An elitist quantum-inspired evolutionary algorithm for the flexible job-shop scheduling problem

Xiuli Wu[1], Shaomin Wu[2*]

[1]School of Mechanical Engineering, University of Science and Technology Beijing, Beijing, China

[2]Kent Business School, University of Kent, Canterbury, Kent CT2 7PE, UK

Abstract: The flexible job shop scheduling problem (FJSP) is vital to manufacturers especially in today's constantly changing environment. It is a strongly NP-hard problem and therefore metaheuristics or heuristics are usually pursued to solve it. Most of the existing metaheuristics and heuristics, however, have low efficiency in convergence speed. To overcome this drawback, this paper develops an elitist quantum-inspired evolutionary algorithm. The algorithm aims to minimise the maximum completion time (makespan). It performs a global search with the quantum-inspired evolutionary algorithm and a local search with a method that is inspired by the motion mechanism of the electrons around atomic nucleuses. Three novel algorithms are proposed and their effect on the whole search is discussed. The elitist strategy is adopted to prevent the optimal solution from being destroyed during the evolutionary process. The results show that the proposed algorithm outperforms the best-known algorithms for FJSPs on most of the FJSP benchmarks.

Keywords: Flexible job shop scheduling problem, Quantum-inspired evolutionary algorithm, Convergence speed, local search

[*]Corresponding author. Email: s.m.wu@kent.ac.uk.

Suggested citation: Wu, X., & Wu, S. (2015). An elitist quantum-inspired evolutionary algorithm for the flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, DOI: 10.1007/s10845-015-1060-6.

1. Introduction

Efficiently solving the job shop scheduling problem is vital for many manufacturers in today's constantly changing environment (Gen and Lin 2014). It has therefore attracted considerable researcher for recent decades (see Ho and Tay 2004, Xia et al.2005, Baykasoglu et al 2004, Xing et al. 2009 for example). The flexible job shop scheduling problem (FJSP) is an extension of the job shop scheduling problem (JSP). Different from JSP, FJSP allows operations to be processed on any of available machines. As a result, two new challenges facing FJSP are (1) to assign each operation to an appropriate machine and (2) further to schedule the assigned operations on the machine. It has been proved that the FJSP is strongly NP-hard in 1993 (Brandimarte, 1993). It is therefore natural to look for heuristics or metaheuristics to search the optimal or near-optimal solutions for FJSP.

The FJSP was first addressed by Brucker and Schlie (1990) and it can be categorised into two groups: (1) the group aiming at the single objective FJSP and (2) the group aiming at the multi-objective FJSP.

The single objective FJSP only takes the makespan as the optimization objective. In this category, the existing heuristic algorithms can be further classified into two sub-categories: the hierarchical and the integrated. The hierarchical one assigns operations to machines and then optimises the sequence of the assigned operations on the machines separately, whereas the integrated one combines the two steps together. The hierarchical algorithms attempt to solve the difficulty of the FJSP by decomposing a FJSP into a sequence of sub-problems. Brandimarte (1993) was the first to apply a hierarchical taboo search (TS) algorithm to solve the FJSP, with an emphasis on the minimum makespan problem. Ho and Tay (2004) integrated dispatching rules and genetic algorithm to solve the two sub-problems, respectively. In contrast, the integrated approaches generally achieve better results (Dauzère-Pérès and Paulli, 1997) although they are much more difficult in implmentation. Many heuristics and metaheuristics are proposed to solve FJSP, such as the TS algorithm (Dauzère-Pérès and Paulli ,1997), the greedy algorithm (Mati et al.,2001), the climbing discrepancy search approach (Hmida et al., 2010), the genetic algorithm (Pezzella

et al., 2008, Nagamani et al., 2012), the parallel variable neighborhood search algorithm (Yazdani et al., 2010), the ant colony optimization algorithm (Xing et al., 2010), the bi-population based estimation-of-distribution algorithm (Wang et al., 2012), etc..

As for the multi-objective FJSP, the machine load and the maximal total load have been taken as the optimization objectives as well as makspan. This is to balance the working load on machines. Following a hierarchical approach, Kacem et al. (2002) adopted a local search algorithm to select machines and solved the operation scheduling problem with the genetic algorithm. They aimed to optimise the makespan, total machine load and the maximal load. Similarly, Xia and Wu (2005) used particle swarm optimisation algorithm to solve the machine assignment problem and used the simulated annealing algorithm to solve the operation scheduling problem. Recently, Chen et al. (2012) developed a scheduling algorithm based on GA. As to the integrated approach, Loukil, et al. (2005) utilised the simulated annealing algorithm to solve the FJSP. Gao et al. (2008) proposed a genetic algorithm hybridizing with the variable neighborhood search. Li and Pan (2012) proposed an effective discrete chemical-reaction optimisation algorithm for solving the FJSP subject to maintenance activity constraints.

In summary, various methods for solving the FJSP have been proposed in the existing literature. Nevertheless, an open challenge is how the convergence speed of the search process can be improved, especially for large-size FJSP. Many heuristics and metaheuristics spend thousands of iterations before reaching the optimal solution. Such low efficiency may hamper their practical applications because the production scheduling requires rapid responses to changing demands. This may be also the reason that so many different evolutionary algorithms have been developed to solve the FJSP. In this paper, we will propose an evolutionary algorithm, the quantum-inspired evolutionary algorithm (QEA), which was originally introduced by Shor (1994), to tackle the FJSP. QEA has found a wide spectrum of applications in recent years due to its fast convergence speed (see Shor 1994, Wang et al. 2012a, Lu and Yu 2013, for example). This strength attracts us to use it to solve the FJSP.

It is noteworthy that, to the best of our knowledge, QEA has not been previously used

for FJSP. In this paper, we develop a fast algorithm based on QEA to solve the FJSP. The global search is performed with the QEA and the local search is performed with a novel method inspired by the mechanism of the motion of the electrons around an atomic nucleus. Besides, a new chromosome representation is proposed to enrich the FJSP representation ways.

The rest of this paper is organised as follows. Section 2 formulates the problem. Section 3 proposes a new algorithm for FJSP. Section 4 shows the experiment results. Section 5 concludes our research.

2. Model Formulation for FJSP

2.1 Notations and assumptions

Some notations are given in Table 1.

Table1 Notations for FJSP formulation

Notations	Description
n	total number of jobs
m	total number of machines
J_j	job index, j=1,2,…n
n_j	number of operations of job J_j
M_k	Machine index, k=1,2,…m
O_{ij}	*i-th* operation of job J_j
p_{ijk}	processing time of the operation O_{ij} on machine M_k
S_{ij}	set of available machines for the operation O_{ij}
C_{ij}	completion time of operation O_{ij}
C_{max}	The makespan
X_{ijk}	X_{ijk}=1 if machine M_k is selected for operation O_{ij}, X_{ijk}=0 otherwise
Y_{hgij}	$=\begin{cases}-1 & O_{hg} \text{ is executed immediately before } O_{ij} \\ 0 & O_{hg} \text{ and } O_{ij} \text{ is nonadjacent on machine } M_k \\ 1 & O_{hg} \text{ is executed immediately after } O_{ij}\end{cases}$
gap	idle time interval between two adjacent operations

Assumptions are set as following.

(1) All jobs and machines are available at time 0.

(2) Jobs have no associated priority.

(3) At a time point, a machine cannot perform more than one operation.

(4) The value p_{ijk} is given in advance.

(5) Setup time for each operation is negligible.

(6) Non-preemptive. An operation must be completed without interruption once started.

(7) Jobs are available for processing on a next machine immediately after completing processing on the previous machine.

2.2 Model Formulation

The single objective FJSP can be defined as follows. There are n jobs indexed by $\boldsymbol{J}$= ($J_1, J_2,\ldots, J_n$) to be processed on m machines. Job J_j comprises n_j operations to be executed one after another according to a pre-specified sequence. More than one machine (S_{ij}) is available for each operation O_{ij}. The i^{th} operation of job J_j can be processed by machine M_k from the m machines (O_{ijk}) and occupies the machine M_k for p_{ijk} time units. The scheduling problem is to assign operations to machines in an appropriate way and to schedule the job operations to optimise makespan subject to the above assumptions.

We formulate the FJSP based on the recommendation from Demir and Kürşat Işleyen (2013).

The objective: $\min C_{max} = \min(\max(C_{ij}))$ (1)

s.t.

$C_{ij} - C_{(i-1)j} \geq p_{ijk}X_{ijk} \quad i = 2,3,\ldots,n_j$ (2)

$$\left(C_{ij} - C_{hg} - p_{ijk}\right)X_{hgk}X_{ijk}\left(\frac{Y_{hgij}}{2}\right)\left(Y_{hgij} - 1\right) + \left(C_{hg} - C_{ij} - p_{hgk}\right)X_{hgk}X_{ijk}\left(\frac{Y_{hgij}}{2}\right)\left(Y_{hgij} + 1\right) \geq 0 \quad (3)$$

$$gap = \left(C_{ij} - C_{hg} - p_{ijk}\right)X_{hgk}X_{ijk}\left(\frac{Y_{hgij}}{2}\right)\left(Y_{hgij} - 1\right) + \left(C_{hg} - C_{ij} - p_{hgk}\right)X_{hgk}X_{ijk}\left(\frac{Y_{hgij}}{2}\right)\left(Y_{hgij} + 1\right) \quad (4)$$

$\sum_k X_{ijk} = 1, k \in S_{ij}, \forall i, j$ (5)

$$Y_{hgij} \in \{-1,0,1\} \quad (6)$$

$$X_{ijk} \in \{0,1\} \quad (7)$$

Equation (1) is the objective function. Inequality (2) is the precedence constraints. Inequality (3) ensures that there are no overlaps between operations on each machine. Equation (4) computes the length of each idle time interval. Equations (5) - (7) are constraints on the decision variables.

Table 2 shows an FJSP example. The number in each entry is p_{ijk}. If p_{ijk}=0, it means the machine is not available for O_{ij}.

Table 2 An instance of FJSP

		Machine 1	Machine 2	Machine 3
J_1	O_{11}	2	3	0
	O_{21}	0	5	2
	O_{31}	3	6	4
J_2	O_{12}	0	4	5
	O_{22}	5	5	6
	O_{32}	2	4	8
J_3	O_{13}	4	0	6
	O_{23}	4	4	4
	O_{33}	5	6	7

3. Elitist Quantum-inspired Evolutionary Algorithm for FJSP

This section investigates how to solve the FJSP with QEA for obtaining the minimal makespan.

3.1 Procedure of EQEA for FJSP

QEA utilises the concepts of a quantum bit, a superposition of states and the collapse of states. Like other evolutionary algorithms, QEA is also characterised by the representation of the individuals, the evaluation function and the population dynamics. Instead of using binary, numeric or symbolic sequences to represent feasible solutions, QEA uses quantum bit (Q-bit) chromosomes to encode probabilistic representation. A Q-bit chromosome can represent a linear superposition of states in the search space. As such, the Q-bit representation has a better characteristic of population diversity than any other

representation. Meanwhile, a quantum rotation gate is used as the updating mechanism. The mechanism helps guide the search direction to the optimal area, and therefore increase the convergence speed.

To avoid the oscillation, the elitist strategy can be integrated into the QEA. As such an integrated QEA is formed. We name it elitist QEA (EQEA) for short. The main structure of the algorithm (Fig. 1) creates novelty in the following four aspects.

- a new representation for the FJSP;
- the integration of the niche technology, which can prevent the simple QEA from converging to local optima;
- the integration of the elitist strategy, which can thoroughly prevent the convergence process from oscillation and can also speed up the convergence process efficiently; and
- the local search process, which enhances the local search ability.

The forementioned four points will be elaborated in Section 3.2.

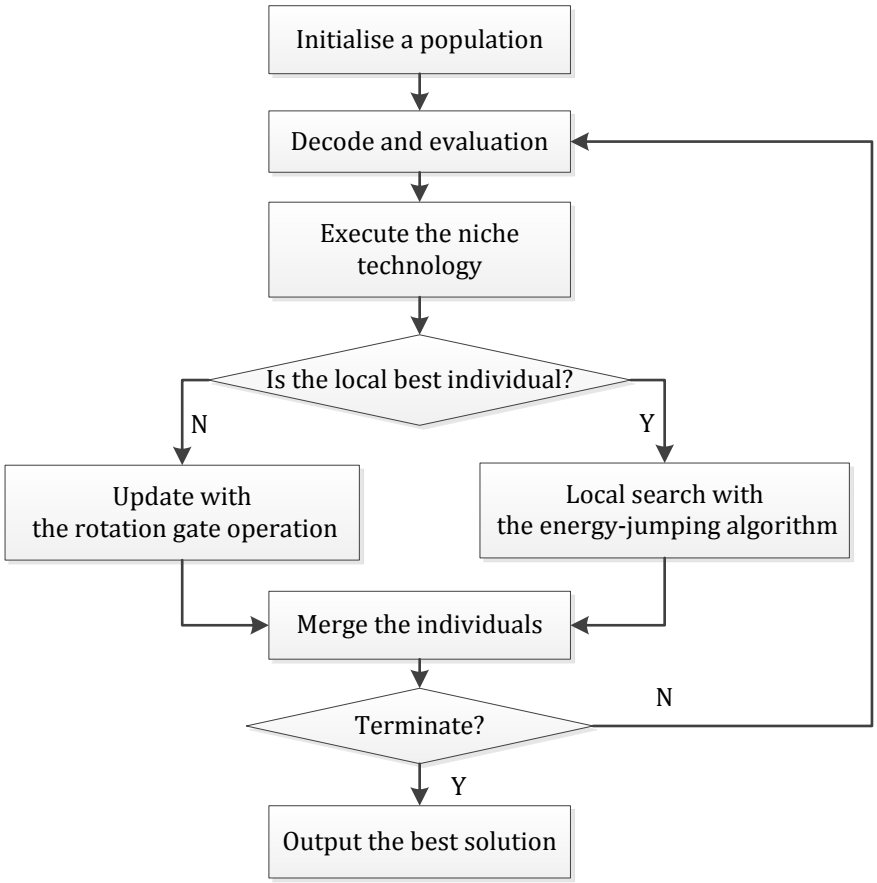


Fig. 1. The framework of the EQEA

3.2 Details of the EQEA

3.2.1 Q-bit chromosome encoding and the population initialisation

The EQEA evolves with a population like GA. The population comprises a number of chromosomes. A chromosome is made up of Q-bit genes. Unlike the binary, numeric, or symbolic representation, the state of a Q-bit gene can be represented by (8).

$$\psi = \alpha|0\rangle + \beta|1\rangle \tag{8}$$

where $|0\rangle$ and $|1\rangle$ represent bit values '0' and '1', respectively; α and β are complex numbers that specify the probability amplitudes of the corresponding states. $|\alpha|^2$ and $|\beta|^2$, satisfying $|\alpha|^2 + |\beta|^2 = 1$, denote the probability of the Q-bit gene that will be found in the state '1' or the state '0', respectively. A Q-bit gene may be in state '1', state '0', or in any linear superposition of the two. The advantage of using the Q-bit gene is that it can represent a linear superposition of solutions. For example, if there is a system represented by q Q-bits, the system can represent 2^q states at the same time. However, in the act of observing a quantum state, it collapses to a single state. For simplicity, we refer this Q-bit gene as a q-Q-bit gene, where q is the number of Q-bits in a gene. Each q-Q-bit is the smallest unit of information.

A Q-bit chromosome consists of a certain number of q-Q-bit genes. For example, if q=1, the chromosome is defined by:

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_f \\ \beta_1 & \beta_2 & \cdots & \beta_f \end{bmatrix} \tag{9}$$

where $|\alpha_i|^2 + |\beta_i|^2 = 1, i = 1,2, \ldots f$ and there are f units separated by vertical lines.

For simplicity, the sine function and the cosine function are used to generate α and β, respectively. Obviously, the condition $|\alpha|^2 + |\beta|^2 = 1$ is satisfied whatever value the angle has. Therefore, the angle is computed by a random number $\gamma \in (0,1)$ multiplied with 2π. For example, if f=6, a chromosome composing of the 1-Q-bit genes is generated as the following:

$$p_i = \begin{bmatrix} -0.8452 & 0.7849 & 0.9346 & -0.6023 & 0.3865 & 0.9981 \\ 0.5344 & -0.6196 & 0.3558 & 0.7983 & -0.9223 & 0.0619 \end{bmatrix}$$

The evolving process is executed on a population. A population is initialised by randomly generating a group of Q-bit chromosomes.

3.2.2 Chromosome converting mechanism for FJSP

The chromosome composing of *m*-Q-bit genes cannot directly be used to represent FJSP. A converting mechanism, therefore, is needed. When we structure converting mechanism, we must consider the characters of the FJSP presentation firstly to propose a targeting chromosome representation as the converting target.

(1) Targeting chromosome representation

In existing literature, there are four types of chromosome representations for FJSP.

Chromosome A (Chen et al, 1999) comprises two integer strings (A1 and A2). The length of each string equals the total number of operations. String A1 assigns a machine index to each operation. The value of the j-th position of the string A1 indicates the machine performing the j-th operation. String A2 encodes the order of operations on each machine.

Chromosome B (Paredis, 1992) also comprises two strings (B1 and B2). String B1 is identical to A1. String B2 is a bit string that gives the order of any pair of operations. A bit value of 0 indicates that the first operation in the paired-combination must be performed before the second operation.

Chromosome C (Ho and Tay, 2004) is composed of two strings (C1 and C2), too. It represents an instance of the FJSP. String C1 encodes the order of the operations. It does not specify the order of operations for the same job as this is already implied by its index value. String C2 represents the machine assignment to operations (like A1 and B1) but with a twist. To ensure solution feasibility, the machine index is manipulated so that the string will always be valid.

Chromosome D (Tay and Wibowo, 2004) is composed of strings like that in chromosome B and C. It comprises three strings (D1, D2 and D3). D1 and D2 are equivalent to C1 and C2, respectively, while D3 is similar to B2.

Different from the aforementioned four representation ways, a new chromosome representation for FJSP is proposed. This is the targeting chromosome and we name it chromosome E according to the naming rule in Tay and Wibowo (2004).

Chromosome E modifies the operation-based encoding (Gen et al.,1994). Each chromosome is composed of $n\times\max\{n_j\}$ numbers and each number corresponding a job occurs $\max\{n_j\}$ times. A surplus part represents those jobs whose operation number are less than $\max\{n_j\}$. The surplus part is referred to as virtual operations that don't take any machine time. For the 3×3 problem (Table 2), a feasible chromosome encoded with job number is [2 1 3 3 2 2 1 1 3]. The matchup between the chromosome and the sequence of the operations is described in Fig. 2.

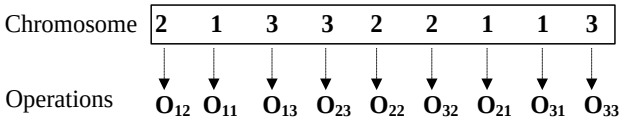


Fig. 2. Matchup between the chromosome and the operations

The space complexity of the chromosome E is less than the others. Following Tay and Wibowo (2004), we denote T as the total number of job operations in an FJSP. In the best case when the numbers of operations for each job are the same, the length of chromosome E is T. In the worst case, however, when there is a special job whose operation number is far greater than those of the others (assuming there is only 1 operation for these jobs), the length of the chromosome is $n^*(T\text{-}n+1)$. The worst case occurs with a very low probability because there are always similar jobs to be processed in a practical production workshop. So one can confidently conclude that the length of the chromosome E is normally shorter than that of the others. To make a comparison, we list the space complexity of the five chromosome representations in Table 3. Variable d denotes the length of the string $D3$ (Tay and Wibowo 2004).

Table 3 Space complexity of the representations

Chromosome representation	**Chromosome length**
Chromosome A (Chen et al., 1999)	$2T$
Chromosome B (Paredis, 1992)	$T + 0.5T\,(T - 1)$

Chromosome C (Ho and Tay, 2004)	$T + 0.5T\,(T - 1)$
Chromosome D (Tay and Wibowo, 2004)	$2T + d$
Chromosome E	$n \times \max\{n_j\} \in [T, n(T\text{-}n\text{+}1)]$

(2) Converting mechanism

Once having determined the structure of the targeting chromosome presentation, one can convert the Q-bit chromosome to the targeting chromosome with the following converting algorithm.

First, convert a Q-bit chromosome to a binary row vector ***Bstring***. Observe a state, i.e., if $|\alpha_i|^2 > |\beta_i|^2$, then let *Bstring*[*i*] =1, otherwise, let *Bstring*[*i*] =0.

Second, convert the binary row vector ***Bstring*** to a decimal row vector ***Dstring*** according to the binary to decimal conversion rule. It is notable that this conversion is calculated in an information unit which consists of *q* Q-bit genes.

Third, convert the decimal row vector ***Dstring*** to the targeting chromosome vector ***Ostring***. To keep a higher diversity, a new converting method is presented. Different from Shor (1994), where the job numbers to replace the decimal elements follows a predefined order, we don't predefine the order of the job numbers. Instead, we adopt a random order which provides more chance to generate more diversity among chromosomes. The steps are illustrated as following.

Step 1. Randomly rank all the job numbers and form a set ***randorder***, which has *n* elements.

Step 2. Copy the ***Dstring*** to the ***Ostring***.

Step 3. From the beginning of the ***Ostring***, locate the first *n* minimal genes and replace them with job numbers in ***randorder*** one by one. Then locate the second *n* minimal genes and replace them by ***randorder***, etc, until all the genes are replaced.

To summarize the converting process, we use Fig. 3(a) to illustrate the converting result for the FJSP (Table 2) with the following steps**.**

(1) Encode the 1-Q-bit chromosome,

(2) Convert the 1-Q-bit chromosome to a binary chromosome,

(3) Convert to a decimal chromosome, and

(4) Finally to convert to the targeting chromosome.

In these steps, we assume the random order set of the job numbers ***randorder*** to be [2 1 3]. The elements in the 2nd, 3rd, and 6th are the first minimal 3 genes, so [2 1 3] are set their positions respectively. The elements in the 7th, 1st and 4th are the second minimal 3 genes, so [2 1 3] are set to these positions. The 5th, 8th and 9th are the last minimal 3 genes, set [2 1 3] to the respective positions. As such, we obtain the final targeting chromosome is [1 2 1 3 2 3 2 1 3].

To make a comparison, we also give an example, shown in Fig. 3(b), following Shor (1994). Assume a pre-defined job number order is [1 2 3], hence each position in the first *n* minimal elements are replaced with "1" (see the 2nd, 3rd, and 6th positions), each position in the second *n* minimal elements are replaced with "2" (see the 7th, 1st and 4th positions), and each position in the last *n* minimal elements are replaced with "3" (see the 5th, 8th and 9th positions). Thus, the ***Ostring***=[2 1 1 2 3 1 2 3 3]. Comparing [1 2 1 3 2 3 2 1 3] with [2 1 1 2 3 1 2 3 3], one can see that the entropy of ***Ostring*** in Fig. 3(a) is larger than that in Fig. 3(b). Moreover, different ***randorder*** is generated when we convert each Q-bit chromosome to ***Ostring***. Hence the gene in the same position is different in most cases. When the method (Shor,1994) is used, however, the same order is adopted for each chromosome. This causes a problem that the diversity is weakened.

1-Q-bit chromosome	0.9941 -0.1084	0.3484 -0.9373	-0.2876 -0.9577	-0.9955 0.0943	0.7517 0.6595	-0.5098 -0.8603	-0.6631 0.7486	0.6372 0.7707	-0.9133 -0.4073
Bstring	1	0	0	1	1	0	0	1	1
Dstring	1	0	0	1	1	0	0	1	1
Ostring	1	2	1	3	2	3	2	1	3

(a) A chromosome with ***randorder***

1-Q-bit	0.9941	0.3484	-0.2876	-0.9955	0.7517	-0.5098	-0.6631	0.6372	-0.9133
chromosome	-0.1084	-0.9373	-0.9577	0.0943	0.6595	-0.8603	0.7486	0.7707	-0.4073
Bstring	1	0	0	1	1	0	0	1	1
Dstring	1	0	0	1	1	0	0	1	1
Ostring	2	1	1	2	3	1	2	3	3

(b) A chromosome with a predefined order

Fig. 3. An example of the converting mechanism

3.2.3 Decoding and fitness evaluation

Before evaluation, each chromosome should be decoded to be a scheduling solution. Thus, a scheduling algorithm is needed. We first give two variables.

Machine sequence matrix $\boldsymbol{J_M}$—define the available machines for each operations. Its element $\boldsymbol{J_M}(i, j)$ denotes the available machine for the $(\lceil j/m \rceil + 1)$-th operation of the job J_i (where the symbol ⌈ ⌉ is to get integral part of a float number). Each row $\boldsymbol{J_M}(i, :)$ denotes all the available machines for the operations of job J_i. The length of $\boldsymbol{J_M}(i, :)$ equals $(\max_{j=1}^{n} n_j) \times m$. In each row, from the beginning, every m numbers form a fragment which denotes all the available machines for an operation. Among the machines, if machine M_k is available for an operation, we denote it as M_k; otherwise we denote it 0. In the total FJSP, where each operation can be processed on any of the machines, the fragment equals to '1 2 … m'. As for the partial FJSP, where all machines are not always available, we can use '0' to ensure the length of the fragments. For example, in the FJSP (Table 2), the operation O_{11} can be processed on machine M_1 and machine M_2 only, so the first fragment of $\boldsymbol{J_M}(1, :)$ should be '1 2 0'; the operation O_{21} can be processed on machine M_2 and machine M_3 only, so the second fragment of $\boldsymbol{J_M}(1, :)$ should be '2 3 0'; the operation O_{31} can be processed on all the machines, so the third fragment of $\boldsymbol{J_M}(1, :)$ should be '1 2 3'. Thus, $\boldsymbol{J_M}(1, :)$=[1 2 0 2 3 0 1 2 3]. The number of fragments is determined by the maximum operation number $\max_{j=1}^{n} n_j$. For those jobs whose operation number is less than $\max_{j=1}^{n} n_j$, it is designed to arrange the available machines following the above rule firstly and then set '0' $\left(\max_{j=1}^{n} n_j - n_i\right) \times m$

times to achieve the length $\max_{j=1}^{n} n_j \times m$.

Processing time matrix $\boldsymbol{T}$—define the processing time on an available machine. Its element $\boldsymbol{T}(i, j)$ denotes the processing time on machine $\boldsymbol{J_M}(i, j)$ for the operation $\lceil j/m \rceil + 1$ of the job J_i. If $\boldsymbol{J_M}(i, j)$=0, it means the machine $\mathrm{mod}(j/m)$ (where the symbol "mod" is a function to obtain the reminder of j/m) is not available for the operation $\lceil j/m \rceil + 1$, its processing time is, therefore, set to 0 as well. For example, in the FJSP in Table 2, the operation O_{11} can be processed on machine M_1 and machine M_2 only and occupies 2 and 3 units of time respectively, so the first fragment of $\boldsymbol{T}(1, :)$ should be '2 3 0'.

As such, the machine sequence matrix and the processing time matrix for the FJSP (Table 2) can be defined as follows:

$$\boldsymbol{J_M} = \begin{bmatrix} 1\,2\,0\,2\,3\,0\,1\,2\,3 \\ 2\,3\,0\,1\,2\,3\,1\,2\,3 \\ 1\,3\,0\,1\,2\,3\,1\,2\,3 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 2\,3\,0\,5\,2\,0\,3\,6\,4 \\ 4\,5\,0\,5\,5\,6\,2\,4\,8 \\ 4\,6\,0\,4\,4\,4\,5\,6\,7 \end{bmatrix}.$$

Once $\boldsymbol{J_M}$ and $\boldsymbol{T}$ have been structured, a targeting chromosome can be decoded to a scheduling solution. The decoding algorithm is illustrated in Fig.4. The process is shown as following.

Step 1. Obtain a chromosome *chrom* and set *x*=1;

Step 2. Repeat when *x* is less than the length of the chromosome (=len(*chrom*))

Step 2.1. Obtain the *x*-th gene *chrom*(*x*), which is a job number;

Step 2.2. Determine the operation order *r* (=count(*chrom*(*x*))) according to the appearing times of *chrom*(*x*). Search the available machines for the current operation $O_{r,chrom(x)}$, and select the one on which the operation can be finished at the earliest time *en*(*chrom*(*x*), *r*). Note, ***en*** is a matrix recording the ending time of the operation $O_{r,chrom(x)}$. If there is more than one machine available, select the one on which the processing time $p_{chrom(x),\,r,\,k}$ is the shortest.

Step 2.3: Compare the idle time of machine M_k, if the processing time $p_{chrom(x),\,r,\,k}$ is shorter than the idle time *idletime*(*k*), go to step 2.4; otherwise go to step 2.5;

Step 2.4: Insert the operation $O_{r,chrom(x)}$ and determine its beginning time *st*(*chrom*(*x*),*r*) and ending time *en*(*chrom*(*x*), *r*). Update the available time and

the idle time of machine M_k. Go to step 2.6.

Step 2.5: Append the operation $O_{r,chrom(x)}$ at the end of machine M_k and set its beginning time $st(chrom(x), r)$ to be the ending time of the machine ($mach(k)$). Update its the ending time $en(chrom(x), r)$ and the ending time $mach(k)$ of machine M_k by adding $p_{chrom(x), r,k}$ to $st(chrom(x), r)$. Go to step 2.6.

Step 2.6: Obtain the next gene, set $x=x+1$, and return to the step 2.1.

Step 3: output the scheduling solution.

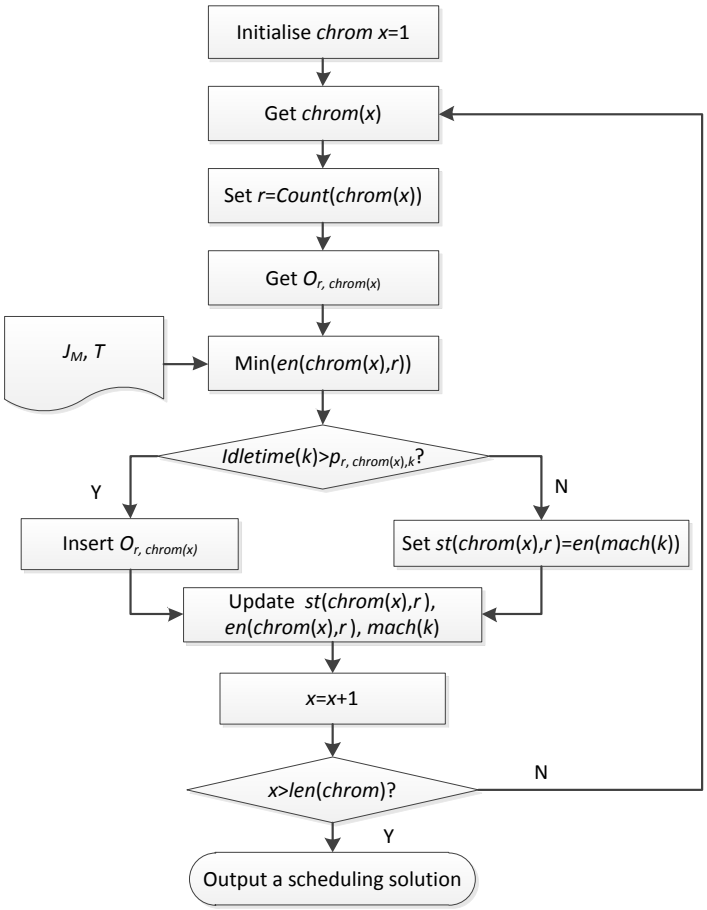


Fig. 4. The decoding algorithm

An operation can be inserted into a machine's idle interval $[t_1, t_2]$ if and only if its current operation processing time is shorter than t_2-t_1. The algorithm solves the machine assignment problem and the operation sequence problem simultaneously. The algorithm outputs a scheduling solution from which we can evaluate the fitness of the targeting chromosome according to the following fitness function.

$$fitness = \frac{1}{\max(C_{ij})} \tag{10}$$

The decoding algorithm proposed in this paper has similar computational complexity to that of the existing methods. The major computational complexity lies in the loop of step 2. This loop needs repeating $n\times\max_{j=1}^{n} n_j$ times. The time on running step 2.1 is $\boldsymbol{O}(1)$. All the available machines for the current operation should be considered. In the worst case scenario, the running time of the step 2.2 is $\boldsymbol{O}(m\times n\times\max_{j=1}^{n} n_j)$. The time on running step 2.3 is $\boldsymbol{O}(1)$. The running time for step 2.4 or step 2.5 is $\boldsymbol{O}(1)$. But only one of them should be executed. The running time on step 2.6 is $\boldsymbol{O}(1)$. Therefore, the time complexity of the decoding algorithm is given by,

$Ta=\boldsymbol{O}((\ n\times\max_{j=1}^{n} n_j)\times(1+m\times\ n\times\max_{j=1}^{n} n_j+1+1+1))$

$=\boldsymbol{O}(mn^2(\max_{j=1}^{n} n_j)^2+4(n\times\max_{j=1}^{n} n_j))$

$=\boldsymbol{O}(mn^2(\max_{j=1}^{n} n_j)^2)$

$=\boldsymbol{O}(T^2)$.

All the time complexity of converting the five types of chromosome representations to scheduling solutions is listed in Table 4 in which the variable c denotes the number of precedence constraints.

Table 4 Conversion complexity of the representations

Chromosome representation	**Conversion complexity**
Chromosome A (Chen et al 1999)	$O(T+c)$
Chromosome B (Paredis 1992)	$O(T^2+c)$
Chromosome C (Ho and Tay 2004)	$O(T+c)$
Chromosome D (Tay and Wibowo 2004)	$O(T+c+d)$
Chromosome E	$O(T^2)$

3.2.4 Updating operation for the Q-bit chromosome

The difference among all the population-based evolutionary algorithms is their population updating mechanism. Updating destroys old individuals and generates offsprings. During the destroy-and-generate process, simply exchanging part of genes of two parents usually generates an infeasible solution. Therefore, the existing crossover operations for FJSP, including partial-mapped crossover (PMX),

order crossover (OX), position-based crossover (PBX), order-based crossover (OBX), cycle crossover (CX), liner order crossover (LOX), subsequence exchange crossover (SXX), partial schedule exchange crossover (PSXX), precedence preservative crossover (PPX) and precedence operation crossover (POX) (Akay B., Yao X.,2013), most need extra computing steps to adjust the infeasible solution to a feasible one. As a result, the computing time is added necessarily, and thus the convergence speed is influenced mostly. Hence, a simple and easy-to-conduct updating operator is vital to an evolutionary algorithm.

The dynamics of the evolution in the EQEA are controlled by the Schrödinger's equation. We choose the rotation gate (RG) to update the Q-bit chromosome. Its form is given by Eq. (11).

$$U(\theta) = \begin{bmatrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{bmatrix} \tag{11}$$

where θ_i is the rotation angle and $\theta_i = s(\alpha_i\beta_i)\Delta\theta_i$. The values of $\Delta\theta_i$ and $s(\alpha_i\beta_i)$ are determined in Table 5, where b_i and x_i are the i^{th} gene of the best chromosome and the current chromosome in the current population, respectively. Each Q-bit gene can be updated according to Eq. (12).

$$\begin{bmatrix} \alpha_i^{t+1} \\ \beta_i^{t+1} \end{bmatrix} = \begin{bmatrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{bmatrix} \begin{bmatrix} \alpha_i^{t} \\ \beta_i^{t} \end{bmatrix} \tag{12}$$

Table 5 The lookup table of θ_i

x_i	b_i	$f(x) \geq f(b)$	$\Delta\theta_i$	$s(\alpha_i\beta_i)$			
				$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	False	0	0	0	0	0
0	0	True	0	0	0	0	0
0	1	False	0	0	0	0	0
0	1	True	0.05π	-1	+1	±1	0
1	0	False	0.01π	-1	+1	±1	0
1	0	True	0.025π	+1	-1	0	±1
1	1	False	0.005π	+1	-1	0	±1
1	1	True	0.025π	+1	-1	0	±1

From (12), one can see that the rotation gate updating mechanism in EQEA only needs one simple step. To make a comparison, Table 6 lists the computing

steps of the existing crossover operators plus the rotation gate. The offspring are all feasible in that the generated quantum-based chromosome still needs to be converted to the target chromosome. Such advantages all benefit from this special quantum chromosome. It is, therefore, confidently to conclude that quantum-based representation helps facilitate problem very well.

Table 6 Computing steps of crossover operators

	PMX	OX	PBX	OBX	CX	LOX	SXX	PSXX	PPX	POX	rotation gate
Steps	3	4	3	3	5	3	2	4	2	3	1

3.2.5 The niche technology

Since the rotation gate operation for each gene is the same, it is, however, easy to trap into the local optima and lack of diversity of genes when the population evolves more than certain times. Therefore, we integrate the niche technology.

The niche technology is proposed by Hyun et al. (1998). It can avoid trapping into local optima when there are too many similar individuals in the population. A niche domain is the space whose sizes are determined by Eq. (13).

$$\sigma_t = \frac{\max t - \min t}{Ps} \tag{13}$$

where max t and min t are the maximal and the minimal of the objective during the t^{th} generation, respectively, and *Ps* is the population size. The more chromosomes there are in the space, the more similar they are. Therefore, the structure of the niche has a direct impact on the quality of the diversity. We count the total number of the similar chromosomes in the range σ_t for each chromosome. The one which has the most similar chromosome is focused. Half of its similar chromosomes are replaced by generating randomly new chromosomes to increase the diversity of the population.

3.2.6 The elitist selection

To speed up the convergence, the elitist strategy is adopted to prevent the loss of the best chromosome during the evolutions. The best chromosome with highest fitness

individual will be identified and recorded. If the best chromosome is lost or becomes weaker after evolution, it will be inserted back into the evolving population. The integration of the elitist strategy with the QEA speeds up the convergence and reduces the influence of the random factors during the evolutionary process.

3.2.7 A local search based on the atom structure energy distribution

It is widely accepted that a local search procedure is efficient in improving the solutions generated by QEA (Zheng, T., Yamashiro, M., 2010). Inspired by the mechanism of the motion of the electrons around an atomic nucleus, we design a local search to enhance the local exploitation around the best solution.

Scientists have discovered that an atom consists of a nucleus and electrons. For example, Fig.5 shows the atom structure of natrium, which comprises 11 protons in the nucleus, 2 electrons in the first orbit, 8 electrons in the second orbit and 1 electron in the outermost orbit. The energy of an electron depends on the position of the orbit and is lower in smaller orbits. The atoms are stable in the state with the smallest orbit in that there is no orbit of lower energy into which the electron can jump. The closer to the atomic nucleus an electron is, the lower energy it has. That is, those electrons located in the larger orbits have larger energy and are less stable.

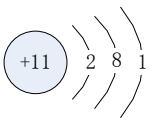


Fig. 5. The atom structure of natrium

Inspired by this phenomenon, we regard a solution for the FJSP as an atom, each operation corresponds to an electron, and each electron orbit corresponds to an energy level. To improve the scheduling solution, the operations compete to jump from a higher energy level to a lower one. We define the energy level of a gene according to its position in the targeting chromosome. The more frontal the gene lies in the chromosome, the lower energy it has. A more stable solution may therefore be obtained.

The local search process is designed by letting the critical operation jump from the position with a higher level energy to another position with a lower level energy. We call it

an energy-jumping process. This process is executed only on the best individual. According to the decoding algorithm (Fig. 4), moving the last gene forward in the critical path gives it more chance to select better machines so that the whole energy of the new solution decreases. If the result becomes better after the local search, we replace the best individual with this new individual. If the result is no better than the original best individual, we keep the original best individual. Based on the jumping extent, a shallow energy-jumping algorithm, a deep energy-jumping algorithm, and a moderate-jumping algorithm can be developed. The common point among the three algorithms is to move the operations in the critical path. The distinction among them is how to and when to jump the operations.

The shallow energy-jumping algorithm is to move the last operation in the critical path to the next position of its previous operation. A chromosome for the FJSP (Table 2) is shown in Fig. 6. The last gene '3' jumps to the fifth position and the gene '2' in the fifth position moves to the last position. After the jumping is completed, the makespan is shortened from 16 to 15 so that the solution quality is improved.

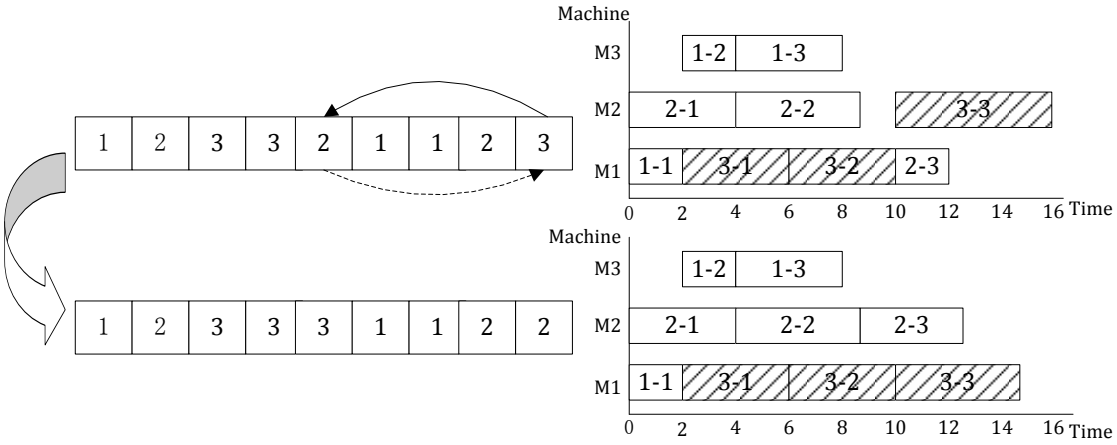


Fig. 6. The shallow energy-jumping algorithm

The deep energy-jumping algorithm is to move all the operations in the critical path to the frontal of the chromosome. A chromosome for the FJSP (Table 2) is shown in Fig. 7. The job number in the critical is '3', therefore all the '3's move forward and the replaced genes moves back to the original position of the gene '3' in turn. After the jumping is finished, the makespan is shortened from 15 to 14 so that the solution quality is improved.

In some cases, there are multiple jobs in the critical path. The job with the longest total processing time (we name it the most troublesome job) has the first chance to move to the front of the chromosome. Next, the job with the second longest total processing time moves behind the end of the most troublesome job, etc.

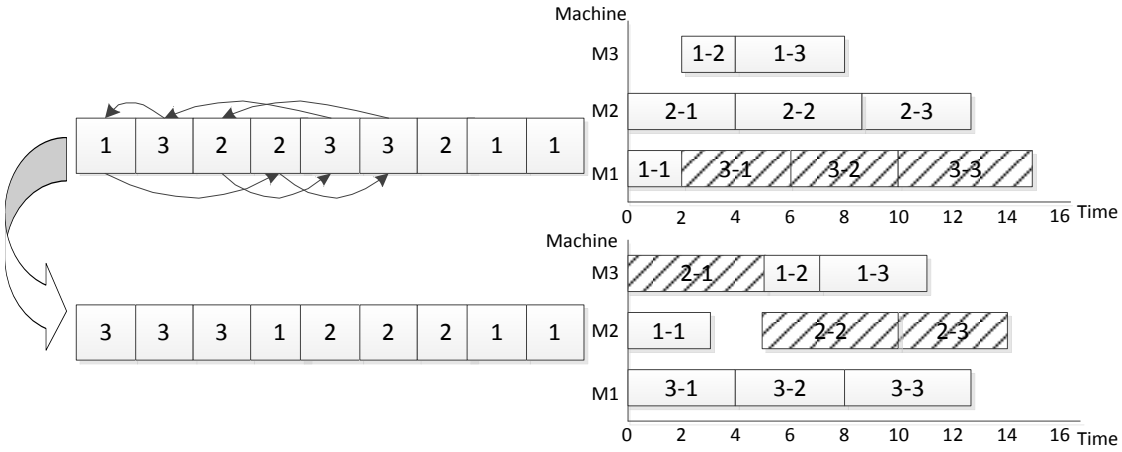


Fig.7. The deep energy-jumping algorithm

The moving strategy in the moderate energy-jumping algorithm is between that in the shallow energy-jumping algorithm and that in the deep energy-jumping algorithm. If the last gene in the critical path is just next to its previous operation, then we execute the deep energy-jumping algorithm. Otherwise, we execute the shallow energy-jumping algorithm.

4. Evaluation

4.1 Design of experiment

To compare our approach with the existing methods (Hmida et al. 2010, Pezzella et al. 2008, Wang et al. 2012b, Gao et al. 2008, Mastrolilli and Gambardella 2000, Li et al. 2012, Chiang and Lin 2013), we conduct five groups experiments:

(1) Setting parameters,

(2) Comparing three local search methods,

(3) Comparing the success rates,

(4) Comparing the convergence speed, and

(5) Comparing the two technologies for diversity.

All the experiments were conducted in a desktop computer with a Pentium dual-core CPU E6600 3.0-GHz CPU, 2.0G RAM, WIN-XP OS, and Matlab©. Four groups of benchmark instances (Table 6) are adopted. Columns 1, 2, 3 and 4 show the names of the instances, the ranges of the job numbers, the ranges of the machine numbers, and the ranges of the operations numbers for all jobs, respectively.

Table 6 Benchmark instances and their settings

instance name	No. of jobs	No. of machines	No. of operations
Kacem_Data (Kacem et al., 2002)	4--15	5--15	12--56
BR_data (Brandimarte, 1993)	10--20	4--15	55--240
BC_data (Barnes et al., 1996)	10--15	11--18	100--225
DP_data (Dauzère-Pérès et al., 1997)	4--15	5--10	196--387

4.2 Computaional Results

4.2.1 Setting parameters

Parameters influence an algorithm's performance. Four parameters are included:

(1) population size,

(2) iteration times,

(3) Q-bit number q in an information unit, and

(4) niche size Ns.

When more chromosomes are generated to form a population, more solutions are provided so that the probability to find the optimal solution within fewer steps increases. But this will take more computing time and space. We therefore set a medium population size, i.e. 50. Similarly, the number of iterations is proportional to the searching result. We set it as 200.

To determine the values of the Q-bit number and the niche size, 2 groups of experiments are conducted on MK03 instance (Brandimarte, 1993). 4 levels, q=1,2,3,4, of the Q-bit numbers are considered. The niche size has 4 levels: Ns=0.5d, d, 1.5d, 2d, where

$$d = \frac{\max(makespan) - \min(makespan)}{Ps/2}.$$

We fix one parameter value and change another parameter value from the first level to the fourth level. Each test runs 10 times. Fig. 8 presents the results influenced by the different levels of the niche size when q=3. Results show that Ns=d can output the best solutions. Fig. 9 presents the results influenced by the different levels of the Q-bit numbers when Ns=d. Results indicate that q=3 and q=4 generate the same optimal value. For saving computing resource, we choose q=3.

In all, all of the parameters are set as: the population size: 50, the iteration times: 200, the Q-bit number: 3, and the niche size: d.

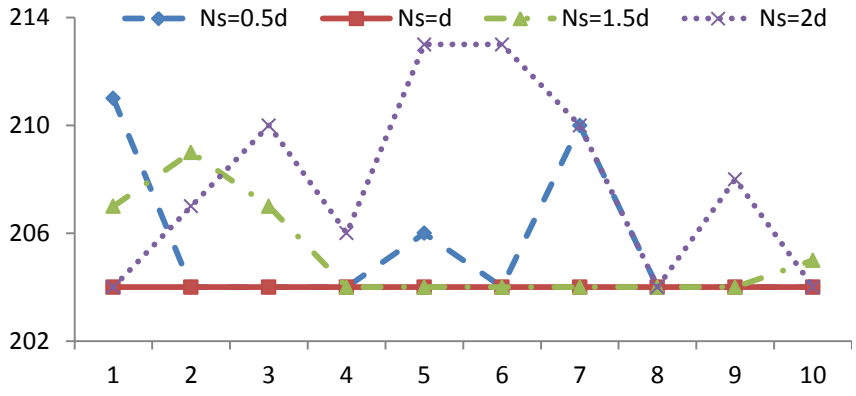


Fig.8. Results influenced by *Ns*

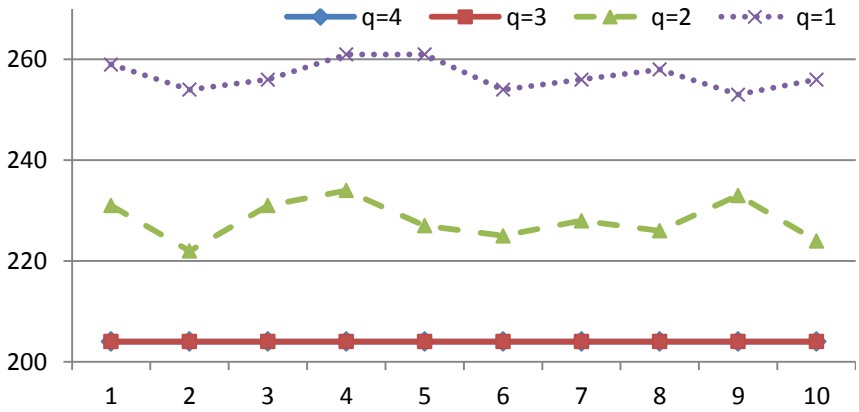


Fig.9. Results influenced by q

4.2.2 Comparing the three local search algorithms

To evaluate the three local search algorithms' influence on the search process, two groups of experiments are conducted on the MK03 instance (Brandimarte, 1993): EQEA with or without local search. In the first group of EQEA, the three local search algorithms

are also compared furtherly. Each test runs 10 times to obtain an average result. Table 7 lists the iteration times that the best result (makespan=204) appears by each local search algorithm, and the total CPU computing time that the 200 iterations costs totally in seconds. The average values are listed in the last row.

Table 7 Comparison result for the local search algorithms

No.	EQEA with local search						EQEA without local search	
	Shallow		Moderate		Deep			
	iteration	time	iteration	time	iteration	time	iterations	time
1	2	63.61	7	69.63	37	68.21	17	62.22
2	1	63.33	25	68.64	13	67.32	47	63.98
3	2	63.68	18	63.35	38	69.19	8	63.06
4	6	64.49	1	64.47	20	63.63	53	62.77
5	1	64.16	5	64.49	5	63.84	21	62.99
6	19	64.08	10	64.04	5	63.49	45	63.01
7	9	67.51	19	64.26	44	63.47	7	63.43
8	32	63.22	4	65.19	7	64.02	5	63.56
9	9	63.87	9	64.46	9	63.98	13	62.98
10	2	66.38	2	62.81	26	64.64	4	62.82
Mean	8.3	64.43	10	65.13	20.4	65.17	22	63.082

Results (Table 7) indicate that even the worst result (from deep energy-jumping algorithm) in the first group outperforms that in the second group. Furtherly, in the first group, the shallow energy-jumping algorithm performs best (which only needs 8.3 iterations averagely), followed by the moderate energy-jumping algorithm (which needs 10 iterations averagely), and the deep energy-jumping algorithm (which needs 20.4 iterations averagely). Since the local search process is only executed on the current best solution, the shallow energy-jumping algorithm only exploits its neighborhood, which can ensure most excellence genes can be preserved in the population. From this perspective, the deep

energy-jumping algorithm destroys most of the genes in the best chromosome so that it takes more time to achieve to the best solution. However, it can help to explore a new area and help protect the search process from trapping to prematurity. Of course, it may guide the searching process to a worse area. To avoid deterioration, we compare the solutions before and after the local search. If the deterioration appears, we cancel the local search result. Thus, the deep energy-jumping algorithm is an effective method to avoid the prematurity. The moderate energy-jumping algorithm performs slightly inferior to the shallow energy-jumping algorithm, which implies that the probability that the last gene in the critical path is located close to its previous operation is very low.

Based on the above analysis, we adopt the moderate energy-jumping algorithm. We dynamically check the frequency of the current best solutions. If it exceeds a given threshold (20 iterations, e.g.), we call the deep energy-jumping algorithm. This strategy can balance the exploration ability and the exploitation ability of the search process.

4.2.3 Comparing the success rates

We repeatedly run the EQEA for 10 times on each of the benchmark instances and report the best result. The results from some of the existing research are compared with our algorithm.

Table 8 illustrates the comparison of the success rates between the EQEA and those in the literatures (Wang et al. 2012b, Li et al. 2012, Chiang et al. 2013) on the Kacem_data instances(Kacem et al., 2002). The first column reports the instance name. The second column is the problem size: the number of jobs×the number of machines. Columns 3-5 report the results from other studies. Column 6 reports our best makespan over 10 runs of EQEA. The success rate is the ratio of the number of those instances whose optimal solutions can be sought by an algorithm to the total number of the instances. It can be seen that all algorithms obtain the optimal solution with a 100% success rate.

Table 8 Results for the Kacem_data instances

Instance	Job×machine	Wang et al., 2012b	Li et al., 2012	Chiang et al., 2013	EQEA
Kacem 1	4×5	**11**	**11**	**11**	**11**
Kacem 2	8×8	**14**	**14**	**14**	**14**

Kacem 3	10×7	**11**	**11**	**11**	**11**
Kacem 4	10×10	**7**	**7**	**7**	**7**
Kacem 5	15×10	**11**	**11**	**11**	**11**
Success rate		100%	100%	100%	100%

Wang et al. 2012b: implemented in C++ on a 3.2 GHz Intel Core i5 processor
Li et al. 2012: implemented in C++ on a Pentium IV 1.8 GHz with 512 M memory.
Chiang et al. 2013: not given

Table 9 reports the results of the BR_data instances (Brandimarte, 1993). Results from Hmida et al.(2010), Pezzella et al.(2008), Wang et al. (2012), and Gao et al.(2008) are compared with the EQEA, respectively. The table has a similar structure as Table 8, except that the third column reports the lower bound and the upper bound. It can be found that the EQEA has a 90% success rate, whereas the algorithms shown in Hmida et al.(2010), Pezzella et al.(2008), Wang et al. (2012), and Gao et al.(2008) achieved success rates of 90%, 60%, 90%, 90% and 90%, respectively.

Table 9 Results for the BR_data 10 instances

Instance	**Job× machine**	**(LB, UB)**	Hmida et al.,2010	Pezzella et al.,2008	Wang et al., 2012b	Gao et al.,2008	**EQEA**
MK01	10×6	(36,42)	**40**	**40**	**40**	**40**	**40**
MK02	10×6	(24,32)	**26**	**26**	**26**	**26**	**26**
MK03	15×8	(204,211)	**204**	**204**	**204**	**204**	**204**
MK04	15×8	(48,81)	**60**	**60**	**60**	**60**	**60**
MK05	15×4	(168,186)	173	173	**172**	**172**	**172**
MK06	10×15	(33,86)	**58**	63	**58**	**58**	**58**
MK07	20×5	(133,157)	**139**	**139**	**139**	**139**	**139**
MK08	20×10	(523)	**523**	**523**	**523**	**523**	**523**
MK09	20×10	(299,369)	**307**	311	**307**	**307**	**307**
MK10	20×15	(165,296)	**197**	212	198	212	212
Success rate			90%	60%	90%	90%	90%

Hmida et al.2010: implemented in C on an Intel Core 2 Duo 2.9GHz Personal Computer with 2GB of RAM.
Pezzella F et al.2008: implemented on a 1.8MHz Pentium IV processor
Wang et al. 2012b: implemented in C++ on a 3.2 GHz Intel Core i5 processor
Gao et al.2008: implemented in Delphi on a 3.0GHz Pentium.

Table 10 reports the results of the BC_data instances (Barnes et al., 1996). Results of Hmida et al.(2010), Gao et al.(2008), and Mastrolilli et al.(2000) are compared with the EQEA, respectively. Results show that the EQEA has a 86% success rate, whereas the

algorithms shown in Hmida et al. (2010), Gao et al. (2008), and Mastrolilli et al.(2000) achieved success rates of 81%, 62% and 86%, respectively.

Table 10 Results for the BC_data 22 instances

Instance	Job×machine	(LB, UB)	Hmida et al.,2010	Gao et al.,2008	Mastrolilli et al.,2000	EQEA
mt10x	10×11	(655,929)	**918**	**918**	**918**	**918**
mt10xx	10×12	(655,929)	**918**	**918**	**918**	**918**
mt10xxx	10×13	(655,936)	**918**	**918**	**918**	**918**
mt10xy	10×11	(655,913)	906	**905**	906	**905**
mt10xyz	10×12	(655,849)	849	849	**847**	849
mt10c1	10×11	(655,927)	928	**927**	928	928
mt10cc	10×12	(655,914)	**910**	**910**	**910**	**910**
setb4x	15×11	(846,937)	**925**	**925**	**925**	**925**
setb4xx	15×12	(846,930)	**925**	**925**	**925**	**925**
setb4xxx	15×13	(846,925)	**925**	**925**	**925**	**925**
setb4xy	15×12	(846,924)	**916**	**916**	**916**	**916**
setb4xyz	15×13	(838,914)	**905**	**905**	**905**	**905**
setb4c9	15×11	(857,924)	919	**914**	919	919
setb4cc	15×12	(857,909)	**909**	914	**909**	**909**
seti5x	15×16	(955,1218)	**1201**	1204	**1201**	**1201**
seti5xx	15×17	(955,1204)	**1199**	1202	**1199**	**1199**
seti5xxx	15×18	(955,1213)	**1197**	1204	**1197**	**1197**
seti5xy	15×17	(955,1148)	**1136**	**1136**	**1136**	**1136**
seti5xyz	15×18	(955,1127)	**1125**	1126	**1125**	**1125**
seti5c12	15×16	(1027,1185)	**1174**	1175	**1174**	**1174**
seti5cc	15×17	(955,1136)	**1136**	1138	**1136**	**1136**
Success rate			81%	62%	86%	86%

Hmida et al.2010: implemented in C on an Intel Core 2 Duo 2.9GHz Personal Computer with 2GB of RAM.

Gao et al.2008: implemented in Delphi on a 3.0GHz Pentium.

Mastrolilli et al.2000: implemented in C++ on a 266 Pentium.

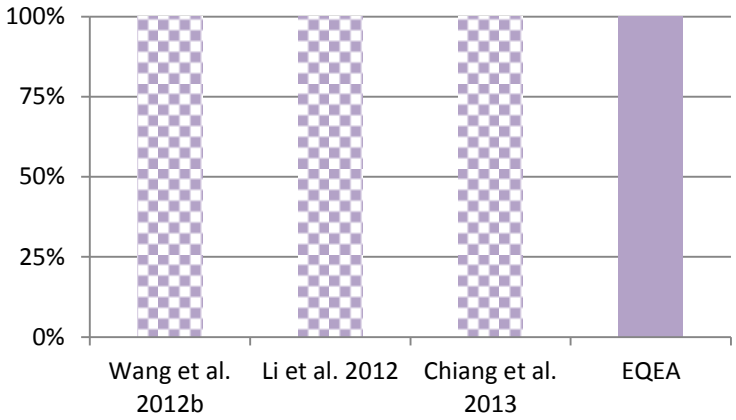
Table 11 reports the results of the DP_data instances (Dauzère-Pérès et al., 1997). The results from Hmida et al.(2010), Gao et al.(2008), and Mastrolilli et al.(2000) are compared with the EQEA, respectively. The EQEA achieves a success rate of 83%, whereas the algorithms of Hmida et al. (2010), Gao et al. (2008) and Mastrolilli et al. (2000) had 89%, 61% and 67.7%, respectively.

Table 11 Results for the DP_data 18 instances

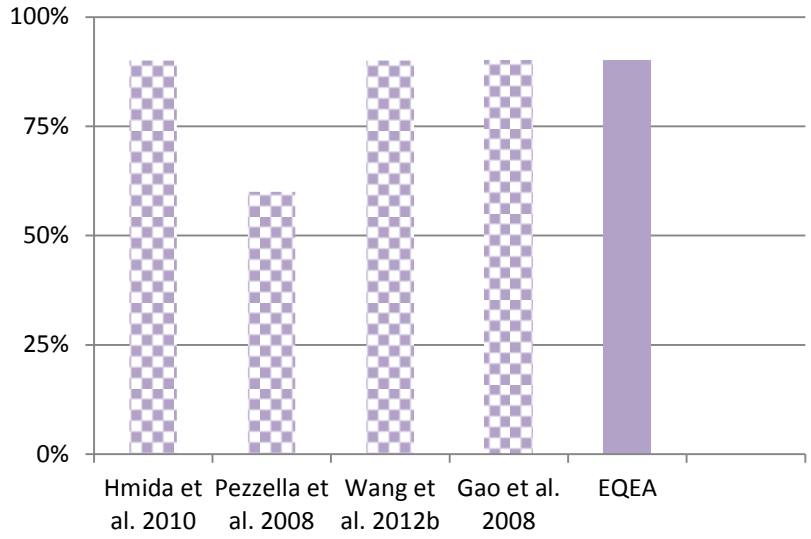
Instance	Job× machine	(LB, UB)	Hmida et al.,2010	Gao et al.,2008	Mastrolilli et al.,2000	EQEA

01a	10×5	(2505,2530)	**2518**	**2518**	**2518**	**2518**
02a		(2228,2244)	**2231**	**2231**	**2231**	**2231**
03a		(2228,2235)	**2229**	**2229**	**2229**	**2229**
04a		(2503, 2565)	**2503**	2515	**2503**	**2503**
05a		(2189, 2229)	**2216**	2217	**2216**	**2216**
06a		(2162, 2216)	**2196**	**2196**	2203	**2196**
07a	15×8	(2187, 2408)	**2283**	2307	**2283**	2305
08a		(2061, 2093)	**2069**	2073	**2069**	**2069**
09a		(2061, 2074)	**2066**	**2066**	**2066**	**2066**
10a		(2178, 2362)	**2291**	2315	**2291**	**2291**
11a		(2017, 2078)	**2063**	2071	**2063**	2065
12a		(1969, 2047)	2031	**2030**	2034	2031
13a	20×10	(2161, 2302)	**2257**	**2257**	2260	**2257**
14a		(2161, 2183)	**2167**	**2167**	**2167**	**2167**
15a		(2161, 2171)	**2165**	**2165**	2167	**2165**
16a		(2148, 2301)	2256	2256	**2255**	**2255**
17a		(2088, 2169)	**2140**	**2140**	2141	**2140**
18a		(2057, 2139)	**2127**	**2127**	2137	**2127**
Success rate			89%	61%	67.7%	83%

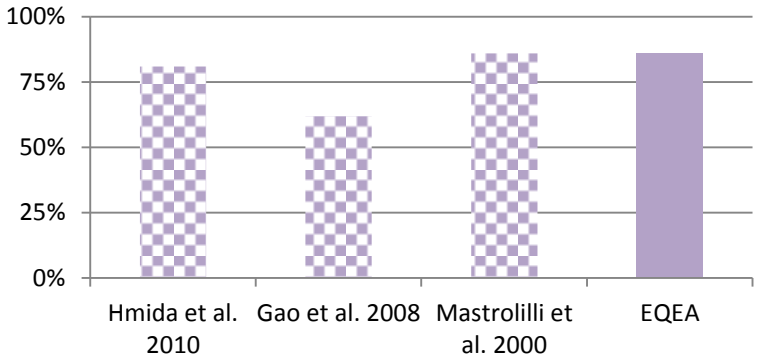
To conclude, results of the tests on the Kacem_data (Kacem et al., 2002) prove that the perfect convergence speed of the EQEA, On the other hand, the results of the tests on the BR_data (Brandimarte, 1993), BC_data (Barnes et al., 1996) and the DP_data (Dauzère-Pérès et al., 1997) show good exploration and exploitation ability of the EQEA. Fig.10 includes four figures comparing the success rates on the four group benchmark instances. Those comparisons show the outstanding performance of the EQEA.



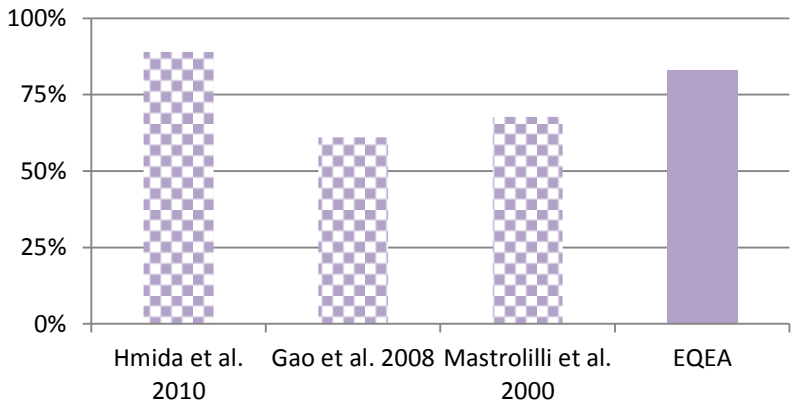
(a) Success rates on Kacem_Data



(b) Success rates on BR_data



(c) Success rates on BC_data



(d) Success rates on DP_data

Fig. 10. Comparison of success rates on different benchmark instances

4.2.4 Comparing the convergence speed

Table 12 compares the CPU computing time in seconds based on the Kacem_data (Kacem et al., 2002) with the EQEA and the one developed by Wang et al. (2012). The results show that the EQEA outperforms the algorithm developed by Wang et al. (2012).

Table 12 The comparison of computing time (s)

Instance	Job×machine	Wang et al., 2012b	EQEA
Kacem1	4×5	0.01	0.008
Kacem2	8×8	0.23	0.122
Kacem3	10×7	0.3	0.071
Kacem4	10×10	0.42	0.347
Kacem5	15×10	14.88	9.694

4.2.5 Comparing the two technologies for diversity

From the comparison of the success rate (Section 4.2.3) and the convergence speed (Section 4.2.4), it can be concluded that the proposed EQEA yields good performance. Compared with other algorithms, the EQEA adopts two technologies---the niche technology, and an energy-jumping local search---to increase diversity of the solutions. Another interesting question is which component makes a better contribution to the performance. To cater to such an interest, we conduct some more experiments to analyze the impact of these components.

The first experiment is to analyze the impact of the niche technology. We delete the niche technology from the EQEA and run this new EQEA 10 times. The iterations that the optimal solution (makespan=204) appears and the total computing time (200 iterations) are listed in columns 2 and 3 in Table 13. The second experiment is to analyze the impact of the local search. Similarly, we delete the local search from the EQEA and run this new EQEA 10 times. The iterations that the optimal solution (makespan=204) appears and the total computing time (200 iterations) are listed in columns 4 and 5. The results of EQEA (makespan=204) are listed in columns 6 and 7 to make a comparison. Their average value is listed in the last row. One can see that it takes 36.6 iterations and 22 iterations on average, respectively, when there is no niche technology or local search. Hence, it can be concluded that the niche technique influences the convergence speed of EQEA more than the local

search algorithm. This is due to the extent to be affected by these two technologies. The niche technology is executed on the whole population and the local search is only executed on the local best individual.

Table 13. Comparison of the two technologies for population diversity

No.	No niche		No local search		EQEA	
	iterations	time	iterations	time	iterations	time
1	4	73.74	17	62.22	7	69.63
2	18	65.83	47	63.98	25	68.64
3	13	65.17	8	63.06	18	63.35
4	104	65.04	53	62.77	1	64.47
5	14	65.02	21	62.99	5	64.49
6	6	64.64	45	63.01	10	64.04
7	127	64.71	7	63.43	19	64.26
8	53	64.65	5	63.56	4	65.19
9	17	65.36	13	62.98	9	64.46
10	10	65.44	4	62.82	2	62.81
average	36.6	65.96	22	63.082	10	65.134

5. Conclusions

In this study, a new fast algorithm that integrates the quantum-inspired evolutionary algorithm with the elitist strategy was developed to solve the FJSP. Two novel methods were proposed to increase the diversity of the population: one is the niche technology conducted on the whole population aiming to reduce the similarity among individuals, and the other is a new local search technology, inspired by the motion mechanism of the electrons around an atomic nucleus, conducted on the elitist individual. The local search comprises three energy-jumping algorithms aiming at exploiting a better neighbor solution. The performance of the proposed approach was assessed on well- known benchmarks. The results show the proposed approach can solve the FJSP more efficiently and effectively than those compared in this paper.

In addition, to the best of our knowledge, this is the first reported application of the quantum-inspired evolutionary algorithm to solve FJSP. In the future, it will be interesting to investigate on the following issues:

1) to improve the local search process;
2) to develop multi-objective EQEA to solve the FJSP with multi-objectives; and
3) to explore more practical constraints such as the random breakdown or the preventive maintenance activities.

Acknowledgment

This paper is partially supported by the National Natural Science Foundation of China under Grant (Grant No.51305024) and Fundamental Research Funds for the Central Universities (Grant No. FRF-TP-14-031A2). We greatly acknowledge the two anonymous reviewers and Professor Xin Yao from University of Birmingham, UK for their suggestions to improve the paper.

References

Akay B., Yao X.(2013). Recent Advances in Evolutionary Algorithms for Job Shop Scheduling. Automated Scheduling and Planning Studies in Computational Intelligence. Springer Berlin Heidelberg.505: 191-224

Barnes, J.W., Chambers, J.B. (1996). Flexible job shop scheduling by tabu search. Graduate Program in Operations Research and Industrial Engineering, the University of Texas at Austin, Technical Report Series.

Baykasoglu, A., Ozbakir, L., & Sönmez, A. I. (2004). Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, *15*(6), 777–78

Brandimarte, P.(1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operation Research*, 41:157-83.

Brucker, P., Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45:369-75.

Chiang, T.C., Lin, H.J. (2013). A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling, *International Journal of Production Economy*, 141:87-98.

Chen, H., Ihlow, J., Lehmann, C. (1999). A genetic algorithm for flexible job-shop scheduling, Proceedings of IEEE International Conference on Robotics and Automation 2: 1120–1125.

Chen, J.C., Wu, C., Chen, C., Chen, K. (2012). Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm. *Expert Systems with Applications*, 39: 10016-10021.

Dauzère-Pérès, S., Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operation Research*,70:281-306.

Demir, Y., Kürşat Işleyen, S. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematics Modelling*, 37:977-88.

Gao, J., Sun, L., Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research*, 35:2892-907.

Gen, M., Lin L., Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. Journal of Intelligent Manufacturing. 2014,25(5): 849-866

Gen, M., Tsujimura, Y., Kubota, E. (1994). Solving job-shop scheduling problems by genetic algorithm. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2:1577-82.

Hmida, A.B., Haouari, M., Huguet, M.J., Lopez, P. (2010). Discrepancy search for the flexible job shop scheduling problem. *Computers and Operations Research*, 37: 2192–2201.

Ho, N.B., Tay, J.C. (2004). GENACE: An efficient cultural algorithm for solving the flexible job-shop problem. Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004.2:1759-66.

Hyun, C.J., Kim, Y., Kim, Y.K. (1998). A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers and Operations Research*, 25:675-90.

Jia, H.Z., Nee, A.Y.C., Fuh, J.Y.H., & Zhang Y.F. (2003). A modified genetic algorithm for distributed scheduling problems. *International Journal of Intelligent Manufacturing.* 14: 351–62.

Kacem, I., Hammadi, S., Borne, P. (2002). Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems Man and Cybernetics C*, 32:408-19.

Li, J., Pan, Q. (2012). Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity. *Applied Soft Computing* ,12:2896-912.

Li, J.Q., Pan, Q.K., Gao, K.Z. (2012). Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems, *International Journal of Advanced Manufacturing Technology*,55:1159-1169.

Loukil, T., Teghem, J., Tuyttens, D. (2005). Solving multi-objective production scheduling problems using metaheuristics. *European Journal of Operational Research*, 161:42-61.

Lu, T.C., Yu, G.R. (2013). An adaptive population multi-objective quantum-inspired evolutionary algorithm for multi-objective 0/1 knapsack problems. *Inform Sciences*, 243:39–56.

Mati, Y., Rezg, N., Xie, X. (2001). An integrated greedy heuristic for a flexible job shop scheduling problem. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 4:2534-9.

Mastrolilli, M., Gambardella, L.M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling,* 3:3-20.

Nagamani, M., Chandrasekaran, E., Saravanan, D. (2012). Single Objective Evolutionary Algorithm for Flexible Job-shop Scheduling Problem. International Journal of Mathematics Trends and Technology, 3(2):78-81

Paredis, J. (1992). Exploiting constraints as background knowledge for genetic algorithms: A case study for scheduling. *Parallel Problem Solving from Nature: PPSN I*I:281-90.

Pezzella, F., Morganti, G., Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers and Operations Research*, 35:3202-12.

Shor, P.W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. Proceedings of the 35th Annual Symposium on the Foundations of Computer Science:124-134.

Tay, J.C., Wibowo, D. (2004). An effective chromosome representation for evolving flexible job shop schedules. *Lecture Notes in Computer Science*, 3103:210-21.

Wang, L.X., Kowk ,S. K., Ip, W. H. (2012a). Design of an improved quantum-inspired evolutionary algorithm for a transportation problem in logistics systems. *Journal of Intelligent Manufacturing*, 23:2227–2236.

Wang, L., Wang, S., Xu, Y., Zhou, G., Liu, M. (2012b). A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 62:917-26.

Xia, W., Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & industrial engineering* ,48:409-25.

Xing, L.N., Chen, Y.W., Wang, P., Zhao, Q.S., Xiong, J. (2010). A knowledge-basedant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*,10: 888-896.

Xing, L. N., Chen, Y. W., & Yang, K. W. (2009). An efficient search method for multi-objective flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, *20*(3), 283–293.

Yazdani, M., Amiri, M., Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37(1): 678-687.

Zheng, T., Yamashiro, M. (2010). Solving flow shop scheduling problems by quantum differential evolutionary algorithm. *International Journal of Advanced Manufacturing Technology*, 49:643 – 662.