Computers & Operations Research 39 (2012) 2291–2299

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A new hybrid genetic algorithm for job shop scheduling problem

Ren Qing-dao-er-ji [a], Yuping Wang [b,*]

[a] School of Science, Xidian University, Xi'an 710071, China
[b] School of Computer Science and Technology, Xidian University, Xi'an 710071, China

ARTICLE INFO

Available online 17 December 2011

Keywords:
Genetic algorithm
Job shop scheduling problem
Crossover operator
Mutation operator
Local search

ABSTRACT

Job shop scheduling problem is a typical NP-hard problem. To solve the job shop scheduling problem more effectively, some genetic operators were designed in this paper. In order to increase the diversity of the population, a mixed selection operator based on the fitness value and the concentration value was given. To make full use of the characteristics of the problem itself, new crossover operator based on the machine and mutation operator based on the critical path were specifically designed. To find the critical path, a new algorithm to find the critical path from schedule was presented. Furthermore, a local search operator was designed, which can improve the local search ability of GA greatly. Based on all these, a hybrid genetic algorithm was proposed and its convergence was proved. The computer simulations were made on a set of benchmark problems and the results demonstrated the effectiveness of the proposed algorithm.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

Scheduling is one of the most critical issues in the planning and managing of manufacturing processes. The difficulty of finding the optimal schedule depends on the shop environment, the process constraints and the performance indicator. One of the most difficult problems in this area is the job shop scheduling problem (JSSP), which has been proved to be an NP-complete problem [1]. Since the benchmarks of JSSP were presented by Fisher and Thomson in 1963 [2], JSSP has been studied by a great number of researchers, and many exact methods and approximation algorithms have been proposed [3–7]. Exact methods, such as branch and bound, linear programming and Lagrangian relaxation guarantee global convergence and have been successful in solving small instances. However, they require a very high computing time as the size of problem increases. So a lot of researchers paid their attention to meta-heuristic and intelligent hybrid search optimization strategies for solving JSSP. Among these, the shifting bottleneck approach, particle swarm optimization, ant colony optimization, simulated annealing, Tabu search, genetic algorithm, neural network, immune algorithm are the typical examples. These intelligent optimization algorithms are relatively easy to implement and could conveniently be adapted to different kinds of scheduling problems, and especially, it has been proven by experiments that they can find high-quality solutions within reasonable computational time. These have made the research on intelligent hybrid algorithm for JSSP increasingly popular in the recent years.

Genetic algorithms were proposed by Holland [8] and have been successfully used in a variety of practical problems. Davis [9] first applied a genetic algorithm to the JSSP in 1985 successfully, and now genetic algorithms have been proved to be an effective approach for the JSSP. There are many such works, e.g., a genetic algorithm with search area adaptation was proposed by Someya and Yamamura [10]. Zhou and Feng [11] proposed a hybrid heuristics GA for JSSP, where the scheduling rules, such as shortest processing time (SPT) and most work remaining (MWKR), were integrated into the process of genetic evolution. Park et al. developed an efficient method based on genetic algorithm to address JSSP. The scheduling method based on parallel genetic algorithm was designed in [12]. Mattfeld and Bierwirth [13] considered a multi-objective job shop scheduling problem with release and due dates, as well as tardiness as objectives. Watanabe et al. [14] proposed a modified genetic algorithm with search area adaptation for solving the job shop scheduling problem. Li presented a two-row chromosome structured new genetic algorithm based on working procedure and machine distribution [15]. The relevant crossover and mutation operations were also designed in [15].

However, the existing genetic algorithms for the JSSP are usually with a slow convergence speed and easy to trap into local optimal solutions. In order to enhance the convergence speed, many researchers focused their attention on combining the genetic algorithm with local search schemes to develop some hybrid optimization strategies for JSSP. Wang and Zheng [16] developed a hybrid optimization strategy for JSSP by combining the simulated annealing algorithm and the genetic algorithm.

* Corresponding author.
E-mail addresses: renqingln@sina.com (R. Qing-dao-er-ji), ywang@xidian.edu.cn (Y. Wang).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.12.005

Gonçalves et al. [17] presented a hybrid genetic algorithm for the JSSP, in which the schedules were constructed by using a priority rule and the priorities were defined by the genetic algorithm, and then a local search heuristic was applied to improve the solution. Zhang proposed a genetic simulated algorithm to solve the JSSP by combining the GA and SA [18]. Liu et al. [19] combined the traditional genetic algorithm with the Taguchi method and the proposed algorithm possessed the merits of global exploration and robustness. Xu proposed an immune genetic algorithm by combining the immune theory and the genetic algorithm [20]. Vilcot proposed a fast and elitist genetic algorithm based on NSGA-II for solving the multi-objective SSSP [21]. Zhang combined the TS algorithm and the genetic algorithm to develop a hybrid algorithm for JSSP [22]. Zhang proposed a hybrid simulated annealing algorithm based on a novel immune mechanism for the JSSP [23].

However, there are the following shortcomings for the aforementioned algorithms: (1) these algorithms have not taken into account the diversity of population. This can easily lead to the "premature" convergence because of the population being easily filled with more similar individuals. (2) Most of the crossover operators and mutation operators have not made use of the characteristics of the JSSP structure itself. Many researchers designed these operators according to the structure of code and only changed the form of code. Thus, these operators usually cannot guide the search to move to the better solutions, and are also difficult to integrate the merit of the parent individuals. (3) The local search ability is not satisfactory.

In order to overcome these drawbacks of the algorithms, in this paper, a new crossover operator and mutation operator were designed by sufficiently making use of the information and structure of the JSSP, and these operators are efficient for JSSP. Moreover, in order to enhance the diversity of the population and avoid trapping into the local optimal solutions, a specifically designed mixed selection scheme is presented. Furthermore, to improve the speed of the algorithm, an efficient local search approach is proposed and integrated into the proposed genetic algorithm. The detailed contributions are as follows:

(1) Clear definitions of the similarity and the concentration were given. And then a mixed selection operator based on the fitness value and the concentration value was proposed. This operator increased the diversity of the population and could prevent the "premature" convergence to some extent.
(2) New kinds of crossover operator based on the machine and mutation operator based on the critical path were specifically designed according to the graph model of JSSP. To calculate the critical path, we presented a new algorithm for finding it from schedule.
(3) A local search operator was designed so as to improve the local search ability of GA.
(4) Based on these genetic operators, we proposed a hybrid genetic algorithm (HGA) and proved its convergence. Finally, the efficiency of the proposed algorithm was verified by computer simulations on some typical scheduling problems.

The remainder of the paper is organized as follows. In Section 2, the models of the JSSP are set up. In Section 3, a hybrid genetic algorithm to the JSSP is presented and its convergence is proved. Section 4 presents the experimental results. The conclusions are made in Section 5.

2. Modeling the job shop scheduling problem

The JSSP with which we are concerned can be described as follows [24]: There are n different jobs to be processed on m different machines. Each job needs m operations and each operation needs to be processed without preemption for a fixed processing time on a given machine. There are several constraints on jobs and machines:

- A job can visit a machine once and only once.
- There are no precedence constraints among the operations of different jobs.
- Preemption of operations is not allowed.
- Each machine can process only one job at a time.
- Each job can be processed by only one machine at a time.
- Neither release times nor due dates are specified.

The problem is to find a schedule to minimize the makespan, that is, to minimize the time required to complete all jobs.

In order to clearly describe the proposed genetic operators, we briefly introduce the disjunctive graph theory model for the JSSP [25].

Given an instance of job shop scheduling problem, we can associate it with a disjunctive graph $G=(N,A,E)$, where N is a set of all operations (seen as vertices), $N=\{0,1,O_{ij}, i=1,2,\ldots,n; j=1,2,\ldots,m|\}$, O_{ij} represents the jth operation of the ith job. 0 and 1 are two virtual processes, respectively, and represent the beginning and end of the schedule. A is a set of edges which connect the adjacent operations of the same job. $A=\{(0,O_{i1})|i=1,2,\ldots,n\}\cup\{(O_{i,j},O_{i,j+1})|i=1,2,\ldots,n,j=1,2,\ldots,m-1\}\cup\{(O_{im},1)|i=1,2,\ldots,n\}$. It reflects the process constraint between the different operations of the same job (indicated with directed solid line in Fig. 1). E is a set of non-directed edges which connect the operations processed on the same machine. Non-directed edge is the edge there are two possible directions (indicated with dashed line in Fig. 1). $E=E_1\cup E_2\cup\cdots\cup E_m$, where the subset $E_j(j=1,2,\ldots,m)$ expresses the non-repetition edges which connect the operations processed on the same machine j. If the number of the operations processed on machine j is d_j, the number of edges on subset E_j is $C_{d_j}^2$.

If we can fix the directions of non-directed edges on each subset and form a path over all nodes on each subset, then we can determine the sequence of the operations processed on each machine. These non-directed edges are replaced with directed edges (indicated with directed dashed line in Fig. 2) after the directions are fixed. Thus, we have found a schedule to the corresponding JSSP. Therefore, selecting a schedule is equivalent to choosing a graph $G(N,A,E')$, where $E'=E'_1\cup E'_2\cup\cdots\cup E'_m$, E'_j is a directed connected path which traverse all vertices associated with E_j. The number of the edges on the subset E'_j is d_j-1. If the selected graph is acyclic, it corresponds to a feasible schedule.

Each vertex $O_{ij}\in N$ is corresponding to a weight t_{ij}, where t_{ij} is the processing time of the jth operation of the ith job. Vertices 0 and 1 have weight zero. We define the longest path from vertex 0 to vertex 1 as the critical path. Then the length of critical path corresponds to the maximum completion time of the scheduling. Finding the minimum critical path from all the directed graph $G(N,A,E')$ is equivalent to finding the optimal solution of the JSSP (i.e. the solution corresponding to the minimum makespan).

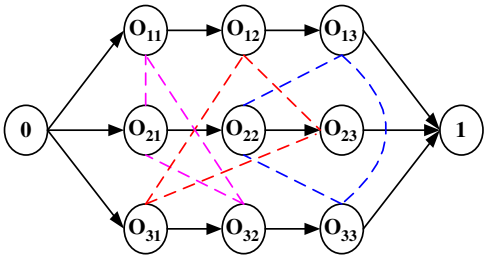

Fig. 1. The model of the problem.

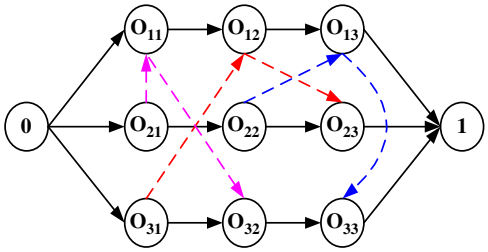

Fig. 2. A solution of the problem.

Table 1
A 3-job and 3-machine JSSP.

	Jobs	Operations		
		1	2	3
Processing time	J_1	3	3	2
	J_2	1	5	3
	J_3	3	2	3
Machine order	J_1	M_1	M_2	M_3
	J_2	M_1	M_3	M_2
	J_3	M_2	M_1	M_3

Consider a 3-job and 3-machine problem given in Table 1 as an example. A graph $G=(N,A,E)$ can be corresponding to this problem. From the above discussion, we get

$$N=\{0,1,O_{11},O_{12},O_{13},O_{21},O_{22},O_{23},O_{31},O_{32},O_{33}\}$$

$$A=\left\{\begin{array}{l}(0,O_{11}),(0,O_{21}),(0,O_{31}),(O_{11},O_{12}),(O_{12},O_{13}),(O_{21},O_{22}),(O_{22},O_{23}),\\(O_{31},O_{32}),(O_{32},O_{33}),(O_{13},1),(O_{23},1),(O_{33},1)\end{array}\right\}$$

$$E=E_1\cup E_2\cup E_3$$

where $E_1=\{(O_{11},O_{21}),(O_{11},O_{32}),(O_{21},O_{32})\}$, $E_2=\{(O_{12},O_{31}),(O_{12},O_{23}),(O_{23},O_{31})\}$, $E_3=\{(O_{13},O_{22}),(O_{13},O_{33}),(O_{22},O_{33})\}$, O_{ij} represents the jth operation of the ith job. We use directed solid line to indicate the edges in A and dashed line to indicate the edges in E. Thus, we get the model of the problem as shown in Fig. 1. If we fix the directions of non-directed edges in E_i ($i=1,2,3$) and form a path over all nodes on each E_i, denoted by E_i', such as $E_1'=\{(O_{21},O_{11}),(O_{11},O_{32})\}$, $E_1'=\{(O_{31},O_{12}),(O_{12},O_{23})\}$, $E_3'=\{(O_{22},O_{13}),(O_{13},O_{33})\}$, then we find a schedule to this problem as shown in Fig. 2 (edges in E_i ($i=1,2,3$) are indicated with directed dashed line).

3. A new hybrid genetic algorithm for JSSP

3.1. Encoding and decoding

Defining an appropriate representation of individual is the most important and critical issue for constructing an efficient GA. As mentioned above, determining a schedule is equivalent to determine a graph $G(N,A,E')$, where $E'=E_1'\cup E_2'\cup\cdots\cup E_m'$, and E_j' is a directed connected path which traverses all vertices associated with E_j, that is, if we can make sure the sequence of the operations processed on each machine, we can determine a schedule. Thus, the following encoding scheme proposed by Falkenauer and Bouffoix [26] is adopted in this paper: a chromosome is composed of m substrings, corresponding to m different machines. Each substring represents the sequence of the operations processed on each machine, denoted by a permutation of integers from 1 to n. The total length of a chromosome is $n\times m$. Consider a 3-job and 3-machine problem given in Table 1 as an example. Suppose a schedule is given as shown in Fig. 2, then the corresponding chromosome is: [(2 1 3) (3 1 2) (2 1 3)]. Here, substrings (2 1 3), (3 1 2), (2 1 3) represent, respectively, the sequences of the operations processed on machine 1, 2 and 3. Conversely, from the chromosome [(2 1 3) (3 1 2) (2 1 3)], we also can get the corresponding schedule as shown in Fig. 2.

If the selected graph $G(N,A,E')$ is acyclic, it corresponds to a feasible schedule. We can use the following method to decode all the feasible schedules. We introduce the decoding method through decoding the chromosome [(2 1 3) (3 1 2) (2 1 3)]. According to Table 1, operations O_{11},O_{21},O_{32} must be scheduled on machine 1, operations O_{12},O_{23},O_{31} must be scheduled on machine 2 and operations O_{13},O_{22},O_{33} must be scheduled on machine 3, where O_{ij} represents the jth operation of the ith job. And the schedule must meet the process constraint $O_{11}\rightarrow O_{12}\rightarrow O_{13}$, $O_{21}\rightarrow O_{22}\rightarrow O_{23}$ and $O_{31}\rightarrow O_{32}\rightarrow O_{33}$, where $O_{ij}\rightarrow O_{pq}$ means operation O_{ij} must be scheduled before O_{pq}. The substrings (2 1 3), (3 1 2), (2 1 3) represent the sequences of the operations processed on machine 1, 2 and 3, respectively. Therefore, the sequences of the operations processed on machine 1 is $O_{21}\rightarrow O_{11}\rightarrow O_{32}$, machine 2 is $O_{31}\rightarrow O_{12}\rightarrow O_{23}$ and machine 3 is $O_{22}\rightarrow O_{13}\rightarrow O_{33}$. So the first operations which can operate on the machine 1, 2, 3 are operations O_{21},O_{31},O_{22}, respectively. After considering the process constraint, we schedule operation O_{21} on machine 1, operation O_{31} on machine 2 and operation O_{22} on machine 3 one after another at the earliest allowable time. Then the operations which can operate on machine 1, 2, 3 are, respectively, operations O_{11},O_{12},O_{13}. After considering the process constraint, we schedule operations O_{11},O_{12},O_{13} on machine 1, 2, 3 one after another at the earliest allowable time. And then the operations can operate on machine 1, 2, 3 are, respectively, operations O_{32},O_{23},O_{33}. After considering the process constraint, we schedule operation O_{32} on machine 1, operation O_{23} on machine 2 and operation O_{33} on machine 3 one after another at the earliest allowable time. Then we get the corresponding schedule shown in Fig. 3.

Obviously, if the selected graph $G(N,A,E')$ includes one or more cycles, it corresponds to an infeasible schedule. Then we use the following method to change an infeasible schedule into a feasible schedule. Suppose a schedule is given as shown in Fig. 4, then the corresponding chromosome is: [(2 3 1) (1 3 2) (2 1 3)]. So the

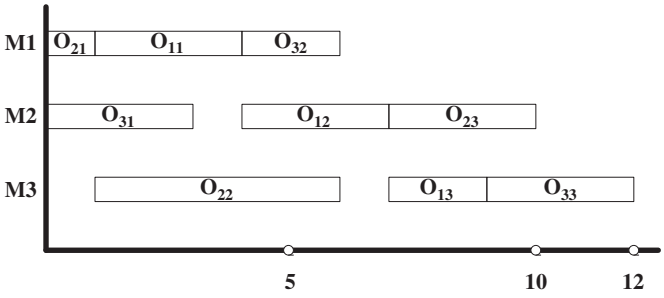

Fig. 3. Decoding of the chromosome [(2 1 3)(3 1 2)(2 1 3)].

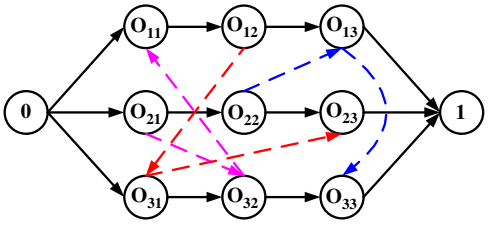

Fig. 4. An infeasible scheduling.

sequences of the operations processed on machine 1 is $O_{21}\rightarrow O_{32}\rightarrow O_{11}$, machine 2 is $O_{12}\rightarrow O_{31}\rightarrow O_{23}$ and machine 3 is $O_{22}\rightarrow O_{13}\rightarrow O_{33}$ and the process constraint is same to above example. So the first operations which can operate on machine 1, 2, 3 are, respectively, operations O_{21},O_{12},O_{22}. After considering the process constraint, we schedule operation O_{21} on machine 1, operation O_{22} on machine 3 at the earliest allowable time, respectively. Then the operations which can operate on machine 1, 2, 3 are, respectively, operations O_{32},O_{12},O_{13}. After considering the process constraint, we find none of the operations meets the process constraint because operations O_{31}, O_{11} and O_{12} have not been scheduled yet. That means the schedule corresponding to this chromosome is an infeasible schedule. Then we consider the next operations which can operate on each machine. They are operations O_{11},O_{31},O_{33}, respectively, processed on machine 1, 2, 3. After considering the process constraint, we schedule operation O_{11} on machine 1, operation O_{31} on machine 2 at the earliest allowable time, respectively. Then we schedule operation O_{32} on machine 1, operation O_{11} on machine 2 at the earliest allowable time. After that we schedule operation O_{23} on machine 2, operation O_{13} on machine 3 at the earliest allowable time. At last we schedule operation O_{33} on machine 3. Then we get the corresponding schedule as shown in Fig. 3. Finally, we find the chromosome corresponding to the Fig. 3 is [(2 1 3) (3 1 2) (2 1 3)].That is, we change the infeasible schedule [(2 3 1) (1 3 2) (2 1 3)] into a feasible schedule [(2 1 3) (3 1 2) (2 1 3)] through above method.

3.2. Fitness function and selection operation

Fitness function is defined so as to determine the quality of individuals, and it is usually relevant to the objective function to be optimized. The greater the fitness of a chromosome is, the greater its survival probability. In this study, the fitness function is defined as $f(x)=1/makespan$, where x is any individual in the population.

The selection phase is responsible to choose individuals for reproduction. Most of the selection strategies of GA adopt the method that the probability of selecting an individual is proportional to its fitness value. This can easily lead to the "premature" convergence because of the population being filled with more similar individuals [27]. To avoid this problem, a mixed selection operator based on the fitness value and the concentration value was proposed in this paper. This operator increases the diversity of the population and can prevent the "premature" convergence to some extent. Given any two individuals $x=[(x_1,x_2,\ldots,x_n),(x_{n+1},x_{n+2},\ldots,x_{2n}),\ldots(x_{n(m-1)+1},x_{n(m-1)+2},\ldots,x_{n\times m})]$ and $y=[(y_1,y_2,\ldots,y_n),(y_{n+1},y_{n+2},\ldots,y_{2n}),\ \ldots(y_{n(m-1)+1},y_{n(m-1)+2},\ldots,y_{n\times m})]$, we first give the definitions of the similarly degree, the similarity and the concentration.

Definition 1. For job i in the pth substrings (i.e., the pth parenthesis) of x and y ($i\in(1,2,\ldots,n)$, $p\in(1,2,\ldots,m)$), there may be some jobs which are in a position after i or before i in both corresponding strings of x and y. We called the number of these jobs as the similarity degree of job i in the pth substrings on x and y, and denoted as $SDE(i,p)$.

Definition 2. We called the sum of the similarity degrees of all jobs in substring p ($p\in(1,2,\ldots,m)$) as the similarity degree of substring p on x and y, and denoted as $SDS(p)$. $SDS(p)=\sum_{i=1}^{n} SDE(i,p)$. The larger the similarity degree, the more similar the corresponding substrings in two individuals.

Definition 3. Suppose $SDE(i,p)$ is the similarity degree of job i in the pth substrings on individuals x and y. Then the similarity of x and y is defined as

$$s(x,y)=\frac{\sum_{p=1}^{m}\sum_{i=1}^{n} SDE(i,p)}{n\times(n-1)m}$$

We use the 4-job and 3-machine problem as an example to explain the similarity degree. Assume that x=[(1 2 4 3) (3 1 4 2) (4 3 1 2)] and y=[(2 1 4 3) (3 1 4 2) (4 1 3 2)] are two given individuals. By comparison of the corresponding substrings of these two individuals, we observe that: for substring 1 of both individuals, there are two jobs 3 and 4 in a position after 1. There are also two jobs 3 and 4 in a position after 2. There are three jobs 1, 2 and 4 in a position before 3, and there are two jobs 1 and 2 in a position before 4 and there is one job 3 in a position after 4. Therefore, $SDE(1,1)=2$, $SDE(2,1)=2$, $SDE(3,1)=3$, $SDE(4,1)=3$ and $SDS(1)=2+2+3+3=10$. As similar, we get $SDS(2)=3+3+3+3=12$ and $SDS(3)=2+3+2+3=10$, respectively. Finally, the similarity of these two individuals is $s(x,y)=(10+12+10)/36=0.889$.

Definition 4. Suppose L is the population size and λ is a threshold value. In the current population, we use $Q(x,\lambda)$ to denote the total number of the individuals whose similarity with x is greater than or equal to the threshold λ. Then the concentration value of individual x is defined as $c(x)=Q(x,\lambda)/L$.

Obviously, the greater the value $s(x,y)\in[0,1]$ is, the more similar the individuals x and y. If $s(x,y)=1$, the individuals x and y are the same individual. The greater the value $c(x)\in[0,1]$ is, the more individuals are similar with individual x.

Suppose $x_1,x_2,\ldots x_L$ are the individuals in the population, L is the population size, $f(x_l)$ ($l=1,2,\ldots,L$) is the fitness function and $c(x_l)$ is the concentration value of individual x_l. The following selection operator based on the fitness value and the concentration value was given in this paper.

Algorithm 1. (A mixed selection operator based on the fitness value and the concentration value):

Step 1: For each individual x_l in the population, compute the pre-selection probability by using two different ways: one is by $p_f(x_l)=f(x_l)/\sum_{k=1}^{L} f(x_k)$, and another is by $p_c(x_l)=(1-c(x_l))/(L-\sum_{k=1}^{L} c(x_k))$. The former is based on fitness value and the later is based on the concentration. If $L=\sum_{k=1}^{L} c(x_k)$, we define $p_c(x_l)=0$.

Step 2: For each individual x_l in the population, define the selection probability $p_s(x_l)=\mu p_f(x_l)+(1-\mu)p_c(x_l)$ called mixed selection probability by mixing pre-selection probabilities $p_f(x_l)$ and $p_c(x_l)$, where $\mu\in(0,1)$ is a coefficient.

Step 3: Use the roulette wheel selection strategy with the probability $p_s(x_l)$ to select individuals from the population and generate new population

3.3. A new crossover operator

The goal of the crossover is to obtain better offspring by making use of the parental information. In this paper, a new crossover operator based on the characteristic of the JSSP itself was designed. Suppose x and y are two parent individuals, the crossover operator designed in this paper can be described as follows.

Algorithm 2. (A new crossover operator based on the machine):

Step 1: Divide the machines $1,2,\cdots,m$ into two complementary sets A and B.

Step 2: For two parent individuals x and y, swap the genes in the substrings that belong to set B with probability of p_c and get two children x' and y'.

Swapping the genes in the substrings that belong to set B is equivalent to swapping the sequences of the operations processed on machines which belong to set B. In this way, the children can effectively inherits the sequences of the operations processed on machines that belong to set A.

We used a 3-job and 3-machine problem as an example to explain the steps of the crossover operator.

Suppose the two parent individuals parent 1 and parent 2 are [(213) (312) (213)] and [(321) (321) (321)]. We get the graph model of the two parents shown in Figs. 5 and 6 at first.

Second, we divide the machines into two complementary sets, e.g., {1, 3} and {2}. Third, we swap the substring 2 and get two children [(213) (321) (213)] and [(321) (312) (321)]. Figs. 7 and 8 show the child 1 and 2 respectively.

The crossover operator designed in this paper is similar to the crossover operator in [26]. However, the crossover operator in [26] is like to a single point crossover, it can only separate the machines 1,2,…,m from the middle to two set, such as {1,2,…,m_1},{m_1+1, m_1+2,…,m}and then executes the crossover. The crossover operator designed in this paper is like to a multi-point crossover, it can randomly divide the machines 1,2,…,m into two complementary sets. Obviously, the crossover operator designed in this paper is more flexible, and it increases the diversity of the population.

From the proposed crossover operator, we can observe that the children can inherit the sequences of the operations processed on some machines of their parents and the crossover operator is easy to execute.

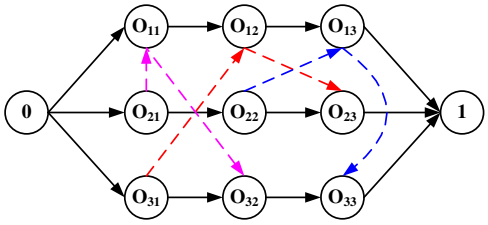

Fig. 5. Parent 1.

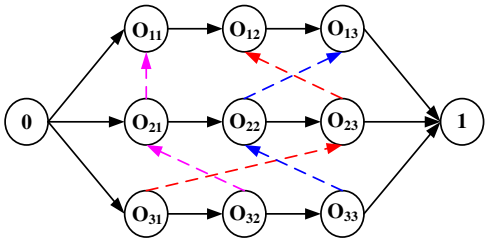

Fig. 6. Parent 2.

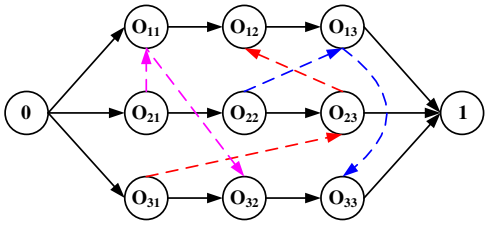

Fig. 7. Child 1.

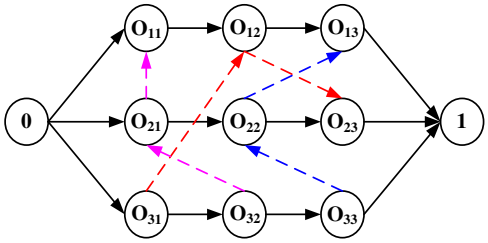

Fig. 8. Child 2.

3.4. A new mutation operator

The mutation operator further carries out some adjustments for a parent individual and gets an offspring. It can increase the diversity of the population and is helpful to jump out local optimal solutions.

According to Dell'Amica and Tubian [28], there are the following properties for JSSP:

(1) If x is a feasible solution, then reversing one of the oriented edges on a critical path of x can never lead to an infeasible solution;
(2) If the reversal of an oriented edge of a feasible solution x that does not belong to a critical path leads to a feasible solution x', then the critical path in x' cannot be shorter than the critical path in x.

According to these properties, a mutation operator based on the critical path was designed. Let $PM(O_{ij})$ and $SM(O_{ij})$ denote the immediate predecessor and the immediate successor of operation O_{ij}, respectively, on the same machine if they exist, otherwise, $PM(O_{ij})=0$, $SM(O_{ij})=1$, where O_{ij} represents the jth operation of the ith job. Then the proposed mutation operator can be described as follows.

Algorithm 3. (A new mutation operator based on the critical path):

Step 1. Calculate the critical path of a parent. Suppose that the obtained critical path is $cp=\{O_{i_1j_1},O_{i_2j_2},\ldots,O_{i_qj_q}\}$. Let $k=1$, the child=the parent.

Step 2. If operations $O_{i_kj_k}$ and $O_{i_{k+1}j_{k+1}}$ are processed on the same machine, then consider all possible permutations of $\{PM(O_{i_kj_k}),O_{i_kj_k},O_{i_{k+1}j_{k+1}}\}$ and $\{O_{i_kj_k},O_{i_{k+1}j_{k+1}},SM(O_{i_{k+1}j_{k+1}})\}$ in which arc $(O_{i_kj_k},O_{i_{k+1}j_{k+1}})$ is inverted, randomly select one of them, and use this operations sequence to replace the previous sequence in the child with probability of p_m.

Step 3. If $k<q$ then let $k=k+1$ and go to step2, else, output the child.

In order to calculate the critical path, we presented a new algorithm for finding the critical path. Suppose $S(O_{ij})$ and $C(O_{ij})$ are the starting time and the completing time of operation O_{ij}, where O_{ij} represents the jth operation of the ith job. $i=1,2,\ldots,n$, $j=1,2,\ldots,m$. The algorithm is as follows.

Algorithm 4. (A new algorithm for finding the critical path):

Step 1: Decode the given chromosome to get a feasible schedule.

Step 2: Let $K=\{O_{ij}|i=1,2,\ldots,n;j=1,2,\ldots,m\}$, set $Q=\phi$ (ϕ represents an empty set).

Step 3: While ($\exists\alpha\in K$, there is no $\beta\in K$ fulfilling $C(\alpha)=S(\beta)$), then delete α from K and update K.

Step 4: Select an operation $\alpha_1 \in K$ fulfilling $S(\alpha_1)=0$ and insert it into Q, Let $k=1$.

Step 5: While($C(\alpha_k) \neq makespan$), then $k=k+1$, select an operation $\alpha_k \in K$ fulfilling $S(\alpha_k)=C(\alpha_{k-1})$ and insert it into Q.

Step 6: List the elements $\alpha_1, \alpha_2,\ldots,\alpha_k$ in the set Q. Obviously, $\alpha_1 \to \alpha_2 \to \cdots \to \alpha_k$ is the critical path.

We used the 3-job and 3-machine problem as an example to explain the steps of the mutation operator. Suppose the parent individual is [(2 1 3) (3 1 2) (2 1 3)].

At first, we calculate the critical path of the parent by Algorithm 4, and decode chromosome [(2 1 3) (3 1 2) (2 1 3)] and get the corresponding schedule shown in Fig. 2.

Let $K=\{O_{11},O_{12},O_{13},O_{21},\ldots,O_{33}\}$, $Q=\phi$. For the operations $\alpha=O_{32},O_{31},O_{23},O_{22}$, there has no $\beta \in K$ fulfilling $C(\alpha)=S(\beta)$, so delete them from K and get $K=\{O_{11},O_{12},O_{13},O_{21},O_{33}\}$.

Since $S(O_{21})=0$, so put O_{21} into set Q at first. As $C(O_{21})=S(O_{11})$,$C(O_{11})=S(O_{12})$,$C(O_{12})=S(O_{13})$, $C(O_{13})=S(O_{33})$, and $S(O_{33})=makespan$, therefore, insert them into set Q one after another and get set $K=\{O_{21},O_{11},O_{12},O_{13},O_{33}\}$. As a result, we get a critical path $O_{21} \to O_{11} \to O_{12} \to O_{13} \to O_{33}$.

On the critical path, operations O_{21} and O_{11} are assigned to machine 1, operations O_{13} and O_{33} are assigned to machine 3. For operations O_{21} and O_{11}, we consider schedules $\{(O_{11},O_{21},O_{32}),(O_{11},O_{32},O_{21}),(O_{32},O_{11},O_{21})\}$. Select one of them, such as (O_{11},O_{21},O_{32}), to replace the previous sequence (O_{21},O_{11},O_{32}) with probability of p_m. For operations O_{13} and O_{33}, we consider schedules $\{(O_{22},O_{33},O_{13}),(O_{33},O_{22},O_{13}),(O_{33},O_{13},O_{22})\}$. Select one of them, such as (O_{22},O_{33},O_{13}), to replace the previous sequence (O_{22},O_{13},O_{33}) with probability of p_m, and get the child as shown in Fig. 9.

According to Dell'Amico and Trubian [28], for each feasible solution x, it is possible to construct a finite sequence of mutation operator, which will lead from x to a globally optimal solution. Therefore, this mutation operator is helpful to produce excellent individuals.

3.5. A new local search operator

To enhance the speed of the proposed algorithm, a new local search operator is proposed in this paper. Assume that x is a given individual and x' is the resulting individual through local search operator from x, and σ is the number of the new solutions to be generated in our local search operator.

Algorithm 5. (A new local search operator):

Step 1: Let $k=1$, $N_\varepsilon=\phi$ (ϕ represents an empty set).

Step 2: If $k \leq \sigma$, go to step 3, else, go to step 6.

Step 3: Randomly select a machine number $m_k \in \{1,2,\ldots,m\}$.

Step 4: Change the sequence of the operations processed on machine m_k and gets a new solution N_k.

Step 5: Put solution N_k into set N_ε, $k=k+1$, and then turn to step 2.

Step 6: Choose the best individual from N_ε as x'.

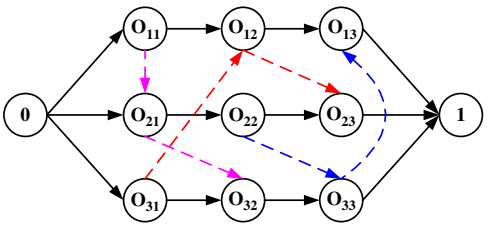

Fig. 9. The child.

3.6. The framework of the HGA

Based on aforementioned genetic operators, we proposed a new hybrid genetic algorithm (briefly HGA) to solve the JSSP. The framework of the algorithm can be described as follows:

Algorithm 6. (Hybrid genetic algorithm: HGA):

Step 1. Initialization: Generate initial population $P(0)$, and let $t=0$. Evaluate the fitness value of each individual in $P(0)$.

Step 2. Crossover: Group all the individuals in $P(t)$ into pair wise randomly, and use Algorithm 2 to each pair of individuals and obtain a set of offspring, denoted by $P'(t)$.

Step 3. Mutation: Use Algorithm 3 to all the individuals in $P'(t)$ and obtain a set of offspring, denoted by $P''(t)$.

Step 4. Evaluate fitness value: Evaluate the fitness value of each individual in $P''(t)$.

Step 5. Local Search: Use Algorithm 5 to each individual in $P''(t)$ with probability of p_l and obtain a set of offspring, denoted by $P'''(t)$.

Step 6. Selection: Select N individuals from $P'''(t)$ by Algorithm 1 to get a tentative population, still denoted as $P'''(t)$, then use the elitist strategy to the union of $P'''(t)$ and $P(t)$ to get the next generation population $P(t+1)$.

Step 7. If the stop criterion is satisfied, then stop. Otherwise, let $t=t+1$, and turn to step 2.

3.7. Convergence analysis of the HGA

At first, we introduce several symbols.

- Ω: The feasible solution space.
- x: Any feasible solution, $x \in \Omega$.
- $f(x)$: The objective function, $f(x)=makespan$.
- f^*: The globally minimum of the objective function, i.e., $f^*=\min\{f(x)|x \in \Omega\}$.
- X^*: The set of optimal solutions, i.e., $X^*=\{x \in \Omega | f(x)=f^*\}$.
- T: The first time (generation) to find the optimal solution, i.e., $T=\min\{t|f_t^*=f^*\}$, where f_t^* is the minimum of the tth generation of population.

For proposed algorithm, we can get the following conclusions.

Theorem 1. *For algorithm HGA with any initial population $P(0)$, suppose the crossover, mutation, local search and selection operators are independent of each other and the crossover probability is p_c, the mutation probability is p_m and the local search probability is p_l. HGA can find the global optimum in limited iterations with probability* 1, *i.e.,* $p\{T<+\infty\}=1$.

Proof. According to Dell'Amico and Trubian [28], for $\forall x \in \Omega$, it is possible to construct a finite sequence of mutation operations, which will lead from x to a globally minimal solution $x^* \in X^*$. This process can be represented as: $x=x_0 \to x_1 \to \cdots \to x_t=x^*$. Suppose the critical path of x_k is $cp=\{O_{i_1j_1},O_{i_2j_2},\ldots,O_{i_qj_q}\}$. Form Algorithm 3, if operations $O_{i_kj_k}$ and $O_{i_{k+1}j_{k+1}}$ are processed on the same machine, then consider all possible permutations of $\{PM(O_{i_kj_k}),O_{i_kj_k},O_{i_{k+1}j_{k+1}}\}$ and $\{O_{i_kj_k},O_{i_{k+1}j_{k+1}},SM(O_{i_{k+1}j_{k+1}})\}$ (a total of six cases) in which arc $(O_{i_kj_k},O_{i_{k+1}j_{k+1}})$ is inverted, randomly select one of them, and use this operations sequence to replace the previous sequence with probability of p_m.

Therefore, the probability of getting x_{k+1} from x_k ($k=0,1,2,\ldots,t-1$) after mutation should fulfill $p\{x_{k+1}=\text{mutation}\{x_k\}\} \geq (p_m(1-p_m)^{q-1}/6)=\delta_m>0$, where q is the number of operations on the critical path. Note k_x as the length of the shortest path from x to a point in X^* through a finite successive mutation. Let $k^*=\max\{k_x|x \notin X^*\}$.

For $\forall x \in P(t)$, the probability of x not being destroyed after crossover operator is $p(x \in P'(t)) \geq 1-p_c = \delta_c > 0$.

Thus, after mutation, the probability of getting x_1 from x is $p\{x_1 \in P''(t)\} \geq \delta_m > 0$. The probability of x_1 not being destroyed after local search operator is $p(x_1 \in P'''(t)) \geq 1-p_l = \delta_l > 0$. The probability of x_1 being selected to the next generation population $P(t+1)$ is $P\{x_1 \in P(t+1)\} \geq p_s(x_1) > 0$. Let $\delta_s = \min\{p_s(x_1), p_s(x_2), \ldots, p_s(x_t)\}$, then, after the crossover, mutation, local search and selection once, the probability of getting x_1 from x is $p\{x_1 \in P(t+1)\} \geq \delta_c\ \delta_m\ \delta_l\ \delta_s > 0$.

Let $p(k_x)$ denote the probability of getting $x^* \in X^*$ from x after k_x generations, then $p(k_x) \geq (\delta_c \cdot \delta_m \cdot \delta_l \cdot \delta_s)^{k_x} > 0$. Let $p(k^*)$ denote the probability of getting an optimal solution from x after k^* generation, then $p(k^*) \geq (\delta_c \cdot \delta_m \cdot \delta_l \cdot \delta_s)^{k^*} = \delta > 0$. The probability not finding the optimal solution after k^* generation is $p_{not}(k^*) \leq 1-\delta > 0$. So $p_{not}(1) \leq (1-\delta)^{1/k^*} > 0$ and $p_{not}(k) \leq (1-\delta)^{k/k^*} > 0$.

Therefore, $\lim_{k\to\infty} p_{not}(k) \leq \lim_{k\to\infty}(1-\delta)^{k/k^*} = 0$. Since $p\{T < +\infty\} \geq$ {find the optimal solution at k generations} $= 1-p_{not}(k)$, let $k \to \infty$, we can get P{ $T < +\infty$ } ≥ 1. Thus, p{ $T < +\infty$ } $= 1$. □

Theorem 2. *Algorithm HGA converges to the optimal solution with probability* 1.

Proof. According to Theorem 1, HGA can find the global optimum in limited iterations with probability 1. Because of adopting the elitist strategy in HGA, the algorithm converges to the optimal solution with probability 1. □

From the proof process, it can be easily found that because of using the new mutation operator based on the critical path designed in this paper, algorithm HGA can find the global optimum in limited iterations with probability 1. Therefore, if we do not use the mutation operator designed in this paper, the convergence of the algorithm cannot be guaranteed.

4. Simulation results

In order to verify the good performance of the proposed hybrid genetic algorithm, we use 43 instances from two classes of standard JSSP test problems: Fischer and Thompson [2] instances FT06, FT10, FT20 and Lawrence [29] instances LA01–LA40.

Table 2
Experimental results.

Problem	Size	Best Known Solution (BKS)	HGA		Yang (2008) MA[30]	Goncalves (2005) Param. active[31]	Ombuki (2004) LSGA[32]	Coello (2003) AIS[33]	Wang (2001) MGA[16]	Binato (2001) GRASP[34]	Sabuncuoglu (1999) Beam search[35]
			1	2							
FT06	6 × 6	55	**55**	**55**	55	55	–	–	55	55	–
FT10	10 × 10	930	**930**	**938**	930	930	–	941	930	938	1016
FT20	20 × 5	1165	**1165**	**1165**	1165	1165	–	–	1165	1169	–
LA01	10 × 5	666	**666**	**666**	666	666	–	666	666	666	666
LA02	10 × 5	655	**655**	**655**	655	655	–	655	–	655	704
LA03	10 × 5	597	**597**	**597**	597	597	–	597	–	604	650
LA04	10 × 5	590	**590**	**590**	590	590	–	590	–	590	620
LA05	10 × 5	593	**593**	**593**	593	593	–	593	–	593	593
LA06	15 × 5	926	**926**	**926**	926	926	–	926	926	926	926
LA07	15 × 5	890	**890**	**890**	890	890	–	890	–	890	890
LA08	15 × 5	863	**863**	**863**	863	863	–	863	–	863	863
LA09	15 × 5	951	**951**	**951**	951	951	–	951	–	951	951
LA10	15 × 5	958	**958**	**958**	958	958	–	958	–	958	958
LA11	20 × 5	1222	**1222**	**1222**	1222	1222	–	–	1222	1222	1222
LA12	20 × 5	1039	**1039**	**1039**	1039	1039	–	–	–	1039	1039
LA13	20 × 5	1150	**1150**	**1150**	1150	1150	–	–	–	1150	1150
LA14	20 × 5	1292	**1292**	**1292**	1292	1292	–	–	–	1292	1292
LA15	20 × 5	1207	**1207**	**1207**	1207	1207	–	–	–	1207	1207
LA16	10 × 10	945	**945**	**945**	945	945	959	945	945	946	988
LA17	10 × 10	784	**784**	**784**	784	784	792	785	–	784	827
LA18	10 × 10	848	**848**	**848**	848	848	857	848	–	848	881
LA19	10 × 10	842	**844**	**844**	844	842	860	848	–	842	882
LA20	10 × 10	902	**907**	**911**	907	907	907	907	–	907	948
LA21	15 × 10	1046	**1046**	**1046**	–	1046	1114	–	1058	1091	1154
LA22	15 × 10	927	**935**	**935**	–	935	989	–	–	960	985
LA23	15 × 10	1032	**1032**	**1032**	–	1032	1035	–	–	1032	1051
LA24	15 × 10	935	**953**	**953**	–	953	1032	–	–	978	992
LA25	15 × 10	977	**981**	**984**	–	986	1047	1022	–	1028	1073
LA26	20 × 10	1218	**1218**	**1218**	–	1218	1307	–	1218	1271	1269
LA27	20 × 10	1235	**1236**	**1256**	–	1256	1350	–	–	1320	1316
LA28	20 × 10	1216	**1216**	**1225**	–	1232	1312	1277	–	1293	1373
LA29	20 × 10	1152	**1160**	**1196**	–	1196	1311	1248	–	1293	1252
LA30	20 × 10	1355	**1355**	**1355**	–	1355	1451	–	–	1368	1435
LA31	30 × 10	1784	**1784**	**1784**	–	1784	1784	–	1784	1784	1784
LA32	30 × 10	1850	**1850**	**1850**	–	1850	1850	–	–	1850	1850
LA33	30 × 10	1719	**1719**	**1719**	–	1719	1745	–	–	1719	1719
LA34	30 × 10	1721	**1721**	**1721**	–	1721	1784	–	–	1753	1780
LA35	30 × 10	1888	**1888**	**1888**	–	1888	1958	1903	–	1888	1888
LA36	15 × 15	1268	**1287**	**1287**	–	1279	1358	1323	1291	1334	1401
LA37	15 × 15	1397	**1407**	**1408**	–	1408	1517	–	–	1457	1503
LA38	15 × 15	1196	**1196**	**1219**		1219	1362	1274	–	1267	1297
LA39	15 × 15	1233	**1233**	**1245**		1246	1391	1270	–	1290	1369
LA40	15 × 15	1222	**1229**	**1241**		1241	1323	1258	–	1259	1347

Table 3
Average relative deviation to the BKS.

Algorithm	NIS	ARD		Improvement
		OA (%)	HGA (%)	HGA (%)
Yang (2008) MA	13	0.06	0.04	0.02
Goncalves (2005) Param. active	43	0.39	0.17	0.22
Ombuki (2004) LSGA	25	5.22	0.29	4.94
Coello (2003) AIS	24	1.51	0.15	1.36
Wang (2001) MGA	11	0.27	0.13	0.14
Binato (2001) GRASP	43	1.68	0.17	1.51
Sabuncuoglu (1999) Beam search	41	4.02	0.18	3.84
Dauzere (1997) Tabu search	43	0.86	0.17	0.69

The HGA was compared with some algorithms reported in literature [16,30–35] in recent years.

The parameters used in simulations are as follows: the population size $N=100$, the crossover probability $p_c=0.7$, the mutation probability $p_m=0.1$, and the local search probability $p_l=0.5$. In the selection operator, $\lambda=0.8$ and $\mu=0.7$. In the local search operator, $\sigma=10$. The algorithm was run 10 independent times on each test problem.

The HGA was implemented in Visual C++ on a computer with a 2.66 GHz Intel Core 2 Duo CPU. Table 2 shows the experimental results. It lists problem name, problem size (number of jobs $\times$ number of operations), the best known solution (BKS), and the solution obtained by each of the compared algorithms. Among them, column HGA1 lists the results obtained by using the local search operator in the HGA and column HGA2 lists the results obtained by not using the local search operator.

Table 2 shows that the proposed algorithm is able to find the best known solution for 33 instances, i.e., in about 77% of the instances. For small problems FT06, FT10, FT20 and LA01–LA15, almost all the algorithms can find the optimal solution. For relatively large problems LA16–LA40, the results of the proposed algorithm (HGA) are better than or equal to those of other algorithms. For HGA, the results obtained by using the local search operator are better than the results obtained by not using the local search operator. This shows that the proposed local search operator takes an important role in improving the quality of the solutions.

For each algorithm, we can use formula $RD=100\times(MFM-BKS)/BKS$ for each instance to calculate the relative deviation, where MFM means the minimum makespan found and BKS means the best known solution. We use ARD to denote the average value of relative deviations for all the instances. Table 3 shows the number of instances solved (NIS), and the average relative deviation (ARD). The ARD was calculated for the Hybrid Genetic Algorithm (HGA), and for the other algorithms (OA) listed in the table. The last column (Improvement) presents the reduction of ARD obtained by the HGA with respect to each of the other algorithms.

From Table 3, we know the deviation of the minimum makespan found from the best known solution of HGA is only on average 0.17%. The proposed algorithm yields a significant improvement in solution quality with respect to all other algorithms.

5. Conclusions

To solve the JSSP more effectively, a mixed selection operator based on the fitness value and the concentration was designed in order to increase the diversity of the population. Crossover operator based on the machine, and mutation operator based on the critical path were designed according to the graph model of JSSP. To calculate the critical path, a new algorithm was presented. A local search operator was designed in order to improve the quality of the solutions. Based on these, a hybrid genetic algorithm was proposed, and the convergence of HGA to the global optimal solution in finite generation with probability 1 is proved. Then the convergence of HGA to the optimal solution with probability 1 is proved. The experimental results show that the proposed algorithm is effective and performs better than the compared algorithms.

Acknowledgment

The work is supported by National Natural Science Foundation of China (60873099) and Ph.D. Programs Foundation of Ministry of Education of China (20090203110005).

References

[1] Lenstra JK, Kan AHG, Brucker P. Complexity of machine scheduling problem. Annals of Discrete Mathematics 1977;1:343–62.
[2] Fisher H, Thompson GL. Probabilistic learning combinations of local job-shop scheduling rules. Englewood Cliffs, NJ: Prentice-Hall; 1963. p. 225–51.
[3] Ge HW, Sun L. An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. IEEE Transactions on System, Man, and Cybernetics-part A: Systems and Humans 2008;38(2):358–68.
[4] Dey S, Sarkar D, Basu A. A tag machine based performance evaluation method for job-shop schedules. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2010;29(7):1028–41.
[5] Wang L, Cai N, Feng HY, Ma J. ASP: an adaptive setup planning approach for dynamic machine assignments. IEEE Transactions on Automation Science and Engineering 2010;7(1):2–14.
[6] Gokbayrak K, Selvi O. Service time optimization of mixed-line flow shop systems. IEEE Transactions on Automatic Control 2010;55(2):395–404.
[7] Wang SF, Zou YR. Techniques for the job shop scheduling problem: a survey. Systems Engineering - Theory & Practice 2003;23:49–55.
[8] Holland JH. Genetic algorithm. Scientific American 1992;266:44–50.
[9] Davis L. Job shop scheduling with genetic algorithms. In: Proceedings of the first international conference on genetic algorithms, vol. 5, 1985. p. 136–40.
[10] Someya H, Yamamura M. Genetic algorithm with search area adaptation for the function optimization and its experimental analysis. In: Proceedings of the 2001 congress on evolutionary computation 2001. p. 933–40.
[11] Zhou H, Feng Y. The hybrid heuristic genetic algorithm for job shop scheduling. Computers & Industrial Engineering 2001;40(3):191–200.
[12] Park BJ, Choi HR, Kim HS. A hybrid genetic algorithm for the job shop scheduling problems. Computers & Industrial Engineering 2003;l4:597–613.
[13] Mattfeld DC, Bierwirth C. An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 2004;155:616–30.
[14] Watanabe M, Idab K, Gen M. A genetic algorithm with modified crossover operator and search area adaptation for the job shop scheduling problem. Computers & Industrial Engineering 2005;48:743–52.
[15] Li Y, Chen YA. Genetic algorithm for job shop scheduling. Journal of Software 2010;5(3):269–74.
[16] Wang L, Zheng D. An effective hybrid optimisation strategy for job-shop scheduling problems. Computers & Operations Research 2001;28(6):585–96.
[17] Gonçalves JF, Mendes JJM, Resende MGC. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 2005;167:77–95.
[18] Zhang C, Li P, Rao Y, Li S. A new hybrid GA/SA algorithm for the job shop scheduling problem. Evolutionary Computation in Combinatorial Optimization 2005;3448:246–59.
[19] Liu TK, Tsai JT, Chou JH. Improved genetic algorithm for the job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology 2005;27:1021–9.
[20] Xu X, Li C. Research on immune genetic algorithm for solving the job-shop scheduling problem. International Journal of Advanced Manufacturing Technology 2007;34:783–9.
[21] Vilcot G, Billaut JC. A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. European Journal of Operational Research 2008;190:398–411.
[22] Zhang C, Rao Y, Li P. An effective hybrid genetic algorithm for the job shop scheduling problem. International Journal of Advanced Manufacturing Technology 2008;39:965–74.
[23] Zhang R, Wu C. A hybrid immune simulated annealing algorithm for the job shop scheduling problem. Applied Soft Computing 2010;10:79–89.
[24] Gen M, Cheng R. Genetic algorithms and engineering design. New York: Wiley; 1997.
[25] Balas E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. Operations Research 1969;17:941–57.
[26] Falkenauer E, Bouffoix S. A genetic algorithm for job shop. In: Proceedings of the 1991 IEEE international conference on robotics and automation, 1991.
[27] Yan L. Isolation niche genetic algorithm. Journal of Systems Engineering 2000;15(1):86–91.

[28] Dell'Amico M, Trubian M. Applying tabu search to the job shop scheduling problem. Annals of Operation Research 1993;41:231–52.
[29] Lawrence S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Technical Report, GSIA, Carnegie Mellon University; 1984.
[30] Yang J, Sun L. Clonal selection based memetic algorithm for job shop scheduling problems. Journal of Bionic Engineering 2008;5:111–9.
[31] Goncalves JF, Mendes JJDM, Resende MGC. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 2005;167:77–95.
[32] Ombuki BM, Entresca MV. Local search genetic algorithms for the job shop scheduling problem. Applied Intelligence 2004;21:99–109.
[33] Coello C, Rivera D, Cortez N. Use of an artificial immune system for job shop scheduling. Artificial immune systems: Proceedings of the ICARIS 2003. p. 1–10.
[34] Binato S, Hery WJ, Loewenstern DM, Resende MGC. A GRASP for job shop scheduling. Essays and Surveys in Metaheuristics 2002:59–79.
[35] Sabuncuoglu I, Bayiz M. Job shop scheduling with beam search. European Journal of Operational Research 1999;118(2):390–412.