Improving Schedule Robustness via Stochastic Analysis and Dynamic Adaptation

Erhan Kutanoglu
S. David Wu

Manufacturing Logistics Institute
Department of Industrial and Manufacturing Systems Engineering

Lehigh University

Improving Schedule Robustness via Stochastic Analysis and Dynamic Adaptation

by

Erhan Kutanoglu and S. David Wu

Manufacturing Logistics Institute
Department of Industrial and Manufacturing Systems Engineering

Lehigh University

Abstract

We study methods that improve scheduling robustness under uncertain disturbances and dynamic shop conditions. A new method is proposed based on the notion of preprocess-first-schedule-later by Wu et al. [19]. Preprocessing starts at the beginning of the planning period where we use a Lagrangean Relaxation of the scheduling model to form network-structured job subproblems. For each job subproblem we introduce stochastic constraints which capture *a priori* information in the form of processing time uncertainty. This stochastic information comes at minimal computational expenses since the subproblems retain their special network structure. Using a subgradient search algorithm, we iteratively improve the lower and upper bounds of the scheduling instance. The preprocessing produces a partially resolved sequence or a *Lagrangean Ranking*. Actual schedule generation is performed dynamically over time using the*Lagrangean ranking*. Intensive computational experiments show that the proposed preprocessing significantly outperforms its deterministic counterparts without extra computing burden, achieves robust performance under highly uncertain conditions, and could be used to improve the quality of dynamic dispatching.

1 Introduction

We study scheduling robustness in a production environment where frequent changes in shop conditions are to be expected. These uncertain events complicate the role of production scheduling since two conflicting goals must be reconciled: first, the system must perform at a globally satisfactory level for efficient resource usages, and second, the system must allow sufficient local flexibility for changes and adjustments.

To achieve the first goal of high global performance a well-planned *a priori* schedule is required. In general, *static scheduling* techniques are used to optimize the operations schedule at the beginning of a planning horizon. This approach typically generates a detailed schedule assuming perfect information about the system's current and future states. However, unexpected disturbances make this pre-planned schedule very difficult, if not impossible, to follow. Moreover, overall system performance deteriorates when updates are made for the system changes.

To achieve the second goal of local flexibility, dynamic dispatching policies are typically used which make decisions as needed throughout the course of production. This *dynamic scheduling* approach typically uses myopic priority indices and greedy heuristics which may lead to poor global performance. However, its inherent flexibility and ability to utilize up-to-date local information makes *dynamic scheduling* a practical tool in uncertain environments.

In this study, we propose to address scheduling robustness by a method that reconciles the benefits of *static* and *dynamic* scheduling. We shift the focus of *a priori scheduling* to identifying decisions that are critical to global performance under the presence of random disturbances. These decisions are optimized using *a priori* stochastic information, which in turn preprocess, or partially solve the scheduling problem. The actual schedule unfolds over time using *dynamic scheduling.*

Treating scheduling problems in uncertain environments did not attract much attention in the literature until the early 80s. One approach is to reschedule all the jobs from scratch every time an uncertain event occurs [13, 22, 5, 1, 20, 21]. Researchers also proposed schedule repair methods that compute a temporary schedule after each machine breakdown with the attempt of bringing the system back to schedule in a finite amount of time. The "match-up scheduling" method proposed by Bean et al. [2] and Gallego [7, 8] belongs to this category.

Instead of repairing a pre-specified schedule, Mehta and Uzsoy [10] propose a method that insert idle times into the schedule in anticipation of schedule disruptions. Another approach is to split the scheduling activity into *planning* and *dispatching* stages. The planning stage computes the pricing or resource usage costs using global information, while the dispatching phase makes use of the resource pricing and up-to-date shop conditions and uncertainties. This approach is both general and effective as demonstrated by Roundy, et al. [17], and Morton et al. [11, 12]. A similar approach by Wu et al. [19] partitions the operations into an ordered sequence of subsets in the planning phase. This identifies and resolves the supposedly critical scheduling decisions through a graph-theoretic preprocessing. The second stage completes the schedule dynamically using a dispatching heuristic.

The approach described in this paper is in parallel to the idea of preprocessing scheduling problems using *a priori* information while dynamically generating the actual schedule.

2 Problem Statement and Analysis

2.1 The Mathematical Model

We consider the classical job-shop scheduling model where a set of jobs is to be completed on multiple machines. Each *job* consists of a series of *operations* that represent the production steps of the job. Due to processing requirements the operations of a job are subject to *precedence constraints*. Each production step (an operation) requires a specific *machine* and once the processing of an operation starts it can not be pre-empted by any other operations. The collection of operation *processing times* and their corresponding machine requirements represent the *job routing*. Each machine can only process one operation at a time. This requirement defines the *machine capacity constraints*. Each job has a *due-date* and a *weight*, and the objective is to find a schedule which minimizes the total weighted tardiness.

We first state an integer programming formulation of the job-shop scheduling problem due to Pritsker et al. [16]. The notation is summarized in Table 1. The decision variable of this model (X_{ijt}) is defined as follows:

$$X_{ijt} = \begin{cases} 1 & \text{if operation } j \text{ of job } i \text{ has started by time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The JSP then can be formulated as follows (We set the variables with out-of-range indices to 0):

(JSP)

$$\min \sum_i W_i \left[\sum_{t > d_i - p_{i,n_i}} (1 - X_{i,n_i,t}) \right] \tag{2}$$

Table 1: Notation used throughout the paper

i	Job index, $i = 1, \ldots, N$ where N is number of jobs
j	Operation index, $j = 1, \ldots, n_i$ where n_i is number of operations of job i
k	Machine index, $k = 1, \ldots, M$ where M is number of machines
t	Time index, $t = 1, \ldots, T$ where T is number of time periods in planning horizon
W_i	Weight or tardiness penalty of job i
d_i	Due date of job i
C_i	Completion time of job i
T_i	Tardiness of job i, $\max\{0, C_i - d_i\}$
m_{ij}	Machine required for operation j of job i
p_{ij}	Processing time of operation j of job i
C_{ij}	Completion time of operation j of job i
S_{ij}	Start time of operation j of job i

$$\begin{aligned}
\text{s.t. } X_{i,j,t+1} &\geq X_{i,j,t}, \quad \forall i, j, t < T && (3)\\
X_{i,j,t} &\leq X_{i,j-1,t-p_{i,j-1}}, \quad \forall i, j > 1, t && (4)\\
\sum_{i,j:m_{ij}=k} (X_{i,j,t} - X_{i,j,t-p_{ij}}) &\leq 1, \quad \forall k, t && (5)\\
X_{i,j,t} &\in \{0,1\}, \quad \forall i, j, t && (6)
\end{aligned}$$

The first set of constraints (3) makes sure that once an operation is started, it remains so in all subsequent time periods. This is due to the definition of X_{ijt} and the non-preemption requirement. Constraints (4) state that an operation cannot start until all its predecessors are completed. Finally, the machine capacity constraints (5) state that at most one job can be processed on a particular machine in a given time period. The objective function (2) is the total weighted tardiness derived from the fact that a cost of W_i is incurred for each time period after d_i during which job i has not been completed.

The objective function can be rewritten as follows:

$$\sum_i W_i \left[\sum_{t > d_i - p_{i,n_i}} (1 - X_{i,n_i,t}) \right] = \sum_i W_i (T - d_i + p_{i,n_i}) + \sum_i \sum_j \sum_t -w_{ijt} X_{ijt} \qquad (7)$$

where

$$w_{ijt} = \begin{cases} W_i & \text{if } j = n_i \text{ and } t > d_i, \\ 0 & \text{otherwise.} \end{cases} \qquad (8)$$

Since the first term in the objective function (7) is a constant we represent it as $A = \sum_i A_i$ where $A_i = W_i(T - d_i + p_{i,n_i})$.

In solving scheduling problems we know that the processing time information is among the most critical input. This is particularly true when a *regular* performance measure such as (2) is used. Almost all successful dispatching heuristics rely heavily on processing time information. In our IP formulation, both objective function and constraints are defined based on processing times. Any change in processing times will affect the solution and its corresponding weighted tardiness value. Since processing times are typically uncertain in practice, best estimates or expected values are used in deterministic scheduling models. This common treatment of processing time information ignores the effects of processing time variations, which could clearly lead to poor quality schedules in the face of uncertainty. We will address these important issues in the following sections.

2.2 The Proposed Scheduling Scheme and Broader Managerial Insights

Before stating the scheduling scheme, we first make two observations from the industry scheduling environments: (1) at the time of planning a schedule there is typically some information available concerning future state of the shop, and (2) there is no need for a completely specified schedule for all jobs over the entire planning horizon. Using the available information, a seasoned scheduler can typically pinpoint a few critical decisions to overall schedule efficiency, and focus his/her attention on these decisions. Detailed scheduling decisions are often delayed till later during actual production when up-to-date information is available with much higher certainty.

Motivated by this practical paradigm of decision making we propose a few adaptations to the IP model that changes the role it plays in the scheduling process. Instead of using the (JSP) model to generate detailed static schedules, we use it to analyze the criticality of scheduling decisions based on *a priori* information. Specifically, we modify (JSP) to include *chance constraints*, which captures the effects of processing time variations. After a Lagrangean relaxation of the machine capacity constraints, the chance-constrained program has a special network structure in its subproblems. The solution to this relaxed problem is a partially solved scheduling problem (a lower bound) since many resource conflicts are left unresolved. To improve the lower bound we start iterating on the relaxed problem using a subgradient search algorithm. The algorithm stops when the partial schedule contains sufficient details to be released to the shop for planning purposes (or when the computing budget runs out). We leave the remainder of the scheduling decisions to system dynamics, i.e., we resolve the remaining resource conflicts as needed during the actual implementation of the schedule.

The proposed scheduling scheme can be considered a version of the preprocess-first-schedule-later (PFSL) scheme proposed by Wu et al. [19]. The preprocessing is performed by the Lagrangean-relaxed, chance-constrained mathematical program, while the detailed scheduling is carried out by dynamic dispatching. The PFSL scheme is based on the following conjecture:

Conjecture 1: The robustness of *static scheduling* model such as (JSP) can be significantly improved by preprocessing the problem instance using *a priori* analysis of its structure, while making detailed scheduling decisions dynamically. The extent of the improvement depends on the underlying statistical distribution and the level of uncertainty.

This conjecture has been strongly supported by the experiments in [19] where they use a branch and bound algorithm for the *a priori* analysis and a dynamic dispatching rule for detailed scheduling. After testing under Uniform and Exponentially distributed variations, they show that the marginal value of *a priori* analysis deteriorates as the level of uncertainty increases. Not surprisingly, their preprocessing scheme uses only the structure of the deterministic model, i.e. no stochastic information is used.

Knowing that *a priori* analysis of problem structure may improve scheduling robustness, the goal of the current study is to improve scheduling robustness using *a priori* stochastic information. A question of general managerial interest is "what are the insights one could extract from problem structure in the context of *a priori* stochastic information, and how do they improve scheduling robustness under uncertainty." To address this broader research question, we state three main conjectures which our proposed scheme builds on. These conjectures will be thoroughly tested in the computational experiments.

Conjecture 2: Scheduling robustness can be significantly improved by incorporating not only the mean but also the second moment of the stochastic information. Moreover, the value of this information increases as the level of uncertainty increases.

Conjecture 3: The robustness of *dynamic dispatching* can be significantly improved by *a priori* static analysis using stochastic information. The extent of the improvement varies by problem instances, the type of underlying distributions, and the level of uncertainty.

Conjecture 4: The marginal benefit of *a priori* stochastic analysis (to the PFSL scheme) increases as the level of uncertainty increases, and its effect varies by the type of underlying distribution.

Testing these conjectures provides support (and counter cases) for the belief that one could reconcile the benefits of static analysis and dynamic adaptation, while making effective use of *a priori* stochastic information.

3 Solution Methodology

We will start constructing our solution methodology applying Lagrangean relaxation to the JSP formulation as suggested by [17]. We first relax machine capacity constraints (5), and decomposed the problem into job-level subproblems with integrality property. We then employ chance-constrained programming at the subproblem level which incorporates *a priori* stochastic information.

3.1 Lagrangean Relaxation

When we dualize the machine capacity constraints (5) of JSP with nonnegative Lagrangean multipliers $\lambda_{kt}, \forall k, t$ we obtain the following Lagrangean Relaxation problem (LR):

(LR_λ)

$$\min \sum_i A_i + \sum_i \sum_j \sum_t (\lambda_{m_{ij},t} - \lambda_{m_{ij},t+p_{ij}} - w_{ijt}) X_{ijt} - \sum_k \sum_t \lambda_{kt} \tag{9}$$

$$\begin{aligned} \text{s.t. } X_{i,j,t+1} &\geq X_{i,j,t} \ \forall i, j, t < T && (10) \\ X_{i,j,t} &\leq X_{i,j-1,t-p_{i,j-1}} \ \forall i, j > 1, t && (11) \\ X_{i,j,t} &\in \{0,1\} \ \forall i, j, t && (12) \end{aligned}$$

We decompose LR_λ into independent job-level subproblems as follows:

$$v(LR_\lambda) = \sum_i v(LR_{\lambda,i}) - \sum_k \sum_t \lambda_{kt} \tag{13}$$

where $v(P)$ denotes the value of optimal solution of problem P, and

$$v(LR_{\lambda,i}) = \min A_i + \sum_j \sum_t (\lambda_{m_{ij},t} - \lambda_{m_{ij},t+p_{ij}} - w_{ijt}) X_{ijt} \tag{14}$$

$$\begin{aligned} \text{s.t. } X_{i,j,t+1} &\geq X_{i,j,t} \ \forall j, t < T && (15) \\ X_{i,j,t} &\leq X_{i,j-1,t-p_{i,j-1}} \ \forall j > 1, t && (16) \\ X_{i,j,t} &\in \{0,1\} \ \forall j, t && (17) \end{aligned}$$

The subproblems satisfy integrality property, in fact they are dual of a specially structured maximum flow network problem [17]. We can therefore relax the binary constraints (17) as below:

$$X_{ijt} \leq 1, \ \forall j, t \tag{18}$$

$$X_{ijt} \geq 0, \; \forall j, t \tag{19}$$

Repeated application of first set of constraints (15) leads us to the following constraints instead of relaxed constraints (18), (19):

$$X_{ijT} \leq 1, \; \forall j \tag{20}$$

$$X_{ijt} \geq 0, \; \forall j, t \tag{21}$$

Since the dual to $(LR_{\lambda,i})$ is a max-flow problem on a network, we may use special dual network flow algorithms to solve each job-level subproblem.

3.2 Chance-Constrained Programming

We will now construct a chance-constrained model for each job-level subproblem by first accepting the fact that the processing times are uncertain. We assume that we have information about the type of processing time distribution, and its mean and variance. This information is typically easy to estimate in a real system using historic data.

Chance-constrained programming was first proposed by Charnes and Cooper [4]. In this approach, we use chance-constraints to represent processing time variation instead of using deterministic constraints that assume expected processing times. For example, if we want to represent a constraint, say $x < b$ (b is not perfectly known), that holds with a probability of at least β, the corresponding chance-constraint is $P[x < b] \geq \beta$ where P[.] is probability function for b. In job-level subproblems, precedence constraints are the only constraints that are affected by processing time uncertainty. For a specific job i, the precedence constraints are as follows:

$$X_{ijt} \leq X_{i,j-1,t-p_{i,j-1}}, \; \forall \; j > 1, t \tag{22}$$

Since processing times p_{ij} occur in indices in (22), it is difficult to convert these constraints as they are. However, we can obtain start and completion times (S_{ij} and C_{ij}) of each operation as follows:

$$S_{ij} = T - \sum_t X_{ijt} + 1 \text{ and } C_{ij} = T - \sum_t X_{ijt} + p_{ij} \tag{23}$$

Hence we can rewrite the precedence constraints (22) as follows:

$$C_{ij} \leq S_{i,j+1} - 1, \; \forall \; j < n_i, \tag{24}$$

or equivalently

$$\sum_t X_{ijt} - \sum_t X_{i,j+1,t} \geq p_{ij}, \; \forall \; j < n_i \tag{25}$$

We can now incorporate probabilistic processing times as follows: Suppose we want a *confidence level* of β_{ij} for each operation j of job i to satisfy its stated precedence constraint (25), then we have the following chance constraints:

$$P\left[\sum_t X_{ijt} - \sum_t X_{i,j+1,t} \geq p_{ij}\right] \geq \beta_{ij},\ \forall j < n_i \tag{26}$$

Further, instead of chance constraints (26), we substitute the following deterministic and equivalent constraints:

$$\sum_t X_{ijt} - \sum_t X_{i,j+1,t} \geq P_{ij},\ \forall j < n_i \tag{27}$$

where P_{ij} is the smallest number satisfying

$$P[p_{ij} \leq P_{ij}] \geq \beta_{ij} \text{ or } F[P_{ij}] \geq \beta_{ij} \tag{28}$$

where $F[.]$ is cumulative distribution function. Specifically, if the processing time p_{ij} is known to be Uniformly distributed between a_{ij} and b_{ij}, then the corresponding *adjusted processing time* $P_{ij} = a_{ij} + \beta_{ij}(b_{ij} - a_{ij})$. If p_{ij} is predicted to be Normally distributed with $N(\bar{p_{ij}}, \sigma_{ij}^2)$, then $P_{ij} = \bar{p_{ij}} + K_{\beta_{ij}}\sigma_{ij}$ where $K_{\beta_{ij}}$ is standard Normal value corresponding to β_{ij}.

We can now convert the chance-constraint equivalents into their original form with the *adjusted processing times* included in indices. This results in a job-level subproblem (referred as $LRC_{\lambda,i}$ for LR with *chance-constraints*) that minimizes the subproblem objective defined in (14) subject to chance-constrained precedence, non-preemption constraints and bound constraints for the decision variables. The job level subproblem with chance constraints for job i can be stated as follows:

$$v(LRC_{\lambda,i}) = \min A_i + \sum_j \sum_t (\lambda_{m_{ij},t} - \lambda_{m_{ij},t+P_{ij}} - w_{ijt})X_{ijt} \tag{29}$$

$$\text{s.t. } X_{i,j,t} \leq X_{i,j-1,t-P_{i,j-1}}\ \forall j > 1, t \tag{30}$$

and constraints (15), (20), (21).

The revised Lagrangean objective is:

$$v(LRC_\lambda) = \sum_i v(LRC_{\lambda,i}) - \sum_k \sum_t \lambda_{kt} \tag{31}$$

We can still solve each job-level subproblem using dual network algorithm because the structure of the current model is the same as the deterministic version. We use a *subgradient search algorithm* for the Lagrangean relaxation problem with chance constraints.

Our subgradient optimization starts with an initial value for the multipliers $\lambda^0 = 0$. Then the algorithm generates a sequence $\{\lambda^r\}$ over the iterations r by the rule

$$\lambda_{kt}^{r+1} = \max\{0, \lambda_{kt}^r + s_r g_{kt}(X^r)\} \tag{32}$$

where s_r is an appropriately selected step size and $g_{kt}(X^r)$ is the subgradient of the capacity constraint of machine k for time period t defined by the optimal solution X^r of LRC_{λ^r}:

$$g_{kt}(X^r) = \sum_i \sum_{j:m_{ij}=k} (X_{i,j,t}^r - X_{i,j,t-P_{ij}}^r) - 1 \tag{33}$$

Step size s_r is calculated as follows:

$$s_r = \alpha_r \left(\frac{UB - v(LRC_{\lambda^r})}{\sum_k \sum_t g_{kt}^2(X^r)} \right) \tag{34}$$

where α_r is a scalar satisfying $0 < \alpha_r \leq 2$ and UB is a target upper bound value for JSP which can be updated over the iterations. Details of the subgradient search algorithm is given in Section 3.4.

3.3 Scaling, Lagrangean Ranking and Schedule Preprocessing

Before going into further details of the subgradient search algorithm, note that in the JSP formulation, the number of discrete time periods T has a significantly effect to the computational tractability of the model. We scaled the processing times by a *scaling factor* which provides an upper bound to the original model at a reduced computational cost. In addition to reducing computer time, the scaling factor brings about several interesting issues. First, since the JSP model is used to preprocess (rather than solve) the scheduling instance at hand, we are more interested in analyzing the aggregate structure of the problem rather than the minute-to-minute details. The *scaling factor* allows us to adjust the "resolution" of the analysis. Second, since we consider uncertain processing times, the stochastic information about processing times could be used to determine a proper scaling factor. Third, for regular performance measures (such as weighted tardiness) a unique schedule can be determined by the operation ranking in each machine. For example, one does not need to know the actual processing times to implement an SPT dispatching rule. Knowing the relative processing time ranking would suffice. Thus, in theory, one could maximize the scale of the processing time so long as the operational ranking is still distinguishable. In general, the *scaling factor* provides a parametric trade-off between model resolution and computational efficiency. Throughout our implementation, we use scaled, adjusted processing times, P_{ij}^s for the model.

Another important aspect of our proposed scheme is that we use Lagrangean Relaxation and subgradient search to find a partially established ranking among the operations.

Specifically, in each subgradient iteration, we combine all solutions from job-level subproblems to get a capacity-infeasible shop schedule. Based on this *Lagrangean Schedule*, we define *Lagrangean Ranking* as follows:

Definition: (*Lagrangean Ranking*) Consider three specific conditions between operation pairs (i, j) and (i', j') in the *Lagrangean Schedule* as follows:

(1) (i, j) and (i', j') require the same machine, i.e., $m_{ij} = m_{i'j'}$,

(2) there is no resource conflict between (i, j) and (i', j') in the *Lagrangean Schedule*, i.e., $(X_{i,j,t} - X_{i,j,t-P_{ij}}) + (X_{i',j',t} - X_{i',j',t-P_{i'j'}}) \leq 1$, and

(3) operation (i, j) starts earlier than (i', j') in the *Lagrangean Schedule*, i.e., $S_{ij} \leq S_{i'j'}$

If an operation pair (i, j) and (i', j') satisfies (1)-(3)then we say that it is *fully ranked*, and operation (i, j) has a higher priority than (i', j'). For operation pairs which satisfy conditions (1) and (3) but not (2), we say that they are *weakly ranked.* All the other operation pairs are *unranked.*

It is clear from the definition that only operation pairs with no resource conflict in the *Lagrangean Schedule* are *fully ranked.* Using the *Lagrangean Ranking* we can now define a preprocessed scheduling instance by imposing a precedence constraint (4) between each *fully ranked* pair of operations but not the *weakly ranked* pairs. Thus, the *weakly ranked* pairs of operations represent scheduling decisions remain to be made. Note that under dynamic dispatching, the Lagrangean ranking only takes into effect when the ranked pair (i, j) and (i', j') are both in the candidate set. When they do not appear in the candidate set at the same time, the Lagrangean Ranking is relaxed. This is necessary in order to maintain the semi-active scheduling property [15]. Suppose we apply an active or non-delay schedule generation routine on this preprocessed scheduling instance using any priority dispatching rule, a feasible upper bound (UB) solution for JSP can be generated. In this context the priority specified by the *weak ranking* could be used for job dispatching. Alternatively, one could use dynamic dispatching rule such as Apparent Tardiness Cost (ATC) rule [18] which makes use of dynamic shop information over time.

The procedure described above can be used to generate upper bounds during each Lagrangean iteration. More importantly, we will use the procedure as a means to implement the Preprocess-First-Schedule-Later scheme described earlier. This PFSL scheme is outlined as follows:

1. At the beginning of the planning period, use *a priori* scheduling data and stochastic information to formulate a chance-constrained, Lagrangean-relaxed scheduling problem (as described in Sections 3.1 and 3.2).

2. Solve the stated scheduling problem using a subgradient search algorithm (Section 3.4). In each iteration, compute LB using the *Lagrangean Schedule*, compute UB using the associated *Lagrangean Ranking.* At the end of the subgradient search, the *Lagrangean Ranking* associated with the best UB is returned for preprocessing.

3. Preprocess the scheduling instance,i.e., adding precedence constraints for each *fully ranked* operation pairs.

4. Release the preprocessed partial schedule to the shop. Generate the detailed schedule dynamically using priority dispatching.

3.4 The Subgradient Search Algorithm

The subgradient search algorithm is summarized in Figure 1. When we implement the algorithm, we generate the initial upper bound (Step 3) using the Apparent Tardiness Cost (ATC) rule [18]. ATC is shown to be an effective heuristic for the weighted tardiness objective [9]. We implemented a *non-delay* and an *active* versions of the ATC heuristic. The Lagrange multipliers are initialized at 0. In computing the upper bound at each subgradient iteration (Step 5.2) we implemented a non-delay and an active version using priority dispatching defined by (1)*weak ranking*, or (2) the ATC rule. Results of all four combinations are reported in the computational testing.

Another important implementation choice is the use of scaled or unscaled processing time for upper bound calculation. Clearly the upper bound UB_r used for step size calculation in Steps 5.2-5.5 must be computed from scaled processing times since it should be consistent with the lower bound values. However, in Steps 5.3 and 6 when we try to determine the "best" upper bound, we could use an upper bound generated with the unscaled processing times. This is because the unscaled processing times may provide a more accurate assessment of the ultimate scheduling performance given a particular *Lagrangean Ranking.* Both implementations are reported in the computational study.

Step 1. Read input data: number of jobs N, number of machines M, job weights W_i, due dates d_i, mean (unscaled) processing times p_{ij}, machine requirements m_{ij}. Read distribution information (a_{ij} and b_{ij} in the Uniform case, σ_{ij}^2 in the Normal case).

Step 2. Calculate adjusted processing times P_{ij} for constraints (30) according to the distribution type and confidence level β. Scale the adjusted processing times to obtain scaled processing times P_{ij}^s. Scale the due dates accordingly.

Step 3. Upper bound initialization: generate an active or a non-delay schedule using a dispatching rule and scaled processing times. Use the upper bound schedule to set a reasonable planning horizon T.

Step 4. Initialize the Lagrange multipliers λ_{kt}, the iteration counter r, and the scalar parameter α_r.

Step 5. Main Step: Perform the subgradient search for K iterations.

- Step 5.1 Compute the Lagrangean lower bound (LB_r): solve the job-level subproblems (29), (30), (15), (20), (21) using scaled processing times P_{ij}^s. This results in a capacity-infeasible schedule with objective value $v(LRC_{\lambda^r})$ which provides a lower bound for iteration r.
- Step 5.2 Compute the Lagrangean upper bound (UB_r):Determine the *Lagrangean Ranking* from the lower bound schedule. Preprocess the scheduling instance using the *fully ranked* operation pairs, i.e., add a precedence constraint for each *fully ranked* pairs. Use a non-delay or an active schedule generation routine and a specified priority dispatching rule, generate a feasible schedule from the preprocessed scheduling instance. This provides an upper bound for iteration r.
- Step 5.3 If the upper bound value UB_r is lower than the best UB so far, then update the upper bound, $UB \leftarrow UB_r$. Record the current *Lagrangean Ranking.*
- Step 5.4 Update scalar α_r if necessary.
- Step 5.5 Compute the subgradients $g_{kt}(X^r)$, step size s_r and update the Lagrangean multipliers λ_{kt}^{r+1} based on (33), (34), and (32), respectively.

Step 6. Report the *Lagrangean Ranking* corresponding to the best UB over the K iterations (or alternatively, report the *Lagrangean Ranking* corresponding to the best lower bound *Lagrangean Schedule*). This is the *Lagrangean Ranking* to be released for schedule preprocessing.

Figure 1: The Subgradient Search Algorithm

Table 2: Experimental factors and their levels in the simulation experiments (Each factor combination is run 10 replications).

Factor	Levels	No of Levels
Problem Size	10x10, 20x15, 30x10	3
Confidence Level	(50%), (80%)	2
Distribution Type	Uniform, Normal	2
Uncertainty Level	Low (10%), Medium (30%), High (50%)	3

4 Computational Study

4.1 Experimental Design

We tested the proposed solution methodology in a Monte-Carlo environment simulating shop disturbances. The base test problems are generated from standard makespan job shop scheduling problems as described in [3]. We use 5 10x10 (10 jobs, 10 machines), 3 20x15, and 3 30x10 problems for testing. We coded each job-level scaled subproblem with chance-constraints in AMPL [6] and solved using dual network simplex algorithm in CPLEX [14]. A PentiumPro 200 Mhz personal computer with 64MB of RAM is used for all testing. We set initial parameter values as follows: $\lambda_{kt}^0 = 0 \; \forall k, t$, and $\alpha_0 = 2$ for the subgradient optimization procedure. We halve the value of α if there is no improvement in the Lagrangean lower bound for 3 iterations and set the total number of iterations K at 100 for 10x10 problems, and 200 for the others. For scaling, we divided adjusted processing times by a fixed value and rounded them to nearest integer values to obtain *scaled processing times* between 1 and 10. We scaled the due dates similarly. We used a common confidence value β for all chance-constraints. We tested two confidence levels as β= 50%, and β=80%. Note that 50% confidence corresponds to using only mean (original) processing times whereas 80% confidence uses the variance information.

To simulate the stochastic environment, we generate actual processing times p'_{ij} using two distribution types:(1)*Uniform*:$U[a_{ij}, b_{ij}]$ where $p_{ij} = \frac{a_{ij}+b_{ij}}{2}$, and (2)*Normal*: $N[p_{ij}, \sigma_{ij}^2]$. Note that we assume the processing times p_{ij} specified in the scheduling data is the mean of the underlying processing time distribution. We tested three levels of processing time variation (V): 10%, 30% and 50%. In the case of Uniform distribution, the range parameters are calculated as $a_{ij} = p_{ij} - V * \bar{p}$ and $b_{ij} = p_{ij} + V * \bar{p}$ where $\bar{p}$ is the overall mean of the processing times. Thus, actual processing times p'_{ij} are generated using discrete uniform distribution $U[a_{ij}, b_{ij}]$. For the Normal distribution case, we equalize the variance with the Uniform case as follows, i.e. we set

$$\sigma_{ij}^2 = \frac{(b_{ij} - a_{ij})^2}{12}$$

where b_{ij} and a_{ij} are from the Uniform case with corresponding variation level V. Note that since a_{ij} and b_{ij} are calculated using single overall mean, the Normal variance values are equal for all the operations, i.e. $\sigma_{ij}^2 = \sigma^2 = V^2\bar{p}^2/3$. We truncated negative numbers and 0 generated in the implementation of these formulas to 1. For a summary of experimental factors, see Table 2.

We test the performance of the proposed scheduling scheme under stochastic disturbances using the following experimental steps:

Step 1. Given a scheduling problem and the distribution information about the processing times, use the proposed PFSL scheme to find a preprocessed partial schedule using the *Lagrangean Ranking.*

Step 2. Complete the remainder of scheduling decisions over time using a specified dispatching method (i.e., Weak Ranking or ATC-ranking). Simulate this process using a Monte Carlo experiment where the operation processing times vary according to the specified distribution. At each dispatching point, the dispatching method uses actual processing time that are known up to point while using the original (mean) processing time for future operations. Report the total weighted tardiness at the end of the simulation.

4.2 The Dynamic Dispatching Benchmark

To establish a benchmark for scheduling robustness we use the ATC dynamic dispatching heuristic. As demonstrated in previous research [19] [3] dynamic dispatching heuristics such as ATC serve as an excellent benchmark since they provide consistent performance under a wide variety of shop conditions. To provide a broader base for the benchmark we implemented four different versions of the ATC heuristic using *active* and *non-delay* schedule generation routines assuming *perfect* and *mean-value* information on the processing times. In the *perfect information* case, we first generate the actual (after perturbation) processing times p'_{ij} and make this information available for each ATC priority calculation. In the *mean-value* case the ATC heuristic uses the mean processing time provided with the scheduling problem for future jobs and actual processing times for jobs that are already dispatched. In both cases, the heuristic is evaluated using Monte-Carlo simulation with Uniform and Normal processing-time perturbation described above. In Table 3 we first summarize the results of the four different ATC implementations. As shown in the table, ATC appears to be quite robust under different processing time variations.

Table 3: Experimental results with ATC dispatching (Each cell shows the average total weighted tardiness over 50 instances: 5 10x10 problems each with 10 replications).

	Mean-Value Information		Perfect Information	
	Active	Nondelay	Active	Nondelay
Variation Level	Variation Type: Uniform Distribution			
10%	8941.90	6417.16	8508.92	6497.04
30%	8847.56	7277.32	9058.54	7223.88
50%	9417.68	8052.56	10452.96	7969.96
Variation Level	Variation Type: Normal Distribution			
10%	8905.22	6470.26	8865.3	6595.34
30%	8543.24	6951.28	8663.44	6767.08
50%	8373.46	7239.32	9454.66	6960.24

The heuristic yields better performance in the normal variation case than in the uniform case at the same variation level. While non-delay ATC schedules significantly outperform their active counterparts, the difference between perfect vs. mean-value information is insignificant. This non-intuitive result can be explained by the fact that the ATC index makes look ahead approximation at each dispatching point using aggregated future information on each job. As a result, using mean or actual processing times may not yield significant performance difference. Based on this result, we will compare our approach with non-delay ATC procedure using mean-value processing times.

4.3 Testing Implementation Choices

To test the proposed PFSL scheme using Lagrangean preprocessing we first compare the effects of three different implementation choices using 10x10 test problems:

- *Upper Bound Calculation*: use unscaled vs. scaled processing time for upper bound selection and schedule evaluation (see Section 3.4),
- *Dynamic Dispatching*: After the Lagrangean pre-processing, complete the schedule dynamically using *weak-ranking* vs. ATC-ranking, and
- *Schedule Generation*: Use active vs. non-delay procedure for schedule generation

The above implementation choices result in eight combinations, for each combination we tested 50 10x10 instances (i.e., 5 base problems each repeated for 10 replications for

Table 4: Experimental results of the PFSL methodology (Each cell shows the average total weighted tardiness over 50 instances: 5 10x10 problems each with 10 replications).

UB Calculation	Unscaled Processing times				Scaled Processing Times			
Dispatching	Weak Ranking		ATC-Ranking		Weak Ranking		ATC-Ranking	
Sch. Routine	Active	Nondelay	Active	Nondelay	Active	Nondelay	Active	Nondelay
β, V	*Variation Type: Uniform Distribution*							
80%, 10%	4566.70	6327.86	4980.12	6015.16	4973.52	6186.70	5323.54	5722.92
80%, 30%	5165.32	7170.28	5911.54	6917.92	5413.18	7037.78	6080.42	7191.02
80%, 50%	6627.56	8163.36	6959.26	8561.28	7714.94	8585.10	8011.72	8286.26
50%, 10%	4489.72	6181.14	5525.48	6839.14	5079.28	7500.76	5519.96	6956.40
50%, 30%	5879.22	7694.74	6964.66	7650.18	5853.82	7908.18	6481.36	7772.06
50%, 50%	8042.38	9094.54	9020.22	9019.42	7995.12	9347.64	8165.32	9060.42
β, V	*Variation Type: Normal Distribution*							
80%, 10%	4318.56	6557.92	5380.60	6110.76	4660.90	6839.98	5940.38	5869.78
80%, 30%	5431.26	6945.08	6121.26	6720.94	5942.60	6861.90	5829.60	6587.56
80%, 50%	6013.62	7721.78	6737.46	7887.30	6590.66	8080.44	6283.48	7258.70
50%, 10%	4630.64	6198.38	5299.34	6358.92	4969.24	6969.36	5552.00	7004.94
50%, 30%	5516.22	6865.04	6551.74	6738.52	6058.16	7517.72	6412.38	7129.40
50%, 50%	6792.16	7759.62	7913.74	7237.02	7448.34	8154.90	7412.70	7828.72

processing time variation). The results are summarized in Table 4. We will first make a few observations on the implementation differences before providing more detailed analysis in subsequent sections.

Results:

- Contrary to the ATC benchmark in Table 3 active schedule generation works significantly better under the PFSL scheme.
- Among the active scheduling results, using weak Lagrangean ranking for dynamic dispatching appears to be better than using the ATC-ranking in almost all cases. Interestingly, the situation reverses among the non-delay scheduling results: ATC-ranking yields better tardiness in most cases.
- Comparing the results of different upper bound evaluation during the subgradient search (using scaled vs. unscaled processing times), we see that the results are quite similar and there is no clear dominance between either implementation.

4.4 Computational Results

We conducted a second set of experiments for larger size (20x15 and 30x10) test problems using unscaled processing times, weak Lagrangean ranking and active schedule generation. In the following we examine the conjectures stated in Section 2.2 using the 10x10, 20x15 and 30x10 computational results.

4.4.1 Tests for Conjecture 2: The Value of Second Moment Information

The second conjecture stated earlier claims that preprocessing strategies that incorporate the second moment information of processing time variation are more robust than ones that use only expected values. We test this conjecture by comparing the 80% and 50% confidence levels (β) used to characterize the chance constraints. Recall that in the 80%-confidence case both the mean and variance of processing time variations are used in Lagrangean preprocessing, while in the 50% cases the variance information is ignored. As the results in Table 4 show, in 85.4% (or 41 out of 48) of the cases the 80%-confidence cases yield significantly lower (an average of 9.7%) weighted tardiness when compared to the 50% cases.

The conjecture also state that the value of the second moment information increases as the level of uncertainty increases. As can be computed from Table 4, in the high uncertainty (V= 50%) cases, the algorithm benefits from the use of the variance information much more frequently then the low uncertainty (V=10%) cases (94% vs. 69%). However, the amount of improvement is roughly the same (10.8% vs. 10.2%).

Furthermore, the value of the second moment information is affected by the type of distribution. It yields a performance improvement of up to 29.6% (11.3% average) in the Uniform cases, and up to 19.3% (7.8% average) in the Normal cases.

The effect of using the second moment information is further depicted in Figures 2 and 3 where we plot the results from the implementation using weak ranking, unscaled processing times and active scheduling. As the graphs show, the 80% confidence level draws the lower envelope in both variation types and in all variation levels. Corresponding plots for large-size problems are shown in Figures 4 and 7. As the figures show, the results from 20x15 and 30x10 problems support the conjecture that the second moment information significantly improves scheduling robustness.

The figures also show that the performance improvement achieved by the second moment information (comparing the 80%- and the 50%-confidence curves) is more pronounced in Uniform distribution cases.

Result: The above observations suggest that the second moment information add significant value to the *a priori* analysis used for preprocessing. Moreover, this information has a higher marginal value when the level of uncertainty is higher (as is the case when V=30%, 50%), and when the processing time variation is Uniformly (rather than Normally) distributed.

4.4.2 Tests for Conjecture 3: The Value of *A Priori* Analysis

In the third conjecture, we state that the performance of dynamic dispatching such as ATC can be significantly improved by *a priori* analysis using stochastic information. To test this conjecture we compare the results from straight-forward ATC dispatching (Table 3) and ATC dispatching after Lagrangean preprocessing (i.e., ATC-ranking for dynamic dispatching in Table 4). As shown from the tables, if the Lagrangean preprocessing is implemented using the second moment information (β=80%) and active schedule generation, it significantly outperforms straight-forward non-delay ATC dispatching under all levels of uncertainty in all test cases (an average of 34% improvement, ranging from 7% to 80%). Figures 2-7 show graphically the performance of this preprocessing scheme (with and without the variance information) as compared to the non-delay ATC heuristic. As the graphs show, the performance of Lagrangean Preprocessing clearly dominate that of ATC.

Conjecture 3 also states that the extent of the improvement varies depending on the level of uncertainty, problem instances and the underlying distribution. Interestingly, the problem size also plays a role here. For 10x10 problems, we observe that the performance difference over ATC is consistent across different level of uncertainty. While for 20x15 and 30x10 problems the performance difference is more significant at high uncertainty levels. On the other hand, the distribution type does not seem to have a significant effect to the extent of improvement.

Result: The above observations suggest that *a priori* analysis using stochastic information can significantly improve the performance of dynamic dispatching. The extent of the improvement varies by the problem size and the level of uncertainty.

4.4.3 Tests for Conjecture 4: The Marginal Benefit of Stochastic Analysis

Conjecture 4 states that the marginal benefit of *a priori* stochastic information increases as the level of uncertainty increases, and its effect varies depending on the characteristics of its underlying statistical distributions. We will examine the first half of this conjecture

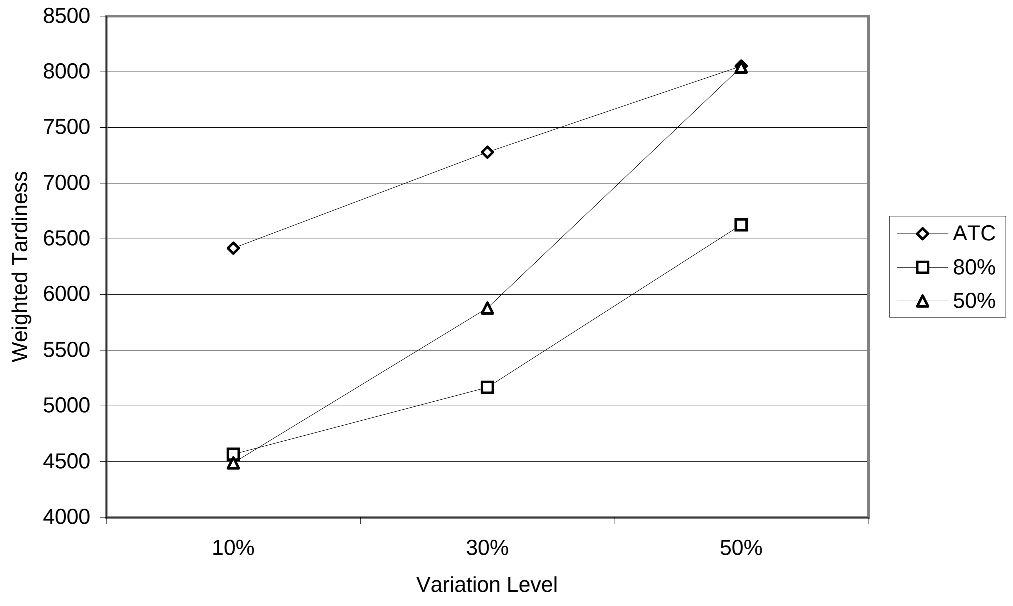

Figure 2: Average Weighted Tardiness versus Uncertainty Level for selected PFSL settings in *Uniform* processing time variation case (10x10 problems)

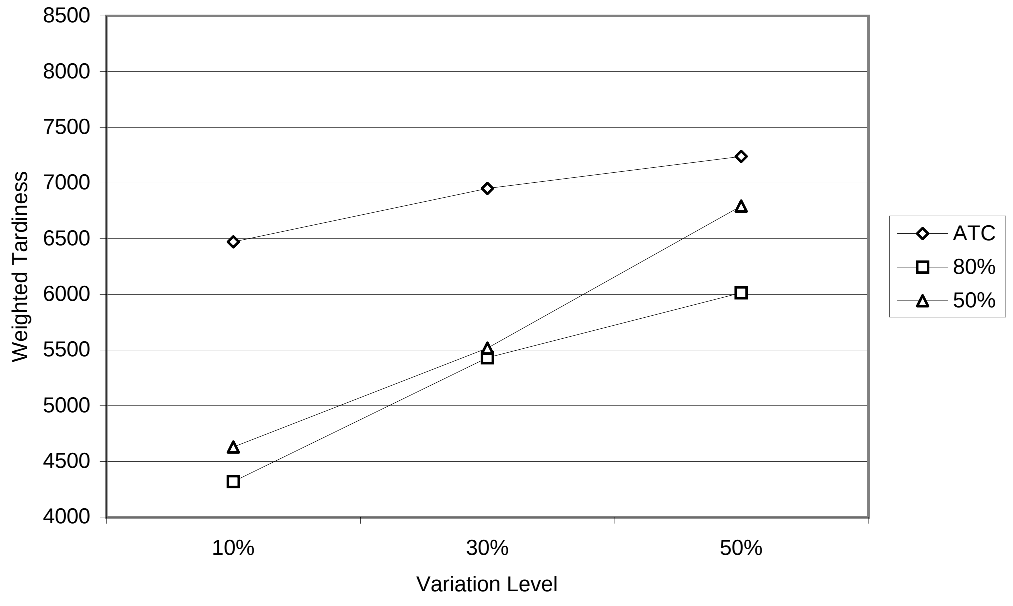

Figure 3: Average Weighted Tardiness versus Uncertainty Level for selected PFSL settings in *Normal* processing time variation case (10x10 problems)

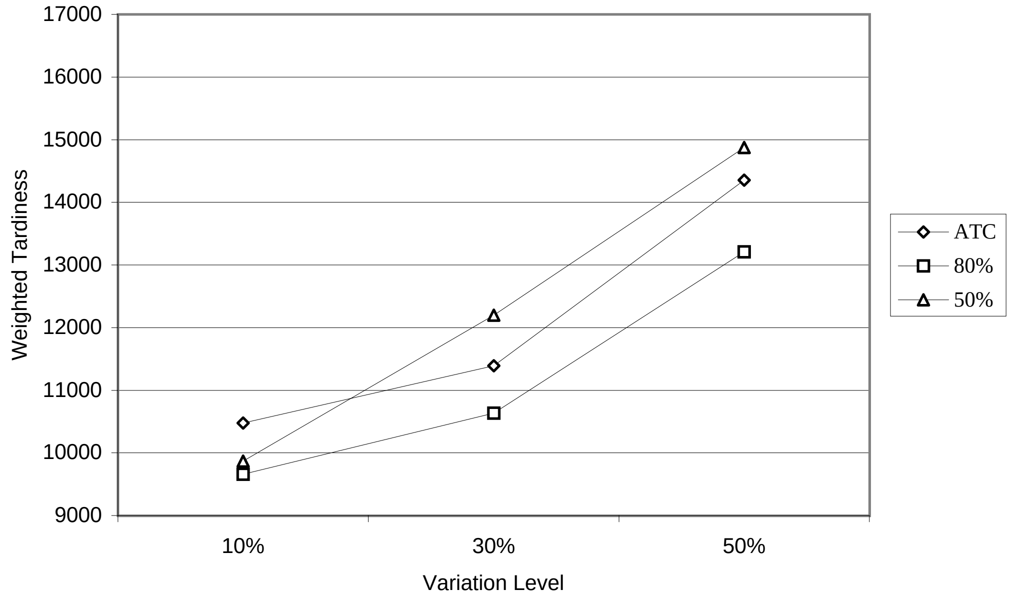

Figure 4: Average Weighted Tardiness versus Uncertainty Level for selected PFSL settings in *Uniform* processing time variation case (20x15 problems)

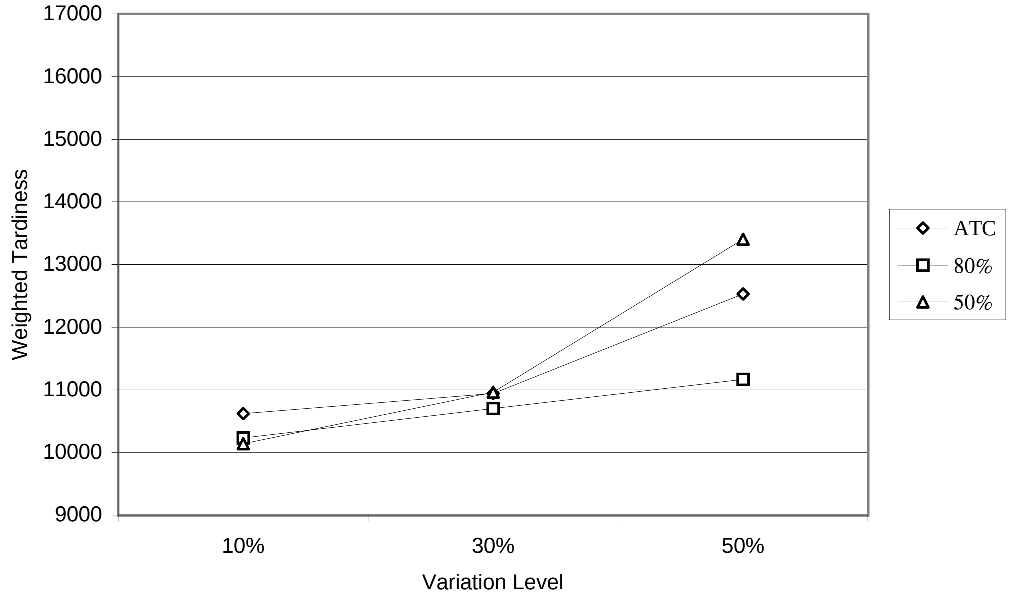

Figure 5: Average Weighted Tardiness versus Uncertainty Level for selected PFSL settings in *Normal* processing time variation case (20x15 problems)

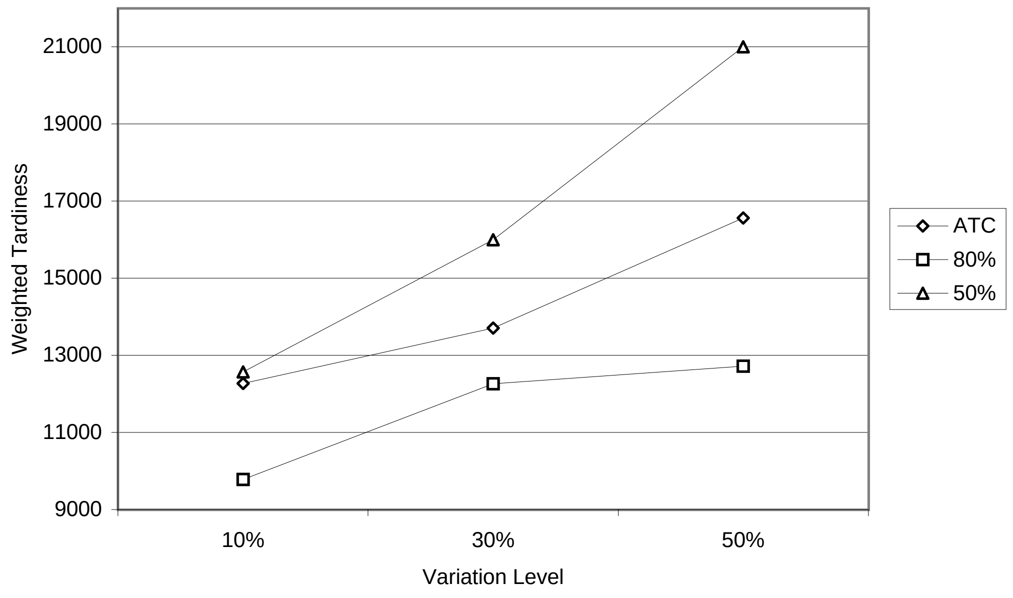

Figure 6: Average Weighted Tardiness versus Uncertainty Level for selected PFSL settings in *Uniform* processing time variation case (30x10 problems)

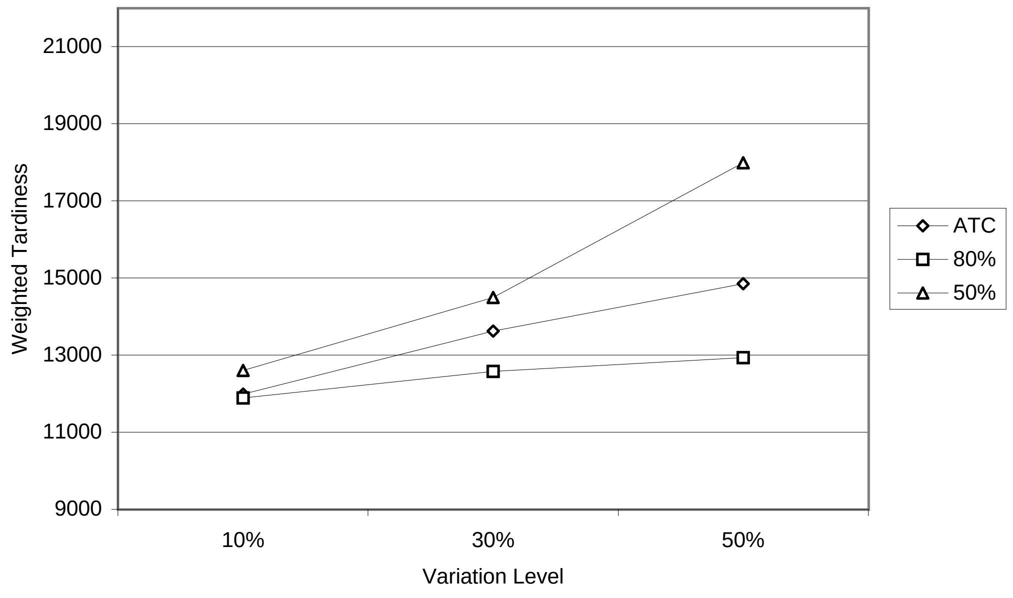

Figure 7: Average Weighted Tardiness versus Uncertainty Level for selected PFSL settings in *Normal* processing time variation case (30x10 problems)

using Figures 2-7. As the figures show, while the performance of ATC and 50%-confidence preprocessing without variance information (both methods ignore the stochastic information) deteriorate rapidly as the level of uncertainty increases, 80%-confidence preprocessing appears to be very robust to the changes in processing time uncertainty. This supports the conjecture that the stochastic analysis has an increasing importance as the level of uncertainty increases.

Now consider the second half of the conjecture. As we compare the slopes of the curves, we see that all methods are more sensitive to the uncertainty levels under Uniform distribution. As pointed out earlier, the plots also show that the performance improvement achieved by the second moment information (comparing the 80%- 50%- confidence cases) is more pronounced in the Uniform distribution cases. Recall that we have equalized the variance between the Uniform and the Normal distributions. If we compare the Uniform vs. Normal distribution results in Table 4, (at three corresponding levels of uncertainty), we find that 71% of the time (or 34 out of 48 cases) the preprocessing algorithm achieve a better performance under Normally distributed variations.

Result: The above observations suggest that *a priori* stochastic analysis indeed improve scheduling robustness. Compared to methods ignoring stochastic information, its benefit is more evident when (1) the level of uncertainty is high, and (2)when the variation is Uniformly distributed. However, comparing different statistical assumptions (Uniform vs. Normal distribution under equalized variance), Normal distribution appears to provide more valuable *a priori* information.

4.4.4 Remarks on Due Date Tightness

As the results presented in the previous sections show, problem instances do affect the performance of the PFSL scheme. When we first tested the 20x15 problem instances an interesting observation was made: during preprocessing the Lagrangean relaxation lower bound did not improve over 200 subgradient iterations, and the performance of PFSL was no better than that of a straight-forward ATC. Further investigation revealed that the LP-relaxation lower bound (which presents a theoretic upper bound for the Lagrangean relaxation lower bound in our model) is equal to 0 for almost all problems in this test set. That is, if we relax the binary constrains on the decision variables, we can process all the jobs on or before their due dates and obtain 0 weighted tardiness. Hence, the subgradient search algorithm does not have the needed information to improve the LR lower bound (the LB is always 0).

To address this issue we solve a surrogate Lagrangean relaxation problem during the preprocessing phase. The surrogate problem is created by subtracting a fixed value Δ from the original due dates for the LR problem and solved the revised problem using the

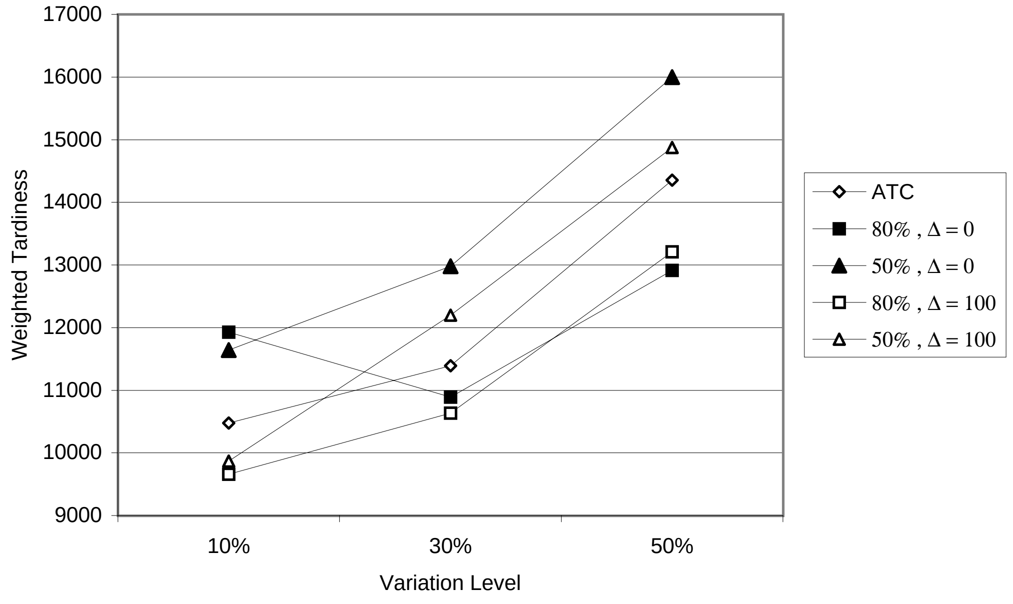

Figure 8: Effects of surrogate scheme ($\Delta = 100$) on PFSL algorithm. (Uniform processing time variation case, 20x15 problems)

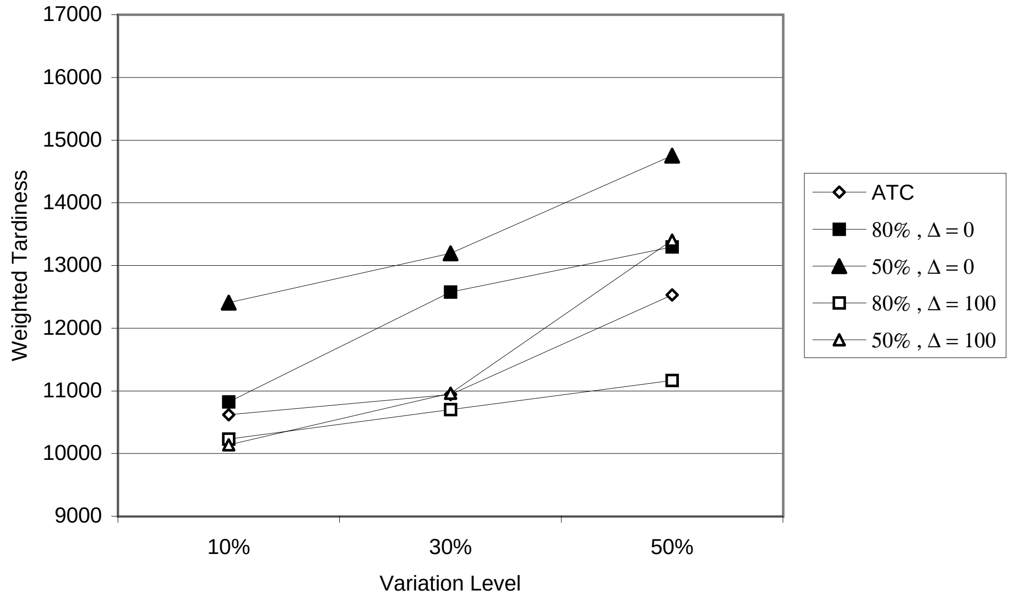

Figure 9: Effects of surrogate scheme ($\Delta = 100$) on PFSL algorithm. (Normal processing time variation case, 20x15 problems)

subgradient search algorithm. At the end of the search we use the established Lagrangean ranking (both *full* and *weak ranking*) as before. When simulating the performance of this PFSL scheme, we use the original due dates. With this simple scheme, we have successfully overcome the “degeneracy” problem during the subgradient search. Figures 8 and 9 show the results of this set of experiments for 20x15 problems. As the graph depicts, the surrogate scheme significantly improve the performance of the preprocessing algorithm. All 20x15 results shown in earlier sections correspond to this improved version.

5 Conclusions

We have developed a preprocess-first-schedule-later scheme for job shop scheduling with the purpose of improving scheduling robustness. We first perform a Lagrangean relaxation of the machine capacity constrains which results in network-structured job subproblems. Based on stochastic information about the jobs, we add stochastic processing time constrains to the job subproblems. The resulting chance-constrained stochastic program makes use of the first and second moment information on the processing time. This extra information comes at little computational expense since the subproblems still have the integrality property. Using a subgradient search algorithm, we preprocess each scheduling instance by solving the stochastic model. After preprocessing, a Lagrangean Ranking is established for the schedule. The actual schedule is generated dynamically using the Lagrangean ranking. Intensive computational testing supports many of the conjectures listed in Section 2.2. The main results are summarized at the end of each subsection in Section 4.

We demonstrated that both stochastic analysis and dynamic adaptation could significantly improve scheduling robustness. Making use of second moment stochastic information achieves significant improvement in a majority of the cases. More importantly, the benefit of stochastic analysis are more significant in larger size problems at a higher level of uncertainty. Not surprisingly, the marginal benefit of stochastic preprocessing is also affected by the assumed statistical distributions and problem instances.

Acknowledgement

The research is supported, in part, by National Science Foundation Grant DMI-9634808 and SBR-9422862.

References

[1] P. Baptiste and J. Favrel. Taking into account the rescheduling problem during the scheduling phase. *Production Planning and Control*, 4:349–360, 1993.

[2] J. C. Bean, J. R. Birge, J. Mittenthal, and C. E. Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.

[3] E. S. Byeon, S. D. Wu, and R. H. Storer. Decomposition heuristics for robust job shop scheduling. Technical Report 93T-008, Lehigh University, Bethlehem, PA, 1993, to appear in *IEEE Transactions on Robotics and Automation.*

[4] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6:73–79, 1959.

[5] L. K. Church and R. Uzsoy. Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, 5:153–163, 1992.

[6] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Boyd and Fraser Publishing Company, Danvers, MA, 1993.

[7] G. Gallego. Linear control policies for scheduling a single facility after an initial disruption. Technical Report 770, Cornell University, Ithaca, NY, 1988.

[8] G. Gallego. Produce-up-to policicies for scheduling a single facility after an initial disruption. Technical Report 771, Cornell University, Ithaca, NY, 1988.

[9] E. Kutanoglu and I. Sabuncuoglu. Experimental investigation of scheduling rules in a dynamic job shop with weighted tardiness costs. In *Proceedings of 3rd Industrial Engineering Research Conference*, Atlanta, GA, 1994.

[10] S. V. Mehta and R. Uzsoy. Predictable scheduling of a job-shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, Forthcoming:149–156, 1997.

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

[12] T. E. Morton, S. Lawrence, S. Rajagopalan, and S. Kekre. A price-based shop scheduling module. *Working Paper, Graduate School of Industrial Administration, Carnegie Mellon University*, 1988.

[13] A. P. Muhlemann, A. G. Lockett, and C. K. Farn. Job shop scheduling heuristic and frequency of scheduling. *International Journal of Production Research*, 20(2):227–241, 1982.

[14] CPLEX Optimization. Using the CPLEX base system including CPLEX mixed integer solver and barrier solver options, 1995.

[15] M. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[16] A. Pritsker, L. Watters, and P. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science: Theory*, 16(1):93–108, 1969.

[17] R. O. Roundy, W. L. Maxwell, Y. T. Herer, S. R. Tayur, and A. W. Getzler. A price-directed approach to real-time scheduling of production operations. *IIE Transactions*, 23(2):149–160, 1991.

[18] A. P. J. Vepsalainen and T. E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047, 1987.

[19] S. D. Wu, E. S. Byeon, and R. H. Storer. A graph-theoretic decomposition of job shop scheduling problems to achieve scheduling robustness. Technical Report 93T-009, Lehigh University, Bethlehem, PA, 1993, to appear in *Operations Research.*

[20] S. D. Wu, R. H. Storer, and P. C. Chang. A rescheduling procedure for manufacturing systems under random disruptions. In *New Directions for OR in Manufacturing.* Springer-Verlag, 1992.

[21] S. D. Wu, R. H. Storer, and P. C. Chang. One machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research*, 20:1–14, 1993.

[22] M. Yamamoto and S. Y. Nof. Scheduling/rescheduling in the manufacturing operating system environment. *International Journal of Production Research*, 23(4):705–722, 1985.