Advanced Materials Research *Online: 2012-06-14*
ISSN: 1662-8985, Vols. 542-543, pp 1251-1259
doi:10.4028/www.scientific.net/AMR.542-543.1251
© 2012 Trans Tech Publications, Switzerland

The Effect of Crossover and Mutation Operators on Genetic Algorithm for Job Shop Scheduling Problem

Long Xu[1, a], Wenbin Hu[1, b]

[1]School of Computer, Wuhan University, China

[a]cerio@live.com, [b]hwb@whu.edu.cn

Keywords: JSSP, Genetic algorithm, Crossover operator, Mutation operator

Abstract. Job Shop Scheduling Problem (JSSP) is a famous NP-hard problem in scheduling field. The concentration of JSSP is to find a feasible scheduling plan to figure out the earliest completion time under machine and processing sequence constraints. At present, genetic algorithm has been widely adopted in varies of operation research problems including JSSP, and good performance have been achieved. However, few work have stress the selection of varies operators when implemented for JSSP. Using benchmark problems, this paper compares the effect of crossover and mutation operators on genetic algorithm for JSSP.

Introduction

The job shop scheduling problem (JSSP) is not only a common scheduling optimization problem in the field of modern manufacturing and computer science but also a typical NP-hard combinatorial optimization problem. Because the solution space gains a non-linear increase when job number and machine number increase, and machine operation sequence of each job exists. It is very hard to use traditional optimization methods to solve a large scale JSSP. Modern evolution algorithms are random searching algorithms which imitate the evolution progress of natural creature and social behavior. They have unparalleled advantages on traditional optimization methods when adopted in solving complex optimization problems. In recent years, meta-heuristic methods like genetic algorithm (GA), simulated annealing (SA), tabu search (TS), ant colony optimization (ACO), particle swarm optimization (PSO) and so on have been implemented to solve the JSSP [1-4].

GA was first proposed by Holland [5] and Davis [6] first applied GA in solving JSSP. Since then extensive research have been made to study GA-based JSSP scheduling method. It can be seen that GA has a good performance in solving JSSP. In GA, the selection of crossover operator, crossover probability and mutation operator have a significant impact on the performance. For different optimization problems, the selection of crossover operator and mutation operator varies, and the value of crossover probability also varies.

In most papers which implement GA-based methods to solve JSSP, the reason of choosing a certain operator is not described in detail [7-8]. This paper compares a series of crossover operator and mutation operator. And a combination of these two operators is found which has a good performance on solving JSSP.

The remainder of this paper is organized as follows. An introduction of JSSP is presented in Section 2. Section 3 makes a brief description of genetic algorithm. In Section 4 and 5, we explain the procedure of crossover and mutation operator used in comparison respectively. Section 6 presents the computation result of experiments. Finally, section 7 draws the conclusion.

The Job Shop Scheduling Problem

The job shop scheduling problem can be described as follows: there are n jobs and m machines; each job has a specific collection of operations to be processed in order by m machines. Given the processing sequence of $\{O_{i1}, O_{i2} \dots O_{im}\}$ $(i = 1,2 \dots n)$ and its processing time $\{t_{i1}, t_{i2} \dots t_{im}\}$, it is required to determine the start time s_{ij} of operation O_{ij} to minimize the total completion time [9].

The constraints and assumption of JSSP are as follows:

(1) Every operation cannot be stopped half way once selected to start, in other words, cannot be preempted.
(2) Each machine could only process one job at any time.
(3) Each job could only be processed on one machine at any time.
(4) The transit time of job between different machines is ignored.

Genetic Algorithm

Genetic algorithm (GA) is a Randomized search method which borrows ideas of the law of biology evolution and has been implemented to solve varies types of problems [10]. It starts from a population which stands for a corresponding solution set. The population is composed of a certain number of individuals, and each individual is encoded by chromosomes. The chromosomes of one individual determine a potential solution of a certain problem. After the initial population is created, according to survival of the fittest principle, the evolution of each generation gives birth to increasingly better approximate solutions. In each generation, offspring are produced by crossover and mutation between parents and then individuals are selected according to their fitness value, finally a new generation is produced. This procedure will lead to a population more adapt to the environment compared with their ancestors. The individual with the best fitness of the last generation can be regarded as the optimal approximate solution of the whole problem.

The flow chart of GA is presented in Fig. 1.

Genetic representation and decoding

The selection of chromosome representation has a great effect on the solving complex of problems. For JSSP, different representations have been proposed, operation-based representation, machine-based representation, random keys representation and so on [11] make the globe solution procedure different from each other. In this paper, operation-based representation is used to solve JSSP. This representation has several advantages, a suitable scheduling can always be got after chromosome replacement by crossover operator, deadlock is avoided and the whole solution space is covered. It is one of the successful GA chromosome representations applied in JSSP.

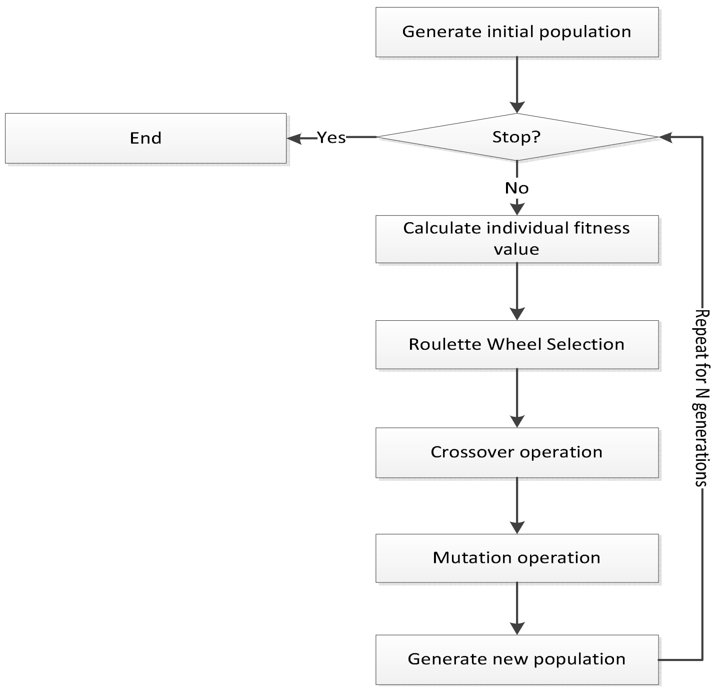

Fig. 1. Flowchart of the standard genetic algorithm

The gene number of chromosome is equal to the number of total operations. For a n job and m machine problem, each job appears m time in the chromosome. By scanning the chromosome from left to right, the jth occurrence of the ith job stands for the jth operation of ith job. For example, in the 3*3 problem illustrated by table 1, the chromosome of one feasible solution can be encode as [2 1 3 2 3 1 2 3 1]. Assume Oij stand for the jth operation of ith job, it can be translated into a more obvious operation representation $[O_{21}, O_{11}, O_{31}, O_{22}, O_{32}, O_{12}, O_{23}, O_{33}, O_{13}]$.

Table 1. A 3 jobs, 3 machines scheduling problem

Job	Processing sequence (machine number, processing time)		
J_1	(1,3)	(2,2)	(3,3)
J_2	(1,3)	(3,5)	(2,4)
J_3	(2,4)	(1,6)	(3,3)

This paper uses a greedy approach to decode the chromosomes to generate a feasible schedule. In principle, a chromosome can be decoded into several practical schedules. Schedules can be categorized into 3 classes: semi-active schedule, active schedule and no-delay schedule [12]. A shift is called local shift if some operations could be started earlier without alter the operation sequence. Global shift is a kind of shift that some operations could be started earlier even altering the operation sequence. In active schedule, no global shift exists, so it has the least makepan among the other schedules. It is described as follows: the chromosome is regarded as a process sequence of jobs, and it is decoded by the position of operations, every operation is arranged as soon as possible considering the processing order constraints. The completion time of the last operation is the completion time of the whole schedule. The Gantt chart of problem illustrated by chart 1 is shown in Fig. 2.

Genetic Operators

Crossover operators exchange the gene between parent individuals and then produce offspring, so that better individual could appear. Mutation operators produce new individuals by changing inner gene of old chromosomes. Crossover and mutation operators both have the effect of creating new individuals and maintain the population diversity. Crossover operator plays a major role and it determines the global search ability which is the key of algorithm convergence. On the other hand, mutation operator determines the local search ability.

The crossover probability of GA has a direct impact on the algorithm convergence. The producing speed of a new individual becomes faster when the value of crossover probability increases. However, the chromosome structure of a high fitness individual could be easily modified if crossover probability is too high. At the same time, if crossover probability is too low, searching process becomes very slow even stagnant.

In the next 2 sections, we introduce several crossover and mutation operators for operation-based representation chromosome structure. They are commonly applied in genetic algorithms and proofed to be effiecient in cases.

Crossover Operator

Crossover operator intersects two individuals by a certain probability to generate two new individuals [13].

One Point Crossover (1PX). If n is the length of a chromosome, in one point crossover, there are n-1 crossover points. The chromosome segment before the crossover point is copied into the offspring. And the offspring segment after the crossover point is filled with the gene from the other parent in order of appearance. The second offspring is produced by changing the role of parents. An illustration of 1PX is shown in Fig. 2.

Order Crossover (OX). Two crossover points are randomly selected and the genes between the two points are copied to the same position of the offspring. Then the copied genes are deleted from the other parent, remained genes are inherited into the offspring starting from the second crossover point, then start from the head of the chromosome. The second offspring is produced by changing the role of parents. An illustration of OX is shown in Fig. 3.

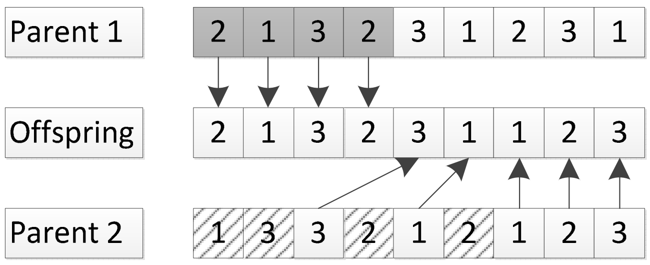

Fig. 2. One point crossover operator (1PX)

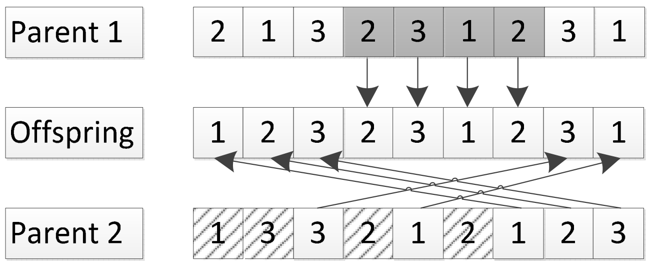

Fig. 3. Order crossover operator (OX)

Linear order crossover operator (LOX). LOX is a modified version of OX. Two random crossover points are selected, and then the genes between the two points are copied into the offspring. Rather than filling the offspring in cycle, LOX start filling from the head to the end. The second offspring is produced by changing the role of parents. An illustration of LOX is shown in Fig. 4.

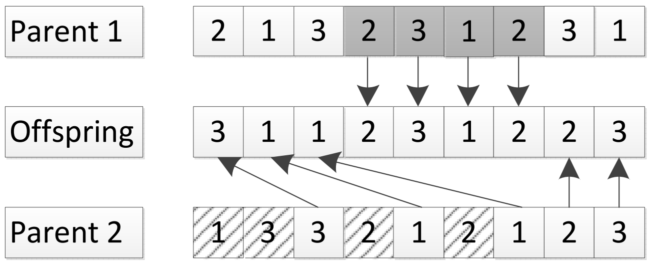

Fig. 4. Linear order crossover operator (LOX)

Position based crossover operator (PBX). A set of positions from one parent is selected randomly in a probability of 0.5. Then the selected genes are copied to offspring in the original place. Selected genes are deleted from the other parent from left to right. The remained genes are used to fill the blank positions of offspring. The second offspring is produced by changing the role of parents. An illustration of PBX is shown in Fig. 5.

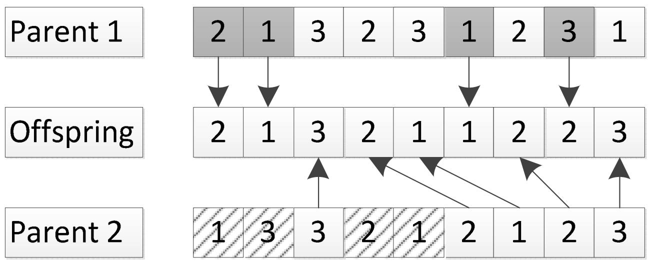

Fig. 5. Position based crossover operator (PBX)

Precedence Preservative Crossover (PPX). A binary sequence with the same length of parents is created. Then scan the sequence from left to right, if the value of a position is 0, then copy the first gene of one parent to the offspring in the same position. After copying, the gene is deleted from both parents. If the value of a position is 1, then copy the first gene of the other parent to the offspring and delete the gene in both parents again. An offspring is created by completing scanning the sequence. The second offspring is produced by changing the role of parents [14]. An illustration of PPX is shown in Fig. 6.

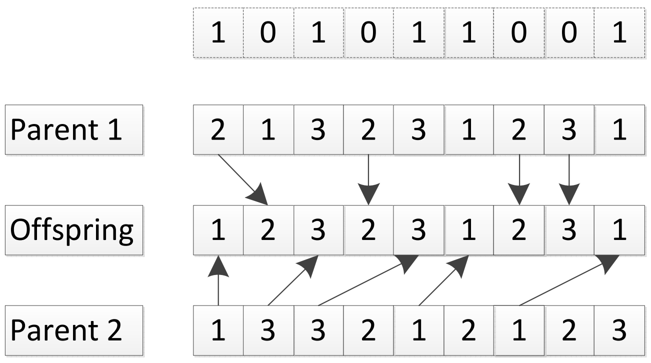

Fig. 6. Precedence Preservative Crossover (PPX)

Two Point Crossover (2PX). Two crossover points are chosen randomly and the chromosome is divided into three segments. The first and the last segment are copied into the offspring, and then genes that have appeared in the two segments are deleted from the other parent in order of appearance. The left genes of the other parent are placed into the offspring. The second offspring is produced by changing the role of parents. An illustration of 2PX is shown in Fig. 7.

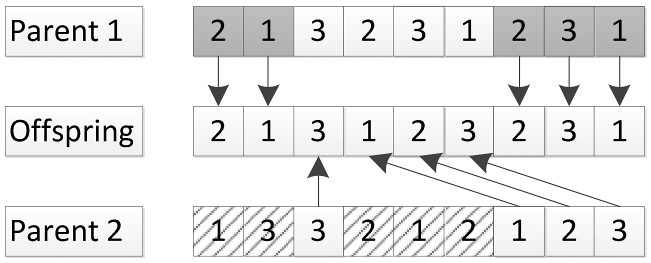

Fig. 7. Two point crossover operator (2PX)

Mutation Operator

Mutation refers to the rearrangement of chromosome according to the mutation probability to form a new individual [15].

Adjacent exchange mutation (AEM). Two adjacent genes are selected and their positions are swapped. A new individual is formed. Fig. 8(a) illustrates the process.

Random exchange mutation (REM). For a chromosome composed of n genes, there are n positions. Two genes are randomly selected and their positions are swapped. Fig. 8(b) illustrates the process.

Shift mutation (SM). A gene is chosen randomly and inserted into another randomly selected position. The segment between the two points is shifted without disrupting the original order. Fig. 8(c) illustrates the process.

Displacement mutation (DM). A chromosome segment is chosen randomly and moved left or right for some positions without disrupting the remained gene order. Fig. 8(d) illustrates the process.

Inversion mutation (IM). Two positions of the chromosome is chosen at random and the genes between them inverse their order. Fig. 8(e) illustrates the process.

Inversion/displacement mutation (IDM). It is a combination of DM and IM, before doing the DM operator the segment inverses its order. Fig. 8(f) illustrates the process.

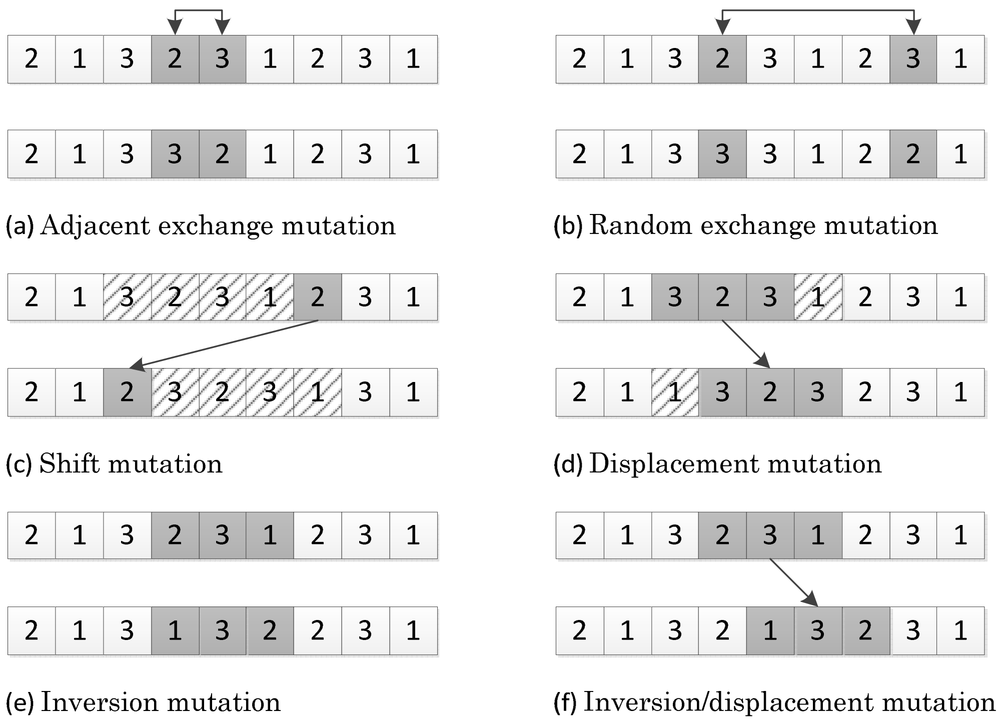
(a) Adjacent exchange mutation
(b) Random exchange mutation
(c) Shift mutation
(d) Displacement mutation
(e) Inversion mutation
(f) Inversion/displacement mutation

Fig. 8. Several mutation operator types

Experimental Results

In this section, samples from OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop 1.txt) are used to compare the performances. There are 82 JSSP test instances in total which are commonly cited by literatures. Solutions are encoded in operation-based representation, the population size is set to 100, and 'Roulette Wheel Selection' is used to select the individuals. After 100 generations, the search progress is terminated. Because mutation operation has little effect on the whole process of GA, mutation probability is given as 0.01 according to convention.

Comparison between crossover probabilities

Instance la01 that has a minimal makespan of 666 is selected to do compare crossover probabilities. Crossover probability ranges from 0.6 to 0.95 with step of 0.5. For each combination of crossover operator and mutation operator, 10 experiments are conducted, and then average values are calculated. The results are shown in table 2. It can be seen that, for different crossover operator, the optimal crossover probability differs. For example, the optimal probability of OX crossover operator is 0.6 while that of 1PX crossover operator is 0.7.

Comparison between combinations of crossover and mutation crossovers

When it comes to compare the efficiencies of different combinations of crossover and mutation operator, the crossover probability is selected according to table 2. The result is presented in table 3 which shows that among the entire mutation operator, combination with 1PX crossover operator has the greatest performance. 1PX combined with REM, SM or IDM has the best performance which is able to figure the optimal solution out.

To further compare the efficiencies of the 3 combinations mentioned above, more experiments are conducted with different crossover probabilities. Table 4 illustrates the computational results. It shows that combination of 1PX and SM occupies 4 places in the top 6 results. It can be concluded that for instance la01, combination of 1PX and SM has the best performance and robustness. The optimal solution of la01 is illustrated in the form of Gantt chart intuitively in Fig. 9.

Table 2. The effect of crossover probability

	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
1PX	677	679	670	676	676	672	674	677
OX	724	731	736	736	748	756	752	760
LOX	714	712	725	721	730	736	747	761
PBX	955	964	982	996	974	1049	1065	1019
PPX	677	680	679	678	680	679	684	684
2PX	673	689	693	704	722	718	753	778

Table 3. Combination comparison results

	AEM	REM	SM	DM	IM	IDM
1PX	668	666	666	670	667	666
OX	740	695	714	693	714	714
LOX	718	717	710	678	687	695
PBX	1008	928	1021	869	794	824
PPX	678	678	673	678	676	677
2PX	681	679	677	676	678	677

Performance of the best combination

For the other instances of JSSP rather than la01, the computational results are illustrated in table 5. The columns of the table include the name and size of the problems, the makespan of the best solution achieved so far (BKS), optimal solutions of the 1PX and SM combination, and solutions obtained by the other combinations. It can be figured out that combination of 1PX and SM performs better than the other combinations almost for every instances. The combination is the best for most of the instances of JSSP.

Table 4. Further comparison results

	probability	Minimal value	Maximal value	Average value
1PX & IDM	0.75	666	744	701
1PX & REM	0.9	666	740	686
1PX & SM	0.7	666	744	717
1PX & SM	0.8	666	740	701
1PX & SM	0.85	666	734	707
1PX & SM	0.9	666	747	699
1PX & SM	0.65	671	724	683
1PX & IDM	0.85	673	743	684
1PX & REM	0.95	673	740	700
1PX & SM	0.75	673	747	696

Conclusions

In this paper, the efficiencies of 36 combinations of 6 crossover operators and 6 mutation operators are compared, and a study of effect of crossover probability on the performance is also done. It is found that for different crossover operators, the corresponding optimal crossover probability varies. Crossover operator, mutation operator and crossover probability all together have a huge impact on the perfermance. For operation-based chromosome representation, combination of 1PX crossover operator and SM mutation operator outperforms the other combinations for the sample JSSP instances of the OR-Library.

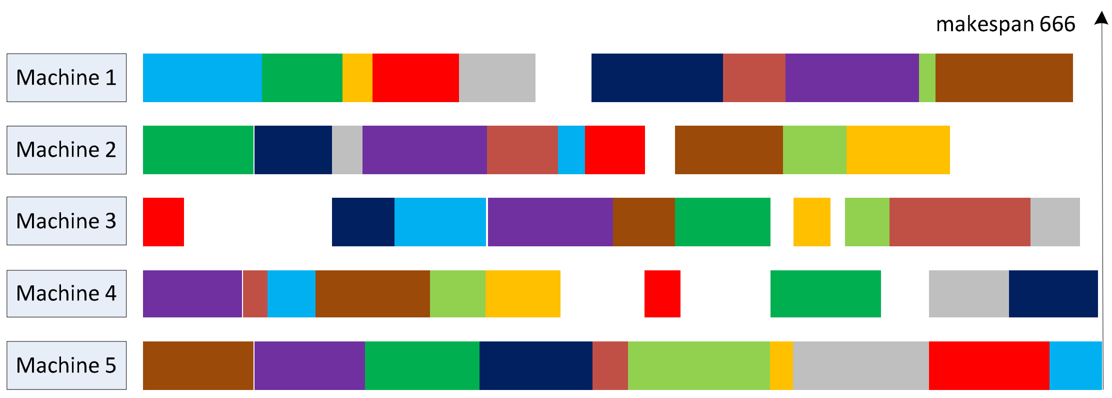

Fig. 9. Gantt chart of one optimal solution of instance la01

Table 5. Computational results of several JSSP benchmarks

	size	BKS	1PX&SM	OX	LOX	PBX	PPX	2PX
ft06	6×6	55	55	56	55	60	55	55
ft10	10×10	930	1052	1283	1114	1414	1115	1134
ft20	20×5	1165	1438	1503	1362	1544	1525	1290
la01	10×5	666	666	693	678	794	673	676
la02	10×5	655	685	748	722	824	714	721
la03	10×5	597	622	681	636	728	630	646
la04	10×5	590	607	652	632	716	607	614
la05	10×5	593	593	593	593	655	593	593
la06	15×5	926	926	946	926	1120	926	926
la07	15×5	890	911	987	971	1121	923	904
la08	15×5	863	863	939	925	1088	863	890
la09	15×5	951	951	999	955	1170	951	968
la10	15×5	958	958	958	966	1131	958	958
la11	20×5	1222	1222	1297	1241	1516	1222	1243
la12	20×5	1039	1039	1117	1077	1248	1039	1064
la13	20×5	1150	1150	1206	1198	1334	1150	1150
la14	20×5	1292	1292	1292	1292	1453	1292	1292
la15	20×5	1207	1284	1442	1348	1594	1329	1323
la16	10×10	945	1003	1137	1074	1355	999	1049
la17	10×10	784	825	979	899	1231	823	884
la18	10×10	848	905	1069	1002	1323	894	958
la19	10×10	842	904	1067	1021	1357	917	1011
la20	10×10	902	959	1161	1065	1326	936	1044
orb01	10×10	1059	1265	1392	1329	1558	1299	1282
orb02	10×10	888	983	1153	1051	1388	1009	1048
orb03	10×10	1005	1192	1428	1216	1420	1239	1205
orb04	10×10	1005	1127	1268	1146	1535	1138	1143
orb05	10×10	887	1054	1158	1048	1229	1069	1011
orb06	10×10	1010	1209	1389	1233	1468	1195	1230
orb07	10×10	397	439	497	462	633	458	474
orb08	10×10	899	1093	1228	1054	1278	1132	1083
orb09	10×10	934	1041	1217	1029	1401	1062	1050
orb10	10×10	944	1093	1253	1108	1404	1068	1105

Acknowledgements

This work is partially supported by National Natural Science Foundation, China(No.70901060); Hubei Province Natural Science Foundation (No. 2011CDB461); State Key Lab of Software Engineering Open Foundation (No.SKLSE2010-08-15) ;Youth Plan Found of Wuhan City(No.201150431101)and the Fundamental Research Funds for the Central Universities. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

References

[1] Byung Joo Park, Hyung Rim Choi, Hyun Soo Kim, A hybrid genetic algorithm for the job shop scheduling problems, Computers & Industrial Engineering, Vol. 45 (2003), p. 597-613

[2] Chao Yong Zhang, PeiGen Li, YunQing Rao, ZaiLin Guan, A very fast TS/SA algorithm for the job shop scheduling problem, Computers & Operations Research, Vol.35 (2008), p. 282-294

[3] Tsung-Lieh Lin et al., An efficient job-shop scheduling algorithm based on particle swarm optimization, Expert Systems with Applications, Vol.37 (2010), p. 2629-2636

[4] J. Heinonen et al., Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem, Applied Mathematics and Computation, Vol.187 (2007), p. 989–998

[5] Holland, J. H. Adaption in natural and artificial systems. Ann Arbor: The University of Michigan Press, (1975)

[6] Davis L. Job shop scheduling with genetic algorithms. Proceedings of the first International Conference on Genetic Algorithms (1985), p. 136-140

[7] Leila Asadzadeh, Kamran Zamanifar, An agent-based parallel approach for the job shop scheduling problem with genetic algorithms, Mathematical and Computer Modelling, Vol.52 (2010), p.1957-1965

[8] Ren Qing-dao-er-ji, Yuping Wang, A new hybrid genetic algorithm for job shop scheduling problem, Computers & Operations Research, Vol.39 (2012), p. 2291-2299

[9] Goncalves, J. F., Mendes, J. J. D. M., & Resende, M. G. C. A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research, Vol.167 (2005), p. 77–95,

[10] Beasley, D., Bull, D.R., Martin, R.R., An overview of genetic algorithms: Part 1, Fundamentals, University Computing, vol.15 (1993), p. 58–69

[11] Runwei Cheng, A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies, Computers & Industrial Engineering, Vol.36 (1999), p. 343-364,

[12] Arno Sprecher, Rainer Kolisch, Andreas Drexl, Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem, European Journal of Operational Research, Vol.80 (1995), p. 94-102,

[13] Talip Kellegöz, Bilal Toklu, John Wilson, Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem, Applied Mathematics and Computation, Vol.199 (2008), p.590-598,

[14] Bierwirth, C., Mattfeld, D., & Kopfer, H. In H. M. Voigt (Ed.), On permutation representations for scheduling problems, Proceedings of Parallel Problem Solving from Nature IV, Berlin, Germany: Springer (1996), p. 310-318,

[15] Andreas C Nearchou, The effect of various operators on the genetic search for large scheduling problems, International Journal of Production Economics, Vol.88 (2004), p. 191-203,