Carnegie Mellon University

PITTSBURGH, PENNSYLVANIA 15213

Graduate School of Industrial Administration

WILLIAM LARIMER MELLON, FOUNDER

Management Science Research Report #MSRR-589

ONE MACHINE SCHEDULING WITH DELAYED PRECEDENCE CONSTRAINTS

by

Egon Balas*
Jan Karel Lenstra**
Alkis Vazacopoulos*

December 1992

The research underlying this report was supported by the National Science Foundation, Grant #DDM-8901495 and the Office of Naval Research through Contract N00014-85-K-0198.

Management Science Research Group
Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213-3890

* Graduate School of Industrial Administration, Carnegie Mellon University.
** Department of Mathematics & Computing Science, Eindhoven University; CWI, Amsterdam.

Abstract

We study the one machine scheduling problem with release and delivery times and the minimum makespan objective, in the presence of constraints that for certain pairs of jobs require a delay between the completion of the first job and the start of the second (delayed precedence constraints). This problem arises naturally in the context of the Shifting Bottleneck Procedure for the general job shop scheduling problem, as a relaxation of the latter, tighter than the standard one machine relaxation. The paper first highlights the difference between the two relaxations through some relevant complexity results. Then it introduces a modified Longest Tail Heuristic whose analysis identifies those situations that permit efficient branching. As a result, an optimization algorithm is developed whose performance is comparable to that of the best algorithms for the standard one machine problem. Embedding this algorithm into a modified version of the Shifting Bottleneck Procedure that uses the tighter one machine relaxation discussed here results in a considerable overall improvement in performance on all classes of job shop scheduling problems.

1 Introduction

The problem that we address in this paper arises in the context of general job shop scheduling. In the job shop scheduling problem, jobs are to be processed on machines with the objective of minimizing some function of the completion times of the jobs, subject to the constraints that (i) the sequence of machines for each job is prescribed, and (ii) each machine can sequence at most one job at a time. The processing of a job on a machine is called an operation, and its duration is a given constant. The objective chosen here is that of minimizing the time needed to complete all the jobs, called the makespan.

Let $N = \{0, 1, \ldots, n\}$ denote the set of operations, with 0 and n the dummy "start" and "finish" operations, respectively, M the set of machines, A the set of pairs of operations constrained by precedence relations representing condition (i) above, and E_k the set of pairs of operations to be performed on machine k and which therefore cannot overlap in time, as specified in (ii). Further, let d_j denote the (fixed) duration or processing time, and t_j the (variable) start time of operation j. The problem can then be stated as

$$
\begin{array}{lll}
 & \text{minimize } t_n & \\
\text{subject to} & & \\
 & t_j - t_i \geq d_i & (i,j) \in A \\
(P) & t_i \geq 0 & i \in N \\
 & t_j - t_i \geq d_i \quad \vee \quad t_i - t_j \geq d_j & (i,j) \in E_k,\ k \in M.
\end{array}
$$

Any feasible solution to (P) is called a schedule.

It is useful to represent this problem on a *disjunctive graph* $G := (N, A, E)$ with node set N, (directed) arc set A, and (undirected, but orientable) edge set E. The length of an arc $(i,j) \in A$ is d_i, whereas the length of an edge $(i,j) \in E$ is either d_i or d_j, depending on its orientation. Each machine k corresponds to a set N_k of nodes (operations) and a set E_k of edges that together form a disjunctive clique. Figure 1 shows a disjunctive graph representing a job shop scheduling problem with 3 jobs and 3 machines. The disjunctive cliques corresponding to the 3 machines have node sets $N_1 := \{1, 5, 8\}$, $N_2 := \{2, 4, 9\}$, $N_3 := \{3, 6, 7\}$, with their edges drawn in dotted lines.

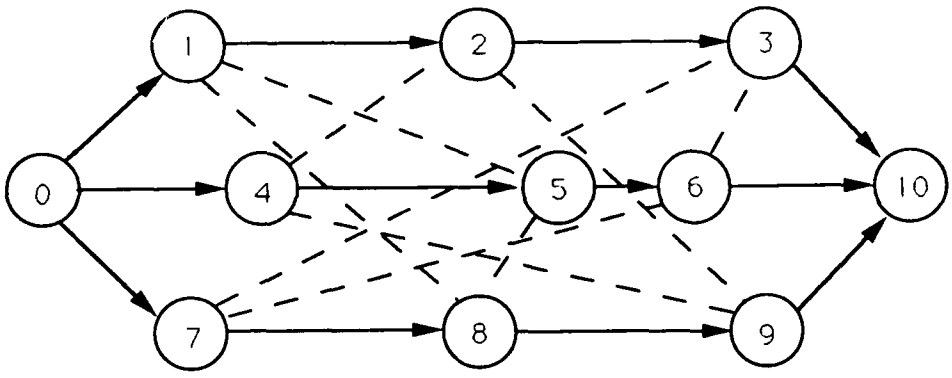

Figure 1: Disjunctive graph for a 3-job 3-machine problem

The job shop scheduling problem, known to be strongly NP-hard, is among the hardest. This is dramatically illustrated by the fact that an instance involving 10 jobs and 10 machines, proposed by Fisher and Thompson [FT63] in 1963, remained unsolved for more than a quarter of a century, although every available algorithm was tried on it. This makes it imperative to design efficient approximation methods.

The Shifting Bottleneck (SB) Procedure [ABZ88] is one such method. It sequences the machines consecutively, one at a time, with the remaining unsequenced machines ignored (i.e. the corresponding edge sets removed) and the machines already sequenced held fixed (i.e. the corresponding edges replaced by directed arcs). At each step a bottleneck machine is determined from among those not yet sequenced, by solving a one machine scheduling problem for each unsequenced machine and choosing the one with the maximum makespan. This bottleneck machine is then sequenced optimally by using the solution to the corresponding one machine problem. Once all the machines have been sequenced, each machine in turn is freed up and resequenced, with the sequences on the remaining machines held fixed.

While the Shifting Bottleneck Procedure, having been extensively tested, was found

more efficient than any version of the traditional dispatching rule-based heuristics, whether deterministic or randomized, several researchers (see Dauzière-Peres and Lasserre [DL90], as well as Tiozzo [T88]) found the following weakness or shortcoming in the procedure. When the procedure fixes the sequence on a machine, it may thereby create a precedence constraint between some pair of jobs on some unsequenced machine. In terms of the disjunctive graph, orienting the edges of a disjunctive clique (i.e. replacing them with directed arcs) creates new paths in the graph, some of which may join two nodes of a disjunctive clique. If a (directed) path is created between nodes i and j of a disjunctive clique, this implies the constraint $t_j - t_i \geq L(i,j)$, where $L(i,j)$ is the length of the path from i to j. For instance, if in the graph of Figure 1 the edges joining the node set $N_1 := \{1,5,8\}$ are replaced, as a result of sequencing machine 1, by the directed arcs $\{(5,1),(5,8),(1,8)\}$, that replacement creates the path $\{(4,5),(5,1),(1,2)\}$ from 4 to 2, of length $L(4,2) = d_4 + d_5 + d_1$, which in turn imposes the constraint $t_2 - t_4 \geq L(4,2)$ (see Figure 2).

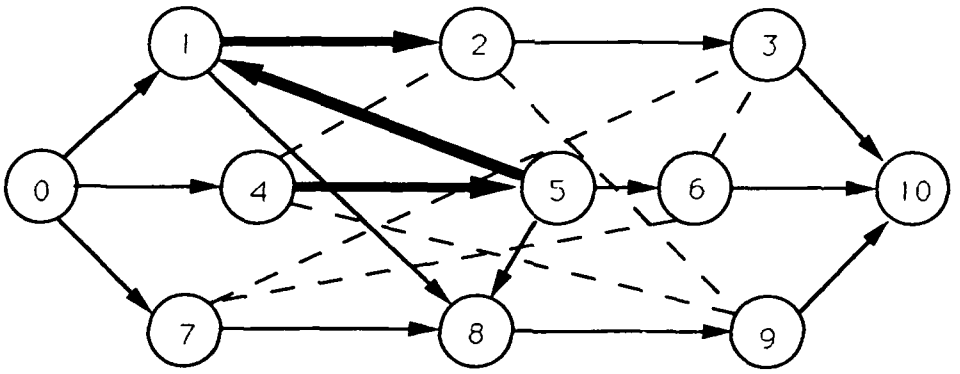

Figure 2: Disjunctive graph after sequencing machine 1

Note that the constraint $t_j - t_i \geq L(i,j)$ is different from the standard precedence constraint $t_j - t_i \geq d_i$ that would result from arbitrating the disjunction $t_j - t_i \geq d_i \vee t_i - t_j \geq d_j$,

in that the delay $L(i,j)$ involved in the constraint is different from (typically larger than) d_i, the delay involved in the standard precedence constraint. To make this distinction, we call the constraints described here precedence constraints with delay or, for short, *delayed precedence constraints* (DPC).

As [T88] and [DL90] correctly observed, the appearance of DPC's during the application of the SB Procedure vitiates the intent of the heuristic and may negatively affect the outcome. Indeed, by ignoring DPC's while solving a one machine problem, we are solving a less constrained problem than we should, hence we may make an unintended choice for the bottleneck machine and for the job sequence on that machine; all of which may negatively affect the quality of the schedule found by the SB Procedure. To remedy this situation, Dauzière-Peres and Lasserre [DL90] developed a heuristic for the one machine problem with DPC's. Using this heuristic instead of Carlier's exact algorithm for solving the one machine problems that arise in the SB Procedure, in other words substituting an approximation method applied to a better chosen problem for the exact method applied to the earlier problem, they managed to obtain better overall results.

Our purpose in this paper is to develop an optimization algorithm for solving the one machine problem with DPC's, in order to further improve the SB Procedure. Of course, using an optimization algorithm only pays if the gain in the quality of solutions to the one machine problem with DPC's is not outweighed by the excess of computational effort required. The kind of one machine problem solved in the SB Procedure (i.e. with release and delivery times) is strongly NP-complete even without DPC's; nevertheless, it can be solved in most cases quite efficiently due to the possibility of using a branching rule that skips large subsets of potential search tree nodes (see Potts [P80] and Carlier [C82]). That same efficient branching rule can also be used if some standard precedence constraints (as opposed to DPC's) are imposed, i.e. if some of the disjunctive arc pairs are arbitrated before solving the problem. However, this is no longer the case in the presence of DPC's.

The main accomplishment of this paper is an analysis of the structure of the one machine

scheduling problem with DPC's that identifies the situations in which the above mentioned "strong" branching rule can legitimately be used. As a result, an algorithm is developed which uses the strong branching rule whenever possible, and an alternative rule when necessary; with the practical outcome that the overall efficiency of the algorithm is comparable to that for the standard one machine problem.

The rest of the paper is organized as follows. The next section (2) discusses the relation between the standard one machine problem and its counterpart involving delayed precedence constraints, highlighting the difference through some relevant complexity results. Section 3 states and analyzes the Longest Tail Heuristic for the one machine problem with DPC's. Its focus is on identifying the situations in which the Potts-Carlier property still holds, thus permitting efficient branching. Section 4 describes a branch and bound algorithm based on the foregoing analysis. Finally, the last section (5) discusses computational results with the algorithm both as a stand-alone and as part of an improved SB Procedure.

Some of the results of this paper were presented in [BV91].

2 The One Machine Problem with DPC's Versus the Standard One Machine Problem

At some point during the SB Procedure, let M_0 be the set of machines already sequenced, and let $D(M_0)$ be the directed graph obtained from G by replacing all disjunctive arc sets corresponding to the machines in M_0 with the conjunctive arc sets representing the sequences chosen for those machines, and deleting all disjunctive arc sets corresponding to the machines in $M \setminus (M_0 \cup \{k\})$ for some fixed $k \in M \setminus M_0$. Then the standard one machine scheduling problem to be solved for machine k can be stated as

$$
\begin{array}{lll}
 & \text{minimize } t_n & \\
\text{subject to} & & \\
 & t_n - t_i \geq d_i + q_i & i \in I \\
P(k, m_0) & t_i \geq r_i & i \in I \\
 & t_j - t_i \geq d_i \ \vee \ t_i - t_j \geq d_j, & (i,j) \in E_k
\end{array}
$$

where I is the set of jobs to be processed on machine k, r_i is the release time (or head) of job i, equal to the length $L(0,i)$ of a longest path from node 0 to node i in $D(M_0)$, and q_i is the delivery time (or tail) of job i, equal to $L(i,n) - d_i$, where $L(i,n)$ is the length of a longest path from node i to node n in $D(M_0)$.

On the other hand, the one machine problem with DPC's for the same machine k can be stated as

$$
\begin{array}{rl}
 & \text{minimize } t_n \\
\text{subject to} & \\
DPC(k,M_0) & \begin{array}{rcll}
t_n - t_i & \geq & d_i + q_i & i \in I \\
t_i & \geq & r_i & i \in I \\
t_j - t_i & \geq & L(i,j) & (i,j) \in F \\
t_j - t_i \geq d_i & \vee & t_i - t_j \geq d_j & (i,j) \in E_k,
\end{array}
\end{array}
$$

where F is the set of precedence arcs, i.e. pairs with delayed precedence constraints, and $L(i,j)$ is the length of a longest path from i to j in the $D(M_0)$.

It is a well known fact (see [GJ79]) that the standard one machine problem $P(k,M_0)$ is NP-complete in the strong sense. Since the problem $DPC(k,M_0)$ is a generalization of $P(k,M_0)$ (it becomes $P(k,M_0)$ when $F = \emptyset$), it is also NP-complete in the strong sense. The question that we want to address now, is how much more difficult is $DPC(k,M_0)$ than $P(k,M_0)$?

To gain some insight into this issue, we recall two well known facts about $P(k,M_0)$. We call the Longest Tail rule the scheduling of jobs in order of decreasing tail length, and the Shortest Head rule the scheduling of jobs in order of increasing head length.

Fact 1. If $r_i = r_j$ for all $i,j \in I$, then the Longest Tail rule yields an optimal schedule. If $q_i = q_j$ for all $i,j \in I$, then the Shortest Head rule yields an optimal schedule.

Fact 2. Fact 1 remains true if preemption (job splitting) is allowed.

Note that Longest Tail (Shortest Head) scheduling requires $O(n \log n)$ time.

If standard (as opposed to delayed) precedence constraints are imposed on the one machine problem, i.e. constraints of the form $t_j - t_i \geq d_i$ for some $(i,j) \in E_k$, Facts 1 and 2 remain true, provided that the Longest Tail (Shortest Head) rule is applied subject to the

precedence constraints. However, in the presence of delayed precedence constraints, both of the above properties break down and, as shown by the next two theorems, there is no polynomial-time algorithm even for the more restricted case where $r_i = r_j$ and $q_i = q_j$ for all $i, j \in I$, unless $P = NP$.

For the next two Theorems we will use a slightly modified version of the DPC's. Namely, instead of requiring that the start time of job i precede the start time of job j by at least $L(i,j)$ units, we will require that the completion time of job i precede the start time of job j by at least $L'(i,j) := L(i,j) - d_i$ units. For the one machine scheduling problem with DPC's in which preemption is not allowed this makes no difference, since the completion time of job i is $C_i := t_i + d_i$, hence the two forms of the DPC, $t_j - t_i \geq L(i,j)$ and $t_j - C_i \geq L'(i,j)$ are equivalent. But in the case when preemption is allowed this is no longer true, since a preempted job i will be completed later than its starting time plus its processing time; so the new condition yields a stronger relaxation of the one machine problem with DPC's.

Theorem 2.1 *$DPC(k, M_0)$ is NP-hard in the strong sense even if $r_i = r_j$ and $q_i = q_j$ for all $i, j \in I$.*

Proof. Our proof is based on a reduction of *3-PARTITION*. The problem is defined as follows: Given $3n + 1$ positive integers $a_1, \ldots, a_{3n}, b$, with $b/4 < a_i < b/2$ for $i = 1, \ldots, 3n$, and $\sum_{i=1}^{3n} a_i = nb$, does there exist a partition of $\{1, \ldots, 3n\}$ into n pairwise disjoint subsets $S_1, \ldots, S_n$ such that $\sum_{i \in S_h} a_i = b$ for $h = 1, \ldots, n$?

We will show that for every instance of 3-PARTITION we can compute an instance of $DPC(k, M_0)$ and a value z in polynomial time, such that the instance of the former problem has a yes answer if and only if the instance of the latter problem has a schedule of length at most z. The Theorem then follows from the known fact that 3-PARTITION is NP-complete in the strong sense [GJ79].

Given an instance of 3-PARTITION as stated above, we construct the following instance of $DPC(k, M_0)$. There are $4n + 1$ jobs. All their release and delivery times are 0. The first

$3n$ jobs have processing times

$$d_i = a_i \qquad \text{for } i = 1, \ldots, 3n;$$

there are no precedence constraints between these jobs. The remaining $n + 1$ jobs have processing times

$$d_i = b \quad \text{for } i = 3n + 1, \ldots, 4n + 1$$

and also are subject to the following:

$$i \text{ has to precede } i + 1 \text{ with } L'(i, i + 1) = b \text{ for } i = 3n + 1, \ldots, 4n.$$

Finally, we define $z = 2nb + b$. Note that the precedence constraints define a single chain.

Suppose now that the instance of 3-PARTITION is a yes instance, i.e. there exist n pairwise disjoint subsets S_h with $\sum_{i \in S_h} a_i = b$ for $h = 1, \ldots, n$. We then construct a schedule of length z as follows. The jobs $3n + 1, \ldots, 4n + 1$ are started as early as possible, subject to the delayed precedence constraints, i.e. $t_{3n+h} = (2h - 2)b$ for $h = 1, \ldots, n + 1$. Note that job $4n + 1$ finishes at time z. This leaves n idle intervals $[(2h - 1)b, 2hb]$, $h = 1, \ldots, n$, of length b each, in which we process the jobs i with $i \in S_h$ for $h = 1, \ldots, n$.

Conversely, suppose that a schedule of length z exists. In such a schedule we must have $t_{3h+h} = (2h - 2)b$ for $h = 1, \ldots, n + 1$: none of these jobs can start earlier in view of the DPC's, and none of them can start later in view of the schedule length. It follows that all of the jobs $1, \ldots, 3n$ are processed in the n intervals $[(2h - 1)b, 2hb]$, $h = 1, \ldots, n$. Now define S_h as the set of jobs processed in the h^{th} of these intervals. We have $\sum_{i \in S_h} a_i = b$ for $h = 1, \ldots, n$, and hence we have a yes instance of 3-PARTITION. □

Theorem 2.2 *Theorem 2.1 remains true even if preemption is allowed.*

Proof. We again give a reduction of 3-PARTITION. The reader should verify that, when preemption is allowed, the reduction given in the proof of Theorem 2.1 no longer works: a schedule of length z always exists, irrespective of the answer to the 3-PARTITION instance.

Given an instance of 3-PARTITION, we construct the following instance of the preemptive scheduling problem with DPC's, with release and delivery times all equal to 0. There are $8n + 1$ jobs. For each a_i, there are now two jobs, i and $3n + i$, with

$$d_i = d_{3n+i} = a_i \qquad \text{for } i = 1, \ldots, 3n$$

and the condition that

$$i \text{ has to precede } 3n + i, \text{ with } L'(i, 3n + i) = 2nb - b \text{ for } i = 1, \ldots, 3n.$$

The remaining jobs have processing times

$$d_i = b \qquad \text{for } i = 6n + 1, \ldots, 8n + 1$$

and are subject to the condition that

$$i \text{ has to precede } i + 1, \text{ with } L'(i, i + 1) = b \text{ for } i = 6n + 1, \ldots, 8n.$$

Finally, we define $z = 4nb + b$. Note that the precedence constraints define a collection of chains (one chain of length $2n - 1$ and $3n$ chains of length 1).

Suppose we have a yes instance of 3-PARTITION. A schedule of length z is then constructed as follows. We start the jobs $6n + 1, \ldots, 8n + 1$ as early as possible, i.e. $t_{6n+h} = (2h - 2)b$ for $h = 1, \ldots, 2n + 1$. Again, the last job finishes at time z. We are left with $2n$ intervals $[(2h - 1)b, 2hb]$ of length b each for the jobs $1, \ldots, 6n$. For $h = 1, \ldots, n$, we process the jobs i with $i \in S_h$ in $[(2n - 1)b, 2nb]$, and their successors $3n + i$ with $i \in S_h$ in $[(2(n + h) - 1)b, 2(n + h)b]$; this is feasible since the time span from the end of the former interval to the beginning of the latter is equal to the required minimum delay of $2nb - b$.

Conversely, suppose that we have a schedule of length z. As before, we must have $t_{6n+h} = (2h - 2)b$ for $h = 1, \ldots, 2n + 1$. Hence all of the jobs $1, \ldots, 6n$ are processed in the intervals $[2(h - 1)b, 2hb]$, $h = 1, \ldots, 2n$. Note that the total length of these intervals equals the total required processing time of these jobs, so that the schedule contains no idle

time. Also, in view of the DPC's, the jobs $1, \ldots, 3n$ must be processed in the first half of the schedule and their successors $3n+1, \ldots, 6n$ in the second half.

Consider the first interval, $[b, 2b]$, and let S_1 be the set of jobs started and completed in this interval. Assume that $\sum_{i \in S_1} a_i = b - c$ for some $c > 0$. The only jobs $3n + i$ that are available for processing in $[(2n+1)b, (2n+2)b]$ are those for which $i \in S_1$, and hence that interval contains c units of idle time. It follows that our assumption is incorrect and that $\sum_{i \in S_1} a_i = b$. An iterative application of this argument leads to the identification of sets S_h with $\sum_{i \in S_h} a_i = b$ for $h = 2, \ldots, n$, and to the conclusion that we have a yes instance of 3-PARTITION. □

As mentioned in the introduction, the standard one machine problem, although NP-complete, can in practice be solved efficiently for fairly large sizes. This can be done by an algorithm developed by Carlier [C82], based on an analysis of the properties of the Longest Tail Heuristic (for the latter see also Potts [P80]). The Longest Tail Heuristic for the standard one machine problem schedules the jobs sequentially, choosing at each step the job with the longest tail among those available for scheduling. If C is a critical path in the digraph corresponding to the resulting schedule, let p be the first node encountered when traversing C backwards starting from n (not counting n), let c be the first node (if any) encountered such that $q_c < q_p$, and let J be the set of nodes traversed (not counting n and c). Potts' and Carlier's analysis established that (i) if there is no node c with the above property, the schedule at hand is optimal; and (ii) otherwise in any optimal schedule job c comes either before or after all the jobs in J.

Thus Carlier's branch and bound algorithm applies the Longest Tail Heuristic to the problem at hand at any given node of the search tree. If the resulting schedule is optimal (i.e. satisfies (i) above), the subproblem corresponding to the node is discarded; otherwise it is replaced by two new subproblems, obtained from the parent problem by imposing the condition that job c has to precede (first descendant), respectively succeed (second

descendant) all jobs in J. Carlier also uses the fact that for any $I \subseteq N$ the value

$$h(I) := \min_{i \in I} r_i + \sum_{i \in I} d_i + \min_{i \in I} q_i$$

is a valid lower bound on the makespan, and applies it to the subproblems generated by branching, with $I = J$ for the first and $I = J \cup \{c\}$ for the second descendant.

The efficiency of Carlier's procedure is due to the fact that in the absence of property (ii) above, all the situations in which job c precedes some but not all the jobs in J would have to be considered.

It is easy to see that in the presence of standard (as opposed to delayed) precedence constraints, properties (i) and (ii) still hold. Therefore Carlier's branch and bound algorithm can be (and has been) modified to accommodate such constraints without any loss in efficiency.

This, however, is not the case when delayed precedence constraints are imposed, as will be seen in the next section.

3 The Longest Tail Heuristic in the Presence of DPC's, and Its Analysis

We will now state a modified version of the Longest Tail Procedure, that takes into account the presence of DPC's, and analyze its properties. We denote, as before, by I the set of jobs to be processed on the given machine. Further, we introduce

$$\begin{aligned} S &:= \{i \in I : i \text{ has been scheduled }\} \\ \pi(i) &:= \{j \in I : j \text{ is required to precede } i\} \\ &\quad (= \text{ predecessors of } i) \\ \sigma(i) &:= \{j \in I : j \text{ is required to succeed } i\} \\ &\quad (= \text{ successors of } i) \end{aligned}$$

The sets $\pi(i)$ and $\sigma(i)$ can be used to tighten the release times and tails by capturing the logical implications of the precedence constraints. Thus the numbers r_j, q_j can be updated as follows:

$$r_j := \max\{r_j, \max_{i \in \pi(j)}\{r_i + L(i,j)\}, \min_{i \in \pi(j)} r_i + \sum_{i \in \pi(j)} d_i\} \qquad (1)$$

and

$$q_j := \max\{q_j,\ \max_{i\in\sigma(j)}\{L(j,i) - d_j + d_i + q_i\},\ \sum_{i\in\pi(j)} d_i + \min_{i\in\pi(j)} q_i\} \tag{2}$$

Longest Tail Heuristic for DPC's

Initialization. Set $t := 0$, $S := \emptyset$, $r'_i = r_i$, $i \in I$.

Iterative Step. Let Q be the set of unscheduled jobs whose predecessors have all been scheduled, i.e.

$$Q := \{i \in I \setminus S : \pi(i) \subseteq S\}$$

Choose k such that

$$q_k := \begin{cases} \max\{q_i : i \in Q \text{ and } r'_i \le t\} & \text{if such } i \text{ exists} \\ \max\{q_i : i \in Q \text{ and } r'_i := \min\{r'_j : j \in Q\}\} & \text{otherwise.} \end{cases}$$

Set $t_k := \max\{t, r'_k\}$, $S := S \cup \{k\}$, $t := t_k + d_k$.

For all $j \in \sigma(k)$, set $r'_j := \max\{r'_j,\ t_k + L(k,j)\}$.

If $I \setminus S \neq \emptyset$, repeat the Iterative Step. Otherwise, define the set of *delayed jobs*

$$D := \{i \in I : r'_i > r_i\}$$

and stop.

The outcome of the longest tail procedure is a schedule $t := (t_i)$ which can be represented by a directed graph $G(t)$ with one or several longest paths (critical paths) from 0 to n. Figure 3 shows such a solution digraph for a problem with 6 jobs. The critical path (0,4,2,6,7) is drawn in heavy lines. The i^{th} component of t, i.e. the start time of the i^{th} job (not shown in Figure 3), is equal to the length of a longest path from 0 to node i. The makespan is the length of a longest path (critical path) from 0 to n.

For the standard one machine problem, it is a well known fact that for any subset $K \subset I$, the number

$$h(K) := \min_{i\in K} r_i + \sum_{i\in K} d_i + \min_{i\in K} q_i$$

is a lower bound on the makespan. Furthermore, the strongest such bound, $\max\{h(K) : K \subseteq I\}$, is known to be equal to the minimum makespan of the preemptive version of

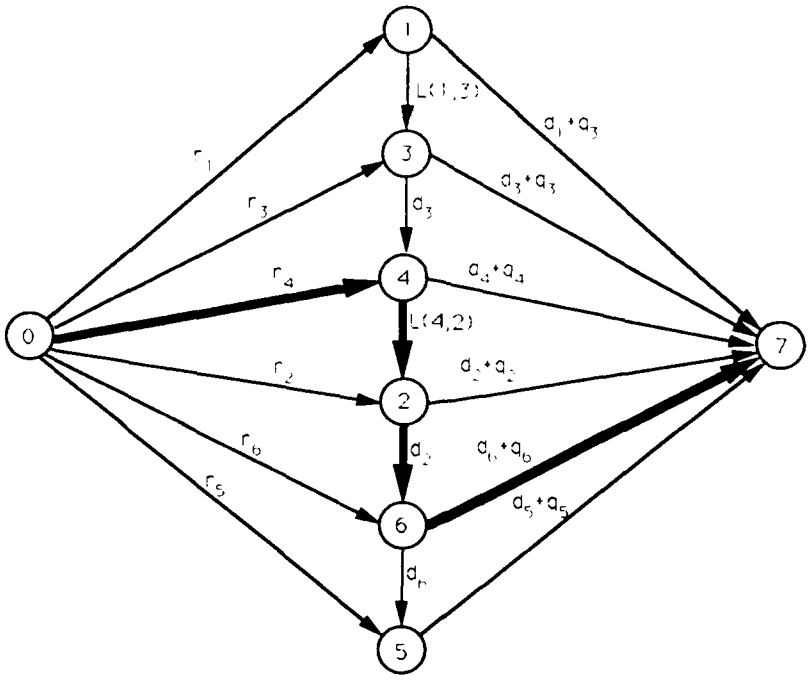

Figure 3: Digraph of a schedule, with critical path

the problem. It is easy to see that these bounds are also valid for our problem, of which the standard problem is a relaxation. In particular, if $C := \{(0, i_1), (i_1, i_2), \ldots, (i_p, n)\}$ is a critical path in $G(t)$ and $I_C := \{i_1, \ldots, i_p\}$, then

$$h(I_C) := \min_{i \in I_C} r_i + \sum_{i \in I_C} d_i + \min_{i \in I_C} q_i$$

is a lower bound on the makespan.

Now let t be the schedule produced by the Longest Tail Procedure, $G(t)$ the associated digraph, and $C := \{(0, i_1), (i_1, i_2), \ldots, (i_p, n)\}$ a critical path in $G(t)$, with $I_C := \{i_1, \ldots, i_p\}$. Further, let c be the first node encountered when traversing C backwards from node n such that $q_c < q_{i_p}$, if such c exists, or else let $c = 0$, with $t_0 := 0$. Finally, let J be the set of nodes of C between n and c (not counting n and c). We then have the following

Theorem 3.1 *Suppose* $r_i \geq \max\{t_{i_1}, t_c\}$, $i \in J$, *and the segment* $C(c, i_p)$ *of the critical path* C *between* c *and* i_p *contains no precedence arc. Then*

(i) If $c = 0$, t is optimal;

(ii) If $c > 0$, then in any schedule better than t, job c either precedes or succeeds all jobs in J.

Proof. (i) Let $c = 0$. By assumption $r_i \geq r_{i_1}$ for each $i \in J$, hence for all $i \in I_C$; hence $r_{i_1} = \min_{i \in I_C} r_i$. Also, since the arc $(0, i_1)$ is on a critical path, $t_{i_1} = r_{i_1}$. Further, $q_i \geq q_{i_p}$ for all $i \in I_C$ by hypothesis, hence $q_{i_p} = \min_{i \in I_C} q_i$. Thus the lower bound $h(I_C)$ is in this case $r_{i_1} + \sum_{i \in I_C} d_i + q_{i_p}$. But this is precisely the length of C, i.e. the makespan, since $t_{i_1} = r_{i_1}$, $t_{i_{k+1}} = t_{i_k} + d_{i_k}$ for $k = 1, \ldots, p-1$, and $t_n = t_{i_p} + d_{i_p} + q_{i_p}$. Thus t is optimal.

(ii) Suppose now that $c > 0$, and let t' be a schedule in which c comes after some (but not all) jobs in J. Since the total processing time of all jobs in $J \cup \{c\}$ is in both schedules the same, and the tail of the last job in t' is at least as long as q_{i_p}, we can have $t'_n < t_n$ only if $t'_{j_*} < t_c$ for some $j_* \in J$, which would imply $r_{j_*} < t_c$. But $r_j \geq t_c$ for all $j \in J$ by hypothesis, hence $t'_n \geq t_n$, i.e. t' is not better than t. □

Notice that the condition $r_i \geq \max\{t_{i_1}, t_c\}$ for all $i \in J$, always satisfied in a schedule produced by the Longest Tail Heuristic for the standard one machine problem, may not hold here and therefore needs to be explicitly imposed. Indeed, although the updated release times r'_i generated during the procedure always satisfy $r'_i \geq \max\{t_{i_1}, t_c\}$ for all $i \in J$ (which explains why job i was not chosen before jobs i_1 or c), one may still have $r_i < \max\{t_{i_1}, t_c\}$.

Theorem 3.1 identifies the situation in which the same efficient branching rule used in Carlier's algorithm can be used in the presence of DPC's. Next we address the situation in which the assumptions of Theorem 3.1 do not hold.

First of all, note that if a critical path C in $G(t)$ contains a precedence arc (i, j) such that $r_j = t_j$, then $\{(0, j)\} \cup C(j, n)$ is a critical path that does not contain (i, j). Similarly, if C contains a precedence arc (i, j) such that $q_i = L(0, n) - t_i - d_i$, where $L(0, n)$ is the makespan associated with t, then $C(0, i) \cup \{(i, n)\}$ is a critical path not containing (i, j). Such precedence arcs are of no consequence, in the sense that a critical path containing such an

arc can be replaced by one not containing it. Also, the DPC's discussed here are not the only kind of precedence constraints that can occur. Whenever some disjunction can be arbitrated as a result of logical tests (see the next section), the replacement of the corresponding edge with a directed arc creates a (nondelayed) precedence arc with a constraint of the type $t_j - t_i \geq d_i$. These nondelayed precedence arcs are also of no consequence. In order to distinguish those precedence arcs that matter, we will call a precedence arc (i,j) *essential* (with respect to t) if $L(i,j) > d_i$, $r_j < t_j$ and $q_i < L(0,n) - t_i - d_i$.

Proposition 3.2 *If every critical path of $G(t)$ contains a precedence arc, then every critical path of $G(t)$ contains an essential precedence arc.*

Proof. Suppose every critical path of $G(t)$ contains a precedence arc, but there is a critical path C that contains no essential precedence arc. Let (i,j) be the first precedence arc of C. Since (i,j) is not essential, either $r_j = t_j$ or $q_i = L(0,n) - t_i - d_i$. Let $C' := \{(0,j)\} \cup C(j,n)$ in the first case, and $C' := C(0,i) \cup \{(i,n)\}$ in the second. In either case, C' is a critical path containing at least one less precedence arcs than C. Repeating this argument for as long as the critical path at hand continues to have a precedence arc yields a critical path without any precedence arc, a contradiction. □

From now on we focus on essential precedence arcs.

The reasoning used in the proof of Theorem 3.1 can also be used to identify other situations in which certain jobs have to be scheduled before (or after) other jobs in any schedule better than t. Recognizing such situations and acting on them can in turn be used to get rid of certain essential precedence arcs.

Proposition 3.3 *Suppose the segment $C(j,n)$ of C, where $j = i_l$, contains no precedence arc. Let $K := \{i_{l+1}, \ldots, i_p\}$ and suppose $q_k \geq q_{i_p}$ for all $k \in K \cup \{j\}$, and $r_k \geq t_j$ for all $k \in K \setminus \sigma(j)$.*

Then in any schedule better than t, job j precedes all jobs in K.

Proof. Let t' be a schedule in which job j comes after some or all of the jobs in K. Since the total processing time of all jobs in $K \cup \{j\}$ in t' cannot be less than what it is in t, namely $\sum_{k \in K \cup \{j\}} d_i$, and the tail of the last job in t' is by hypothesis at least equal to q_{i_p}, we can have $t'_n < t_n$ only if $t'_{j_*} < t_j$ for some $j_* \in K$, which would imply $r_{j_*} < t_j$. But according to our assumptions, $r_k \geq t_j$ for all $k \in K \setminus \sigma(j)$, and for $k \in \sigma(j)$ k cannot precede j in t'. Hence $t'_n \geq t_n$, i.e. t' is not better than t. □

Now suppose the segment $C(j,n)$ with the properties stated in Proposition 3.3 is preceded on the critical path C by an (essential) precedence arc $(i,j) = (i_{l-1}, i_l)$. Then in any schedule better than t, job i precedes all the jobs in $\{j\} \cup K$. Hence one can set

$$\begin{array}{rcl} \sigma(i) & := & \sigma(i) \cup K, \\ \pi(k) & := & \pi(k) \cup \{i\}, \quad k \in K \end{array} \tag{3}$$

and the tail of job i can be increased to

$$\begin{array}{rcl} q_i & := & L(i,j) + d_j + \sum_{k \in K} d_k + q_{i_p} - d_i \\ & = & L(0,n) - t_i - d_i, \end{array} \tag{4}$$

which gives rise to a new critical path, $C(0,i) \cup \{(i,n)\}$, that does not contain the precedence arc (i,j). Thus a situation like this can be used to render an essential precedence arc inessential.

Next we consider the converse of the above situation.

Proposition 3.4 *Suppose the segment $C(0,i)$ of C, where $i = i_l$, contains no precedence arc. Let $K := \{i_1, \ldots, i_{l-1}\}$ and suppose $r_k \geq r_{i_1}$ for all $k \in K \cup \{i\}$ and $q_k \geq L(0,n) - t_i - d_i$ for all $k \in K \setminus \pi(i)$.*

Then in any schedule better than t, job i succeeds all jobs in K.

Proof. The makespan associated with t is $t_n = r_{i_1} + \sum_{h \in K} d_h + L(i,n)$, where $L(i,n) = L(0,n) - t_i$ is the length of the critical path segment $C(i,n)$. Now let t' be a schedule in which job i precedes some or all of the jobs in K. Since the processing time of all jobs in K in t' cannot be less than $\sum_{h \in K} d_h$ and the release time of each job in $K \cup \{i\}$ is by hypothesis

at least equal to r_{i_1}, it follows that $\max_{h\in K\cup\{i\}}\{t'_h + d_h\} \geq t_i + d_i$. Therefore we can have $t'_n < t_n$ only if the part of the schedule t' starting after completion of the job k such that $t'_k + d_k := \max_{k\in K\cup\{i\}}\{t'_h + d_h\}$, is shorter than the corresponding part of the schedule t (i.e. the part of t starting after the completion of job i), whose length is $L(i,n) - d_i = L(0,n) - t_i - d_i$. But since $q_k \geq L(0,n) - t_i - d_i$ for all $k \in K \setminus \pi(i)$ by assumption, and i cannot precede $k \in \pi(i)$ in t', this is impossible and therefore $t'_n \geq t_n$, i.e. t' is not better than n. □

Similarly to the previous case, if the segment $C(0,i)$ that is the object of Proposition 3.4 is followed on the critical path C by an (essential) precedence arc $(i,j) = (i_l, i_{l+1})$, then in any schedule better than t, job j succeeds all the jobs in $K \cup \{i\}$. Hence one can set

$$\begin{array}{rcl} \pi(j) & := & \pi(j) \cup K \\ \sigma(k) & := & \sigma(k) \cup \{j\}, \quad k \in K \end{array} \tag{5}$$

and the release time of job j can be increased to

$$r_j := r_{i_1} + \sum_{h\in K} d_h + L(i,j), \tag{6}$$

which again gives rise to a new critical path, $\{(0,j)\} \cup C(j,n)$, not containing the precedence arc (i,j). Thus we have a second way of rendering inessential a formerly essential precedence arc.

4 Branch and Bound for the One Machine Problem with DPC's

We implemented a branch-and-bound procedure that works as follows.

0. Initialization. The original problem is put on the list of active subproblems, after updating the release times and tails according to expressions (1), (2) of section 3, initializing the upper bound at $f := \infty$ and calculating a lower bound $h := \max_{Q\subseteq I} h(Q)$, where

$$h(Q) := \min_{i\in Q} r_i + \sum_{i\in Q} d_i + \min_{i\in Q} q_i \tag{7}$$

It is known (see [C82]) that $h(Q)$ can be calculated in $O(n \log n)$ time by solving the preemptive version of the standard one machine problem.

1. Subproblem Selection. The problem with the weakest (i.e. smallest) lower bound is selected for processing next, and is removed from the list.

2. Longest Tail Heuristic. The selected problem is processed by the longest tail heuristic of section 3 to obtain a schedule t. If $t_n < f$, the upper bound is updated to $f := t_n$.

3. Postprocessing. Propositions 3.3 and 3.4 are applied: the predecessor and successor sets, and the release times and tails, are updated by using expressions 3, 4, 5 and 6. If any tail has been changed, go to 2; otherwise go to 4.

4. Branching. We search for a critical path satisfying the conditions of Theorem 3.1, i.e. one that contains no precedence arc in the segment $C(c,n)$ and for which $r_i \geq \max\{t_{i_1}, t_c\}$ for all $i \in J \cap D$. If found, then either $c = 0$ and the schedule t is optimal for the given subproblem, hence the latter is discarded and we go back to step 1; or else $c > 0$ and we apply the *strong branching rule* to be described below. If the search fails, we repeat it on the reverse problem. This is obtained by reversing the direction of every arc, interchanging the release times with the tails, and reversing the precedence constraints, with the understanding that if the delay between i and j in the original problem is $L(i,j)$, then the delay between j and i in the reverse problem is $L(j,i) = L(i,j) - d_i + d_j$. The reverse problem is easily seen to be equivalent to the original one; thus if the search is successful on the reverse problem, we proceed in the same way as in case of success on the original problem.

If the search fails on both problems, i.e. either for every critical path C that contains no precedence arc in the segment $C(c,n)$ we have $r_i < \max\{t_{i_1}, t_c\}$ for some $i \in J \cap D$ (case 1), or else every critical path C contains some essential precedence arc in the segment $C(c,n)$ (case 2), we apply the *weak branching rule* (see below).

a. Strong branching rule. We replace the current subproblem by two new ones; a left subproblem, in which c is required to precede all jobs in J, i.e.

$$\begin{aligned} \sigma(c) &:= \sigma(c) \cup J, \\ \pi(k) &:= \pi(k) \cup \{c\}, \quad k \in J; \end{aligned} \tag{8}$$

and a right subproblem, in which c is required to succeed all jobs in J, i.e.

$$\begin{array}{rcl} \pi(c) & := & \pi(c) \cup J \\ \sigma(k) & := & \sigma(k) \cup \{c\}, \quad k \in J. \end{array} \tag{9}$$

The strong branching rule is preceded by the logical tests proposed by Carlier [C82] for the standard one machine problem, which obviously preserve their validity in the presence of DPC's for the situation when the strong branching rule applies.

Let $K := \{k : k \in I \setminus J \cup \{c\} \text{ and } d_k > f - h(J)\}$.

Test 1. If $h \in K$ and

$$r_k + d_k + \sum_{i \in J} d_i + q_{i_p} > f \tag{10}$$

then job k has to succeed all the jobs in J; so we set

$$\begin{array}{rcl} \pi(k) & := & \pi(k) \cup J \\ \sigma(i) & := & \sigma(i) \cup \{k\}, \quad i \in J. \end{array} \tag{11}$$

Test 2. If $k \in K$ and

$$\min_{i \in J} r_i + \sum_{i \in J} d_i + d_k + q_k > f \tag{12}$$

then job k has to precede all the jobs in J; so we set

$$\begin{array}{rcl} \sigma(k) & := & \sigma(k) \cup J \\ \pi(i) & := & \pi(i) \cup \{k\}, \quad i \in J. \end{array} \tag{13}$$

b. Weak branching rule. Select a pair of jobs (i, j) as follows. In case 1, let C be one of the critical paths without precedence arcs in $C(c, n)$. Let $j = c$ if $c \neq 0$, $j = i_1$ otherwise, and let i be the first node encountered when traversing the segment $C(c, n)$, such that $r_i < \max\{t_{i_1}, t_c\}$. In case 2, let C be a critical path containing a minimum number of precedence arcs in $C(c, n)$, and let (k, l) be the one nearest to n. Then, (k, l) is essential (since it was not eliminated by postprocessing), and $r_m < t_l$ for at least one $m \in J$ (otherwise Proposition 3.3 would apply). Let $j = l$ and let i be the first node encountered when traversing the segment $C(l, n)$ such that $r_i < t_l$.

In both cases, we replace the current subproblem by two new ones: a left subproblem, in which i is required to precede j, i.e.

$$\begin{array}{rcl} \sigma(i) & := & \sigma(i) \cup \{j\} \\ \pi(j) & := & \pi(j) \cup \{i\}, \end{array} \tag{14}$$

and a right subproblem, in which i is required to succeed j, i.e.

$$\begin{aligned} \pi(i) &:= \pi(i) \cup \{j\} \\ \sigma(j) &:= \sigma(j) \cup \{i\}. \end{aligned} \tag{15}$$

5. Updating and Lower Bounding

For each of the two new subproblems created, we update the release times and tails by using expressions (1) and (2). In fact, we use a strengthened version of (1) and (2), defined as follows:

Let $|\pi(j)| = m$. Let $H_0 := \pi(j)$,

$$r_{h(1)} := \min\{r_h : h \in H_0\},\ H_1 := H_0 \setminus \{h(1)\},$$

and for $k = 2, \ldots, m$, define recursively

$$r_{h(k)} := \min\{r_h : h \in H_{k-1}\},\ H_k := H_{k-1} \setminus \{h(k)\}.$$

Then (1) can be strengthened to

$$(1') \qquad r_j := \max\{r_j, \max_{i \in \pi(j)}\{r_i + L(i,j)\}, \max_{k \in \{0,1,\ldots,m\}}\{\min_{i \in H_k} r_i + \sum_{i \in H_k} d_i\}\}.$$

Similarly, if $|\sigma(j)| = s$, let $K_0 = \sigma(j)$,

$$q_{h(1)} := \min\{q_h : h \in K_0\},\ K_1 := K_0 \setminus \{h(1)\},$$

and for $l = 2, \ldots, s$, define

$$q_{h(l)} := \min\{q_h : h \in K_{l-1}\},\ K_l := K_{l-1} \setminus \{h(l)\}.$$

Then (2) can be strengthened to

$$(2') \qquad q_j := \max\{q_j, \max_{i \in \sigma(j)}\{q_i + L(i,j) + d_i - d_j\}, \max_{l \in \{0,1,\ldots,s\}}\{\sum_{i \in K_l} d_i + \min_{i \in K_l} q_i\}\}.$$

Next we calculate the lower bound $h(Q)$ of expression (7) by solving the preemptive version of the standard one machine problem. Finally, we put each of the two subproblems on the list of active problems and go to 1.

5 Computational Results

The branch and bound method of section 4 was implemented in C and tested on a number of randomly generated one machine problems with DPC's. It was then embedded into the Shifting Bottleneck Procedure in place of the subroutine for solving one machine problems. The resulting Modified Shifting Bottleneck Procedure (SB3) was tested on a number of job shop scheduling problems from the literature, and compared with other procedures.

We first discuss the computational testing of the algorithm of section 4 by itself. The test problems were randomly generated in the same way as those of [C82], except for the DPC's that were added. We set $d_{\max} \in \{50, 100\}$, $r_{\max} = q_{\max} = \frac{1}{50}nkd_{\max}$, where $n \in \{20, 50, 100\}$ and $k \in \{10, 15, 20\}$. Here n is the number of jobs and k is Carlier's "interval multiplier." The numbers r_i, q_i, d_i were randomly drawn from uniform distributions between 1 and $r_{\max}$, $q_{\max}$, and $d_{\max}$, respectively. A precedence constraint was generated between jobs i and j with a probability of $p_{ij} \in \{0, 0.02, 0.04, \ldots, 0.20\}$, and the delays $L(i,j)$ were generated as follows: for each $(i,j) \in F$, a number $l(i,j)$ was drawn from a uniform distribution over the interval $[1, \frac{1}{50}nkd_{\max}]$; and $L(i,j)$ was set to $l(i,j)$ if $l(i,j) > d_i$, and to $l(i,j) + d_i$ otherwise. For every combination of p_{ij} and k, 40 instances were generated. The results are shown in Tables 1-6. As one can see from these tables, the number of search tree nodes rarely exceeds the number of jobs, which is in contrast with the typical behavior of branch and bound algorithms on hard problems, but is comparable to (though not quite as good as) the performance of Carlier's algorithm on the standard one machine problem. The remarkable fact is that typically strong branching occurs more frequently than weak branching, and every strong branching discards a large number of potential nodes of the search tree.

The main test of the usefulness of the algorithm for the one machine problem with delayed precedence constraints is the effect on the performance of the Shifting Bottleneck Procedure of replacing the standard one machine problem by the one machine problem with DPC's as the tool for identifying bottleneck machines and scheduling them. This modified

Table 1. $n = 20$, $d_{\max} = 50$

k	Density of DPC's (%)	Search Tree Nodes Avg. (Max.)	CPU SPARC330 Seconds Avg. (Max.)	Proportion of Strong Branchings (%)
10	0	2.13 (45)	0.008 (0.183)	100.0
	2	5.98 (72)	0.027 (0.400)	81.7
	4	6.08 (33)	0.028 (0.217)	72.2
	6	39.50 (485)	0.293 (4.500)	57.2
	8	29.50 (198)	0.220 (1.567)	58.6
	10	80.85 (1,818)	0.587 (14.799)	50.4
	12	16.38 (100)	0.102 (0.700)	57.3
	14	29.75 (201)	0.212 (1.817)	51.1
	16	16.58 (134)	0.115 (1.317)	41.5
	18	20.20 (107)	0.147 (0.900)	50.8
	20	16.55 (126)	0.111 (1.200)	51.7
15	0	5.53 (21)	0.023 (0.083)	100.0
	2	8.88 (25)	0.035 (0.183)	86.9
	4	14.35 (135)	0.071 (0.867)	65.5
	6	14.60 (124)	0.088 (1.083)	60.6
	8	7.85 (18)	0.037 (0.100)	86.4
	10	8.38 (24)	0.047 (0.200)	73.8
	12	10.75 (66)	0.057 (0.400)	67.1
	14	6.30 (18)	0.031 (0.117)	84.3
	16	6.63 (23)	0.038 (0.150)	75.9
	18	5.13 (22)	0.031 (0.133)	67.3
	20	4.90 (18)	0.300 (0.117)	65.4
20	0	11.93 (168)	0.061 (1.183)	100.0
	2	6.45 (15)	0.025 (0.100)	97.6
	4	6.33 (22)	0.024 (0.133)	97.6
	6	5.28 (16)	0.024 (0.133)	94.6
	8	3.63 (8)	0.016 (0.033)	100.0
	10	4.00 (13)	0.016 (0.050)	83.8
	12	4.18 (14)	0.022 (0.117)	89.1
	14	3.85 (24)	0.024 (0.150)	79.8
	16	3.30 (35)	0.019 (0.200)	90.3
	18	3.60 (11)	0.021 (0.067)	89.2
	20	3.00 (18)	0.019 (0.100)	86.2

Table 2. $n = 20$, $d_{\max} = 100$

k	Density of DPC's (%)	Search Tree Nodes Avg. (Max.)	CPU SPARC330 Seconds Avg. (Max.)	Proportion of Strong Branchings (%)
10	0	1.43 (9)	0.006 (0.050)	100.0
	2	15.28 (239)	0.091 (1.250)	57.3
	4	8.15 (73)	0.043 (0.600)	68.6
	6	70.50 (981)	0.510 (7.683)	50.1
	8	34.85 (484)	0.276 (4.666)	42.2
	10	46.55 (459)	0.342 (4.700)	54.0
	12	43.10 (424)	0.332 (4.483)	54.1
	14	32.65 (324)	0.215 (2.550)	64.1
	16	52.80 (1273)	0.465 (12.450)	58.3
	18	13.20 (74)	0.085 (0.750)	54.0
	20	11.80 (126)	0.080 (1.183)	47.8
15	0	5.65 (24)	0.020 (0.100)	100.0
	2	15.18 (157)	0.081 (1.183)	80.1
	4	15.55 (185)	0.078 (1.217)	66.4
	6	13.68 (47)	0.074 (0.350)	63.9
	8	9.28 (49)	0.048 (0.417)	79.8
	10	7.83 (32)	0.040 (0.283)	77.7
	12	9.93 (64)	0.050 (0.400)	70.4
	14	7.75 (31)	0.043 (0.267)	83.9
	16	6.08 (21)	0.034 (0.150)	81.5
	18	5.10 (25)	0.031 (0.183)	80.0
	20	5.30 (18)	0.031 (0.100)	72.7
20	0	11.40 (154)	0.055 (1.067)	100.0
	2	6.43 (17)	0.023 (0.067)	94.1
	4	5.63 (15)	0.020 (0.050)	100.0
	6	5.30 (18)	0.024 (0.117)	92.6
	8	3.78 (9)	0.017 (0.033)	98.5
	10	4.05 (13)	0.016 (0.050)	85.5
	12	4.20 (15)	0.021 (0.117)	93.0
	14	3.95 (19)	0.024 (0.150)	89.3
	16	2.63 (15)	0.017 (0.100)	86.8
	18	3.63 (11)	0.019 (0.067)	89.2
	20	2.90 (15)	0.020 (0.067)	85.5

Table 3. $n = 50$, $d_{\max} = 50$

k	Density of DPC's (%)	Search Tree Nodes Avg. (Max.)	CPU SPARC330 Seconds Avg. (Max.)	Proportion of Strong Branchings (%)
10	0	1.03 (2)	0.014 (0.050)	100.0
	2	34.60 (163)	0.587 (3.117)	54.8
	4	36.48 (259)	0.646 (5.550)	47.9
	6	18.40 (58)	0.333 (1.067)	56.5
	8	13.95 (85)	0.291 (1.950)	54.2
	10	83.30 (1,752)	2.449 (50.398)	45.1
	12	15.43 (167)	0.411 (4.266)	43.4
	14	9.18 (49)	0.265 (1.217)	55.3
	16	6.85 (16)	0.222 (0.450)	62.5
	18	8.28 (112)	0.264 (2.983)	46.0
	20	5.03 (13)	0.193 (0.433)	67.0
15	0	32.90 (798)	0.980 (31.132)	100.0
	2	22.83 (141)	0.365 (3.300)	75.4
	4	11.55 (44)	0.188 (0.767)	77.6
	6	8.15 (42)	0.150 (0.800)	66.9
	8	7.10 (33)	0.156 (0.667)	70.5
	10	4.83 (12)	0.125 (0.317)	78.6
	12	5.25 (24)	0.152 (0.817)	69.4
	14	3.60 (12)	0.123 (0.317)	87.0
	16	4.88 (18)	0.172 (0.500)	64.4
	18	4.18 (23)	0.168 (1.017)	75.8
	20	4.05 (16)	0.156 (0.450)	75.0
20	0	18.58 (39)	0.238 (0.533)	100.0
	2	6.98 (19)	0.103 (0.300)	90.4
	4	5.28 (26)	0.090 (0.550)	81.9
	6	3.53 (13)	0.072 (0.300)	90.9
	8	4.05 (9)	0.095 (0.200)	81.5
	10	3.43 (10)	0.096 (0.217)	84.8
	12	2.98 (10)	0.100 (0.250)	93.2
	14	3.10 (11)	0.112 (0.317)	86.4
	16	2.83 (8)	0.112 (0.233)	81.1
	18	2.45 (10)	0.116 (0.267)	82.0
	20	2.50 (7)	0.119 (0.267)	87.2

Table 4. $n = 50$, $d_{\max} = 100$

k	Density of DPC's (%)	Search Tree Nodes Avg. (Max.)	CPU SPARC330 Seconds Avg. (Max.)	Proportion of Strong Branchings (%)
	0	1.05 (2)	0.019 (0.067)	100.0
	2	34.03 (123)	0.578 (2.383)	53.6
	4	39.15 (179)	0.694 (3.567)	45.2
	6	99.53 (1866)	2.814 (48.465)	45.3
10	8	36.23 (899)	0.860 (23.032)	43.7
	10	11.95 (47)	0.267 (1.033)	67.2
	12	8.95 (31)	0.241 (0.800)	66.3
	14	9.90 (32)	0.267 (0.833)	53.5
	16	12.33 (137)	0.440 (6.433)	53.3
	18	7.78 (31)	0.251 (0.933)	57.4
	20	5.60 (28)	0.216 (0.817)	71.3
	0	24.60 (45)	0.338 (0.617)	100.0
	2	17.60 (58)	0.260 (1.017)	73.4
	4	8.20 (23)	0.131 (0.417)	84.4
	6	6.63 (24)	0.134 (0.483)	72.5
15	8	5.95 (27)	0.134 (0.500)	69.2
	10	4.48 (18)	0.118 (0.417)	79.3
	12	3.60 (11)	0.112 (0.483)	84.1
	14	4.38 (12)	0.155 (0.667)	85.1
	16	3.75 (17)	0.134 (0.417)	82.4
	18	3.43 (15)	0.143 (0.467)	82.1
	20	3.83 (11)	0.163 (0.367)	74.7
	0	19.18 (35)	0.240 (0.450)	100.0
	2	9.18 (19)	0.123 (0.267)	90.8
	4	4.65 (14)	0.074 (0.233)	84.3
	6	3.55 (11)	0.071 (0.167)	82.8
20	8	3.98 (10)	0.094 (0.250)	92.1
	10	3.18 (11)	0.093 (0.217)	86.7
	12	3.25 (10)	0.101 (0.267)	83.3
	14	2.75 (8)	0.107 (0.233)	85.5
	16	2.95 (12)	0.122 (0.450)	78.7
	18	2.58 (8)	0.120 (0.283)	88.5
	20	2.45 (7)	0.116 (0.233)	84.4

Table 5. $n = 100$, $d_{\max} = 50$

k	Density of DPC's (%)	Search Tree Nodes Avg. (Max.)	CPU SPARC330 Seconds Avg. (Max.)	Proportion of Strong Branchings (%)
10	0	1.00 (1)	0.047 (0.050)	100.0
	2	48.65 (122)	2.755 (7.566)	44.4
	4	15.00 (58)	0.955 (3.583)	51.7
	6	11.08 (25)	0.876 (1.800)	54.6
	8	8.30 (23)	0.883 (2.200)	62.0
	10	7.90 (19)	0.955 (2.083)	59.1
	12	6.98 (29)	0.931 (3.667)	63.2
	14	6.15 (18)	0.907 (2.150)	61.6
	16	5.30 (19)	0.840 (2.217)	64.7
	18	4.23 (14)	0.795 (2.317)	72.4
	20	5.00 (17)	0.899 (2.417)	63.4
15	0	42.50 (93)	1.920 (4.200)	100.0
	2	12.58 (33)	0.608 (1.650)	76.8
	4	6.78 (35)	0.450 (2.250)	71.5
	6	4.58 (14)	0.431 (0.983)	78.9
	8	4.65 (15)	0.548 (1.667)	92.1
	10	4.43 (16)	0.615 (1.583)	70.0
	12	4.00 (12)	0.638 (1.283)	83.5
	14	3.76 (11)	0.655 (1.300)	73.7
	16	3.43 (9)	0.654 (1.200)	89.2
	18	3.13 (8)	0.671 (1.317)	79.0
	20	2.85 (12)	0.658 (1.650)	75.0
20	0	31.70 (55)	1.314 (2.333)	100.0
	2	5.00 (14)	0.248 (0.717)	92.6
	4	4.43 (19)	0.324 (1.083)	73.6
	6	4.18 (18)	0.398 (1.317)	76.5
	8	2.80 (7)	0.387 (0.867)	92.2
	10	2.68 (8)	0.457 (0.900)	75.0
	12	3.75 (17)	0.623 (2.083)	70.7
	14	2.73 (9)	0.525 (1.050)	87.5
	16	2.35 (5)	0.560 (1.217)	87.5
	18	2.35 (7)	0.560 (0.950)	88.4
	20	2.08 (6)	0.570 (1.017)	75.6

Table 6. $n = 100$, $d_{\max} = 100$

k	Density of DPC's (%)	Search Tree Nodes Avg. (Max.)	CPU SPARC330 Seconds Avg. (Max.)	Proportion of Strong Branchings (%)
10	0	1.00 (1)	0.049 (0.133)	100.0
	2	49.95 (129)	2.721 (7.950)	54.9
	4	13.53 (28)	0.868 (1.817)	60.8
	6	9.30 (30)	0.787 (2.283)	71.3
	8	8.43 (29)	0.908 (3.467)	54.3
	10	8.95 (22)	1.034 (2.433)	56.9
	12	6.63 (17)	0.905 (1.967)	63.9
	14	7.25 (19)	1.026 (2.533)	60.8
	16	5.58 (31)	0.985 (6.283)	61.5
	18	6.20 (12)	1.017 (1.833)	57.4
	20	5.93 (59)	1.109 (10.916)	63.8
15	0	55.20 (169)	2.593 (9.816)	100.0
	2	10.18 (29)	0.481 (1.350)	92.3
	4	7.85 (27)	0.539 (1.667)	76.9
	6	6.43 (20)	0.595 (2.083)	68.7
	8	4.75 (21)	0.580 (2.117)	70.1
	10	4.43 (16)	0.614 (1.667)	74.2
	12	4.03 (15)	0.662 (1.983)	71.4
	14	3.68 (15)	0.654 (1.867)	69.4
	16	3.25 (15)	0.665 (2.117)	78.5
	18	2.15 (7)	0.552 (1.183)	86.5
	20	3.20 (7)	0.688 (1.167)	87.3
20	0	31.73 (71)	1.313 (3.217)	100.0
	2	5.48 (19)	0.274 (1.033)	83.7
	4	4.60 (18)	0.325 (0.917)	75.9
	6	3.90 (18)	0.381 (1.333)	76.6
	8	2.83 (7)	0.395 (0.883)	90.4
	10	2.95 (10)	0.483 (1.100)	77.2
	12	4.08 (17)	0.663 (2.083)	73.2
	14	2.65 (9)	0.523 (1.050)	87.0
	16	2.55 (8)	0.589 (1.233)	83.0
	18	2.53 (7)	0.575 (0.967)	93.3
	20	2.08 (6)	0.562 (1.033)	79.5

SB Procedure, which we call SB3, differs from SB1, the original SB Procedure of [ABZ88], in several minor details besides the major differences of using the one machine problem with DPC's, and the number of reoptimization cycles. We limit the number of reoptimization cycles to at most six.

We also implemented a slightly different version of the new procedure which we call SB4. In SB3, upon completion of the local reoptimization procedure for a given set M_0 of machines already sequenced, the procedure is repeated after temporarily removing from the problem the last α non-critical machines (see [ABZ88] for details). In SB4 after applying this local reoptimization as described above, we apply the procedure a second time in the reverse order, i.e. by first temporarily removing from the problem the last α non-critical machines, and then applying again the reoptimization step with all the machines present. As it can be expected, SB4 is computationally more expensive than SB3 (roughly by a factor of 2), but it often finds a better solution.

The Procedures SB3 and SB4 were tested on the 40 problems generated by Lawrence [L84] and compared to SB1, as well as to the modified SB procedure of Dauzière-Peres and Lasserre [DL90], which also uses the one machine problem with DPC's, but solves it heuristically rather than to optimality. We also compare our procedures to Applegate and Cook's [AC91] implementation of the shifting bottleneck procedure. The results are shown in Tables 7-9, where m and n denote the number of machines and jobs, respectively, DL stands for the procedure of Dauzière-Lasserre, and AC for that of Applegate and Cook. The data for the DL procedure were kindly communicated to us by Dauzière-Peres [D91]. The data for the AC column were obtained by us with the code of Applegate and Cook [AC91], kindly provided to us by David Applegate. The optimal makespans are from [AC91].

In addition, we solved the problems FT1-FT3 generated by Fisher and Thompson [FT63], ABZ5-ABZ6 generated by Adams, Balas and Zawack for [ABZ88], and the problems ORB1-ORB5 generated by Applegate and Cook for [AC91]. We ran SB1, SB3, SB4 and AC on these problems, with the outcome shown in Table 10 and Table 11. Table 12 shows the

Table 7. $m = 5$

n	Problem	SB1 Make-span	SB1 CPU SPARC330 Seconds	SB3 Make-span	SB3 CPU SPARC330 Seconds	SB4 Make-span	SB4 CPU SPARC330 Seconds	DL Make-span	DL CPU SUN 4 Seconds	AC Make-span	AC CPU SPARC330 Seconds	Optimal Make-span
10	1	666+	0.39	666+	0.13	-	-	666+	0.5	666+	0.20	666
	2	720	0.39	667	0.90	667	1.43	684	1.6	684	0.23	655
	3	623	0.55	626	0.77	626	1.22	651	1.2	605	0.18	597
	4	597	0.48	593	1.00	593	1.72	598	1.5	603	0.30	590
	5	593+	0.42	593+	0.70	-	-	593+	0.3	593+	0.15	593
15	6	926+	0.42	926+	0.20	-	-	926+	1.3	926+	0.33	926
	7	890+	0.43	890+	0.25	-	-	890+	1.2	890+	0.40	890
	8	868	0.49	863+	0.50	-	-	863+	3.4	863+	0.45	863
	9	951+	0.43	951+	0.23	-	-	951+	0.8	951+	0.26	951
	10	959	0.40	958+	0.22	-	-	958+	0.7	958+	0.33	958
20	11	1222+	0.50	1222+	0.48	-	-	1222+	1.6	1222+	0.45	1222
	12	1039+	0.47	1039+	0.30	-	-	1039+	1.4	1039+	0.30	1039
	13	1150+	0.51	1150+	0.35	-	-	1150+	2.0	1150+	0.48	1150
	14	1292+	0.42	1292+	0.27	-	-	1292+	1.1	1292+	0.40	1292
	15	1207+	0.59	1207+	0.37	-	-	1207+	1.5	1207+	0.41	1207

+Optimal solution found by SB3; therefore SB4 was not run.

Table 8. $m = 10$

n	Problem	SB1 Make-span	SB1 CPU SPARC330 Seconds	SB3 Make-span	SB3 CPU SPARC330 Seconds	SB4 Make-span	SB4 CPU SPARC330 Seconds	DL Make-span	DL CPU SUN 4 Seconds	AC Make-span	AC CPU SPARC330 Seconds	Optimal Make-span
	16	1021	1.37	961	3.35	961	8.97	964	12.3	1076	1.20	945
	17	796	0.75	796	3.98	796	7.53	827	12.0	829	1.65	784
10	18	891	1.33	866	4.38	861	7.67	900	10.3	855	1.56	848
	19	875	0.95	902	4.33	878	8.80	856	13.2	863	2.08	842
	20	924	0.98	932	4.50	922	8.12	1011	11.0	918	2.10	902
	21	1172	2.13	1111	10.92	1071	19.88	1154	29.3	1128	3.23	1053*
	22	1040	1.71	954	11.85	954	21.52	955	31.5	968	3.63	927
15	23	1061	2.43	1032+	6.92	-	-	1090	33.4	1071	4.46	1032
	24	1000	2.38	976	10.82	976	19.82	999	28.7	1015	3.86	935
	25	1048	2.90	1012	13.00	1012	22.70	1012	28.2	1061	3.13	977
	26	1304	5.29	1239	14.80	1224	35.06	1261	65.2	1393	4.33	1218
	27	1325	4.78	1272	19.42	1272	37.90	1379	82.5	1353	6.05	1269*
20	28	1256	2.84	1245	17.72	1245	32.31	1240	62.7	1281	5.68	1216
	29	1294	5.11	1227	21.10	1227	39.00	1198	76.3	1233	6.50	1195*
	30	1403	4.09	1355+	4.52	-	-	1379	68.5	1355+	5.68	1355
	31	1784+	3.41	1784+	6.93	-	-	1784+	90.3	1784+	12.35	1784
	32	1850+	2.14	1850+	5.50	-	-	1850+	58.0	1850+	8.50	1850
30	33	1719+	2.31	1719+	9.20	-	-	1719+	54.3	1719+	6.46	1719
	34	1721+	4.25	1721+	13.87	-	-	1721+	160.1	1721+	9.63	1721
	35	1888+	3.47	1888+	5.73	-	-	1888+	43.1	1888+	8.56	1888

*Best value found by [AC91] - optimum unknown.

+Optimal solution found by SB3; therefore SB4 was not run.

Table 9. $m = 15$

n	Problem	SB1 Make-span	SB1 CPU SPARC330 Seconds	SB3 Make-span	SB3 CPU SPARC330 Seconds	SB4 Make-span	SB4 CPU SPARC330 Seconds	DL Make-span	DL CPU SUN 4 Seconds	AC Make-span	AC CPU SPARC330 Seconds	Optimal Make-span
15	36	1351	4.87	1319	27.80	1319	55.83	1372	122.4	1326	10.08	1268
	37	1485	6.21	1425	26.25	1425	53.28	1490	101.6	1471	6.71	1397
	38	1280	4.41	1318	29.63	1294	59.26	1298	128.0	1307	10.73	1209*
	39	1321	6.30	1278	25.40	1278	50.63	1306	102.8	1301	12.76	1233
	40	1326	6.87	1266	26.23	1262	52.41	1305	112.9	1347	12.74	1222

*Best value found by [AC91] - optimum unknown.

Table 10. $m = 10$

n	Problem	SB1 Make-span	SB1 CPU SPARC330 Seconds	SB3 Make-span	SB3 CPU SPARC330 span	SB4 Make-Seconds	SB4 CPU SPARC330 Seconds	AC Make-span	AC CPU SPARC330 Seconds	Optimal Make-span
10	ABZ5	1306	3.14	1258	3.83	1258	8.00	1270	1.58	1234
	ABZ6	962	3.16	960	4.45	960	8.40	952	2.00	943
	ORB1	1152	3.31	1123	5.18	1121	12.13	1176	1.85	1059
	ORB2	924	3.46	933	3.90	933	8.15	927	2.10	888
	ORB3	1147	3.14	1083	4.70	1054	9.58	1090	1.98	1005
	ORB4	1052	3.40	1086	5.10	1056	9.39	1065	1.86	1005
	ORB5	931	3.28	899	4.53	899	9.06	939	0.78	887

*Best value found by [AC91] - optimum unknown.

Table 11.

n	m	Prob-	SB1		SB3		SB4		DL		AC		Optimal
	lem		Make-span	CPU SPARC330 Seconds	Make-span	CPU SPARC330 Seconds	Make-span	CPU SPARC330 Seconds	Make-span	CPU SUN 4 Seconds	Make-span	CPU SPARC330 Seconds	Make-span
20	5	FT1	1290	0.75	1199	7.28	1199	13.22	1216	8	1228	1.90	1165
6	6	FT2	55	0.41	55	0.47	55	0.87	55	1	59	0.21	55
10	10	FT3	1015	1.11	981	5.82	940	11.17	950	15	952	1.90	930

Table 12. Number of times each procedure found best and strictly better schedules.

SB1		SB4		DL*		AC	
Best	Strictly Better	Best	Strictly Better	Best	Strictly Better	Best	Strictly Better
20	3	40	19	22	3	22	4

*Excluding problems ABZ5-ORB5

Table 13.

SB1		SB4		DL*		AC	
Best	Strictly Better	Best	Strictly Better	Best	Strictly Better	Best	Strictly Better
5	3	23	19	5	3	5	4

*Excluding problems ABZ5-ORB5

number of times each procedure ranked first, counting and not counting ties. Since some problem sets are easy for all three procedures, in Table 13 we show the same data for the problems 2-4, 16-30, 36-40, ABZ5-ORB5, FT1-FT3, that remain after eliminating the easy sets.

We conclude that the Modified Shifting Bottleneck Procedure (SB3), and its variant (SB4) find consistently better schedules than SB1, DL or AC, at a computational cost somewhat higher than that of SB1 and AC, but not higher than that of DL.

References

[ABZ88] J. Adams, E. Balas, and D. Zawack "The Shifting Bottleneck Procedure for Job Shop Scheduling." *Management Science, 34,* No. 3, March 1988. pp. 391-401.

[AC91] D. Applegate and W. Cook, "A Computational Study of Job Shop Scheduling." *ORSA Journal on Computing, 3,* No. 2, Spring 1991.

[BV91] E. Balas and A. Vazacopoulos, "Improvements of the Shifting Bottleneck Procedure for Job Shop Scheduling Problems." ORSA/TIMS joint National Meeting in Anaheim, November 3-6, 1991.

[C82] J. Carlier, "The One Machine Sequencing Problem." *European Journal of Operational Research, 11,* 1982, pp. 42-47.

[DL90] S. Dauzière-Peres and J.B. Lasserre, "A Modified Shifting Bottleneck Procedure." *LAAS, Rapport LAAS N, 90106,* April 1990.

[D91] S. Dauzière-Peres, private communication.

[GJ79] M.R. Garey and D.S. Johnson, *Computers and Intractability,* W.H. Freeman and Co., San Francisco, 1979.

[FT63] S. Fisher and G.L. Thompson, "Probabilistic Learning Combinations of Local Job Shop Scheduling Rules." J.F. Muth and G.L. Thompson (editors), *Industrial Scheduling.* Prentice-Hall, Englewood Cliffs, NJ, 1963.

[J55] J.R. Jackson, "Scheduling a Production Line to Minimize Maximum Tardiness." *Management Science Research Project, 43,* University of California, Los Angeles, 1955.

[L84] S. Lawrence, Supplement to "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques." GSIA, Carnegie Mellon University, Pittsburgh, PA, October 1984.

[P80] C.N. Potts, "Analysis of a Heuristic for One Machine Sequencing With Release Dates and Delivery Times." *Operations Research, 28(6),* 1980, pp. 1436-1441.

[T88] F. Tiozzo, "Building a Decision Support System for Operation Scheduling in a Large Industrial Department: A Preliminary Algorithmic Study." University of Udine.

REPORT DOCUMENTATION PAGE

REPORT NUMBER	2. GOVT ACCESSION NO	READ INSTRUCTIONS BEFORE COMPLETING FORM 3. RECIPIENT'S CATALOG NUMBER
MSRR-589		

TITLE (and Subtitle)

ONE MACHINE SCHEDULING WITH DELAYED PRECEDENCE CONSTRAINTS

5. TYPE OF REPORT & PERIOD COVERED: Technical Report, December 1992

6. PERFORMING ORG. REPORT NUMBER

AUTHOR(S)

Egon Balas
Jan Karel Lenstra
Alkis Vazacopoulos

8. CONTRACT OR GRANT NUMBER(S): DDM-8901495

PERFORMING ORGANIZATION NAME AND ADDRESS

Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213-3890

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

CONTROLLING OFFICE NAME AND ADDRESS

Personnel and Training Research Programs
Office of Naval Research (Code 434)
Arlington, VA 22217

12. REPORT DATE: December 1992

13. NUMBER OF PAGES: 34

MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)

15. SECURITY CLASS (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

SUPPLEMENTARY NOTES

KEY WORDS (Continue on reverse side if necessary and identify by block number)

Scheduling
Delayed Precedence Constraints
Shifting Bottleneck

ABSTRACT (Continue on reverse side if necessary and identify by block number)

We study the one machine scheduling problem with release and delivery times and the minimum makespan objective, in the presence of constraints that for certain pairs of jobs require a delay between the completion of the first job and the start of the second (delayed precedence constraints). This problem arises naturally in the context of the Shifting Bottleneck Procedure for the general job shop scheduling problem, as a relaxation of the latter, tighter than the standard one machine relaxation. The paper first highlights the difference

between the two relaxations through some relevant complexity results. Then it introduces a modified Longest Tail Heuristic whose analysis identifies those situations that permit efficient branching. As a result, an optimization algorithm is developed whose performance is comparable to that of the best algorithms for the standard one machine problem. Embedding this algorithm into a modified version of the Shifting Bottleneck Procedure that uses the tighter one machine relaxation discussed here results in a considerable overall improvement in performance on all classes of job shop scheduling problems.