able to produce only a heuristic solution, having makespan $\widetilde{T}_n(\tau) \geq \widehat{T}_n(\tau)$, then only Proposition 1 can be applied, namely if $\widetilde{T}_n(\tau) \leq \tau$ then the schedule t associated with $\widetilde{T}$ is feasible for RJSSD.

4. THE SHIFTING BOTTLENECK PROCEDURE

In this section we generalize the Shifting Bottleneck Procedure to the JSSD. The Shifting Bottleneck Procedure is based on the idea of sequencing the machines one at a time. Priority is dynamically assigned to the current most critical machine. A brief review of the Shifting Bottleneck Procedure of Adams, Balas and Zawack (1988) (called here SB1) for the Job Shop Problem (without deadlines) is as follows. Let M' be the set of machines already sequenced, that is the set of machines for which a selection has been computed ($M' = \emptyset$ at the start).

Step 1. Identify a bottleneck machine k among the unscheduled machines $M \setminus M'$ and sequence it optimally. Set $M' \leftarrow M' \cup \{k\}$ and go to step 2.

Step 2. Reoptimize the sequence on the machines in M'. If $M' = M$, stop; otherwise go to 1.

The resequencing phase in Step 2 is executed on one machine at a time. This task and the one of identifying the bottleneck machine were carried out in the original version of the SB1 by solving a one machine problem with the algorithm of Carlier, (1982). Later an improvement has been obtained by Balas, Lenstra and Vazacopoulos (1995) by defining and solving a one machine problem with delayed precedence constraints (DPC's).

Recently a variant of the Shifting Bottleneck Procedure has been proposed by Balas and Vazacopoulos (1994), in which Step 2 is carried out by a new local search procedure based on pairwise interchange, instead of computing a sequence of one machine problems. This local search depends on a particular definition of neighborhood of a solution and relies on the properties of critical paths in the disjunctive graph D_S. This new approach, which we call SB2, has given better computational results than SB1.

We will use a procedure like SB2 in order to solve the Job Shop Problem with Deadlines. Note that in SB2 there are two basic tools: the Balas, Lenstra and Vazacopoulos (1995) algorithm for the one machine problem and a local search for a JSS Problem proposed in Balas and Vazacopoulos (1994). Thus we have to modify both the one machine problem and the local search in order to deal with the deadlines.

As far as the one machine problem is concerned we have designed an exact procedure for solving a one machine problem with deadlines. We describe this method in the next section. As for the local search we have slightly modified the local search defined in Balas and Vazacopoulos, (1994), by relying on the auxiliary job shop problems. This will be described in Section 6.

A general scheme of the resulting procedure, denoted SBD, is as follows:

Let M' be the set of machines already sequenced, that is the set of machines for which a selection has been computed ($M' = \emptyset$ at the start).

Step 1. Identify a bottleneck machine k among the unscheduled machines $M \setminus M'$ and sequence it optimally by solving a one machine problem with deadlines. Set $M' \leftarrow M' \cup \{k\}$ and go to step 2.

Step 2. Improve the current partial selection through the modified local search procedure. If $M' = M$, stop; otherwise go to 1.

The criterion to identify a bottleneck machine in Step 1 is as follows: if all one machine problems are feasible then the bottleneck machine is the one with the largest maskespan; if there exist infeasible one machine problems then the bottleneck machine is the one with the largest tardiness, that is with the largest deadline violation.

5. THE ONE MACHINE SCHEDULING PROBLEM WITH DEADLINES

In this section we describe the model of one machine problem adopted in Step 1 of SBD and the algorithm for its solution. We denote this new problem as MPD. Formally it can be stated as

One Machine Problem with Delayed Precedence constraints and Deadlines (MPD): a set I of operations, and a partial order $\prec$ on I are given. Let $R \subset I \times I$ be the set of unordered pairs $\{i, j\}$ such that neither $i \prec j$ nor $j \prec i$. To each pair of operations such that $i \prec j$ a nonnegative integer l_{ij} (delay) is assigned. To each $i \in I$ four nonnegative integer quantities r_i (heads), p_i (processing times), q_i (tails) and dl_i (deadlines) are assigned. The problem consists in finding a schedule t_i, $i \in I$ such that

$$
\begin{aligned}
t_i &\geq r_i && i \in I \\
t_j - t_i &\geq p_i + l_{ij} && i \prec j \\
t_i + p_i &\leq dl_i && i \in I \\
t_j - t_i \geq p_i \quad \vee \quad t_i - t_j &\geq p_j && \{i, j\} \in R
\end{aligned}
$$

and the makespan

$$\max_{i \in I} \{t_i + p_i + q_i\}$$

is minimized.

The relationship of Problem MPD with the procedure SBD is the following. The input of Step 1 consists of a set M' of machines already sequenced with partial selection S, and a corresponding graph D_S. Let

$L(i,j)$ denote the length of a longest path from i to j in D_S ($L(i,j) = -\infty$ if such a path does not exist). Let $k \notin M'$ be any machine not sequenced yet. Then a problem MPD is defined with the following data:

– $I := N_k$

– $i \prec j$ if there exists a directed path from i to j in D_S;

– $l_{ij} := L(i,j)$ for all $i \prec j$;

– $r_i := L(0,i)$ for all $i \in N_k$;

– $p_i := p_i$ for all $i \in N_k$;

– $q_i := L(i,n) - p_i$ for all $i \in N_k$;

– $dl_i := \min_{j \in J} \left\{ d_j - p_{\omega(j)} - L(i, \omega(j)) \right\} + p_i$.

The one machine problem investigated by Balas, Lenstra and Vazacopoulos (1995) differs from MPD only in the missing deadline constraints. We refer to this problem as MP. We solve MPD through a sequence of MP until either a feasible optimal solution is obtained or it can be asserted that no feasible solution exists. In the latter case we want to find a schedule that minimizes the maximum tardiness.

As in the case of the job shop problem our strategy consists in defining an auxiliary problem based on a guess of the optimal makespan for problem MPD. Results like those of Section 3 can be obtained also for this case. The difference is that now an exact algorithm is available to solve the auxiliary one machine problems so that it is possible to fully exploit those results and derive an exact algorithm also for MPD.

Before describing in detail the algorithm we remark that problem MPD is worth studying in its own, besides its use as a submodule of the job shop problem. A special case of this problem, without the delayed precedences, has been already investigated by Leon and Wu (1992). They consider the case of several forbidden times for the machine, that is time intervals in which the machine is unavailable for processing. Our model can easily take care of a forbidden time (a_i, b_i) by using a dummy job with head a_i, deadline b_i and processing time $b_i - a_i$ which compel the dummy job to be executed during the forbidden time. This model with forbidden times can be useful in many circumstances like breakdown or maintenance periods, priority scheduling and others.

Let us now define a family of auxiliary one machine problems $MP(\tau)$. These problems have the same data as problem MPD with the only difference that deadlines are missing and the tails q_i are reset as:

$$q_i' := \max \{ q_i \ ; \ \tau - dl_i \} \tag{6}$$

Again the relationship between problem MPD and the family of auxiliary problems is characterized by results analogous to those in Section 3. We restate them here for the sake of clarity but we do not provide proofs.

Let $m(\tau)$ be the optimal makespan of the auxiliary problem $MP(\tau)$ and $T(\tau)$ be the corresponding optimal schedule. Note that the makespan of the original problem is never larger than the makespan of the auxiliary problem, for any value of τ.

Proposition 5: *If $m(\tau) \leq \tau$ then the schedule $T(\tau)$ is feasible for MPD, i.e. $T_i(\tau) + p_i \leq dl_i$, for all $i \in I$.*

Proposition 6: *If $m(\tau) > \tau$ then all feasible schedules of MPD have makespan at least equal to $m(\tau)$.*

Proposition 7: *$m(\tau) > \tau$ and feasibility of $T(\tau)$ imply optimality.*

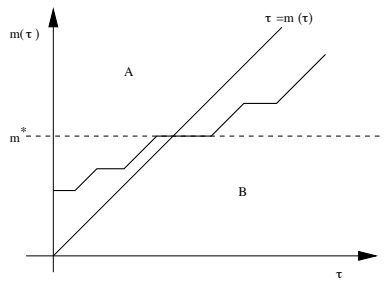

Figure 2 - Graph of the map $\tau \mapsto m(\tau)$

Proposition 8: *$\tau \leq \max_i(q_i + dl_i)$ and $m(\tau) > \max_i(q_i + dl_i)$ imply infeasibility of MPD.*

Proposition 8 provides a feasibility test consisting of solving a problem MP($\max_i(q_i + dl_i)$). If the instance is not feasible, the computed solution minimizes the maximum tardiness, as it is easy to see.

We can view this process of computing a makespan $m(\tau)$ from a guess τ as a map $\tau \mapsto m(\tau)$. This map is piecewise linear and nondecreasing, the optimal makespan corresponds to the smallest fixed point of the map and the slope of each piece is either zero or one, according to the following result.

Proposition 9: *For every $\delta \geq 0$, $m(\tau) \leq m(\tau + \delta) \leq m(\tau) + \delta$. Further, if $\delta = 1$, then either $m(\tau + 1) = m(\tau)$ or $m(\tau + 1) = m(\tau) + 1$.*

Proof: Clearly $m(\tau) \leq m(\tau + \delta)$ as the tails increase from one problem to the other. Further, since they increase by at most δ, the solution $T(\tau)$ has a makespan not greater than $m(\tau) + \delta$ in $MP(\tau + \delta)$, thus showing $m(\tau + \delta) \leq m(\tau) + \delta$. The second statement then follows from the integrality of the data. ∎

The graph of this map is given in Figure 2 for a feasible instance. There we denote by m^* the sought optimal makespan of MPD. According to Proposition 6, the map cannot intersect the region A delimited by $m(\tau) > \tau$ and $m(\tau) > m^*$. Analogously, because of Proposition 5, and since feasible schedules have makespan at least m^*, the region B delimited by $m(\tau) \leq \tau$ and $m(\tau) < m^*$ is forbidden.

These propositions suggest designing two alternative strategies; one consists of monotonically increasing guesses and the second one of guesses computed in a binary search fashion.

In the first strategy we test feasibility and if the instance is feasible we iterate according to

$$\tau^0 := 0 \qquad \text{repeat } \tau^{i+1} := m(\tau^i) \qquad \text{until } \tau^{i+1} = \tau^i \tag{7}$$

Proposition 10: *: If the instance of MPD is feasible, the iteration (7) converges to an optimal solution.*

Proof: : Let τ^i and τ^{i+1} be two consecutive guesses, i.e. $\tau^{i+1} = m(\tau^i)$. First, we need to show that the method eventually halts. Second, that it terminates at an optimal solution. As for finiteness, note that if the instance is feasible, by Proposition 6 the condition $m(\tau^i) > \tau^i$ cannot be repeated infinitely many times, or otherwise the makespan of MPD would be unbounded. Therefore at some iteration we must have $m(\tau^{i+1}) = \tau^{i+1}$ and $m(\tau^i) > \tau^i$. By Proposition 5 the schedule $T(\tau^{i+1})$ is feasible. Further, by Proposition 6, $m(\tau^i)$ is a lower bound for the optimal makestpan. Therefore, since $m(\tau^{i+1}) = \tau^{i+1} = m(\tau^i)$, then $T(\tau^{i+1})$ is an optimal schedule. ∎

A crucial point of the above approach is the number of auxiliary problems to be solved. First we note that the difference between two consecutive guesses is nonincreasing.

Proposition 11: $\tau^{i+1} - \tau^i \leq \tau^i - \tau^{i-1}$.

Proof: Let $\delta = \tau^i - \tau^{i-1}$; by Proposition 9, $m(\tau^{i-1}+\delta) \leq m(\tau^{i-1})+\delta$, i.e. $m(\tau^i) \leq \tau^i + \tau^i - \tau^{i-1}$. Therefore $\tau^{i+1} - \tau^i \leq \tau^i - \tau^{i-1}$. ∎

According to Proposition 11, the convergence speed of the method decreases, i.e. the largest steps toward the optimal guess are the first ones. Moreover, once the guess increases by one at a certain iteration, it will always increase by one until the end. Therefore, in the worst case, the monotonic search requires a pseudopolynomial number of auxiliary problems $M(\tau)$ to be solved.

There are examples where the tails are increased by one at each iteration. For instance consider two operations with $r_1 := 0$, $p_1 := a$, $q_1 := b$, $dl_1 = \infty$, $r_2 := a - 1$, $p_2 := b$, $q_2 := 0$, $dl_2 := a + b - 1$. This example requires $a + b + 1$ iterations and the guess is increased by one at each iteration.

This drawback can be avoided by adopting a binary search over the possible values of the makespan. At a generic step of the binary search there is a lower bound and an upper bound for the optimal makespan, and a solution is available with makespan equal to the upper bound. Let us denote by τ_L the lower bound and by τ_U the upper bound. From these two values a guess τ is computed according to:

$$\tau := \left\lfloor \frac{\tau_L + \tau_U}{2} \right\rfloor \tag{8}$$

and the problem MP(τ) is solved yielding a value $m(\tau)$.

If $m(\tau) > \tau$ then, according to Proposition 6, the optimal makespan is not smaller than $m(\tau)$ and therefore the lower bound is reset to $\tau_L := m(\tau)$ (unless the solution is feasible in which case we exit the binary search since the solution is also optimal according to Proposition 7). If $m(\tau) \leq \tau$ then, according to Proposition 5, the schedule is feasible for MPD, with makespan $v = \max_{i \in I} \{T_i + p_i + q_i\} \leq \max_{i \in I} \{T_i + p_i + q'_i\} = m(\tau)$; then the upper bound is reset to $\tau_U := v$.

Before starting the binary search we test feasibility. If the solution is feasible let v be the corresponding makespan. Then the binary search is initialized with $\tau_L := 0$ and $\tau_U := v$. It terminates when $\tau_L = \tau_U$ this being also the optimal makespan of the MPD.

The binary search is superior to the monotonic search in the worst case. However, while the binary search requires almost invariably the same number of steps, most of the time the monotonic search finds the optimal value in a few steps (say two or three). In view of these empiric results we have adopted a mixed strategy by first testing feasibility, then starting a monotonic search for at most three steps and, if no solution has been found, switching to the binary search. In Section 7 we report some computational experiments.

6. THE LOCAL SEARCH PROCEDURE

The output of Step 1 in SBD consists of a partial selection S over a subset M' of machines and a corresponding makespan $t_n(S)$. These data are given as input to the first iteration of the local search. The input of each iteration of the local search consists again of a partial selection S over the same subset M' and

a corresponding makespan $t_n(S)$, which are received from the previous iteration. Furthermore each iteration uses a guess τ in order to define the auxiliary problem $\mathrm{PQ}(M', \tau)$. The neighbourhood of the selection S in $\mathrm{PQ}(M', \tau)$ is explored to generate a new selection by using the same definitions and techniques as in Balas and Vazacopoulos, (1994). Also the stopping rule for the local search is the same.

The guess is computed in the following way: first set $\tau := t_n(S)$, then if at the end of an iteration a selection S' is found such that $T_n(\tau) \leq \tau$ reset $\tau := t_n(S') - 1$. Note that if $m(\tau) \leq \tau$ then the current selection is feasible according to Proposition 1 (whose conclusion holds also with an approximate value for $m(\tau)$ as already remarked).

A twofold goal is pursued in the local search, namely minimizing the makespan and obtaining feasibility. In the auxiliary problem the guess sets a trade–off between the two goals. For a large value of the guess the objective of the auxiliary problem becomes simply feasibility. For this purpose it is enough to have a value $\tau = d_{\max}$ as apparent from Proposition 4. For a small value of the guess the objective of the auxiliary problem consists in the mere minimization of the makespan ignoring the presence of the deadlines. For this purpose it is enough to set $\tau = \min_j d_j$ as obvious from the definitions. Values of τ between these two extremes realize a compromise between the two goals. Hence we keep the guess corresponding to the best makespan found so far, so that in the following iterations feasible solutions will be found within this target value of makespan.

If the partial selection received from Step 1 is infeasible, it might seem reasonable to start the local search with a guess $\tau = d_{\max}$. However, computational experiments have shown that it is more effective to start with $\tau = t_n(S)$ even if S is infeasible. Note also that the local search is not necessarily improving at each step (if so at every step the selection would be feasible by Proposition 1) so that it might happen that infeasible selections are produced during the local search. In these cases the guess is not changed.

In conclusion, the guess is either equal to the best feasible makespan found so far or to the initial makespan if no feasible solution has been found (including the initial solution), and is clearly monotonically decreasing during the local search.

7. COMPUTATIONAL RESULTS FOR THE ONE MACHINE PROBLEM

We have carried out two sets of computational experiments. The first one verifies that the exact procedure for solving the One Machine Scheduling Problem with Delayed Precedence constraints and Deadlines (MPD) takes, on the average, a reasonable computating time so it can be used within the general SB procedure. Furthermore, we have compared our results with those of Leon and Wu who address a special case of the problem we define.

As far as the MPDP is concerned, the branch and bound algorithm was implemented in C on a SUNSparc-330 workstation, then it was applied to two types of experiments. First, a set of data was kindly provided by Professor Leon, (1992), which consisted of 5 sets of one machine problems with 50 jobs and forbidden times. Notice that this is a special instance of our model since a forbidden time from t_1 to t_2 is a (dummy) job with a single operation, with release time t_1, processing time $t_2 - t_1$, and deadline time t_2. Leon and Wu generated this set as follows: the processing times, p_i, are generated based on a normal distribution, $N(50, 10^2)$, and the release dates of jobs are generated exponentially with parameter

$\lambda = 40$ (a random variable X is exponentially distributed with parameter λ if Prob$[X \leq x] = 1 - e^{-x/\lambda}$; its expected value is λ). The tails are generated as $q_i = \sum_{j=1}^{s} p_j$, with s uniformly distributed between 1 and 50 (for more details see at Leon and Wu, 1992).

In Table 1 we compare our computational results with those from Leon and Wu, (1992). In our algorithm we solve a series of one machine scheduling problems, therefore the number of explored nodes is equal to the total number of nodes for all the subproblems (a node is considered explored when its lower bound is calculated). We have succeeded in solving every problem by using less than 100 nodes. Comparing the results with those from Leon and Wu, (1992), we can see that our method needs considerably fewer explored nodes.

We generated a second set of problems which is similar to Leon and Wu (1992). These are 1,400, 50-job problems. The r_i, p_i, q_i are randomly generated with uniform distribution between 1 and $r_{\max}, p_{\max}, q_{\max}$ (with $p_{\max}$ held fixed to 100). The forbidden times are assigned by generating exponentially with parameter λ_1 their durations and by generating exponentially with parameter λ_2 the times between two consecutive forbidden times. Forbidden times are generated only within the interval $[0, \sum_{j=1}^{50} p_j]$. We report these computational results in Table 2. Note here that we have solved all the problems to optimality. Leon and Wu, (1992), in their computational experience have not solved 89 problems among 1,400 using less than 1,500 nodes. Our algorithm has successfully solved all the problems and the maximum number of nodes used is 216.

We generated a third set of problems as follows: for every problem of the second set delayed precedence constraints are added. A precedence constraint is generated between jobs i and j with probability $p_{ij} \in \{0.02, 0.05, 0.10\}$ and the delay $L(i,j)$ is generated as follows. For each (i,j) in F a number $l(i,j)$ is drawn from a uniform distribution over the interval $[1, (r_{\max} + q_{\max})/2]$; and $L(i,j)$ is set to $l(i,j)$ if $l(i,j) > p_i$, and to $l(i,j) + p_i$ otherwise. The results are shown in Tables 3-5.

8. COMPUTATIONAL RESULTS FOR THE JSSD

We have tested the overall procedure both on a real set of data and on some generated sets of data. The real set of data has been provided by a factory near Pittsburgh which produces card board boxes. The production environment is typical of job-shop models. There are 13 machines devoted to various operations like cutting, slitting, printing, flexing, flattening, stitching, and others. An order consists in a request of producing a high number of boxes. Each order has a predefinite sequence of operations and consequently of machines. Hence an order has the same structure of a job. The number of operations for each job can vary between one and five and most jobs have two or three operations. The scheduling manager decides the batch size independently of the schedule. Thus the processing time of each operation is fixed and given.

Every day new orders become available with a definite deadline already contracted with the customer. Hence a job shop problem should be solved every day in order to schedule the jobs arrived on that day plus the jobs still to be processed or to be completed from the previous days. We have approached this problem by using deadlines and minimizing the makespan. The operations carried over from one day to the other receive a new deadline if they are already tardy. However, if an operation starts in one day and has to be finished in the next day, then the remaining part of the operation is viewed as a new operation which has

to be restarted immediately and therefore is assigned a deadline equal to the remaining processing time. Therefore by using deadlines we both take care of real deadlines and control the production flow. Moreover by minimizing the makespan we try to improve the lead time of the work in process.

We have used the data of one full month, January 1996. This corresponds to 22 working days and each working day consists of ten working hours. The time unit is the minute. In the first column of Table 6 we have indicated the day of the month, in the second column we have reported the number of new operations for that day, in the third column the number of operations whose schedule has to be computed on that day (not to be processed on that day), in the fourth column the number of operations which have been processed beyond the deadline and in the fifth column the maximum tardiness value (in minutes) of tardy jobs. The schedule has been computed cyclically (but in one run) for all 22 days and has required less than one minute CPU time. So the procedure has exhibited a good behaviour in term of computing time. As for the schedule quality this was better than the one actually implemented. The presence of tardy jobs seems to be unavoidable due to the sudden accumulation of many orders at the same time.

We have then made a similar computation with a generated set of data. We have taken the famous $6 \times 6 \times 6$ instance (6 jobs with 6 operations each on 6 machines) defined in Fisher and Thompson (1963) and have simulated a production environment by supposing that every day a new $6 \times 6 \times 6$ set of jobs is sent to the shop (every day the same). It is not difficult to show that the work backlog is not accumulating indefinitely if the day working time is at least equal to the sum of the processing times of the critical machine, that is the one most heavily loaded (43 time units in this case). Then, on the average, every day 6 jobs are processed, although the lead time for each of them may be longer than one day. Minimizing the makespan is equivalent to minimizing the lead time of the operations. On every run the deadlines of the jobs have been fixed in the following way: the new jobs have no deadline, and the old jobs still to be completed are assigned a deadline which corresponds to the previously computed completion time.

Our goal has been to show that, by defining a fictitious day of length 43 time units, our procedure is able to compute every day the schedule (of old and new jobs) reaching a steady state in which there is no accumulation of old jobs. We point out that unless the critical machine works without interruption this steady state behaviour cannot be obtained. Therefore the procedure must find out a schedule which does not allow the critical machine to be idle. Because of this fact we consider this as a robust test to judge the quality of the solution.

The solution is shown in Figures 3-a,b,c,d for the first four runs. As can be seen, the solution becomes periodic with a constant makespan of 62 after the fourth run. We recall that the optimal makespan of the $6 \times 6 \times 6$ instance is 55 (see Figure 3-a). This is clearly a transient value obtained in the first computation. In the next runs there are a few more jobs from the previous days and the makespan value is higher than 55.

We have also tested our procedure against some instances described in the literature. We have considered 160 instances (DMU1–DMU160, indicated as $J//L_{max}$ by the authors) generated by Demirkol, Mehta, Uzsoy (1996). In these instances the authors aim at minimizing the lateness with respect to given due dates. We consider these due dates as deadlines.

The instances have four values for the number of jobs ($n = 20, 30, 40, 50$) and two for the number of machines ($m = 15, 20$). Hence the total number of operations varies from 300 to 1000.

In Tables 7, 8, 9 and 10 we compare our results with those from Demirkol, Mehta, Uzsoy (1996).

We report in column DMU the best tardiness found by Demirkol, Mehta, Uzsoy (1996) after running eleven dispatching rules and three different versions of the Shifting Bottleneck Procedure. We report in the adjacent CPU column the computation time in seconds. This is the computation time taken by the method yielding the best solution. In column SBD1 we report the best tardiness obtained by our procedure and in the adjacent CPU column its computing time in seconds. We observe that for all the problems we have obtained better solutions by using less computing time in almost all cases. In addition we report for every instance a lower bound produced by using the one machine relaxation. This is obtained in the first iteration of the Shifting Bottleneck Procedure. We observe that all the instances are infeasible.

In Tables 11 and 12 we report our results for other 80 instances generated from DMU1–DMU80 as follows: for every instance we changed the due date (deadline in our case) of every job by adding to the old due date the quantity $3(UB + LB)/4$, where UB is the upper bound obtained by Demirkol, Mehta, Uzsoy (1996), and LB is the corresponding lower bound. We call these instances BLSVk, where $k = 1, \ldots, 80$. For all these instances we obtained feasible schedules with respect to the deadlines. We report in column SBD1 the best makespan found by our procedure, in column CPU he corresponding CPU time in seconds and in column LB the lower bound obtained by using the one machine relaxation.

Demirkol, Mehta, Uzsoy (1996) used a SUN SPARCserver 1000 Model 1104 with four 50Mhz CPUs and 256MB of RAM. Our algorithm was run on an alpha workstation.

9. REFERENCES

Adams, J., E. Balas and D. Zawack,1988, "The Shifting Bottleneck Procedure for Job Shop Scheduling", *Management Science*, **34**, 391-401.

Balas, E., J.K. Lenstra and A. Vazacopoulos, 1995, "The One Machine Problem with Delayed Precedence Constraints and its Use in Job Shop Scheduling", *Management Science*, **41**, 94-109.

Balas, E. and A. Vazacopoulos, 1994, "Guided Local Search and Shifting Bottleneck Procedure for Job Shop Scheduling", Management Science Research Report #MSSR-609, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Carlier, J., 1982, "The One-Machine Sequencing Problem", *European Journal of Operational Research*, **11**, 42-47.

Demirkol, E., S. Mehta, R. Uzsoy, 1996, "Benchmarking for Shop Scheduling Problems", Research Memorandum No. 96-4, Purdue University.

Fisher, H., and G.L. Thompson, 1963, "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules", in *Industrial Scheduling*, J.F. Muth and G.L. Thompson (editors), Prentice-Hall, Englewood Cliffs, NJ.

Leon, V.J., 1992, , personal communication.

Leon, V.J. and S.D. Wu, 1992, "On Scheduling with Ready-Times, Due-Dates and Vacations", *Naval Research Logistics*, **39**, 53-65.

Table 1: Results for the One Machine Scheduling with Forbidden Times

Set	A	B	C	D	E
1	205	555	45.16	86	0.22
2	132	285	33.68	78	0.20
3	95	268	53.47	84	0.23
4	232	467	25.68	44	0.08
5	229	566	19.05	32	0.07

A = Average number of nodes (LW)
B = Maximum number of nodes (LW)
C = Average number of nodes (BLSV)
D = Maximum number of nodes (BLSV)
E = Average CPU seconds (BLSV)

Table 2, $n = 50$.

$r_{\max}$	$q_{\max}$	λ_1	λ_2	A	B	C	D	E
5000	5000	1000	400	1.80	3.4000	8	0.0075	0.0170
5000	5000	1000	100	2.08	3.0800	6	0.0071	0.0170
5000	5000	500	400	2.84	3.0400	10	0.0068	0.0330
5000	5000	500	100	3.62	3.0800	10	0.0064	0.0330
4000	4000	1000	400	1.86	4.0400	12	0.0088	0.0170
4000	4000	1000	100	2.70	4.5600	12	0.0102	0.0330
4000	4000	500	400	2.70	4.0400	8	0.0085	0.0330
4000	4000	500	100	3.92	3.6000	10	0.0081	0.0330
3000	3000	1000	400	2.10	6.2000	22	0.0155	0.0500
3000	3000	1000	100	2.24	5.3200	20	0.0121	0.0670
3000	3000	500	400	3.08	7.0000	38	0.0168	0.0830
3000	3000	500	100	4.32	5.8400	16	0.0138	0.0500
2500	2500	1000	400	1.88	9.7600	22	0.0257	0.0670
2500	2500	1000	100	2.36	10.8800	90	0.0294	0.2670
2500	2500	500	400	3.00	9.8400	18	0.0251	0.0500
2500	2500	500	100	4.40	10.2800	32	0.0261	0.1000
4000	5000	1000	400	1.96	4.0000	14	0.0095	0.0330
4000	5000	1000	100	2.28	3.8000	8	0.0082	0.0170
4000	5000	500	400	3.20	4.2000	12	0.0098	0.0330
4000	5000	500	100	4.40	4.0000	12	0.0085	0.0330
3000	5000	1000	400	2.16	5.4000	18	0.0125	0.0330
3000	5000	1000	100	2.22	10.0800	216	0.0258	0.5670
3000	5000	500	400	3.00	5.6800	16	0.0141	0.0330
3000	5000	500	100	4.18	4.9600	12	0.0122	0.0330
2500	5000	1000	400	1.98	10.0400	20	0.0237	0.0500
2500	5000	1000	100	2.60	9.4800	26	0.0241	0.0670
2500	5000	500	400	2.86	9.9600	20	0.0251	0.0670
2500	5000	500	100	3.86	9.2400	24	0.0231	0.0670

A = Average forbidden times
B = Average nodes
C = Maximum nodes
D = Average CPU seconds
E = Maximum CPU seconds

Table 3. $n = 50$, Density = 2%

$r_{\max}$	$q_{\max}$	λ_1	λ_2	A	B	C	D	E
5000	5000	1000	400	1.80	3.56	10	0.0109	0.0330
5000	5000	1000	100	2.08	3.90	10	0.0112	0.0330
5000	5000	500	400	2.84	4.12	10	0.0139	0.0330
5000	5000	500	100	3.62	3.58	10	0.0128	0.0330
4000	4000	1000	400	1.86	4.48	30	0.0125	0.1170
4000	4000	1000	100	2.70	5.08	16	0.0148	0.0330
4000	4000	500	400	2.70	3.86	16	0.0111	0.0330
4000	4000	500	100	3.92	3.98	11	0.0112	0.0330
3000	3000	1000	400	2.10	7.10	28	0.0200	0.0830
3000	3000	1000	100	2.24	5.20	21	0.0148	0.0670
3000	3000	500	400	3.08	6.64	42	0.0178	0.1500
3000	3000	500	100	4.32	6.40	22	0.0181	0.0500
2500	2500	1000	400	1.88	12.78	268	0.0471	1.4000
2500	2500	1000	100	2.36	11.22	50	0.0294	0.1670
2500	2500	500	400	3.00	8.54	33	0.0244	0.1170
2500	2500	500	100	4.40	9.06	36	0.0277	0.1000
4000	5000	1000	400	1.96	5.46	16	0.0141	0.0500
4000	5000	1000	100	2.28	5.88	49	0.0168	0.1500
4000	5000	500	400	3.20	4.02	12	0.0111	0.0330
4000	5000	500	100	4.40	4.36	10	0.0115	0.0330
3000	5000	1000	400	2.16	4.94	14	0.0138	0.0330
3000	5000	1000	100	2.22	6.18	20	0.0182	0.0500
3000	5000	500	400	3.00	5.40	20	0.0165	0.0500
3000	5000	500	100	4.18	5.60	19	0.0165	0.0500
2500	5000	1000	400	1.98	6.96	44	0.0201	0.1330
2500	5000	1000	100	2.60	7.22	24	0.0204	0.0670
2500	5000	500	400	2.86	6.18	18	0.0184	0.0500
2500	5000	500	100	3.86	7.24	54	0.0208	0.1670

A = Average forbidden times
B = Average nodes
C = Maximum nodes
D = Average CPU seconds
E = Maximum CPU seconds

Table 4. $n = 50$, Density = 5%

$r_{\max}$	$q_{\max}$	λ_1	λ_2	A	B	C	D	E
5000	5000	1000	400	1.80	3.44	10	0.0128	0.0330
5000	5000	1000	100	2.08	3.50	12	0.0119	0.0330
5000	5000	500	400	2.84	3.60	8	0.0131	0.0330
5000	5000	500	100	3.62	3.48	10	0.0128	0.0330
4000	4000	1000	400	1.86	4.00	16	0.0138	0.0830
4000	4000	1000	100	2.70	4.08	10	0.0162	0.0500
4000	4000	500	400	2.70	4.02	12	0.0142	0.0330
4000	4000	500	100	3.92	3.78	10	0.0155	0.0330
3000	3000	1000	400	2.10	5.40	14	0.0201	0.0670
3000	3000	1000	100	2.24	4.60	16	0.0165	0.0670
3000	3000	500	400	3.08	5.30	12	0.0184	0.0500
3000	3000	500	100	4.32	4.52	14	0.0165	0.0500
2500	2500	1000	400	1.88	6.32	24	0.0211	0.0670
2500	2500	1000	100	2.36	7.26	28	0.0234	0.0830
2500	2500	500	400	3.00	6.24	14	0.0261	0.0670
2500	2500	500	100	4.40	5.94	31	0.0221	0.1500
4000	5000	1000	400	1.96	4.48	15	0.0191	0.0500
4000	5000	1000	100	2.28	4.66	36	0.0162	0.1170
4000	5000	500	400	3.20	3.50	14	0.0118	0.0330
4000	5000	500	100	4.40	3.74	11	0.0145	0.0500
3000	5000	1000	400	2.16	4.02	14	0.0138	0.0330
3000	5000	1000	100	2.22	3.90	10	0.0162	0.0330
3000	5000	500	400	3.00	4.20	12	0.0141	0.0330
3000	5000	500	100	4.18	4.02	10	0.0152	0.0330
2500	5000	1000	400	1.98	3.90	10	0.0141	0.0330
2500	5000	1000	100	2.60	5.26	16	0.0191	0.0670
2500	5000	500	400	2.86	5.34	11	0.0204	0.0500
2500	5000	500	100	3.86	5.30	32	0.0185	0.0830

A = Average forbidden times
B = Average nodes
C = Maximum nodes
D = Average CPU seconds
E = Maximum CPU seconds

Table 5. $n = 50$, Density = 10%

$r_{\max}$	$q_{\max}$	λ_1	λ_2	A	B	C	D	E
5000	5000	1000	400	1.80	3.28	8	0.0174	0.0500
5000	5000	1000	100	2.08	3.34	12	0.0192	0.0500
5000	5000	500	400	2.84	2.90	10	0.0152	0.0330
5000	5000	500	100	3.62	3.30	10	0.0204	0.0500
4000	4000	1000	400	1.86	3.78	8	0.0244	0.0500
4000	4000	1000	100	2.70	4.28	13	0.0240	0.0670
4000	4000	500	400	2.70	3.70	10	0.0224	0.0500
4000	4000	500	100	3.92	3.12	10	0.0198	0.0330
3000	3000	1000	400	2.10	4.14	8	0.0243	0.0500
3000	3000	1000	100	2.24	4.34	10	0.0260	0.0500
3000	3000	500	400	3.08	4.32	16	0.0234	0.0830
3000	3000	500	100	4.32	3.92	14	0.0247	0.0670
2500	2500	1000	400	1.88	4.62	18	0.0274	0.1000
2500	2500	1000	100	2.36	4.32	11	0.0267	0.0500
2500	2500	500	400	3.00	5.30	16	0.0313	0.0670
2500	2500	500	100	4.40	5.44	36	0.0320	0.1830
4000	5000	1000	400	1.96	3.26	7	0.0194	0.0330
4000	5000	1000	100	2.28	3.40	14	0.0207	0.0500
4000	5000	500	400	3.20	3.48	20	0.0184	0.0830
4000	5000	500	100	4.40	3.30	10	0.0188	0.0330
3000	5000	1000	400	2.16	3.90	18	0.0227	0.0670
3000	5000	1000	100	2.22	3.64	12	0.0204	0.0670
3000	5000	500	400	3.00	3.36	8	0.0201	0.0500
3000	5000	500	100	4.18	3.32	8	0.0185	0.0670
2500	5000	1000	400	1.98	3.56	11	0.0218	0.0670
2500	5000	1000	100	2.60	4.76	32	0.0277	0.1330
2500	5000	500	400	2.86	3.82	12	0.0241	0.0500
2500	5000	500	100	3.86	3.60	8	0.0231	0.0500

A = Average forbidden times
B = Average nodes
C = Maximum nodes
D = Average CPU seconds
E = Maximum CPU seconds

Table 6

day	DO	SO	TO	T
Tu 2	27	27		
We 3	35	35		
Th 4	44	48		
Fr 5	97	121		
Mo 8	40	133	1	60
Tu 9	13	121		
We 10	19	119	5	286
Th 11	41	133		
Fr 12	38	38		
Mo 15	43	49		
Tu 16	89	112		
We 17	34	90		
Th 18	23	83	2	466
Fr 19	28	72	6	421
Mo 22	127	172		
Tu 23	33	60	4	312
We 24	17	34		
Th 25	24	33		
Fr 26	42	51		
Mo 29	38	56		
Tu 30	28	43		
We 31	31	34		

DO = Number of Day Operations (operations arrived on that day)

SO = Number of Scheduled Operations (operations whose schedule has been computed on that day)

TO = Number of Tardy Operations on that day

T = Maximum tardiness on that day

Table 7

20 Jobs 15 Machines						**30 Jobs 15 Machines**					
Problem	DMU	CPU	SBD1	CPU	LB	Problem	DMU	CPU	SBD1	CPU	LB
DMU1	1448	157	1271	145	1027	DMU41	1379	401	1295	133	1185
DMU2	1552	145	1459	135	1127	DMU42	1459	315	1308	167	1263
DMU3	1492	155	1396	127	1160	DMU43	1441	394	1270	108	1255
DMU4	1464	143	1263	175	1140	DMU44	1360	255	1209	134	1205
DMU5	1501	157	1304	98	1182	DMU45	1483	394	1386	142	1320
DMU6	2090	132	1817	85	1769	DMU46	2810	631	2455	108	2240
DMU7	2092	123	1873	39	1775	DMU47	3029	648	2526	85	2436
DMU8	2246	155	2020	37	1956	DMU48	2311	611	1982	59	1935
DMU9	2181	152	1949	37	1925	DMU49	2940	605	2601	115	2475
DMU10	1785	137	1636	15	1599	DMU50	2531	493	2320	77	2169
DMU11	1957	162	1908	118	1575	DMU51	2750	521	2493	191	2252
DMU12	2100	130	1976	131	1727	DMU52	2347	537	2157	192	2042
DMU13	2165	158	2000	11	1785	DMU53	2470	484	2339	124	2189
DMU14	1839	144	1726	121	1521	DMU54	2496	472	2366	194	2224
DMU15	2143	118	1968	110	1858	DMU55	2666	447	2549	184	2401
DMU16	1682	149	1541	119	1282	DMU56	2433	571	2149	183	1734
DMU17	2174	160	1877	123	1688	DMU57	2678	661	2377	133	2068
DMU18	2381	146	2118	119	1894	DMU58	2515	672	2212	141	1960
DMU19	1943	180	1778	93	1596	DMU59	2380	516	2195	133	1922
DMU20	2018	143	1762	77	1663	DMU60	2510	456	2196	149	2075

Table 8

20 Jobs 20 Machines						**30 Jobs 20 Machines**					
Problem	DMU	CPU	SBD1	CPU	LB	Problem	DMU	CPU	SBD1	CPU	LB
DMU21	2013	399	1911	174	1391	DMU61	1816	1238	1587	335	1268
DMU22	1708	345	1583	145	1182	DMU62	1952	1166	1656	269	1412
DMU23	1962	369	1756	147	1366	DMU63	2173	1210	1846	303	1575
DMU24	2248	340	2023	205	1569	DMU64	2237	1073	1880	202	1710
DMU25	1753	367	1576	158	1226	DMU65	2094	1239	1855	317	1611
DMU26	2631	281	2249	91	2147	DMU66	3260	1162	2527	223	2314
DMU27	2842	338	2533	126	2376	DMU67	3577	1133	3023	226	2713
DMU28	2465	336	2199	81	2106	DMU68	3601	1096	2897	142	2817
DMU29	2835	313	2469	10	2469	DMU69	3021	1130	2641	246	2386
DMU30	2712	274	2378	9	2378	DMU70	2896	1	2315	42	2292
DMU31	2638	392	2431	192	1776	DMU71	2953	1264	2669	318	2178
DMU32	2647	415	2247	179	1868	DMU72	3032	1048	2780	361	2298
DMU33	2535	356	2392	189	1845	DMU73	3116	1106	2766	303	2461
DMU34	2627	395	2429	172	1927	DMU74	3074	1025	2873	272	2496
DMU35	2640	369	2391	165	1947	DMU75	3104	902	2833	285	2562
DMU36	2617	345	2344	146	1982	DMU76	2959	1370	2349	340	2061
DMU37	3118	401	2621	128	2419	DMU77	3106	1181	2560	211	2254
DMU38	3029	388	2567	109	2401	DMU78	3409	1139	2781	228	2485
DMU39	2851	383	2651	111	2294	DMU79	3330	1046	2713	224	2570
DMU40	2966	390	2720	112	2518	DMU80	3088	1179	2514	198	2412

Table 9

40 Jobs 15 Machines						**50 Jobs 15 Machines**					
Problem	DMU	CPU	SBD1	CPU	LB	Problem	DMU	CPU	SBD1	CPU	LB
DMU81	1431	1199	1308	212	1191	DMU121	1419	2822	1072	381	1050
DMU82	1787	1155	1630	311	1533	DMU122	1545	1193	1418	330	1418
DMU83	1468	806	1308	185	1299	DMU123	2042	1331	1983	201	1957
DMU84	1648	1011	1601	162	1542	DMU124	1764	1528	1707	267	1707
DMU85	1527	456	1527	76	1460	DMU125	1804	1605	1757	150	1757
DMU86	2397	1509	1706	128	1563	DMU126	3024	4943	2364	251	2217
DMU87	2459	2030	1923	187	1669	DMU127	3102	4619	2346	133	2284
DMU88	2053	1792	1589	114	1545	DMU128	2834	5714	2301	122	2192
DMU89	2191	2168	1746	81	1695	DMU129	2692	4212	2154	207	2085
DMU90	2420	1720	1940	37	1936	DMU130	2661	4095	2195	107	2137
DMU91	3093	1180	2911	204	2893	DMU131	3492	2238	3216	273	3216
DMU92	2894	928	2855	128	2815	DMU132	3525	1723	3397	228	3391
DMU93	3120	642	3048	43	3048	DMU133	3466	1526	3396	332	3396
DMU94	2875	831	2854	78	2818	DMU134	3220	1188	3181	389	3181
DMU95	2924	841	2878	146	2878	DMU135	3316	1556	3281	237	3277
DMU96	3042	1574	2575	269	2125	DMU136	3338	3816	2724	375	2323
DMU97	2617	1855	2125	269	1836	DMU137	3415	1	2812	329	2464
DMU98	2539	2308	2234	231	1896	DMU138	3186	1	2615	272	2381
DMU99	2767	1577	2350	242	2119	DMU139	3130	4027	2619	342	2345
DMU100	2641	1709	2275	208	2038	DMU140	3307	5608	2775	434	2486

Table 10

40 Jobs 20 Machines						**50 Jobs 20 Machines**					
Problem	DMU	CPU	SBD1	CPU	LB	Problem	DMU	CPU	SBD1	CPU	LB
DMU101	2058	2800	1679	461	1395	DMU141	2181	5657	1757	551	1591
DMU102	2337	2940	1810	481	1597	DMU142	2390	6402	1948	743	1746
DMU103	2143	2526	1888	326	1640	DMU143	2355	4870	2007	551	1794
DMU104	1834	2602	1643	394	1411	DMU144	2219	4888	1912	464	1845
DMU105	2212	2460	1941	499	1835	DMU145	2142	4612	1881	608	1786
DMU106	3444	1	2630	180	2610	DMU146	3455	1	2595	361	2363
DMU107	3862	2280	3146	204	2964	DMU147	3385	1	2445	176	2440
DMU108	3565	3258	2819	192	2798	DMU148	3898	9540	2914	305	2824
DMU109	3895	2672	3071	38	3059	DMU149	3852	1	3044	327	2918
DMU110	3064	2336	2500	98	2441	DMU150	4091	6795	3229	287	3205
DMU111	3430	2784	3051	405	2827	DMU151	3788	7068	3277	467	3189
DMU112	3691	2823	3360	437	3113	DMU152	3875	7102	3504	552	3419
DMU113	3366	2449	3043	496	2843	DMU153	3789	4632	3522	435	3407
DMU114	3572	2384	3273	468	3025	DMU154	3971	4140	3708	246	3642
DMU115	3535	1978	3334	380	3129	DMU155	3758	4570	3562	547	3527
DMU116	3985	4245	3310	431	2687	DMU156	4042	5504	3142	602	2628
DMU117	3154	1	2443	362	2234	DMU157	4184	1	3347	550	2774
DMU118	3469	1	2788	387	2479	DMU158	3712	7791	3020	473	2500
DMU119	3560	5717	3071	438	2643	DMU159	3649	7573	3099	636	2472
DMU120	3540	3044	3038	431	2673	DMU160	3762	5752	3072	614	2551

Table 11

20 Jobs 15 Machines				**30 Jobs 15 Machines**			
Problem	SBD1	CPU	LB	Problem	SBD1	CPU	LB
BLSV1	2724	320	2427	BLSV41	3545	311	3473
BLSV2	2874	412	2532	BLSV42	3523	401	3457
BLSV3	2736	255	2520	BLSV43	3417	318	3408
BLSV4	2856	246	2646	BLSV44	3465	92	3465
BLSV5	2774	295	2689	BLSV45	3469	315	3427
BLSV6	2717	36	2717	BLSV46	3721	32	3721
BLSV7	3077	16	3077	BLSV47	3811	300	3691
BLSV8	2791	323	2750	BLSV48	3603	292	3587
BLSV9	2792	214	2564	BLSV49	3536	608	3344
BLSV10	2625	264	2407	BLSV50	3730	584	3635
BLSV11	2745	245	2467	BLSV51	3689	441	3560
BLSV12	2831	254	2583	BLSV52	3458	43	3458
BLSV13	2802	177	2617	BLSV53	3549	401	3463
BLSV14	2595	225	2473	BLSV54	3594	219	3550
BLSV15	2851	327	2749	BLSV55	3867	351	3824
BLSV16	2831	253	2576	BLSV56	3975	500	3671
BLSV17	2707	309	2438	BLSV57	3454	178	3454
BLSV18	3021	75	2984	BLSV58	3756	281	3590
BLSV19	2693	307	2446	BLSV59	3852	324	3829
BLSV20	2851	342	2547	BLSV60	3663	182	3628

Table 12

20 Jobs 20 Machines				**30 Jobs 20 Machines**			
BLSV21	3327	362	2758	BLSV61	3773	944	3580
BLSV22	3213	276	2704	BLSV62	3719	874	3498
BLSV23	3168	268	2694	BLSV63	3993	879	3707
BLSV24	3429	357	2905	BLSV64	3976	428	3966
BLSV25	3008	250	2618	BLSV65	4000	941	3705
BLSV26	3302	272	2978	BLSV66	4153	988	3666
BLSV27	3378	314	3046	BLSV67	3930	467	3843
BLSV28	3164	339	2936	BLSV68	4066	293	4017
BLSV29	3106	240	2699	BLSV69	4167	823	3930
BLSV30	3100	324	2829	BLSV70	4095	1023	3827
BLSV31	3162	336	2634	BLSV71	3887	858	3430
BLSV32	3186	284	2751	BLSV72	3941	954	3544
BLSV33	3198	389	2698	BLSV73	3955	730	3665
BLSV34	3166	337	2707	BLSV74	4193	910	3936
BLSV35	3161	306	2828	BLSV75	4038	1093	3826
BLSV36	3213	265	2831	BLSV76	4021	751	3664
BLSV37	3450	287	2917	BLSV77	3841	865	3460
BLSV38	3242	286	2712	BLSV78	4203	611	4103
BLSV39	3387	428	3003	BLSV79	4069	700	3996
BLSV40	3251	329	2821	BLSV80	3845	1058	3547

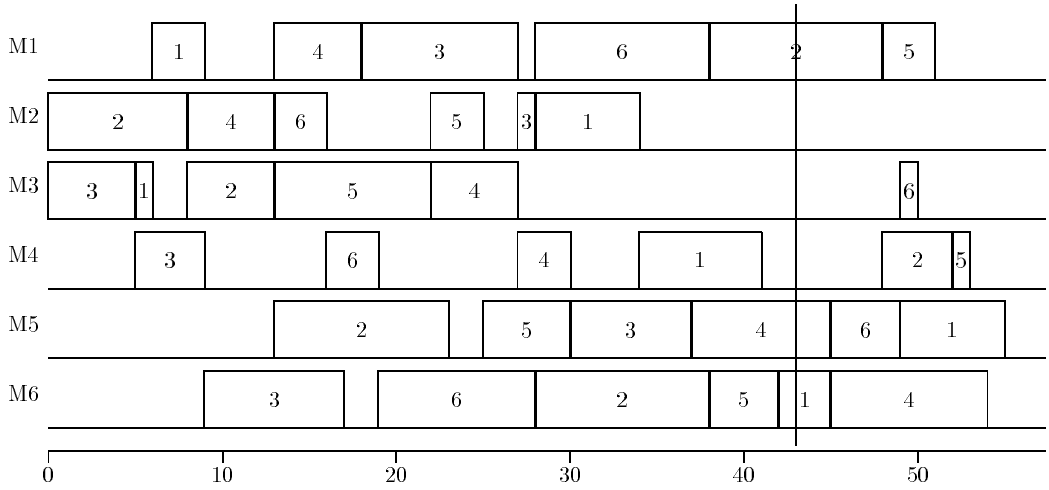

Figure 3 - a

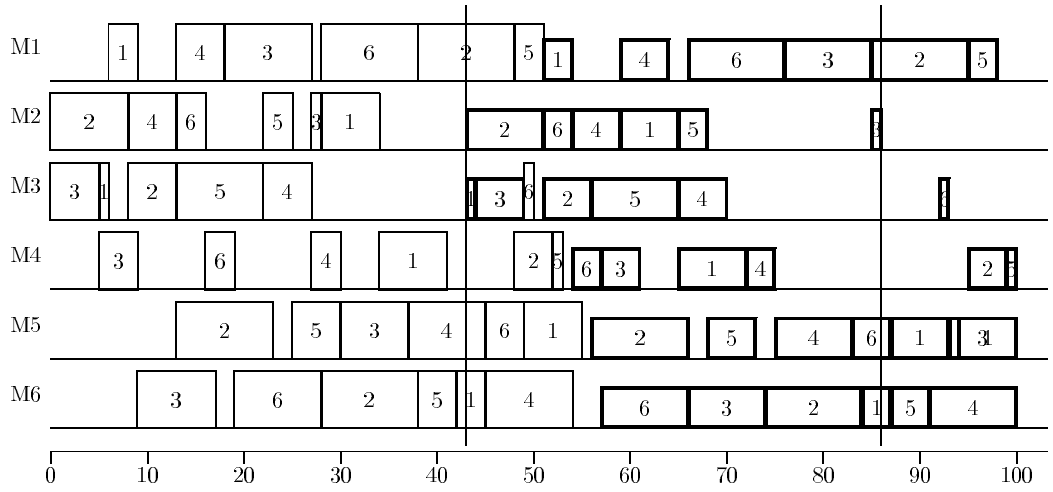

Figure 3 - b

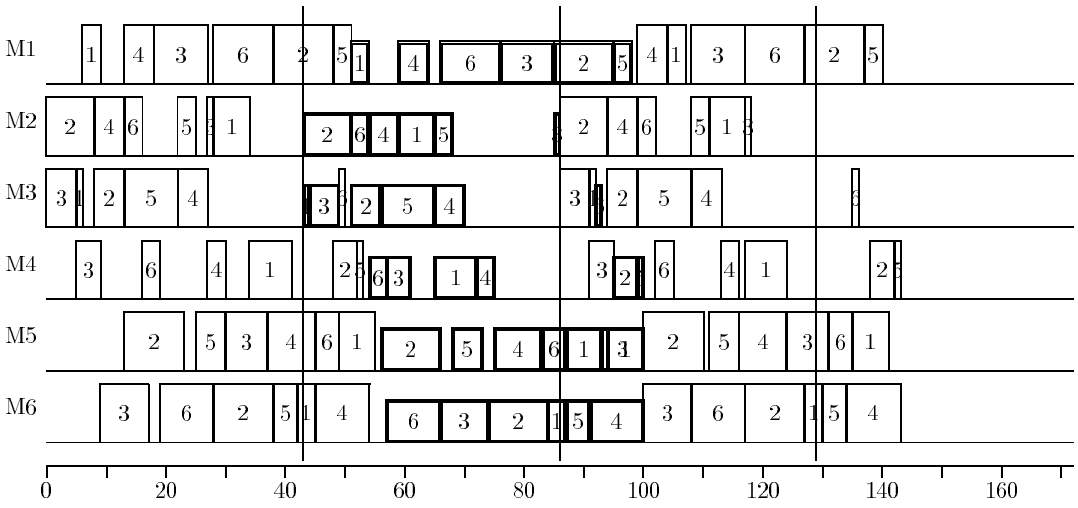

Figure 3 - c

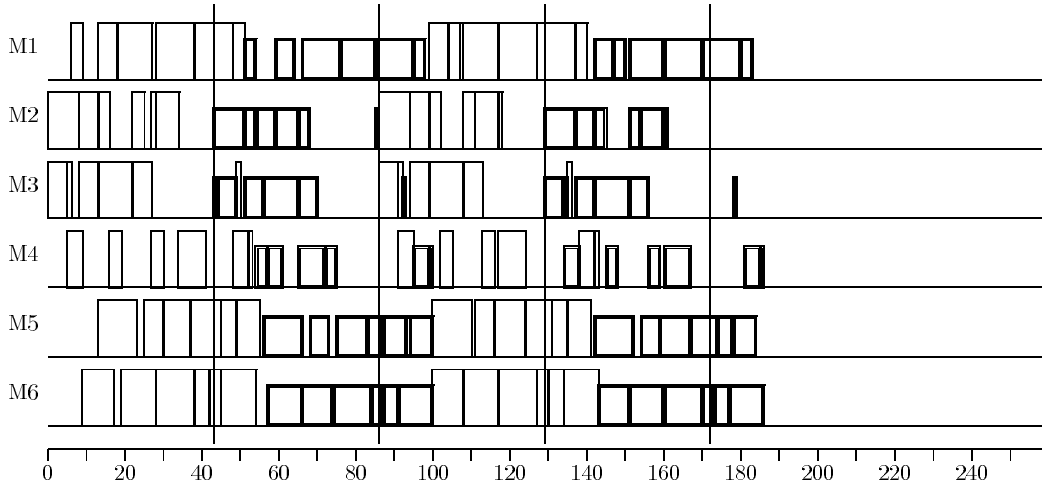

Figure 3 - d