Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Graph-Theoretic Decomposition of the Job Shop Scheduling Problem to Achieve Scheduling Robustness

S. David Wu, Eui-Seok Byeon, Robert H. Storer,

To cite this article:
S. David Wu, Eui-Seok Byeon, Robert H. Storer, (1999) A Graph-Theoretic Decomposition of the Job Shop Scheduling Problem to Achieve Scheduling Robustness. Operations Research 47(1):113-124. http://dx.doi.org/10.1287/opre.47.1.113

A GRAPH-THEORETIC DECOMPOSITION OF THE JOB SHOP SCHEDULING PROBLEM TO ACHIEVE SCHEDULING ROBUSTNESS

S. DAVID WU

Lehigh University, Bethlehem, Pennsylvania

EUI-SEOK BYEON

The Korea Transport Institute, Seoul, Korea

ROBERT H. STORER

Lehigh University, Bethlehem, Pennsylvania

(Received June 1994; revisions received September 1995, October 1996; accepted May 1997)

In this paper we study the weighted tardiness job-shop scheduling problem, taking into consideration the presence of random shop disturbances. A basic thesis of the paper is that global scheduling performance is determined primarily by a subset of the scheduling decisions to be made. By making these decisions in an a priori static fashion, which maintains a global perspective, overall performance efficiency can be achieved. Further, by allowing the remaining decisions to be made dynamically, flexibility can be retained in the schedule to compensate for unforeseen system disturbances. We develop a decomposition method that partitions job operations into an ordered sequence of subsets. This decomposition identifies and resolves a "crucial subset" of scheduling decisions through the use of a branch-and-bound algorithm. We conduct computational experiments that demonstrate the performance of the approach under deterministic cases, and the robustness of the approach under a wide range of processing time perturbations. We show that the performance of the method is superior, particularly for low to medium levels of disturbances.

This paper addresses a problem of practical importance in planning, scheduling, and control exemplified by the following quandary that exists in numerous industrial applications: detailed scheduling will maximize the efficiency of the use of different resources such as tooling, material handling, and labor; while on the other hand, a multitude of unforeseen disturbances will occur subsequently, which make it impossible to follow this schedule as originally conceived. It is therefore important to produce schedules that are both robust and adaptable to system disturbances. Motivated by experiences with solving these problems in practice, we submit that determining "when and how to make which decisions" represents the most crucial aspect of robust planning and scheduling. Answers to these questions might provide a more realistic perspective on scheduling problems. More importantly, they offer unique properties that lead to more effective planning and control methods for systems under uncertainty.

Solving a typical scheduling problem involves a tremendous combination of discrete decisions, all to be made at the moment a schedule is generated (typically the beginning of some planning horizon). Unfortunately, as a result of various uncertainties and dynamics present in the production system, the schedule often becomes obsolete almost the moment it is released for execution. We propose an alternative approach to the problem as follows. First, identify a critical subset of scheduling decisions that, to a large extent, dictate global schedule performance. Solve for this subset of decisions at the beginning of the planning horizon, and relegate the rest of the scheduling decisions to future points in time. This particular view of scheduling offers a "sketch" of a schedule that is sufficiently detailed to serve as a basis for other system planning requirements, provides a global view of the system, and yet retains much flexibility for change and adaptation.

1. RELATIONSHIP WITH OTHER WORK IN THE LITERATURE

Hierarchical Production Planning

Hierarchical production planning (HPP) has received much attention in the production planning literature. Holt et al. (1960), Winters (1962), Hax and Meal (1975), and Bitran and Hax (1977) are among the pioneering works. Various implementations of HPP remain an area of active research. Our proposed method can be considered as a variation of the hierarchical decomposition philosophy proposed by HPP. However, our approach is also significantly different for the following reasons.

1. HPP typically assumes that an existing (hierarchical) decision structure within the organization dictates an a priori, hierarchical separation of decisions. We propose an alternative view where "when and how to make which decisions" represents the very decisions to be made. We form this decision structure by solving an optimization problem

Subject classifications: Production/scheduling, approximations/heuristic: job-shop scheduling problem. Decision analysis, risk: robust optimization and control. Programming, integer, algorithms, branch-and-bound: combinatorial optimization.
Area of review: MANUFACTURING OPERATIONS.

0030-364X/99/4701-0113 $05.00

(i.e., ordered assignment), rather than assuming its existence a priori. Thus, the proposed concept can be generalized to a large variety of optimization and control problems where no existing decision structure can be found.

2. The basis of the proposed decomposition is decision "criticality" as captured in the problem graph. In hierarchical decomposition the basis is either functional, organizational, spatial, or temporal.

Scheduling and Control Methodologies

In the past decade, numerous researchers have addressed the issue of scheduling and control subject to various assumptions of system dynamics. Bean et al. (1991), Birge and Dempster (1987), and Gallego (1988a, 1988b) are among those who first consider the scheduling problem under system disturbances. They investigated "match-up" scheduling heuristics, which compute a transient schedule after a machine disruption. Wu et al. (1992a, b; 1993) proposed a "rescheduling" procedure that generates a new schedule on each occurrence of a shop disruption. The procedure considers simultaneously shop efficiency and deviation from the preschedule.

Roundy et al. (1991) suggested a reformulation of a job-shop scheduling problem to better handle shop dynamics. They solve an integer programming formulation of a static scheduling problem using Lagrangian relaxation. The final set of Lagrange multipliers is used, in turn, to derive cost functions which later guide a "price-directed dispatching module." A similar approach proposed by Morton et al. (1986, 1988) suggests an evolving-cost based method where job dispatching decisions are made based on pre-computed machine usage costs, tardiness costs, etc. Leon et al. (1994) consider "robustness" measures and robust scheduling methods that generate job-shop schedules that maintain high performance over a range of system disturbances.

The above research presents a philosophy of great practical importance: local scheduling should be allowed sufficient flexibility without losing a global view of the system. Our approach is complementary to this philosophy.

2. A DECOMPOSITION OF THE JOB SHOP SCHEDULING PROBLEM

Associated with a job-shop scheduling problem is a disjunctive graph $G(N, A, E)$ (Rinnooy Kan 1976), where $N \equiv \{0, 1, \ldots, o, *\}$ is the set of o operations (including fictitious start and final operations, 0 and *, respectively) to be processed on the set of machines, M. A is the set of conjunctive arcs representing the precedence constraints among operations. E_k is the set of disjunctive arcs representing the possible processing orders on machine k, and $E = \cup_{k \in M} E_k$ (see Figure 1a). The particular example in Figure 1 is from Adams et al. (1988). In a traditional JSP model with a regular performance measure, a solution is a conjunctive (directed) acyclic graph $D(N, A \cup \sigma)$, where σ is a consistent complete selection of the disjunctive arcs.

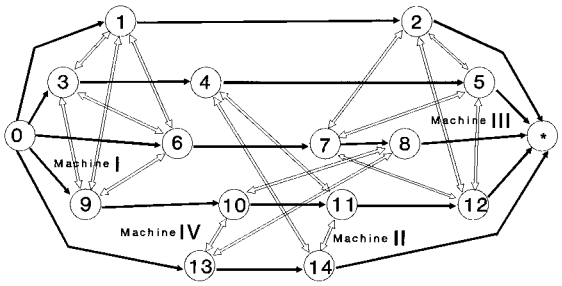

Figure 1(a). A disjunctive graph $G = (N, A, E)$.

Let $C \subset N$ denote the set of final operations of jobs, and let u_j and dd_j denote the weight and due-date associated with each job j, respectively. Then the weighted tardiness scheduling problem can be stated as follows.

The Job Shop Scheduling Problem (JSP_0). Given a disjunctive graph $G(N, A, E)$, find a schedule specified by starting times t, which minimizes the total weighted tardiness (WT).

$$\text{Minimize WT} = \sum_{j \in C} u_j (t_j + d_j - dd_j)^+$$

subject to

$$t_i + d_i \leq t_j, \quad (i, j) \in A, \tag{2.1}$$

$$t_i + d_i \leq t_j \vee t_j + d_j \leq t_i, \quad (i, j) \in E_k, k \in M, \tag{2.2}$$

$$t_i \geq 0, \quad i \in N. \tag{2.3}$$

Constraints (2.1)–(2.3) are well-known JSP constraints from (Balas 1979).

In this paper we propose a different treatment of scheduling problems. Rather than computing the entire schedule in detail, at the beginning of a planning horizon we instead decompose the JSP into a series of subproblems by solving a variant of the classical assignment problem which we term OAP. By solving OAP, all operations of the n jobs are assigned to a sequence of mutually exclusive subsets of a specified size. This, in turn, divides the disjunctive graph into a sequence of subgraphs (see Figure 1b). The effect of this assignment is that some arcs in the disjunctive set E (those with end nodes in different subsets) are resolved thus becoming conjunctive (see Figure 1c). Thus, the assignment decision not only provides a decomposition of

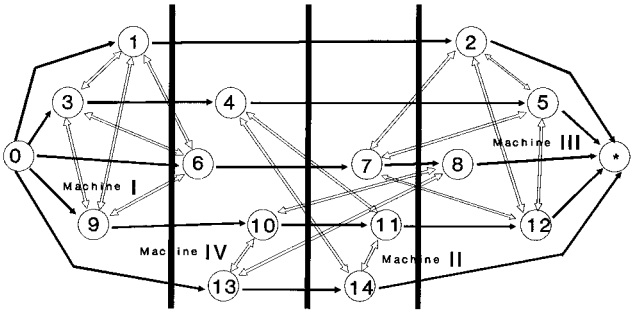

Figure 1(b). Graph with subsets.

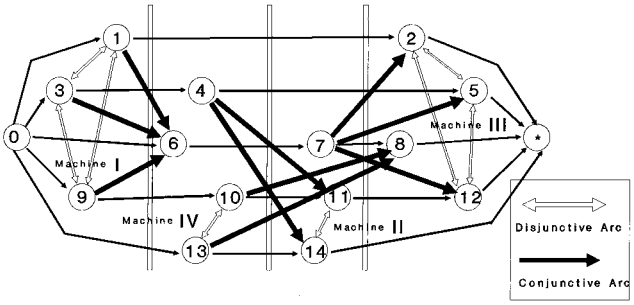

Figure 1(c). The effect of ordered assignment.

the JSP but also constitutes a significant part of the sequencing decisions. Thus, an assignment defines a "partially specified" schedule in which much flexibility is retained to allow adaptation to various disturbances in the shop.

In the remainder of this paper, we will show that (1) in the static case, the optimum of the original JSP (i.e., JSP_0) remains obtainable from the partial schedules, and (2) in the presence of shop disturbances, these schedules outperform the traditional static and dynamic schedules. We begin our exposition with the following formalism.

2.1. The Ordered Assignment and Detailed Scheduling Problems

To decompose the disjunctive graph as described previously, we define an *ordered assignment*, $\mathcal{R}$, which divides operations (in N) into a sequence of subsets. This is specified as follows.

Definition 1. Given a disjunctive graph $G(N, A, E)$ an *ordered assignment* $\mathcal{R} \equiv \{e_1, \ldots, e_s, \ldots, e_p\}$ is a set of p nonempty subsets of N which satisfies the following conditions:

(1) $\mathcal{R}$ covers the set N, i.e., $\cup_{e_s \in \mathcal{R}} e_s = N$,

(2) each element of $\mathcal{R}$ has a specified cardinality, i.e., $\forall e_s \in \mathcal{R}$, $|e_s| = \alpha_s$,

(3) the elements of $\mathcal{R}$ are mutually exclusive, i.e., $\forall e_s \in \mathcal{R}$, $\forall e_t \in \mathcal{R}$, $e_s \cap e_t = \emptyset$ if $s \neq t$,

(4) the elements of $\mathcal{R}$ are ordered, i.e., $\forall e_s \in \mathcal{R}$, $\forall e_t \in \mathcal{R}$, if $s < t$ we say that e_s precedes e_t.

Simply stated, an ordered assignment divides the nodes (operations) in the disjunctive graph $G(N, A, E)$ into a sequence of mutually exclusive, exhaustive subsets of prespecified sizes. The order in which the subsets appear in $\mathcal{R}$ establishes additional precedence relationships among operations. Obviously, this assignment of operations to subsets must be accomplished so as not to violate existing precedence relations between operations. With each assignment $\mathcal{R}$ we associate a cost $c(\mathcal{R})$ which represents the minimum possible weighted tardiness of the JSP given the precedence constraints imposed by $\mathcal{R}$. This cost function will be discussed in more detail shortly.

The problem of interest is to find an ordered assignment $\mathcal{R}$ which minimizes the cost $c(\mathcal{R})$. This optimization problem can be stated as follows.

Ordered Assignment Problem (OAP). Given a disjunctive graph $G(N, A, E)$ assign all operations in N into mutually exclusive subsets by finding an *ordered assignment* $\mathcal{R}$ such that (1) the order established by $\mathcal{R}$ does not violate the precedence constraints defined by A, i.e., $\forall e_s, e_t \in \mathcal{R}$, $\forall (i, j) \in A$, $i \in e_s, j \in e_t$ if and only if $s \leq t$, and (2) the cost $c(\mathcal{R})$ is minimized.

An OAP solution $\mathcal{R}$ defines a new disjunctive graph $G_{\mathcal{R}}(N, A \cup \sigma_{\mathcal{R}}, E_{\mathcal{R}})$ where $\sigma_{\mathcal{R}}$ is a selection of disjunctive arcs defined by $\mathcal{R}$ and $E_{\mathcal{R}} \subseteq E$ contains the remaining disjunctive arcs within the subsets (see Figure 1c). The assignment cost $c(\mathcal{R})$ represents the minimum possible scheduling cost given assignment $\mathcal{R}$. Clearly, to provide an exact evaluation of the cost $c(\mathcal{R})$ involves finding the optimal solution of the job shop scheduling problem defined by $G_{\mathcal{R}}$, which is NP-hard. To use this decomposition scheme in the context of planning and scheduling, we will have to approximate the cost $c(\mathcal{R})$ when solving OAP. In the following, we first define $c(\mathcal{R})$ mathematically as the solution of a scheduling subproblem.

The Detailed Scheduling Subproblem ($DS_{\mathcal{R}}$). Given an assignment $\mathcal{R}$ and its associated disjunctive graph $G_{\mathcal{R}}(N, A \cup \sigma_{\mathcal{R}}, E_{\mathcal{R}})$, find a schedule specified by starting times t, which minimizes the total weighted tardiness (WT).

$$(\mathbf{DS}_{\mathcal{R}}): c(\mathcal{R}) = \text{Min} \sum_{j \in C} u_j (t_j + d_j - dd_j)^+ \tag{2.4}$$

subject to

conjunctive (precedence) constraints:

$$t_i + d_i \leq t_j, \quad (i, j) \in A \cup \sigma_{\mathcal{R}}, \tag{2.5}$$

disjunctive constraints (capacity constraints for machine):

$$t_i + d_i \leq t_j \vee t_j + d_j \leq t_i, \quad (i, j) \in E_{\mathcal{R}k}, k \in M, \tag{2.6}$$

nonnegativity constraints:

$$t_i \geq 0, \quad i \in N. \tag{2.7}$$

Constraints (2.5)–(2.7) in ($DS_{\mathcal{R}}$) are the (JSP_0) constraints (2.1)–(2.3) with additional precedence arcs $\sigma_{\mathcal{R}}$, and reduced disjunctive arcs $E_{\mathcal{R}} \subseteq E$. This is the case because when an assignment decision is made by solving OAP, the disjunctive graph $G(N, A, E)$ is divided into a number of subsets in which the only disjunctive arcs remaining are within the subsets (see Figure 1c). Suppose arcs (i, j) and (j, i) are both in E_k and operations i, j are assigned to subsets s and l, respectively, with $s < l$, then (i, j) and (j, i) should be dropped from E_k and (i, j) added to the conjunctive set A. Thus, with each assignment $\mathcal{R}$ we can associate a disjunctive set $E_{\mathcal{R}}$ and a conjunctive set $A \cup \sigma_{\mathcal{R}}$. As stated in the formulation, an OAP solution $\mathcal{R}$ defines a new disjunctive graph $G_{\mathcal{R}}(N, A \cup \sigma_{\mathcal{R}}, E_{\mathcal{R}})$ where $\sigma_{\mathcal{R}}$ is a selection of disjunctive arcs defined by $\mathcal{R}$ and $E_{\mathcal{R}} \subseteq E$ contains the remaining disjunctive arcs within the subsets. For the assignment in Figure 1c, for instance, set $E_{\mathcal{R}} = \{(1, 3), (3, 9), (1, 9), (10, 13), (11, 14), (2, 5), (2, 12), (5, 12)\}$ and set $\sigma_{\mathcal{R}} = \{(1, 6), (3, 6), (9, 6), (4, 11), (4, 14), (1, 8), (13, 8), (7, 2), (7, 5), (7, 12)\}$.

OAP is NP-hard since the job shop scheduling problem is reducible to OAP. This can be shown informally as follows. Consider a special case of OAP where there are $|N| = o$ subsets each with a size of 1. An algorithm that provides an optimal solution to this OAP also provides an optimal solution to the corresponding JSP.

2.2. Some Properties of OAP

In the following, we state two important properties of OAP. First, there is at least one ordered assignment corresponding to each feasible schedule in problem JSP_0. Second, the optimum of the original scheduling problem (JSP_0) can be obtained from the detailed scheduling problem ($DS_{\mathcal{R}}$). In other words, the proposed decomposition scheme does not exclude the possibility of finding an optimal job-shop schedule. A formal proof is omitted here for the interest of brevity.

Property 1. For any feasible schedule S in problem JSP_0 there exists at least one feasible ordered assignment $\mathcal{R}(S)$ that "contains" the schedule, i.e., if we associate an acyclic digraph $D_S(N, A \cup \sigma_S)$ with schedule S, then $\sigma_{\mathcal{R}(s)} \subseteq \sigma_s$.

Property 2. Suppose set ξ is the set of optimal schedules for problem (JSP_0). Also, suppose there exists an optimal ordered assignment $\mathcal{R}^*$, and an optimal schedule S^* corresponding to problem ($DS_{\mathcal{R}^*}$). Then S^* is also optimal to problem (JSP_0), i.e., $S^* \in \xi$.

2.3. A "Preprocess First Schedule Later" Scheme for Job-Shop Scheduling

The proposed disjunctive graph decomposition motivates a more general "Preprocess First Schedule Later" (PFSL) scheme for job-shop scheduling as follows. At the beginning of a planning horizon, *preprocess* the disjunctive graph $G(N, A, E)$ for the overall scheduling problem by resolving a selected subset of disjunctive arcs in E. The graph after preprocessing, $G_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}}, E_{\mathcal{P}})$ where $(E_{\mathcal{P}} \subseteq E)$, is a more "restrictive" disjunctive graph with an expanded set of precedence constraints $(A \cup \sigma_{\mathcal{P}})$. It is a requirement that the associated digraph $D_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}})$ is acyclic. The disjunctive graph $G_{\mathcal{P}}$ is then used as a basis for the remaining scheduling decisions which are made dynamically throughout the planning period. Following the above convention, we refer to this *detailed scheduling* subproblem on $G_{\mathcal{P}}$ as ($DS_{\mathcal{P}}$).

The preprocessed graph $G_{\mathcal{P}}$ provides a means of maintaining global planning performance of the schedule and at the same time retaining options (adaptability) so that disturbances and detailed shop constraints can be dealt with via dynamic scheduling decisions. There are a number of ways the preprocessing and the dynamic scheduling can be accomplished which we explore further in the following.

Preprocessing. In general, the disjunctive graph can be preprocessed according to the inherent structure of the scheduling problem, or by a priori static information that is available at the decision point. We list a number of possibilities for preprocessing as follows:

a. Preprocessing by Assignment. Solve the ordered assignment problem (OAP) on graph G for the entire planning horizon. An assignment $\mathcal{R}$ defines an expanded set of precedence constraints $(A \cup \sigma_{\mathcal{R}})$ which defines a more "restrictive" disjunctive graph $G_{\mathcal{R}}(N, A \cup \sigma_{\mathcal{R}}, E_{\mathcal{R}})$ for dynamic scheduling. The intuition behind this preprocessing is the concept of "load balancing" via the ordered assignment of job operations. In essence, OAP places precedence constraints in the disjunctive graph such that during dynamic scheduling a subset of operations are strategically excluded from the candidate lists for a certain period of time.

b. Preprocessing by Partial Scheduling. Partial scheduling can be implemented as an additional step to *preprocessing by assignment*. Given the disjunctive graph $G_{\mathcal{R}}$, additional disjunctive arcs can be fixed in $E_{\mathcal{R}}$ by solving a priori the scheduling subproblem(s) in one or more subset(s). For instance, it is common in practice that the shop conditions are predictable in the near future while becoming progressively more uncertain further out in the planning horizon. In this case, one may choose to fix a priori all scheduling decisions for the first (several) subset(s). This preprocessing results in a further specified disjunctive graph, $G_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}}, E_{\mathcal{P}})$ where $\sigma_{\mathcal{P}} \supseteq \sigma_{\mathcal{R}}$ and $E_{\mathcal{P}} \subseteq E_{\mathcal{R}}$. $G_{\mathcal{P}}$ will thus guide the dynamic scheduling decisions throughout the planning period. In theory, the notion of *preprocessing by partial scheduling* can be applied directly on the original disjunctive graph (G) without ordered assignment. The difficulty here is that careful evaluations are needed to determine which scheduling decisions should be solved a priori and which should be solved dynamically.

c. Preprocessing by Dominance Properties. Consider a disjunctive arc (i, j), if one can find sufficient conditions that in an optimal solution operation i must precede j, then the direction of arc (i, j) can be fixed when preprocessing G. A simple example is the "immediate selection" rule proposed by Carlier and Pinson (1989, 1994) for makespan problems.

In the remainder of the paper we will focus our attention on the PFSL scheme using the notion of *preprocessing by assignment*, and *by partial scheduling*.

Detailed Scheduling. There are a number of ways one may choose to solve the local scheduling problem ($DS_{\mathcal{P}}$) on the preprocessed graph $G_{\mathcal{P}}$. To streamline the analysis and allow for testing, we propose a simple approach which uses a *dynamic dispatching* method to generate the detailed schedule. That is, we make scheduling (dispatching) decisions at each point in time when an operation finishes its processing and a number of *candidate operations* are available for processing. This candidate list is obviously dictated by the current set of precedence constraints $A \cup \sigma_{\mathcal{P}}$. We compute a heuristic ranking for each candidate and then dispatch the candidate with the highest ranking. This process is repeated until the end of the planning

horizon. An example of such a dynamic dispatching heuristic for weighted tardy problems can be found in Vepsalainen and Morton (1987).

Further, suppose we define *dynamic scheduling* as a method that makes scheduling (dispatching) decisions at each point in time when an operation finishes its processing, and *static scheduling* as a method that generates a detailed operations plan (sequence) at the beginning of a planning horizon (as an effort to "optimize" global performance). Then the following observations can be made.

1. If we assign all operations to one subset (i.e., set $\alpha_s \leftarrow |N|$), do not apply any preprocessing steps, and generate the schedule over time using *dynamic dispatching*, then the PFSL scheme is exactly a *dynamic scheduling* method.

2. If we limit the size of each subset in $\mathcal{R}$ to 1 (set $\alpha_s \leftarrow 1 \forall s$) and solve (OAP), then the PFSL scheme is exactly *static scheduling*.

3. By adjusting the preprocessing steps, for instance, impose specific size constraints α_s on the subsets in $\mathcal{R}$, the PFSL scheme can be made arbitrarily "close" to *static* or *dynamic scheduling*.

It can be seen that the PFSL approach is a generalization, of which the *static*, and the *dynamic scheduling* methods are both special cases.

3. A BRANCH-AND-BOUND ALGORITHM FOR GRAPH PREPROCESSING

In the following we develop a branch and bound algorithm for the preprocessing of disjunctive graph G. We combine the notion of *preprocessing by assignment*, and *preprocessing by partial scheduling* as integral parts of one algorithm. In the following, we first develop lower and upper bounds for the scheduling subproblem ($DS_{\mathcal{P}}$), which will be used in the branch-and-bound algorithm.

3.1. Lower Bounds for Problem ($DS_{\mathcal{P}}$)

We first consider the lower bound for problem ($DS_{\mathcal{P}}$), the scheduling subproblem resulting from *preprocessing by assignment*. Given preprocessing $_{\mathcal{P}}$, we seek a lower bound for the scheduling problem corresponding to disjunctive graph $G_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}}, E_{\mathcal{P}})$. An obvious lower bound can be computed by considering the acyclic (directed) graph $D_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}})$, which is generated by dropping all the disjunctive arcs $E_{\mathcal{P}}$ from $G_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}}, E_{\mathcal{P}})$. We then compute a lower bound for the starting times (t_i) of each operation i. Let $L(i, j; G)$ denote the longest (node) path distance between nodes i and j in graph G, excluding the processing times d_i and d_j. The (start time) lower bound can be computed as follows:

$$\mathrm{LB1}(t_i) = L(0, i; D_{\mathcal{P}}). \tag{3.1}$$

Given (3.1), a lower bound on total weighted tardiness can then be computed as follows:

$$\mathrm{LB1}(\mathrm{WT}) = \sum_{i \in C} u_i(\mathrm{LB1}(t_i) + d_i - dd_i)^+. \tag{3.2}$$

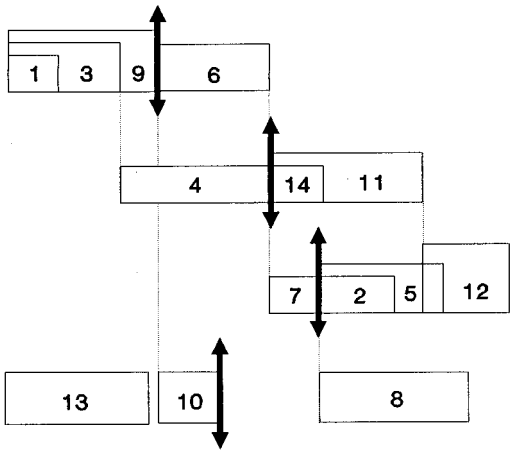

Figure 2(a). Gantt-Chart representation of ordered assignment.

Recall that C is the set of final operations for each job. For example, if we were to compute a lower bound for the assignment given in Figure 1c, we will first compute the start-time lower bound for operations 2, 5, 8, 12, and 14 using (3.1), then apply (3.2). As indicated by Adams et al. (1988), the longest path algorithm can be implemented with $O(n)$ for this specific graph structure.

LB1 can be quite weak since the disjunctive arc set $E_{\mathcal{P}}$ is ignored entirely. We can strengthen the bound by considering the operations in $E_{\mathcal{P}}$ that are competing for the same machine. Suppose we put all operations in digraph $D_{\mathcal{P}}(N, A \cup \sigma_{\mathcal{P}})$ along the time axis, grouped by machines, the result is a Gantt-Chart-like representation of the schedule. Figure 2a, for example, shows such a graph given the assignment in Figure 1c. In the figure, the starting times of each operation are given from the longest path computation in (3.1). Some operations overlap each other since their processing sequence remains undecided (as defined by $E_{\mathcal{P}}$). The vertical arrows represent the result of an ordered assignment. For instance, on machine 1, operations 1, 3, and 9 are assigned a subset preceding the subset

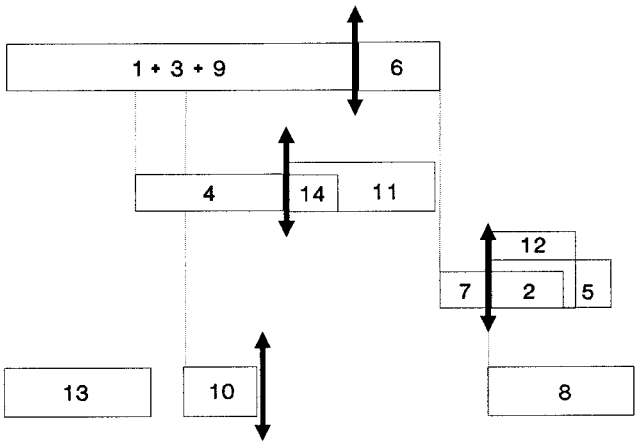

Figure 2(b). Gantt-Chart representation of lower bound 2 (LB2).

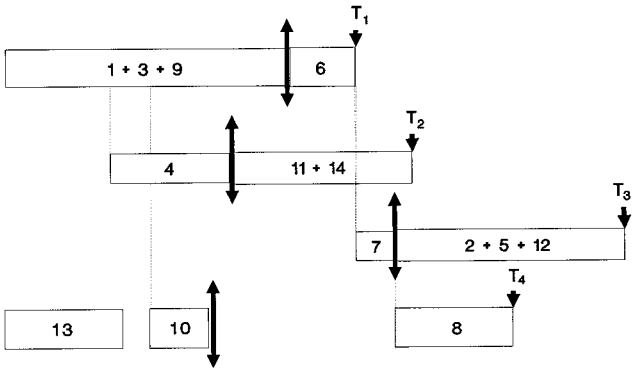

Figure 2(c). Gantt-Chart representation of lower bound 3 (LB3).

containing 6; therefore, 1, 3, 9 precede 6. Similarly, operation 4 precedes 11 and 14; 7 precedes 2, 5, and 12; 10 and 13 precede 8. The vertical dotted lines in the figure show precedence constraints from set A.

Given the above representation, it is not too difficult to see that operation 6 cannot start before 1, 3, and 9 are completed, i.e., $t_6 \geqslant \text{Min}\{t_1, t_3, t_9\} + d_1 + d_3 + d_9$. In more general terms, we can define a lower bound on operation starting times as follows. Denote B_i as the set of operations that are assigned to a subset immediately preceding operation i, and that share the same machine. Thus,

$$\text{LB2}(t_i) = \text{Max}\left\{ \text{LB1}(t_i), \min_{j \in B_i} \{\text{LB2}(t_j)\} + \sum_{j \in B_i} d_j \right\}. \tag{3.3}$$

A tighter lower bound on the total weighted tardiness, LB2(WT), can be thus obtained by applying (3.2) with the term LB1(t_i) replaced by LB2(t_i). This is expressed as follows:

$$\text{LB2(WT)} = \sum_{i \in C} u_i (\text{LB2}(t_i) + d_i - dd_i)^+. \tag{3.4}$$

Obviously, LB2(WT) $\geqslant$ LB1(WT) since completion times are at least as great in LB2. In Figure 2b, the implication of LB2 can be clearly seen. For instance, LB2(t_8) = Max{$L(0, 8; D_{\mathcal{P}})$, Min{t_{10}, t_{13}} + $d_{10} + d_{13}$} = $L(0, 8; D_{\mathcal{P}})$ and LB2(t_6) = Max{$L(0, 6; D_{\mathcal{P}})$, Min{t_1, t_3, t_9} + $d_1 + d_3 + d_9$} = $d_1 + d_3 + d_9$.

Following the logic used to obtain LB2, an alternative lower bound may be computed by considering the earliest "machine completion times." From Figure 2b, it can be seen clearly that machine 3, for example, will not finish processing jobs until operations 2, 5, and 12 are all completed. Denote T_k the earliest completion time for machine k, thus we have $T_3 \geqslant \text{Min}\{t_2, t_5, t_{12}\} + d_2 + d_5 + d_{12}$. More generally, let L_k be the last subset of operations on machine k, i.e.,

$$L_k = \{j \mid \forall (i, j) \in E_k, (i, j) \in (\sigma_{\mathcal{R}} \cap E_k) \text{ and } (j, i) \notin (\sigma_{\mathcal{R}} \cap E_k)\},$$

then we have

$$T_k \geqslant \min_{i \in L_k} \{\text{LB2}(t_i)\} + \sum_{i \in L_k} d_i.$$

By observing machine 3 in Figure 2b, the following condition is also true:

$T_3 \geqslant \text{Max}\{t_2 + d_2, t_5 + d_5, t_{12} + d_{12}\}$. In general, T_k can be tightened as follows:

$$T_k = \text{Max}\left\{ \min_{i \in L_k} \{\text{LB2}(t_i)\} + \sum_{i \in L_k} d_i, \max_{i \in L_k} \{\text{LB2}(t_i) + d_i\} \right\}. \tag{3.5}$$

We can then compute a new lower bound for the total weighted tardiness as follows:

$$\text{LB3(WT)} = \sum_{k \in M} (\min_{i \in C_k} \{u, (T_k - dd_i)^+\}), \tag{3.6}$$

where $C_k \subseteq C$ is the set of last operations on machine k. Note that the last operation of a job is not necessarily in the last subset of a machine (L_k). In the example, $C_1 = \emptyset$, $C_2 = \{14\}$, $C_3 = \{8\}$ and $C_4 = \{2, 5, 12\}$, thus LB3 can be computed as follows:

$$\text{LB3(WT)} = u_{14}(T_2 - dd_{14})^+ + u_8(T_3 - dd_8)^+ + \text{Min}\{u_2(T_4 - dd_2)^+, u_5(T_4 - dd_5)^+, u_{12}(T_4 - dd_{12})^+\}.$$

The computation of LB3 is illustrated in 2c. Note that LB3 and LB2 do not dominate each other. Therefore, we set

$$\text{LB(WT)} = \text{Max}\{\text{LB2(WT)}, \text{LB3(WT)}\}. \tag{3.7}$$

3.2. Upper Bound for Problem ($DS_{\mathcal{P}}$)

An upper bound can be computed for each subproblem ($DS_{\mathcal{P}}$) by simply completing the partially defined disjunctive graph $G_{\mathcal{P}}$ using a scheduling heuristic. In this paper, we use the well known ATC heuristic proposed by Vepsalainen and Morton (1987) for the upper bound calculation. The ATC heuristic has been shown to deliver the best performance for weighted tardy job shop scheduling problems (Kutanoglu and Sabuncuoglu 1994) when compared with a large family of dispatching heuristics.

3.3. The Branching Scheme

Before describing the branching scheme we first examine the cost function $c(\mathcal{P})$. $c(\mathcal{P})$ represents the minimum possible weighted tardiness of the JSP given the precedence constraints $A \cup \sigma_{\mathcal{P}}$ imposed by preprocessing. Obviously, exact evaluation of $c(\mathcal{P})$ is complex since it requires a complete specification of the optimal schedule on $G_{\mathcal{P}}$, which is NP-hard. Alternatively, we compute for each candidate solution $\mathcal{P}$ lower and upper bounds (for $c(\mathcal{P})$) corresponding to the detailed scheduling problem ($DS_{\mathcal{P}}$). The lower and upper bound pair indicates a specific range of scheduling performance given the assignment. It is important to note that since the lower and upper bound pair of

the *scheduling problem* ($DS_{\mathcal{P}}$) were used in lieu of the exact cost $c(\mathcal{P})$, the lower bounds do not necessarily equal the upper bounds when B&B terminates. Consequently we are left with a set of unpruned "nondominated" solutions. These nondominated solutions form a super set that contains all optimal solutions.

We now describe an implicit enumeration scheme for the preprocessing of disjunctive graph G. We consider a special construction where two subsets ($p = 2$) with sizes α_1 and α_2 are used. This is equivalent to considering the assignment of operations to the first subset, since operations not in subset 1 are assumed to be a member of subset 2. We investigate a number of different combinations of subset sizes [α_1, α_2] and their impact on overall performance.

Each node in the B&B tree represents a partial assignment of operations to e_1 without exceeding the subset size α_1, and lower and upper bounds can be computed for each node as specified in Sections 3.1 and 3.2. Each node in the B&B tree is assigned an unique index i. For each node i we associate a string: (i, pred(i), σ_i, $o_1, \ldots, o_j, \ldots, o_n$) where pred($i$) is the index of i's immediate predecessor node (pred(root) = *nil*), n is the total number of jobs, and σ_i contains the set of arcs selected (fixed) by the partial assignment at node i and all its predecessor nodes. The value of o_j is the number of operations from job j that have currently been assigned to e_1. For example, in a five-job problem (as in Figure 1), a node with string (4, 2, σ_4, 1, 1, 0, 1, 0) represents that its index is 4, its immediate predecessor index is 2, the set of fixed disjunctive arcs is σ_4, and one operation (the first) from jobs 1, 2, and 4 are currently assigned to set e_1. A node with string (i, _, σ_i, $o_1, \ldots, o_j, \ldots, o_n$) generates up to n children as follows: (_, i, _, $o_1 + 1, \ldots, o_j, \ldots, o_n$), ..., (_, i, _, $o_1, \ldots, o_j + 1, \ldots, o_n$), ..., and (_, i, _, $o_1, \ldots, o_j, \ldots, o_n + 1$). For instance, string (5, 3, σ_5, 1, 1, 0, 1, 0) would generate the following children nodes: (22, 5, σ_{22}, 2, 1, 0, 1, 0), (23, 5, σ_{23} 1, 2, 0, 1, 0), (24, 5, σ_{24}, 1, 1, 1, 1, 0), (25, 5, σ_{25}, 1, 1, 0, 2, 0) and (26, 5, σ_{26}, 1, 1, 0, 1, 1), where $\sigma_{22} \cap \sigma_{23} \cap \sigma_{24} \cap \sigma_{25} \cap \sigma_{26} = \sigma_5$. The above encoding guarantees that each node satisfies the precedence constraints since the operations of each job are assigned in order. Further, this encoding maintains a monotonically nondecreasing lower bound for the B&B tree since a child node inherits all the disjunctive arcs fixed by its predecessor nodes.

3.4. The Enumeration Scheme

If one views the B&B algorithm as a means of preprocessing the disjunctive graph, the above algorithm combines the notion of *preprocessing by assignment* and *preprocessing by partial scheduling*. Note that a leaf node in the B&B tree contains not only all the assignment decisions of its branch, but also the sequence of which these decisions are made (as stored in σ_i). This sequence determines the directions for all the disjunctive arcs in the first subset. As a result, the B&B algorithm enumerates not only the ordered assignment decisions but also scheduling decisions of the first subset.

Several tree search strategies are implemented to improve pruning efficiency of the B&B algorithm. First, the current best upper bound is kept as the overall upper bound. To tighten the bounds at the beginning of the search, the algorithm heuristically chooses k "promising" branches with the lowest node upper bounds, quickly reaches their leaf nodes, then passes back their lower and upper bounds. The value of k is an algorithm parameter to be determined empirically. The B&B procedure terminates when no more branching is possible. As stated earlier, when the B&B terminates we get a set of unpruned nondominated solutions. These nondominated solutions form a super set that contains all optimal assignments, therefore all optimal schedules.

4. COMPUTATIONAL EXPERIMENTS

We implemented the branch-and-bound procedure in FORTRAN 77 on an IBM RS-6000 workstation. Fifteen 5×10 and eighteen 5×20 weighted tardy test problems (i.e., $A \times B$ represents A jobs, B machines, each job has B operations, one on each machine) were generated from the makespan problems in Applegate and Cook (1991). Each 5×10 problem was obtained from the 10 machine test problems [LA21–LA35] by taking operations from the first five jobs. The 5×20 problems were obtained from the 10×10 problems by combining the operations of two jobs into one (i.e., extending the routing of job 1 with the routing of job 6, the routing of job 2 with that of job 7, and so on). The job due-dates were generated by first obtaining a nondelay schedule from the problem using random dispatching, finding the completion times for each job (in set C) in the schedule, then setting the job due-date equal to (*job completion time* − 100). Job weights were generated from an integer uniform [1, 10] distribution. This particular problem generation scheme creates rather challenging, medium-tight due-date problems. All test problems are available upon request.

We tested different combinations of subset sizes (α). For 5×10 problems we considered three different subset sizes: [15, 35] [25, 25] and [35, 15] (i.e., [α_1, α_2] represents α_1 operations in the first subset and α_2 operations in the second). A 15,000 limit on the number of open nodes was imposed on the B&B memory. Note that this node limit does not represent the total number of nodes in the B&B tree, rather, it represents the total number of unpruned nodes allowed in the memory at any one time. Problems which exceed the node limitation during run time were terminated. The solutions for these problems will be legitimate assignments although suboptimal. The testing of 5×20 problems was similar to the 5×10 case except we consider subset sizes: [15, 85], [25, 75]. Because of a higher memory requirement on each node (i.e., a larger size disjunctive graph), a 10,000 node limit was imposed on the B&B memory.

Table I
Performance of PFSL in Deterministic Cases

Splits	% Outperform IATC	B&B basic steps	CPU sec	IATC basic steps	CPU sec	Node limit reached
Averaged Over 15 (5 × 10) Problems[a] (B&B memory limitation is 15,000 open nodes)						
[15, 35]	38.0%	53,269	1,257.1	21,000	51.3	4/15
[25, 25]	25.8%	201,244	10,689.6	same	same	7/15
[35, 15]	21.0%	239,029	16,006.8	same	same	13/15
Averaged Over 18 (5 × 20) Problems[a] (B&B memory limitation is 10,000 open nodes)						
[15, 85]	43.5%	34,013	763.3	21,000	147.3	15/18
[25, 75]	5.8%	46,856	3578.6	same	same	18/18

[a]Test problems are generated from the makespan problems in Applegate and Cook (1991):
5 × 10 problems tested: generated from [LA21] to [LA35].
5 × 20 problems tested: generated from [ABZ5], [ABZ6], [LA16]–[LA20], [MT10], and [ORB1]–[ORB10].
Splits: [α_1, α_2] where α_1 and α_2 represent the size of the first and second subset, respectively.
% outperform IATC: the value is calculated as (v(IATC) − v(B&B))/v(B&B), where v(B&B) is the best upper bound when B&B terminates, and v(IATC) is the weighted tardiness value when IATC terminates.
B&B basic steps: the total number of B&B nodes used, or equivalently, the total number of calls to the ATC heuristic (for upper bound calculation). CPU time is in seconds on IBM RISC 6000 workstation.
IATC basic steps: the total number of calls to the ATC heuristic. Each run of IATC consists of 7 step sizes, 3 lead-time values and 1000 iterations [Byeon et al. 1993]. CPU time is based on a IBM RISC 6000 workstation.
Node limit reached: number of times B&B reaches its open node limit.

When B&B terminates, it returns a number of nondominated solutions (i.e., solutions that can not be fathomed), each represents a preprocessed disjunctive graph $G_{\mathcal{P}}$. In our experiments, we assume that the remaining scheduling decisions are completed eventually using the Vepsalainen-Morton ATC heuristic. This can be viewed as a specific implementation of the PFSL scheme where the B&B algorithm preprocesses the disjunctive graph using ordered assignment and complete scheduling of the first subset, while the ATC heuristic makes the remaining scheduling decisions.

4.1. Performance Under A Deterministic Shop Condition

We first test the PFSL scheme under a deterministic shop condition in which no shop disturbances were present after the partial schedule is released. We use the best upper bound among nondominated solutions as the performance measure. For comparison purpose, we implemented an iterative improvement version of the ATC heuristic (Byeon et al. 1993). To the best of our knowledge, the ATC heuristic is among the best for weighted tardiness JSP. Our iterative implementation of ATC (i.e., IATC) has shown an average of 56.4% improvement over the original ATC in the problems tested. Each run of IATC uses 7 step sizes, 3 lead time values, and 1,000 iterations, or 21,000 calls to the ATC heuristic. Unlike ATC, IATC can be implemented only in a static fashion where the entire a priori schedule is generated and fixed at the beginning of the planning horizon. We use IATC as a benchmark to gage the quality of the nondominated solutions. Later in testing the stochastic cases, IATC will be used as the static scheduling method.

Table I summarizes a comparison where we compute the margin by which the best UB among nondominated B&B solutions outperforms the IATC heuristic. As shown in the table, the quality of the nondominated solutions are very high in general. This result is important since in essence the PFSL scheme preprocesses the problem graph before applying the ATC heuristic. More interestingly, subset size has a noticeable impact on performance. The split with a smaller first subset (i.e., [15, 35] and [15, 85]) consistently performs best.

As can be observed from the table, node limitation has a significant impact on the CPU time. For 5 × 20 problems, the node limitation was set lower at 10,000 due to a higher memory requirement per node. As a result the node limitation was reached more frequently, which means the algorithm terminates before completion. Although the solutions returned by B&B are suboptimal, performance remains quite high when compared to IATC.

To determine if the difference between B&B and IATC is merely the result of more computation time spent, we compare the number of calls to the ATC heuristic (i.e., the number of basic steps) the B&B algorithm made. One observes from the table that this number does not correspond to the quality of the B&B solutions. We tried further 5000-iteration (or 105,000 calls to ATC) runs on IATC. In all test cases IATC does not make any further improvement after the first 1000 iterations.

4.2. Performance Under Stochastic Conditions: Scheduling Robustness

We now demonstrate the robustness of the PFSL scheme under stochastic conditions. We conducted a set of Monte

Carlo experiments aimed at testing the robustness of the partial schedules generated by B&B when shop disturbances are present. We used a variety of random processing time variations to represent disturbances. Due to the dimension and the computational requirement of the simulation, we will restrict our robustness experiments to 5×10 test problems [LA21] to [LA29] using a [15, 35] split.

Robustness Measures. Multiple nondominated solutions are produced by B&B, thus we need a way to select a specific assignment. In the experiments we use a surrogate measure defined based on a linear combination of the lower (LB) and the upper bound (UB) of each nondominated solution with weight β ranging from 0 to 1. LC(β) can be described as follows:

$$\mathrm{LC}(\beta) = (1 - \beta) \cdot \mathrm{LB} + \beta \cdot \mathrm{UB}.$$

We tested β values 0.25, 0.33, 0.41, 0.5, 0.625, 0.75, and 0.875. For each proposed robustness measure, we calculated their correlation with a simulated expected weighted tardiness, E[WT] (estimated using the average weighted tardiness value over 100 simulation replicates under random processing times).

Experimental Results. Next we conducted simulation experiments to assess the robustness of the proposed PFSL scheme. For each test problem, we first select a single assignment from the nondominated solutions in the B&B tree using the measure LC(β), where β is determined by the correlation study according to shop disturbance conditions. Associated with the selected nondominated solution $\mathcal{P}$ is a partial schedule defined by disjunctive graph $G_{\mathcal{P}}$. To simulate the performance of this partial schedule defined by $G_{\mathcal{P}}$ under shop disturbance conditions, we complete the remaining scheduling decisions over time using the dynamic dispatching heuristic ATC. Note that these decisions are not made until the time has come for job dispatching. As a result, all the up-to-the-minute processing time variations will be taken into account in job dispatching. These results are then compared with the simulation of two conventional scheduling methods under the same set of shop disturbances. These two methods are *dynamic dispatching* using ATC, and *static scheduling* using IATC. In static scheduling, we handle the disturbances by following the operations sequence (i.e., not starting times) specified by the IATC schedule.

To generate a variety of shop disturbances we perturb operation processing times based on Uniform and Double Exponential distributions. Perturbations are generated in two ways termed "dependent" and "independent." Specifically, four different patterns of disturbances were generated, each at 12 levels of variation. Let d_i be the original processing times assumed when generating static schedules and partial schedules in the PFSL scheme. Let d_i' be the actual processing times observed as the schedule unfolds. Note that for all the test problems, the original processing times d_i were generated from a Discrete Uniform [1, 100] distribution. The actual observed d_i' are generated from the original d_i as follows.

Case 1. "Dependent Uniform" (DU) disturbances: processing times are perturbed based on a Uniform distribution; the amount of perturbation is dependent on the original processing time as well as the level of variation. Letting U be a Uniform $(-1, 1)$ random variate, the d_i' are generated using:

$$d_i' = d_i + d_i \cdot p_1 \cdot U$$

where $p_1 = 0.05, 0.10, \ldots, 0.60$ (variation levels 1 to 12).

Case 2. "Independent Uniform" (IU) disturbances: processing times are perturbed based on a Uniform distribution; the amount of perturbation is independent of the original processing times. Negative actual processing times are set equal to zero.

$$d_i' = \mathrm{Max}\{0,\ d_i + p_2 \cdot U\}$$

where $p_2 = 5, 10, \ldots, 60$ (variation levels 1 to 12).

Case 3. "Dependent Double Exponential" (DDE) disturbances: processing times are perturbed based on a Double Exponential distribution; the amount of perturbation is dependent on the original processing time. Negative actual processing times are set equal to zero.

$$d_i' = \mathrm{Max}\{0,\ d_i + z \cdot d_i \cdot \mathrm{Exp}(1/\tau_1)\}$$

where $\Pr\{z = 1\} = \Pr\{z = -1\} = 0.5$, $\mathrm{Exp}(1/\tau_1)$ is an exponentially distributed random variate with mean τ_1, and $\tau_1 = 0.05, 0.1, \ldots, 0.6$ (variation levels 1 to 12).

Case 4. "Independent Double Exponential" (IDE) disturbances: processing times are perturbed based on a Double Exponential distribution independent of the original processing times. Negative actual processing times are set equal to zero.

$$d_i' = \mathrm{Max}\{0,\ d_i + z \cdot \mathrm{Exp}(1/\tau_2)\},$$

where $\Pr\{z = 1\} = \Pr\{z = -1\} = 0.5$, $\mathrm{Exp}(1/\tau_2)$ is an exponentially distributed random variate with mean τ_2, and $\tau_2 = 5, 10, \ldots, 60$ (variation levels 1 to 12).

These four methods of processing time perturbation represent distinctly different patterns of shop disturbances. For instance, it is clear that more variation in processing times exists in the Double Exponential cases than in the Uniform cases. Further, a greater variation can be expected in the independent cases as compared to the dependent cases. The possibility of negative actual processing times that are subsequently set to zero exists in all but the DU case. This in turn causes expected total processing time to be greater for the actual processing times than for the original times. This effect is clearly most pronounced in the IDE case. It is important to note that operations with zero processing times still had to be scheduled in the simulations. That is, successors of a zero processing time operation could not be started until the zero processing time operation had been completed.

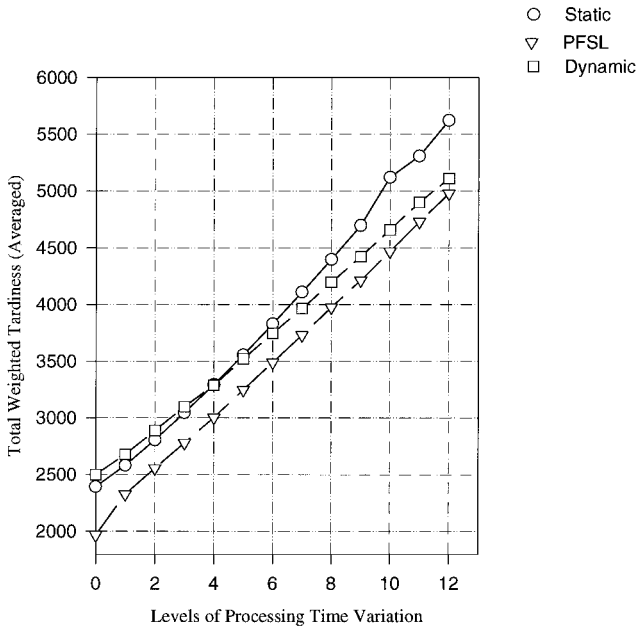

Figure 3. Simulation results for dependent uniform disturbances.

Simulation results under the four cases of disturbances DU, IU, DDE, and IDE are shown in Figures 3, 4, 5, and 6 respectively. The x-axis represents different levels of processing time variation where Level 0 corresponds to the deterministic case. Each point in the plot represents total weighted tardiness averaged over the test problems [LA21]–[LA29] where 100 simulation replicates were used for each test problem. The following observations can be made from the simulation results.

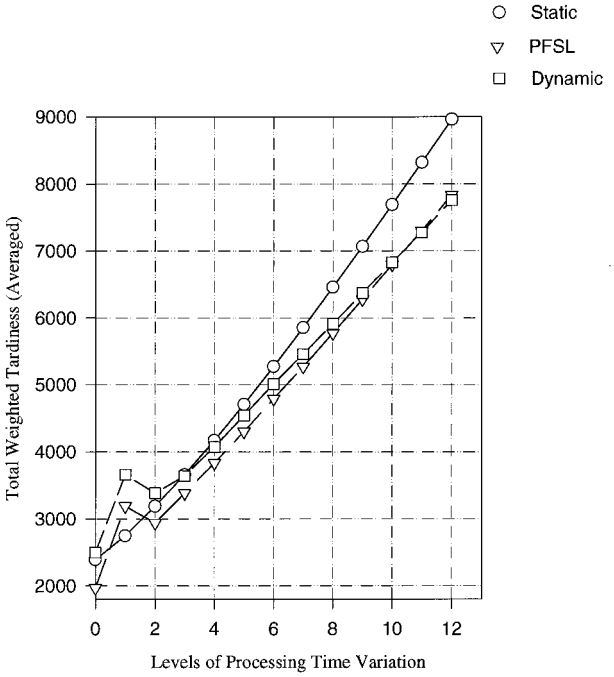

Figure 4. Simulation results for independent uniform disturbances.

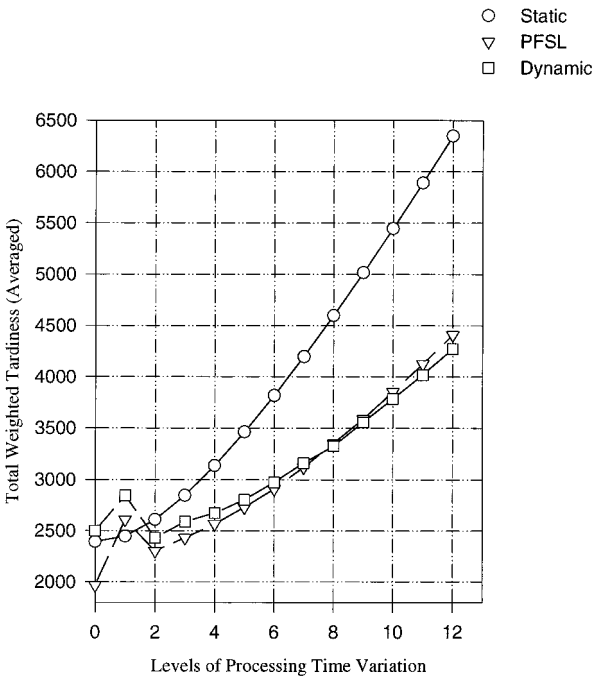

Figure 5. Simulation results for dependent double exponential disturbances.

1. While the overall performance of the PFSL scheme is very good compared to the static and the dynamic methods, it is particularly superior in the cases of dependent uniform (DU) disturbances (Figure 3). Its performance is good even when a 60% perturbation is introduced. We have observed noticeable variation among different test problems, nonetheless, the PFSL solution shows clear dominance in 6 out of 9 test problems. As shown in Figure

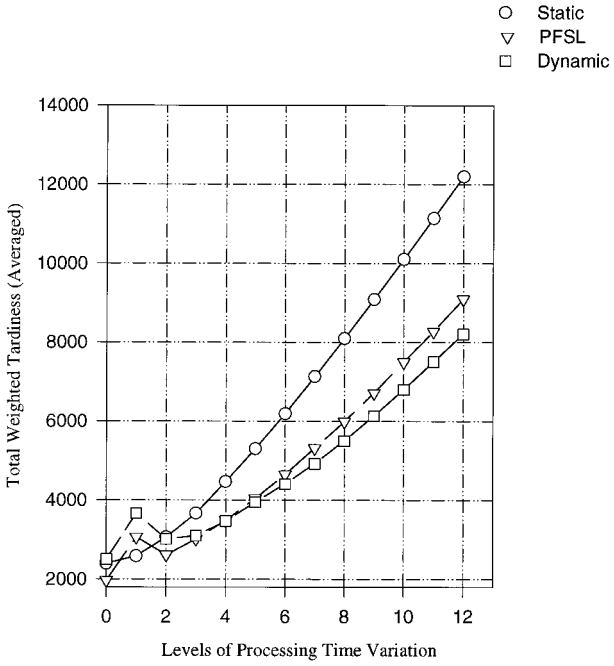

Figure 6. Simulation results for independent double exponential disturbances.

3, the average performance of PFSL dominates both the static and the dynamic methods.

2. For independent uniform (IU) disturbances (Figure 4), the performance of PFSL remains superior. When compared with the static method, PFSL shows a better average performance at all levels of processing time variation. When compared with the dynamic method, PFSL shows better average performance up to a fairly high level of processing time variation (i.e., level 9). For levels 10–12, the performance of PFSL and dynamic remain close.

3. For dependent double exponential (DDE) disturbances (Figure 5), the performance of PFSL solutions remain quite consistent but the performance of the static method deteriorates significantly. This particular set of results demonstrates the unstable nature of a priori static scheduling methods.

4. For independent double exponential (IDE) disturbances (Figure 6), the weighted tardiness increases sharply for all three methods. It is interesting to see that even under these conditions the PFSL solutions are reasonably robust. As discussed previously, higher values of the Exponential mean τ_2 generate chaotic shop conditions. Not surprisingly, as the mean goes up, dynamic dispatching outperforms the other two methods. Note, however, that the performance of the dynamic approach suffers at a lower level of processing time variation.

5. Most importantly, throughout the experiments the PFSL solutions show remarkable consistency and stability over a wide range of disturbances and test cases. By comparing the results in Figures 3 to 6, one gets a sense of the robustness of the PFSL scheme as the pattern of disturbance changes. For the cases IU, DU and DDE, the PFSL scheme's performance is superior to that of dynamic dispatching. For the IDE case, the dynamic method performs better. This provides a fairly clear picture of the magnitude of disturbances which the PFSL scheme can tolerate. The results of these experiments are very encouraging and at the same time revealing. The "comfort zone" identified for the PFSL scheme, fortunately, corresponds to the conditions in practice, which call for more careful planning and scheduling. Specifically, these are cases where low to moderate disturbances are experienced in the shop throughout the unfolding of the schedule. Obviously, in the cases where chaotic disruptions are the norm, a priori planning is of little use, and dynamic scheduling would be a better choice.

5. CONCLUSIONS

We developed a decomposition for job shop scheduling problems based on an ordered assignment of job operations to subsets. Some basic properties of the decomposition were examined. The method can be viewed as extracting crucial scheduling decisions by means of ordered assignment. A *preprocess first schedule later* scheme was proposed which makes these crucial decisions through a branch and bound algorithm while leaving the remaining scheduling decisions to be made dynamically. Computational experiments were conducted to test the effectiveness of the scheme. We first considered a deterministic case where no disturbances are present in the shop. We found that the preprocessing significantly improves the traditional scheduling heuristic. We then considered stochastic cases where various levels of shop disturbances are examined. Through a set of Monte Carlo simulations, we demonstrated that the PFSL scheme indeed provides more robust performance under a wide range of disturbances as compared to traditional static and dynamic scheduling methods. For moderate disturbances, PFSL showed particularly superior performance.

ACKNOWLEDGMENT

We thank the associate editor and two anonymous referees for many insightful comments and suggestions. The research is supported in part by the National Science Foundation Grant SBR 94-22862.

REFERENCES

Adams, J., E. Balas, D. Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Sci.* **34,** 391–401.

Applegate, D., W. Cook. 1991. A computational study of the job-shop scheduling problem. *ORSA J. Computing* **3,** 149–156.

Balas, E. 1979. Disjunctive programming. *Ann. Discrete Math.* **5,** 3–51.

Bean, J. C., J. R. Birge, J. Mittenthal, C. E. Noon. 1991. Matchup scheduling with multiple resources, release dates and disruptions. *Oper. Res.* **39,** 470–483.

Birge, J., M. Dempster. 1987. Optimality conditions for match-up strategies in stochastic scheduling and related dynamic stochastic optimization problems. Working Paper. Department of Industrial Engineering and Operations Engineering, The University of Michigan, Ann Arbor, MI.

Bitran, G. R., A. C. Hax. 1977. On the design of hierarchical production planning systems. *Decision Sci.* **8,** 28–54.

Byeon, E. S., S. D. Wu, R. H. Storer. 1993. Decomposition heuristics for robust job-shop scheduling. Technical Report 93T-008. Department of Industrial Engineering, Lehigh University, Bethlehem, PA.

Carlier, J., E. Pinson. 1989. An algorithm for solving the job-shop problem. *Management Sci.* **35,** 164–176.

——, ——. 1994. Adjustment of heads and tails for the job-shop problem. *Euro. J. Oper. Res.* **78,** 146–161.

Gallego, G. 1988a. Linear control policies for scheduling a single facility after an initial disruption. Technical Report No. 770. School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

——. 1988b. Produce-up-to policies for scheduling a single facility after an initial disruption. Technical Report No. 771. School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

Hax, A. C., H. C. Meal. 1975. Hierarchical integration of production planning and scheduling: M. Geisler (ed.). *TIMS Studies in Management Sciences: Vol. 1, Logistics.* North-Holland/American Elsevier, NY.

Holt, C. C., F. Modigliani, J. F. Muth, H. A. Simon. 1960. *Planning, Production Inventories and Work Force.* Prentice-Hall, Englewood Cliffs, NJ.

Kutanoglu, E., Sabuncuoglu, I. 1994. Experimental investigation of scheduling rules in a dynamic job shop with weighted tardiness costs. *Proc. 3rd Industrial Eng. Res. Conference*, Atlanta, GA.

Leon, V. J., S. D. Wu, R. H. Storer. 1994. Robustness measures and robust scheduling for job shops." *IIE Trans.* **26,** 32–43.

Morton, T. E., S. Lawrence, S. Rajagopalan, S. Kekre. 1986. MRP-STAR PATRIARCH's planning module. Working Paper. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

——, ——, ——, ——. 1988. SCHED-STAR. A price-based shop scheduling module. Working Paper. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

Rinnooy Kan, A. H. G. 1976. *Machine Scheduling Problems: Classification, Complexity and Computations*. Nijhoff, The Hague.

Roundy, R., W. Maxwell, Y. Herer, S. Tayur, A. Getzler. 1991. A price-directed approach to real time scheduling of production operations. *IIE Trans.* **23,** 149–160.

Vepsalainen, A. P. J., T. E. Morton. 1987. Priority rules for job shops with weighted tardiness costs. *Management Sci.* **33,** 1035–1047.

Winters, P. R. 1962. Constrained inventory rules for production smoothing. *Management Sci.* **8,** 470–481.

Wu, S. D., R. H. Storer, P. C. Chang. 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. *Comp. Oper. Res.* **20,** 1–14.

——, ——, ——. 1992a. A rescheduling procedure for manufacturing systems under random disruptions. *New Directions for OR in Manufacturing*. Springer-Verlag.

——, V. J. Leon, R. H. Storer. 1992b. Scheduling, control and rescheduling methodologies for job shops in the presence of disruptions. *Proc. 1992 NSF Design and Manufacturing Systems Conference*, Atlanta, GA, 947–959.