Int J Adv Manuf Technol (2006) 30: 302–308
DOI 10.1007/s00170-005-0082-1

ORIGINAL ARTICLE

Ling Wang · Liang Zhang

Determining optimal combination of genetic operators for flow shop scheduling

Received: 16 September 2004 / Accepted: 22 March 2005 / Published online: 11 November 2005

Abstract Genetic algorithms (GAs) have gained wide research and applications in production scheduling fields, but the efficiency and effectiveness of a GA significantly depend on its parameters and operators. In contrast to the rich research on determination of optimal and adaptive parameters, little research has been done on determining optimal combination of genetic operators. Different from the traditional way by trial and error, this paper presents a novel and systematical approach based on ordinal optimisation (OO) and optimal computing budget allocation (OCBA) technique to determine optimal combination of genetic operators for flow shop scheduling problems. Simulation results show that the proposed methodology is able to determine optimal combination of genetic operators and simultaneously to provide a good solution with reasonable performance evaluation for scheduling problem.

Keywords Flow shop scheduling · Genetic algorithm · Genetic operators · Optimal combination · Ordinal optimisation

1 Introduction

During the past two decades, meta-heuristics [1] have gained wide attention and applications in production scheduling fields. To our best knowledge, genetic algorithms (GAs) may be the one most widely used for shop scheduling problems. GAs are a class of powerful and robust search techniques based on genetic inheritance and Darwinian metaphor of natural selection, applying population-based stochastic selection, crossover and mutation operators to perform evolution [2, 3]. The efficiency and effectiveness of GAs have been demonstrated by many applications, but it is well known that the efficiency and effectiveness significantly depend on genetic parameters and operators [4]. Unsuitable parameters and operators may lead to significantly inferior results and inconsistency when the evolution generation is limited. However, even setting suitable parameters and operators often suffers from tedious trial and error, and it often needs many simulation replications for evaluation. Currently, it is still an open problem for the GA research to determine optimal parameters and operators [5].

In contrast to the rich research on optimal parameter selection and adaptive parameter design [4–12], little research has been done so far on determining optimal genetic operators. However, Ong and Keane [13] pointed out that the operators perhaps have the greatest influence on performance. Recently, Wang et al. [14] investigated the effects of different crossover and mutation operators on scheduling performance, and they developed an effective hybrid heuristic. Later, they proposed an adaptive GA with multiple operators whose utilisation times depend on their rewarded index [15]. However, crossover and mutation operators were not considered together but separately in their papers. In this paper, the issue to determine an optimal combination of a crossover operator and a mutation operator for flow shop scheduling is considered. After formulating the considered problem as a stochastic optimisation problem, a novel methodology based on ordinal optimisation (OO) [16] and optimal computing budget allocation (OCBA) [17] is presented. And we use simulation results to demonstrate the feasibility and effectiveness.

The remaining paper is organized as follows. In Section 2, the description of the GA and its implementation for permutation flow shop scheduling problem (PFSSP) are briefly presented. In Section 3, after briefly introducing OO and OCBA, the methodology for determining an optimal combination of genetic operators is presented. Simulation results are provided in Section 4, and finally we end the paper with conclusions in Section 5.

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

2 GA and its implementation for PFSSP

2.1 Description of GA

Based on the mechanics of artificial selection and genetics, GAs combine the concept of survival of the fittest among solutions with a structured yet randomized information exchange and offspring creation [2]. A GA is a population based searching technique, which repeats evaluation, selection, crossover and mutation after initialisation until the stopping condition is satisfied. Even if, with random initialisation, a selection operator can select some "good" solutions as seeds, a crossover operator can generate new solutions that hopefully retain good features from parents, and a mutation operator can enhance the diversity and provide a chance to escape from local optima. A general and well-used elitist GA is described as follows.

Step 0
: Given parameters and operators required, set $k=1$ and randomly generate an initial population $P(k) = \{X_1(k), X_2(k), \cdots, X_{P_s}(k)\}$.

Step 1
: Evaluate all the individuals in $P(k)$, and determine the best one X^*.

Step 2
: Output X^* and its performance if k reaches the maximum generation N_g; otherwise, continue the following steps.

Step 3
: Set $l=0$.

Step 4
: Select two parents from $P(k)$, which are denoted by X_1 and X_2.

Step 5
: If a real number r that is randomly generated between 0 and 1 is less than crossover probability P_c, then perform a crossover for X_1 and X_2 to generate two new individual X_1' and X_2'; otherwise, let $X_1' = X_1$ and $X_2' = X_2$.

Step 6
: If a real number r that is randomly generated between 0 and 1 is less than mutation probability P_m, then perform mutation for X_1' to generate individual X_1''; else let $X_1'' = X_1'$. Similarly, X_2'' is generated. Then, put X_1'' and X_2'' into $P(k+1)$, and let $l=l+1$.

Step 7
: If $l < P_s/2$ then go to step 4; otherwise, go to step 8.

Step 8
: Evaluate all the individuals in $P(k+1)$ and the best one is used to update X^* if it is better than X^*, then let $k=k+1$ and go to step 2.

2.2 Description of PFSSP

The permutation flow shop scheduling problem is a class of widely studied scheduling problem with strong engineering background and 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. The PFSSP with n jobs and m machines is commonly defined as follows: Each of n jobs is to be sequentially processed on machine 1 to machine 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 is to find a permutation of jobs to minimise the maximum completion time, i.e., the makespan $C_{\max}$ [18].

The PFSSP is NP–hard, i.e., no method can guarantee global optima in polynomial time. Exact search techniques only can be applied to problems with small scale. The quality of the constructive method is often unsatisfied. Meta-heuristics, such as genetic algorithm, simulated annealing and tabu search [1], are general and can achieve satisfied qualities, but their performances highly depend on the parameters and operators employed. So far, many kinds of GAs have been developed for flow shop scheduling problem [14, 18], but there is little research on determining an optimal combination of genetic operators for such a problem.

2.3 Implementation of GA for PFSSP

Based on the framework in the last section, the GA is simply implemented for PFSSP as follows.

2.3.1 Encoding and fitness value

Job permutation [14, 18] is employed as the encoding scheme. And let the fitness function be $f(X_i) = 1 - C_{\max}(X_i) \Big/ \sum_{i=1}^{P_s} C_{\max}(X_i)$, which is transformed from the makespan.

2.3.2 Proportional selection operator

If a real number r that is randomly generated between 0 and 1 satisfies $\sum_{j=1}^{i-1} f(X_j) \Big/ \sum_{j=1}^{P_s} f(X_j) < r \le \sum_{j=1}^{i} f(X_j) \Big/ \sum_{j=1}^{P_s} f(X_j)$, then individual X_i is selected.

2.3.3 Crossover and mutation operators

In this paper, we mainly discuss the optimal combination of a crossover operator and a mutation operator for PFSSP. Based on the permutation encoding, four crossover operators and three mutation operators are employed for discussion, i.e., partially mapped crossover (PMX), linear order crossover (LOX), non-abel crossover (NAX) and C1 operator, SWAP mutation, inverse (INV) mutation and insert (INS) mutation [14, 18], which are explained with examples as follows.

- PMX: firstly chooses two crossover points and exchanges the sub-section of the parents between the two points, then fills up the chromosomes by partial mapping. Considering the above parents, if the two crossover points are 3 and 7, then the children will be (2 3 4| 1 8 7 6| 9 5) and (4 1 2| 7 3 5 8| 9 6).
- LOX: firstly, two cutting sites of the parents, e.g. (2 6 4 7 3 5 8 9 1) and (4 5 2 1 8 7 6 9 3), are chosen randomly, e.g. 2 and 5. Secondly, the symbols that appear in the cross section of the first parent (positions between the two cutting sites) are removed from the second parent leaving some “holes”, i.e., (H 5| 2 1 8| H 6 9 H) and (H 6| 4 7 3 |5 H 9 H). Then the holes are slid from the extremities towards the centre until they reach the cross section, i.e., (5 2| H H H| 1 8 6 9) and (6 4| H H H|7 3 5 9). Finally the cross sections are substituted with that of the corresponding parent to obtain the children, i.e., (5 2| 4 7 3| 1 8 6 9) and (6 4| 2 1 8| 7 3 5 9).
- NAX: simply, the children of parents $a[i]$ and $b[i]$ are $c[i]=[b[i]]$ and $d[i]=b[a[i]]$, $i=1,2,\ldots,n$. Considering the above two parents, the children will be (7 3 6 2 9 8 5 1 4) and (5 7 1 6 2 8 9 3 4).
- C1: firstly chooses one crossover point randomly, and takes the sub-section of first parent before the crossover point, and then fills up the chromosome by taking in order each “legitimate” element from the second parent. Considering the above two parents, if the crossover point is 3, then the children will be (2 6 4| 5 1 8 7 9 3) and (4 5 2| 6 7 3 8 9 1).
- SWAP: two distinct elements of the parent are randomly selected and swapped.
- INV: inverse the sub-sequence between two different random positions.
- INS: choose two elements randomly and insert the back one before the front one.

2.3.4 Stopping condition

In this paper, computational effort for a GA is limited. In particular, the total evaluation number for a whole evolution process is fixed as N_e, i.e., the maximum generation N_g is fixed as N_e/P_s.

Since limited computational efforts are given, the performance of the GA depends on those specified parameters and operators and so it would not be consistent. Thus, the result of a single run is meaningless. A typical experimental analysis is to carry out many independent runs, and present average makespan value and standard deviation for every implementation of the GA, and use a t-test for comparative experiment [21]. It is very time-consuming, and it is not sure how many runs are required. Besides, the total simulation effort is often limited. So, more effective approaches are in demand. In [12] we discussed the optimal combination of parameters, while in this paper we address the optimal combination of genetic operators based on the idea of OO.

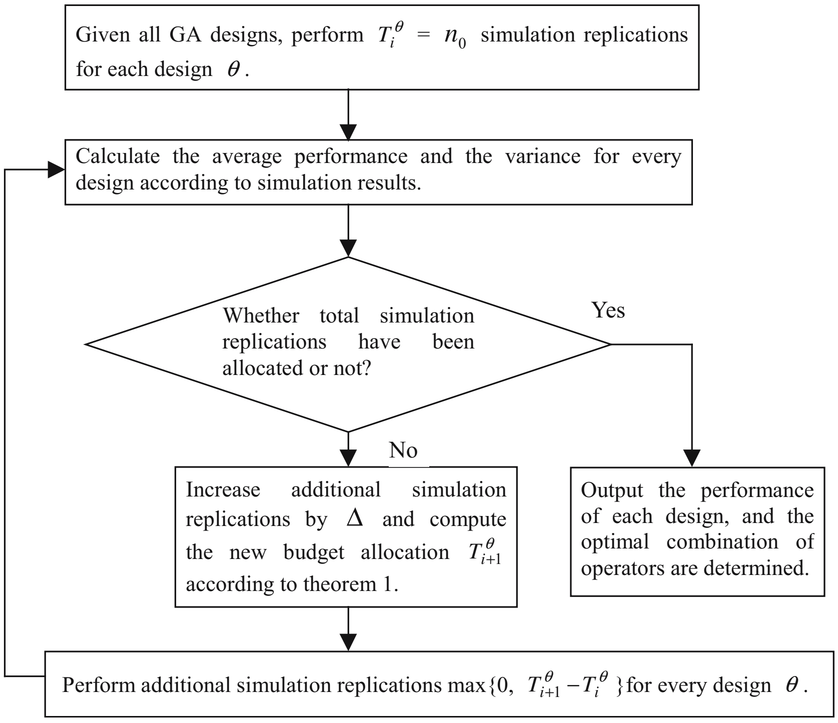
Given all GA designs, perform $T_i^\theta = n_0$ simulation replications for each design θ.
Calculate the average performance and the variance for every design according to simulation results.
Whether total simulation replications have been allocated or not?
Yes
No
Increase additional simulation replications by Δ and compute the new budget allocation T_{i+1}^θ according to theorem 1.
Output the performance of each design, and the optimal combination of operators are determined.
Perform additional simulation replications max{0, $T_{i+1}^\theta - T_i^\theta$ }for every design θ.

Fig. 1 The approach to determine an optimal combination of genetic operators

3 Methodology

3.1 Ordinal optimisation

Consider the following stochastic optimisation problem:

$$\min_{\theta i} J(\theta_i) = E[L(\theta_i, \xi)] \tag{1}$$

where θ is the solution in a finite set, J is the expectation of sample performance L as a function of θ and ξ (randomness).

Due to the stochastic nature of the problem, it requires certain simulation replications to provide a suitable estimation for J_i by the sample mean value $\bar{J}_i = \sum_{j=1}^{T_i} L(\theta_i, \xi_{ij})$, where T_i is simulation replications for θ_i. And its variance is $\sigma_i^2 = \mathrm{Var}[L(\theta_i, \xi)]$ which can be approximated by sample variance when σ_i^2 is unknown beforehand. As T_i increases, $\bar{J}_i$ becomes a better approximation to J_i in the sense that its corresponding confidence interval becomes narrower. However, the ultimate accuracy of this estimation cannot improve faster than $1/\sqrt{T_i}$, so it requires a large number of simulation replications for all the designs to achieve rather correct estimations, which may be very time-consuming [16].

OO involves two important viewpoints [16]. One is that, instead of using accurate performances that presumably take a long time to obtain, one can use the relative order of noisy performance estimates as a basis for comparing and choosing designs. Another is that, instead of picking one single design that is exactly the true optimum in the design space that is very improbable in the presence of large noise, one can pick a subset in which some good enough designs are contained with high probability. That is, "order is easier to determine than value" as well as "goal softening". So, relaxing the requirement from finding the overall best design to concentrates on finding a good design in some good enough designs, OO can significantly reduce computation time and greatly improve the convergence. Dai [20] showed that an alignment probability of ordinal comparison could converge to 1.0 exponentially. Such probability is also called the probability of correct selection, i.e. $P(CS)$, which can be estimated by the very time-consuming Monte Carlo simulation [16]. Recently, Chen et al. [17] provided an approximate probability of correct selection ($APCS$) as the lower bound of $P(CS)$.

$$\begin{aligned} P(CS) &= P\{\text{design } b \text{ is actually the best design}\} \\ &= P\{J_b < J_i, i \neq b | L(\theta_i, \xi_{ij}), j \\ &= 1, 2, \cdots, T_i, i = 1, 2, \cdots K\} \\ &= P\left\{\bigcap_{i=1, i\neq b}^{K} (\tilde{J}_b < \tilde{J}_i)\right\} \geq 1 - \sum_{i=1, i\neq b}^{K} P\{\tilde{J}_b > \tilde{J}_i\} \\ &= 1 - \sum_{i=1, i\neq b}^{K} \int_{-\delta_{b,i}/\sigma_{b,i}}^{\infty} e^{-t^2/2} \Big/ \sqrt{2\pi} \cdot dt = APCS \end{aligned} \tag{2}$$

Table 1 Average makespan value of each combination for Rec01 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	1355.2	**1354.2**	1355.6
LOX	1358.4	1359.3	1358.9
C1	1363.7	1375.8	1368.3
NAX	1361.4	1374.0	1376.6

where $\tilde{J}_i \sim N(\bar{J}_i, \sigma_i^2/T_i)$, $\bar{J}_b \leq \min_i \bar{J}_i$, $\delta_{b,i} = \bar{J}_b - \bar{J}_i$, $\sigma_{b,i}^2 = \sigma_b^2/T_b + \sigma_i^2/T_i$, K is the number of total designs.

3.2 OCBA

Although OO could significantly reduce the computational cost for identifying the good enough solution, there is still potential for further improvement of performance by intelligently determining the number of simulation replications among different designs. Intuitively, to ensure a high $P(CS)$ or $APCS$, a large portion of the computing budget should be allocated to those designs which are potentially good or critical ones in order to reduce estimator variance. On the other hand, limited computational effort should be expanded on non-critical designs that have little effect on identifying the good designs even if they have large variances. Based on such motivation, Chen et al. [17] proposed an OCBA technique to optimally choose the number of simulation replications for all the designs to maximize simulation efficiency with a given computing budget or confidence level.

In particular, OCBA is use to solve the problem described as follows.

$$\max P(CS) \text{ or } APCS, \; s.t. T_1 + T_2 + \cdots + T_K = T \tag{3}$$

$$or \; \min \sum_{i=1}^{K} T_i, s.t. P(CS) \; or \; APCS \geq P^* \tag{4}$$

where T and P^* are total simulation replications and predefined confidence level respectively.

Table 2 Average makespan value of each combination for Rec07 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	1686.7	1688.1	**1685.0**
LOX	1692.3	1690.8	1688.5
C1	1692.2	1707.2	1699.7
NAX	1694.5	1713.6	1715.5

Table 3 Average makespan value of each combination for Rec13 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	2126.2	**2121.5**	2130.8
LOX	2128.4	2135.3	2130.8
C1	2137.3	2151.0	2141.4
NAX	2132.9	2156.0	2169.9

Table 5 Average makespan value of each combination for Rec25 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	**2826.0**	2829.5	2829.7
LOX	2835.3	2835.9	2834.6
C1	2840.8	2847.4	2852.8
NAX	2860.1	2858.0	2866.5

Chen et al. [17] offered an asymptotic solution as follows.

Theorem 1 Given total simulation replications T for K competing designs with the performances depicted by random variables with means $\bar{J}(\theta_1), \bar{J}(\theta_2), \cdots, \bar{J}(\theta_K)$ and finite variances $\sigma_1^2, \sigma_2^2, \cdots, \sigma_K^2$ respectively, as $T \to \infty$, *APCS* can be asymptotically maximized when $T_b = \sigma_b \left(\sum_{i=1, i \neq b}^{K} T_i^2 / \sigma_i^2 \right)^{1/2}$ and $T_i / T_j = \left(\frac{\sigma_i / \delta_{b,i}}{\sigma_j / \delta_{b,j}} \right)^2$, $i, j \in \{1, 2, \cdots, K\}$, and $i \neq j \neq b$, where $\delta_{b,i} = \bar{J}_b - \bar{J}_i$ and $\bar{J}_b \leq \min_i \bar{J}_i$.

3.3 Methodology for determining an optimal combination of operators

In this paper we only discuss the optimal combination of a crossover and a mutation operator, so the genetic parameters are fixed. Thus, the performance of the GA with limited computational effort depends on those operators employed. Because the results of different random runs would not be consistent, it needs to run the GA with a certain operator combination many times to achieve rather accurate performance estimation. Now that the performance of each design only can be estimated by simulation, the determination of an optimal combination of operators can be regarded as a stochastic optimisation problem, where every solution is an implementation of GA (i.e., a combination of genetic operators). Usually, the simulation-based evaluation is very time-consuming, and only limited simulation replications are given. Thus, we develop a novel methodology motivated by OO-based solution for the stochastic optimisation problem to automatically determine an optimal combination of operators with high confidence and simultaneously to provide suitable performance estimation under the total simulation replication constraint is presented and illustrated in Fig. 1.

In particular, in the above algorithm, every design θ_i is a combination of a crossover operator and a mutation operator. To allocate computing budget T_i to θ_i means to carry out T_i independent simulation replications for θ_i, where one simulation means independently running the GA for the PFSSP one time. After certain simulation replications, $\bar{J}$ and σ^2 can be estimated as the mean and sample variance of the performance. Thus, after running the methodology for all designs whose performances are estimated by applying GA with certain times, the optimal combination of operators and its performance estimation can be reasonably determined simultaneously.

Due to the order comparison idea of OO and the intelligent computing budget allocation of OCBA, optimal combination of operators can be determined with a high confidence level. It is regarded that a good algorithm for scheduling problems would have two properties [21, 22]. That is, it would produce a lower makespan average of the best scheduling reached in the replicated runs, and it would produce these best values with high consistency or lower variance. Obviously, the above methodology commits itself to determine the optimal combination of operators with the above two properties with high confidence.

In addition, two parameters are used in the methodology, where n_0 is used to coarsely estimate the performances of all the solutions and Δ is an increment of simulation replications allocated for all the solutions. According to the guidance in [17], n_0 cannot be too small as the estimates of the mean and the variance may be very poor, resulting in premature termination of the comparison. Besides, a large Δ can result in waste of computational time to obtain an unnecessarily high confidence level, but if Δ is small we need to apply theorem 1 many times for budget allocation. A suggested choice for n_0 is between 5 to 20, and 5 to $K/10$ for Δ.

Table 4 Average makespan value of each combination for Rec19 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	2361.4	2370.2	**2361.3**
LOX	2367.2	2377.2	2368.0
C1	2381.0	2385.4	2387.8
NAX	2372.5	2404.2	2397.9

Table 6 Average makespan value of each combination for Rec31 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	3488.3	**3483.3**	3493.3
LOX	3485.6	3494.8	3495.5
C1	3528.0	3505.6	3504.2
NAX	3529.0	3528.5	3530.4

Table 7 Average makespan value of each combination for Rec37 benchmark

Crossover	Mutation		
	SWAP	INS	INV
PMX	5835.1	5838.9	**5832.0**
LOX	5861.4	5846.3	5837.3
C1	5873.2	5859.8	5857.2
NAX	5889.4	5890.2	5893.3

4 Simulation test

In this paper, seven PFSSP benchmarks named as Rec01, Rec07, Rec13, Rec19, Rec25, Rec31 and Rec37 respectively given by Reeves [18] are used for the simulation test. These well-known hard problems with different scales (see Table 8) also can be downloaded from http://mscmga.ms.ic.ac.uk. For every benchmark, there are four choices {PMX, LOX, C1, NAX} for crossover operator and three choices {SWAP, INS, INV} for mutation operator, so there are 12 combinations of operators in all, i.e., 12 kinds of implementations for the GA. For every problem, set the total evaluation number of the GA as 30,000, population size as 40, crossover probability as 0.95 and mutation probability as 0.05. And set n_0=10, Δ=5 and total simulation replication $T = 12 \times 50 = 600$ (total simulation replication for allocation).

By using the proposed methodology, the average makespan value of every combination of a crossover operator and a mutation operator is obtained and shown in Tables 1, 2, 3, 4, 5, 6, 7 for every scheduling problem. And the suggested optimal combinations of genetic operators with the best average performances resulted by our approach for all benchmarks are summarized in Table 8.

From the simulation results, it can be seen that the combination of PMX crossover operator and INS or INV mutation operator is the best among all candidates. While the combination of a mutation operator and C1 or NAX crossover operator behave the worst, especially for large-scale problems. This phenomenon can be attributed to the viewpoint that C1 and NAX disrupt the chromosome too excessively while PMX can well preserve the excellent schemas. On the other hand, seen from the results in Tables 1, 2, 3, 4, 5, 6, 7, 8, there is no significant difference among the three mutation operators that are applied together with a certain crossover operator under the pre-

Table 8 The suggested optimal combination of genetic operators

Problem	n	m	Crossover	Mutation	Average makespan
Rec01	20	5	PMX	INS	1354.2
Rec07	20	10	PMX	INV	1685.0
Rec13	20	15	PMX	INS	2121.5
Rec19	30	10	PMX	INV	2361.3
Rec25	30	15	PMX	SWAP	2826.0
Rec31	50	10	PMX	INS	3483.3
Rec37	75	20	PMX	INV	5832.0

given parameters. This result well accords with the fact that the crossover operator ensures that the GA can make the best of the information obtained from the individuals of the current generation to fully exert its learning ability. That is, to guarantee the promising schema in the current generation to be inherited by offspring to the greatest extent. However, mutation operator only performs a limited local search and introduces certain diversity. So, it especially needs to design crossover operator carefully for GAs.

In a word, based on the simulation results and the effectiveness of OO and OCBA, it is concluded that the presented methodology is reasonable and effective for determining optimal combination of genetic operators, while providing a good solution with reliable performance evaluation for the flow shop scheduling problem.

5 Conclusions

In this paper, a novel methodology based on OO and OCBA was proposed to determine an optimal combination of genetic operators by formulating the issue as a stochastic optimisation problem. Simulation results demonstrated the feasibility and effectiveness of the proposed methodology, which can automatically identify good operators with high confidence under limited simulation replications and simultaneously provide a good solution with reliable performance estimation for flow shop scheduling. The future work is to apply such a methodology to determine optimal robust parameters and operators for stochastic scheduling problems, and discuss the optimal combination of genetic operators for numerical optimisation problems.

Nomenclature

n	Number of jobs
m	Number of machines
$P_{i,j}$	Processing time of job i on machine j
$C_{\max}$	Makespan
f	Fitness value
$J, \bar{J}$	Expectation and mean performances
L	Sample performance
P_s	Population size
P_c, P_m	Crossover probability and mutation probability
$P(k)$	Population at k-th generation
X, X', X''	Individual, and temporary individual
r	A random number between 0 and 1
θ	A combination of a crossover operator and a mutation operator
$\theta*$	The true best combination of operators
θ_b	The best combination determined by simulation
N_e	Total number of evaluation for GA
N_g	Maximum generation number for GA
$P(CS)$	Probability of correct selection
$APCS$	Approximate value of $P(CS)$
$P*$	Pre-defined confidence level for correct selection
T	Total simulation replications

T_i	Simulation replications for i-th operator combination
σ_i^2	Sample variance of makespan for i-th operator combination
$\delta_{b,i}$	$\delta_{b,i} = \bar{C}_{\max}(\theta_b) - \bar{C}_{\max}(\theta_i)$
$\sigma_{b,i}^2$	$\sigma_{b,i}^2 = \sigma_b^2/T_b + \sigma_i^2/T_i$
l	Iteration number for operator
k	Integer for counting generation
i, j	Indication number
n_0	Initial simulation replications in OCBA
Δ	Increment of simulation replications in OCBA

Acknowledgement This paper is partially supported by National Natural Science Foundation under the Grant 60204008 and 60374060, and 973 program under the Grant 2002CB312200.

References

1. Osman IH, Laporte G (1996) Meta-heuristics: a bibliography. Ann Oper Res 63:513–628
2. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison–Wesley, Reading
3. Davis L (1991) Handbook of genetic algorithm. Van Nostrad, New York
4. Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. IEEE Trans Syst Man Cybern 16(1):122–128
5. Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3 (2):124–141
6. De Jong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. Dissertation, University of Michigan
7. Pham QT (1995) Competitive evolution: a natural approach to operator selection. Lecture Notes Artif Intell 956:49–60
8. Srinivas M, Patnaik M (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667
9. Ponnambalam SG, Jawahar N, Kumar BS (2002) Estimation of optimum genetic control parameters for job shop scheduling. Int J Adv Manuf Technol 19(3):224–234
10. Esquivel SC, Ferrero SW, Gallard RH (2002) Parameter settings and representations in pareto-based optimization for job shop scheduling. Cybern Syst 33(6):559–578
11. Pongcharoen P, Hicks C, Braiden PM, Stewardson DJ (2002) Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex products. Int J Prod Econ 78:311–322
12. Wang L, Zhang L, Zheng DZ (2004) The ordinal optimisation of genetic control parameters for flow shop scheduling. Int J Adv Manuf Technol 23:812–819
13. Ong YS, Keane AJ (2004) Meta-lamarckian learning in memetic algorithms. IEEE Trans Evol Comput 8(2):99–110
14. Wang L, Zheng (2003) An effective hybrid heuristic for flow shop scheduling. Int J Adv Manuf Technol 21(1):38–44
15. Zhang L, Wang L, Zheng DZ (in press) An adaptive genetic algorithm with multiple operators for flow shop scheduling. Int J Adv Manuf Technol
16. Ho YC, Sreenivas R, Vakili P (1992) Ordinal optimization of discrete event dynamic systems. Discret Event Dyn Syst 2 (2):61–88
17. Chen CH, Lin J, Yucesan E, Chick SE (2000) Simulation budget allocation for further enhancing the efficiency of ordinal optimization. Discret Event Dyn Syst 10(3):251–270
18. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22(1):5–13
19. Eiben AE, Schoenauer M (2002) Evolutionary computing. Inf Process Lett 82:1–6
20. Dai L (1996) Convergence properties of ordinal comparison in the simulation of discrete event dynamic systems. J Optim Theory Appl 91(2):363–388
21. Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C (2001) Minimizing makespan in a blocking flowshop using genetic algorithms. Int J Prod Econ 70:101–115
22. Wang L, Zheng DZ (2001) An effective hybrid optimization strategy for job-shop scheduling problems. Comput Oper Res 28(6):585–596