Int J Adv Manuf Technol (2010) 49:253–262
DOI 10.1007/s00170-009-2379-y

ORIGINAL ARTICLE

Solving fuzzy job shop scheduling problems using random key genetic algorithm

Deming Lei

Received: 26 July 2008 / Accepted: 13 October 2009 / Published online: 31 October 2009

Abstract This paper addresses job shop scheduling problems with fuzzy processing time and fuzzy trapezoid or doublet due date. An efficient random key genetic algorithm (RKGA) is suggested to maximize the minimum agreement index and to minimize the maximum fuzzy completion time. In RKGA, a random key representation and a new decoding strategy are proposed and *two-point crossover* (TPX) and *discrete crossover* (DX) are considered. RKGA is applied to some fuzzy scheduling instances and performance analyses on random key representation, and the comparison between RKGA and other algorithms are done. Computation results validate the effectiveness of random key representation and the promising advantage of RKGA on fuzzy scheduling.

Keywords Job shop scheduling · Fuzzy processing time · Random key representation · Genetic algorithm

1 Introduction

In general, the various factors of the job shop scheduling problem (JSSP) are treated as crisp value. However, this assumption is not realistic in many cases. In order to reflect the real-world situations, it may be more appropriate to consider fuzzy processing time due to man-made factors and fuzzy due date tolerating a certain amount of delay in due date.

In the past decade, some results have been obtained for fuzzy job shop scheduling problem (FJSSP). Kuroda and Wang [1] discussed the static JSSP and the dynamic JSSP with fuzzy information. A branch-and-bound algorithm is used to solve the static JSSP and the methods for the dynamic JSSP are also considered. Sakawa and Mori [2] presented an efficient genetic algorithm (GA) by incorporating the concept of similarity among individuals. Based on the work of Sakawa and Mori [2], Sakawa and Kubota [3] presented a GA for multiobjective JSSP with fuzzy processing time and fuzzy due date. The objective is to maximize the minimum agreement index, to maximize the average agreement index, and to minimize the maximum fuzzy completion time. Song et al. [4] presented a combined strategy of GA and ant colony optimization. They also designed a new neighborhood search method and an improved tabu search to improve the local search ability of the hybrid algorithm. Lei [5] proposed an efficient Pareto archive particle swarm optimization. FJSSP is first converted into a continuous optimization problem and then the proposed algorithm is used to obtain a set of Pareto optimal solutions of the continuous problems with three objectives. Li et al. [6] proposed a GA for FJSSP with alternative machines by adopting two-chromosome presentation and the extended version of the Giffler–Thompson (GT) procedure [7]. Wu et al. [8] designed the fuzzy shifting bottleneck procedure by combing fuzzy ranking method and shifting bottleneck procedure and applied the fuzzy procedure to FJSSP.

GA is an evolutionary computation method based on the genetic process of biological organisms. Before a GA is applied, a suitable representation and the corresponding decoding method must be determined. In fuzzy scheduling context, several representations have been suggested. The first representation is the matrix method, which uses a $3n \times m$ matrix of fuzzy completion time to represent a schedule of the problems with n jobs and m machines [2]. Lei [5] presented a priority rule-based representation

D. Lei (✉)
School of Automation, Wuhan University of Technology,
122 Luoshi Road,
Wuhan, Hubei Province, People's Republic of China
e-mail: deminglei11@163.com

method in which the chromosome consists of $n \times m$ genes and each gene corresponds to a fuzzy priority rule. The operation-based representation is also adopted [4, 6].

The existing representations of FJSSP have some unsatisfactory features, such as the complex structure of matrix representation and the special genetic operators including order crossover required for the operation-based representation. With respect to decoding, a schedule can be obtained by using the ordered operation list or job permutation or using the GT procedure in the deterministic context; however, in fuzzy case, the decoding strategy is not investigated fully and only the GT procedure-based strategy is used.

In this study, a random key scheduling algorithm is presented, which uses a random key representation and the extended version of the first decoding strategy. The new representation has a simple structure and does not require special genetic operators for producing new solutions. The random key genetic algorithm (RKGA) is tested and compared with the GA with the operation-based representation and other GA from literature. The results show good performance of RKGA.

The remainder of the paper is organized as follows. The problem formulation and the operations on fuzzy processing time are introduced in Section 2. Section 3 describes a random key scheduling algorithm. Numerical test experiments on the proposed algorithm are reported in Section 4 and the conclusions are summarized in the final section.

2 Fuzzy job shop scheduling

2.1 Problem description

The $n \times m$ FJSSP can be described as follows: given n jobs J_i and m machines M_j, $i=1, 2, \ldots, n$, $j=1, 2, \ldots, m$, each job is composed of several operations and operation o_{ij} indicates the jth operation of job J_i. The processing time of operation o_{ij} is represented as a triangular fuzzy number (TFN) $p_{ij} = \left(a_{ij}^1, a_{ij}^2, a_{ij}^3\right)$. For job J_i, doublet due date $\left(d_i^1, d_i^2\right)$ and trapezoid due date $d_i = \left(e_i^1, e_i^2, d_i^1, d_i^2\right)$ are considered. Other constraints of JSSP are still suitable to FJSSP. For instance, it is assumed that only one operation can be processed on each machine at a time and each operation cannot be commenced if the precedent operation is still being processed.

In the deterministic context, tardiness or earliness are used to describe the grade of the satisfaction of the customer for delivery. The agreement index can be regarded as the extended version of the above objective in the fuzzy case. The agreement index AI_i of job J_i is defined as follows:

$$\mathrm{AI}_i = \mathrm{area}(C_i \cap d_i)/\mathrm{area}(C_i) \tag{1}$$

where the fuzzy completion time of job J_i is expressed as TFN C_i.

For trapezoid due date $d_i = \left(e_i^1, e_i^2, d_i^1, d_i^2\right)$, if the completion time of job J_i belongs to the interval $\left[e_i^2, d_i^1\right]$, the grade of satisfaction is equal to 1. In other cases, the grade of the satisfaction diminishes with the increase of the tardiness or earliness. Figure 1 describes the fuzzy processing time and fuzzy due date.

In this paper, two objectives are considered, respectively:

$$\mathrm{AI}_{\min} = \min_{i=1,2,\ldots,n} \mathrm{AI}_i, \tag{2}$$

$$C_{\max} = \max_{i=1,2,\ldots,n} C_i \tag{3}$$

where $C_{\max}$ is the maximum fuzzy completion time and $\mathrm{AI}_{\min}$ is the minimum agreement index.

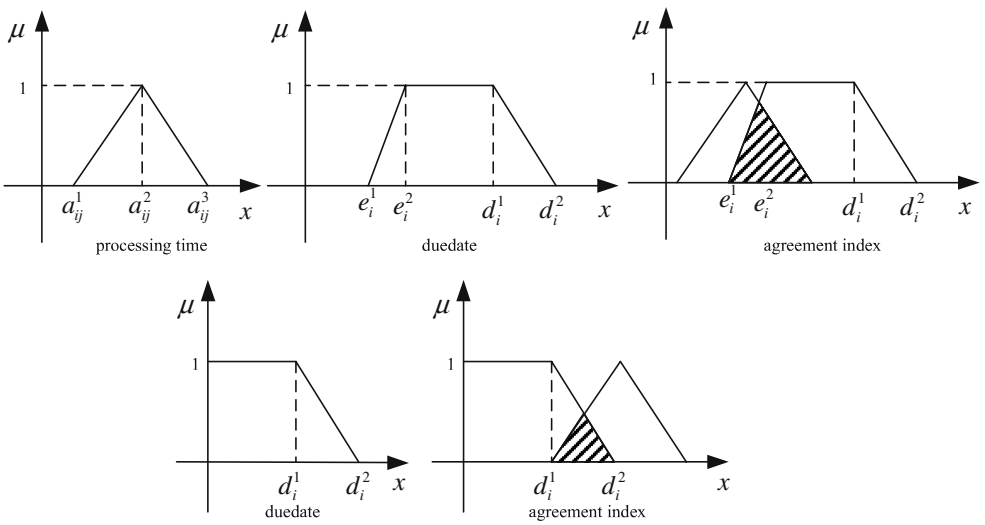

Fig. 1 Fuzzy processing time, fuzzy due date, and agreement index

2.2 Operations on fuzzy processing time

In fuzzy context, some operations of fuzzy number are required to be redefined to build a schedule. These operations involve addition operation and max operation of two fuzzy numbers as well as the ranking methods of fuzzy numbers. Addition operation is used to calculate the fuzzy completion time of operation. Max operation is used to determine the fuzzy beginning time of operation and the ranking method is used to compare the maximum fuzzy completion time.

For two TFNs $\widetilde{s} = (s_1,s_2,s_3)$ and $\widetilde{t} = (t_1,t_2,t_3)$, their addition is shown by the following formula:

$$\widetilde{s} + \widetilde{t} = (s_1 + t_1, s_2 + t_2, s_3 + t_3). \tag{4}$$

The following criteria are adopted to rank $\widetilde{s} = (s_1,s_2,s_3)$ and $\widetilde{t} = (t_1,t_2,t_3)$:

Criterion 1: If $c_1(\widetilde{s}) = \frac{s_1+2s_2+s_3}{4} > (<) c_1(\widetilde{t}) = \frac{t_1+2t_2+t_3}{4}$, then $\widetilde{s} > (<) \widetilde{t}$;

Criterion 2: If $c_1(\widetilde{s}) = c_1(\widetilde{t})$, then $c_2(\widetilde{s}) = s_2$ is compared with $c_2(\widetilde{t}) = t_2$ to rank them;

Criterion 3: If they have the identical c_1 and c_2, the difference of spreads $c_3(\widetilde{s}) = s_3 - s_1$ is chosen as a third criterion.

For $\widetilde{s} = (s_1, s_2, s_3)$ and $\widetilde{t} = (t_1,t_2,t_3)$, the membership function $\mu_{\widetilde{s}\vee\widetilde{t}}(z)$ of $\widetilde{s} \vee \widetilde{t}$ is defined as follows:

$$\mu_{\widetilde{s}\vee\widetilde{t}}(z) = \sup_{z=x\vee y} \min\left(\mu_{\widetilde{s}}(x),\mu_{\widetilde{t}}(y)\right). \tag{5}$$

In this paper, the max of two TFNs $\widetilde{s}$ and $\widetilde{t}$ is approximated with the following criterion:

$$\text{if } \widetilde{s} > \widetilde{t}, \text{ then } \widetilde{s} \vee \widetilde{t} = \widetilde{s}; \text{ else } \widetilde{s} \vee \widetilde{t} = \widetilde{t}. \tag{6}$$

The criterion $\widetilde{s} \vee \widetilde{t} \approx (s_1 \vee t_1, s_2 \vee t_2, s_3 \vee t_3)$ is first used by Sakawa and Mori [2] and named Sakawa criterion for simplicity. The Sakawa criterion has been extensively applied to build a complete scheduling of the fuzzy problem. Figure 2 shows the real max of two fuzzy numbers and two criteria for the approximate max. Compared with the Sakawa criterion, the new criterion has the following features:

(1) For $\widetilde{s}$ and $\widetilde{t}$, their approximate max is either $\widetilde{s}$ or $\widetilde{t}$;
(2) Only three pairs of special points (s_i, t_i) are compared in the Sakawa criterion and three criteria to rank them are used in the new criterion. The approximate max of the new criterion approaches the real max better than that of the Sakawa criterion.

3 Random key genetic algorithm

Compared with the GA with the operation-based representation, the RKGA has the following features: the algorithm itself is real-coded; however, an operation-based integer string can also be obtained in the decoding procedure; the implementation of RKGA is simple, it is easy to apply TPX or DX, and the illegal individual never occurs in the search process.

The RKGA is described as follows:

(1) Randomly generate an initial population P with N individuals and determine the solution with the best fitness as elite individual.
(2) Calculate the fitness of each individual and perform binary tournament selection on P.
(3) Perform crossover and mutation on population P, maintain the elite individual if possible.
(4) If the predetermined number of generations is met, RKGA terminates its search; else, go to step 2.

In the next sections, some steps of RKGA are discussed.

3.1 Random key representation

Random key representation is first proposed by Bean [9], which encodes a solution of JSSP with random numbers. For $n \times m$ JSSP, each gene consists of two parts: an integer in set $\{1, 2, \ldots, m\}$ and a fraction generated randomly from (0,1). The integer part of the random key is interpreted as the machine assignment for job and the decimal part is used to construct the operation sequence on each machine.

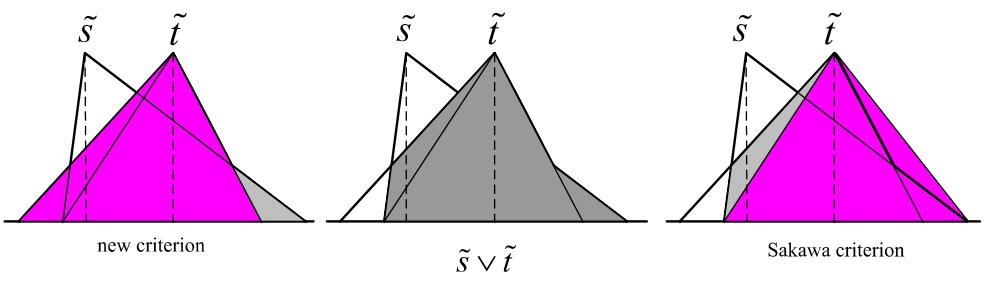

Fig. 2 Comparison between real max and approximate max

The above representation method is seldom considered for the violation of the precedence constraints and the requirement of the special genetic operators. In this study, we present a new random key representation for FJSSP, which encodes a schedule of $n \times m$ problem as a real string $(p_1, p_2, \ldots, p_n, \ldots, p_{mn})$ with $n \times m$ random numbers in the same interval $[a, b]$.

To obtain a feasible schedule, the following decoding procedure is adopted:

(1) Divide the interval $[a, b]$ into a group of subintervals $[a_1, a_2), \ldots, [a_i, a_{i+1}), \ldots, [a_l, a_{l+1}]$ where $a = a_1 < a_2 < \ldots a_l < a_{l+1} = b$. Classify all genes of the chromosome into l groups and make the genes of each group in the same subinterval;
(2) Let $t=1$, $h=0$, start with the first group, choose the gene from small to big, and assign the chosen gene a new value of t and let $h=h+1$, if $h=m$, then $t=t+1$ and $h=0$; repeat the above procedure until each gene is assigned a new value and an integer string is obtained;
(3) Translate the integer string into a list of ordered operations;
(4) The first operation of the list is arranged first, and then the second operation and so on; each operation is assigned the best available beginning time on its required machine. The procedure is repeated until a final schedule is obtained. The procedure is identical to the one proposed by Cheng et al. [10], except that the processing time is fuzzy.

Suppose a chromosome of the 4×2 example in Table 1 is (0.1, 1.3, 2.5, 2.7, 3.9, 1.1, 4.5, 0.8) and p_i is in [0, 5]. The interval is first divided into five subintervals and then genes are separated into five groups; the integer string (1 2 3 3 4 2 4 1) is obtained after step 2 and the chromosome is finally converted into a list of ordered operations of (o_{11}, o_{21}, o_{31}, o_{32}, o_{41}, o_{22}, o_{42}, o_{12}) in step 3. Figure 3a describes the operation sequence on each machine. The chart can be regarded as the modified version of the Gannt chart in fuzzy context and called the fuzzy Gannt chart. The TFN above the line is the fuzzy completion time of operation and the TFN under the line is the fuzzy beginning time. The schedule produced in the above procedure is always feasible.

As shown in Fig. 3a, for operation o_{11}, its beginning time is (0, 0, 0) and the completion time is (1, 2, 3); for operation o_{21}, its beginning time is (1, 2, 3), its completion time is (1, 2, 3)+(2, 4, 5); for operation o_{22}, (3, 6, 8)>(4, 5, 7), (5, 9, 15)<(6, 10, 14), so its beginning time is (3, 6, 8)∨(4, 5, 7) and the completion time is (6, 10, 14). If (5, 9, 15)> (6,10,14), the processing cannot begin at (3, 6, 8) and have to be done after o_{32}.

When the actual processing time is decided for each operation, the actual schedule can be obtained by using operation sequences in Fig. 3a. Figure 3b shows the Gannt chart using the actual processing time in Table 1.

Compared with the random key representation proposed by Bean [9], the gene of the new representation also consists of the integer part and the decimal part; however, the corresponding list of the ordered operation never violates the precedence constraints.

3.2 Fitness, elitism, and reproduction

In this paper, we make the fitness function of an individual equal to its objective function.

The usual elite strategy is used in which the optimal solution produced by RKGA is stored as an elite individual; moreover, the elite individual is always added into the population before reproduction.

Roulette wheel reproduction and breeding pool reproduction cannot be applied for the maximum fuzzy completion time, so tournament selection is used. *Tournament selection* [11] is performed in the following way: first, two individuals are randomly selected from the population and then an individual is chosen if the individual has smaller fitness than the other individual. Finally, the selected individuals go back to the population and can be chosen again.

3.3 Crossover, mutation, and termination condition

TPX and DX are frequently used in the real-coded GA. TPX is shown below: first, randomly select two cut-off points and then exchange genes between the chosen points. DX is done in the following way: produce the random number s following the uniform distribution on [0, 1]; if $s<0.5$, select the gene of one parent; otherwise, select the gene of another parent; repeat the above step until an offspring is obtained.

Mutation is just used to produce small perturbations on chromosomes in order to maintain the diversity of the population. The swap mutation is adopted and described as follows: randomly select two genes from the chosen individual and then exchange them. After TPX, DX, and swap are done, all genes of offspring are still in the same

Table 1 Example of four jobs, two machines FJSSP

Job	Operations					
	1	2	1	2	1	2
	Processing time		Processing sequence		Actual processing time	
1	1, 2, 3	3, 4, 5	M_1	M_2	2	4
2	2, 3, 4	2, 4, 6	M_2	M_1	3	4
3	2, 4, 5	3, 5, 7	M_1	M_2	4	5
4	1, 3, 4	2, 3, 5	M_2	M_1	4	3

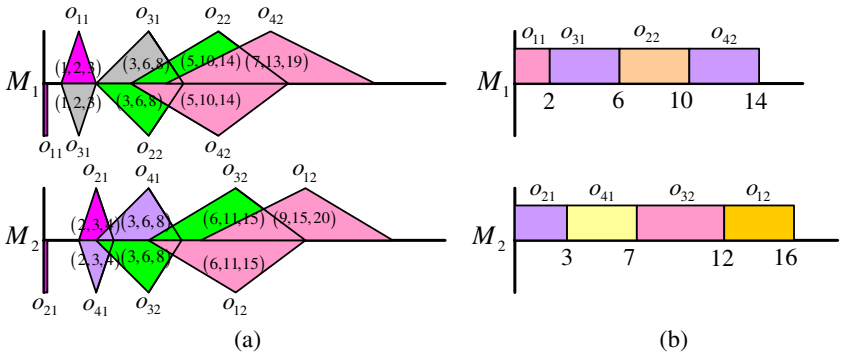

Fig. 3 Fuzzy Gannt chart (**a**) and Gannt chart (**b**)

interval as genes of parents. Offspring also can be converted into a feasible list of ordered operations. No legal individuals are produced by these operators.

When the predetermined number of generations is met, RKGA terminates its search.

4 Computational experiments

In this study, performance analyses on random key representation are first done and then RKGA is compared with the GA proposed by Sakawa and Mori [2]. We call the latter SMGA. Ten benchmark problems are used. Problems 1, 2, 3, 5, 6, and 7 are designed by Sakawa and Mori [2] and problems 4 and 8 by Sakawa and Kubota [3]. Problems 1, 2, 3, and 4 are 6×6 FJSSP and problems 5, 6, 7, and 8 are 10×10 FJSSP. Two 15×10 instances 9 and 10 are designed and shown in the Appendix.

4.1 Performance analyses on random key representation

By comparing RKGA with the GA with the operation-based representation (OPGA), the performance analyses on random key are done. OPGA has the same parameter settings and flow as RKGA. Binary tournament selection and swap mutation of RKGA are also adopted. Generalized order crossover (GOX) and precedence preservative crossover (PPX) are considered in OPGA. Figure 4 describes an illustration of these crossovers. GOX is proposed by Bierwirth [12]. First, randomly select a substring ℓ of the first parent, determine the position of the first element of ℓ on the second parent, and remove the substring ℓ from the second parent. By inserting ℓ into the position of its first element, the offspring is obtained.

PPX is presented by Bierwirth et al. [13]. A string with the same length as the chromosome is filled at random with the elements of set {1, 2}. This string defines the order in which the genes are successively drawn from parents 1 and 2. The offspring is initially empty. A gene which occurs leftmost in two parents is selected. The chosen gene is appended to the offspring and deleted from two parents. This step is repeated until a complete offspring is obtained.

Two variants of RKGA including RKGA1 and RKGA2 are produced. RKGA1 and RKGA2 adopt TPX and DX as crossover operator, respectively. OPGA also has two variants OPGA1 and OPGA2, which use GOX and PPX as crossover, respectively. The parameters of the four algorithms are as follows: crossover probability p_c=0.8, mutation probability p_m=0.1, population scale N=100, the maximum generation of 200 is chosen for 6×6 FJSSP and 300 for 10×10 FJSSP and 500 for other instances.

Four algorithms are implemented by using Microsoft Visual C++ 7.0 and run on Pentium 2.0G PC. All algorithms randomly run 20 times with respect to each instance and the computational results are recorded in Table 2 in which "avg." indicates the average value of the best solutions found in all runs and "opt." denotes the best solutions. In each data grid, there are three kinds of data related to the first objective using doublet and trapezoid and the second objective, respectively. The computational times

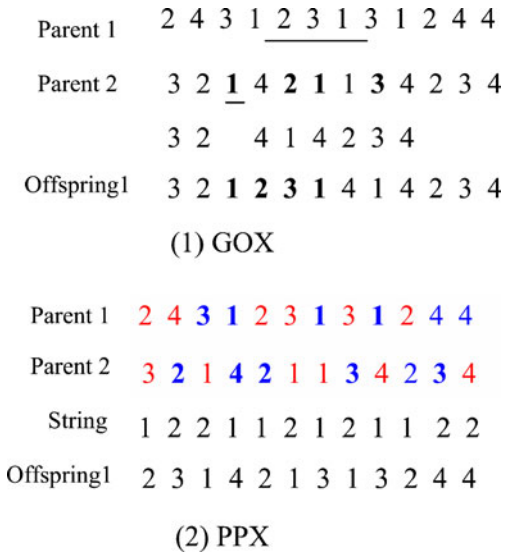

Fig. 4 An illustration of GOX (*1*) and PPX (*2*)

Table 2 Computational results of four algorithms

Problem	Random key representation				Operation-based representation			
	RKGA1		RKGA2		OPGA1		OPGA2	
	avg.	opt.	avg.	opt.	avg.	opt.	avg.	opt.
5	0.532040	0.694340	0.687534	0.802372	0.614565	0.694340	0.289757	0.505051
	0.491025	0.560185	0.551560	0.590909	0.514098	0.578396	0.327204	0.476190
	95.8, 131.5, 163.1	96, 129, 160	95.1, 130.9, 162.2	96, 129, 160	94.9, 130.8, 162	96, 129, 160	96.9, 135, 164.4	95, 133, 161
6	0.787296	0.900000	0.853892	0.90000	0.831986	0.869048	0.625471	0.795152
	0.638198	0.739409	0.673270	0.739734	0.661061	0.725746	0.576391	0.692553
	94, 128.4, 164.2	95, 125, 164	93, 126.2, 163.6	89, 123, 158	94.1, 125.9, 164.9	95, 125, 164	96, 128.8, 165.9	95, 125, 164
7	0.262834	0.603175	0.439251	0.606061	0.377192	0.505480	0.188766	0.256061
	0.358008	0.480114	0.454249	0.585366	0.360857	0.491497	0.221607	0.316356
	85, 117.3, 147.9	85, 116, 143	84.6, 115.9, 148.6	85, 116, 143	85.3, 115.4, 147.5	85, 116, 143	86.1, 118, 147.8	85, 116, 143
8	0.901013	0.968750	0.911335	0.968750	0.906522	0.951220	0.887999	0.967480
	0.863954	0.967480	0.931485	0.96748	0.875967	0.939394	0.894332	0.968750
	28.7, 47.9, 65.6	27, 47, 64	28.4, 48, 64.1	28, 47, 62	28.4, 47.8, 64.3	28, 47, 62	29.1, 48.2, 64.3	26, 47, 64

of each algorithm are shown in Table 3. Problems 5, 6, 7, and 8 have the same number of job and machine, the search process of each algorithm for these problems nearly spend the same duration. So Table 3 only describes the average computational times. The values inside the parentheses are the average computational times about the first objective using the trapezoid due date.

When the doublet due date is considered, two variants of RKGA obtain the maximum value of the first objective of problems 2 and 4, respectively. OPGA1 and OPGA2 cannot approximate the best solutions of any instances. With respect to the average value of the first objective, it can be concluded that RKGA2 obtains better results than two variants of OPGA for four instances. The corresponding results of RKGA1 are also better than those of OPGA2.

When the trapezoid due date is considered, RKGA2 produces the best results of three instances, OPGA2 approximate the best solution of one problem, and both RKGA1 and OPGA1 cannot obtain the maximum objective value for any instances; however, for problems 5, 6, and 7, the maximum value of the first objective generated by OPGA2 is less than that of RKGA1, RKGA2, and OPGA1. With respect to the average results, RKGA2 performs better than the other algorithms for four instances and OPGA2 is inferior to any other algorithm.

With respect to the second objective, RKGA2 and OPGA1 obtain the similar average maximum completion time for four 10×10 instances and these average results are smaller than the corresponding results of RKGA1and OPGA2. On the other hand, RKGA2 finds the best solution of four instances, especially for problem 6; the best solution is only generated by this algorithm. Both RKGA1 and OPGA1 converge to the best solutions of two problems and OPGA2 only approximates the best solution of one instance. Table 3 shows that the computational times of RKGA1 and RKGA2 are close to or smaller than those of OPGA1 and OPGA2. Thus, it can be concluded that two variants of RKGA have better performance than or similar performance with the GAs with the operation-based representation when spending nearly equal times. This conclusion proves that the new representation is effective.

4.2 Results and discussions

RKGA is tested on ten instances and compared with SMGA. We adopt the parameter settings proposed by

Table 3 Computational times of four algorithms

Problem	RKGA1		RKGA2		OPGA1		OPGA2	
	t/*s*		*t*/*s*		*t*/*s*		*t*/*s*	
	AI_{min}	C_{max}	AI_{min}	C_{max}	AI_{min}	C_{max}	AI_{min}	C_{max}
10×10	7.53 (6.84)	7.45	7.40 (6.98)	7.60	7.01 (7.15)	6.99	8.01 (7.68)	7.60

Table 4 Computational results of RKGA and SMGA on the first objective

	RKGA		SMGA	
	avg.	opt.	avg.	opt.
1	0.867418	0.868072	0.826914	0.868072
	0.557809	0.609420	0.531258	0.609420
2	0.972718	0.984321	0.951325	0.984321
	0.747367	0.770032	0.732537	0.770032
3	0.923943	0.933824	0.923943	0.933824
	0.674154	0.700000	0.600433	0.700000
4	0.692308	0.692308	0.692308	0.692308
	0.692308	0.692308	0.692308	0.692308
5	0.687534	0.802372	0.290757	0.495251
	0.551560	0.590909	0.330201	0.476190
6	0.853892	0.90000	0.615423	0.795152
	0.673270	0.739734	0.563191	0.692553
7	0.439251	0.606061	0.176851	0.256061
	0.454249	0.585366	0.231725	0.326532
8	0.911335	0.968750	0.889673	0.941176
	0.931485	0.96748	0.864826	0.96748
9	0.883247	0.953747	0.637628	0.761536
	0.693155	0.780091	0.496435	0.601665
10	0.737256	0.842843	0.512367	0.584548
	0.752662	0.792570	0.505022	0.634521

Sakawa and Mori [2] except the number of objective function evaluation. The parameters and DX described in Section 4.1 are used. Table 4 shows the computational results of RKGA and SMGA on the first objective. Table 5 depicts the comparison between two algorithms on the second objective.

From Tables 4 and 5, it can be concluded that RKGA performs better than SMGA for all instances. For four simple problems, two algorithms have similar performance. For four 10×10 instances and two 15×10 instances, the results generated by SMGA are notably worse than those of RKGA.

Two different representations and two decoding strategies are respectively used in RKGA and SMGA. In RKGA, the schedule of the real chromosome is always feasible and the feasibility is not destroyed by genetic operator; moreover, new solutions can be produced continuously by using DX and swap, meanwhile, some similarities must exist between the schedule of parent and offspring for the feature of the decoding. RKGA can maintain the good balance between exploration and exploitation, so RKGA excels in fuzzy scheduling. On the other hand, SMGA cannot guarantee the feasibility of new solutions and its low performance mainly results from its restricted optimization ability caused by the shortcoming of matrix representation.

5 Conclusions

The GA-based scheduling algorithm for FJSSP frequently uses the integer string to represent the solution of the problem. This paper presents a random key scheduling algorithm, which is based on random key representation, a new decoding procedure, elite strategy, binary tournament selection, TPX or DX, and swap mutation. RKGA is tested and compared with the GA with the operation-based representation and SMGA. Computational results demonstrate the effectiveness of the new representation and good optimization ability of RKGA on fuzzy job shop scheduling.

Local search is often combined with the scheduling algorithms to intensity the optimization ability of the latter. RKGA also can be directly merged with local search such as 2-opt and 3-opt. We will consider the merging of the RKGA and local search. With respect to the application of RKGA, RKGA can be applied to flexible job shop scheduling with fuzzy processing time if the chromosome of machine assignment part is combined with random key

Table 5 Computational results of RKGA and SMGA on the second objective

	RKGA		SMGA	
	avg.	opt.	avg.	opt.
1	56, 80, 103	56, 80, 103	56, 80, 103	56, 80, 103
2	52.2, 71, 87.6	51, 70, 86	52.6, 71.5, 88.5	51, 70, 86
3	50, 65, 84	50, 65, 84	50, 65, 84	50, 65, 84
4	28.9, 36, 43.1	29, 36, 43	28.2, 36.1, 44.4	29, 36, 43
5	95.1, 130.9, 162.2	96, 129, 160	96.8, 134.9, 164.7	95, 133, 161
6	93, 126.2, 163.6	89, 123, 158	96.5, 129.7, 168.3	93, 129, 168
7	84.6, 115.9, 148.6	85, 116, 143	86.1, 118, 147.8	88, 115, 146
8	28.4, 48, 64.1	28, 47, 62	29.1, 48.3, 64.5	28, 47, 66
9	144.7, 211.2, 274.7	142, 207, 271	149.1, 216.1, 279.6	146, 212, 272
10	122.7, 176.2, 227.3	118, 170, 223	125.9, 180.2, 231.7	121, 176, 231

chromosome. RKGA can be used to solve FJSSP with multiple objectives by combining the simplified mechanism of multiobjective optimization. We will discuss these topics in the near future.

Acknowledgements This research is supported by China National Science Foundation (70901064). The authors also want to express their deepest gratitude to the anonymous reviewers for their incisive and seasoned suggestion.

Appendix

Table 6 Trapezoid due date

Problem	Job1	Job2	Job3	Job4	Job5	Job6
1	(94, 100, 112, 121)	(65, 76, 82, 91)	(30, 40, 49, 60)	(78, 85, 97, 102)	(65, 77, 83, 89)	(35, 44, 54, 59)
2	(65, 70, 81, 89)	(50, 58, 69, 80)	(65, 72, 84, 92)	(35, 43, 51, 60)	(72, 80, 90, 96)	(58, 65, 75, 78)
3	(25, 33, 43, 50)	(75, 86, 96, 102)	(74, 83, 93, 103)	(58, 65, 71, 75)	(33, 42, 49, 54)	(42, 52, 62, 70)
4	(18, 25, 30, 40)	(18, 27, 35, 40)	(13, 15, 20, 28)	(19, 26, 32, 40)	(15, 25, 30, 35)	(23, 30, 40, 45)

Table 7 Trapezoid due date

Problem	Job1	Job2	Job3	Job4	Job5	Job6	Job7	Job8	Job9	Job10
5	(115, 169, 184, 195)	(115, 123, 134, 145)	(90, 100, 110, 120)	(90, 102, 105, 115)	(110, 121, 136, 146)	(150, 167, 174, 185)	(110, 120, 130, 140)	(150, 163, 176, 185)	(70, 79, 94, 105)	(150, 160, 163, 170)
6	(120, 130, 151, 156)	(120, 130, 154, 157)	(90, 100, 106, 117)	(100, 115, 123, 138)	(70, 80, 85, 88)	(70, 80, 86, 94)	(100, 110, 120, 135)	(118, 130, 149, 158)	(92, 108, 117, 124)	(110, 121, 142, 148)
7	(100, 108, 124, 128)	(60, 72, 81, 95)	(70, 83, 92, 99)	(70, 83, 91, 103)	(90, 101, 109, 115)	(85, 92, 102, 107)	(96, 108, 118, 128)	(145, 159, 170, 178)	(55, 66, 75, 86)	(78, 86, 94, 107)
8	(24, 34, 45, 60)	(30, 40, 50, 60)	(35, 45, 50, 65)	(30, 40, 50, 65)	(30, 40, 50, 65)	(25, 35, 45, 60)	(25, 36, 45, 60)	(30, 41, 50, 60)	(25, 36, 45, 60)	(30, 40, 50, 60)

Table 8 Processing data of problem 9

	Processing time and processing sequences									
Job1	{2,4,7}3	{9,13,16}4	{5,8,11}6	{8,11,15}10	{10,15,20}5	{7,11,14}7	{9,14,17}1	{7,11,15}9	{10,14,17}2	{6,9,12}8
Job2	{8,11,15}4	{7,10,12}3	{4,6,8}1	{11,15,20}2	{12,17,21}10	{9,11,15}9	{8,12,16}7	{5,6,9}6	{8,10,13}5	{7,11,15}8
Job3	{7,9,12}2	{6,7,9}1	{9,13,17}4	{10,15,20}5	{5,8,12}7	{11,17,19}10	{8,12,16}9	{7,9,13}6	{8,13,17}3	{9,14,18}8
Job4	{10,14,18}5	{11,15,21}3	{9,13,17}9	{8,12,16}6	{7,11,12}4	{11,15,19}8	{10,14,18}2	{8,13,17}7	{9,14,18}10	{3,5,8}1
Job5	{7,10,13}9	{7,11,15}10	{8,12,15}3	{9,13,16}5	{10,14,17}4	{9,12,16}1	{10,15,17}8	{10,13,15}7	{11,14,17}2	{9,12,16}6
Job6	{11,15,21}9	{9,15,18}8	{8,12,16}7	{10,13,16}10	{7,11,14}3	{9,13,17}2	{8,12,15}6	{10,14,17}5	{8,12,18}1	{12,17,20}4
Job7	{6,9,12}5	{7,10,13}6	{8,11,15}10	{9,12,16}4	{7,11,14}1	{8,10,14}9	{10,14,16}7	{7,11,15}8	{7,10,11}3	{5,8,11}2
Job8	{9,12,16}6	{7,10,13}5	{8,11,14}3	{6,9,13}7	{4,7,9}2	{9,13,17}8	{8,10,13}1	{7,8,11}4	{8,9,12}10	{6,8,10}9
Job9	{5,8,11}2	{7,12,14}6	{8,10,13}1	{6,7,8}4	{4,5,8}3	{7,9,11}8	{8,11,13}9	{9,10,14}7	{7,9,12}10	{6,8,12}5
Job10	{4,5,8}3	{7,10,12}6	{8,12,16}7	{5,8,11}10	{3,5,8}2	{4,7,9}4	{6,9,12}9	{7,10,12}1	{5,7,9}8	{10,14,17}5
Job11	{7,8,11}2	{8,9,12}5	{3,5,8}1	{5,7,10}3	{6,9,11}10	{8,10,13}9	{7,10,11}6	{4,5,7}4	{7,11,12}8	{9,13,17}7
Job12	{6,8,11}6	{4,7,10}10	{5,6,9}1	{6,9,12}5	{5,8,10}7	{3,5,9}4	{4,6,9}3	{5,8,12}2	{6,9,12}9	{4,7,10}8
Job13	{3,5,9}6	{7,10,12}10	{5,7,9}9	{6,9,11}8	{4,6,9}5	{8,10,13}7	{9,11,15}4	{7,11,13}1	{5,8,9}2	{7,8,10}3
Job14	{5,8,11}2	{5,7,8}9	{4,5,8}1	{7,11,14}3	{6,9,12}10	{5,9,10}4	{4,5,6}6	{8,11,14}7	{6,9,13}5	{5,8,11}8
Job15	{8,11,15}5	{7,10,12}4	{6,9,10}7	{5,9,10}6	{7,9,12}3	{8,10,13}9	{4,6,9}2	{4,7,10}10	{6,7,11}8	{3,5,8}1

Table 9 Processing data of problem 10

	Processing time and processing sequences									
Job1	{5,7,8}10	{9,10,13}6	{4,5,8}5	{7,8,11}3	{8,9,11}8	{5,8,9}4	{6,7,10}2	{4,6,9}1	{8,11,14}9	{4,7,9}7
Job2	{3,5,8}4	{6,8,10}3	{7,10,12}5	{4,5,8}2	{2,4,6}10	{5,8,11}1	{6,9,12}7	{4,5,7}6	{3,4,6}8	{6,8,11}9
Job3	{6,9,10}9	{3,5,7}8	{2,4,6}3	{4,6,8}1	{5,7,9}10	{7,10,11}6	{5,6,9}7	{4,5,8}4	{1,3,5}2	{7,11,13}5
Job4	{7,11,14}4	{8,12,16}3	{6,9,11}7	{5,8,10}5	{9,13,17}8	{10,14,18}9	{4,7,9}6	{8,10,13}10	{3,5,7}1	{6,8,10}2
Job5	{5,6,8}5	{7,9,10}7	{8,11,16}2	{3,4,5}3	{10,14,18}8	{7,10,13}1	{7,11,14}9	{6,8,9}6	{7,8,10}4	{2,3,6}10
Job6	{8,12,16}7	{6,9,12}1	{7,10,12}5	{5,8,11}4	{4,6,9}8	{3,5,8}9	{6,9,12}2	{4,5,8}6	{8,12,16}3	{9,13,19}10
Job7	{10,14,17}4	{9,13,15}10	{11,15,19}7	{7,11,14}6	{9,14,18}1	{6,10,11}9	{7,11,12}5	{12,18,21}3	{8,10,13}8	{10,12,14}2
Job8	{7,11,14}5	{3,5,8}2	{6,9,12}9	{4,6,9}1	{10,14,18}8	{5,8,11}7	{6,7,9}6	{4,7,10}4	{4,5,8}10	{9,13,17}3
Job9	{13,17,22}10	{11,14,17}2	{9,11,16}5	{7,10,12}4	{6,9,13}9	{7,11,14}3	{5,7,10}7	{8,12,16}1	{11,17,21}8	{1,2,3}6
Job10	{8,12,15}4	{7,10,13}3	{9,12,16}7	{10,13,15}10	{8,10,13}8	{7,9,11}1	{6,8,10}5	{7,8,10}6	{6,7,10}2	{5,6,8}9
Job11	{7,11,14}2	{9,13,17}5	{6,9,12}1	{8,10,14}3	{10,14,17}10	{8,11,15}7	{5,8,11}8	{8,12,16}9	{6,9,12}6	{4,6,9}4
Job12	{4,6,9}2	{5,7,10}4	{6,8,11}1	{5,9,12}3	{4,7,10}10	{9,10,14}8	{7,9,12}9	{10,14,17}5	{7,9,11}7	{5,6,9}6
Job13	{3,5,8}6	{5,7,9}4	{6,8,10}7	{5,8,11}2	{4,6,9}1	{3,4,6}8	{8,10,13}9	{4,6,8}10	{6,8,10}3	{8,10,13}5
Job14	{2,3,5}2	{4,5,8}1	{6,7,10}8	{5,6,9}5	{3,6,9}4	{7,10,11}6	{6,9,12}10	{4,8,9}9	{5,6,8}7	{7,8,9}3
Job15	{5,8,11}5	{7,10,11}9	{6,8,9}3	{5,7,8}4	{4,6,7}2	{8,12,16}7	{5,8,12}8	{9,13,17}10	{4,5,7}6	{3,6,9}1

Table 10 Fuzzy due date of problems 9 and 10

Part	Problem 9		Problem 10	
	Doublet	Trapezoid	Doublet	Trapezoid
Job1	217, 251	171, 208, 217, 251	152, 179	92, 110, 152, 179
Job2	223, 258	164, 194, 223, 258	127, 157	81, 101, 127, 157
Job3	233, 269	180, 217, 233, 269	120, 147	76, 98, 120, 147
Job4	250, 288	194, 234, 250, 288	181, 225	115, 146, 181, 225
Job5	247, 278	183, 219, 247, 278	161, 193	99, 121, 161, 193
Job6	264, 302	204, 246, 264, 302	173, 221	113, 142, 173, 221
Job7	211, 242	159, 191, 211, 242	233, 283	144, 183, 233, 283
Job8	200, 231	143, 168, 200, 231	164, 207	106, 133, 164, 207
Job9	183, 210	128, 150, 183, 210	212, 263	134, 166, 212, 263
Job10	173, 200	132, 160, 173, 200	184, 217	111, 133, 184, 217
Job11	176, 201	125, 148, 176, 201	198, 249	127, 159, 198, 249
Job12	152, 183	119, 144, 152, 183	167, 205	105, 128, 167, 205
Job13	171, 196	124, 148, 171, 196	139, 169	87, 107, 139, 169
Job14	162, 187	124, 151, 162, 187	129, 155	80, 99, 129, 155
Job15	168, 195	125, 150, 168, 195	153, 189	97, 124, 153, 189

References

1. Kuroda M, Wang Z (1996) Fuzzy job shop scheduling. Int J Prod Econ 44(1):45–51
2. Sakawa M, Mori T (1999) An efficient genetic algorithm for job shop scheduling problems with fuzzy processing time and fuzzy due date. Comput Ind Eng 36(2):325–341
3. Sakawa M, Kubota R (2000) Fuzzy programming for multi-objective job shop scheduling with fuzzy processing time and fuzzy due date through genetic algorithm. Eur J Oper Res 120 (2):393–407
4. Song XY, Zhu YL, Yin CW, Li FM (2006) Study on the combination of genetic algorithms and ant colony algorithms for solving fuzzy job shop scheduling problems. In Proceedings of the IMACS Multi-Conferences on Computational Engineering in Systems Applications, Beijing, pp 1904–1909
5. Lei DM (2007) Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. Int J Adv Manuf Technol 37(1–2):157–165
6. Li FM, Zhu YL, Yin CW, Song XY (2005) Fuzzy programming for multi-objective fuzzy job shop scheduling with alternative machines through genetic algorithm. In: Wang L, Chen K, Ong YS (eds) Advance in natural computation. Springer, Berlin, pp 992–1004
7. Giffler B, Thompson GL (1960) Algorithm for solving production scheduling problems. Oper Res 8:487–503
8. Wu C-S, Li D-C, Tsai T-I (2006) Applying the fuzzy ranking method to the shifting bottleneck procedure to solve scheduling problems of uncertainty. Int J Adv Manuf Technol 31(1–2):98–106
9. Bean J (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6:154–160
10. Cheng RW, Gen M, Tsujimura Y (1996) A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation. Comput Ind Eng 30(4):983–997
11. Goldberg DE, Deb K (1991) A comparative analysis of selection schemes used in genetic algorithms. In: Rawlins G (ed) Foundations of genetic algorithms. Morgan Kaufmann, San Francisco
12. Bierwirth C (1995) A generalized permutation approach for job shop scheduling with genetic algorithms. OR Spektrum 17(1):87–92
13. Bierwirth C, Mattfeld D, Kopfer H (1996) On permutation representations for scheduling problems. In: Voigt HM (ed) Proceedings of parallel problem solving from nature IV. Springer, Berlin, pp 310–318