

UNIVERSITY OF
ILLINOIS LIBRARY
AT URBANA-CHAMPAIGN
BOOKSTACKS

330
B385
1992:131 COPY 2

STX

A Single-Machine Scheduling Model with Fixed-Interval Deliveries

The Library of the
JUN 8 1992

University of Illinois
of Urbana-Champaign

Suresh Chand
Krannert School of Management
Purdue University

Dilip Chhajed
Department of Business Administration
University of Illinois

Rodney Traub
Krannert School of Management
Purdue University

BEBR

FACULTY WORKING PAPER NO. 92-0131

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign


May 1992

A Single-Machine Scheduling Model with Fixed-Interval Deliveries

Suresh Chand
Purdue University

Dilip Chhajed
University of Illinois at Urbana-Champaign

Rodney Traub
Purdue University



Digitized by the Internet Archive
in 2012 with funding from
University of Illinois Urbana-Champaign

<http://www.archive.org/details/singlemachinesch92131chan>

**A Single-Machine Scheduling Model
with Fixed-Interval Deliveries**

By

Suresh Chand
Krannert School of Management
Purdue University
West Lafayette, IN 47907

Dilip Chhajed
Department of Business Administration
University of Illinois at Urbana-Champaign
Champaign, IL 61820

and

Rodney Traub
Krannert School of Management
Purdue University
West Lafayette, IN 47907

March 9, 1992

Abstract

This paper considers an environment where a single-machine job shop needs to assign delivery dates to several orders and find a feasible sequence. Tardy jobs are not allowed. The delivery dates must be at prespecified fixed intervals. The objective is to minimize the due date penalty and the cost of earliness.

Subject Areas: Dynamic Programming, Production/Operations Management, Scheduling

1. Introduction

Consider a single-machine shop with a list of customer orders. The shop needs to specify delivery dates to customers and find a schedule such that the delivery promises are met without any tardy jobs. The delivery dates are assumed to be multiples of a fixed interval. One example of such a situation is when there is a delivery system in place that delivers the completed jobs from the shop to customers at fixed intervals. Each customer would like to receive its order as soon as possible. For each job, there is a lead time penalty proportional to its delivery date that reflects the discount the customer may have to be offered for waiting to receive his order. The shop incurs a holding cost for a job for the time between its completion in the shop and delivery to the customer.

The shop has a service policy of delivering to its customers on a first-come first-served basis. Thus if customer A's order was received before customer B's order, then the delivery date to customer A cannot be after the delivery date to customer B.

The problem of scheduling a given number of jobs and determining their due dates to minimize the sum of earliness, tardiness, and due date costs, where the due date cost of an order is proportional to its assigned due date, is well studied. Panwalkar, Smith, and Seidmann (1982) consider the case of a single due date and Chand and Chhajed (1992) consider the case of multiple due dates. Models in which there is a single fixed due date and the objective is to find a schedule of minimum total earliness and/or tardiness costs are also considered by Hall, Kubiak, and Sethi (1989), Kanet (1981), Bagchi, Sullivan, and Chang (1986), Garey, Tarjan, and Wilfong (1988), and De, Ghosh, and Wells (1991). Our problem is different from these because the due dates in our model are at fixed intervals.

In the problem that we consider, the due dates cannot be violated and so these are deadlines as in Ahmadi and Bagchi (1986), and Chand and Schneeberger (1988). However, unlike these papers, we consider the due dates as decision variables. Matsuo (1988) considers the problem with fixed shipping times and minimizes the sum of overtime and tardiness costs. Cheng and Kahlbacher (1991) consider the problem with fixed delivery cost and earliness cost. They do not

restrict the deliveries at fixed intervals and show that the problem is NP-hard. Chhajed (1991) has considered a similar problem but *without* the first-come first-served policy. He showed the problem to be NP-complete and gave bounds on the optimal solution value in the case when there are two delivery dates. Garey, Tarjan, and Wilfong (1988) consider the problem of scheduling jobs with a given ordering with preferred starting times. They give an $O(n \log n)$ algorithm to minimize the sum of the absolute deviation from the preferred starting times.

For the problem considered in this paper, we provide a dynamic programming algorithm to find an optimal sequence and delivery dates. Later, we provide several dominance results to reduce the computational time taken by the dynamic program. Our computational results indicate that the dominance results reduce the computational requirements by an order of magnitude. The rest of the paper is organized as follows. In section 2, we introduce the notation and formulate the problem as a dynamic program. In section 3, we develop dominance results to reduce the time taken by the dynamic program. In section 4, we discuss the computational requirements and provide an example. Section 5 provides a computational study to examine the savings realized by the use of our dominance results in the dynamic programming algorithm. Concluding remarks are in section 6.

2. Model Formulation and Dynamic Programming Algorithm

The following problem is considered: Jobs $\{1,2,\dots,N\}$ are available at the beginning of Period 1 for processing on a single machine. The jobs are numbered in the order of their arrival. (Thus, for $0 < i < j \leq N$, the arrival time of Job i is less than or equal to the arrival time of Job j .) The shop needs to assign a due date to each job and determine a schedule for processing these N jobs on the single machine to optimize a certain measure of performance. The following constraints are observed:

- (1) Each job is to be delivered exactly at its due date.
- (2) Deliveries of completed jobs to customers occur at some predetermined fixed intervals.
- (3) Jobs must be delivered on a first-come, first-served basis. Thus, if Jobs i and j are delivered at the same time, then Jobs $\{i + 1, i + 2, \dots, j - 1\}$ must also be delivered at that time. However, any processing order can be used among the jobs scheduled in an interval.

The performance measure considered in this paper is assumed to include the earliness cost and the lead time penalty. Earliness cost for a job is incurred from the time the job completes processing until it is delivered. The shop incurs this cost because it has to hold the job between its completion and delivery. Lead time penalty for a job is incurred for the time taken to deliver the job to the customer. It represents the discount the customer may have to be offered for the delay in meeting his order.

We now define the necessary notation and develop expressions for different costs. Throughout the paper, we assume that the set notations $\{\ell, \ell + 1, \dots, \ell + n\}$ and $\langle \ell, \ell + n \rangle$ are equivalent.

Let P_i denote the processing time for Job i and let W_i denote the earliness cost per unit time for Job i . It is easy to see that the cost of earliness for jobs scheduled in an interval is minimized by arranging them in the order of increasing $\frac{W}{P}$ ratios. Let $E[K]$ denote the minimum cost of

earliness for scheduling jobs in the set K in an interval. Then, for $K = \langle \ell+1, \ell+k \rangle$, it is possible to see that

$$E[K] = \Sigma [\min \{W_i P_j, W_j P_i\} \mid i \in \langle \ell + 1, \ell + k - 1 \rangle, j \in \langle i + 1, \ell + k \rangle]. \quad (1)$$

In this formulation, it is assumed that the last job processed in an interval completes at the end of the interval. Also, the interval length is assumed to be larger than, or equal to, the processing time of any job.

If π_i denotes the per period lead time penalty for Job i , then the lead time penalty of delivering Job i at the end of the m^{th} period (or the m^{th} interval), is given by $m\pi_i$.

We are now ready to develop a Dynamic Programming algorithm to solve this problem. To develop this algorithm, we need the following additional notation. By n -job problem, we mean the problem when only the first n jobs are to be scheduled on the single machine.

$C(n)$ = the minimum cost for the n -job problem

$C(n,m)$ = the minimum cost for the n -job problem subject to the constraint that there are m deliveries (infinite if infeasible)

$C_k(n,m)$ = the minimum cost for the n -job problem with m deliveries and k jobs in the last interval (infinite if infeasible)

It is now possible to develop the following recursions:

$$C(n,m) = \min_k \{ C_k(n,m) \mid k \in \langle 1, n-m+1 \rangle \}$$

$$C(n) = \min_m \{ C(n,m) \mid m \in \langle 1, n \rangle \}.$$

The following Dynamic Programming algorithm solves the problem. For this algorithm, we first need to initialize

$$C(n,1) = E[\langle 1, n \rangle] + \sum_{i=1}^n \pi_i \quad \text{for } n \in \langle 1, N \rangle.$$

We set $C(n,1)$ to infinity if $\sum_{i=1}^n p_i > \text{the interval length}$.

Steps of the Dynamic Programming Algorithm:

(Loop 1) For $n = 2, 3, \dots, N$

(Loop 2) For $m = n, n-1, \dots, 2$

(Loop 3) For $k = 1, 2, \dots, n-m+1$

$$C_k(n,m) = C(n-k, m-1) + E[<n-k+1, n>] + \sum_{i=n-k+1}^n \pi_i m \quad (\text{End of Loop 3})$$

$$C(n,m) = \min_k \{C_k(n,m) \mid k \in <1, n-m+1>\} \quad (\text{End of Loop 2})$$

$$C(n) = \min_m \{C(n,m) \mid m \in <1, n>\} \quad (\text{End of Loop 1})$$

This algorithm is forward in n (the number of jobs) and backward in m (the number of deliveries). That is, we solve the $C(n)$ - problem for increasing n (starting from $n = 2$ up to $n = N$) and the $C(n,m)$ - problem for decreasing m (starting from $m = n$ up to $n = 1$).

In the next section, we develop several dominance properties to reduce the computations needed to solve the problem using the Dynamic Programming algorithm. Specifically, we develop results to reduce the search region for m for a given n (in Loop 2) and the search region for k for a given (n,m) pair (in Loop 3).

3. Computation Reduction Results

In this section, we develop several dominance results to reduce the computations in the application of the Dynamic Programming Algorithm. To state these results, we need the following additional notation.

$m^*(n)$ = number of deliveries in a $C(n)$ -solution

$NJL(n,m)$ = number of jobs assigned to the last delivery period in a $C(n,m)$ -solution

Theorem 1 below gives an upper bound on the number of deliveries in the $C(n+1)$ - solution.

THEOREM 1: There is an optimal solution for the $C(n+1)$ - problem such that

$$m^*(n+1) \leq m^*(n) + 1.$$

Proof: It is easy to see that the $(n+1)$ - job problem with $m^*(n)+1$ deliveries has an optimal solution such that the Job $(n+1)$ is assigned to the last interval and the remaining n jobs are assigned to the first $m^*(n)$ intervals with the total cost equal to

$$C(n+1, m^*(n) + 1) = C(n) + \pi_{n+1} \cdot (m^*(n) + 1).$$

To prove this theorem, it is sufficient to show that

$$C(n+1, m^*(n) + \varrho) \geq C(n) + \pi_{n+1} \cdot (m^*(n) + \varrho) \text{ for } \varrho \geq 1.$$

This follows from the observation that the cost of the first n jobs in the $C(n+1, m^*(n)+\varrho)$ - solution is at least as large as $C(n)$, and the cost of the last job is at least as large as $\pi_{n+1} \cdot (m^*(n) + \varrho)$. ■

As a result of this theorem, the search for m in Loop 2 of the Dynamic Programming Algorithm can be started from $m = m^*(n-1)+1$ instead of $m = n$ when we go from $n-1$ jobs to n jobs. Also, for the $C(n,m)$ -problem with $m = m^*(n-1)+1$, there is an optimal solution with $NJL(n,m)=1$; that is, one job is assigned to the last due date and the remaining $n-1$ jobs are assigned to the first $m^*(n-1)$ due dates.

The next theorem gives an upper bound on the number of jobs assigned to the last interval in the $C(n,m)$ - solution. To prove this theorem, it is useful to define $\theta(K\Lambda L)$ as follows:

$$\theta(K\Lambda L) = E(K+L) - E(K) - E(L)$$

Here, K and L are two mutually exclusive sets of jobs. $E(K+L)$ is the cost of earliness if all jobs in K and L are processed in the same interval. $E(K)$ and $E(L)$ are similarly defined. $\theta(K\Lambda L)$ denotes the increase in the earliness cost if jobs in K and L are processed in the same interval, instead of in two separate intervals. It is easy to see that

$$\theta(K\Lambda L) = \sum_{i \in K} \sum_{j \in L} \min \{W_i p_j, W_j p_i\}.$$

Also, for $\bar{K} \geq K$, and $\bar{L} \geq L$, we have

$$\theta(\bar{K} \wedge \bar{L}) \geq \theta(K \wedge L).$$

We are now ready to prove the following theorem.

THEOREM 2: The $C(n,m)$ - Problem has an optimal solution such that

$$NJL(n,m) \leq NJL(n-x,m) + x \quad \text{for every } x \in \langle 1, n-m \rangle.$$

Proof: Assume to the contrary that the optimal solution to the $C(n,m)$ - problem has $k + NJL(n-x,m) + x$ jobs in the last interval for some $x \in \langle 1, n-m \rangle$ and k a positive integer. We let $K(\ell)$ denote the set of the last ℓ jobs in $\langle 1, n-x \rangle$ and let $L(\ell)$ denote the set of the last ℓ jobs in $\langle 1, n \rangle$. For convenience of presentation, we denote $NJL(n-x,m)$ by $(n-x)_m$.

Given that the $C(n-x,m)$ - solution has $(n-x)_m$ jobs in the last interval, and the $C(n,m)$ - solution has $k + (n-x)_m + x$ jobs in the last interval, then the $C(n,m)$ - problem must have a feasible solution with $(n-x)_m + x$ jobs in the last interval. One such feasible solution can be constructed from the $C(n-x,m)$ - solution simply by adding the x jobs in $L(x)$ to the last interval.

The cost of this feasible solution is equal to

$$\begin{aligned} A &= C(n-x,m) + E[K((n-x)_m) + L(x)] - E[K((n-x)_m)] + \sum_{i \in \underline{L}(x)} \pi_i \cdot m \\ &= C(n-x,m) + \theta[K((n-x)_m) \wedge L(x)] + E[L(x)] + \sum_{i \in \underline{L}(x)} \pi_i \cdot m. \end{aligned}$$

Now consider the $C(n,m)$ -solution with $k + (n-x)_m + x$ jobs in the last interval. If we pull out each of the x jobs in $L(x)$ from the last interval, and adjust the remaining $k + (n-x)_m$ jobs such that their earliness cost is minimized, then the remaining schedule is feasible for the $C(n-x,m)$ - problem. It is now possible to see that the cost of the $C(n,m)$ - problem with $k + (n-x)_m + x$ jobs in the last interval is equal to

$$B \geq C(n-x,m) + \theta[K(k+(n-x)_m) \wedge L(x)] + E[L(x)] + \sum_{i \in \underline{L}(x)} \pi_i \cdot m.$$

It is easy to see that $B \geq A$. This completes the proof. ■

The search region for k in Loop 3 of the Dynamic Programming algorithm can be reduced as a result of Theorems 1 and 2. For $m = m^*(n-1) + 1$, from the proof of Theorem 1, the search region for k is $\{1\}$ instead of $\langle 1, n-m+1 \rangle$. For $m < m^*(n-1) + 1$, the search region for k is $\langle 1, M \rangle$, where

$$M = \min_x \{NJL(n-x, m) \mid x \in \langle 1, n-m \rangle\}.$$

The next theorem further reduces the search region for k in Loop 3 of the Dynamic Programming algorithm by developing a lower bound on k .

THEOREM 3: The $C(n, m-1)$ -problem has an optimal solution such that $NJL(n, m-1) \geq NJL(n, m)$.

Proof: Let q_j denote the number of jobs in the j^{th} interval in the $C(n, m)$ - solution. Note that $q_m = NJL(n, m)$. We define ℓ_j similarly for the $C(n, m-1)$ solution. Then $\ell_{m-1} = NJL(n, m-1)$. Contrary to the theorem, assume that $\ell_{m-1} < q_m$.

$$\text{Note that } \sum_{i=1}^{m-1} \ell_i = n > \sum_{i=2}^m q_i. \text{ Let } k^* \text{ denote the largest integer such that } \sum_{i=k^*}^{m-1} \ell_i \geq \sum_{i=k^*+1}^m q_i.$$

Clearly, $k^* < m-1$ because $\ell_{m-1} < q_m$. Let us assume $k^* = m-2$ to simplify the presentation.

With $k^* = m-2$, we have $\ell_{m-2} + \ell_{m-1} \geq q_{m-1} + q_m$.

Consider a $C(n, m)$ - solution with the constraint that the m^{th} interval has ℓ_{m-1} jobs and the $(m-1)^{\text{th}}$ interval has $(q_{m-1} + q_m - \ell_{m-1})$ jobs. Let $F(n, m)$ denote the minimum cost for this constrained $C(n, m)$ - problem. Note that transferring $(q_m - \ell_{m-1})$ jobs from the $(m-1)^{\text{th}}$ interval to the m^{th} interval in the $F(n, m)$ - solution gives the $C(n, m)$ - solution; the reduction in cost is equal to

$F(n, m) - C(n, m) =$ saving in the earliness cost in the $(m-1)^{\text{th}}$ interval

- increase in the earliness cost in the m^{th} interval

- increase in the lead time penalty for the jobs that are transferred

$= A - B - C$

It is feasible to make an identical transfer of the same jobs in the $C(n, m-1)$ - solution also from the $(m-2)^{\text{th}}$ interval to the $(m-1)^{\text{th}}$ interval. Let A' denote the saving in the earliness cost in the $(m-1)^{\text{th}}$ interval due to this transfer, B' the increase in the earliness cost in the $(m-1)^{\text{th}}$ interval, and C' the increase in the lead time penalty for the jobs that are transferred. It is easy to see that $B' = B$ and $C' = C$. Also, $A' \geq A$ because the same set of jobs is transferred from an interval consisting of more jobs in A' than in A . (This interval consists of the last ρ_{m-2} jobs in the set $\langle 1, n - \rho_{m-1} \rangle$ in A' compared to the last $q_m + q_{m-1} - \rho_{m-1} (\leq \rho_{m-2})$ in the set $\langle 1, n - \rho_{m-1} \rangle$ in A .) Thus, there is a non-negative saving due to this transfer. This implies that, for the case of $k^* = m-2$, an alternate solution with a reduced or same cost can be constructed such that $\rho_{m-1} = q_m$.

The proof can be easily extended for the case when $k^* < m-2$. In this case, we define the $F(n, m)$ - problem by putting the following constraints on the $C(n, m)$ - problem: The i^{th} interval, for $i \in \langle k^* + 2, m \rangle$, has ρ_{i-1} jobs and the $(k^* + 1)^{\text{th}}$ interval has $\sum_{j=k^*+1}^m q_j - \sum_{j=k^*+1}^{m-1} \rho_j$ jobs. It is possible to find sets of jobs that need to be transferred from the $(k^* + 1)^{\text{th}}$ interval to the $(k^* + 2)^{\text{th}}$ interval, from $(k^* + 2)^{\text{th}}$ interval to the $(k^* + 3)^{\text{th}}$ interval, and so on, to go from the $F(n, m)$ - solution to the $C(n, m)$ - solution. This transfer reduces the cost because $F(n, m) \geq C(n, m)$. An identical transfer can be made in the assumed $C(n, m-1)$ solution also leading to q_m jobs in the last interval, with the saving $\geq F(n, m) - C(n, m) \geq 0$. This completes the proof. ■

With this result, for given (n, m) , the search region for k in Loop 3 of the Dynamic Programming Algorithm starts from $k = \text{NJL}(n, m+1)$ instead of $k = 1$.

We now discuss the computational requirements of the Dynamic Programming Algorithm and use a numerical example to illustrate the dominance results.

4. Computational Requirements and a Numerical Example

In this section, we first discuss the computational requirements and then present an example. For the full dynamic program, the total number of states (combinations of different

n,m,k values) analyzed is $\frac{N^3 + 5N}{6}$ where N is the number of jobs. Thus, the state space for the full algorithm is of the order $O(N^3)$. As explained below, it requires $O(N^4)$ calculations to analyze these states.

These required calculations, and therefore the necessary computer time, has been reduced by taking advantage of the first-come first-served structure of the problem. Consider the x jobs which are assigned to the last interval at some stage of an n-job problem. As loop 3 of the algorithm continues, job x+1 is added to the last interval. From (1), the earliness cost associated with these x+1 jobs can be found as follows:

$$\text{Earliness cost for } x+1 \text{ jobs} = \text{Earliness cost for } x \text{ jobs} + \sum_{i=1}^x \min(W_i p_{x+1}, W_{x+1} p_i).$$

Thus by carrying the earliness cost for the first x jobs forward, only the incremental cost of adding the (x+1)th job to the last interval needs to be determined; this requires $O(x)$ computations. From the above discussion, we can conclude that $O(N)$ computations are required for each state. Thus with $O(N^3)$ states, our dynamic programming algorithm (without any computation reduction results) requires $O(N^4)$ computations to solve the problem.

Next, we present an example. Five jobs, whose processing times in order of arrival are 9, 18, 15, 16, and 2 are received. For all jobs, the lead time penalty cost is \$10 per period and the earliness cost is \$1 per period of earliness. Since there are 5 jobs, 25 states must be analyzed in the full version of the algorithm. Table 1 shows these 25 states in the sequence that the dynamic program checks them. State (1,1) is included for completeness. The cost for each of these states is given in column 5. The asterisks in the fifth column of Table 1 represent optimal solutions for each value of n. The dynamic program provides solutions to each of these subproblems as well.

The third loop of the dynamic program calculates costs for each successive state. For example, state 19 has 5 jobs assigned to 3 intervals with 2 jobs being assigned to the third interval. This leaves 3 jobs to be assigned to the first two intervals. The optimal value for 3 jobs and 2 intervals is determined by taking the minimum over states 5 and 6 and is found to be 49. The earliness cost of assigning the jobs 4 and 5 to an interval is 2. Finally, the lead time penalty of

assigning these two jobs to the third interval is 60. These are the three terms in Loop 3 of the dynamic program and result in the total cost for state 19 of 111.

When the efficiency theorems are utilized the state space can be reduced. For this example, 11 of the 25 states are eliminated when Theorems 1, 2, and 3 are utilized. As indicated by the last column of Table 1, Theorem 1 eliminates states 4, 6, 8, 10, 15 and 17, Theorem 2 eliminates states 13, 20, 23, and 24, and Theorem 3 eliminates state 21.

To demonstrate how efficiency theorems work, consider first how Theorem 1 eliminates states 15 and 17. Since the optimal solution for 4 jobs uses 3 intervals, the most states which must be considered for 5 jobs is 4 intervals, and if 4 intervals are used, only one job will be assigned to the fourth interval. Therefore, state 15, which uses 5 intervals, and state 17 which has two jobs assigned to the fourth interval, need not be considered. Theorem 2 is used to eliminate state 20. Since the optimal solution for 4 jobs and 3 intervals has one job assigned to the third interval, there is no need to consider state 20 which consists of 5 jobs but has 3 jobs assigned to the third interval. Finally, Theorem 3 can be used to eliminate state 21. Since the optimal solution for 5 jobs and 3 intervals has two jobs assigned to the third interval, there is no need to consider state 21 which has one job assigned to the second interval.

5. Computational Results

To analyze the effectiveness of the dominance results, numerous simulations were run. The full dynamic programming algorithm and a version using all efficiency theorems were tested on randomly generated problems. The number of states analyzed by the dynamic program as well as the time required to solve the problems were recorded to determine the effectiveness of the theorems. The simple example above illustrates the appropriateness of using the number of states as a measure of effectiveness. Also, the computer time required to solve each problem is presented as a second measure of the theorems' effectiveness.

The test data was generated as follows. Processing times were generated from a uniform discrete distribution with a range from 1 to 20. Problems of size 10, 20, ..., 150 jobs were tested.

The lead time penalty cost was set at 100 for all jobs. The earliness costs were set at 5, 10, 20, 40, and 80. Note that a problem with an earliness cost of 2 and a lead time interval cost of 10 is equivalent to a problem with an earliness cost of 20 and a lead time interval cost of 100. That is, the ratio of costs, not the absolute value of costs, is what affects the assignment of jobs to intervals. The range of earliness costs used should help us examine the performance of our dominance results under a variety of circumstances. With a large earliness cost, relatively few jobs will be assigned to an interval whereas the use of a small earliness cost results in many jobs being assigned to an interval.

Table 2 gives the number of states checked for the full dynamic program as well as the efficient version. Also, the fraction of states which the efficient version checks is given. The table includes results for all replications, and is also broken down by problem size and by earliness cost.

Note that as problem size increases, the fraction of states checked tends to decrease. In absolute terms, this implies that our algorithm is relatively more efficient for large problems which is a desirable quality. As earliness costs increase, the fraction of states checked also increases. This is due to the additional intervals which must be considered. Regression analysis showed that the number of states for the efficient version of the algorithm is an order of magnitude lower than for the full version.

The CPU time to solve the problems is presented in Table 3. Here too, the improvement that the theorems have on the solution can be observed. The table gives the average CPU time required to solve a problem of a particular size with both the full algorithm and the efficient version. In each case, 50 problems consisting of 10 problems for each value of earliness cost were solved and the total time to solve these problems was recorded. This total time was then divided by 50 to determine the average time required to solve problems of a particular size. As can be seen in Table 3, the savings in number of states examined translates into a corresponding savings in computational time. Again, a regression analysis showed that the calculations for the efficient version are reduced by an order of magnitude. It is clear from both Tables 2 and 3 that our theorems lead to a significant reduction in the effort required to solve the problem.

6. Concluding Remarks

This paper considered a single-machine scheduling problem where the shop has to assign delivery dates to jobs and find a feasible schedule to minimize the lead time penalty and the cost of earliness. A dynamic programming algorithm and several dominance results were developed for the problem. Our computational results show that the dominance results in the paper can reduce the computational requirements by an order of magnitude.

In our model, we assumed that the length of the fixed interval is prespecified. Note that this interval length determines the frequency of delivery. If the annual delivery cost is increasing with the delivery frequency, then the length of the fixed interval could also become a decision variable. The results in this paper should be helpful in solving this more general problem.

TABLE 1

State	n	m	k	$C_k(n,m)$	State Not Calculated Due to Theorem #
1	1	1	1	10 *	
2	2	2	1	30	
3	2	1	2	29 *	
4	3	3	1	60	1
5	3	2	1	49 *	
6	3	2	2	65	1
7	3	1	3	63	
8	4	4	1	100	1
9	4	3	1	79 *	
10	4	3	2	105	1
11	4	2	1	83	
12	4	2	2	84	
13	4	2	3	116	2
14	4	1	4	113	
15	5	5	1	150	1
16	5	4	1	119	
17	5	4	2	142	1
18	5	3	1	113	
19	5	3	2	111	
20	5	3	3	139	2
21	5	2	1	133	3
22	5	2	2	105 *	
23	5	2	3	108	2
24	5	2	4	142	2
25	5	1	5	131	

* Designates the optimal solution for the n job problem

TABLE 2

Average percent of states checked				
	Number of Simulation runs	Average States Checked in Full Program	Average States Checked with Theorems	Average Percent of States Checked
Overall	750	160066.7	3602.4	6.8
By Problem Size				
10	50	175.0	58.1	33.2
20	50	1350.0	228.4	16.9
30	50	4525.0	479.2	10.6
40	50	10700.0	815.6	7.6
50	50	20875.0	1231.6	5.9
60	50	36050.0	1720.2	4.8
70	50	57225.0	2293.1	4.0
80	50	85400.0	2930.0	3.4
90	50	121575.0	3657.4	3.0
100	50	166750.0	4431.6	2.7
110	50	221925.0	5281.8	2.4
120	50	288100.0	6240.4	2.2
130	50	366275.0	7149.7	2.0
140	50	457450.0	8190.3	1.8
150	50	562625.0	9328.9	1.7
By Earliness cost				
5	150	160066.7	2633.8	4.3
10	150	160066.7	3006.7	5.4
20	150	160066.7	3465.3	6.7
40	150	160066.7	4064.2	8.0
80	150	160066.7	4842.2	9.6

TABLE 3

Average seconds required to solve a problem				
Problem Size	Number of Simulation runs	Average Time Full Program	Average Time with Theorems	Percent Time Required
10	50	0.018	0.014	77.8
20	50	0.024	0.016	66.7
30	50	0.058	0.024	41.4
40	50	0.138	0.040	29.0
50	50	0.298	0.062	20.8
60	50	0.572	0.092	16.1
70	50	1.006	0.134	13.3
80	50	1.778	0.190	10.7
90	50	2.598	0.260	10.0
100	50	3.840	0.352	9.2
110	50	5.524	0.450	8.1
120	50	7.696	0.572	7.4
130	50	10.464	0.704	6.7
140	50	13.940	0.866	6.2
150	50	18.210	1.040	5.7

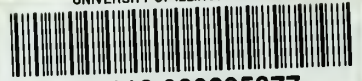
HECKMAN
BINDERY INC.



JUN 95

© N. MANCHESTER

UNIVERSITY OF ILLINOIS-URBANA



3 0112 060295877