Algorithms **2015**, *8*, 19-31; doi:10.3390/a8010019

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

An Improved Shuffled Frog-Leaping Algorithm for Flexible Job Shop Scheduling Problem

Kong Lu [1], Li Ting [1], Wang Keming [1], Zhu Hanbing [1], Takano Makoto [2] and Yu Bin [3,*]

1 Transportation Management College, Dalian Maritime University, Dalian 116026, China; E-Mails: kongluhaishi@163.com (K.L.); liting_dl@163.com (L.T.); wang_ke_ming@163.com (W.K.)

2 Department of transportation Engineering, Indian Institute of Technology, Kharagpur 721302, India; E-Mail: Takmakoto@outlook.com

3 Traffic and transportation college, Beijing Jiaotong University, Beijing 100044, China

* Author to whom correspondence should be addressed; E-Mail: yubin@dlmu.edu.cn; Tel.: +86-35-0072-9947.

Academic Editor: Chen-Chung Liu

Received: 24 October 2014 / Accepted: 15 January 2015 / Published: 4 February 2015

Abstract: The flexible job shop scheduling problem is a well-known combinatorial optimization problem. This paper proposes an improved shuffled frog-leaping algorithm to solve the flexible job shop scheduling problem. The algorithm possesses an adjustment sequence to design the strategy of local searching and an extremal optimization in information exchange. The computational result shows that the proposed algorithm has a powerful search capability in solving the flexible job shop scheduling problem compared with other heuristic algorithms, such as the genetic algorithm, tabu search and ant colony optimization. Moreover, the results also show that the improved strategies could improve the performance of the algorithm effectively.

Keywords: shuffled frog-leaping algorithm; flexible job shop scheduling problem; local search; extremal optimization

1. Introduction

The scheduling of operations has a vital role in the planning and managing of manufacturing processes. The job-shop scheduling problem (JSP) is one of the most popular scheduling models in practice. In the JSP, a set of jobs should be processed with a set of machines, and each job consists of a sequence of consecutive operations. Moreover, a machine can only process one operation at one time, and the operation cannot be interrupted. Additionally, JSP is aimed at minimizing the number of operations for these jobs under the above constraints.

The flexible job-shop scheduling problem (FJSP) is an extension of the conventional JSP, in which operations are allowed to be processed on any one of the existed machines. FJSP is more complicated than JSP, because FJSP not only needs to identify the arrangement of all processes of all machines, but also needs to determine the sequence of processes on each machine. FJSP breaks through the uniqueness restriction of resources. Each process can be completed by many different machines, so that the job shop scheduling problem is closer to the real production process.

JSP has been proven to be an NP-hard problem [1], and even for a simple instance with only ten operations and ten available machines for selection, it is still hard to search for a convincing result. As an extension of JSP, FJSP is more complicated to solve. A lot of the literature believes that adopting a heuristic method is a reasonable approach for solving this kind of complex problem [2–9]. Thus, there are many researchers who have used heuristic methods to solve FJSP. Brandimarte [10] attempted to use a hierarchical approach to solve the flexible job shop scheduling problem, and a tabu search is adopted to enhance the effectiveness of the approach. Fattahi *et al.* [11] proposed a mathematical model for the flexible job shop scheduling problem, and two heuristic approaches (tabu search and simulated annealing heuristics) are also introduced to solve the real size problems. Gao *et al.* [12] developed a hybrid genetic algorithm to solve the flexible job shop scheduling problem. In the algorithm, the two vectors are used to represent the solution, the individuals of GA are improved by a variable neighborhood descent and the advanced genetic operators are presented by a special chromosome structurer. Yao *et al.* [2] presented an improved ant colony optimization to solve FJSP, and an adaptive parameter, crossover operation and pheromone updating strategy are used to improve the performance of the algorithm.

The shuffled frog-leaping algorithm (SFLA) was developed by Eusuff and Lansey [13]. It is a meta-heuristic optimization method, which combines the advantages of the genetic-based memetic algorithm (MA) and the social behavior-based PSO algorithm [14]. A meme is a kind of information body, which can be distributed, reproduced and exchanged by infecting the thoughts of human beings or animals. The most salient characteristic of the meme algorithm is that memes can share and exchange experience, knowledge and information along with relying on a local search method in the process of evolution. Therefore, with the meme algorithm allows an individual of the traditional genetic algorithm model to become more intelligent. The group of SFLA consists of the frog group in which the individual frogs can communicate with each other. Each frog can be seen as a meme carrier. Along with the communication among frogs, the meme evolution can be performed during the searching process of the algorithm. Due to its efficiency and practical value, SFLA has attracted more attention and has been successfully used in a number of classical combinatorial optimization problems [15–19].

In 1999, a new general-purpose local search optimization approach, extremal optimization (EO), was proposed by Boettcher and Percus [20]. The strategy was evolved by the fundamentals of statistical physics and self-organized criticality [21]. The weakest species and its nearest neighbors in the population are always forced to mutate in the evolution process of this method. Thus, EO can effectively eliminate the worst components in the sub-optimal solutions and has been widely used to solve some optimization problems [22,23].

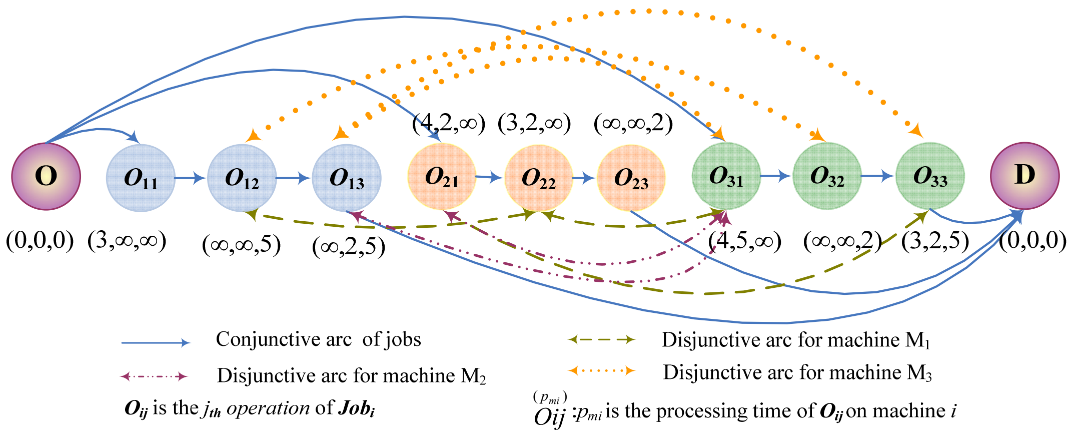
In general, the local exploration strategy that updates the worst solution in each memeplex is important for the performance of the algorithm; because the local exploration reveals that the leaping position of the worst frog is substantially influenced by the local/global best position and limited to the range between the current position and the best position. Thus, the local search strategy based on the adjustment order is adapted to enlarge the local search space. Moreover, EO is introduced to improve the local search ability of SFLA.

The remainder of the paper is organized as follows. A mathematical model for FJSP is presented in Section 2. The shuffled frog-leaping algorithm is introduced in Section 3. In Section 4, we present an improved shuffled frog-leaping algorithm with an extremal optimization and an adjustment strategy. Some computational results are discussed in Section 5, and the conclusions are provided in Section 6.

2. Problem Description

FJSP is a kind of resource allocation problem, *i.e.*, given M machines and the N work pieces, FJSP aims to minimize the operating time by optimizing the arrangement of the operations of all jobs t. A set of machines $M = \{M_1, M_2, \ldots, M_m\}$ and a set of independent jobs $J = \{J_1, J_2, \ldots, J_n\}$ are given. Additionally, each job J_i consists of a sequence of operations O_{ij}, where O_{ij} represents the j-th operation of the job i. All of the operations of each job should be performed one after another in a given order. Each operation O_{ij} can be processed on any machine among a subset $M_{i,j} \subseteq M$ of compatible machines. Based on the different conditions of resource restriction, the flexible job shop scheduling problem can be divided into a total flexible job-shop scheduling (T-FJSP) problem and a partial flexible job-shop scheduling (P-FJSP) problem. In T-FJSP, every process of each operation can choose any one machine to be processed; in P-FJSP, some processes of some work pieces can be processed on any one of the machines, and some certain processes can only choose parts of machines.

All jobs and machines are available at Time 0, and a machine can only execute one operation at a given time. The processing time of each operation is dependent of the machine, and p_{mi} represents the processing time of operation O_{ij} on machine i. If the operation O_{ij} cannot be processed on machine k, the processing time p_{mk} is assumed as ∞. Each operation must be completed without interruption during its process. The objective of FJSP is to assign each operation to one machine that is capable of performing it and to determine the starting time for each operation in order to minimize the makespan. In order to describe the FJSP well, Figure 1 is an example of FJSP with three jobs and five machines. In Figure 1, (0, 0, 0) means spending zero seconds on the first machine, zero seconds on the second one and zero seconds on the third one.



O
O_{11}
O_{12}
O_{13}
O_{21}
O_{22}
O_{23}
O_{31}
O_{32}
O_{33}
D
(0,0,0)
(3,∞,∞)
(∞,∞,5)
(∞,2,5)
(4,2,∞)
(3,2,∞)
(∞,∞,2)
(4,5,∞)
(∞,∞,2)
(3,2,5)
(0,0,0)
Conjunctive arc of jobs
Disjunctive arc for machine M1
Disjunctive arc for machine M2
Disjunctive arc for machine M3
O_{ij} is the j_{th} operation of Job_i
$\overset{(p_{mi})}{Oij}$: p_{mi} is the processing time of O_{ij} on machine i

Figure 1. An example of the flexible job scheduling problem (FJSP).

Therefore, the flexible job-shop scheduling problem model can be described as follows:

$$\text{Minimize } \mathrm{C}_{\max} \tag{1}$$

Subject to

$$s_{jh} + x_{ijh} \times p_{ijh} \le c_{jh} (i = 1, \ldots, n; h = 1, \ldots, h_j) \tag{a}$$

$$c_{jh} \le s_{j(h+1)} (j = 1, \ldots n; h = 1, \ldots, h_j - 1) \tag{b}$$

$$c_{jh_i} \le c_{\max} (j = 1, \ldots n) \tag{c}$$

$$\begin{aligned} & s_{jh} + p_{ijh} \le s_{kl} + L(1 - y_{ijhkl}), \\ & (j = 0, \ldots, n; k = 0, \ldots, n; h = 1, \ldots, h_j; l = 1, \ldots h_k; i = 1, \ldots, m) \end{aligned} \tag{d}$$

$$\begin{aligned} & c_{jh} \le s_{j(h+2)} + L(1 - y_{iklj(h+1)}) \\ & (j = 1, \ldots, n; k = 0, \ldots, n; h = 1, \ldots, h_j - 1; i = 1, \ldots, m) \end{aligned} \tag{e}$$

$$\sum_{i=1}^{m_{jh}} x_{ijh} = 1 (h = 1, \ldots, h_j; j = 1, \ldots, n) \tag{f}$$

$$\sum_{j=1}^{n} \sum_{h=1}^{h_j} y_{ijhkl} = x_{ijh} (i = 1, \ldots, m; k = 1, \ldots, n; l = 1, \ldots, h_k) \tag{g}$$

$$\sum_{k=1}^{n} \sum_{h=1}^{h_k} y_{ijhkl} = x_{ijh} (i = 1, \ldots, m; j = 1, \ldots, n; h = 1, \ldots, h_k) \tag{h}$$

$$s_{ih} \ge 0, c_{jh} \ge 0 (j = 0, 1, \ldots, n; h = 1, \ldots, h_j) \tag{i}$$

where

$$x_{ijh} = \begin{cases} 1 & \text{If the operation } O_{jh} \text{ } is \text{ on machine } i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$y_{ijhkl} = \begin{cases} 1 & \text{if the operation } O_{jh} \text{ is before the operation } O_{kl} \text{ on machine } i \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

n is the total number of jobs; m is the total number of machines; h_j is the total number of operations of job $j(j \in n)$; O_{jh} is the h-th operation of job j ($h = 1,2,\dots,h_j$); p_{ijh} is the processing time of the h-th operation of job j on the machine ($I \in m$); s_{jh} is the start processing time of the h-th operation of job j; c_{jh} is the completion time of the h-th operation of job j; L is a given big integer; C_{max} is the maximum completion time.

Formulas (a) and (b) are the operation constraints of each job. Formula (c) is the time constraint of each job. Formulas (d) and (e) assume that only one machine can process one operation at a time. Formula (f) is the constraint of the machine, which assumes that the same operation can only be processed by one machine. Formulas (g) and (h) show that the circulating operation can be permitted on each machine. Formula (i) shows that each parameter must be positive number.

3. Shuffled Frog-Leaping Algorithm

In the shuffled frog-leaping algorithm, the population consists of frogs of a similar structure. Each frog represents a solution. The entire population is divided into many subgroups. Different subgroups can be regarded as different frog memes. Each subgroup performs a local search. They can communicate with each other and improve their memes among local individuals. After a pre-defined number of memetic evolution steps, information is passed between memeplexes in a shuffling process. Shuffling ensures that the cultural evolution towards any particular interest is free from bias. The local search and the shuffling processes alternate until satisfying the stopping criteria.

For a problem of D dimension, a frog is thought of as $F_i = \{f_{i1}, f_{i2}, \dots, f_{iD}\}$. The algorithm first randomly generates S frogs as the initial population and notes them in descending order according to the fitness of each frog. Then, the entire population is divided into m subgroups, and each subgroup contains n frogs. From the initial population, the first frog is selected in the first sub-group, and the second frog is selected in the second group, until the first m frog is selected in the m-th subgroup. Then, the (m + 1)-th frog is selected in the first subgroup. Repeat the process, until all frogs are distributed. In each subgroup, the frog with the best fitness and the one with the worst fitness are denoted as F_b and F_w, respectively; while in the total population, the frog with the best fitness is denoted as F_g. The main work of SFLA is to update the position of the worst-performing frog through an iterative operation in each sub-memeplex. Its position is improved by learning from the best frog of the sub-memeplex or its own population and position. In each sub-memeplex, the new position of the worst frog is updated according to the following equation.

$$d_i = Rand() \times (F_b - F_w) \quad (4)$$

$$F_w = F_w + d_i; (-d_{\max} \le d_i \le d_{\max}) \quad (5)$$

Formula (4) is used to calculate the updating step d_i. *Rand* ( ) is the random number between zero and one; Formula (5) updates the position of F_w. d_{max} is the maximum step. If a better solution is attained, then the better one will replace the worst individual. Otherwise, F_g will be used instead of F_b. Then, recalculate Formula (4). If you still cannot get a better solution, a new explanation generated randomly will replace the worst individual. Repeat until a predetermined number of iterations. Additionally, complete the round of the local search of various subgroups. Then, all subgroups of the frogs are re-ranked in mixed order and divided into sub-groups for the next round of local search.

The general structure of SFLA can be described as follows:

Step 1. Initialization. According to the characteristics and scale of the problem, a reasonable m and n is determined at first.

Step 2. Produce a virtual population with F individuals ($F = m * n$). For the D dimension optimization problem, the individuals of the population are D dimension variables and represent the frogs' current position. The fitness function is used to determine if the performance of the first individual's position is good.

Step 3. Divide the total population and the number of individuals in descending order.

Step 4. Divide the population into population m: Y_1, Y_2, ..., Y_m. Each sub-population contains n frogs. For example: if m = 3, then the first frog is put into the first population, the second frog is put into the second population, the third frog is put into the third population, the fourth frog is put into the first population, and so on.

Step 5. In each sub-population, with its evolution, the positions of individuals have been improved. The following steps are the process of the sub-population meme evolution.

Step 5.0. Set $i_m = 0$. i_m represents the number of the sub-populations, which is from zero to m. Set it equal to zero. i_m represents the number of evolutions, which is from zero to N (the maximum evolution iteration in each sub-population).

In each sub-population, compute F_b, F_w and F_g, respectively [24].

Step 5.1. $i_m = i_m + 1$.

Step 5.2. $i_n = i_n + 1$.

Step 5.3. Try to adjust the position of the worst frog. The moving distance of the worst frog should be: $d_i = Rand\ () * (F_b - F_w)$. After moving, the new position of the worst frog is: $F_w = F_w$ (the current position) + d_i.

Step 5.4. If Step 5.3 could produce a better solution, use the frog in the new position instead of the worst one; or use F_g instead of F_b.

Step 5.5. If better frogs cannot be generated after trying the above method, immediately generate the next individual to replace the worst frog F_w.

Step 5.6. If $i_n < N$, then perform Step 5.2.

Step 5.7. If $i_m < m$, then perform Step 5.1.

Step 6. Implementation of mixed operations.

After carrying out a certain number of meme evolutions, merge each sub-population Y_1, Y_2, ..., Y_m to X, namely X = $\{Y_k$, k = 1, 2,... $M\}$. Descend X again, and update the best frog F_g in the populations.

Step 7. Check the termination conditions. If the iterative termination conditions are met, then stop. Otherwise, do Step 4 again.

Usually, the frog reaching the defined maximum evolution number or representing the global optimal solution does not changes anymore, and the shuffled frog-leaping algorithm stops.

4. Improved SFLA for FJSP

4.1 Generation of Solutions

Because the flexible job shop scheduling problem is a discrete combinatorial optimization problem, the shuffled frog-leaping algorithm requires discretization to adapt to the optimization of this problem. The form of the code is the key of the algorithm and with an appropriate FJSP could be solved effectively. This paper adopts a segmented integer coding method. The coding method is operated easily, which is adapted to T-FJSP and P-FJSP. In the method, the solution of the code consists of two parts, namely machine selection and operation selection.

Figure 2 is an example of the coding scheme of Figure 1. In the encoding form, the machine section length and the operation section length are both equal to the sum of all operations. In the machine section, each position is ranked according to the process order of jobs and operations. From Figure 2, it can be found that the operation O_{11} can be processed on five machines $\{M_1, M_2, M_3, M_4, M_5\}$, so five is placed in the first square. The operation O_{12} can be processed on two machines $\{M_2, M_4\}$, and in the same way, the machine section is coded. In the operation section, the code is noted by the time sequence of all of the operations of all jobs. For example, O_{21} is the first operation; thus, the first square is placed at two, which is the job number. As shown in Figure 2, the code of the operation section is 2321321, which represents the operation sequence: O_{21}-O_{31}-O_{22}-O_{11}-O_{31}-O_{23}-O_{12}.

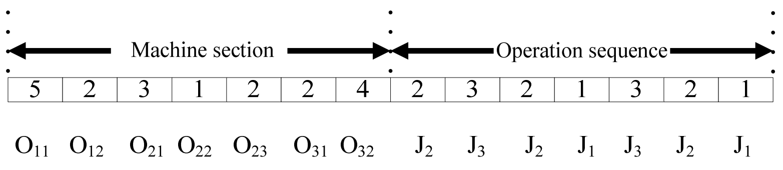


Figure 2. An example of the coding scheme.

4.2 Local Search

A local search strategy was designed to search the local optimum in different directions and to make the local optimal individual in the ethnic groups gradually tend toward the optimal individual. However, the frog population is divided into a plurality of groups, and each group is locally searched; SFLA can easily fall into a premature or local-best solution. In order to avoid frog individuals getting trapped in a local optimum prematurely and to accelerate the convergence rate, a specific principle is used to select a certain number of individual to construct the sub-population.

The local search formula is changed to the following:

$$s = \begin{cases} min\{int[rand(F_b - F_w)], s_{\max}\}, F_b - F_w \geq 0 \\ max\{int[(F_b - F_w)], -s_{\max}\}, F_b - F_w < 0 \end{cases} \tag{6}$$

$$F_g = F_w + S \tag{7}$$

S means the adjustment vector, and S_{max} means the maximum step size allowed to change by frog individual. Set $F_w = \{1, 3, 5, 2, 4\}$, $F_b = \{2, 1, 5, 3, 4\}$; the maximum step size allowed to change $S_{max} = 3$, rand = 0.5. Therefore, $F_g(1) = 1 + \min\{\text{int}[0.5 \times (2 - 1)],3\} = 1$; $F_g(2) = 3 + \min\{\text{int}[0.5 \times (1 - 3)],3\} = 2$, and so on. Finally, the new solution F_g can be attained as $\{1, 2, 5, 4, 3\}$.

4.3 Improvement Strategies

Although in the literature [25], the shuffled frog-leaping algorithm with local search can guarantee the feasibility of the updated solution, the step size is still selected randomly. Therefore, on the basis of the literature [25], this paper designed the local search strategy by introducing the adjustment factor and adjustment order.

Adjustment order is an attempt to adjust one feasible solution to another one [26]. In local search, the method in which the worst solution in a sub-group is optimized by the adjustment order is more reasonable than the simple step size selection. Therefore, this paper introduced this idea into updating the worst solution in the sub-group during the local search.

4.3.1 Adjustment Factor

The steps for the adjustment factor are as follows: If the solution set is $U = (U^i)$ and the operation set is (1, 2,… , d), the adjustment is defined as $TO(i_1,i_2)(i = 1,2,\dots,\mathrm{d})$. Thus, the operation of U_{i1} will be put before U_{i2}. $U' = U + TO(i_1,i_2)$ is the new solution of U based on the adjustment factor. For example, if $U = (1\ 3\ 5\ 2\ 4)$, the adjustment factor is TO (4,2), then $U' = U + TO$ (2,4) = (1 2 3 5 4).

4.3.2 Adjustment Sequence

The adjustment sequence is one or more adjustment factors are ranked in sequence, which is noted as ST. $ST = \left(TO_1, TO_2, \dots, TO_N\right) \cdot TO_1, TO_2, \dots, TO_N$ are the adjustment factors. U_A and U_B are two different solutions. The adjustment sequence $ST\left(U_B \Theta U_A\right)$ means adjusting U_A to U_B, namely, $U_B = U_A + ST\left(U_B \Theta U_A\right) = U_A + \left(TO_1, TO_2, \dots, TO_N\right) = \left[\left(U_A + TO_1\right) + TO_1\right] + \dots + TO_N$. For example, let $U_A = (1\ 3\ 5\ 2\ 4)$, $U_A = (3\ 1\ 4\ 2\ 5)$. We need make an adjustment sequence $ST\left(U_B \Theta U_A\right)$, and make sure $U_A + ST\left(U_B \Theta U_A\right) = U_B \cdot U_B^1 = U_A^2 = 3$. Therefore, for the first adjustment sequence $TO_2\left(2,1\right) \cdot U_A^1 = U_A + TO_1$, we can get $U_A^1 = \left(3\,1\,5\,2\,4\right); U_B^3 = U_{A1}^5 = 4$, and the second adjustment sequence is $TO_2\left(5,3\right) \cdot U_A^2 = U_A^1 + TO_2\left(5,3\right)$, and so on. Then, we can get $ST(U_B \Theta U_A) = (TO_1(2,1), TO_2(5,3), TO_3(5,4))$.

4.3.3 Extremal Optimization

Extremal optimization (EO) is an optimization that evolves a single individual (*i.e.*, chromosome) $S = (x_1, x_2, \dots, x_d)$. Each component x_i in the current individual S is considered to be a species and is assigned a fitness value k_i. There is only a mutation operator in EO, and it is applied to the component with the worst species successively. The individual can update its components and evolve toward the optimal solution. The process requires a suitable representation that allows the solution components to be assigned a quality measure (*i.e.*, fitness). This approach differs from holistic ones, such as evolutionary algorithms, which assign equal fitness to all components of a solution based on their collective evaluation against an objective function. The EO algorithm is as follows [27].

Step 1. Randomly generate an individual $S = (x_1, x_2, x_3, x_4)$. Set the optimal solution $\mathrm{S}_{\mathrm{best}} = S$.

Step 2. For the current S:

(1) Evaluate the fitness λ_i for each component $x_i, i = 1, 2, \ldots, d$. And d is the vector number of the solution space;

(2) Find j satisfying $\lambda_j \le \lambda_i$ for all i , x_j being the worst species;

(3) Choose $S' \in N(S)$, such that x_j must change its state by mutation operator; $N(S)$ is the neighborhood of S ;

(4) Accept $S = S'$ unconditionally;

(5) If the current cost function value is less than the minimum cost function value, *i.e.*, $C(S) < C(S_{\text{best}})$, then set $S_{best} = S$.

Step 3. Repeat Step 2 as long as desired;

Step 4. Return S_{best} and $C(S_{best})$.

4.4 The Update the Strategy of the Frog Individual

The differences among frog individuals are their adjustment sequences. The number of adjustment sequences is not negative. Therefore, the update strategy of the frog individual is:

$$L = \min\left\{\text{int}\left[rand\ length\left(ST\left(U_B \Theta U_W\right)\right)\right], l_{\max}\right\} \tag{8}$$

$$S = \left\{\left(TO_i\right)/TO_i \in ST(U_B \Theta U_W), i = 1, 2, \ldots, l\right\} \tag{9}$$

$$U_q = U_W + s \tag{10}$$

In which the length of $\left(ST\left(U_B \Theta U_W\right)\right)$ means the number of all adjustment factors in adjustment sequence $ST(U_B \Theta U_W)$, l means the number of adjustment factor in $ST(U_B \Theta U_W)$ chosen to update U_W and s means update the adjustment sequence for U_W . Figure 3 is used to explain an example of the update strategy of the frog individual from U_B to Uq . From Figure 3, it can be attained that the length of $\mathrm{ST}\left(\mathrm{U_B}\Theta U_W\right) = 5, l = 3, s = \left(TO1, TO2, TO3\right)$.

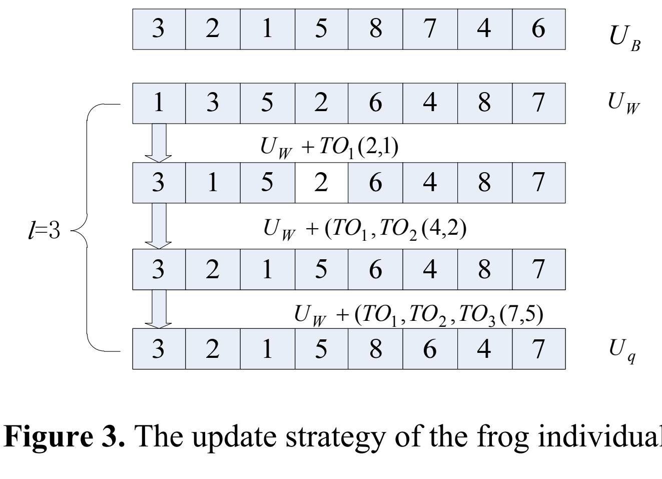


Figure 3. The update strategy of the frog individual.

5. Numerical Experiments and Discussion

This section describes the computational experiments used to test the performance of the proposed algorithm. There are 10 FJSP benchmark problems [27], which have been widely used as benchmark problems to examine the feasibility of the algorithms [2,28,29]. The number of jobs ranges from 10 to 20; the number of machines ranges from four to 15; the number of operations for each job ranges from five to 15; and the number of operations for all jobs ranges from 55 to 240. Lower best-known solutions (*LB*) and upper best-known solutions (*UB*) represent the lower and upper values from the best-known solutions. In this paper, the improved SFLA (ISFLA) parameters used for the instances are $P = 1000$, $Q = 1000$, $N = 5000$, $\rho = 0.8$ and $P_c = 0.4$ In order to conduct the experiment, we implement the algorithm in C++ language and run it on a PC with 2.0 GHz, 512 MB of RAM memory and a Pentium processor running at 1,000 MHz. The results of the ISFLA on the instances are shown in Table 1, which are compared with the genetic algorithm (GA) [28], tabu search (TS) [29], artificial immune algorithm (AIA) [20] and ant colony optimization (ACO) [2]. The numbers in bold are the best among the five approaches. We used 10 standard cases (MK01-MK02) proposed in [27] to test our model.

Table 1. Performance comparison between improved shuffled frog-leaping algorithm (ISFLA) and other algorithms. The numbers in bold are the best among the five approaches. TS, tabu search; AIA, artificial immune algorithm.

Problem	$n \times m$	f	(*LB*, *UB*)	GA	TS	AIA	ACO	ISFLA
MK01	10 × 6	2.09	(36, 42)	40	40	40	40	40
MK02	10 × 6	4.1	(24, 32)	26	26	26	26	26
MK03	15 × 8	3.01	(204, 211)	204	204	204	204	204
MK04	15 × 8	1.91	(48, 81)	60	60	60	60	60
MK05	15 × 4	1.71	(168, 186)	173	173	173	173	173
MK06	10 × 15	3.27	(33, 86)	63	**58**	63	**58**	**58**
MK07	20 × 5	2.83	(133, 157)	**139**	144	140	140	**139**
MK08	20 × 10	1.43	523	523	523	523	523	523
MK09	20 × 10	2.53	(299, 369)	311	**307**	312	**307**	**307**
MK10	20 × 10	2.98	(165, 296)	212	**198**	214	**198**	**198**

From Table 1, it can be found that the ISFLA used in this paper is able to find most of the best-known solutions, especially for problems of higher flexibility. Besides, the method can provide competitive solutions for most problems. We also find that our ISFLA has better performance than or similar performance as the ACO, TS and the AIA. The results indicate that our algorithm is a potential algorithm for FJSP when compared with other heuristic algorithms (GA, TS and AIA).

To evaluate the three improved strategies, four shuffled frog-leaping algorithms with different strategies are constructed. This first is a standard SFLA with the adjustment factor (denoted by SFLA + AF); the second one is a standard SFLA with the adjustment order (AO) (denoted by SFLA + AO); the third one is a standard SFLA with the extremal optimization (denoted by SFLA + EO); and the fourth one is a standard SFLA with the three improved strategies (denoted by ISFLA). The comparison results about the optimal value, average value and relative error and average computing time between the four different SFLAs are shown in Table 2.

For the MK09 problem, the errors are within 10% compared to the approximate optimal solution. The simulation results show that when the improved leapfrog algorithm does not fall into the local convergence, the combination of three improved strategies can improve the searching performance of SFLA and accelerate the convergence speed of the algorithm. From the results in Tables 1 and 2, it can be found that the improved algorithm in this paper is a powerful method for solving FJSP with reasonable precision and computing speed.

Table 2. Computational results of the MK09 problem of several methods. AF, adjustment factor; AO, adjustment order; EO, extremal optimization.

Type	Average value	Optimal value	Relative error/%	Average computing time
SFLA	332.98	325	9.64	51
SFLA + AF	317.17	312	2.77	27
SFLA + AO	316.42	312	2.69	29
SFLA + EO	313.83	307	2.09	38
ISFLA	310.54	307	0.42	32

6. Conclusions

This paper presents an improved SFLA with some strategies for FJSP. The objective of the research is to minimize the makespan in FJSP. The shuffled frog-leaping algorithm with local search can guarantee the feasibility of the updated solution; however, the step size is still selected randomly. To attain a reasonable step size, the adjustment factor and adjustment order are adopted. Moreover, an extremal optimization is used to exchange the information among individuals. The effectiveness of the improved shuffled frog leaping algorithm is evaluated using a set of well-known instances. The results indicate that the results gained by ISFLA are often the same or slightly better than those gained by the other algorithms for the FJSP. Furthermore, the results also show that the improved strategies effectively improve the performance of the algorithm.

References

1. Garey, M.R.; Johnson, D.S.; Sethi, R. The complexity of f1owshop and jobshop scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129.
2. Yao, B.Z.; Hu, P.; Lu, X.H.; Gao, J.J. Zhang, M.H. Transit network design based on travel time reliability. *Transp. Res. Part C* **2014a**, *43*, 233–248.
3. Yao, B.Z.; Yao, J.B.; Zhang, M.H.; Yu, L. Improved support vector machine regression in multi-step-ahead prediction for rock displacement surrounding a tunnel. *Scientia Iranica* **2014b**, *21*, 1309–1316.
4. Yao, B.Z.; Hu, P.; Zhang, M.H.; Jin, M.Q. A Support Vector Machine with the Tabu Search Algorithm For Freeway Incident Detection. *Int. J. Appl. Math. Comput. Sci.* **2014c**, *24*, 397–404.
5. Yu, B.; Yang, Z.Z.; Yao, B.Z. An Improved Ant Colony Optimization for Vehicle Routing Problem. Eur. J. Oper. Res. **2009**, *196*, 171–176.
6. Yu, B.; Yang, Z.Z. An ant colony optimization model: The period vehicle routing problem with time windows. *Transp. Res. Part E* **2011**, *47*, 166–181.
7. Yu, B.; Yang, Z.Z.; Li, S. Real-Time Partway Deadheading Strategy Based on Transit Service Reliability Assessment. *Transp. Res. Part A* **2012a**, *46*, 1265–1279.
8. Yu, B.; Yang, Z.Z.; Jin, P.H.; Wu, S.H.; Yao, B.Z. Transit route network design-maximizing direct and transfer demand density. Transp. Res. Part C **2012b**, *22*, 58–75.
9. Yao, B.Z.; Yang, C.Y.; Yao, J.B.; Hu, J.J.; Sun, J. An Improved Ant Colony Optimization for Flexible Job Shop Scheduling Problems. *Adv. Sci. Lett.* **2011**, *4*, 2127–2131.

10. Bak, P.; Sneppen, K. Punctuated equilibrium and criticality in a simple model of evolution. *Phys. Rev. Lett*. **1993**, *71*, 4083–4086.
11. Fattahi, P.; Mehrabad, M.S.; Jolai, F. Mathematical Modeling and Heuristic Approaches to Flexible Job Shop Scheduling Problems. *J. Intell. Manuf.* **2007**, *18*, 331–342.
12. Gao, L.; Sun, Y.; Gen, M. A Hybrid Genetic and Variable Neighborhood Descent Algorithm for Flexible Job Shop Scheduling Problems. *Comput. Oper. Res*. **2008**, *35*, 2892–2907.
13. Eusuff, M.; Lansey, K. Optimization of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm. *J. Water Resour. Plan. Manag*. **2003**, *129*, 10–25.
14. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 1995; pp. 1942–1948.
15. Alireza, R.V.; Mostafa, D.; Hamed, R.; Ehsan, S. A novel hybrid multi-objective shuffled frog-leaping algorithm for a bi-criteria permutation flow shop scheduling problem. *Int. J. Adv. Manuf. Technol*. **2009**, *41*, 1227–1239.
16. Bhaduri, A. Color image segmentation using clonal selection-based shuffled frog leaping algorithm. In Proceedings of the ARTCom 2009—International Conference on Advances in Recent Technologies in Communication and Computing, Kottayam, India, 27–28 October 2009; pp. 517–520.
17. Babak, A.; Mohammad, F.; Ali, M. Application of shuffled frog-leaping algorithm on clustering. *Int. J. Adv. Manuf. Technol*. **2009**, *45*, 199–209.
18. Li, X.; Luo, J.P.; Chen, M.R.; Wang, N. An improved shuffled frog-leaping algorithm with extremal optimization for continuous optimization. *Inf. Sci*. **2012**, *192*, 143–151.
19. Luo, J.P.; Li, X.; Chen, M.R. Improved Shuffled Frog Leaping Algorithm for Solving CVRP. *J. Electr. Inf. Technol*. **2011**, *33*, 429–434.
20. Boettcher, S.; Percus, A.G. Extremal Optimization: Methods derived from Co-Evolution. In Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA, 13 April 1999; pp. 101–106.
21. Bagheri, A.; Zandieh, M.; Mahdavi, I.; Yazdani, M. An Artificial Immune Algorithm for The Flexible Job-Shop Scheduling Problem. *Future Gener. Comput. Syst.* **2010**, *26*, 533–541.
22. Boettcher, S. Extremal Optimization for the Sherrington-Kirkpatrick Spin Glass. *Eur. Phys. J. B* **2005**, *46*, 501–505.
23. Chen, M.R.; Lu, Y.Z. A novel elitist multiobjective optimization algorithm: Multiobjective extremal optimization. *Eur. J. Oper. Res*. **2008**, *188*, 637–651.
24. Alireza, R.V.A.; Ali, H.M. A hybrid multi-object shuffled frog leaping algorithm for a mixed-model assembly line sequencing problem. *Comput. Ind. Eng*. **2007**, *53*, 642–666.
25. Luo, X.H.; Yang, Y.; Li, X. Solving TSP with shuffled frog-leaping algorithm. In Proceedings of the Eighth International Conference on Intelligent Systems Design and Applications, Kaohsiung, Taiwan, 26–28 November 2008; pp. 228–232.
26. Wang, C.R.; Zhang, J.W.; Yang, J.; Hu, C. A modified particle swarm optimization algorithm and its application for solving traveling salesman problem. In Proceedings of the ICNN&B ′05. International Conference on Neural Networks and Brain, Beijing, China, 13–15 October 2005; pp. 689–694.
27. Brandimarte, P. Routing and Scheduling in a Flexible Job Shop by Taboo Search. *Ann. Oper. Res*. **1993**, *41*, 157–183.

28. Pezzella, F.; Morganti, G.; Ciaschetti, G. A Genetic Algorithm for the Flexible Job-Shop Scheduling Problem. *Comput. Oper. Res*. **2008**, *35*, 3202–3212.
29. Mastrolilli, M.; Gambardella, L.M. Effective Neighbourhood Functions for the Flexible Job Shop Problem. *J. Sched*. **2000**, *3*, 3–20.