European Journal of Operational Research 69 (1993) 187-199
North-Holland

Theory and Methodology

The job shop tardiness problem: A decomposition approach

N. Raman

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

F. Brian Talbot

School of Business Administration, The University of Michigan, Ann Arbor, MI 48109, USA

Received December 1990; revised September 1991

Abstract: An important criterion for evaluating the effectiveness of many manufacturing firms is their ability to meet due dates. In low to medium volume discrete manufacturing, typified by traditional job shops and more recently by flexible manufacturing systems, this criterion is usually operationalized on the shop floor through the use of prioritizing dispatching rules. The widespread use of dispatching rules has led to a number of investigations where the due date performance of various rules is compared. In contrast to previous research on dispatching rules, this paper proposes a new approach that decomposes the dynamic problem into a series of static problems. These static problems are solved in their entirely, and then implemented dynamically on a rolling basis. To illustrate this approach, a specific heuristic is developed that constructs the schedule for the entire system by focusing on the bottleneck machine. Computational results indicate that significant due date performance improvement over traditional dispatching rules can be obtained by using this new approach.

Keywords: Scheduling; Tardiness; Job shop

1. Introduction

In many manufacturing systems, especially those that produce to specific customer orders, the major scheduling objective is to meet order due dates. Considerable research has, therefore, been directed to finding effective solution procedures for minimizing average job tardiness. The bulk of this research on multiple machine systems addresses dynamic job shops, and it deals with the relative effectiveness of

Correspondence to: N. Raman, Department of Business Administration, University of Illinois at Urbana-Champaign, Champaign, IL 61820, USA.

various priority dispatching rules. (See, for example, Conway (1965), Carroll (1965), Baker and Bertrand (1982), Kanet and Hayya (1982), Baker and Kanet (1983), Baker (1984) and Vepsalainen and Morton (1987). Baker (1984) given an excellent summary of this research.)

This paper present an approach that is an alternative to the use of priority dispatching rules. The proposed solution method treats the dynamic problem as a series of static problems. A static problem is generated at each occurrence of a stochastic event such as a new job arrival, a machine breakdown, etc. Each static problem is solved entirely, and the solution is implemented on a rolling basis. In contrast to dispatching rules, which consider only one machine at a time, the suggested procedure schedules the entire system at each instance of the static problem. This approach was used successfully for single machine systems in Raman, Rachamadugu and Talbot (1989a).

Static job shop scheduling for most of the commonly studied objectives presents problems of exponential complexity (Rinnooy Kan 1976, pp. 131–134), and much of the research on this subject has been directed toward minimizing makespan. Although the single machine tardiness problem is well studied, very little is reported in scheduling literature on general job shops. The dominance conditions and bounding mechanisms developed for single machines cannot be easily extended to job shops. Indeed, the only work on multiple machine tardiness problems that we are aware of are those of Ow (1985) and Raman, Talbot and Rachamadugu (1989b). Ow develops conditions for local optimality between adjacent jobs for a 2-machine flow shop, and utilizes these conditions for constructing a heuristic solution method for a general proportionate flow shop. Raman, Talbot and Rachamadugu present an implicit enumeration approach for a general multiple machine system. However, for the reasons noted above, the size of problems solved to known optimality is relatively small.

In summary, dispatching rules are the only solution methods currently available for solving even *static* job shop tardiness problems of reasonable size. While dispatching procedures are computationally efficient, the effectiveness of any one rule depends upon system characteristics such as machine utilization, flow allowance factor, etc. Therefore, given the computing power available today, Adams, Balas and Zawack (1988) contend that there is a need for developing more effective procedures even if they require additional computational effort. This paper presents one such procedure.

The heuristic solution method developed in this paper for solving the static problem is based on decomposing the multiple machine problem, and constructing the schedule for the entire system around the bottleneck machine. This is done by establishing relative job priorities using operation due dates (ODDs). While the notion of operation milestones has been used extensively in the past, the major contribution of the suggested method lies in the manner in which these ODDs are derived by taking into account the impact of other jobs in the system, and in combining ODD assignment with operation scheduling. Although this approach requires greater computational effort, we show that, in general, it results in significantly better system performance for both static and dynamic problems.

The remainder of this paper is organized as follows. The static tardiness problem is formulated in Section 2. In Section 3, the decomposition based heuristic solution procedure is presented. An optimal method for the static problem, which is used for benchmarking, is reviewed in Section 4. In Section 5 and 6, computational results for the static and the dynamic problems are provided. Summary comments are given in Section 7. The notation used in this paper is given in the Appendix.

2. Static scheduling problem

An integer programming formulation is presented below for the static tardiness problem. We assume without loss of generality that the operations for each job are numbered such that any successor operation is indexed higher than its predecessors.

$$\text{Minimize} \sum_j T_j \tag{1}$$

subject to

$$\sum_t x_{tjk} = 1, \quad j = 1,\dots,N, \quad k = 1,\dots,N_j, \tag{2}$$

$$\sum_j (t - p_{jl}) x_{tjl} \geq \sum_t t x_{tjk}, \quad j = 1,\dots,N, \quad k = 1,\dots,N_j \quad \forall (k, l) \in S_j, \tag{3}$$

$$\sum_j \sum_k \sum_{q=t}^{t+p_{jk}-1} R_{jkm} x_{qjk} \leq 1, \quad t = 1,\dots,T, \quad m = 1,\dots,M, \tag{4}$$

$$\sum_t t x_{tjk} + E_j - T_j = d_j, \quad k = N_j, \quad j = 1,\dots,N, \tag{5}$$

$$x_{tjk} \in (0, 1), \quad E_j, T_j \geq 0 \quad \forall j, k, t \tag{6}$$

where

$$x_{tjk} = \begin{cases} 1 & \text{if operation } k \text{ of job } j \text{ is completed at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

Equation (1) minimizes total tardiness. Constraints (2) require that each operation is completed exactly once. Constraints (3) ensure that precedence relationships among the various operations within a job are satisfied, and operation processing times are taken into consideration appropriately. Constraints (4) specify that each operation is assigned to at most one machine at any given time. Constraints (5) measure the tardiness of each job, and constraints (6) specify the integer nature of the variables.

3. Heuristic solution approach

3.1. Background

In view of the complexity of the mean tardiness problem, most solution procedures are based on the use of dispatching rules. These rules prioritize jobs using a criticality index based on job and system status. All machines are forward scheduled, and ties among competing jobs are broken using the priority index. While several schemes are available for generating these indexes (see, for example, Conway 1965), Baker (1984) and Vepsalainen and Morton (1987) show the superiority of decomposing job due dates into operation milestones, and using these operation due dates (ODDs) for setting priorities.

In particular, Baker finds that the Modified Operation Due Date (MOD) rule performs well across a range of due date tightness. MOD selects the operation with the minimum modified operation due date. The modified operation due date of operation i in job j is given by

$$\text{MOD}_{ji} = \max(t + p_{ji}, d_{ji}) \tag{7}$$

where t is the time when the scheduling decision needs to be made.

Remark 1. *For a given set of operation due dates, the total tardiness incurred by two adjacent operations in a non-delay schedule on any given machine does not increase if they are resequenced according to the MOD rule.*

Proof. Refer to Raman, Talbot and Rachamadugu (1989c).

Remark 1 indicates that if ODDs are set optimally, the MOD solution guarantees local optimality between adjacent operations at any machine for a non-delay schedule. ODD assignment is, therefore, central to the effectiveness of the MOD rule. Most previous implementations of this rule determine

ODDs by decomposing the total flow allowance $(d_j - p_j)$ of a given job j heuristically into individual operation flow allowances. (Baker (1984) describes these procedures.) ODDs obtained in this manner are retained throughout the solution procedure. Recently, Vepsalainen and Morton (1987) developed an approach in which the due date of any operation is estimated by netting the lead time for the remaining operations from the job due date. Because this lead time depends upon the *relative* priority of the job under consideration, it considered the interaction among all jobs in the system.

Similar to Vepsalainen and Morton's approach, we account for job interaction effects explicitly. However, our procedure differs from previous approaches in that we consider ODD assignment and operation scheduling simultaneously. Note that, in conjunction with the MOD rule, ODDs essentially provide a mechanism for allocating individual job priorities. Clearly, the best set of ODDs is one which yields the best prioritization scheme, i.e., one that yields the minimum average tardiness. Therefore, the goodness of a given set of ODDs can be determined only when the system is scheduled simultaneously.

As a result, the proposed approach is not a single pass procedure. It considers the global, systemwide impact of each ODD assignment. While this increases computational requirements, the experimental results shown in Sections 5 and 6 indicate that significant benefits are realized.

3.2. The Global Scheduling Procedure (GSP)

The proposed method is a schedule improvement procedure. The initial solution is generated by applying the MOD rule with ODDs set loosely at the maximum values that they can assume without delaying the corresponding jobs. Each machine is then considered in order, and an attempt is made to revise the schedule of operations on that machine by modifying their ODDs. Jobs processed on all machines are ranked in the nonincreasing order of their tardiness. For any operation in a given job with positive tardiness, first we determine the appropriate interval for searching for the ODD. For each possible ODD value in this interval, the entire system is rescheduled. The value which yields the minimum total tardiness is returned as the ODD for that operation. This step is repeated for all other operations of that job processed on the machine under consideration, for all other tardy jobs on that machine following their rank order, and for all machines in the system.

We use the relative workload of a given machine to determine its criticality, and consequently, to determine the order in which the machines are scanned. Because an average job spends a greater portion of its total waiting time at the bottleneck machine(s), any improvement in overall system performance will likely require a revision in the operation due dates and the sequence of operations at these machines. The algorithm ranks all the machines from the most heavily loaded to the least loaded one, and considers them in order. Because the relative ranking of machines remains unchanged, they are numbered according to their rank, machine 1 being the most heavily loaded machine.

The algorithm is now presented. In the following, $\mathcal{M}$ is the set of all machines, $\mathcal{J}_m$ is the ordered set of jobs processed on machine m, n_{jm} is the number of operations that job $j \in \mathcal{J}_m$ requires on m, and i_l^m is the l-th operation of any given job on m.

Step 1. Initialization:
- a) Assign initial operation due dates; ODD of operation i in job j is given by $d_{ji} = d_j - (p_j - P_{ji})$.
- b) Construct the initial sequence using the MOD rule with respect to the ODDs determined above.
- c) Number all machines in the non-increasing order of their total workloads $\sum_{j \in \mathcal{J}_m} \sum_{i=1}^{n_{jm}} p_{ji}$.
- d) Initialize counters: $r = 0$; $z(0) = \infty$.
- e) Assign the set of unscanned machines $\mathcal{M}_1 = \mathcal{M}$. Set $r = r + 1$.

Step 2. Machine and job selection:
- a) Select machine m^* next for scanning where $m^* = \min_{j \in \mathcal{M}_1}\{j\}$.
- b) List all jobs j and $\mathcal{J}_m$, $\forall m$, in non-increasing order of their current tardiness T_j. Select job j^* from the top of the list of all jobs in $\mathcal{J}_{m^*}$. Set $l = 1$.

Step 3. Schedule revision:

a) For operation i_l^{m*} in $j*$, determine the interval of values that its ODD, $d_{j*,i_l^{m*}}$, can assume.

b) For each integer value x in this interval, generate the due dates of other operations in $j*$, and reschedule all machines using the MOD rule. Record total tardiness $\Sigma_j T_j$ for each x.

c) Assign $d_{j*,i_l^{m*}}$ to the x-value that results in the minimum total tardiness. Reassign due dates of other operations in $j*$ accordingly. Update T_j, $\forall j$, based on the most recently assigned operation due dates.

d) If $l = n_{j*m}$, go to Step 4. Else, set $l = l + 2$, and go to Step 3a.

Step 4. Updating:

a) Remove $j*$ from the list of unscanned jobs on machine $m*$: $\mathcal{J}_{m*} = \mathcal{J}_{m*} \setminus \{j*\}$. If $\mathcal{J}_{m*} \neq \emptyset$, go to Step 2b.

b) Update the list of unscanned machines: $\mathcal{M}_1 = \mathcal{M}_1 \setminus \{m*\}$. If $\mathcal{M}_1 \neq \emptyset$, go to Step 2a.

c) Record $z(r) = \Sigma_j T_j$. If $z(r) < z(r-1)$, go to Step 1e. Else, stop.

The major step in the algorithm is that of schedule revision which is now discussed.

3.3 Schedule revision

3.3.1. ODD reassignment

Consider Figure 1 which illustrates the schedule revision procedure. The solution tree shown is similar to a branch-and-bound enumeration tree with the difference that each node represents a complete solution.

Given the initial solution, we start with machine 1 which has the maximum workload, and job j (say) with the maximum tardiness among all jobs in $\mathcal{J}_1$. Let j require operations $i_1, i_2, \ldots, l_1$ on machine 1. (We suppress superscript m for simplicity.) Consider operation i_1 whose initial ODD is d_{j,i_1}. The

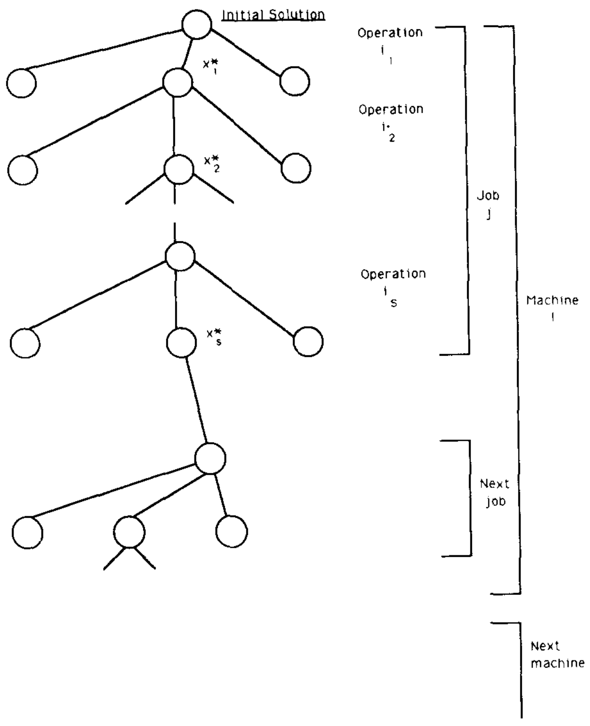

Figure 1. Solution tree for GSP scheduling procedure

algorithm now considers varying this due date to have integer values in an interval $[L_1, U_1]$ where $L_1 = \Sigma_{l=1}^{i_1} p_{jl}$ and $U_1 = d_j$. Note that L_1 is the earliest time that i_1 can be completed. It follows from (7), therefore, that for any $d_{j,i_1} < L_1$, the relative priority of i_1 remains unchanged.

A descendant node is generated for each value of x in this interval. For a given x, the ODDs of other operation in j are generated as follows:

$$d_{jk} = d_{j,k-1} + (x - p_{j,i_1}) p_{jk}/P_{j,i_1-1}, \quad k = 1, \ldots, i_1 - 1,$$

and

$$d_{jk} = d_{j,k-1} + (d_j - x) p_{jk}/(p_j - P_{j,i_1}), \quad k = i_1 + 1, i_1 + 2, \ldots, N_j.$$

In effect, we split job j into three 'sub-jobs' j_1, j_2 and j_3. j_1 consists of all operations prior to i_1, j_2 contains only i_1, and j_3 comprises all operations subsequent to i_1. Due dates of all operations within a sub-job are set independently of other sub-jobs. They are derived from the due date of the corresponding sub-job by assigning them flow allowances proportional to their processing times, due dates of j_1, j_2 and j_3 being $x - p_{j,i_1}$, x and d_j, respectively. ODDs of operations in other jobs remain unchanged.

The solution value for the descendant is determined by rescheduling all jobs at all machines for the revised set of ODDs using the MOD rule. The branch corresponding to the node with the minimum total tardiness is selected, and the ODD of i_1 is frozen at the corresponding value of x, say x_1^*. ODDs of all operations in job j preceding i_1 are updated as follows:

$$d_{jk} = d_{j,k-1} + (x_1^* - p_{j,i_1}) p_{jk}/P_{j,i_1-1}, \quad k = 1, \ldots, i_1 - 1.$$

The due date of operation i_2 is assigned next. The interval scanned for d_{j,i_2} is $[L_2, U_2]$ where $L_2 = \Sigma_{k=i_1+1}^{i_2} p_{jk} + x_1^*$ and $U_2 = d_j$.

For a given value of x for the ODD of i_2, the due dates of operations in job j, excluding i_1, i_2 and those which precede i_1, are generated as follows:

$$d_{jk} = d_{j,k-1} + \frac{(x - x_1^* - p_{j,i_2}) p_{jk}}{P_{j,i_2-1} - P_{j,i_1}}, \quad k = i_1 + 1, i_1 + 2, \ldots, i_2 - 1,$$

and

$$d_{jk} = d_{j,k-1} + \frac{(d_j - x) p_{jk}}{p_j - P_{j,i_2}}, \quad k = i_2 + 1, i_2 + 2, \ldots, N_j.$$

ODDs of operations preceding and including i_1 remain unchanged.

In the general step, suppose we are considering ODD reassignment of operation k of job j at machine m. Suppose further that after investigating machines 1 through $m - 1$, and all operations of job j prior to k on machine m, we have frozen the due dates of operations $u_1, u_2, \ldots, u_z$ in job j. Let k be processed between operations u_l and u_{l+1} with frozen due dates of x_l^* and x_{l+1}^*, respectively. In other words, the ordered sequence of operation in j is

$$(1, 2, \cdots, u_1, \cdots, u_2, \cdots, , \cdots, u_l, \cdots, k, \cdots, u_{l+1}, \cdots, , \cdots, u_z, \cdots, N_j).$$

Then, for assigning the ODD of k, we need to consider only the interval $[\Sigma_{r=u_l+1}^{k} p_{jr} + x_l^*,\ x_{l+1}^* - p_{j,u_{l+1}}]$. In addition, while reassigning the operation due date of k, ODDs need to be generated for only those operations which are processed between u_l and u_{l+1}.

As we go down the list of machines and move from one operation to another of a given job at a machine, the search interval becomes smaller. However, near the top of the tree, it can be quite wide resulting in a large number of descendant nodes from a given parent node. We now describe an efficient procedure for improving the search routine.

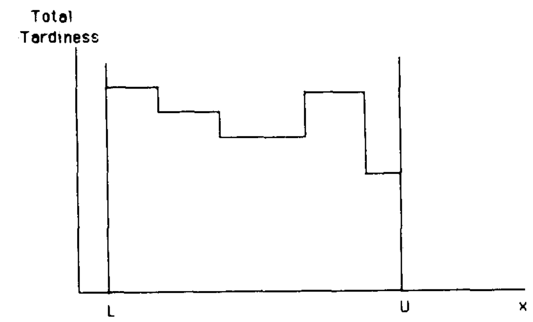

Figure 2. Typical behavior of total tardiness against x

3.4. Search routine

While searching for the reassigned ODD value x for a given operation at any machine, we need to theoretically consider the appropriate interval $[L, U]$ in unit steps. Note, however, that while x can take many values, an operation can only occupy a given number of positions θ in any sequence. For a permutation schedule in a single machine problem, $\theta = N$. In a job shop, θ is large because of the forced idle times at different machines. Nonetheless, it is usually much smaller than the number of different values that x can take. Consequently, operation completion times and, therefore, total tardiness as well, remain unchanged for many sub-intervals within $[L, U]$. Figure 2 illustrates the typical behavior of total tardiness with respect to x for a given operation.

The procedure for searching for the best value of x for an operation in a given job employs a modification of the binary search method. As shown in Figure 3, suppose that we need to search in the interval $[L_0, U_0]$. First, we compute tard(L_0) and tard(U_0), where tard(d) is the total tardiness of all jobs when $x = d$. Starting with the interval $[L_0, U_0]$ we successively divide each interval into two equal halves and compute the total tardiness value at the midpoint of each half-interval. Within any generated interval, scanning for the next half-interval is initially done to the left. In other words, with reference to Figure 3, we have $U_i = \frac{1}{2}(L_0 + U_{i-1})$, $i = 1, 2, 3$.

Scanning to the left within a half-interval terminates when it is fathomed. An interval is said to be fathomed if it is the most recently generated interval and the total tardiness values at its end-points and mid-point are the same. In Figure 3, for example, the interval $[L_0, U_3]$ is fathomed. Note that the fathoming procedure will ignore changes in total tardiness values within an interval if, in spite of such changes, the *same* tardiness value is realized at both end points and the mid-point of that interval. While such occurrences are possible, they are somewhat unlikely in most real problems. (We did not observe it in any one of the 50 randomly generated problems.) Nevertheless, it should be noted that while trying to achieve computational efficiency, this search procedure may not always return the best value of x.

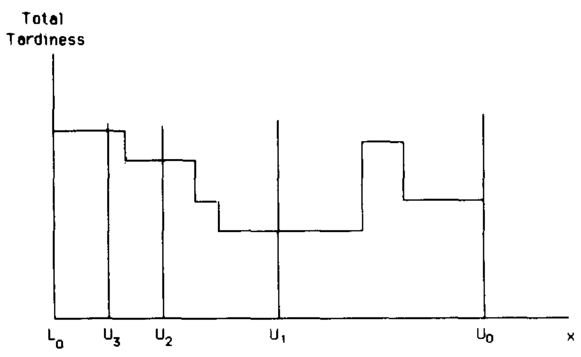

Figure 3. Search procedure

At the termination of left-scanning, the procedure next evaluates the most recently generated and unfathomed interval to its right. If the total tardiness values at both its end-points and its mid-point are not the same, another half-interval is generated and left-scanning is resumed. The procedure terminates when all half-intervals are fathomed. The due date of the operation under consideration is reassigned to the x-value that results in the minimum total tardiness.

Note that it is possible that the position of any operation of a given job which results in the minimum total tardiness may result in that job itself being late (if by doing so, tardiness of other jobs improved significantly). For this reason, it is desirable to increase the upper limit of the search interval for the initial operations of job j from d_j to some arbitrarily large value T. The actual value used for T is of marginal importance because intervals which do not affect total tardiness are rapidly fathomed. In our experimental study, T equaled the makespan of the initial solution.

The search routine reduces the computational effort for each iteration from $O(M^2K^3\Sigma_j p_j)$ to $O(M^2K^4)$, where $K = \Sigma_j N_j$ is the total number of operations. For many medium to large problems, it may be necessary to run this algorithm with a time trap. In such cases, we use an additional mechanism for controlling the breadth of the solution tree shown in Figure 1. The number of intervals searched during the ODD reassignment at machine m is limited to $\nu(m)$ which is a function of the workload of m (more intervals are searched for machines with higher workloads), the time taken for generating a complete schedule for the system using the MOD rule, and the specified time trap.

We describe our computational experience with this algorithm for both static and dynamic problems in Sections 5 and 6. In the next section, we review the exact solution procedure for the static tardiness problem given in Raman et al. (1989b).

4. Exact solution procedure

In order to establish computational benchmarks for the proposed heuristic, an optimum-seeking procedure was also developed and tested. This procedure is a modification of the exact solution approach developed by Talbot (1982) for minimizing makespan in a job shop scheduling problem. The procedure uses a depth-first branch and bound algorithm which builds a schedule forward in time. A node at level L in the solution tree has an associated array A_n which contains the indexes of operations which are schedulable at the next level. The precedence relationships restrict the cardinality of A_n to the number of jobs in the system, which reduces computer storage as well as computational time requirements.

Starting with the unique node at level 0, the procedure selects the next operation based on a priority index associated with each operation or move. Three priority schemes are used in this paper. These are based on the Modified Due Date (MDD) rule, the Modified Operation Due Date (MOD) rule, and the Random (RAN) rule (Raman et al., 1989b). In each case, the descendants (operations) of any node are ranked in the nondecreasing order of their priority indexes. These priority schemes are also used to generate the initial solution. Backtracking, rather than skiptracking, is employed to keep storage requirements at a minimum. This optimum-seeking procedure is used in two ways in the computational study:

1) To act as a CPU 'level of effort' benchmark, and
2) to provide optimal tardiness solutions to a set of static problems.

5. Computational study – static problem

5.1. Experiment design

Two sets of experiments were conducted for the static problem to assess the relative performance of the Global Scheduling Procedure given in Section 3.

The first set compares GSP with five dispatching rules from the literature – Shortest Processing Time (SPT), Earliest Due Date (EDD), Critical Ratio (CRIT), Modified Job Due Date (MDD), Modified Operation Due Date (MOD), and Hybrid (HYB) (Raman et al., 1989b) rules. Since GSP is computationally more demanding than the direct implementation of these rules, the following steps were taken to provide for an unbiased comparison. First, these dispatching rules were implemented in the probabilistic dispatching mode (see, for example, Baker, 1974, pp. 202–205). In probabilistic dispatching, each operation, among those available for scheduling at a given point in time, was assigned a selection probability proportional to the priority assigned by the dispatching rule. Random sampling from this probability distribution was then used to determine the operation to be selected. In every problem instance, each dispatching rule was allowed to run for 15 seconds, and the best tardiness value obtained during this period across all rules was recorded. Similarly, GSP was allowed to run for 15 seconds.

In addition, the implicit enumeration procedure discussed in Section 4 was run with a time trap of 15 seconds. This procedure was implemented in three versions, requiring the use of MDD, MOD and RAN priority schemes respectively. The best solution from these three approaches was selected for reporting purposes.

Two parameters, Z and R, were used to control the tightness and the variation of job due dates respectively. The tardiness factor Z measures approximately the proportion of jobs likely to be tardy, while R determines the range of job due dates. For given Z and R, job due dates were sampled from a uniform distribution in the interval $[\bar{d}(1 - \frac{1}{2}R), \bar{d}(1 + \frac{1}{2}R)]$ where the average job due date $\bar{d}$ is given by $\bar{d}C =_{\max}(1 - Z)$, and $C_{\max}$ is the makespan of the sequence obtained by scheduling all operations on a first-come-first-serve basis. Z and R have been used extensively for generating test data in single machine tardiness problems (see, for example, Srinivasan, 1971). Because of the forced machine idle times, Z is only an approximate measure of the proportion of tardy jobs in a job shop. Nonetheless, it helps anchor due date tightness at various levels.

Several problems scenarios were generated by varying one or more of the following parameters:
1. *Number of machines* (NOM): 5, 10.
2. *Number of jobs* (NJ): 15, 35.
3. *Tardiness factor (Z)*: 0.4, 0.6.
4. *Range of due dates (R)*: 0.5, 1.5.

For each scenario, ten problems were randomly generated by sampling operation processing times from a uniform distribution in the interval [5,100], and the average value across these instances was recorded. All jobs had randomly assigned routing through the system, although successive operations of any job were processed on different machines. In total, therefore, 160 problems were solved for each scheduling approach. For reporting purposes, we used normalized value of total tardiness (NMT), where NMT $= \Sigma_j T_j / (\Sigma_j p_j)$.

The second set of experiments for the static problem considered a 3-machine, 5-job system. The size of the problems considered in this set was deliberately restricted in order to keep the computational costs within reasonable limits. As in the case of the first set of experiments, four combinations of the tardiness factor and job due date range, for $Z = 0.4$ and 0.6, and $R = 0.5$ and 1.5, were considered. For each combination, 25 problems were generated randomly. These 100 problems were solved optimally using the implicit enumeration approach as well as heuristically by the GSP method. The average of the tardiness values obtained across all 25 problems for each scenario under each of these two approaches was recorded.

The experiments reported in this section as well as in Section 6 were conducted on the IBM 3090-600 mainframe computer at the University of Michigan.

5.2. Analysis of static problem results

Results of the first experiment are shown in Table 1. For each of the 16 problem scenarios, the NMT values obtained from GSP are compared to those of the dispatching rule found to be the best under direct and probabilistic dispatching modes individually, and those of the best priority scheme used in the

Table 1
Static problem: normalized mean tradiness results [a]

Scenario (Z; R)	Dispatching		Enume-ration	GSP		
	Direct	Prob.		Value	% Imp.	Time [b]
5 machines, 15 jobs:						
(0.4; 0.5)	0.334	0.285	0.325	0.251	11.9	0.614
(0.4; 1.5)	0.416	0.363	0.376	0.322	11.3	0.542
(0.6; 0.5)	0.835	0.730	0.828	0.693	5.1	0.870
(0.6; 1.5)	0.836	0.767	0.832	0.724	5.6	0.789
5 machines; 35 jobs:						
(0.4; 0.5)	0.422	0.422	0.406	0.325	20.0	5.921
(0.4; 1.5)	0.255	0.255	0.185	0.155	16.2	5.504
(0.6; 0.5)	1.226	1.226	1.218	1.068	12.3	10.082
(0.6; 1.5)	1.157	1.157	1.149	0.949	17.3	11.671
10 machines, 15 jobs:						
(0.4; 0.5)	0.273	0.210	0.273	0.203	3.3	3.850
(0.4; 1.5)	0.368	0.317	0.362	0.309	2.5	3.289
(0.6; 0.5)	0.631	0.532	0.631	0.542	−1.9	5.373
(0.6; 1.5)	0.675	0.576	0.664	0.571	0.9	5.199
10 machines, 35 jobs:						
(0.4; 0.5)	0.333	0.333	0.250	0.230	8.0	15.095
(0.4; 1.5)	0.372	0.372	0.316	0.266	15.8	15.097
(0.6; 0.5)	0.911	0.892	0.916	0.796	10.8	15.098
(0.6; 1.5)	0.867	0.864	0.864	0.748	13.4	15.099

[a] Direct Dispatching rules were run without a time trap. They consistently took less than 0.1 seconds per problem. Probabilistic Dispatching rules and the Implicit Enumeration approach were allowed to run for 15 seconds per problem. The best solution found within this time trap was used for reporting.
[b] Actual CPU time given an 15 second time trap.

implicit enumeration approach. For GSP, we also report the improvement over the next best procedure tested, as well as the average computational effort in CPU seconds.

GSP is seen to provide the best results yielding an average improvement of 12.7% over the next best rule. It performs consistently better than the dispatching rule implemented directly as well as the enumeration based approach. It yields better values than probabilistic dispatching in 15 out 16 scenarios. Recall that the results reported for probabilistic dispatching are those of the rule that performed the best among the 5 rules tested for a given scenario, and for each rule the procedure was allowed to run for a duration equal to the time trap used for GSP. Similarly, the results reported for the enumeration approach is the best among the three priority schemes examined, each being allowed to run for 15 seconds. Note that in all but the case of 10 machines and 35 jobs, GSP terminated well ahead of the time trap.

In general, the relative superiority of GSP is higher for $Z = 0.4$ and $R = 1.5$. For a given number of machines, it also improves with an increase in NJ. The dependence on NOM is, however, less obvious.

Table 2
Static problem: normalized mean tardiness. Comparison of GSP with the operational solution

Scenario (Z; R)	Optimal value	GSP solution value	% Deviation from optimality
(0.4; 0.5)	0.252	0.259 (19) [a]	2.8
(0.4; 1.5)	0.387	0.417 (15)	7.8
(0.6; 0.5)	0.538	0.554 (17)	3.0
(0.6; 1.5)	0.637	0.653 (16)	2.5

[a] Number of problems out of 25 for which GSP found the optimal solution.

When NJ = 35, an increase in NOM does not appear to appreciably affect the relative performance of GSP. On the other hand, for NJ = 15, its effectiveness is reduced with an increase in NOM even though it remains superior in 3 of 4 scenarios.

The results of the second experiment are given in Table 2. The number of times GSP found the optimal solution is shown in parentheses next to the GSP solution value. This table indicates that GSP frequently finds the optimal solution, and in 3 out of the 4 cases, its average solution value is quite close to the optimal value.

6. Computational study – dynamic problem

Experimental investigation of the dynamic scheduling problem addresses the effectiveness of implementing the solution of the static problem obtained by GSP on a rolling basis. In a dynamic environment, a static problem needs to be generated whenever a new job arrives (or in a real system, when a machine breaks down or to reflect actual, versus planned, processing times, etc.). At that point in time, the network depicted in Figure 1 is generated afresh taking into account the operations already in process. Note that at that point in time, one or more machines can be busy. Since preemption is not permitted, such resources are blocked out for the period of commitment. The solution determined by GSP is implemented until the next job arrives when the process of generating and solving the static problem is repeated, thus closely mimicking how GSP would be used in a real system.

6.1. Experiment design

The experimental model considers a job shop with jobs arriving randomly following a Poisson process. Each job is assigned a due date that provides it a flow allowance proportional to its processing time, resulting in a due date $d_j = a_j + Fp_j$ where a_j is the arrival time of job j, p_j is its total processing time and F is the flow allowance factor that controls due date tightness. Four levels of due date tightness are achieved by using F values of 2, 3, 4, and 5.

Each job has a random routing through the system which comprises 5 machines. The operation processing times at each machine are sampled from a uniform distribution. This distribution was varied to yield two levels of relative machine workloads. The actual machine utilizations obtained ranged from 78% to 82% for the case of *balanced workloads*, and from 66% to 93% for *unbalanced workloads*. In both cases, the average shop utilization was approximately 80%.

As in the case of the static problem, GSP is compared with SPT, EDD, CRIT, MOD, MDD and HYB rules. GSP was implemented with a time trap of 1.0 CPU second per static problem in order to keep computational costs within reasonable limits.

6.2. Analysis of dynamic problem results

The computational results are shown in Table 3. Among the dispatching rules, MOD performed the best across all scenarios. It is, therefore, used as the benchmark for evaluating the effectiveness of GSP.

Table 3
Mean tardiness for dynamic problem

Flow allowance	Balanced workloads			Unbalanced workload		
	MOD	GSP	α	MOD	GSP	α
2	268	252	0.15	396	357	0.01
3	151	139	0.04	252	231	0.07
4	84	68	0.09	154	143	0.06
5	39	28	0.11	111	81	0.09

Also given in the table are the values yielded by GSP and the corresponding level of significance α for one-tailed tests concerning paired differences between MOD and GSP. GSP is seen to retain its effectiveness for all flow allowances, and for both levels of workload balance. As shown in the table, these results holds for significance levels of 0.15 or better.

7. Summary

This study examines the effectiveness of decomposing a dynamic mean tardiness problem into a series of static problems and implementing the solution to the static problem on a rolling basis. In so doing, it also evaluates the impact of the solution quality for the static problem within a dynamic framework. To illustrate this approach, a specific heuristic is developed which first determines the sequence for the bottleneck machine. The schedule for the entire system is then constructed around the sequence at the bottleneck machine and implemented dynamically.

The experimental results show that by expending extra computational effort in developing a global schedule for the entire system, significant improvement over local dispatching rules can be achieved for both static and dynamic problems. This improvement is demonstrated even after controlling for relative computational effort. In addition, for the dynamic system, significant performance improvement is achieved even when GSP is allowed an execution time of only 1.0 second. In a real system, such an early termination of the scheduling procedure will rarely be required. Computations can continue until there is a system change (for example, a job arrival) that triggers the need for a new schedule. Generally, this time would be in minutes or hours (not 1.0 second), and hence, more static problems would be solved completely which could possibly lead to further improvement in the performance of GSP. In addition, because of its relatively small memory requirement, GSP can easily be implemented on today's microcomputers.

The approach of treating a dynamic problem as a series of static problems is immediately applicable in practice where it could potentially result in significant due date performance improvement in job shops or flexible manufacturing systems. In a real system, there would be other stochastic events such as machine breakdowns, variations in processing times, etc., in addition to random job arrivals. Under the proposed approach, static problems are generated and solved at each occurrence of such events. Dispatching rules also solve static problems; however, unlike GSP, they use only partial information which is usually local to the machine at which the scheduling decision is to be made.

Appendix. Notation

j	= Job index, $j = 1, \ldots, N$.
J	= Set of available jobs $= \{j\}$.
m	= Machine index, $m = 1, \ldots, M$.
t	= Time period, $t = 1, \ldots, T$, where T is the scheduling horizon.
d_j	= Due date of job j.
p_j	= Processing time of job j.
c_j	= Completion time of job j.
S_j	= Sets of pairs of adjacent operations in job j.
N_j	= Number of operations in job j.
T_j	= Tardiness of job $j = \max(0, c_j - d_j)$.
E_j	= Earliness of job $j = \max(0, d_j - c_j)$.
p_{jk}	= Processing time of operation k in job j.
P_{jk}	= Cumulative processing time for job j up to and including operation $k = \Sigma_{i=1}^{k} p_{ji}$.
d_{jk}	= Due date of operation k in job j.
R_{jkm}	$= \begin{cases} 1 & \text{if operation } k \text{ of job } j \text{ requires machine } m, \\ 0 & \text{otherwise.} \end{cases}$

Acknowledgement

Significant contributions were made to this research by Ram Rachamadugu at the University of Michigan. The authors are also grateful to the anonymous referees for their helpful comments. The research of N. Raman was partially supported by the IBE Summer Research Grant at the University of Illinois at Urbana-Champaign.

References

[1] Adams, J., Balas, E., and Zawack, D. (1988), "The shifting bottleneck procedure in job shop scheduling", *Management Science* 34, 391–401.
[2] Baker, K.R. (1974), *Introduction to Sequencing and Scheduling*, Wiley, New York.
[3] Baker, K.R. (1984), "Sequencing rules and due date assignments in a job shop", *Management Science* 30, 1093–1104.
[4] Baker, K.R., and Bertrand, J.M.W. (1982), "A dynamic priority rule for sequencing against due dates", *Journal of Operations Management* 3, 37–42.
[5] Baker, K.R., and Kanet, J.J. (1983), "Job shop scheduling with modified due dates", *Journal of Operations Management* 4, 11–22.
[6] Carroll, D.C. (1965), "Heuristic sequencing of single and multiple component jobs", Ph.D. Dissertation, MIT, Cambridge, MA.
[7] Conway, R.W. (1965), "Priority dispatching and job lateness in a job shop", *Journal of Industrial Engineering* 16, 123–130.
[8] Kanet, J.J., and Hayya, J.C. (1982), "Priority dispatching with operation due dates in a job shop", *Journal of Operations Management* 2, 155–163.
[9] Ow, P.S. (1985), "Focused scheduling in proportionale flow shops", *Management Science* 31, 852–869.
[10] Raman, N., Rachamadugu, R.V., and Talbot, F.B. (1989a), "Real time scheduling of an automated manufacturing center", *European Journal of Operational Research* 40, 222–242.
[11] Raman, N., Talbot, F.B., and Rachamadugu, R.V. (1989b), "Due date based scheduling in a general Flexible Manufacturing System", *Journal of Operations Management* 8, 115–132.
[12] Raman, N., Talbot, F.B., and Rachamadugu, R.V. (1989c), "Scheduling a general Flexible Manufacturing System to minimize tardiness related costs", Working Paper No. 89-1548, Bureau of Economic and Business Research, University of Illinois at Urbana-Champaign, Champaign, IL.
[13] Rinnooy Kan, A.H.G. (1976), *Machine Scheduling Problems: Classification, Complexity and Computations*, Nijhoff, The Hague, Netherlands.
[14] Srinivasan, V. (1971), "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness", *Naval Research Logistics Quarterly* 18, 317–327.
[15] Talbot, F.B. (1982), "Resource constrained project scheduling with time-resource tradeoffs: The nonpreemptive case", *Management Science* 28, 1197–1210.
[16] Vepsalainen, A.P.J., and Morton, T.E. (1987), "Priority rules for job shops with weighted tardiness costs", *Management Science* 33, 1035–1047.