Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2012, Article ID 623230, 17 pages
doi:10.1155/2012/623230

Research Article

A PMBGA to Optimize the Selection of Rules for Job Shop Scheduling Based on the Giffler-Thompson Algorithm

Rui Zhang[1] and Cheng Wu[2]

[1] *School of Economics and Management, Nanchang University, Nanchang 330031, China*
[2] *Department of Automation, Tsinghua University, Beijing 100084, China*

Correspondence should be addressed to Rui Zhang, r.zhang@ymail.com

Received 4 September 2011; Revised 18 December 2011; Accepted 19 December 2011

Academic Editor: Oluwole D. Makinde

Most existing research on the job shop scheduling problem has been focused on the minimization of makespan (i.e., the completion time of the last job). However, in the fiercely competitive market nowadays, delivery punctuality is more important for maintaining a high service reputation. So in this paper, we aim at solving job shop scheduling problems with the total weighted tardiness objective. Several dispatching rules are adopted in the Giffler-Thompson algorithm for constructing active schedules. It is noticeable that the rule selections for scheduling consecutive operations are not mutually independent but actually interrelated. Under such circumstances, a probabilistic model-building genetic algorithm (PMBGA) is proposed to optimize the sequence of selected rules. First, we use Bayesian networks to model the distribution characteristics of high-quality solutions in the population. Then, the new generation of individuals is produced by sampling the established Bayesian network. Finally, some elitist individuals are further improved by a special local search module based on parameter perturbation. The superiority of the proposed approach is verified by extensive computational experiments and comparisons.

1. Introduction

The job shop scheduling problem (JSSP) has been known as an extremely difficult combinatorial optimization problem ever since the 1950s. In terms of computational complexity, JSSP is strongly $\mathcal{NP}$-hard [1]. Because of its relevance to contemporary manufacturing systems, extensive research has been conducted on the problem [2]. In recent years, the metaheuristics—such as simulated annealing (SA) [3], genetic algorithm (GA) [4–6], tabu search (TS) [7, 8], particle swarm optimization (PSO) [9, 10], and ant colony optimization (ACO) [11, 12]—have clearly become the most popular methods.

However, most existing research is still based on the standard JSSP model, soit is inconvenient to apply these algorithms directly to real-world scheduling scenarios. Especially, most algorithms are designed for the makespan criterion (i.e., minimize $C_{\max}$). Actually, in the make-to-order production environment nowadays, due date related performances are becoming increasingly significant, because the in-time delivery of goods is vital for maintaining a high service reputation. Therefore, the algorithms that aim at minimizing tardiness in JSSP deserve more investigations.

In an attempt to contribute to the scheduling community, in this paper we study the JSSP with the objective of minimizing total weighted tardiness (TWT). TWT is the weighted sum of each job's tardiness against its due date. In some respects, the TWT measure captures the critical factors related with the profitability of a firm more accurately than the makespan. Meanwhile, from the theoretical perspective, the complexity of solving JSSP with TWT objective (abbreviated as TWT-JSSP hereinafter) is much greater than that of solving JSSP with makespan objective [13]. The value of TWT is affected by all the tardy jobs and thus is more sensitive to the changes in the schedule. According to the three-field notation, the studied problem can be described as $Jm|\,|\sum w_j T_j$.

Relatively few publications have discussed the minimization of TWT in JSSP. The only exact method is the branch-and-bound algorithm proposed by Singer and Pinedo [14], while the rest belong to the heuristic category. The first attempt of adapting the shifting bottleneck algorithm (which was very successful for makespan minimization) to TWT-JSSP is reported in [15]. In [16, 17], the authors propose modified shifting bottleneck heuristics for complex job shops (characterized by parallel batching machines, sequence-dependent setup times, and reentrant process flows) with TWT criterion. In these algorithms, the single/parallel machine subproblems are solved basically by the ATC rule or the extended BATCS rule (for the case of batching and setups). The shifting bottleneck heuristic proposed in [18], however, uses genetic algorithms to solve the subproblems. The computations show that using GA as subproblem solution procedures leads to improved results compared to dispatching-based subproblem solution procedures. Besides shifting bottleneck, a large step random walk (LSRW) algorithm is designed in [19], which uses different neighborhood sizes depending on whether a small step or a large step is performed. A small step consists of iterative improvement, while a large step consists of a Metropolis algorithm. In [20, 21], hybrid genetic algorithms are presented for solving TWT-JSSP. The former combines GA with different dispatching rules, while the latter combines GA with an iterated local search procedure that uses the longest path approach on a disjunctive graph model. In [22], a tabu search algorithm is presented for the generalized TWT-JSSP with release times and precedence constraints. Recently, electromagnetic algorithms (EM) have been successfully applied to TWT-JSSP [23]. In fact, most of these algorithms have relatively high computational complexity, and thus they are incapable of solving large-scale TWT-JSSP.

Rule-based scheduling approaches have been investigated for job shop scheduling since the late 1990s [24, 25]. A remarkable advantage of using dispatching rules is that it helps to save computational time and generally produces satisfactory solutions. Recently, modern optimization strategies are applied to find high-quality combinations of dispatching rules. For example, a neural network is designed in [26] to determine the dispatching rule to use on each machine in a job shop (it is necessary to train the neural network beforehand with optimal rule combinations for some known instances). A memetic algorithm with tailored-encoding/decoding schemes, and a local search procedure is proposed in [27] for minimizing the number of tardy jobs in JSSP. The memetic algorithm defeats a multistart hill-climbing approach and a simulated annealing approach. A dispatching rule-based genetic

algorithms (DRGA) is proposed in [28], which searches simultaneously for the best sequence of dispatching rules and the number of operations to be handled by each dispatching rule. The DRGA obtains better results than a GA using the conventional dispatching rule representation and a GA that uses the operation permutation representation. A genetic programming-based data mining approach is proposed in [29] to select dispatching rules under a given set of shop parameters (e.g., interarrival times). The results obtained from simulation show that the selected dispatching rules are appropriate according to the current shop status.

The rest of the paper is organized as follows. The discussed problem is mathematically formulated in Section 2. Section 3 makes a brief introduction to the principles of PMBGA. Section 4 proposes a rule-based PMBGA for solving TWT-JSSP. Section 5 presents the computational results and analysis. Finally, Section 6 concludes the paper.

2. Problem Formulation

In a JSSP instance, a set of n jobs $\{J_j\}_{j=1}^{n}$ are to be processed on a set of m machines $\{M_k\}_{k=1}^{m}$ under the following basic assumptions: (i) there is no machine breakdown; (ii) no preemption of operations is allowed; (iii) all jobs are released at time 0; (iv) the transportation times and the setup times are all neglected; (v) each machine can process only one job at a time; (vi) each job may be processed by only one machine at a time.

Each job has a fixed processing route which traverses the relevant machines (in the standard JSSP benchmark instances, each job is required to visit all the machines. But actually, the number of operations for each job (m_j) can be less than m) in a predetermined order. The manufacturing process of job j on machine k is noted as operation O_{jk}, with a duration of p_{jk}. Besides, a preset due date d_j (describing the level of urgency) and a preset weight w_j (reflecting the importance of the order) are given for each job j. The objective function is defined as TWT $= \sum_{j=1}^{n} w_j T_j$, where $T_j = \max\{C_j - d_j, 0\}$ defines the tardiness of job j.

JSSP can be described by a disjunctive graph $G(O, A, E)$ [30]. $O = \{O_{jk} \mid j = 1, \ldots, n,\ k = 1, \ldots, m\}$ is the set of nodes. A is the set of conjunctive arcs which connect successive operations of the same job, so A describes the technological constraints in the JSSP instance. $E = \bigcup_{k=1}^{m} E_k$ is the set of disjunctive arcs, where E_k denotes the disjunctive arcs corresponding to the operations on machine k. Each arc in E_k connects a pair of operations to be processed by machine k and ensures that the two operations should not be processed simultaneously. Initially, the disjunctive arcs do not have fixed directions.

Under the disjunctive graph representation, finding a feasible schedule for the JSSP is equivalent to orienting all the disjunctive arcs so that no directed cycles exist in the resulting graph. In this paper, we use σ to denote the set of directed disjunctive arcs which are transformed from the original E. Thus, if $A \cup \sigma$ is acyclic, the schedule corresponding to σ is feasible (in the rest of the paper, we do not distinguish between σ and the schedule. For the convenience of expression, we will write σ as a matrix. The kth row of σ represents the processing order of the operations on machine k).

Based on the disjunctive graph model, the discussed TWT-JSSP can be mathematically formulated as follows:

$$\min \quad \text{TWT} = \sum_{j=1}^{n} w_j \left(t_{jk_j} + p_{jk_j} - d_j\right)^{+}, \tag{2.1}$$

s.t.

$$t_{jk} + p_{jk} \leqslant t_{jk'}, \quad \forall (O_{jk}, O_{jk'}) \in A, \tag{a}$$

$$t_{jk} + p_{jk} \leqslant t_{j'k} \vee t_{j'k} + p_{j'k} \leqslant t_{jk}, \quad \forall (O_{jk}, O_{j'k}) \in E_k, \tag{b}$$

$$t_{jk} \geqslant 0, \quad j = 1, \ldots, n, \ k = 1, \ldots, m. \tag{c}$$

In this formulation, $(x)^+ = \max\{x, 0\}$. t_{jk} represents the starting time of operation O_{jk}. k_j denotes the index of the machine that processes the last operation of job j, so the completion time of job j is $t_{jk_j} + p_{jk_j}$. The set of constraints (a) ensure that the processing order of the operations from each job is consistent with the technological routes. The set of constraints (b) ensure that any two operations on the same machine cannot be processed simultaneously.

3. A Brief Introduction to PMBGA

Recently, there has been a growing interest in the evolutionary algorithms that explore the search space by building and utilizing probabilistic models of high-quality solutions. Indeed, these algorithms use the following two steps to replace the conventional crossover and mutation operators in GA:

(1) Build a probabilistic model of the selected promising solutions;

(2) Sample the built model to produce a new generation of candidate solutions.

The evolutionary algorithms based on such a principle are referred to as estimation of distribution algorithms (EDAs) or probabilistic model-building genetic algorithms (PMBGAs). The major steps of a PMBGA implementation are listed as follows, where GN is the maximum number of generations.

Step 1. Set the generation index $g = 0$. Initialize the population of the first generation, that is, $P^{(0)}$.

Step 2. Select a subset S of promising individuals from $P^{(g)}$.

Step 3. Establish a probabilistic model M which somehow describes the distribution characteristics of S.

Step 4. Generate a set N of new individuals by sampling M.

Step 5. Select the best $|P^{(g)}|$ individuals from $P^{(g)} \cup N$ and assign them to the next generation population $P^{(g+1)}$.

Step 6. Let $g \leftarrow g + 1$. If $g <$ GN, return to Step 2. Otherwise, output the best solution in $P^{(g)}$.

The PMBGA is especially useful for the complex optimization problems in which the decision variables are correlated. For such problems, the realized value of a certain decision variable can produce an impact on the optimal value for another decision variable. Therefore, if these variables are optimized in a separate way (or one by one), traditional

GA will be very likely to converge to local optimum. PMBGA improves traditional GA by modeling the relationship between decision variables. Clearly, the most crucial element in the design of PMBGA is the type of the adopted probabilistic model (in Step 3), which directly affects the algorithm's capability of producing high-quality offspring solutions. In the artificial intelligence community, the commonly adopted graphical model for characterizing the relationship between a set of discrete variables is the Bayesian network [31]. A Bayesian network is a directed acyclic graph. A node in the Bayesian network indicates a variable under investigation (each variable actually corresponds to a coding gene for the solutions in PMBGA), and an arc indicates the probabilistic causal relationship between the two nodes connected by it. The direction of the arc implies that the variable corresponding to the head node of the arc is conditioned by the variable corresponding to the tail node. In general, the joint probabilistic distribution of an n-variate random vector $\mathbf{X} = (X_1, \ldots, X_n)$ described by a Bayesian network can be calculated as

$$P(\mathbf{x}) = \prod_{i=1}^{n} P\left(x_i \mid pa(x_i)\right). \tag{3.1}$$

In this formulation, $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of realized values for $\mathbf{X}$; $pa(x_i)$ is a set of realized values for the parents (in a Bayesian network, if there exists a directed arc pointing from node X_j to X_i, then X_j is called a parent of X_i) of the random variable X_i.

A detailed introduction to the PMBGA can be found in [32]. Some important advances and interesting applications of EDA are covered in [33, 34]. Successes in utilizing EDA to solve scheduling problems have been reported in [35, 36] (for flow shop scheduling).

4. The Proposed PMBGA for Solving TWT-JSSP

4.1. Encoding

The proposed PMBGA relies on dispatching rules to record the scheduling policies. Eight scheduling rules are involved in this study. In the following expressions for the priority index, operation i belongs to job j, and w_j and d_j are the corresponding job's weight and due date. $JS(i)$ and $JP(i)$, respectively, denote the set of job successors of operation i and the set of job predecessors of operation i. Z is used to indicate that the operation with the smallest index will be chosen from the conflict set, while Z' is used to indicate that the operation with the largest index will be chosen from the conflict set:

(1) ATC (apparent tardiness cost): $Z_i'(t) = (w_j/p_i) \times \exp\{-(d_j - r_i - p_i - \sum_{i' \in JS(i)} (\widehat{W}_{i'} + p_{i'}))^+/(K \times \overline{p})\}$ (in this expression, r_i denotes the earliest starting time of operation i, and $\overline{p}$ denotes the average processing time of the current operations in the conflict set. K is a scaling parameter (or called "look-ahead" parameter). $\widehat{W}_i$ is the estimated lead time of operation i);

(2) SPT (shortest processing time): $Z_i = p_i$;

(3) LPT (longest processing time): $Z_i' = p_i$;

(4) WSPT (weighted shortest processing time): $Z_i' = w_j/p_i$;

(5) SRPT (shortest remaining processing time): $Z_i = p_i + \sum_{i' \in JS(i)} p_{i'}$;

(6) LRPT (longest remaining processing time): $Z_i' = p_i + \sum_{i' \in JS(i)} p_{i'}$;

(7) EDD (earliest due date): $Z_i = d_j$;

(8) ODD (operation due date): $Z_i(t) = [d_j / \sum_{i' \in JS(i) \cup \{i\}} p_{i'}] \times \sum_{i' \in JP(i) \cup \{i\}} p_{i'}$.

In PMBGA, each encoding digit (i.e., gene) is expressed by the serial number ($1 \sim 8$) of the selected dispatching rule. A solution is represented by a rule sequence $\{R_{ik} : i = 1, \ldots, n,\ k = 1, \ldots, m\}$, where R_{ik} indicates the rule to use when scheduling the ith operation on machine k. Therefore, the encoding length for each solution is $l = n \times m$.

4.2. Decoding

In order to evaluate the fitness of a solution, the Giffler-Thompson algorithm is applied to construct an active schedule based on the specified dispatching rules. The implementation of the Giffler-Thompson algorithm is detailed below.

Input: A sequence of rules $\{R_{ik} : i = 1, \ldots, n,\ k = 1, \ldots, m\}$.

Step 1. Let $Q(1) = O = \{1, \ldots, nm\}$ (the set of all operations), $R(1) = F(O) = \{f_1, \ldots, f_n\}$ (the set of first operations of each job). Set $t = 1$ and $\pi_k = 1$ $(k = 1, \ldots, m)$.

Step 2. Find the operation $i^* = \arg\min_{i \in R(t)} \{r_i + p_i\}$, and let m^* be the index of the machine on which this operation should be processed. Use $B(t)$ to denote all the operations from $R(t)$ which should be processed on machine m^*.

Step 3. Delete from $B(t)$ the operations that satisfy $r_i \geqslant r_{i^*} + p_{i^*}$.

Step 4. Use rule $R_{\pi_{m^*} m^*}$ to identify an operation $\hat{o}$ from $B(t)$ if currently there are more than one candidates. Schedule operation $\hat{o}$ on machine m^* at the earliest possible time. Let $\pi_{m^*} \leftarrow \pi_{m^*} + 1$.

Step 5. Let $Q(t+1) = Q(t) \setminus \{\hat{o}\}$, $R(t+1) = R(t) \setminus \{\hat{o}\} \cup \{\mathrm{suc}(\hat{o})\}$, where $\mathrm{suc}(\hat{o})$ denotes the immediate job successor of operation $\hat{o}$ (if any).

Step 6. If $Q(t+1) \neq \emptyset$, set $t \leftarrow t + 1$ and go to Step 2. Otherwise, the decoding procedure is terminated.

In the above description, the release time r_i equals the earliest possible starting time of operation i (determined from the already scheduled operations). So, $(r_i + p_i)$ is the earliest possible completion time of operation i. $Q(t)$ represents the set of operations yet to be scheduled at iteration t, while $R(t)$ represents the set of ready operations (whose job predecessors have all been scheduled) at iteration t. In Step 4, the operation set $B(t)$ is also called a conflict set.

4.3. Producing Offspring Individuals

(a) *Selection*

In each iteration, we first sort all the individuals in the current population according to their fitness. Then, we select the best $1/4$ of individuals to form the set S, which will subsequently be used to build the Bayesian network in order to produce a new generation of individuals.

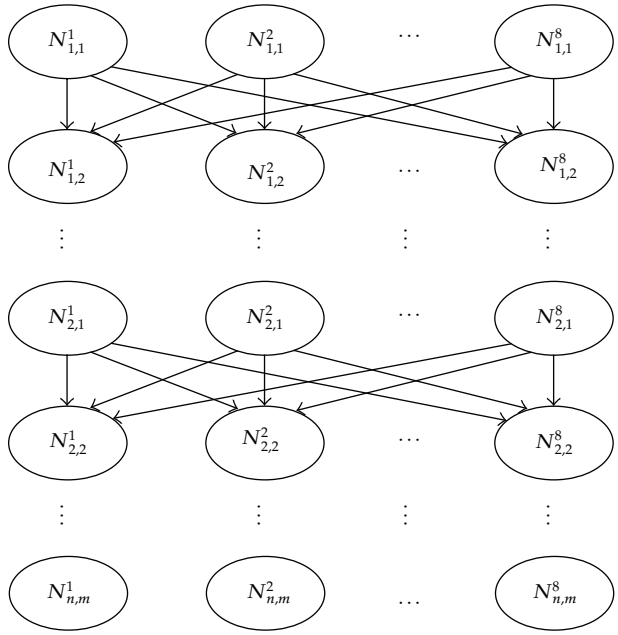

Figure 1: The structure of the Bayesian network used in PMBGA.

(b) *The Adopted Network Structure*

A Bayesian network has to be built to describe the probabilistic distribution of favorable R_{ik} settings based on the elite individuals in S. Each individual can be characterized by a directed acyclic network as shown in Figure 1. In this network, a node $N^r_{i,k}$ ($r \in \{1, 2, \ldots, 8\}$) indicates the fact that the dispatching rule used to schedule the ith operation on machine k is selected as rule Number r. The directed arc from node $N^r_{i,k}$ to node $N^{r'}_{i,k+1}$ (or $N^{r'}_{i+1,1}$) represents the dependency between the two nodes, so it characterizes the possible influence of the rule selection for the ith operation on machine k on the rule selection for the ith operation on machine $k + 1$ (or the $(i + 1)$-th operation on machine 1). Therefore, a directed path from a certain node in the first row to a certain node in the $(n \times m)$-th row can completely describe an individual in the population (because a directed path records an instantiation of all the rule selections).

(c) *Calculation of the Probability Values*

Since we adopt a fixed network structure in PMBGA, building the Bayesian network is equivalent to determining the values of all the conditional probabilities according to the selected solution set S. After that, new individuals will be produced by iteratively sampling these probabilistic distributions, expecting to obtain high-quality offsprings.

Given a number of individuals (i.e., the training set S), an estimate of the conditional probabilities can be obtained simply by counting the frequencies of occurrence.

Example 4.1. Here, we provide a concrete example to illustrate the probability calculation process. For a PMBGA optimization instance with $n = 3, m = 1$ and $r = 3$, let us further suppose the current S contains 40 individuals. In Figure 2, the statistics of these individuals

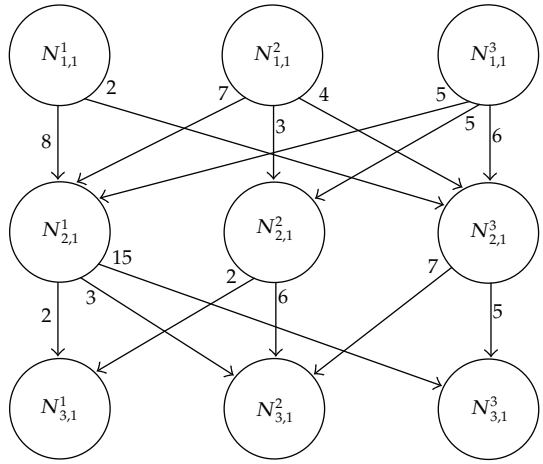

Figure 2: An example network in PMBGA.

are displayed on a network as previously defined. The weight of each arc (the number placed on the arc) indicates the occurring frequency of this arc (counted for all the individuals in S). For example, if there exists an individual coded as "$[3,1,2]$", then the path "$N^3_{1,1} \rightarrow N^1_{2,1} \rightarrow N^2_{3,1}$" in the Bayesian network is used to record this individual, and consequently, the weights (counted frequencies) of the arcs $N^3_{1,1} \rightarrow N^1_{2,1}$ and $N^1_{2,1} \rightarrow N^2_{3,1}$ will be increased by 1, respectively. Note that, in the final network, the sum of the weights of all the incoming arcs of a certain node should be equal to the sum of the weights of all the outgoing arcs of the same node. This is because each individual corresponds to a complete path from the first row to the last row.

By using frequency as an approximation for probability, the relevant (conditional) probabilities should be calculated as follows:

$$
\begin{gathered}
P\left(N^1_{1,1}\right) = \frac{8+2}{40}, \qquad P\left(N^2_{1,1}\right) = \frac{7+3+4}{40}, \qquad P\left(N^3_{1,1}\right) = \frac{5+5+6}{40}, \\
P\left(N^1_{2,1} \mid N^1_{1,1}\right) = \frac{8}{8+2}, \qquad P\left(N^2_{2,1} \mid N^1_{1,1}\right) = 0, \qquad P\left(N^3_{2,1} \mid N^1_{1,1}\right) = \frac{2}{8+2}, \\
P\left(N^1_{2,1} \mid N^2_{1,1}\right) = \frac{7}{7+3+4}, \qquad P\left(N^2_{2,1} \mid N^2_{1,1}\right) = \frac{3}{7+3+4}, \qquad P\left(N^3_{2,1} \mid N^2_{1,1}\right) = \frac{4}{7+3+4}, \\
P\left(N^1_{2,1} \mid N^3_{1,1}\right) = \frac{5}{5+5+6}, \qquad P\left(N^2_{2,1} \mid N^3_{1,1}\right) = \frac{5}{5+5+6}, \qquad P\left(N^3_{2,1} \mid N^3_{1,1}\right) = \frac{6}{5+5+6}, \\
P\left(N^1_{3,1} \mid N^1_{2,1}\right) = \frac{2}{2+3+15}, \qquad P\left(N^2_{3,1} \mid N^1_{2,1}\right) = \frac{3}{2+3+15}, \qquad P\left(N^3_{3,1} \mid N^1_{2,1}\right) = \frac{15}{2+3+15}, \\
P\left(N^1_{3,1} \mid N^2_{2,1}\right) = \frac{2}{2+6}, \qquad P\left(N^2_{3,1} \mid N^2_{2,1}\right) = \frac{6}{2+6}, \qquad P\left(N^3_{3,1} \mid N^2_{2,1}\right) = 0, \\
P\left(N^1_{3,1} \mid N^3_{2,1}\right) = 0, \qquad P\left(N^2_{3,1} \mid N^3_{2,1}\right) = \frac{7}{7+5}, \qquad P\left(N^3_{3,1} \mid N^3_{2,1}\right) = \frac{5}{7+5}.
\end{gathered}
\tag{4.1}
$$

According to the above calculation method, a connection can never be rediscovered in the PMBGA if the corresponding conditional probability is zero (e.g., from $N^2_{2,1}$ to $N^3_{3,1}$). To

overcome this drawback, we can set the minimum count to 1. Taking $N_{2,1}^2$ as an example, the conditional probabilities for the outgoing arcs will then become

$$P\left(N_{3,1}^1 \mid N_{2,1}^2\right) = \frac{3}{3+7+1}, \qquad P\left(N_{3,1}^2 \mid N_{2,1}^2\right) = \frac{7}{3+7+1}, \qquad P\left(N_{3,1}^3 \mid N_{2,1}^2\right) = \frac{1}{3+7+1}. \tag{4.2}$$

Now it is possible to discover $N_{3,1}^3$ from $N_{2,1}^2$, though the probability is small.

(d) *The Sampling Process*

The sampling process for generating a new individual begins from the root nodes of the Bayesian network. By selecting an outgoing arc at each node based on the calculated conditional probabilities, the whole network can be gradually instantiated.

4.4. The Embedded Local Search

The search mechanism of GA guarantees a good performance in the "exploration" of the solution space. However, it has been reported that GA alone cannot achieve satisfactory solution quality for complex optimization problems. Actually, a local search procedure is usually added within the framework of GA in order to provide reliable "exploitation" ability. In this paper, we design such a local optimizer, which attempts to improve the selected solutions. In each iteration of PMBGA, the local search is carried out for the best e% of solutions in the current population. Thus, e is an important parameter for adjusting the frequency of local search and achieving a balance between exploration and exploitation.

In the following, we will describe how to perform the local search on a given solution $\mathbf{s}$ (a sequence of rules).

Step 1. Use the Giffler-Thompson algorithm to generate an active schedule σ_1 based on $\mathbf{s}$. The objective value of the obtained schedule is TWT_{σ_1}.

Step 2. Set $u = 1$.

Step 3. Exert random perturbations on the processing times: generate a new set of processing times $\{p_{jk}^{(u)}\}$ from the normal distribution $\mathcal{N}(p_{jk}, (0.2p_{jk})^2)$.

Step 4. Use the Giffler-Thompson algorithm to generate an active schedule σ_2 based on $\mathbf{s}$. Note that, in this process, the processing time of each operation takes its new value, that is, $p_{jk}^{(u)}$.

Step 5. Evaluate the objective value of σ_2 under the original processing times, obtaining TWT_{σ_2}.

Step 6. If $\mathrm{TWT}_{\sigma_2} < \mathrm{TWT}_{\sigma_1}$, exit the local search. Otherwise, continue with Step 7.

Step 7. Let $u \leftarrow u + 1$. If $u > U$, terminate the local search procedure with no improvement found. Otherwise, go back to Step 3.

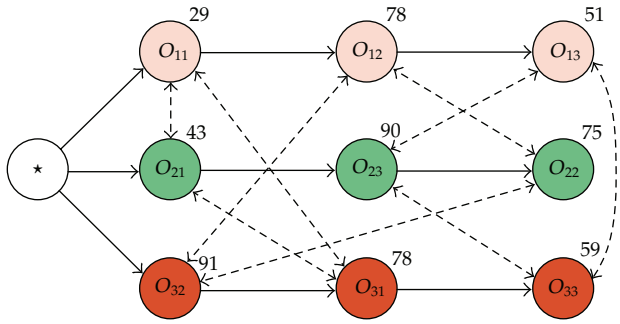

Figure 3: The disjunctive graph of the example instance.

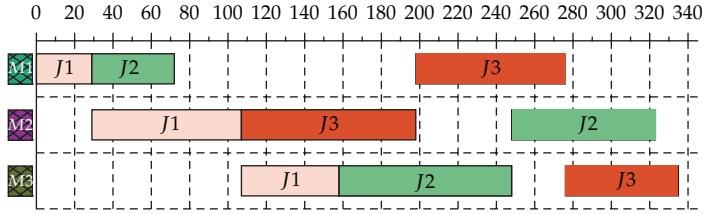

Figure 4: The Gantt chart for the initial schedule.

The local search procedure attempts to find a better schedule from an initial starting solution. When the processing time of each operation varies due to the random perturbations in Step 3, different schedules may be obtained by applying the same set of dispatching rules. This is the fundamental idea of the local search procedure. The relative deviation is set as 0.2 times the mean, which can produce a moderate level of perturbation. If the variance is too large or too small, the local search will be inefficient. Finally, if it turns out that $\mathrm{TWT}_{\sigma_2} < \mathrm{TWT}_{\sigma_1}$, which means that the local search has found an improvement, then the original schedule should be accordingly revised. On the other hand, if no better schedule is found within U trials, the local search will quit, leaving the current solution unchanged.

Example 4.2. Here, we provide a concrete example to illustrate the local search procedure. In the 3×3 TWT-JSSP instance, the processing time of each operation is marked beside the corresponding node in Figure 3 (the disjunctive graph). The due dates of each job are set as $d_1 = 103$, $d_2 = 146$, and $d_3 = 137$. The weights of each job are set as $w_1 = 1$, $w_2 = 7$, and $w_3 = 4$.

Suppose the initial solution is $\mathbf{s} = [2, 2, \ldots]$, which indicates using the SPT rule at all times.

First, the solution is decoded by applying the Giffler-Thompson algorithm under the original values of processing times. The following schedule is obtained:

$$\sigma_1 = \begin{bmatrix} O_{11} & O_{21} & O_{31} \\ O_{12} & O_{32} & O_{22} \\ O_{13} & O_{23} & O_{33} \end{bmatrix}. \tag{4.3}$$

The objective value is $\mathrm{TWT}_{\sigma_1} = 2086$ (see Figure 4).

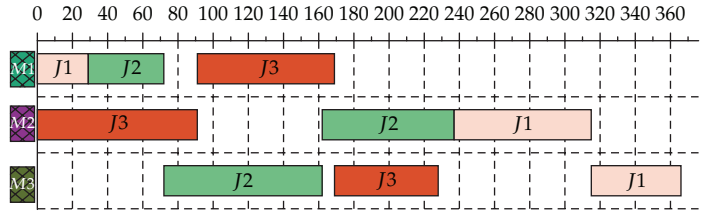

Figure 5: The Gantt chart for the new schedule generated by random perturbation.

Next, we add random perturbations to the processing times and thus generate different new instances of the JSSP. In one of these instances, the processing time of each operation is as follows:

$$\begin{aligned} &p_{11} = 29.0, \quad p_{12} = 85.0, \quad p_{13} = 58.2, \\ &p_{21} = 52.3, \quad p_{23} = 81.9, \quad p_{22} = 65.3, \\ &p_{32} = 78.8, \quad p_{31} = 84.0, \quad p_{33} = 50.3. \end{aligned} \tag{4.4}$$

The solution **s** is decoded again by applying the Giffler-Thompson algorithm under the above new values of p_{jk}. The following schedule can be obtained:

$$\sigma_2 = \begin{bmatrix} O_{11} & O_{21} & O_{31} \\ O_{32} & O_{22} & O_{12} \\ O_{23} & O_{33} & O_{13} \end{bmatrix}, \tag{4.5}$$

when evaluated under the original processing times, the objective value of σ_2 is $\mathrm{TWT}_{\sigma_2} = 1264$ (see Figure 5).

Therefore, an improvement has been found on the initial solution (because $1264 < 2086$).

5. Computational Results

In order to test the performance of the proposed PMBGA, the same benchmark instances as in [15, 19] are used in our computational experiment. In these instances, the due date of each job is set as $d_j = f \times P_j$, where P_j denotes the total processing time of job j, and $f \in \{1.6, 1.5, 1.3\}$ is a coefficient that controls the tightness level of the due date setting. The first 20% of jobs are assigned weighting 4 (very important), the next 60% are assigned weighting 2, and the remaining jobs are assigned weighting 1 (not important). That is, $w_1 = w_2 = 4$, $w_3 = \cdots = w_8 = 2$, and $w_9 = w_{10} = 1$.

Based on extensive computational tests, the algorithm parameters are set as follows. The population size PS = 50; the proportion of individuals selected for local search $e\% = 30\%$; the maximum number of random perturbations $U = 100$. In the implementation of the ATC rule, we set $K = 2$ and $\widehat{W}_i = 0.4\ p_i$.

We compare the performance of the proposed PMBGA with the hybrid optimization algorithm PSO-SA [37] and a rule-based genetic algorithm (RBGA) which uses the same dispatching rules for encoding but adopts classical crossover and mutation operators. In

Table 1: The computational results and comparisons under $f = 1.6$.

Instance	Optimal	PMBGA		PSO-SA		RBGA	
		Average	# opt	Average	# opt	Average	# opt
ABZ5	0	0.0	20	0.0	20	0.0	20
ABZ6	0	0.0	20	0.0	20	0.0	20
LA16	0	0.0	20	0.0	20	0.3	18
LA17	65	65.0	20	65.0	20	65.0	20
LA18	0	0.0	20	0.0	20	0.0	20
LA19	0	0.0	20	0.0	20	0.8	18
LA20	0	0.0	20	0.0	20	0.0	20
LA21	0	0.0	20	0.0	20	0.0	20
LA22	0	0.0	20	0.2	19	0.0	20
LA23	0	0.0	20	0.0	20	0.0	20
LA24	0	0.0	20	0.0	20	0.0	20
MT10	141	143.1	16	145.9	14	149.9	11
ORB1	566	588.7	4	602.0	1	630.7	0
ORB2	44	44.5	18	45.7	13	45.6	12
ORB3	422	425.2	10	428.8	8	452.1	8
ORB4	66	66.3	17	66.7	18	71.0	13
ORB5	163	166.0	6	168.9	4	171.8	1
ORB6	28	28.5	15	28.9	10	29.1	12
ORB7	0	0.0	20	0.7	15	0.7	13
ORB8	621	633.4	2	646.7	0	661.4	0
ORB9	66	66.9	16	68.7	12	71.1	9
ORB10	76	83.2	2	83.5	0	84.2	0

order to make the comparisons meaningful, we set a computational time limit for all the algorithms. In the following, the time limit for solving each instance is determined as 60 seconds. Each algorithm is run for 20 independent times on each TWT-JSSP instance. Tables 1, 2, and 3 report the average objective value obtained from the 20 runs. "# opt" indicates the number of times that the optimum (the optimum for each instance (listed in the second column in the tables) is first given by the branch-and-bound algorithm [14] and recently updated by the hybrid genetic algorithm [21]) has been reached during the 20 runs. The results demonstrate the superiority of the PMBGA.

According to the computational results, the proposed PMBGA systematically outperforms the comparative methods. In addition, the following comments can be made:

(1) The proposed PMBGA performs better than the PSO-SA which adopts operation permutation-based encoding scheme. The advantage of PMBGA is even stronger when the due dates in the TWT-JSSP instances are set tighter. This suggests that the rule-based optimization approach is more effective than sequence-based search when many jobs are prone to be tardy and thus are competing fiercely for the limited machine resources. Applying dispatching rules turns out to be a satisfactory and robust strategy in this situation;

(2) Also, PMBGA outperforms RBGA to a greater extent. This reveals the effectiveness of the proposed approach from two aspects:

Table 2: The computational results and comparisons under $f = 1.5$.

Instance	Optimal	PMBGA		PSO-SA		RBGA	
		Average	# opt	Average	# opt	Average	# opt
ABZ5	69	69.0	20	69.3	18	69.5	17
ABZ6	0	0.0	20	0.0	20	0.5	17
LA16	166	166.0	20	167.9	15	166.9	17
LA17	260	260.0	20	260.7	19	262.1	15
LA18	34	34.6	18	34.5	18	34.7	18
LA19	21	21.0	20	21.1	19	21.3	16
LA20	0	0.0	20	0.1	19	0.7	14
LA21	0	0.0	20	0.0	20	0.6	15
LA22	196	196.0	20	196.5	17	197.1	15
LA23	2	2.0	20	2.0	20	2.0	20
LA24	82	82.1	19	85.9	12	87.5	8
MT10	394	394.2	19	407.7	7	414.5	1
ORB1	1098	1108.4	13	1131.5	1	1163.0	0
ORB2	292	292.0	20	299.1	5	310.9	2
ORB3	918	922.9	10	937.7	6	989.0	0
ORB4	358	360.1	17	366.2	3	381.2	1
ORB5	405	405.3	18	414.8	7	433.8	2
ORB6	426	427.1	15	441.8	8	455.5	4
ORB7	50	50.1	16	52.3	7	53.4	6
ORB8	1023	1032.3	14	1060.2	10	1082.5	6
ORB9	297	308.8	12	313.8	9	329.2	5
ORB10	346	377.1	3	388.7	0	396.8	0

(i) Since RBGA does not involve a local search module, the results show that the specialized local search procedure can help to promote the overall performance of GA. In particular, the local optimizer in PMBGA is closely based on the specific characteristics of the considered JSSP instance: the magnitude of the random sampling is consistent with the processing time (cf. Step 3 of the procedure), which ensures a reasonable size of the search scope.

(ii) The results show that using the estimation of distribution principle to optimize the rule combinations is more effective than the traditional crossover and mutation operators. Noticeably, the essential point in this process is to model the relationship and interactions between the rule selections for different jobs and different machines.

Meanwhile, we also test the impact of the parameter e on the final solution quality of PMBGA. A reasonable selection of e will result in an effective balance between exploration and exploitation. In the following experiment, instance ORB1 under $f = 1.5$ is used and the time limit is set as 40 sec and 60 sec, respectively. The computational results are displayed in Figure 6, where the vertical axis gives the average objective value obtained from 20 independent executions of the proposed PMBGA under each e.

According to the results, the setting of e has a considerable impact on the solution quality, especially when the computational time is scarce (40 sec), which verifies that the proposed local search module is effective in accelerating the overall convergence of PMBGA.

Table 3: The computational results and comparisons under $f = 1.3$.

Instance	Optimal	PMBGA		PSO-SA		RBGA	
		Average	# opt	Average	# opt	Average	# opt
ABZ5	1403	1415.1	12	1458.9	1	1466.9	1
ABZ6	436	436.0	20	446.9	5	456.3	2
LA16	1169	1170.2	19	1212.3	3	1239.5	0
LA17	899	900.4	17	914.7	7	942.5	3
LA18	929	943.0	15	947.6	2	960.1	1
LA19	948	950.6	18	967.0	10	989.9	6
LA20	805	822.1	14	829.3	8	840.5	4
LA21	463	466.8	15	480.0	12	490.5	7
LA22	1064	1095.7	10	1103.9	8	1120.9	3
LA23	835	837.5	12	866.4	2	867.8	0
LA24	835	844.0	13	855.6	4	860.5	1
MT10	1363	1375.0	17	1403.9	1	1436.4	0
ORB1	2568	2613.3	5	2617.2	7	2718.6	2
ORB2	1408	1440.5	4	1452.0	3	1494.7	1
ORB3	2111	2132.9	8	2154.8	3	2205.6	0
ORB4	1623	1637.1	11	1656.1	5	1707.1	2
ORB5	1593	1617.4	3	1638.7	3	1673.8	0
ORB6	1790	1835.9	9	1837.3	7	1884.0	3
ORB7	590	592.3	17	602.2	10	620.0	5
ORB8	2429	2483.1	0	2509.6	0	2563.7	0
ORB9	1316	1316.7	19	1359.6	12	1393.8	8
ORB10	1679	1683.9	17	1709.9	9	1747.1	3

A small e means that only a few solutions in each generation can be improved by the local search, which has little effect on the entire population. A large e suggests that too much time is consumed on local search, which may impair the normal function of PMBGA because of the reduced generations.

The best setting of e under each constraint level is 60 (for tight time budget) and 30 (for loose time budget). When the exogenous restriction on computational time is tight, PMBGA has to rely on frequent local search to find good solutions. This is because, in the short term, local search is more efficient than PMBGA's mechanism (Bayesian network modeling) in improving a solution. However, the price to pay is possibly a premature convergence of the whole optimization process. On the other hand, when the computational time is more sufficient, PMBGA will prefer a larger number of generations to conduct a systematic exploration of the solution space. In this case, the local search need not be used very frequently, otherwise the steady searching and learning process may be disturbed.

Finally, we observe the impact of the parameters in the ATC rule. We write the estimated waiting time $\widehat{W}_i$ of operation i as proportional to its processing time: $\widehat{W}_i = b \times p_i$. Now, we use instance MT10 with $f = 1.3$ to test the two parameters, K and b. The average objective values obtained by PMBGA under each (K, b) combination is shown in Figure 7.

In fact, the influence of the ATC parameters is not so significant as the other parameters. But a clearly inferior setting is $K = 1$, which could be eliminated. Based on additional experimental results which are not listed here, both $K = 2$ and $K = 3$ seem

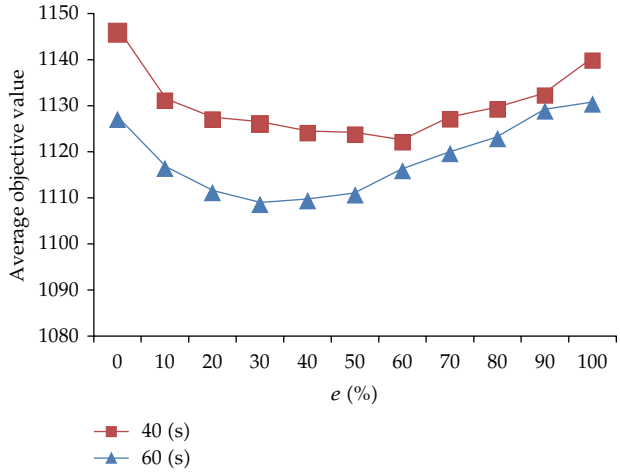

Figure 6: The influence of parameter e.

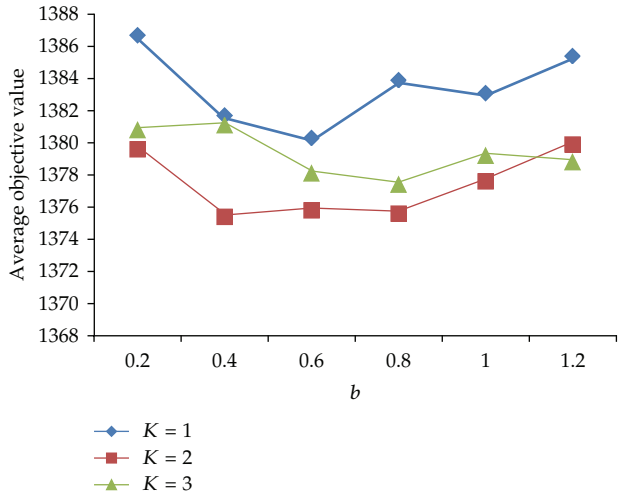

Figure 7: The influence of parameters (K, b).

acceptable. Overall, setting $K = 2$ and $b = 0.4$ yield satisfactory solution quality for most scheduling instances involved in this study.

6. Conclusion

In this paper, we propose a new probabilistic model-building genetic algorithm for solving the job shop scheduling problem under the total weighted tardiness criterion. Since the TWT objective is systematically more difficult to optimize than the conventional makespan objective, we rely on some dispatching rules for schedule construction. PMBGA is used to search for good combinations of these rules, and a specific local search algorithm is embedded into the optimization framework. The computational experiments have shown the superiority of the proposed methodology.

Future research can be conducted from the following aspects. Although the standard Bayesian network is capable of modeling the interactions between decision variables, it is necessary to improve the computational efficiency in the rule optimization process. Meanwhile, it is interesting to try other encoding schemes, which may be beneficial for the discovery of useful structural properties and may enhance the overall performance of the PMBGA.

Acknowledgment

This paper was supported by the National Natural Science Foundation of China (Grant nos. 61104176, and 60874071).

References

[1] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.
[2] Y. Mati, S. Dauzère-Pérès, and C. Lahlou, "A general approach for optimizing regular criteria in the job-shop scheduling problem," *European Journal of Operational Research*, vol. 212, no. 1, pp. 33–42, 2011.
[3] R. Zhang and C. Wu, "A hybrid immune simulated annealing algorithm for the job shop scheduling problem," *Applied Soft Computing Journal*, vol. 10, no. 1, pp. 79–89, 2010.
[4] L. de Giovanni and F. Pezzella, "An Improved Genetic Algorithm for the Distributed and Flexible Job-shop Scheduling problem," *European Journal of Operational Research*, vol. 200, no. 2, pp. 395–408, 2010.
[5] C. Vela, R. Varela, and M. González, "Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times," *Journal of Heuristics*, vol. 16, no. 2, pp. 139–165, 2010.
[6] M. Liu, Z. Sun, J. Yan, and J. Kang, "An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem," *Expert Systems with Applications*, vol. 38, no. 8, pp. 9248–9255, 2011.
[7] E. Nowicki and C. Smutnicki, "An advanced tabu search algorithm for the job shop problem," *Journal of Scheduling*, vol. 8, no. 2, pp. 145–159, 2005.
[8] J.-Q. Li, Q.-K. Pan, P. N. Suganthan, and T. J. Chua, "A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 52, no. 5–8, pp. 683–697, 2011.
[9] D. Y. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers and Industrial Engineering*, vol. 51, no. 4, pp. 791–808, 2006.
[10] R. Zhang and C. Wu, "A divide-and-conquer strategy with particle swarm optimization for the job shop scheduling problem," *Engineering Optimization*, vol. 42, no. 7, pp. 641–670, 2010.
[11] M. Seo and D. Kim, "Ant colony optimisation with parameterised search space for the job shop scheduling problem," *International Journal of Production Research*, vol. 48, no. 4, pp. 1143–1154, 2010.
[12] L.-N. Xing, Y.-W. Chen, P. Wang, Q.-S. Zhao, and J. Xiong, "A knowledge-based ant colony optimization for flexible job shop scheduling problems," *Applied Soft Computing Journal*, vol. 10, no. 3, pp. 888–896, 2010.
[13] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*, Springer, New York, NY, USA, 3rd edition, 2008.
[14] M. Singer and M. Pinedo, "A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops," *IIE Transactions*, vol. 30, no. 2, pp. 109–118, 1998.
[15] M. Pinedo and M. Singer, "A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop," *Naval Research Logistics*, vol. 46, no. 1, pp. 1–17, 1999.
[16] S. J. Mason, J. W. Fowler, and W. M. Carlyle, "A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops," *Journal of Scheduling*, vol. 5, no. 3, pp. 247–262, 2002.
[17] L. Mönch and R. Drießel, "A distributed shifting bottleneck heuristic for complex job shops," *Computers and Industrial Engineering*, vol. 49, no. 3, pp. 363–380, 2005.
[18] L. Mönch, R. Schabacker, D. Pabst, and J. W. Fowler, "Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2100–2118, 2007.
[19] S. Kreipl, "A large step random walk for minimizing total weighted tardiness in a job shop," *Journal of Scheduling*, vol. 3, no. 3, pp. 125–138, 2000.

[20] H. Zhou, W. Cheung, and L. C. Leung, "Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm," *European Journal of Operational Research*, vol. 194, no. 3, pp. 637–649, 2009.
[21] I. Essafi, Y. Mati, and S. Dauzère-Pérès, "A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem," *Computers and Operations Research*, vol. 35, no. 8, pp. 2599–2616, 2008.
[22] K. M. J. Bontridder, "Minimizing total weighted tardiness in a generalized job shop," *Journal of Scheduling*, vol. 8, no. 6, pp. 479–496, 2005.
[23] R. Tavakkoli-Moghaddam, M. Khalili, and B. Naderi, "A hybridization of simulated annealing and electromagnetic-like mechanism for job shop problems with machine availability and sequence-dependent setup times to minimize total weighted tardiness," *Soft Computing*, vol. 13, no. 10, pp. 995–1006, 2009.
[24] S. U. Randhawa and Y. Zeng, "Job shop scheduling: an experimental investigation of the performance of alternative scheduling rules," *Production Planning and Control*, vol. 7, no. 1, pp. 47–56, 1996.
[25] O. Holthaus and H. Ziegler, "Improving job shop performance by coordinating dispatching rules," *International Journal of Production Research*, vol. 35, no. 2, pp. 539–549, 1997.
[26] A. El-Bouri and P. Shah, "A neural network for dispatching rule selection in a job shop," *International Journal of Advanced Manufacturing Technology*, vol. 31, no. 3, pp. 342–349, 2006.
[27] T. C. Chiang and L. C. Fu, "A rule-centric memetic algorithm to minimize the number of tardy jobs in the job shop," *International Journal of Production Research*, vol. 46, no. 24, pp. 6913–6931, 2008.
[28] J. A. Vázquez-Rodríguez and S. Petrovic, "A new dispatching rule based genetic algorithm for the multi-objective job shop problem," *Journal of Heuristics*, vol. 16, no. 6, pp. 771–793, 2010.
[29] A. Baykasoğlu, M. Göçken, and L. Özbakir, "Genetic programming based data mining approach to dispatching rule selection in a simulated job shop," *Simulation*, vol. 86, no. 12, pp. 715–728, 2010.
[30] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.
[31] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: the combination of knowledge and statistical data," *Machine Learning*, vol. 20, no. 3, pp. 197–243, 1995.
[32] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.
[33] P. Larranãga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Optimization*, Kluwer Academic, Boston, Mass, USA, 2002.
[34] J. A. Lozano, P. Larranãga, I. Inza, and E. E. Bengoetxea, *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*, Springer, Berlin, Germany, 2006.
[35] S. Tsutsui and M. Miki, "Solving flow shop scheduling problems with probabilistic model-building genetic algorithms using edge histograms," in *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 465–471, 2002.
[36] B. Jarboui, M. Eddaly, and P. Siarry, "An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems," *Computers and Operations Research*, vol. 36, no. 9, pp. 2638–2646, 2009.
[37] W. Xia and Z. Wu, "A hybrid particle swarm optimization approach for the job-shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 29, no. 3-4, pp. 360–366, 2006.