Entropy **2011**, *13*, 1708-1729; doi:10.3390/e13091708

OPEN ACCESS

entropy

ISSN 1099-4300

www.mdpi.com/journal/entropy

Article

An Artificial Bee Colony Algorithm for the Job Shop Scheduling Problem with Random Processing Times

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;
E-Mail: wuc@tsinghua.edu.cn

⋆ Author to whom correspondence should be addressed; E-Mail: r.zhang@ymail.com;
Tel.: +86-13687918983; Fax: +86-791-83969463.

Received: 13 July 2011; in revised form: 9 September 2011 / Accepted: 9 September 2011 / Published: 19 September 2011

Abstract: Due to the influence of unpredictable random events, the processing time of each operation should be treated as random variables if we aim at a robust production schedule. However, compared with the extensive research on the deterministic model, the stochastic job shop scheduling problem (SJSSP) has not received sufficient attention. In this paper, we propose an artificial bee colony (ABC) algorithm for SJSSP with the objective of minimizing the maximum lateness (which is an index of service quality). First, we propose a performance estimate for preliminary screening of the candidate solutions. Then, the K-armed bandit model is utilized for reducing the computational burden in the exact evaluation (through Monte Carlo simulation) process. Finally, the computational results on different-scale test problems validate the effectiveness and efficiency of the proposed approach.

Keywords: shop scheduling; artificial bee colony algorithm; maximum lateness; simulation

1. Introduction

The job shop scheduling problem (JSSP) has been known as an extremely stubborn combinatorial optimization problem since the 1950s. In terms of computational complexity, JSSP is $\mathcal{NP}$-hard in the strong sense [1]. It models an important decision problem in contemporary manufacturing systems, so extensive research has been conducted on JSSP. Due to its high complexity, the recent research

has focused on the meta-heuristic approaches, such as genetic algorithm (GA) [2], estimation of distribution algorithm (EDA) [3], tabu search (TS) [4], particle swarm optimization (PSO) [5], ant colony optimization (ACO) [6], artificial bee colony (ABC) algorithm [7].

However, most existing research is based on the standard JSSP model, which means it is impossible to directly apply these algorithms in real-world scheduling scenarios. Especially, we emphasize the following two points.

(1) Most research on JSSP has focused on the makespan criterion (*i.e.*, minimizing the maximum completion time). However, in the make-to-order (MTO) manufacturing environment, due date related performances are apparently more relevant for decision makers, because the in-time delivery of goods is vital for maintaining a high service reputation. Therefore, the research that aims at minimizing lateness/tardiness in JSSP deserves more attention.

(2) Most existing algorithms are designed for the deterministic JSSP, in which all the data (e.g., the processing times) are assumed to be fixed and precisely known in advance. In real-world manufacturing, however, the processing of operations is constantly affected by uncertain factors. Machine breakdowns, worker absenteeism, order changes, *etc*. can all lead to variations in the operation times. In this case, solving the deterministic JSSP will not result in a robust production schedule. Therefore, it is rewarding to focus more research effort on the stochastic JSSP.

In an attempt to overcome the drawbacks, in this paper we study the stochastic job shop scheduling problem (SJSSP) with the objective of minimizing maximum lateness ($L_{\max}$) in the sense of expectation. Compared with the standard JSSP, the studied SJSSP moves a step closer to the practical scheduling features. The main contribution of this paper is an efficient optimization approach based on artificial bee colony for solving SJSSP. The algorithm first uses a quick-and-dirty performance estimate to roughly evaluate the candidate solutions. Then, the selected promising solutions undergo a more accurate evaluation through simulation. In this process, a new computing budget allocation policy is adopted in order to promote the computational efficiency.

The rest of the paper is organized as follows. Section 2 briefly reviews the recent publications on stochastic job shop scheduling and artificial bee colony algorithms. Section 3 provides the formulation of SJSSP and an introduction to the standard ABC principles. Section 4 introduces an estimate for the objective function (maximum lateness). Section 5 describes the design of ABC for SJSSP in detail. Section 6 gives the computational results. Finally, some conclusions are made in Section 7.

2. Literature Review

2.1. The Stochastic Job Shop Scheduling Problem

Due to the inevitable uncertainties in manufacturing systems, SJSSP is more realistic than its deterministic counterpart. In SJSSP, the number of jobs is usually known in advance, while the processing time of each operation is a random variable with known probability distribution. The due dates can be regarded as either fixed or random variables, depending on the frequency of changes in customer orders. The aim is to find a feasible schedule (*i.e.*, sequence of operations) that minimizes the objective function in the sense of expectation.

In [8,9], simulation-based GAs are proposed for solving SJSSP with the expected makespan criterion. The individuals that appear through the generations with very high frequency are selected as good solutions. In [10], the authors study the SJSSP with just-in-time objective (completion before or after the due date will both result in a cost). Several decision-making rules are proposed for selecting a job when several jobs are competing for a machine. In [11], the goal is to minimize the processing time variations, the operational costs and the idle costs in SJSSP. A hybrid method is proposed, in which the initial solutions are first generated by a neural network and then improved by SA. Luh et al. [12] presents an algorithm based on Lagrangian relaxation and stochastic dynamic programming for SJSSP with uncertain arrival times, due dates, and part priorities. In [13], exact and heuristic algorithms are proposed to solve SJSSP with mean flow time criterion. The aim is to find a minimum set of schedules which contains at least one optimal schedule for any realization of the random processing times. Singer [14] presents a heuristic which amplifies the expected processing times by a factor and then applies a deterministic scheduling algorithm. Recently, Lei proposes a genetic algorithm for minimizing makespan in a stochastic job shop with machine breakdowns and non-resumable jobs [15]. The processing time, the downtime and the normal operation time between successive breakdowns are all assumed to follow exponential distributions. In addition, a multi-objective genetic algorithm is proposed in [16] for stochastic job shop scheduling problems in which the makespan and the total tardiness ratio should be minimized. In [17,18], quantum genetic algorithms are proposed to solve SJSSP with expected makespan criterion. In [19], a simulation-based decision support system is presented for the production control of a stochastic flexible job shop manufacturing system. In [20], an algorithm based on computer simulation and artificial neural networks is proposed to select the optimal dispatching rule for each machine from a set of rules in order to minimize the makespan in SJSSP.

Generally, the most critical factor that affects the efficiency of solving a stochastic combinatorial optimization problem (SCOP) is the evaluation of solutions [21]. Since the objective function is in the expectation form, the evaluation may not be so straightforward if no closed-form formula is available for calculating the expectation. Due to the complexity of the job shop model, we have to rely on Monte Carlo simulations to obtain an approximation for the expected $L_{\max}$. However, the simulation process is usually very time-consuming. To achieve a balance, we propose an artificial bee colony algorithm for solving SJSSP in this paper. Time-saving strategies have been designed and utilized in the searching process in order to ensure a satisfactory solution within reasonable computational time.

2.2. The Artificial Bee Colony Algorithm

The artificial bee colony (ABC) algorithm is a relatively new swarm intelligence based optimizer. It mimics the cooperative foraging behavior of a swarm of honey bees [22]. ABC was initially proposed by Karaboga in 2005 for optimizing multi-variable and multi-modal continuous functions [23]. The latest research has revealed some good properties of ABC [24–26]. Especially, the number of control parameters in ABC is fewer than that of other population-based algorithms, which makes it easier to be implemented. Meanwhile, the optimization performance of ABC is comparable and sometimes superior to the state-of-the-art meta-heuristics. Therefore, ABC has aroused much interest and has been successfully applied to different kinds of optimization problems [27–29].

The ABC algorithm systematically incorporates exploration and exploitation mechanisms, so it is suitable for complex scheduling problems. For example, Huang and Lin [30] proposed a new ABC for the open shop scheduling problem, which is more difficult to solve than JSSP due to the undecided precedence relations between job operations. In this paper, we will use ABC as the basic optimization framework in the process of solving SJSSP. To our knowledge, this is the first attempt that ABC is applied to a stochastic scheduling problem. Of course, some problem-specific information should be embedded into the searching process of ABC in order to promote the overall optimization efficiency for SJSSP.

3. The Preliminaries

3.1. Formulation of the SJSSP

In an SJSSP 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: (1) there is no machine breakdown; (2) no preemption of operations is allowed; (3) all jobs are released at time 0; (4) the transportation times and the setup times are all neglected; (5) each machine can process only one job at a time; (6) each job may be processed by only one machine at a time.

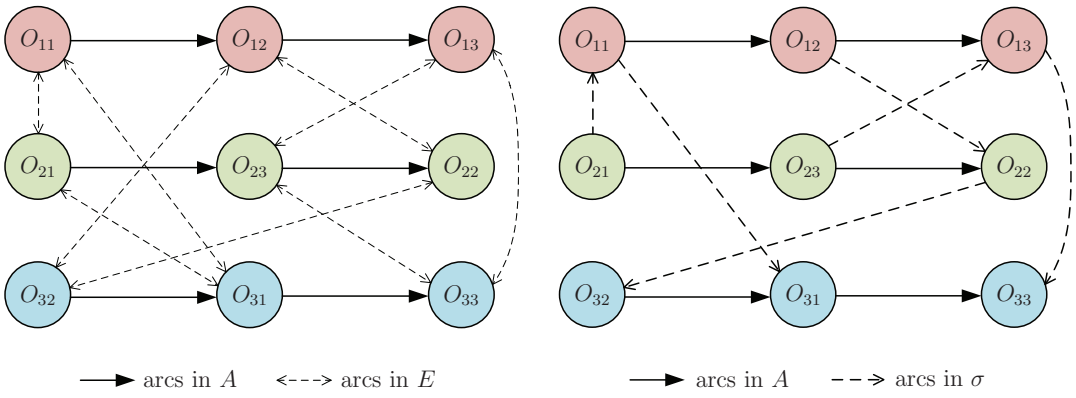
Each job has a fixed processing route which traverses all the machines in a predetermined order. The manufacturing process of job j on machine k is noted as operation O_{jk}. Besides, a preset due date d_j (describing the level of urgency) is given for each job j. The duration of an operation is influenced by many real-time factors such as the condition of workers and machines. Therefore, in the scheduling stage, the processing time of each operation is usually not known exactly. We assume that the processing times (p_{jk}, for each O_{jk}) are independent random variables with known expectation ($\mathrm{E}(p_{jk})$) and variance ($\mathrm{var}(p_{jk})$). The objective function is the expected maximum lateness ($L_{\max}$). If we use C_j to denote the completion time of job j, then $L_{\max}$ is defined as $\max_{j=1}^{n}\{C_j - d_j\}$. We consider $L_{\max}$ because due date related performances are becoming very significant in the make-to-order manufacturing environment nowadays.

Like its deterministic counterpart, SJSSP can also be described by a disjunctive graph $G(O, A, E)$ [31]. $O = \{O_{jk} | j = 1, \ldots, n,\ k = 1, \ldots, m\}$ 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 SJSSP 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, unlike the conjunctive arcs in A.

Under the disjunctive graph representation, finding a feasible schedule for the SJSSP 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 is transformed from the original E. Thus, if $A \cup \sigma$ is acyclic, the schedule corresponding to σ is feasible [32].

Example 1. Figure 1(**a**) shows the disjunctive graph for a 3×3 instance. The solid lines represent the conjunctive arcs while the dashed lines with bidirectional arrows express the disjunctive arcs.

Figure 1. A concrete example for the disjunctive graph representation of SJSSP.



(**a**) An example disjunctive graph (**b**) A feasible schedule

Figure 1(**b**) shows a feasible schedule σ for this problem. The dashed lines with fixed orientations represent the directed disjunctive arcs (the redundant arcs have been omitted). Clearly, there is no cycle in the graph. The schedule can be written in the matrix form as

$$\sigma = \begin{bmatrix} O_{21} & O_{11} & O_{31} \\ O_{12} & O_{22} & O_{32} \\ O_{23} & O_{13} & O_{33} \end{bmatrix}$$

where the k-th row specifies the resolved disjunctive arcs related with machine k ($k = 1, 2, 3$).

Based on the disjunctive graph model, the discussed SJSSP can be mathematically formulated as follows:

$$\begin{cases} \min \quad \mathrm{E}\left(L^{\sigma}_{\max}\right) = \mathrm{E}\left[\max\limits_{j=1}^{n}\left\{t_{jk_j} + p_{jk_j} - d_j\right\}\right] \\ \text{s.t.} \\ \qquad t_{jk} + p_{jk} \le t_{jk'} \ \text{ a.e.} \quad \forall\,(O_{jk}, O_{jk'}) \in A, \qquad \text{(a)} \\ \qquad t_{jk} + p_{jk} \le t_{j'k} \ \text{ a.e.} \quad \forall\,(O_{jk}, O_{j'k}) \in \sigma, \qquad \text{(b)} \\ \qquad t_{jk} \ge 0 \ \text{ a.e.} \quad j = 1, \ldots, n,\ k = 1, \ldots, m. \qquad \text{(c)} \end{cases}$$

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 $t_{jk_j} + p_{jk_j}$. Constraint (a) ensures that the processing order of the operations from each job is consistent with the technological routes. Constraint (b) ensures that the processing order of the operations on each machine complies with the sequence specified by the schedule, σ.

When a feasible schedule σ is given, a minimum $L^{\sigma}_{\max}$ can be achieved for each realization of $\{p_{jk}\}$. Since $\{p_{jk}\}$ are random variables, $\{t_{jk}\}$ and $L^{\sigma}_{\max}$ are also random variables. Therefore, the objective function is expressed in the form of expectation, and constraints (a–c) should hold almost everywhere (a.e.). The aim of solving SJSSP is to find a schedule σ with the minimum $\mathrm{E}\left(L^{\sigma}_{\max}\right)$.

3.2. Principles of the Artificial Bee Colony (ABC) Algorithm

In the ABC algorithm, the artificial bees are classified into three groups: the employed bees, the onlookers and the scouts. A bee that is exploiting a food source is classified as employed. The employed bees share information with the onlooker bees, which are waiting in the hive and watching the dances of the employed bees. The onlooker bees will then choose a food source with probability proportional to the quality of that food source. Therefore, good food sources attract more bees than the bad ones. Scout bees search for new food sources randomly in the vicinity of the hive. When a scout or onlooker bee finds a food source, it becomes employed. When a food source has been fully exploited, all the employed bees associated with it will abandon the position, and may become scouts again. Therefore, scout bees perform the job of "exploration", whereas employed and onlooker bees perform the job of "exploitation". In the algorithm, a food source corresponds to a possible solution to the optimization problem, and the nectar amount of a food source corresponds to the fitness of the associated solution.

In ABC, the first half of the colony consist of employed bees and the other half are onlookers. The number of employed bees is equal to the number of food sources (SN) because it is assumed there is only one employed bee for each food source. Thus, the number of onlooker bees is also equal to the number of solutions under consideration. The ABC algorithm starts with a group of randomly generated food sources. The main procedure of ABC can be described as follows.

Step 1: Initialize the food sources.
Step 2: Each employed bee starts to work on a food source.
Step 3: Each onlooker bee selects a food source according to the nectar information shared by the employed.
Step 4: Determine the scout bees, which will search for food sources in a random manner.
Step 5: Test whether the termination condition is met. If not, go back to Step 2.

The detailed description for each step is given below.

(1) *The initialization phase*. The SN initial solutions are randomly-generated D-dimensional real vectors. Let $\boldsymbol{x}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,D}\}$ represent the i-th food source, which is obtained by

$$x_{i,d} = x_d^{\min} + r \times (x_d^{\max} - x_d^{\min}),\ d = 1, \ldots, D \tag{1}$$

where r is a uniform random number in the range $[0, 1]$, and $x_d^{\min}$ and $x_d^{\max}$ are the lower and upper bounds for dimension d, respectively.

(2) *The employed bee phase*. In this stage, each employed bee is associated with a solution. She exerts a random modification on the solution (original food source) for finding a new solution (new food source). This implements the function of neighborhood search. The new solution $\boldsymbol{v}_i$ is generated from $\boldsymbol{x}_i$ using a differential expression:

$$v_{i,d} = x_{i,d} + r' \times (x_{i,d} - x_{k,d}) \tag{2}$$

where d is randomly selected from $\{1, \ldots, D\}$, k is randomly selected from $\{1, \ldots, SN\}$ such that $k \neq i$, and r' is a uniform random number in the range $[-1, 1]$.

Once $\boldsymbol{v}_i$ is obtained, it will be evaluated and compared to $\boldsymbol{x}_i$. If the fitness of $\boldsymbol{v}_i$ is better than that of $\boldsymbol{x}_i$ (*i.e.*, the nectar amount of the new food source is higher than the old one), the bee will forget the old solution and memorize the new one. Otherwise, she will keep working on $\boldsymbol{x}_i$.

(3) *The onlooker bee phase*. When all employed bees have finished their local search, they share the nectar information of their food source with the onlookers, each of whom will then select a food source in a probabilistic manner. The probability p_i by which an onlooker bee chooses food source $\boldsymbol{x}_i$ is calculated as follows:

$$p_i = \frac{f_i}{\sum_{i=1}^{SN} f_i}$$

where f_i is the fitness value of $\boldsymbol{x}_i$. Obviously, the onlooker bees tend to choose the food sources with higher nectar amount.

Once the onlooker has selected a food source $\boldsymbol{x}_i$, she will also conduct a local search on $\boldsymbol{x}_i$ according to Equation (2). As in the previous case, if the modified solution has a better fitness, the new solution will replace $\boldsymbol{x}_i$.

(4) *The scout bee phase*. In ABC, if the quality of a solution cannot be improved after a predetermined number ($limit$) of trials, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. The scout will then produce a food source randomly by using Equation (1).

To facilitate the understanding of the algorithm, a flow chart [33] is provided as Figure 2.

4. An Estimate for the Expected Maximum Lateness

In the studied SJSSP, the objective function is expressed as an expectation because the processing times are random variables. Meanwhile, due to the $\mathcal{NP}$-hardness of JSSP, there does not exist a closed-form expression to calculate the expected objective value. Therefore, the evaluation of a given schedule is not a simple task. Usually, we can utilize the idea of Monte Carlo simulation, *i.e.*, taking the average objective value in a large number of realizations as an estimate for the expectation. However, this definitely increases the computational burden, especially when used in an optimization framework (frequent solution evaluations are needed).

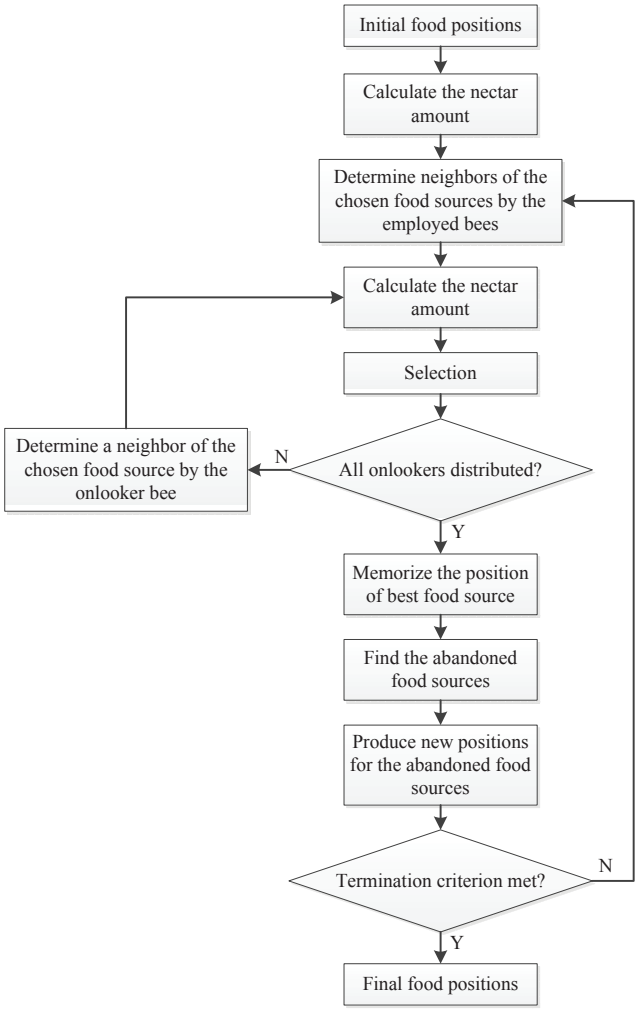
If there is no strict requirement on the evaluation accuracy, it is natural to think of a time-saving strategy: cut down the number of realizations. Furthermore, if we allow only one realization of the random processing times, then a rational choice is to use the mean (expected) value of each processing time. Then, we can show the following property, that is, such an estimate is a lower bound of the true objective value.

Theorem 1. *Let σ denote a feasible schedule of the considered SJSSP instance. The following inequality must hold:*

$$\mathrm{E}\left(L_{\max}^{\sigma}\right) \geq \overline{L}_{\max}^{\sigma}$$

where $L_{\max}^{\sigma}$ (random variable) is the maximum lateness corresponding to the schedule, and $\overline{L}_{\max}^{\sigma}$ (constant value) is the maximum lateness in the case where each random processing time takes the value of its expectation.

Figure 2. Flow chart of the ABC algorithm.



Proof. For the sake of simplicity, we will omit the superscript "σ" in the following proof because we are already focusing on the given schedule.

We will denote the completion time of operation O_{jk} by c_{jk} (a random variable). Furthermore, let $\overline{t}_{jk}$ (resp. $\overline{c}_{jk}$) denote the starting time (resp. completion time) of operation O_{jk} when each processing time is replaced by its expected value. Since $A \cup \sigma$ is acyclic, all the operations can be sorted topologically, yielding an ordered sequence $\{O_{jk}^{[i]}\}_{i=1}^{n \times m}$. In this sequence, the machine predecessors and the job predecessors of any operation O_{jk} must be positioned before O_{jk}. First we will prove that, for any operation O_{jk}, $\mathrm{E}(t_{jk}) \geq \overline{t}_{jk}$.

We start from the first operation ($i = 1$) in the sequence. Because $O_{jk}^{[1]}$ has no predecessor operations, we have $\mathrm{E}(t_{jk}) = \overline{t}_{jk}$ (actually $t_{jk} = 0$). Then, the proof procedure will continue for $i = 2, \ldots, n \times m$. Suppose we have already proved $\mathrm{E}(t_{jk}) \geq \overline{t}_{jk}$ for each operation before $O_{jk}^{[i]}$ in the sequence, and without

loss of generality, we assume that $O_{jk}^{[i]}$ has an immediate machine predecessor $O_{j'k}$ and an immediate job predecessor $O_{jk'}$, then we have

$$\begin{aligned}
\mathrm{E}\left(t_{jk}\right) &= \mathrm{E}\left(\max\left\{c_{j'k}, c_{jk'}\right\}\right) \\
&\geq \max\left\{\mathrm{E}\left(c_{j'k}\right), \mathrm{E}\left(c_{jk'}\right)\right\} \\
&= \max\left\{\mathrm{E}\left(t_{j'k} + p_{j'k}\right), \mathrm{E}\left(t_{jk'} + p_{jk'}\right)\right\} \\
&\geq \max\left\{\bar{t}_{j'k} + \mathrm{E}\left(p_{j'k}\right), \bar{t}_{jk'} + \mathrm{E}\left(p_{jk'}\right)\right\} \\
&= \max\left\{\bar{c}_{j'k}, \bar{c}_{jk'}\right\} \\
&= \bar{t}_{jk}
\end{aligned}$$

where $\mathrm{E}\left(t_{j'k}\right) \geq \bar{t}_{j'k}$ and $\mathrm{E}\left(t_{jk'}\right) \geq \bar{t}_{jk'}$ hold because $O_{j'k}$ and $O_{jk'}$ must come before O_{jk} in the sequence. If O_{jk} has no job predecessor, then we set $c_{jk'} = 0$ in the above proof. Likewise, if O_{jk} has no machine predecessor, then we set $c_{j'k} = 0$. Therefore, the reasoning applies to each operation in the sequence.

Having proved $\mathrm{E}\left(t_{jk}\right) \geq \bar{t}_{jk}$, we can now move to the objective function ($L_{\max}$). Let $\overline{C}_j$ (resp. $\overline{L}_j$) denote the completion time (resp. lateness) of job j when each random processing time takes its expected value. Meanwhile, k_j represents the machine which processes the last operation of job j. Then,

$$\begin{aligned}
\mathrm{E}\Big(\max_{j=1}^{n}\{L_j\}\Big) &\geq \max_{j=1}^{n}\left\{\mathrm{E}\left(L_j\right)\right\} \\
&= \max_{j=1}^{n}\left\{\mathrm{E}\left(t_{jk_j} + p_{jk_j}\right) - d_j\right\} \\
&\geq \max_{j=1}^{n}\left\{\bar{t}_{jk_j} + \mathrm{E}\left(p_{jk_j}\right) - d_j\right\} \\
&= \max_{j=1}^{n}\left\{\overline{C}_j - d_j\right\} \\
&= \max_{j=1}^{n}\{\overline{L}_j\}
\end{aligned}$$

This completes the proof of $\mathrm{E}\left(L_{\max}^{\sigma}\right) \geq \overline{L}_{\max}^{\sigma}$. □

5. The Proposed ABC Algorithm for Solving SJSSP

In this section, the proposed ABC is introduced in detail. First, we describe how ABC can be adapted to a discrete optimization problem like SJSSP. Second, we describe the method for comparing the quality (subject to estimation errors) of different solutions in the proposed ABC. Since the problem is stochastic, in the solution comparison process, we must decide how to allocate the available simulation replications. This problem is resolved by the model proposed in the third subsection. Finally, to facilitate the utilization of the simulation allocation model, we slightly revise the implementation of the local search procedure in standard ABC.

5.1. Adaptation to the Discrete Problem

Since the standard ABC is intended for continuous function optimization, we need to introduce some modifications in order to adapt it to SJSSP.

5.1.1. Encoding and Decoding

First, the encoding scheme for SJSSP is based on operation sequences instead of real number vectors. In particular, each solution (food source) is represented by a sequence of $n \times m$ operations, which can be transformed into a feasible schedule by the "schedule builder" presented below. This procedure functions by iteratively scanning the sequence and then picking out the first schedulable operation [34] in the sequence. In order to build an active schedule, each operation is inserted into the earliest possible idle period (*i.e.*, time interval not occupied by the previously scheduled operations).

Input: A sequence of operations, $\boldsymbol{\pi}$, with a length of $n \times m$.

Step 1: Let σ be an empty matrix of size $m \times n$.

Step 2: If $\boldsymbol{\pi} = \emptyset$, output σ (and the corresponding $L_{\max}$ if necessary) and terminate the procedure. Otherwise, continue the following steps.

Step 3: Find the first schedulable operation in the sequence $\boldsymbol{\pi}$, denoted by O^*.

Step 4: Identify the machine required to process O^* and denote it by k^*. Record the expected processing time of O^* as p^*.

Step 5: Schedule the operation O^*:

(5.1) Scan the Gantt chart of machine k^* (which records the processing information of the already scheduled operations) from time zero and test whether O^* can be inserted into each idle period $[a, b]$, *i.e.*, whether the following condition is met: $\max\{a, C_{JP^*}\} + p^* \leq b$ (where C_{JP^*} denotes the completion time of the immediate job predecessor of operation O^*).

(5.2) If the above inequality is satisfied for the idle interval between operation o_1 and o_2 on machine k^*, then insert O^* between o_1 and o_2 in the k^*-th row of σ. Otherwise (no idle intervals can hold O^*), insert it at the back of the k^*-th row of σ. Update the Gantt chart records for the starting time and completion time of operation O^*.

Step 6: Delete operation O^* from $\boldsymbol{\pi}$. Go back to Step 2.

5.1.2. Initialization

Second, the solution initialization policy (Equation (1) in standard ABC) should be modified to comply with the solution encoding scheme. Here we use the famous ATC (apparent tardiness cost) rule [35] to produce the initial population. To generate the u-th solution, we first make a sample of the random processing times according to their distributions, denoted as $\{p_{jk}^{(u)}\}$. Then, the ATC rule is applied in a simulation-based scheduling procedure, where the operation with the largest Z_{jk} value will be selected for processing when a machine is freed at time t.

$$Z_{jk}(t) = \frac{w_j}{p_{jk}^{(u)}} \cdot \exp\left\{ -\frac{\left[d_j - t - p_{jk}^{(u)} - \sum_{O_{jk'} \in JS(O_{jk})} \left(\widehat{W}_{jk'} + p_{jk'}^{(u)} \right) \right]^+}{K \cdot \overline{p}^{(u)}} \right\}$$

where $JS(O_{jk})$ is the set of job successors of operation O_{jk}. $\overline{p}^{(u)}$ denotes the average processing time of the currently waiting operations in the machine's buffer. K is a scaling parameter (or called

"look-ahead" parameter). $\widehat{W}_{jk'}$ is the estimated lead time of operation $O_{jk'}$. Here it is assumed $K = 2$ and $\widehat{W}_{jk'} = 0.4p^{(u)}_{jk'}$.

5.1.3. Neighborhood Structure

Third, the neighborhood search mechanism (Equation (2) in standard ABC) also needs to be modified accordingly. Let's recall that, in the case of deterministic JSSP, we must focus on the operations that belong to the blocks [36] of tardy jobs in order to improve the current solution. Therefore, the neighborhood search for SJSSP can be performed by using the expected values of processing times to determine the critical paths and blocks. Particularly, we first select two adjacent operations randomly from a block related with a tardy job, and then the SWAP operator is applied to exchange the positions of these two operations in the encoded solution.

5.2. The Comparison of Solutions

In order to approximate $\mathrm{E}\left(L^{\sigma}_{\max}\right)$, we need to implement the schedule σ under different realizations of the random processing times. When σ has been evaluated for a sufficient number of times, say τ, then its objective value can be approximated by $\frac{1}{\tau}\sum_{i=1}^{\tau} L^{\sigma}_{\max}(i)$ (where $L^{\sigma}_{\max}(i)$ corresponds to the i-th realization). This is consistent with the idea of Monte Carlo simulation.

In the employed bee phase and the onlooker bee phase, the newly-generated solution must be compared with the original solution in order to determine which one should be kept. For deterministic optimization problems, this can be done by simply comparing the exact objective values of the two solutions. But in the stochastic case, the comparisons may not be so straightforward because we can only obtain approximated (noisy) objective values as mentioned above. In this study, we will utilize the following two mechanisms for comparison purposes.

(I) *Pre-screening*. Because $\overline{L}^{\sigma}_{\max}$ is a lower bound for $\mathrm{E}\left(L^{\sigma}_{\max}\right)$ (Theorem 1), we can arrive at the following conclusion which is useful for the pre-screening of candidate solutions.

Corollary 1. *For two candidate solutions $\boldsymbol{x}_1$ (the equivalent schedule is denoted by σ_1) and $\boldsymbol{x}_2$ (the equivalent schedule is denoted by σ_2), if $\overline{L}^{\sigma_2}_{\max} \geq \mathrm{E}\left(L^{\sigma_1}_{\max}\right)$, then $\boldsymbol{x}_2$ must be inferior to $\boldsymbol{x}_1$ and thus need not be considered.*

(II) *Hypothesis test*. If the candidate solution has passed the pre-screening, then hypothesis test is used for comparing the quality of the two solutions.

Suppose we have implemented n_i simulation replications for solution $\boldsymbol{x}_i$ whose true objective value is $f\left(\boldsymbol{x}_i\right) = \mathrm{E}\left(L^{\sigma_i}_{\max}\right)$ $(i = 1, 2)$. Then, the sample mean and sample variance can be calculated by

$$\overline{f}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} f_i^{(j)}$$

$$s_i^2 = \frac{1}{n_i - 1}\sum_{j=1}^{n_i}\left(f_i^{(j)} - \overline{f}_i\right)^2$$

where $f_i^{(j)}$ is the objective value obtained in the j-th simulation replication for solution $\boldsymbol{x}_i$.

Let the null hypothesis H_0 be "$f(\boldsymbol{x}_1) = f(\boldsymbol{x}_2)$", and thus the alternative hypothesis H_1 is "$f(\boldsymbol{x}_1) \neq f(\boldsymbol{x}_2)$". According to the statistical theory, the critical region of H_0 is

$$\left|\overline{f}_1 - \overline{f}_2\right| \geq Z = z_{\alpha/2}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

where $z_{\alpha/2}$ is the value such that the area to its right under the standard normal curve is exactly $\alpha/2$. Therefore, if $\left|\overline{f}_1 - \overline{f}_2\right| < Z$ (*i.e.*, the null hypothesis holds), it is said that there exists no statistical difference between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Otherwise, if $\overline{f}_1 - \overline{f}_2 \geq Z$, $\boldsymbol{x}_2$ is statistically better than $\boldsymbol{x}_1$; if $\overline{f}_1 - \overline{f}_2 \leq -Z$, $\boldsymbol{x}_1$ is statistically better than $\boldsymbol{x}_2$.

5.3. The Allocation of Simulation Replications

In the previous subsection, we assume that n_i simulation replications have been performed for solution $\boldsymbol{x}_i$ before the hypothesis test. But how should the value of n_i be determined? We know that full-length simulation is very time-consuming, so a smaller value is desirable for n_i. On the other hand, however, if n_i is too small, the approximation for the objective values will be over-imprecise, which can mislead the hypothesis test. To achieve a balance, here we borrow an idea from the solution to the famous K-armed bandit problem.

The K-armed bandit problem can be described as K random variables, X_i ($1 \leq i \leq K$), where X_i represents the stochastic reward given by the i-th gambling machine. The distributions of X_i are independent but generally not identical. The laws of the distributions and the expectations μ_i for the rewards X_i are unknown. The goal is to find a strategy that determines the next machine to play (based on the past plays and the received rewards) that maximizes the expected total reward for the player.

Since a player does not know which of the machines is the best, he can only guess the reward distributions from successive plays. Clearly, he has to make a trade-off between the following two choices:

- The player tends to concentrate his efforts on the machines that gave the highest rewards in the past plays. Intuitively, this choice will maximize the potential gains.
- The player also wants to try the machines which he has played very few times in the past. The reward distributions of these machines are quite unclear, but it is likely that they provide even higher rewards.

Therefore, it is necessary to strike a wise balance between exploiting the currently best machine and exploring the other machines to make sure that none of those is even better.

When allocating the simulation replications, we are actually facing a similar situation. There are a number of solutions waiting to be evaluated. For each solution, the objective value obtained from one simulation is a random variable following an unknown distribution. We want to find out the quality of these solutions with the minimum total number of simulation replications. The question is to decide which solution should get the next chance of simulation. Obviously, the solutions are analogous to the gambling machines, and the simulations are analogous to the gambler's tries.

Here we use an algorithm called UCB1 [38] for making the allocation decisions. If the number of candidate solutions is K, while the available computing resource is only enough to support a total of T

replications of simulation ($T > \delta K$, where δ is the number of replications assigned at a time), then the algorithm can be adapted to our problem as follows.

Input: A set of K candidate solutions.

Step 1: Perform δ simulation replications for each solution $\boldsymbol{x}_k$. Denote the mean objective value by $\overline{f}_k$, and denote the standard deviation by s_k. Calculate the estimated "reward" as $\tilde{r}_k = s_k/\overline{f}_k$. Set $\nu_k = \delta$ ($k = 1, \ldots, K$) and $\nu = \delta K$.

Step 2: Calculate a priority index for each solution $\boldsymbol{x}_k$: $\rho_k = \tilde{r}_k + \sqrt{\frac{2 \ln \nu}{\nu_k}}$.

Step 3: Perform δ additional simulation replications for the solution $\boldsymbol{x}_{k^*}$ with the maximum ρ value. Update $\overline{f}_{k^*}$, s_{k^*} and $\tilde{r}_{k^*}$ for this solution. Let $\nu_{k^*} \leftarrow \nu_{k^*} + \delta$ and $\nu \leftarrow \nu + \delta$.

Step 4: If $\nu < T$, go back to Step 2. Otherwise, terminate the procedure.

In the adapted UCB1 (noted as A-UCB1) algorithm, ν_k records the number of times that solution $\boldsymbol{x}_k$ has been evaluated through simulation, while ν is the total number of simulation replications that have been assigned so far. The "reward" from trying $\boldsymbol{x}_k$ is defined as the relative standard deviation of the simulated objective values. Under the effect of ρ, the definition of r_k ensures that the simulation replications tend to be allocated to the solutions whose quality is still unclear (represented by the relatively large s_k). Meanwhile, the algorithm will also allocate simulation replications to the solutions which have been evaluated very few times (represented by the small ν_k). In sum, the A-UCB1 algorithm aims at promoting the computational efficiency in the evaluation of solutions. Based on extensive computational tests, the value of δ is set as $\lfloor T/100 \rfloor$.

5.4. Revised Implementation of the Local Search in ABC

In the employed bee phase and the onlooker bee phase, the artificial bees are actually performing the local search task. Considering the special features of the stochastic optimization problem, we slightly modify the implementation of this local search mechanism. In particular, we evaluate and compare the solutions group by group rather than one by one. The aim is to utilize the benefits of the A-UCB1 algorithm (for controlling the computational burden) and the hypothesis test (for increasing the diversity of the whole population). Our implementation is detailed as follows. Suppose the bees are already associated with each solution.

Step 1: Use A-UCB1 to allocate a total of T simulation replications to the solutions in the current population P. Save the estimated mean objective value and variance for each solution.

Step 2: Generate a new solution based on each original solution using the SWAP operator. Apply the pre-screening method (Corollary 1) to ensure the quality of each new solution [39]. Denote the set of new solutions by P_{new}.

Step 3: Use A-UCB1 to allocate a total of T simulation replications to the solutions in P_{new}. Save the estimated mean objective value and variance for each solution.

Step 4: Perform hypothesis tests to determine the new population:

(4.1) Sort all the solutions in $P \cup P_{\text{new}}$ in non-decreasing order of the estimated objective values, yielding a sequence $\{\boldsymbol{x}_{[1]}, \ldots, \boldsymbol{x}_{[2SN]}\}$. Put $\boldsymbol{x}_{[1]}$ into the ultimate population P^*. Let $i = 1$, $j = 2$.

(4.2) Perform hypothesis test for $\boldsymbol{x}_{[j]}$ and the i-th solution (the latest) in P^*. If the null hypothesis holds, $\boldsymbol{x}_{[j]}$ is rejected. Otherwise, $\boldsymbol{x}_{[j]}$ is appended to P^* and let $i \leftarrow i+1$.

(4.3) If $i < SN$ and $j < 2SN$, then let $j \leftarrow j+1$ and go to Step 4.2. Otherwise, go to Step 4.4.

(4.4) If $i = SN$, the ultimate population has been determined. Otherwise, generate $(SN - i)$ new solutions randomly, evaluate them (each one is evaluated with $\lfloor T/SN \rfloor$ simulation replications) and append them to P^*.

Step 5: Compare P^* with P and check whether each new solution has been accepted (*i.e.*, in P^*). If accepted, let the corresponding bee fly to the new solution.

In the proposed ABC algorithm, the above procedure is used in place of the one-to-one greedy selection policy in standard ABC. This is in consideration of the specific features of stochastic optimization. Evaluating a group of solutions together is beneficial, because A-UCB1 will allocate more simulation replications to good solutions (with smaller $\overline{f}_k$) to make sure about their true performance and avoid wasting time on inferior solutions (with larger $\overline{f}_k$). The hypothesis test is used for maintaining an adequate level of diversity of the solution population. In Step 4.2, if $\boldsymbol{x}_{[j]}$ does not significantly differ from the most recent solution in P^*, it will not have a chance to enter P^*. Finally, if the number of qualified solutions is less than SN, the rest of solutions will be generated randomly.

6. The Computational Experiments

6.1. Generation of Test Instances

To test the effectiveness of the proposed algorithm, computational experiments are conducted on a number of randomly-generated test instances. In each instance, the route of each job is a random permutation of m machines. Three types of distribution patterns are considered for the random processing times: normal distribution, uniform distribution and exponential distribution. In all cases, the mean values (μ_{jk}) are generated from the uniform distribution $\mathcal{U}(1, 99)$. In the case of normal distributions (*i.e.*, $p_{jk} \sim \mathcal{N}(\mu_{jk}, \sigma_{jk}^2)$), the standard deviation is controlled by $\sigma_{jk} = \theta \times \mu_{jk}$ (θ describes the level of variability). In the case of uniform distributions (*i.e.*, $p_{jk} \sim \mathcal{U}(\mu_{jk} - \omega_{jk}, \mu_{jk} + \omega_{jk})$), the width parameter is given by $\omega_{jk} = \theta \times \mu_{jk}$. In the case of exponential distributions (*i.e.*, $p_{jk} \sim \mathrm{Exp}(\lambda_{jk})$), the only parameter is given by $\lambda_{jk} = 1/\mu_{jk}$. The due dates are obtained by a series of simulation runs which apply different priority rules (such as SPT, EDD, *etc.*) on each machine, and the due date of each job is finally set as its average completion time. This method can generate reasonably tight due dates. Meanwhile, the weight of each job is generated from the uniform distribution $\mathcal{U}(1, 10)$. The following computational experiments are conducted in Visual C++ 2010 on an Intel Core i5-750/3GB RAM/Windows 7 PC.

6.2. The Computational Results and Comparisons

Based on extensive computational tests (not listed due to space limitations), the settings of key parameters in the final ABC algorithm are: $SN = 30$, $limit = 40$ and $T = 1000$ (for each evaluation). The confidence level for the hypothesis test is $\alpha = 0.05$.

In the following, we will use the proposed ABC to solve different-sized SJSSP instances. The results are compared with the hybrid optimization method PSO-SA [40], which utilizes SA (simulated annealing) as a local optimizer for PSO. In order to make the comparisons meaningful, we set a computational time limit for both algorithms. Normally, the time limit should be set according to the size of the instance. In our experiment, the time limit for solving a $n \times m$ instance is determined as $t = 0.2 \times (n \times m)$. For example, the allowed computational time for a 20×10 instance is 40 sec. Each algorithm is run for 20 independent times on each SJSSP instance.

We report the best, average and worst objective values (exact evaluation [41] for the output solutions) in the 20 runs. Tables 1–3 report the results under normal distributions, Tables 4–6 report the results under uniform distributions, and Table 7 reports the results under exponential distributions.

Table 1. The computational results under normal distributions with $\theta = 0.1$.

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	46.1	52.1	58.0	50.5	53.2	59.8
	2	47.4	49.1	53.2	46.7	53.1	54.8
	3	50.0	51.7	54.0	50.4	52.3	54.9
	4	56.2	61.8	62.7	60.8	62.5	66.2
	5	50.9	56.0	61.3	53.5	56.3	61.1
20×10	6	66.0	69.0	75.3	70.3	75.8	86.1
	7	63.0	67.6	71.6	67.0	72.1	83.8
	8	61.8	65.5	67.5	65.0	70.9	75.8
	9	64.9	70.0	74.9	69.4	76.7	78.2
	10	60.5	68.7	70.0	64.0	70.5	76.1
20×15	11	80.6	83.1	84.1	85.1	86.2	88.2
	12	80.9	84.8	89.2	86.1	87.8	90.9
	13	67.1	72.5	83.2	75.7	77.0	84.5
	14	77.4	81.8	85.6	79.1	83.7	89.3
	15	72.3	78.6	83.6	74.9	81.8	86.6
20×20	16	99.9	106.9	121.2	103.5	111.0	124.7
	17	97.4	103.2	118.2	114.9	128.0	134.9
	18	104.6	114.9	123.8	120.8	129.8	137.1
	19	111.4	117.5	120.3	120.7	123.9	131.6
	20	98.5	115.2	121.4	121.5	128.6	132.7

In addition, we have also performed Mann–Whitney U tests to statistically compare the computational results. Because each algorithm is run for 20 times, we have $n_1 = n_2 = 20$ in the Mann–Whitney U test. The null hypothesis is that there is no difference between the results of the two compared algorithms. Therefore, if the obtained U (the lesser of U_1 and U_2) is below 127, the null hypothesis can be rejected at the 5% level. Furthermore, if $U < 105$, the null can be rejected at the 1% level. The values of U are listed in Table 8.

Table 2. The computational results under normal distributions with $\theta = 0.2$.

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	51.1	62.7	67.8	60.0	63.3	68.5
	2	53.1	55.6	60.6	54.9	60.2	64.9
	3	54.3	57.5	60.8	58.0	60.1	66.1
	4	63.8	70.7	75.4	70.0	74.2	80.6
	5	59.0	62.7	69.0	63.5	66.5	69.5
20×10	6	67.1	72.6	76.1	73.3	81.1	93.0
	7	64.2	69.4	71.3	72.1	75.8	86.6
	8	64.6	66.9	69.8	64.2	73.1	78.9
	9	64.6	72.1	74.7	73.9	80.6	82.3
	10	61.1	68.4	71.2	67.2	73.9	82.2
20×15	11	84.0	87.6	90.1	84.6	90.6	94.1
	12	86.5	91.5	93.2	96.7	98.6	103.8
	13	74.2	80.3	90.8	81.0	87.3	96.7
	14	84.8	88.8	90.7	85.1	91.2	98.8
	15	79.4	83.9	90.9	85.3	89.2	94.4
20×20	16	111.3	117.5	133.5	115.4	122.7	137.3
	17	100.2	108.8	127.6	127.1	131.1	148.9
	18	110.9	118.1	126.9	127.3	138.4	146.8
	19	118.5	123.7	129.2	131.5	139.2	143.6
	20	101.7	122.2	130.0	128.7	136.7	140.8

Table 3. The computational results under normal distributions with $\theta = 0.3$.

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	53.4	67.4	69.1	60.6	67.5	73.3
	2	56.0	60.0	63.1	56.5	66.1	68.6
	3	60.2	61.9	65.1	62.5	63.5	69.6
	4	69.1	72.8	74.5	74.7	77.8	82.5
	5	60.9	67.7	71.7	66.0	70.1	71.1
20×10	6	74.3	77.3	81.4	81.8	86.8	100.5
	7	70.8	75.8	81.6	77.1	80.6	95.6
	8	69.3	70.4	74.5	70.9	79.2	88.1
	9	72.0	76.5	81.9	77.6	87.5	90.4
	10	66.5	77.9	78.9	71.8	80.0	85.5
20×15	11	91.2	92.6	96.0	87.4	95.2	99.8
	12	90.5	94.6	98.9	101.0	104.6	105.7
	13	76.1	83.2	95.0	87.1	89.8	98.6
	14	88.6	91.7	97.8	86.3	93.1	99.1
	15	79.7	88.6	92.0	85.9	92.8	99.4
20×20	16	116.5	120.3	131.1	115.3	124.4	140.1
	17	104.4	111.2	130.9	128.6	146.3	151.2
	18	114.8	121.7	134.5	140.7	151.3	157.3
	19	121.3	125.7	131.3	133.7	137.6	147.0
	20	105.8	126.5	133.8	139.1	145.7	152.6

Table 4. The computational results under uniform distributions with $\theta = 0.1$.

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	38.0	41.9	47.5	40.8	44.7	50.4
	2	38.9	41.0	44.4	39.1	44.4	46.7
	3	41.6	42.7	45.6	40.6	42.7	45.1
	4	46.2	51.7	53.2	48.7	52.1	54.5
	5	41.6	46.5	51.4	43.5	45.8	51.4
20×10	6	55.8	58.6	64.7	59.6	63.2	71.3
	7	51.4	58.1	59.6	54.4	59.8	68.9
	8	52.9	53.7	55.1	53.0	57.6	62.0
	9	54.7	58.8	60.1	59.1	61.8	64.2
	10	50.7	57.2	59.0	53.5	59.0	64.3
20×15	11	65.8	68.4	71.6	67.2	70.2	73.9
	12	67.0	70.3	75.3	69.1	73.0	76.8
	13	56.7	58.4	66.7	61.0	62.4	71.7
	14	62.3	67.7	70.6	64.1	68.9	75.8
	15	60.2	67.3	69.0	62.6	66.6	70.5
20×20	16	80.1	86.9	103.4	88.4	93.3	106.9
	17	82.8	88.1	100.8	97.1	104.0	115.6
	18	86.5	95.9	105.6	99.1	108.2	115.4
	19	89.3	96.9	98.7	101.8	106.2	108.3
	20	83.2	92.5	103.7	103.3	107.0	110.9

Table 5. The computational results under uniform distributions with $\theta = 0.2$.

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	42.5	54.6	59.4	51.0	54.0	60.1
	2	44.6	46.6	52.1	46.4	53.3	54.6
	3	47.8	49.7	52.3	51.0	52.2	58.0
	4	54.2	62.5	67.1	58.9	64.7	67.7
	5	51.8	53.0	58.5	56.3	58.5	60.7
20×10	6	58.4	64.3	67.3	64.7	71.4	82.2
	7	54.9	60.1	63.3	61.6	66.8	72.4
	8	54.8	57.6	59.7	53.9	64.2	65.9
	9	56.5	61.9	67.2	62.4	70.5	72.1
	10	53.0	61.2	62.9	56.6	63.6	69.9
20×15	11	71.5	75.8	77.9	74.5	79.9	81.5
	12	77.0	80.0	84.2	84.4	87.2	90.9
	13	65.6	70.5	79.9	69.1	73.8	82.8
	14	72.6	78.8	85.0	71.6	77.3	87.5
	15	67.8	70.6	79.2	73.5	77.8	81.2
20×20	16	95.6	98.5	113.0	101.0	105.7	114.8
	17	83.4	90.7	111.5	99.4	109.5	130.4
	18	93.9	98.9	108.0	112.2	121.1	125.9
	19	99.9	102.8	109.3	115.0	119.2	120.4
	20	86.9	102.0	111.4	108.4	117.3	120.6

Table 6. The computational results under uniform distributions with $\theta = 0.3$.

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	48.1	58.4	60.4	52.5	61.0	63.3
	2	51.2	52.2	54.5	48.7	58.2	61.3
	3	53.2	55.6	58.7	56.6	58.3	61.8
	4	60.9	63.0	65.6	66.7	67.9	74.2
	5	55.0	60.0	62.7	59.5	61.5	63.9
20×10	6	67.9	69.4	70.8	75.2	78.9	87.4
	7	61.1	68.9	73.5	68.8	71.7	84.8
	8	60.2	63.2	66.2	64.8	71.4	78.7
	9	63.5	66.1	73.1	68.0	78.1	79.3
	10	57.9	69.2	72.0	64.5	71.5	76.0
20×15	11	79.5	81.8	85.7	81.4	85.4	90.4
	12	83.2	85.5	90.2	90.7	92.3	95.6
	13	67.9	76.1	86.4	75.7	80.5	88.2
	14	78.6	80.3	87.0	79.5	85.3	89.9
	15	71.6	79.1	81.0	74.0	81.5	90.9
20×20	16	101.3	107.2	113.2	102.5	110.2	127.3
	17	93.0	100.0	116.8	111.6	126.5	139.0
	18	101.0	105.4	122.2	129.3	132.1	143.9
	19	108.9	110.3	114.8	122.4	125.6	133.0
	20	91.9	115.5	121.5	122.1	127.7	139.3

Table 7. The computational results under exponential distributions

Size $n \times m$	Instance No.	ABC			PSO-SA		
		best	average	worst	best	average	worst
10×10	1	85.6	109.6	114.3	100.8	109.4	118.2
	2	91.3	100.2	102.0	93.3	109.4	114.1
	3	96.1	98.3	108.3	103.1	104.3	111.5
	4	114.4	121.0	129.7	123.3	128.5	132.1
	5	101.1	111.4	119.9	106.2	115.7	117.7
20×10	6	123.7	126.8	131.9	135.2	145.8	161.8
	7	113.8	124.8	134.2	124.6	134.9	154.3
	8	113.3	116.2	123.0	116.2	130.2	143.2
	9	116.6	125.7	132.7	130.3	146.4	147.6
	10	107.9	126.1	131.4	117.6	131.3	141.9
20×15	11	141.5	149.4	156.9	146.0	157.0	160.2
	12	148.8	153.4	162.4	167.5	175.2	177.0
	13	122.8	139.3	159.5	142.9	146.8	158.3
	14	143.7	150.5	160.7	144.5	149.8	163.6
	15	129.3	146.8	150.0	138.2	149.7	164.9
20×20	16	193.8	199.8	215.2	192.4	202.4	229.1
	17	168.0	184.4	217.6	213.0	242.9	251.8
	18	185.7	196.2	215.7	226.8	242.4	251.9
	19	199.6	209.5	213.6	214.6	225.0	241.9
	20	170.2	209.2	221.0	228.5	240.0	249.9

Table 8. Mann–Whitney U tests on the computational results.

Instance No.	Normal			Uniform			Exponential
	$\theta = 0.1$	$\theta = 0.2$	$\theta = 0.3$	$\theta = 0.1$	$\theta = 0.2$	$\theta = 0.3$	
1	97	94	82	99	97	86	76
2	93	85	79	94	89	78	70
3	95	84	79	94	87	80	76
4	87	78	70	90	79	72	68
5	90	80	79	89	81	78	72
6	80	74	68	83	76	69	62
7	82	78	69	84	78	69	64
8	84	79	70	86	79	73	66
9	79	71	64	78	72	63	61
10	81	74	67	83	73	65	64
11	86	81	74	85	81	76	70
12	91	86	79	90	86	78	70
13	93	79	77	90	79	81	73
14	88	87	84	91	88	86	82
15	96	84	78	94	89	81	73
16	88	77	70	89	76	70	64
17	77	68	60	74	70	63	59
18	70	66	59	72	64	61	55
19	80	67	66	84	66	66	61
20	69	64	60	70	64	62	53

Based on the statistical tests, we can conclude that ABC is significantly more effective than the comparative method. In addition, the following comments can be made.

(1) According to the tables, the advantage of ABC over PSO-SA is greater when the variability level of processing times is higher (represented by larger θ or the case of exponential distribution). This can be attributed to the function of A-UCB1, which is responsible for the allocation of the limited computational resources. If θ is small, the objective value of a solution can be obtained with only a few replications. In this case, the A-UCB1 strategy is not significantly better than an equal allocation of the available replications (the case of PSO-SA). However, when the variability increases, the computational time becomes a relatively scarce resource. In order to correctly identify high-quality solutions, the limited replications should be allocated in an efficient manner rather than evenly. So in this case, the advantage of using A-UCB1 becomes evident.

(2) According to the tables, ABC outperforms PSO-SA to a greater extent when solving the larger-scale instances. If the solution space is huge, PSO must rely on the additional local search module (SA) to promote the searching efficiency. However, SA is not so efficient as the inherent local search mechanism (employed and onlooker bees) in ABC, especially under tight time budgets. From another perspective, ABC systematically combines the exploration and exploitation abilities, and thus it works in a coordinated fashion. By contrast, the hybrid algorithm uses PSO for exploration and SA for exploitation, but the two algorithms have different search

patterns. This may weaken the cooperation between PSO and SA in solving large-scale SJSSP. Therefore, ABC alone is more efficient than the hybrid algorithm.

7. Conclusions

In this paper, an artificial bee colony algorithm is proposed for solving the job shop scheduling problem with random processing times. The objective function is to minimize the expected maximum lateness. In view of the stochastic nature of the problem, two mechanisms are devised for the evaluation of solutions. First, a quick-and-dirty performance estimate is used to pre-screen the candidate solutions, so that the obviously inferior solutions can be eliminated at an early stage. Then, Monte Carlo simulation is applied to obtain a more accurate evaluation for the surviving solutions. In this process, a simulation budget allocation method is designed based on the K-armed bandit metaphor. This helps to utilize the limited computational time in an efficient manner. The computational results on a wide range of test instances reveal the superiority of the proposed approach.

The future research can be conducted from the following aspects:

(1) It is worthwhile to consider other types of randomness in job shops, for example, the uncertainty in the processing routes of certain jobs.
(2) It is worthwhile to investigate the method for discovering and utilizing problem-specific characteristics of SJSSP. This will make the ABC algorithm more pertinent to this problem.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61104176, 60874071. We would express our thanks to the two anonymous referees for their pertinent comments that help to improve the paper.

References and Notes

1. Lenstra, J.K.; Kan, A.H.G.R.; Brucker, P. Complexity of machine scheduling problems. *Ann. Discret. Math.* **1977**, *1*, 343–362.
2. Pezzella, F.; Morganti, G.; Ciaschetti, G. A genetic algorithm for the flexible job-shop scheduling problem. *Comput. Oper. Res.* **2008**, *35*, 3202–3212.
3. Chen, S.H.; Chen, M.C.; Chang, P.C.; Chen, Y.M. EA/G-GA for single machine scheduling problems with earliness/tardiness costs. *Entropy* **2011**, *13*, 1152–1169.
4. Nowicki, E.; Smutnicki, C. An advanced tabu search algorithm for the job shop problem. *J. Sched.* **2005**, *8*, 145–159.
5. Lei, D. Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. *Int. J. Adv. Manuf. Technol.* **2008**, *37*, 157–165.
6. Rossi, A.; Dini, G. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robot. Comput. Integr. Manuf.* **2007**, *23*, 503–516.
7. Li, J.; Pan, Q.; Xie, S.; Wang, S. A hybrid artificial bee colony algorithm for flexible job shop scheduling problems. *Int. J. Comput. Commun. Control* **2011**, *6*, 286–296.

8. Yoshitomi, Y. A genetic algorithm approach to solving stochastic job-shop scheduling problems. *Int. Trans. Oper. Res.* **2002**, *9*, 479–495.
9. Yoshitomi, Y.; Yamaguchi, R. A genetic algorithm and the Monte Carlo method for stochastic job-shop scheduling. *Int. Trans. Oper. Res.* **2003**, *10*, 577–596.
10. Golenko-Ginzburg, D.; Gonik, A. Optimal job-shop scheduling with random operations and cost objectives. *Int. J. Prod. Econ.* **2002**, *76*, 147–157.
11. Tavakkoli-Moghaddam, R.; Jolai, F.; Vaziri, F.; Ahmed, P.K.; Azaron, A. A hybrid method for solving stochastic job shop scheduling problems. *Appl. Math. Comput.* **2005**, *170*, 185–206.
12. Luh, P.; Chen, D.; Thakur, L. An effective approach for job-shop scheduling with uncertain processing requirements. *IEEE Trans. Robot. Autom.* **1999**, *15*, 328–339.
13. Lai, T.; Sotskov, Y.; Sotskova, N.; Werner, F. Mean flow time minimization with given bounds of processing times. *Eur. J. Oper. Res.* **2004**, *159*, 558–573.
14. Singer, M. Forecasting policies for scheduling a stochastic due date job shop. *Int. J. Prod. Res.* **2000**, *38*, 3623–3637.
15. Lei, D. Scheduling stochastic job shop subject to random breakdown to minimize makespan. *Int. J. Adv. Manuf. Technol.* **2011**, *55*, 1183–1192.
16. Lei, D. Simplified multi-objective genetic algorithms for stochastic job shop scheduling. *Appl. Soft Comput.* **2011**, doi:10.1016/j.asoc.2011.06.001.
17. Gu, J.; Gu, X.; Gu, M. A novel parallel quantum genetic algorithm for stochastic job shop scheduling. *J. Math. Anal. Appl.* **2009**, *355*, 63–81.
18. Gu, J.; Gu, M.; Cao, C.; Gu, X. A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem. *Comput. Oper. Res.* **2010**, *37*, 927–937.
19. Mahdavi, I.; Shirazi, B.; Solimanpur, M. Development of a simulation-based decision support system for controlling stochastic flexible job shop manufacturing systems. *Simul. Model. Pract. Theory* **2010**, *18*, 768–786.
20. Azadeh, A.; Negahban, A.; Moghaddam, M. A hybrid computer simulation-artificial neural network algorithm for optimisation of dispatching rule selection in stochastic job shop scheduling problems. *Int. J. Prod. Res.* **2011**, doi:10.1080/00207543.2010.539281.
21. Bianchi, L.; Dorigo, M.; Gambardella, L.; Gutjahr, W. A survey on metaheuristics for stochastic combinatorial optimization. *Nat. Comput.* **2009**, *8*, 239–287.
22. Karaboga, D.; Akay, B. A survey: algorithms simulating bee swarm intelligence. *Artif. Intell. Rev.* **2009**, *31*, 61–85.
23. Karaboga, D. *An Idea Based on Honey Bee Swarm for Numerical Optimization*. Technical Report for Computer Engineering Department, Erciyes University, Kayseri, Turkey, October 2005.
24. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697.
25. Karaboga, D.; Akay, B. A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.* **2009**, *214*, 108–132.
26. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471.

27. Kang, F.; Li, J.; Xu, Q. Structural inverse analysis by hybrid simplex artificial bee colony algorithms. *Comput. Struct.* **2009**, *87*, 861–870.
28. Sonmez, M. Discrete optimum design of truss structures using artificial bee colony algorithm. *Struct. Multidiscip. Optim.* **2011**, *43*, 85–97.
29. Samanta, S.; Chakraborty, S. Parametric optimization of some non-traditional machining processes using artificial bee colony algorithm. *Eng. Appl. Artif. Intell.* **2011**, *24*, 946–957.
30. Huang, Y.; Lin, J. A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems. *Expert Syst. Appl.* **2011**, *38*, 5438–5447.
31. Jain, A.S.; Meeran, S. Deterministic job-shop scheduling: past, present and future. *Eur. J. Oper. Res.* **1999**, *113*, 390–434.
32. 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 k-th row of σ represents the processing order of the operations on machine k.
33. Akay, B.; Karaboga, D. Artificial bee colony algorithm for large-scale problems and engineering design optimization. *J. Intell. Manuf.* **2011**, doi:10.1007/s10845-010-0393-4.
34. An operation is schedulable if its immediate job predecessor has already been scheduled or if it is the first operation of a certain job.
35. Vepsalainen, A.P.; Morton, T.E. Priority rules for job shops with weighted tardy costs. *Manag. Sci.* **1987**, *33*, 1035–1047.
36. A sequence of operations in the critical path is called a block if (1) it contains at least two operations and (2) the sequence includes a maximum number of operations that are consecutively processed by the same machine [37].
37. Pinedo, M. *Scheduling: Theory, Algorithms and Systems*, 3rd ed.; Springer: New York, NY, USA, 2008.
38. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* **2002**, *47*, 235–256.
39. If a newly-generated solution does not pass the pre-screening test, then simply generate another solution from the neighborhood, and so on.
40. Liu, B.; Wang, L.; Jin, Y.H. Hybrid Particle Swarm Optimization for Flow Shop Scheduling with Stochastic Processing Time. In *Lecture Notes in Computer Science*; Springer: Berlin, Heidelberg, Germany, 2005; Volume 3801, pp. 630–637.
41. The mean value resulted from 1000 simulation replications (which is large enough for the considered test instances) is regarded as the exact evaluation of a solution.