Expert Systems with Applications 38 (2011) 3563–3573

Contents lists available at ScienceDirect

Expert Systems with Applications

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

An effective genetic algorithm for the flexible job-shop scheduling problem

Guohui Zhang [a], Liang Gao [b,*], Yang Shi [b]

[a] Zhengzhou Institute of Aeronautical Industry Management, Middle Daxue Road, Zhengzhou 450015, China
[b] The State Key Laboratory of Digital Manufacturing Equipment and Technology, 1037 Luoyu Road, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Keywords:
Genetic algorithm
Flexible job-shop scheduling
Chromosome representation
Initialization

ABSTRACT

In this paper, we proposed an effective genetic algorithm for solving the flexible job-shop scheduling problem (FJSP) to minimize makespan time. In the proposed algorithm, Global Selection (GS) and Local Selection (LS) are designed to generate high-quality initial population in the initialization stage. An improved chromosome representation is used to conveniently represent a solution of the FJSP, and different strategies for crossover and mutation operator are adopted. Various benchmark data taken from literature are tested. Computational results prove the proposed genetic algorithm effective and efficient for solving flexible job-shop scheduling problem.

1. Introduction

Scheduling is one of the most important issues in the planning and operation of manufacturing systems (Chen, Ihlow, & Lehmann, 1999), and scheduling has gained much attention increasingly in recent years (Ho, Tay, Edmund, & Lai, 2007). The classical job-shop scheduling problem (JSP) is one of the most difficult problems in this area. It consists of scheduling a set of jobs on a set of machines with the objective to minimize a certain criterion. Each machine is continuously available from time zero, processing one operation at a time without preemption. Each job has a specified processing order on the machines which are fixed and known in advance. Moreover, processing time is also fixed and known.

The flexible job-shop scheduling problem (FJSP) is a generalization of the classical JSP for flexible manufacturing systems (Pezzella, Morganti, & Ciaschetti, 2007). Each machine may have the ability of performing more than one type of operations, i.e., for a given operation must be associated with at least one machine. The problem of scheduling jobs in FJSP could be decomposed into two sub-problems: the routing sub-problem that assigns each operation to a machine selected out of a set of capable machines, the scheduling sub-problem that consists of sequencing the assigned operations on all machines in order to obtain a feasible schedule to minimize the predefined objective function.

Unlike the classical JSP where each operation is processed on a predefined machine, each operation in the FJSP can be processed on one out of several machines. This makes FJSP more difficult to solve due to the consideration of both routing of jobs and scheduling of operations. Moreover, it is a complex combinatorial optimization problem. JSP is known to be NP-hard (Garey, Johnson, & Sethi, 1976). FJSP is therefore NP-hard too.

In this paper, we propose an effective GA to solve the FJSP. Global Selection (GS) and Local Selection (LS) are designed to generate high-quality initial population in the initialization stage which could accelerate convergent speed. In order to assist the initialization method and assure the algorithm perform well, we design an improved chromosome representation method "Machine Selection and Operation Sequence". In this method, we try to find an efficient coding scheme of the individuals which respects all constraints of the FJSP. At the same time, different strategies for crossover and mutation operator are employed. Computational results show that the proposed algorithm could get good solutions.

The paper is organized as follows. Section 2 gives the formulation of FJSP and shows an illustrative instance. An overview of relevant literature on the subject is provided in Section 3. Section 4 presents the approach of Machine Selection and Operation Sequence, encoding and decoding scheme, Global Selection, Local Selection and genetic operators. Section 5 presents and analyzes the performance results of effective genetic algorithm when it is applied to solve some common benchmarks from literature. Some final concluding remarks and future study directions are given in Section 6.

2. Problem formulation

The flexible job-shop scheduling problem can be formulated as follows. There is a set of N jobs $J = \{J_1, J_2, \ldots, J_i, \ldots, J_N\}$ and a set of M machines $M = \{M_1, M_2, \ldots, M_k, \ldots, M_M\}$. Each job J_i consists of a predetermined sequence of operations. Each operation requires one machine selected out of a set of available machines, namely the first sub-problem: the routing sub-problem. In addition, the FJSP

* Corresponding author.
E-mail address: gaoliang@mail.hust.edu.cn (L. Gao).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.08.145

sets its starting and ending time on each machine, namely the second sub-problem: the scheduling sub-problem. The FJSP is thus to determine an assignment and a sequence of the operations on the machines so that some criteria are satisfied. However, the FJSP is more complex and challenging than the classical JSP because it requires a proper selection of a machine from a set of available machines to process each operation of each job (Ho et al., 2007).

Some symbols used in our paper are listed as follows.

Ω	the set of all machines
N	the number of the total jobs
M	the number of the total machines
k, x	the index of alternative machine set
i	the index of the ith job
j	the index of the jth operation of job J_i
O_{ij}	the jth operation of job J_i
J_{io}	the number of the total operations of job J_i
Ω_{ij}	the set of available machines of O_{ij}
P_{ijk}	the processing time of operation O_{ij} on machine k
S_{ijk}	the start time of operation O_{ij} on machine k
E_{ijk}	the end time of operation O_{ij} on machine k
S_x	the start time of the idle time interval on machine M_x
E_x	the end time of the idle time interval on machine M_x
$L = \sum_{i=1}^{N} J_{io}$	the sum of all operations of all jobs

Kacem, Hammadi, and Borne (2002) classified the FJSP into P-FJSP and T-FJSP as follows:

If $\Omega_{ij} \subset \Omega$, then it has partial flexibility, it is partial FJSP (P-FJSP). Each operation could be processed on one machine of subset of Ω; If $\Omega_{ij} = \Omega$, then it has total flexibility, it is total FJSP (T-FJSP). Each operation could be processed on any machine of Ω. With the same number of machines and jobs, the P-FJSP is more difficult to solve than the T-FJSP (Kacem et al., 2002).

Hypotheses considered in this paper are summarized as follows:

(1) All machines are available at time 0;
(2) All jobs are released at time 0;
(3) Each machine can process only one operation at a time;
(4) Each operation can be processed without interruption on one of a set of available machines;
(5) Recirculation occurs when a job could visit a machine more than once;
(6) The order of operations for each job is predefined and cannot be modified.

For the simplicity of presenting the algorithm, we designed a sample instance of FJSP which will be used throughout the paper. In Table 1, there are 2 jobs and 5 machines, where rows correspond to operations and columns correspond to machines. Each cell denotes the processing time of that operation on the corresponding machine. In the table, the "–" means that the machine cannot execute the corresponding operation, i.e., it does not belong to the alternative machine set of the operation. So this instance is a P-FJSP.

Table 1
Processing time table of an instance of P-FJSP.

Job	Operation	M_1	M_2	M_3	M_4	M_5
J_1	O_{11}	2	6	5	3	4
	O_{12}	–	8	–	4	–
J_2	O_{21}	3	–	6	–	5
	O_{22}	4	6	5	–	–
	O_{23}	–	7	11	5	8

3. Literature review

Brucker and Schile (1990) were the first to address this problem in 1990. They developed a polynomial graphical algorithm for a two-job problem. However, exact algorithms are not effective for solving FJSP and large instances (Pezzella et al., 2007). Several heuristic procedures such as dispatching rules, tabu search (TS), simulated annealing (SA) and genetic algorithm (GA) have been developed in recent years for the FJSP. They could produce reasonably good schedules in a reasonable computational time, and could get near optimal solution easily. However, the meta-heuristics methods have led to better results than the traditional dispatching or greedy heuristic algorithm (Mastrolilli & Gambardella, 1996; Najid, Dauzère-Pérès, & Zaidat, 2002; Pezzella et al., 2007).

FJSP could be turned into the Job-shop scheduling problem when a routing is chosen, so when solving FJSP, hierarchical approach and integrated approach have been used. The hierarchical approach could reduce difficulty by decomposing the FJSP into a sequence of sub-problems. Brandimarte (1993) was the first to apply the hierarchical approach into the FJSP. Paulli (1995) solved the routing sub-problem using some existing dispatching rules, and then solved the scheduling sub-problem by different tabu search methods. Integrated approach could achieve better results, but it is rather difficult to be implemented in real operations. Hurink, Jurisch, and Thole (1994) and Dauzère-Pérès (1997) proposed different tabu search heuristic approach to solve the FJSP using an integrated approach. Mastrolilli and Gambardella (2000) proposed some neighborhood functions for the FJSP, which can be used in meta-heuristic optimization techniques, and achieve better computational results than any other heuristic developed so far, both in terms of computational time and solution quality. Then Yazdani, Amiri and Zandieh (2010) proposed a parallel variable neighborhood search algorithm that solves the FJSP to minimize makespan time.

GA is an effective meta-heuristic to solve combinatorial optimization problems, and has been successfully adopted to solve the FJSP. Recently, more and more papers are talking about this topic. They differ from each other in encoding and decoding schemes, initial population method, and offspring generation strategy. Chen et al. (1999) used integrated approach to solve the FJSP. The genes of the chromosomes respectively describe a concrete allocation of operations to each machine and the sequence of operations on each machine. Yang (2001) proposed a GA-based discrete dynamic programming approach. Zhang and Gen (2005) proposed a multistage operation-based genetic algorithm to deal with the problem from the point view of dynamic programming. Jia, Nee, Fuh, and Zhang (2003) presented a modified GA that is able to solve distribute scheduling problems and FJSP.

Kacem, Hammadi, and Borne (2002) and Kacem (2003) used tasks sequencing list coding scheme that combines both routing and sequencing information to form a chromosome representation, and developed an approach by localization (AL) to find promising initial assignment. Then, dispatching rules were applied to sequence the operations. Tay and Wibowo (2004) compared four different chromosome representations, testing their performance on some problem instances. Ho et al. (2007) proposed an architecture for learning and evolving of FJSP called Learnable Genetic Architecture (LEGA). LEGA provides an effective integration between evolution and learning within a random search process. Pezzella et al. (2007) integrate different strategies for generating the initial population, selecting the individuals for reproducing new individuals. Tay & Wibowo (2004) combined the GA and a variable neighborhood descent (VND) for solving the FJSP.

4. An effective GA for FJSP

The advantage of GA with respect to other local search algorithms is due to the fact that more strategies could be adopted together to find good individuals to add to the mating pool in a GA framework, both in the initial population phase and in the dynamic generation phase (Pezzella et al., 2007). In this paper, the proposed GA adopts an improved chromosome representation and a novel initialization approach, which can balance the workload of the machines well and converge to suboptimal solution in short time.

4.1. Chromosome representation

Better efficiency of GA-based search could be achieved by modifying the chromosome representation and its related operators so as to generate feasible solutions and avoid repair mechanism.

Ho et al. (2007) developed extensive review and investigated insightfully on chromosome representation of FJSP. Mesghouni, Hammadi, and Borne (1997) proposed parallel job representation for solving the FJSP. The chromosome is represented by a matrix where each row is an ordered sequence of each job. Each element of the row contains two terms, the first one is the machine processing the operation, and the second one is the starting time of this operation. The approach requires a repair mechanism and the decoding representation is complex. Chen et al. (1999) divided the chromosome into two parts: *A*-string and *B*-string. *A*-string denotes the routing policy of the problem, and *B*-string denotes the sequence of the operations on each machine, however this method needs to consider the order of operations and require a repair mechanism. Kacem et al. (2002) represented the chromosome by an assignment table representation. A data structure of the assignment table must necessarily describe the set of all machines. This increases the overall computational complexity due to the presence of redundant assignments. Ho et al. (2007) also divided the chromosome into two strings, one represents the operation order, the other represents the machine by an array of binary values. This structure can represent the problem clearly and conveniently, but the binary-coded increases the memory space and the operation is not convenient, so when the scale of problem is oversized, the memory space and computational time will increase tremendously (Liu, Abraham, & Grosan, 2007).

Based on the analysis of the approach from the above literature, we design an improved chromosome representation to reduce the cost of decoding, due to its structure and encoding rule, it requires no repair mechanism. Our chromosome representation has two components: Machine Selection and Operation Sequence (called MSOS) (see Fig. 1).

4.1.1. Machine Selection part (MS)

We use an array of integer values to represent Machine Selection. The length equals to *L*. Each integer value equals the index of the array of alternative machine set of each operation. For the problem in Table 1, one possible encoding of the Machine Selection part is shown in Fig. 2. For instance, M_2 is selected to process operation O_{12} since the value in the array of alternative machine set is 1. The value could also equal 2 since operation O_{12} can be processed on two machines M_2 or M_4, the valid values are 1 and 2. This demonstrates a FJSP with recirculation if more than one operation of the same job is processed on the same machine. For example, O_{11} and O_{12} both belong to J_1 and may be processed on the same machine M_2 or M_4. If only one machine could be selected for some operations, the value is 1 in the Machine Selection array. Therefore, the MS representation is flexible enough to encode the FJSP. It may easily use the same structure to represent P-FJSP or T-FJSP. This property improves the search process by requiring less memory and ignoring unused data especially to P-FJSP.

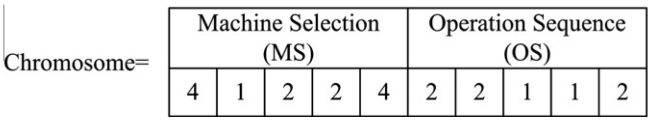


Fig. 1. Structure of proposed MSOS chromosome.

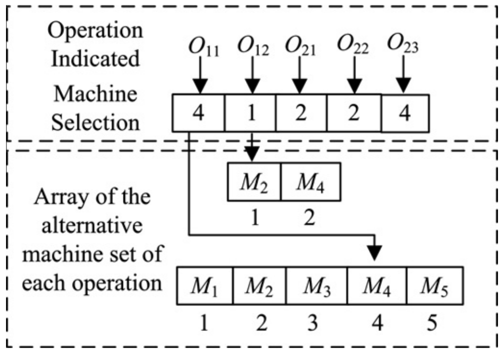

Operation Indicated
O_{11} O_{12} O_{21} O_{22} O_{23}
Machine Selection
4 1 2 2 4
Array of the alternative machine set of each operation
M_2 M_4
1 2
M_1 M_2 M_3 M_4 M_5
1 2 3 4 5

Fig. 2. Machine Selection part.

4.1.2. Operation Sequence part (OS)

We use the operation-based representation, which defines all operations for a job with the same symbol and then interpret them according to the sequence of a given chromosome. The length equals to *L* too. The index *i* of job J_i appears in the Operation Sequence part J_{io} times to represent its J_{io} ordered operations. It can avoid generating an infeasible schedule by replacing each operation corresponding to job index. For the instance in Table 1, one possible encoding of the Operation Sequence part is shown in Fig. 3. J_1 has two operations O_{11} and O_{12}; J_2 has three operations O_{21}, O_{22} and O_{23}. Reading the data from left to right and increasing operation index of each job, the Operation Sequence 2-2-1-1-2 depicted could be translated into a list of ordered operations: O_{21}-O_{22}-O_{11}-O_{12}-O_{23}.

4.2. Decoding the MSOS chromosome to a feasible and active schedule

In the literature (Pinedo, 2002, chap. 2), schedules are categorized into three classes: non-delay schedule, active schedule and semi-active schedule. It has been verified and denoted in a Venn diagram in Pinedo (2002, chap. 2) that active schedule contains optimal schedule, so only active schedule is considered in our decoding approach in order to reduce the search space. The steps for decoding an MSOS chromosome to a feasible and active schedule for a FJSP are as follows.

Step 1: Machine Selection Part is read from left to right. Then, each gene integer will be transferred to machine matrix and time matrix depending on processing time table. For instance, Fig. 2 could be transferred to machine matrix (1) and time matrix (2) depending on Table 1. Rows in each matrix correspond to jobs, and columns correspond

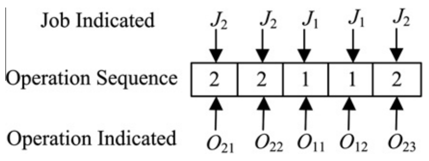

Job Indicated
J_2 J_2 J_1 J_1 J_2
Operation Sequence
2 2 1 1 2
Operation Indicated
O_{21} O_{22} O_{11} O_{12} O_{23}

Fig. 3. Operation Sequence part.

to operations of each job. For example, in machine matrix the integer 4 denotes operation O_{11} which is processed on M_4 and the corresponding processing time is 3 in time matrix.

$$Machine = \begin{bmatrix} 4 & 2 & \\ 3 & 2 & 5 \end{bmatrix}. \quad (1)$$

$$Time = \begin{bmatrix} 3 & 8 & \\ 6 & 6 & 8 \end{bmatrix}. \quad (2)$$

Step 2: Operation Sequence Part is also read from left to right. The selected machines and processing time correspond to machine matrix and time matrix respectively in *Step* 1.

(a) Decode each integer to the corresponding operation O_{ij};

(b) Refer to machine matrix and time matrix to obtain the selected machine M_k and processing time P_{ijk};

(c) Let $O_{i(j+1)}$ be processed on machine M_x and $P_{i(j+1)x}$ its processing time. Let T_x be the end time of the last operation on M_x. Then find all time interval on M_x, i.e., get an idle time interval $[S_x, E_x]$ beginning from S_x and ending at E_x on M_x. Because of precedence constraints among operations of the same job, operation $O_{i(j+1)}$ could only be started when its immediate job predecessor O_{ij} has been completed. We could get the earliest starting process time t_b of operation $O_{i(j+1)}$ according to Eq. (3);

$$t_b = \max\{E_{ijk}, S_x\}. \quad (3)$$

(d) According to the Eq. (4), the time interval $[S_x, E_x]$ is available for $O_{i(j+1)}$ if there is enough time span from the starting of $O_{i(j+1)}$ until the end of the interval to complete it. As can be seen from Fig. 4, if Eq. (4) is true, assign $O_{i(j+1)}$ to M_x starting at t_b (Fig. 4(a)). Otherwise, operation $O_{i(j+1)}$ is allocated at the end of the last operation on M_x (i.e. it starts at T_x) (Fig. 4(b)).

$$\max\{E_{ijk}, S_x\} + P_{i(j+1)x} \le E_x. \quad (4)$$

(e) Go to (c) until each operation of Operation Sequence Part is processed on corresponding machine;

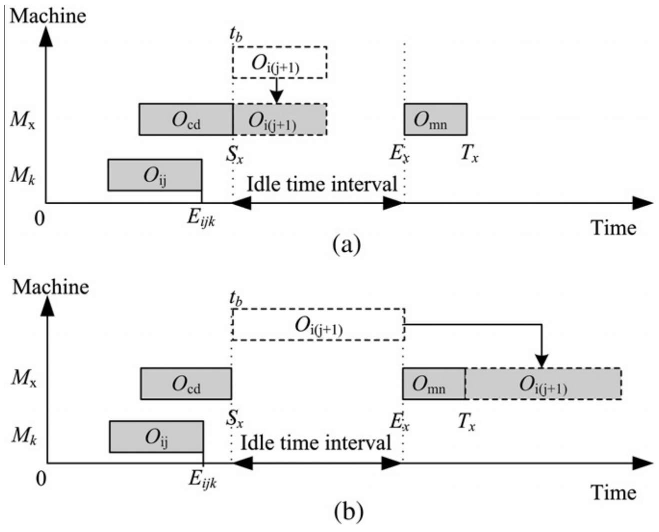


Fig. 4. Finding enough interval and inserting $Oi_{(j+1)}$.

4.3. Initial Population

Population initialization is a crucial task in evolutionary algorithms because it can affect the convergence speed and the quality of the final solution (Shahryar, Hamid, & Magdy, 2007). In this section, we mainly present two methods to solve the first sub-problem through assigning each operation to the suitable machine. These methods take into account both the processing time and the workload of the machines.

4.3.1. Global Selection (GS)

We define that a stage is the process of selecting a suitable machine for an operation. Thus this method records the sum of the processing time of each machine in the whole processing stage. Then the machine which has the minimum processing time in every stage is selected. In particular, the first job and next job are randomly selected. Detailed steps are as follows:

Step 1: Create a new array to record all machines' processing time, initialize each element to 0;
Step 2: Select a job randomly and insure one job to be selected only once, then select the first operation of the job;
Step 3: Add the processing time of each machine in the available machines and the corresponding machine's time in the time array together;
Step 4: Compare the added time to find the shortest time, then select the index k of the machine which has the shortest time. If there is the same time among different machines, a machine is selected randomly among them;
Step 5: Set the allele which corresponds to the current operation in the MS part to k;
Step 6: Add the current selected machine's processing time and its corresponding allele in the time array together in order to update the time array;
Step 7: Select the next operation of the current job, and execute Step 3 to *Step* 6 until all operations of the current job are selected, then go to Step 8;
Step 8: Go to step 2 until all jobs are all selected once.

The implementation of GS is given in Fig. 5. We assume that the first selected job is J_1, and the next job is J_2. From Fig. 5, we easily see that the processing time on M_1 is the shortest in the alternative machine set of operation O_{11}. So the machine M_1 is selected to process the operation O_{11} of job J_1, and set corresponding allele in MS to the index of M_1. Then the processing time is added to the corresponding position in time array. Finally, the selected machines of all operations may be M_1-M_4-M_1-M_3-M_2, and the corresponding chromosome representation is 1-2-1-3-1.

4.3.2. Local Selection (LS)

This method is different from GS, it records the processing time of machines when a single job has been processed. So it records the stages which belong to a single job instead of the whole stages. Then the machine which has the minimum processing time in current stage is selected. Detailed procedures of LS are as follows.

Step 1: In order to record all machines' processing time, create a new array (called time array), the length equals to L, and set each element 0;
Step 2: Select the first job, and its first operation;
Step 3: Set each allele 0 in the array;
Step 4: Add the processing time of each machine in the alternative machine set and the corresponding machines' time in the array together;

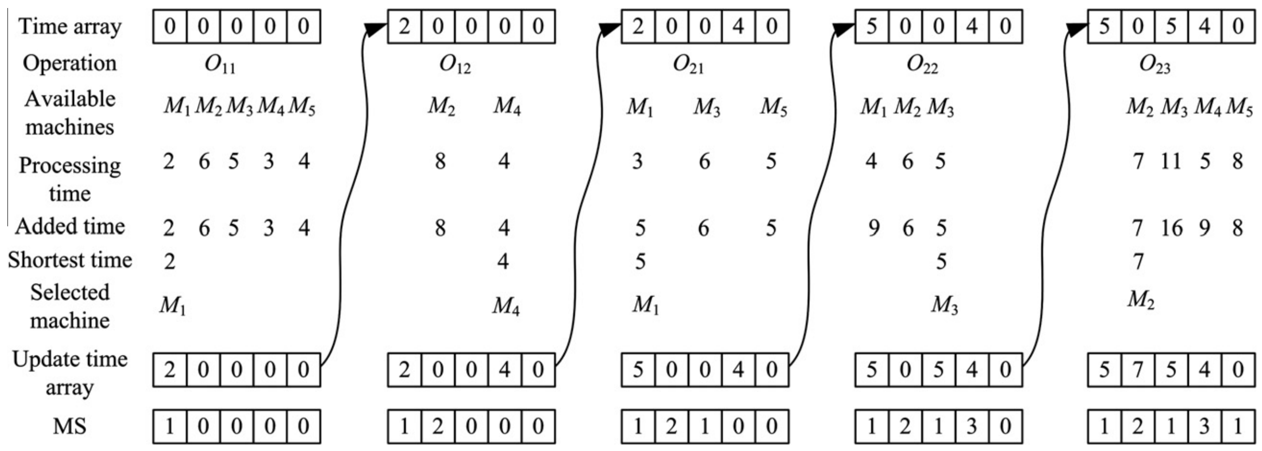


Fig. 5. The process of GS.

Step 5: Compare the added time to find the index k of the machine which has shortest time. If there is the same time among different machines, a machine is randomly selected among them;
Step 6: Set the allele which corresponds to the current operation in the MS part to k;
Step 7: Add the current selected machine's processing time and its corresponding allele in the time array together to update the time array;
Step 8: Select the next operation of the current job, and go to *Step* 4 until all the operations of the current job are selected, then go to *Step* 9;
Step 9: Select the next job, and select the first operation of the current job;
Step 10: Go to *Step* 3 until all jobs are selected once.

We also take the data in Table 1 for instance. The implementation of LS is given in Fig. 6. We assume that the first selected job is J_1, and the next job is J_2. From Fig. 6, we easily see that the processing time on M_1 is the shortest in the alternative machine set of operation O_{11}. So the machine M_1 is selected to process the operation O_{11} of job J_1. Then the processing time is added to the corresponding position in time array. When all operations of job J_1 have been arranged on the suitable machine, each element in the time array is set to 0. Then the next job J_2 is executed in the same way. Finally, the selected machines of all operations may be M_1-M_4-M_1-M_3-M_4, and the corresponding chromosome representation is 1-2-1-3-3. From above, we can see that the difference between GS and LS is that the updating style of the time array, i.e., the time array in GS records all the time of all jobs, however, the time array in LS records the time of each job.

In addition, we could also generate initial assignments by randomly selecting a machine in alternative machine set of each operation. In practice, we mix the two methods with Random Selection (RS) to generate initial population. The advantage of using GS is that it could find different initial assignments in different runs of the algorithm, better exploring the search space and considering the workload of machines. LS could find the machine which has shortest processing time machine in alternative machine set of each job. Without loss of generality and enhancing randomicity, we adopt randomly selecting machine to generate initial assignments. For example, 60% of initial population could be generated by GS, 30% by LS, and 10% by RS.

Once the assignments are settled, we have to determine how to sequence the operations on the machines. Obviously, the scheduling is feasible if it respects the precedence constraints among operations of the same job, i.e., operation O_{ij} cannot be processed before operation $O_{i(j+1)}$.

4.4. Selection operator

Choosing individuals for reproduction is the task of selection. The chosen individuals are moved into a mating pool. They could reproduce once or more times. The roulette wheel selection was

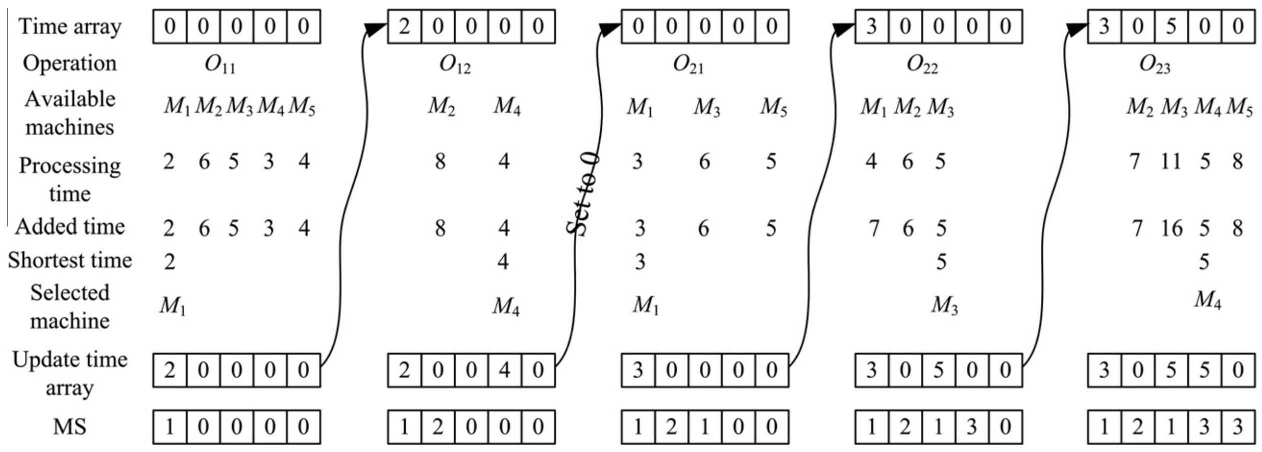


Fig. 6. The process of LS.

popular in the past, which is a fitness-based approach. Therefore, the objective value that is achieved by decoding chromosome must be repaired. The parameters for repairing may influence the chosen results. Tournament approach could overcome the scaling problem of the direct fitness-based approach, and make good individuals have more "survival" opportunity. At the same time, it only adapts the relative value of the fitness as the criteria to select the individuals instead of using the proportion of the fitness directly. So it can avoid both the influence of the "super individual" and premature convergence. In this paper, tournament approach is adopted. Three individuals are randomly chosen from parent population, and three objective values are compared to select the best individual to move into the mating pool.

4.5. Crossover operator

The goal of the crossover is to obtain better chromosomes to improve the result by exchanging information contained in the current good ones. In our work we carried out two kinds of crossover operator for the chromosomes.

4.5.1. Machine Selection part

The crossover operation of MS is performed on two Machine Selection parts and generates two new Machine Selection parts each of which corresponds to a new allocation of operations to machines. Each machine of the new Machine Selection parts must be effective, i.e., the machine must be included in the alternative machine set of the corresponding operation. We adopt two different crossover operators.

The first crossover operator is the two-point crossover. The detailed process of two-point crossover was described in Watanabe, Ida, and Gen (2005). And the second crossover operator is uniform crossover (Gao, Sun & Gen, 2008). MS crossover operator only changes some alleles, while their location in each individual i.e., their preceding constraints are not changed. Therefore, the individuals after crossover are also feasible. The procedure could be illustrated in Fig. 7.

4.5.2. Operation Sequence part

The crossover operation of OS is different from that of MS. During the past decades, several crossover operators have been proposed for permutation representation, such as order crossover, partial mapped crossover, and so on. Here we apply a Precedence preserving order-based crossover (POX) for the Operation Sequence (Lee, Yamakawa, & Lee, 1998). Detailed implementing steps of POX are as follows:

Step 1: Generate two sub-job set $Js1/Js2$ from all jobs and select two parent individuals as $p1$ and $p2$ randomly;
Step 2: Copy any allele in $p1/\ p2$ that belong to $Js1/Js2$ into two child individuals $c1/c2$, and retain in the same position in $c1/c2$;
Step 3: Delete the alleles that are already in the sub-job $Js1/Js2$ from $p2/\ p1$;
Step 4: Orderly fill the empty position in $c1/c2$ with the alleles of $p2/p1$ that belong to in their previous sequence.

In Table 1, there are only two jobs. So it is difficult to present the process of POX clearly. In Fig. 8 we use five jobs to illustrate the procedure of generating two child individuals.

4.6. Mutation operator

Mutation introduces some extra variability into the population to enhance the diversity of population. Usually, mutation is applied with small probability. Large probability may destroy the good chromosome.

4.6.1. Machine Selection part

MS mutation operator only changes the assignment property of the chromosomes. We select the shortest processing time from alternative machine set to balance the workload of the machines.

Taking the chromosome from Fig. 1 for example, MS mutation is described as follows:

Step 1: Select one individual from the population;
Step 2: Read the chromosomes of the individual from left to right and generate a probability value randomly; if all the chromosomes have been read, then end the procedure;
Step 3: If the probability value is less than or equal to the mutation probability then go to Step 4; otherwise, go to Step 2;
Step 4: Select the shortest processing time from the alternative machine set and assign it to the mutation position;

An illustrative instance is shown in Fig. 9. Suppose the mutative operation is O23, before the mutation, O23 is processed on M5, which is the forth machine in the alternative machine set, so the allele is 4. In the mutation, the rule that selecting the machine of the shortest processing time is obeyed, so M4 is selected, and the allele in the chromosome changes into 3.

4.6.2. Operation Sequence part

The OS mutation probability is the same as the MS mutation probability. If one position in the OS chromosome is to be mutated,

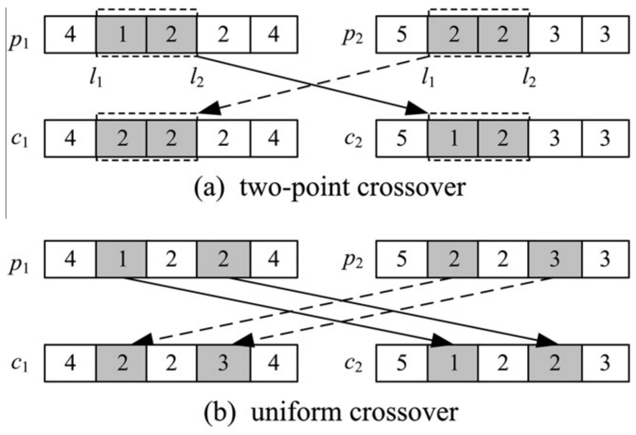
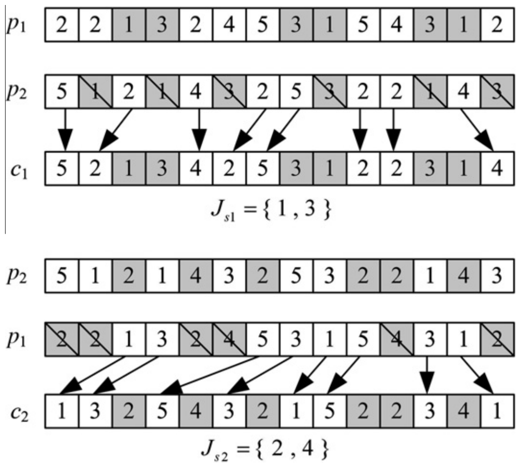
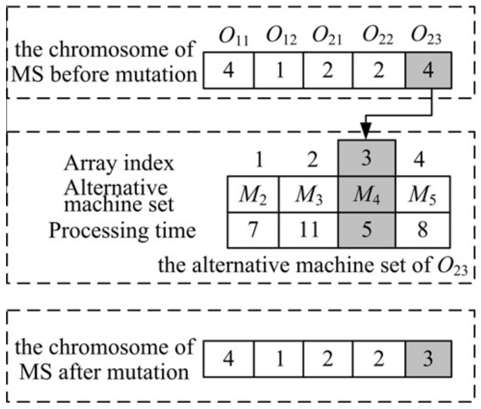


Fig. 7. MS crossover operator.



p_1 2 2 1 3 2 4 5 3 1 5 4 3 1 2
p_2 5 1 2 1 4 3 2 5 3 2 2 1 4 3
c_1 5 2 1 3 4 2 5 3 1 2 2 3 1 4
$J_{s1} = \{1, 3\}$
p_2 5 1 2 1 4 3 2 5 3 2 2 1 4 3
p_1 2 2 1 3 2 4 5 3 1 5 4 3 1 2
c_2 1 3 2 5 4 3 2 1 5 2 2 3 4 1
$J_{s2} = \{2, 4\}$

Fig. 8. OS crossover operator.



O_{11} O_{12} O_{21} O_{22} O_{23}
the chromosome of MS before mutation
4 1 2 2 4
Array index
1 2 3 4
Alternative machine set
M_2 M_3 M_4 M_5
Processing time
7 11 5 8
the alternative machine set of O_{23}
the chromosome of MS after mutation
4 1 2 2 3

Fig. 9. MS mutation operator.

then another position is randomly generated in $[0, L-1]$ to exchange with it. According to our chromosome representation, the new individual is a feasible solution all the same.

4.7. Framework of the effective GA

The framework of the proposed genetic algorithm is illustrated in Fig. 10.

Each individual in the initial population of MS part is generated by GS, LS and RS. This procedure makes it possible to assign each operation to the suitable machine by taking into account the processing time and workload of machines. And each individual in the initial population of OS part is randomly generated according to the chromosome representation in Section 4.1. The adoption of MSOS improved the chromosome representation and it could generate feasible schedules and avoid repair mechanism. The proposed effective genetic algorithm terminates when a maximal number of iteration is reached, and the best individual, together with the corresponding schedule, is output.

5. Computational results

The proposed effective genetic algorithm (eGA) was implemented in C++ on a Pentium IV running at 1.8 GHz and tested on a large number of instances from the literature. Test problems include both the P-FJSP and the T-FJSP. We know the P-FJSP is more complex than the T-FJSP from above, when considering the search space and the computational cost, the approach for solving it is very important. However, in our experiments, the approach matches the P-FJSP well because of the mechanism of adopting real-coded based on alternative machine set in chromosome representation. Moreover, it can maintain feasibility without any mechanism. We compare our computational results with results obtained by other researchers.

Firstly, we tested the performance of the initialization method (GS + LS + RS), we selected Mk04 test problem with 15 jobs and eight machines randomly from Brandimarte (1993). The results are shown in Table 2 and Fig. 11. So we could see that GS + LS + RS could improve the quality of initial efficiently.

In Fig. 12 we draw the decrease of the average best makespan over five runs for the Mk04 problem with 15 jobs and eight machines initialized by GS + LS + RS and RS respectively. Compared the results by the two initial methods, the two broken lines have a gap before the generation 84, GS + LS + RS is the lower one, and it could achieve the optimal solution at Generation 40 while Random could gain the same optimal solution until Generation 84. So we could obviously know that GS and LS are effective methods for generating high-quality initial population, they could decrease the average best makespan a lot in the whole computational stage.

In order to obtain meaningful results, we ran our algorithm five times on the same instance. The parameters used in the effective GA are chosen experimentally in order to get a satisfactory solution in an acceptable time span. According to the complexity of the problems, the population size of the effective GA ranges from 50 to 300. Through experimentation, other parameter values were chosen as follows:

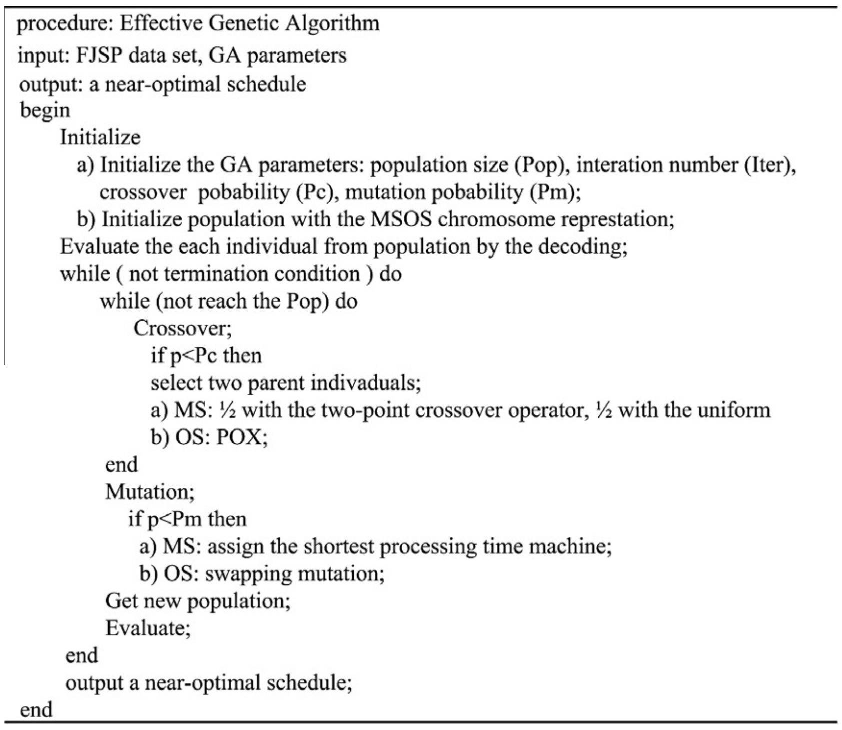

```
procedure: Effective Genetic Algorithm
input: FJSP data set, GA parameters
output: a near-optimal schedule
begin
    Initialize
      a) Initialize the GA parameters: population size (Pop), interation number (Iter),
         crossover pobability (Pc), mutation pobability (Pm);
      b) Initialize population with the MSOS chromosome represtation;
    Evaluate the each individual from population by the decoding;
    while ( not termination condition ) do
        while (not reach the Pop) do
            Crossover;
              if p<Pc then
              select two parent indivaduals;
              a) MS: ½ with the two-point crossover operator, ½ with the uniform
              b) OS: POX;
        end
        Mutation;
           if p<Pm then
             a) MS: assign the shortest processing time machine;
             b) OS: swapping mutation;
        Get new population;
        Evaluate;
    end
    output a near-optimal schedule;
end
```

Fig. 10. Framework of the proposed effective GA.

Table 2
The comparison of the initialization methods.

Population size	GS + LS + RS		RS	
	Average of best makespan	Average of makespan	Average of best makespan	Average of makespan
50	75.0	91.7	93.6	116.0
100	73.2	92.8	88.8	116.5
150	73.0	92.8	88.4	115.9
200	73.4	92.1	87.6	115.9
250	73.2	92.5	88.2	116.3
300	72.8	92.8	87.0	116.3
350	72.8	92.4	87.0	115.5
400	73.8	92.6	87.8	115.5
450	71.4	92.5	86.4	115.2
500	70.8	92.4	86.8	115.5

number of generation: 100,
rate of initial assignments with GS: 0.6,
rate of initial assignments with LS: 0.3,
rate of initial assignments with RS: 0.1,
crossover probability: 0.7,
rate of two-point crossover in MS: 0.5,
rate of uniform crossover in MS: 0.5,
mutation probability: 0.01.

The proposed effective GA is firstly tested from Brandimarte's data set (BRdata) (Brandimarte, 1993). The data set consists of ten problems with number of jobs ranging from 10 to 20, number of machines ranging from 4 to 15 and number of operations for each job ranging from 5 to 15.

The second data set is a set of 21 problems from Barnes and Chambers (1996) (BCdata). The data set were constructed from the three most challenging classical Job-shop scheduling problems (mt 10, la24, la40) Fisher & Thompson, 1963; Lawrence, 1984 by replicating machine. The processing times for operations on replicated machines are assumed to be identical to the original. The number of jobs ranges from 10 to 15, the number of machines ranges from 11 to 18, and the number of operations for each job ranges from 10 to 15.

The third data set is a set of 18 problems from Dauzère-Pérès (1997) (DPdata). The number of jobs ranges from 10 to 20, the number of machines ranges from 5 to 10, and the number of operations

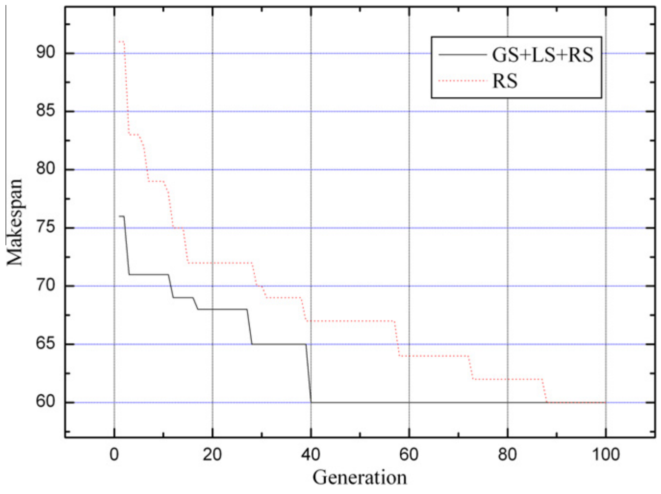


Fig. 12. Decreasing of the makespan (Mk04).

for each job ranges from 15 to 25. The set of machines capable of performing an operation was constructed by letting a machine be in that set with a probability that ranges from 0.1 to 0.5.

In the Tables 3–5, $n \times m$ denotes the jobs and machines about the each instance. *Flex.* denotes the average number of equivalent machines per operation. T_o denotes the total number of all operations of all jobs. (LB, UB) denotes the optimum makespan if known, otherwise, the best lower and upper bound found to date. C_m denotes makespan time. * indicates the best known solution to date. AV(C_m) stands for the average makespan out of five runs. *t* denotes the average running time over five runs. M&G is the approach proposed by Mastrolilli and Gambardella (2000).

From Table 3, the compared computational results show that the speed of searching for optimal solution by our algorithm is very fast so far. Among the 10 test problems, Mk03 and Mk08 could get the optimal solution in the first generation by using eGA. Eight problems out of the 10 problems could gain the same good results as M&G. Compared with GENACE, eGA could get better solutions of 8 problems.

From Tables 4 and 5, totally, we found new better 33 solutions in terms of best solutions out of five runs in the 39 test problems. And, the average makespan of the eGA over five runs is better than

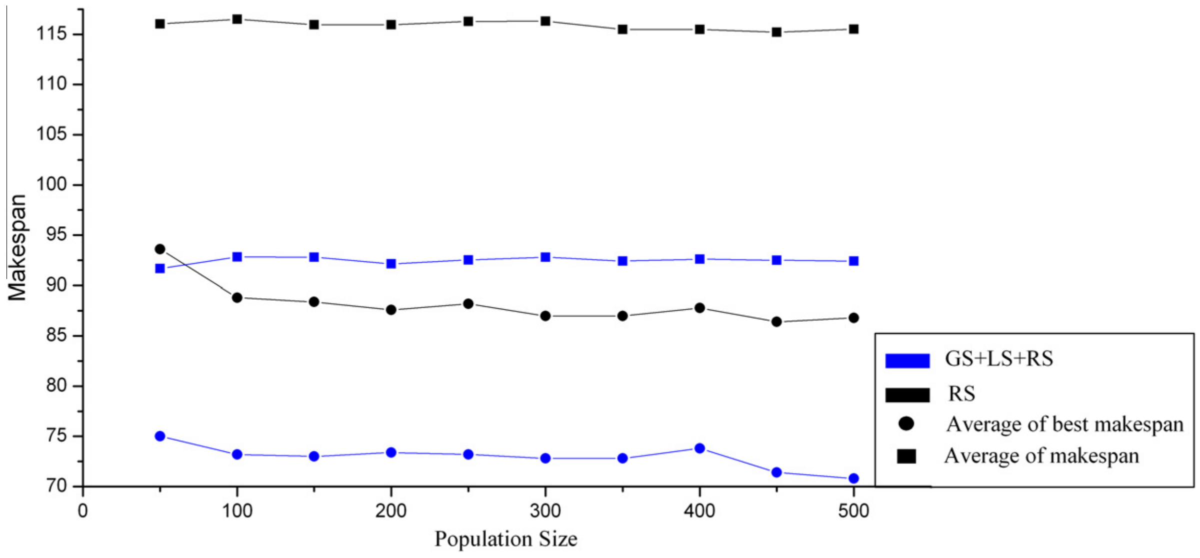


Fig. 11. The comparison of the initialization methods.

Table 3
Results of BRdata.

Problem	$n \times m$	T_o	*Flex.*	LB, UB	M&G		GENACE			Proposed eGA			
					C_m	AV(C_m)	Pop	C_m	t	Pop	C_m	AV(C_m)	t
mk01	10 × 6	55	2.09	36, 42	40*	40.0	200	40	3.2	100	40*	40	1.6
mk02	10 × 6	58	4.01	24, 32	26*	26.0	200	32	3.6	300	26*	26	2.6
mk03	15 × 8	150	3.01	204, 211	204*	204.0	N/A	N/A	N/A	50	204*	204	1.3
mk04	15 × 8	90	1.91	48, 81	60*	60.0	200	67	6.1	100	60*	60	6.2
mk05	15 × 4	106	1.71	168, 186	173*	173.0	200	176	7.2	200	173*	173	7.3
mk06	10 × 15	150	3.27	33, 86	58*	58.4	200	67	10.7	200	58*	58	15.7
mk07	20 × 5	100	2.83	133, 157	144*	147.0	200	147	6.9	200	144*	145	17.3
mk08	20 × 10	225	1.43	523	523*	523.0	200	523	19.1	50	523*	523	2.2
mk09	20 × 10	240	2.53	299, 369	307*	307.0	200	320	20.4	300	307*	307	30.2
mk10	20 × 15	240	2.98	165, 296	198*	199.2	200	229	27.6	300	198*	199	36.6

Table 4
Results of BCdata.

Problem	$n \times m$	T_o	*Flex.*	LB, UB	M&G		Proposed eGA			
					C_m	AV(C_{9uu})	Pop	C_m	AV(C_m)	t
mt10c1	10 × 11	100	1.10	655, 927	928	928	200	927*	928	23.25
mt10cc	10 × 12	100	1.20	655, 914	910*	910	200	910*	910	19.27
mt10x	10 × 11	100	1.10	655, 929	918*	918	1000	918*	918	21.45
mt10xx	10 × 12	100	1.20	655, 929	918*	918	1000	918*	918	20.38
mt10xxx	10 × 13	100	1.30	655, 936	918*	918	1000	918*	918	25.39
mt10xy	10 × 12	100	1.20	655, 913	906	906	300	905*	906	24.37
mt10xyz	10 × 13	100	1.30	655, 849	847*	850	1000	847*	847	30.24
setb4c9	15 × 11	150	1.10	857, 924	919	919.2	1000	914*	914	12.81
setb4cc	15 × 12	150	1.20	857, 909	909*	911.6	1000	909*	910	20.16
setb4x	15 × 11	150	1.10	846, 937	925*	925	200	925*	925	8.92
setb4xx	15 × 12	150	1.20	847, 930	925*	926.4	300	925*	925	54.06
setb4xxx	15 × 13	150	1.30	846, 925	925*	925	1000	925*	925	62.81
setb4xy	15 × 12	150	1.20	845, 924	916*	916	1000	916*	916	27.78
setb4xyz	15 × 13	150	1.30	838, 914	905*	908.2	1000	905*	908.1	40.26
seti5c12	15 × 16	225	1.07	1027, 1185	1174*	1174.2	1000	1174*	1174	70.69
seti5cc	15 × 17	225	1.13	955, 1136	1136*	1136.4	1000	1136*	1136.2	69.53
seti5x	15 × 16	225	1.07	955, 1218	1201*	1203.6	1000	1209	1209	67.56
seti5xx	15 × 17	225	1.13	955, 1204	1199*	1200.6	1000	1204	1204	78.29
seti5xxx	15 × 18	225	1.20	955, 1213	1197*	1198.4	1000	1204	1204	105.25
seti5xy	15 × 17	225	1.13	955, 1148	1136*	1136.4	1000	1136*	1136.3	70.47
seti5xyz	15 × 18	225	1.20	955, 1127	1125*	1126.6	1000	1125*	1126.5	70.56

Table 5
Results of DPdata.

Problem	$n{\times}m$	T_o	*Flex.*	LB, UB	M&G		Proposed eGA			
					C_m	AV(C_m)	Pop	C_m	AV(C_m)	t
01a	10 × 5	100	1.13	2505, 2530	2518	2528	300	2516*	2518	98.35
02a	10 × 5	100	1.69	2228, 2244	2231*	2234	1000	2231*	2231	164.75
03a	10 × 5	100	2.56	2228, 2235	2229*	2229.6	1000	2232	2232.3	112.02
04a	10 × 5	100	1.13	2503, 2565	2503*	2516.2	300	2515	2515	109.26
05a	10 × 5	100	1.69	2189, 2229	2216	2220	300	2208*	2210	154.89
06a	10 × 5	100	2.56	2162, 2216	2203	2206.4	500	2174*	2175	133.05
07a	15 × 8	100	1.24	2187, 2408	2283	2297.6	200	2217*	2218.4	396.44
08a	15 × 8	150	2.42	2061, 2093	2069*	2071.4	500	2073	2073	388.53
09a	15 × 8	150	4.03	2061, 2074	2066*	2067.4	300	2066*	2066	390.22
10a	15 × 8	150	1.24	2178, 2362	2291	2305.6	500	2189*	2191	373.32
11a	15 × 8	150	2.42	2017, 2078	2063*	2065.6	300	2063*	2065	405.95
12a	15 × 8	150	4.03	1969, 2047	2034	2038	1000	2019*	2022	332.44
13a	20 × 10	150	1.34	2161, 2302	2260	2266.2	500	2194*	2194	522.77
14a	20 × 10	150	2.99	2161, 2183	2167	2168	200	2167*	2168.2	555.15
15a	20 × 10	225	5.02	2161, 2171	2167	2167.2	1000	2165*	2166	765.48
16a	20 × 10	225	1.34	2148, 2301	2255	2258.8	500	2211*	2212.6	494.16
17a	20 × 10	225	2.99	2088, 2168	2141	2144	200	2109*	2110	645.38
18a	20 × 10	225	5.02	2057, 2139	2137	2140.2	300	2089*	2089	735.13

of M&G on 34 test problems. The running time of the eGA is also much shorter than that of M&G. Obviously, our proposed algorithm is effective and efficient. And, the Fig. 13 shows the Gantt chart of optimal solution of seti5xy test problem.

6. Conclusions and future study

In this paper, we proposed an effective genetic algorithm for solving the flexible job-shop scheduling problem (FJSP). An im-

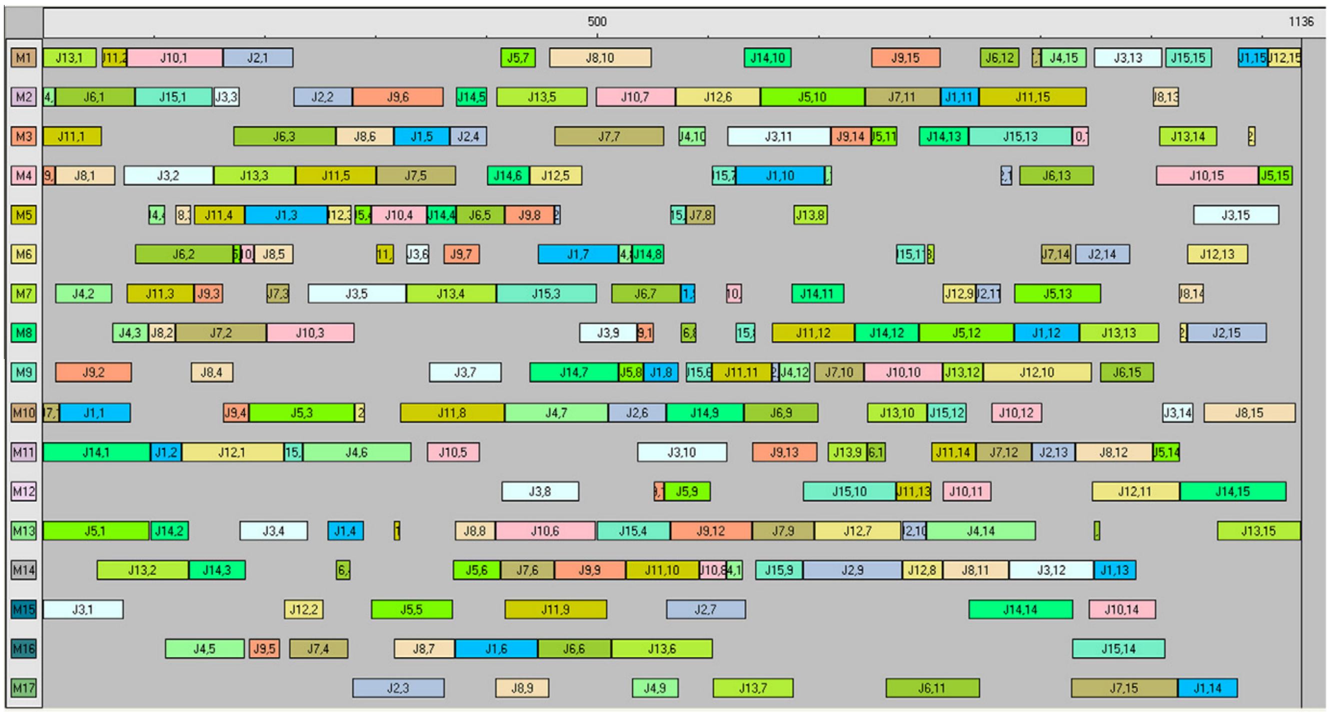


Fig. 13. The Gantt chart of seti5xy.

proved chromosome representation scheme is proposed and an effective decoding method interpreting each chromosome into a feasible active schedule is designed. In order to enhance the quality of initial solution, a new initial assignment method (GS + LS + RS) is designed to generate high-quality initial population integrating different strategies to improve the convergence speed and the quality of final solutions. Then different strategies for selection, crossover and mutation operator are adopted. This makes it possible to solve the problem of trade-off resources allocation.

Some benchmark problems taken from other literature are solved. The computational results show that the proposed effective genetic algorithm leads to the same level or even better results in computational time and quality compared with other genetic algorithms. These results not only prove that the proposed method is effective and efficient for solving flexible job-shop scheduling problem, but also demonstrate the initialization method is crucial in genetic algorithm. It can enhance the convergence speed and the quality of the solution, so we expect to gain more insights into the research on the effective initialization method.

In the future, it will be interesting to investigate the following issues:

Adjust the proportion of the GS, LS and RS used in the initialization in order to generate better results. Maintain the best individual of the mating pool in a successive generation of chromosomes. Combine with the good local search algorithm to enhance the capability of Global Selection and Local Selection.

Acknowledgements

This project is supported by 863 High Technology Plan Foundation of China under Grant No. 2006AA04Z131, the National Natural Science Foundation of China No. 50825503 and Program for New Century Excellent Talents in University under Grant No. NCET-08-0232. And we wish to thank the anonymous referees for their constructive and useful comments.

References

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.

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 1996; Technical report series: ORP96-09.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by taboo search. *Annals of Operations Research, 41*, 157–183.

Brucker, P., & Schile, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing, 45*(4), 369–375.

Chen, H., Ihlow, J., & Lehmann, C. (1999). A genetic algorithm for flexible job-shop scheduling. In *IEEE international conference on robotics and automation, Detroit 1999* (Vol. 2, pp. 1120–1125).

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

Fisher, H., & Thompson, G. L. (1963). *Probabilistic learning combinations of local job shop scheduling rules*. Englewood Cliffs, NJ: Prentice-Hall. pp. 225–251.

Gao, J., Sun, L. Y., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computer and Operations Research, 35*(9), 2892–2907.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*, 117–129.

Ho, N. B., Tay, J. C., Edmund, M., & Lai, K. (2007). An effective architecture for learning and evolving flexible job shop schedules. *European Journal of Operational Research, 179*, 316–333.

Hurink, E., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machines. *Operations Research Spectrum, 15*, 205–215.

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*, 351–362.

Kacem, I. (2003). Genetic algorithm for the flexible job-shop scheduling problem. *IEEE International conference on Systems, Man and Cybernetics, 4*, 3464–3469.

Kacem, I., Hammadi, S., & Borne, P. (2002). Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation, 60*, 245–276.

Kacem, I., Hammadi, S., & Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man and Cybernetics, 32*(1), 1–13.

Lawrence, S. (1984). *Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*. Pittsburgh, PA: GSIA, Carnegie Mellon University.

Lee, K. M., Yamakawa, T., & Lee, K. M. (1998). A genetic algorithm for general machine scheduling problems. *International Journal of Knowledge-Based Electronic, 2*, 60–66.

Liu, H. B., Abraham, A., & Grosan, C. (2007). A novel variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. In *Second international conference on digital information management, 2007. ICDIM '07* (pp. 138–145).

Mastrolilli, M., & Gambardella, L. M. (1996). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling, 3*, 3–20.

Mastrolilli, M., & Gambardella, L. M. (2000). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling, 3*(1), 3–20.

Mesghouni, K., Hammadi, S., & Borne, P. (1997). Evolution programs for job-shop scheduling. In *Proceedings of the IEEE international conference on computational cybernetics and simulation* (Vol. 1, pp. 720–725).

Najid, N. M., Dauzère-Pérès, S., & Zaidat, A. (2002). A modified simulated annealing method for flexible job shop scheduling problem. *IEEE International conference on Systems, Man and Cybernetics, 5*, 6–12.

Paulli, J. (1995). A hierarchical approach for the FMS scheduling problem. *European Journal of Operational Research, 86*(1), 32–42.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2007). A genetic algorithm for the flexible job-shop scheduling problem. *Computers and Operations Research, 35*(10), 3202–3212.

Pinedo, M. (2002). *Scheduling theory, algorithms, and systems*. Englewood Cliffs, NJ: Prentice-Hall.

Shahryar, R., Hamid, R. T., & Magdy, M. A. S. (2007). A novel population initialization method for accelerating evolutionary algorithms. *Computers and Mathematics with Application, 53*, 1605–1614.

Tay, J. C., & Wibowo, D. (2004). An effective chromosome representation for evolving flexible job shop schedules, GECCO 2004. *Lecture notes in computer science* (Vol. 3103, pp. 210–221). Berlin: Springer.

Watanabe, M., Ida, K., & Gen, M. (2005). A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. *Computers and Industrial Engineering, 48*, 743–752.

Yang, J. B. (2001). GA-based discrete dynamic programming approach for scheduling in FMS environments. *IEEE Transaction on Systems, Man, and Cybernetics, Part B, 31*(5), 824–835.

Zhang, H. P., & Gen, M. (2005). Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Journal of Complexity International, 48*, 409–425.