Int J Adv Manuf Technol (2003) 22: 828–835
DOI 10.1007/s00170-003-1689-8

ORIGINAL ARTICLE

L. Wang · L. Zhang · D.-Z. Zheng

A class of order-based genetic algorithm for flow shop scheduling

Received: 19 September 2002 / Accepted: 21 February 2003 / Published online: 29 August 2003

Abstract A class of order-based genetic algorithm is presented for flow shop scheduling that is a typical NP-hard combinatorial optimisation problem with a strong engineering background. The proposed order-based genetic algorithm borrows from the idea of ordinal optimisation to ensure the quality of the solution found with a reduction in computation effort and applies the evolutionary searching mechanism and learning capability of genetic algorithms to effectively perform exploration and exploitation. Under the guidance of ordinal optimisation and with an emphasis on order-based searching and elitist-based evolution in the proposed approach, a solution that is ''good enough'' can be guaranteed with a high confidence level and reduced level of computation. The effectiveness of the proposed algorithm is demonstrated by numerical simulation results based on benchmarks, and its optimisation quality is much better than that of the classic genetic algorithm, the well-known NEH heuristic, as well as being better than a pure blind search. Moreover, the effects of some parameters on optimisation performance are discussed.

Keywords Genetic algorithm · Ordinal optimisation · Order-based genetic Algorithm · Flow shop scheduling

Nomenclature

n	number of jobs
m	number of machines
p_{ij}	processing time of job i on machine j
$C_{\max}$, C^*	makespan, optimal makespan value or lower bound value
P_k, $\|P_k\|$	population at kth iteration and its size
S, $\|S\|$	search space and its size
p_0	initial desired solution quality for G
p_e	final desired solution quality for G
p_{s0}	initial desired probability that at least one of the selected solutions is in G
p_{se}	final desired probability that at least one of the selected solutions is in G
G	the set consist of the p percent "best" feasible solutions
L	sampling number
p_{mut}, p'_{mut}	The first and second mutation probabilities
l	best l solution selected from population
k	iteration number
$a[]$, $b[]$	parent strings
f	fitness value
M	a positive number large enough
$c[]$, $d[]$	children strings
RE	the relative error of the result obtained to C^*
BRE	the relative error of the best result obtained to C^*
ARE	the relative error of the average result obtained to C^*
WRE	the relative error of the worst result obtained to C^*

L. Wang (✉)
Department of Automation, CFINS, Tsinghua University,
100084 Beijing, P.R. China
E-mail: wangling@mail.tsinghua.edu.cn
Tel.: +86-10-62783125 ext. 272
Fax: +86-10-62786911

L. Zhang · D.-Z. Zheng
Department of Automation, CFINS, Tsinghua University,
100084 Beijing, P.R. China

1 Introduction

Because of the complexity and NP-hardness of many real engineering scheduling problems and their key roles in manufacturing systems, it is very important to develop efficient and effective advanced manufacturing and scheduling technologies and approaches. Flow shop scheduling is a class of widely studied scheduling problems with a strong engineering background, which illustrates at least some of the demands required by a wide range of real-world problems and has earned a reputation for being difficult to solve [1, 2]. The permutation flow shop problem with n jobs and m

machines, as studied by many researchers, is commonly defined as follows. Each of n jobs is to be sequentially processed on machine 1, ..., m. The processing time $p_{i,j}$ of job i on machine j is given. At any time, each machine can process at most one job and each job can be processed on at most one machine. The sequence in which the jobs are to be processed is the same for each machine. The objective widely used is to find a permutation of jobs to minimise the maximum completion time, i.e. makespan $C_{\max}$ [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. Due to its significance both in theory and applications, it still represents an important issue for demonstrating the efficiency and effectiveness of newly proposed optimisation methods, although it has been widely studied so far.

Flow shop scheduling is a typical NP-hard combinatorial optimisation problem [1]. Exact techniques are only applicable to small-sized problems in practice. The solution quality of constructive methods such as CDS, NEH, etc. [2, 3] is not quite satisfactory although the process is very quick. Meta-heuristics such as simulated annealing (SA) [5, 6], genetic algorithm (GA) [7, 8], tabu search (TS) [9, 10, 11, 12] and evolutionary programming [13] can obtain fairly satisfactory solutions, but they are often very time consuming and parameter dependent, as well as the fact that the stopping criteria are either impracticable or hard to design. Recently, due to the growing paper number in hybrid systems, hybrid optimisation technology that combines the features of different methods in a complementary fashion has been a hot topic [14, 15, 16]. Ordinal optimisation (OO), proposed by Ho et al. [17, 18] has been successfully applied to many stochastic optimisation problems. It concentrates on finding a "good enough" solution with significantly reduced computation quantities instead of on finding the exact best solution. Borrowing the notion of "survival of the fittest", GA is an evolutionary computation algorithm with a learning capability that has been used in wide application in a variety of fields so far. In this paper, a class of order-based genetic algorithm is proposed for flow shop scheduling which borrows from the idea of OO to ensure the quality of the solution found with reduced computation efforts and which applies the evolutionary searching mechanism and learning capability of GA to effectively perform exploration and exploitation. With the guidance of OO, and by emphasising order-based search and elitist-based evolution, a good enough solution can be guaranteed with a high confidence level and reduced computation quantity.

The organisation of the article is as follows. In Sect. 2, the order-based genetic algorithm is proposed after a brief review of OO and GA. The implementation of the order-based genetic algorithm for flow shop scheduling is provided in Sect. 3, and its effectiveness is tested by simulation based on benchmarks in Sect. 4. The effects of some parameters on the optimisation performance are discussed in Sect. 5 and a conclusion follows in Sect. 6.

2 Order-based genetic algorithm

Recently hybrid optimisation systems have developed into promising tools by combining the features of different methods in a complementary fashion [14, 15, 16]. In this section, after briefly reviewing OO and GA, a class of order-based genetic algorithm is proposed.

2.1 Ordinal optimisation

The basic idea of OO is that "order is easier to determine than value" as well as "goal softening". Instead of trying to find the overall best solution, OO concentrates on finding a good solution in some top percentile of all solutions so that it can significantly reduce computation time and greatly improve convergence [17, 18].

Let S be the search space with $|S|$ solutions and let the set G consist of the p_e percent "best" solutions, i.e., $p_e = |G|/|S|$. Usually $|S|$ is extremely large, so it is very hard and time consuming to find the best global solution in S. If the goal is to obtain at least one solution in G with the desired probability p_{se}, then $p_{se} = 1-(1-p_e)^L$ and $L = \ln(1-p_{se})/\ln(1-p_e)$, where L is the number required for random sampling. In Table 1, the required number L is summarised according to a different p_{se} and p_e. For example, the best of the 2301 randomly selected solutions is within the top 0.1% of the whole solution space with at least 90% probability. Obviously, if one uses some prior information for the problem or guidance to orient the search direction, much better results or probability can be achieved even with the same sampling number. This is the motivation behind replacing the blind search with the GA evolution process under the guidance of OO in this paper.

2.2 Genetic algorithm

Based on the mechanics of artificial selection and genetics, GA combines the concept of survival of the fittest among solutions with a structured, yet randomised information exchange and offspring creation, which repeats evaluation, selection, crossover and mutation after initialisation until the stopping condition is satisfied. GA is naturally parallel and exhibits

Table 1 Sampling number required for good enough solution via OO

Desired probability %	Desired solution quality			
	Top 1%	Top 0.1%	Top 0.01%	Top 0.001%
50	69	692	6932	69314
90	229	2301	23025	230257
99	459	4603	46049	460515
99.9	688	6905	69075	690772
99.99	917	9206	92099	921029
99.999	1146	11508	115124	1151287

implicit parallelism [19, 20], which does not evaluate and improve a single solution but analyses and modifies a set of solutions simultaneously. Even with random initialisation, the selection operator can select some "good" solutions as seeds, the crossover operator can generate new solutions while retaining good features from the parents, and the mutation operator can enhance the diversity and provide a chance to escape from the local optima. So, GA is an iterative learning process with a certain learning ability and thus is considered part of computational intelligence for optimisation. Although a number of weaknesses still exist, such as premature convergence, parameter dependence and difficulty in determining stopping criterion, GA has been extensively studied and applied in a number of fields thus far, especially in production scheduling.

2.3 Order-based genetic algorithm

Borrowing the idea of OO to ensure the quality of the solution found with reduced computation efforts and applying the evolutionary searching mechanism and learning capability of GA to effectively perform exploration and exploitation, a class of order-based genetic algorithm (OGA) is proposed as follows.

Step 1 Randomly sample L solutions to form initial population P_0, where L is determined by OO according to a predefined weak p_{s0} and p_0 . Let $k=0$, and let π^* be the best state among the initial solutions.

Step 2 Select the best top-l solutions from the current population P_k, and perform the first mutation operator with probability p_{mut} for all these solutions; evaluate the newly generated solution if mutation happens and update π^* if necessary, else keep the old solution; then order the resulted l solutions.

Step 3 If stopping criterion is satisfied, stop the algorithm and output the best solution found so far, i.e. π^*; otherwise, continue the following steps.

Step 4 Perform $|P_{k+1}|/2$ times order-based selection and crossover operators using the l solutions obtained by the first mutation operator to generate $|P_{k+1}|$ new temporary solutions, where $|P_{k+1}|$ can be determined by OO according to predefined strong p_{se} and p_e.

Step 5 Perform the second mutation operator with probability p'_{mut} for all the temporary solutions to generate $|P_{k+1}|$ solutions and evaluate all the obtained solutions, and perform elitism policy for π^* simultaneously.

Step 6 Select the best $|P_{k+1}|$ solutions from the $|P_{k+1}|$ solutions obtained by the second mutation and the l solutions obtained by the first mutation to form the next population P_{k+1}, and let $k=k+1$, then go back to step 2.

With the idea of OO, the OGA is able to guarantee a top-p_0 solution with at least a p_{se} confidence level at initialisation stage. Moreover, some useful information on solution space can be gained, which may provide help for the GA's evolutionary self-learning search. In the later procedure, the OGA performs selection, crossover and mutation operators to guide the evolution process, with special emphasis on order-based searching and elitist-based evolution. Thus, with the computational time determined by the OO and the learning search behaviour of the GA, the OGA can find a solution with at least a p_{se} confidence level, which is of the much better quality than the top p_e. In comparison to exact searching and to some meta-heuristics, the OGA is highly efficient; in comparison to blind search and some constructive heuristics, the OGA is of great effectiveness. For the sake of clarity, the procedure is briefly illustrated in Fig. 1.

3 Implementation of the OGA for flow shop scheduling

A job-permutation based encoding scheme has been widely used in many papers for permutation flow shop scheduling, so such an encoding scheme was also adopted for this paper. Next, the implementation of the OGA for permutation flow shop scheduling will be discussed in detail.

3.1 Implementation of step 1

First, since there is no prior information and knowledge available about the problem studied, only a blind search can be performed in the solution space. To achieve a certain search quality and an upper bound on the optimal value, a weak confidence level p_{s0} and an initial requirement p_0 are predefined, e.g., $p_{s0}=90\%$ and $p_0=0.1\%$. Thus, it is concluded via OO that 2301 random samples are required.

Obviously, after the above blind search is completed, some knowledge about the search space can be gained that will be useful for the GA in learning the evolutionary search process. So, let π^* be the best solution among all the random samples, which is regarded as the initial population. In addition, if some problem-dependent information or heuristic is available, it can also be used in the initialisation stage, e.g., the well-known NEH heuristic can be applied to generate one solution for flow shop scheduling.

3.2 Implementation of step 2

To emphasise the order-based search and elitist-based evolution, only the best top-l solutions among the current population P_k are selected to perform later genetic operators. Obviously a good choice of l and $|P_k|$ is often problem specific, so the effects of such parameters on the

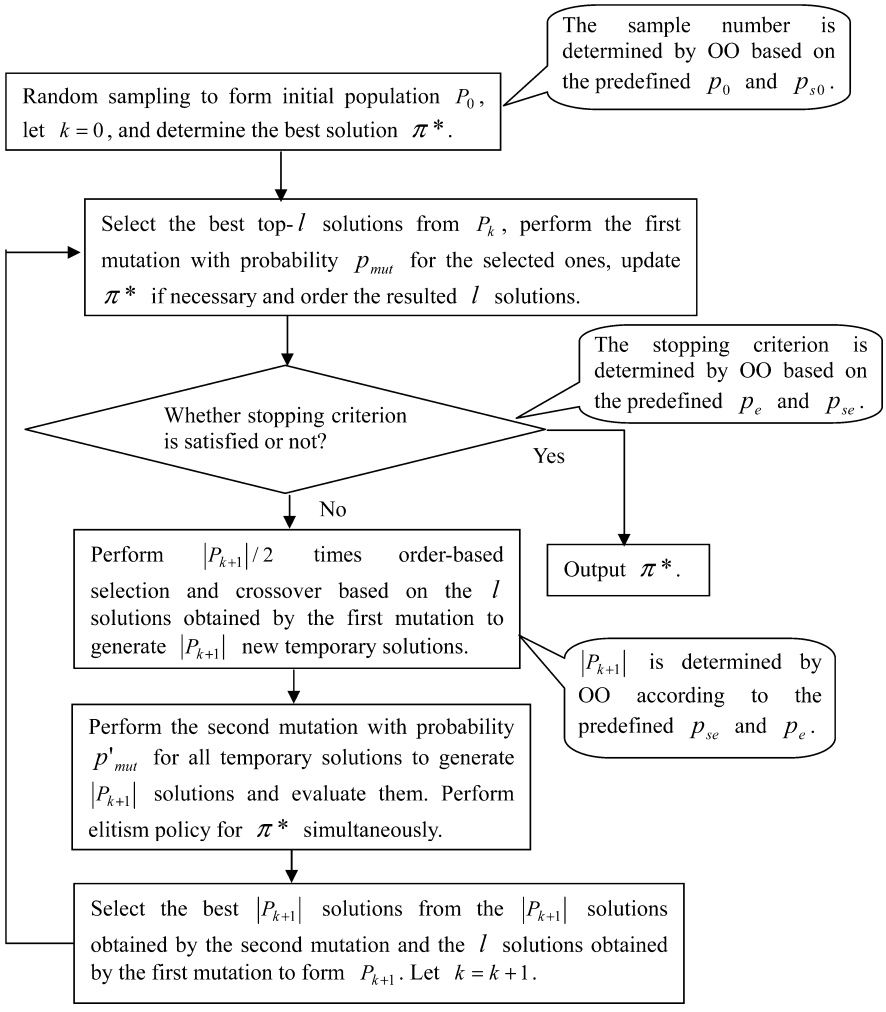

Fig. 1 Brief procedure of the OGA

search quality will be discussed in Sect. 5. To avoid premature convergence, the selected l solutions perform the first SWAP mutation operator [16] with probability p_{mut}, i.e., two distinct elements are randomly selected and swapped. π^* should also be updated if the newly generated solution is better than π^*. Since p_{mut} is set to be very small, the number of evaluation for the newly generated solutions created by the mutation in this step can be ignored.

3.3 Implementation of step 3

To guarantee the final quality of the solution obtained by the OGA, a final desired confidence level p_{se} and a final desired p_e are predefined to determine the total evaluation number by OO. For example, if $p_{se}=99.99\%$ and $p_e=0.01\%$, then 92099 total samples are required. Thus, if population size is set to 100 for each of the remaining generations, almost 900 more generations are needed for the GA, except for the initialisation stage. So the stopping criterion for the OGA is to set maximum generation to 900. Once the OGA stops, the best solution π^* found so far becomes the output.

3.4 Implementation of step 4

The widely used proportional selection operator of GA is based on fitness value, so that one needs to determine a transfer from objective value to fitness value. Here, order-based or rank-based selection is strongly suggested to avoid transfer. That is, the l solutions obtained by the first mutation operator are ordered from the smallest objective value to the largest one, and let $2^{l-i}/(2^l-1)$ be the selected-probability of the ith solution. The crossover operator is only performed for these selected solutions.

The crossover operator used here is a partial mapping crossover (PMX), which may be the most popular one for operating the permutation [20]. In the PMX step, two crossover points are first chosen and the sub-sections of the parents between the two points are exchanged, then the chromosomes are filled up by partial map. For example, let 3 and 7 be two crossover points for parents (2 6 4 7 3 5 8 9 1) and (4 5 2 1 8 7 6 9 3). Then, the children will be (2 3 4| 1 8 7 6| 9 5) and (4 1 2| 7 3 5 8| 9 6). It has been demonstrated that PMX satisfies the fundamental properties enunciated by Holland, namely that it behaves in such a way that the best schemata reproduce themselves better than the others [21].

Moreover, in order to generate $|P_{k+1}|$ new temporary solutions for the next mutation operator, it needs to perform such order-based selection and crossover operators $|P_{k+1}|/2$ times.

3.5 Implementation of step 5

The crossover operator can introduce new individuals by recombining the current population, while the mutation operator serves to maintain diversity in the population and prevent the population from stagnating at local minima. Besides the first mutation operator, the second SWAP mutation operator is applied with mutation probability p'_{mut} for every solution generated by crossover. Mutation also can enhance local search, but to avoid random search p'_{mut} is usually set rather small or with a certain adaptive policy. Then, all $|P_{k+1}|$ solutions resulting from the second mutation are evaluated. Meanwhile, the elitism policy is used in the OGA to avoid losing good solutions, i.e. the best solution found so far, π^*, should be updated during the whole evolution process when better a solution has been found.

3.6 Implementation of step 6

In this step, the best top-$|P_{k+1}|$ solutions are selected from the $|P_{k+1}|$ solutions obtained by the second mutation and the l solutions obtained by the first mutation. These selected solutions are then used to form the next population, i.e., P_{k+1}. Then, the OGA lets $k=k+1$ and goes back to step 2 to repeat the evolution process.

In the next section, some simulations based on benchmarks are carried out and the performance of the OGA is tested.

4 Numerical test and analysis

4.1 Benchmarks selected

Computational simulation is often carried out with benchmarks. In this paper, 29 problems that were contributed to the OR-Library by Mattfeld and Vaessens are selected. The first eight problems were called car1, car2,..., car8, respectively by Carlier [22]. The other 21 problems were called rec01, rec03,..., rec41, respectively by Reeves [7], who used them to compare the performances of SA, GA and neighbourhood search and found these problems to be particularly difficult. All these problems can be downloaded from http://mscmga.ms.ic.ac.uk. Thus far these problems have been used as benchmarks for study with different methods by many researchers [14].

4.2 Simulation results and analysis

Based on the implementation discussed in Sect. 3, 50 independent simulations are carried out for the OGA, classic GA [19] and blind search (BS), respectively with the same computational budget for the OGA, and the statistical results, as well as the results of NEH heuristic, are summarised in Table 2. The parameters of the OGA are set as $p_{s0}=90\%$, $p_0=0.1\%$, $p_{se}=99.99\%$, $p_e=0.01\%$, $p_{\text{mut}}=p'_{\text{mut}}=0.1$, $l=60$, $P_k=100(k\geq 1)$. In the classic GA, population size, maximum evolution generation and initialisation are the same as the OGA and PMX crossover, SWAP mutation with probability 0.1, proportional selection ($f=M-C_{\max}$, where f is fitness value, M is a positive number large enough) are used. The sample number of BS is approximately set to be $2301+900\times100$, i.e., 92301.

From Table 2, it can be obviously concluded that the OGA provides the best optimisation performance for flow shop scheduling among all the methods studied. The results obtained by the OGA are close to being the best results available. In particular, the average relative error of the OGA over the best known is no more than 3.5%. Compared with the NEH method, the OGA can obtain much better results. Compared with the classic GA and BS, not only can the OGA achieve much better results, but its performance is also very stable as its BRE is always very close to the corresponding WRE. Especially when solving larger scale problems, the average performance of the classic GA is only of the same quality level as the NEH method, and the BS is much worse than NEH, but the OGA can do much better than NEH, GA and BS. Thus, it can be concluded that the OGA is superior to NEH, GA and BS with respect to optimisation quality and stability. It also can be concluded that the OGA can achieve the same performance as the classic GA and BS with much reduced computational efforts because more computational effort is required for classic GA and BS to achieve the same solution quality of the OGA. The effectiveness of the OGA can be attributed to the order-based genetic operators, the parallel and self-learning evolution mechanism of GA, and the guidance of OO. In addition, the performance of the OGA is also comparable or even superior to that of taboo search, simulated annealing, genetic algorithm and evolutionary programming and so on as presented in a number of existing papers [5, 7, 9, 10, 13].

5 The effects of some parameters

5.1 Effect of selected size

From the procedure of the OGA presented in Sect. 2, parameter l (number of the selected solutions in every generation) is one of the most important parameters related to the performance of the OGA. The effect of parameter l on the average performance when optimising the problem Rec21 is illustrated in Fig. 2.

Table 2 The statistical results of testing algorithms

Problem	*n*, *m*	*C**	OGA			NEH	GA		BS	
			BRE	*ARE*	*WRE*	*RE*	*BRE*	*ARE*	*BRE*	*ARE*
Car1	11,5	7038	0	0	0	0	0	0.27	0	0
Car2	13,4	7166	0	0	0	2.93	0	4.07	2.62	3.54
Car3	12,5	7312	0	0	0	1.79	1.19	2.95	1.19	3.40
Car4	14,4	8003	0	0	0	0.39	0	2.36	0	1.98
Car5	10,6	7720	0	0	0	4.24	0	1.46	0.48	1.86
Car6	8,9	8505	0	0	0	3.62	0	1.86	0	0
Car7	7,7	6590	0	0	0	6.34	0	1.57	0	0
Car8	8,8	8366	0	0	0	1.09	0	2.59	0	0
Rec01	20,5	1247	0	0.04	0.16	8.42	2.81	6.96	6.50	8.95
Rec03	20,5	1109	0	0.0	0	6.58	1.89	4.45	5.41	7.02
Rec05	20,5	1242	0	0.21	0.32	4.83	1.93	3.82	2.63	3.53
Rec07	20,10	1566	0	0.79	1.15	5.36	1.15	5.31	5.68	7.84
Rec09	20,10	1537	0	0.35	1.17	6.77	3.12	4.73	8.00	11.02
Rec11	20,10	1431	0	0.91	3.07	8.25	3.91	7.39	10.27	12.89
Rec13	20,15	1930	0.26	1.08	1.66	7.62	3.68	5.97	9.64	12.58
Rec15	20,15	1950	0.10	1.23	2.21	4.92	2.21	4.29	6.51	9.77
Rec17	20,15	1902	0	2.08	3.21	7.47	3.15	6.08	10.83	14.04
Rec19	30,10	2093	0.14	1.76	3.01	6.64	4.01	6.07	11.90	13.11
Rec21	30,10	2017	1.44	1.64	3.12	4.56	3.42	6.07	10.36	12.68
Rec23	30,10	2011	0.85	1.90	3.08	10.0	3.83	7.46	12.98	15.24
Rec25	30,15	2513	1.31	2.67	3.74	6.96	4.42	7.20	11.62	15.85
Rec27	30,15	2373	0.97	2.09	3.58	8.51	4.93	6.85	14.08	18.13
Rec29	30,15	2287	1.88	3.28	5.95	5.42	6.21	8.48	15.44	18.41
Rec31	50,10	3045	0.43	1.49	2.59	10.28	6.17	8.02	13.23	16.28
Rec33	50,10	3114	0.61	1.87	4.05	4.75	3.08	5.12	10.48	13.61
Rec35	50,10	3277	0	0	0.33	5.01	1.46	3.30	6.60	8.09
Rec37	75,20	4951	2.46	3.41	4.30	7.80	6.56	8.72	15.69	17.01
Rec39	75,20	5087	1.63	2.28	3.24	7.71	6.39	7.57	14.47	17.24
Rec41	75,20	4960	2.30	3.43	4.69	9.58	7.42	8.92	17.78	20.15

From Fig. 2, it can be concluded that when the value of parameter l is small, the algorithm is easily trapped in local minimums which can lead to premature convergence because there are not enough solutions selected. When l gets larger, the average performance becomes better. But if l is too large, the algorithm allocates a large amount of the search resources to low-quality solutions which induces a decline in the quality of the algorithm. The determination of an optimal parameter l and the adaptive mechanism will the object of the authors' further study.

5.2 Effect of population size

There's a tradeoff in the OGA between population size and generation number under the same computing budget determined by OO, i.e., generation$\times|P_k|$. The effect of population size $|P_k|$ on the average performance under the same computing budget when solving the problem Rec21 is shown in Fig. 3.

From Fig. 3 it can be concluded that when the value of parameter $|P_k|$ is small, the algorithm can not converge at satisfactory solutions in spite of using more

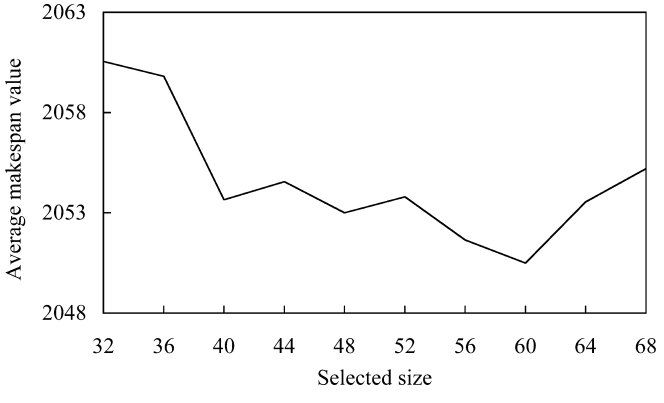

Fig. 2 The effect of selected size l on the average performance when optimising problem Rec21

Fig. 3 The effect of population size $|P_k|$ on the average performance when optimising problem Rec21

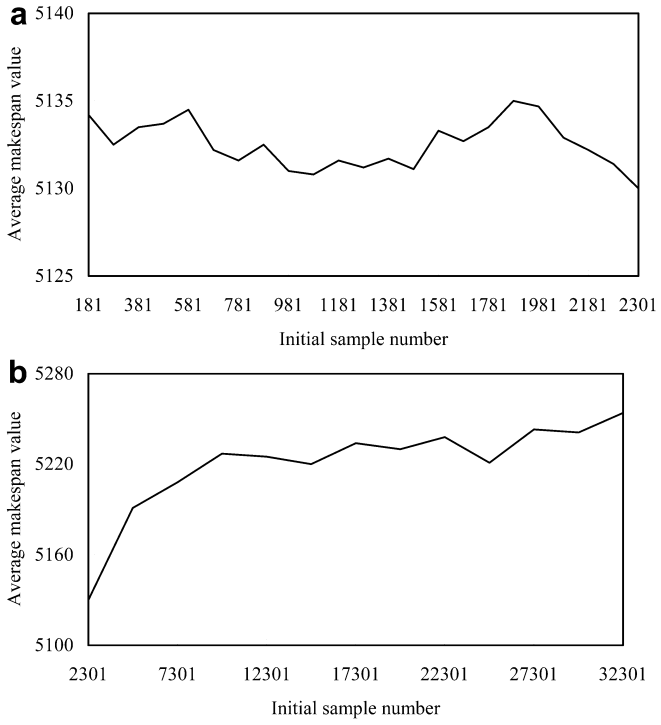

Fig. 4 **a** The effect of initial sample number on the average performance when optimising problem Rec41 and the initial sample number is not large. **b** The effect of initial sample number on the average performance when optimising problem Rec41 and the initial sample number is large

generations of evolution because the advantages of the order-based search mechanism can not be fully applied. When $|P_k|$ grows larger, the result of the algorithm becomes better. But if $|P_k|$ is too large, there are so few generations that the algorithm can not evolve efficiently and the solution quality becomes much worse. Actually, the population size can be variable during the evolution process, which is also a focus of further research work by the authors.

5.3 Effect of initial sample number

Besides parameter l and $|P_k|$, the initial sample number of the algorithm is another important parameter for the OGA. The effect of this parameter on the average performance under the same total computing budget when solving the Rec41 problem is shown in Figs. 4a and 4b. If the initial sample number is not large, it can be seen from Fig. 4a that the results of the algorithm fluctuate very slightly when this parameter is changed. However, if the initial sample number is large enough, it can be seen from Fig. 4b that the search quality becomes much worse when this parameter becomes larger because less computational efforts are allocated to the OGA's evolutionary search. Since this stage is only used to construct the initial population of the OGA, it is suggested that moderate numbers of diversely random solutions should be sampled to form a satisfied initial population and speedy heuristics should be applied in this stage if available.

6 Conclusions

Borrowing the idea of ordinal optimisation to ensure the quality of the best solution found with reduced computation effort, and applying the evolutionary searching mechanism and learning capability of a genetic algorithm to efficiently perform exploration and exploitation, a class of order-based genetic algorithms was proposed for flow shop scheduling problems. Under the guidance of OO and with emphasis on order-based searching and elitist-based evolution, a good enough solution can be guaranteed with a high confidence level and reduced computation quantity, which was demonstrated by benchmark-based computational simulation. Furthermore, the effects of some parameters on the optimisation quality were discussed. Future work will focus on enhancing the guided search ability in conjunction with OO and theoretical study of the convergence behaviour of the OGA, as well as on developing an adaptive mechanism. In addition, due to the generality and easy implementation of the OGA, other applications will be attempted.

Acknowledgement The authors would like to thank Prof. Y. C. Ho (Harvard Univ.) for his helpful discussion on OO. This research is partially supported by National Science Foundation of China (60204008) and the Basic Research Foundation of Tsinghua University, as well as 973 program (2002CB312200).

References

1. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
2. Baker KR (1974) Introduction to sequencing and scheduling. Wiley, New York
3. Nawaz M, Enscore E, Ham I (1983) A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. Omega 11(1):91–95
4. Koulamas C (1998) A new constructive heuristic for the flowshop scheduling problem. Eur J Oper Res 105:66–71
5. Ogbu FA, Smith DK (1990) The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. Comput Oper Res 17(3):243–253
6. Osman IH, Potts CN (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17(6):551–557
7. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22(1):5–13
8. Reeves CR, Yamada T (1998) Genetic algorithms, path relinking and the flowshop sequencing problem. Evol Comput 6:45–60
9. Nowicki E, Smutnicki C (1996) A fast tabu search algorithm for the permutation flow-shop problem. Eur J Oper Res 91:160–175
10. Widmer M, Hertz A (1989) A new heuristic method for the flow shop sequencing problem. Eur J Oper Res 41:186-193
11. Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47:65-74

12. Grabowski J, Pempera J (2001) New block properties for the permutation flowshop problem with application in tabu search. J Oper Res Soc 52:210–220
13. Wang L, Zheng DZ (2003) A modified evolutionary programming for flow shop scheduling. Int J Adv Manuf Technol (in press)
14. Dimopoulos C, Zalzala AMS (2000) Recent development in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. IEEE Trans Evol Comput 4(2):93–113
15. Wang L, Zheng DZ (2001) An effective hybrid optimization strategy for job-shop scheduling problems. Comput Oper Res 28(6):585–596
16. Wang L, Zheng DZ (2003) An effective hybrid heuristic for flow shop scheduling. Int J Adv Manuf Technol 21(1):38–44
17. Ho YC, Sreenivas R, Vakili P (1992) Ordinal optimization of discrete event dynamic systems. Discret Event Dyn Sys 2(2): 61–88
18. Ho YC, Cassandras CC, Chen CH, Dai L (2000) Ordinal optimization and simulation. J Oper Res Soc 51(4):490–500
19. Davis L (1991) Handbook of genetic algorithms. Van Nostrand Reinhold, New York
20. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA
21. Croce FD, Tadei R, Volta G (1995) A genetic algorithm for the job shop problem. Comput Oper Res 22(1):15–24
22. Carlier J (1978) Ordonnancements a contraintes disjonctives. R.A.I.R.O. Recherche operationelle/Oper Res 12:333–351