Pergamon

Computers ind. Engng Vol. 30, No. 4, pp. 1061–1071, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0360-8352/96 $15.00 + 0.00

S0360-8352(96)00053-8

GENETIC ALGORITHMS FOR FLOWSHOP SCHEDULING PROBLEMS

TADAHIKO MURATA, HISAO ISHIBUCHI and HIDEO TANAKA

Department of Industrial Engineering, Osaka Prefecture University, Gakuen-cho 1-1, Sakai, Osaka 593, Japan

Abstract—In this paper, we apply a genetic algorithm to flowshop scheduling problems and examine two hybridizations of the genetic algorithm with other search algorithms. First we examine various genetic operators to design a genetic algorithm for the flowshop scheduling problem with an objective of minimizing the makespan. By computer simulations, we show that the two-point crossover and the shift change mutation are effective for this problem. Next we compare the genetic algorithm with other search algorithms such as local search, taboo search and simulated annealing. Computer simulations show that the genetic algorithm is a bit inferior to the others. In order to improve the performance of the genetic algorithm, we examine the hybridization of the genetic algorithms. We show two hybrid genetic algorithms: genetic local search and genetic simulated annealing. Their high performance is demonstrated by computer simulations. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

Flowshop scheduling for minimizing the makespan is one of the most well-known problems in the area of scheduling. Various approaches to this problem have been proposed since the publication of Johnson's pioneer work [1]. It is difficult, however, to find the optimal solution of a flowshop scheduling problem involving many jobs and machines (e.g., 50 jobs and 10 machines). Recently, several heuristic approaches based on iterative improvement procedures were applied to the flowshop scheduling because the computation power of available computers was rapidly improved. Osman and Potts [2] proposed simulated annealing heuristics, Widmer and Hertz [3] and Taillard [4] proposed taboo search heuristics.

Recently many authors applied genetic algorithms (see, Holland [5], Goldberg [6] and Davis [7]) to combinatorial optimization problems such as traveling salesman problems (for example, see Jog *et al.* [8], Starkweather *et al.* [9] and Ulder *et al.* [10]) and scheduling problems (for example, see Fox and McMahon [11], Glass *et al.* [12], Ishibuchi *et al.* [13], Manderick [14] and Syswerda [15]). Some empirical studies [7, 9, 10] showed that the ability of genetic algorithms to find near optimal solutions was a bit inferior to other search algorithms.

In this paper, we apply a genetic algorithm to flowshop scheduling problems and examine two hybridizations of the genetic algorithm with other search algorithms. First we examine various genetic operators to design the genetic algorithm for minimizing the makespan. By computer simulations, we show that the two-point crossover and the shift change mutation are effective for this problem. Next we compare the genetic algorithm with other search algorithms such as local search, taboo search and simulated annealing by applying these algorithms to randomly generated test problems. By computer simulations, it is shown that the genetic algorithm is much superior to a random sampling algorithm but a bit inferior to other search algorithms. Then we examine two hybrid genetic algorithms to improve the performance of the genetic algorithm. One is genetic local search algorithm and the other is a genetic simulated annealing algorithm. We also introduce some modifications of search mechanisms in these hybrid genetic algorithms. Computer simulations show the high performance of our modified hybrid algorithms.

2. GENETIC OPERATORS

In this section, first we briefly describe the difference between the coding used for sequencing problems and the standard binary coding. Then we examine various genetic operators such as crossover and mutation for the flowshop scheduling problem before comparing the performance of the genetic algorithm with that of other search algorithms in the next section.

General assumptions of the flowshop scheduling problem can be written as follows (see Dudek *et al.* [16]). Jobs are to be processed on multiple stages sequentially. There is one machine at each stage. Machines are available continuously. A job is processed on one machine at a time without preemption, and a machine processes no more than one job at a time. In this paper, we assume that n jobs are processed in the same order on m machines. This means that our flowshop scheduling is the n-job sequencing problem. We employ the makespan as a criterion of our flowshop scheduling. The makespan is the completion time of the last job.

2.1. *Coding*

In genetic algorithms, each solution of an optimization problem is usually encoded as a bit string. That is, binary representation is usually used for the coding of each solution. A sequence of jobs, however, is handled as a string in this paper for the flowshop scheduling problem. For example, the string "ABCDEF" represents a job sequence where "Job A" is processed first, then "Job B" is processed, and so on. If the string "ABCADE" is generated by genetic operators such as crossover and mutation, this string is not a feasible solution of the flowshop scheduling problem because the allele "A" appears twice in the string and the allele "F" does not appear. Therefore the string used in the flowshop scheduling problem should be the permutation of given jobs. In this paper, we denote the sequence of n jobs by an n-dimensional vector $\mathbf{x} = (x_1, \ldots, x_j, \ldots, x_n)$ where x_j denotes the j-th processing job. That is, the n-dimensional vector $\mathbf{x}$ is handled as a string in genetic algorithms.

2.2. *Selection*

Selection is an operation to select two parent strings for generating a new string (i.e., child). Let N_{pop} be the number of solutions in each population in genetic algorithms, i.e., N_{pop} is the population size. We denote N_{pop} solutions in the t-th generation by $\Psi_t = \{\mathbf{x}_t^1, \mathbf{x}_t^2, \ldots, \mathbf{x}_t^{N_{\text{pop}}}\}$. Each solution $\mathbf{x}_t^i$ is selected as a parent string according to the selection probability $P_S(\mathbf{x}_t^i)$. We used the following selection probability in computer simulations:

$$P_S(\mathbf{x}_t^i) = \frac{[f_M(\Psi_t) - f(\mathbf{x}_t^i)]^2}{\sum_{\mathbf{x}_t^i \in \Psi_t} [f_M(\Psi_t) - f(\mathbf{x}_t^i)]^2}, \tag{1}$$

where $f(\mathbf{x}_t^i)$ is the objective function to be minimized in the flowshop scheduling problem (i.e., makespan) and $f_M(\Psi_t)$ is the worst value of $f(\cdot)$ in the t-th generation:

$$f_M(\Psi_t) = \max\{f(\mathbf{x}_t^i) | \mathbf{x}_t^i \in \Psi_t\}. \tag{2}$$

We also examined the following selection probability:

$$P_S(\mathbf{x}_t^i) = \frac{f_M(\Psi_t) - f(\mathbf{x}_t^i)}{\sum_{\mathbf{x}_t^i \in \Psi_t} [f_M(\Psi_t) - f(\mathbf{x}_t^i)]}. \tag{3}$$

By preliminary computer simulations with the selection probabilities in (1) and (3), we found that the performance of (1) was better than that of (3). This is because the selection probability in (1) realizes the stronger selection pressure than that in (3). Therefore we show only simulation results with the selection probability in (1) in this paper.

2.3. *Crossover*

Crossover is an operation to generate a new string (i.e., child) from two parent strings. Since our flowshop scheduling problem is a sequencing problem of n jobs, various crossover operators proposed for traveling salesman problems and scheduling problems are applicable.

In this paper, we examine 10 crossover operators. The following four crossover operators turned out to be effective for the flowshop scheduling problem by computer simulations. Simulation results will be shown later.

(1) One-point crossover. This crossover is illustrated in Fig. 1(a). One point is randomly selected for dividing one parent. The set of jobs on one side (each side is chosen with the same probability) is inherited from one parent to the child, and the other jobs are placed in the order of their appearance in the other parent. In Fig. 1(a), Jobs A, B and C are inherited from Parent 1 to the child, and the other jobs (i.e. Jobs C, D, E, F and G) are placed in the order of their appearance in Parent 2.

(2) Two-point crossover (Version I). This crossover is illustrated in Fig. 1(b). Two points are randomly selected for dividing one parent. The jobs outside the selected two points are always inherited from one parent to the child, and the other jobs are placed in the same manner as the one-point crossover.

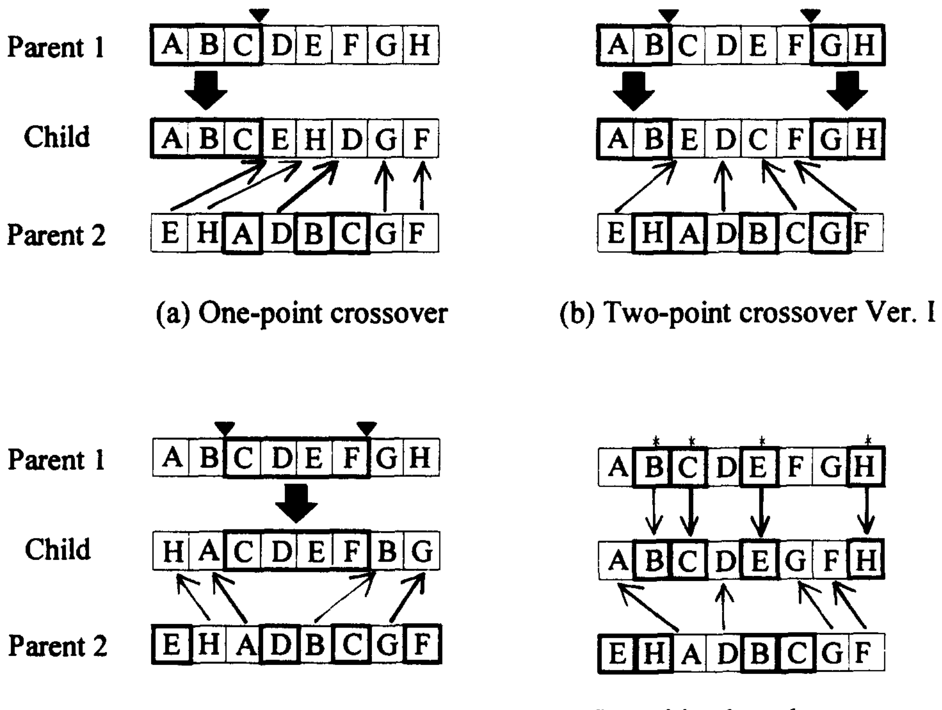
(3) Two-point crossover (Version II). This crossover is illustrated in Fig. 1(c). The set of jobs between two randomly selected points is always inherited from one parent to the child, and the other jobs are placed in the same manner as the one-point crossover.

(4) Two-point crossover (Version III). This crossover is the mixture of the above versions of the two-point crossover (Versions I and II). These two versions are applied to each pair of selected parents with the same probability (i.e., 0.5 for each version).

The following versions of the position based crossover (see, Syswerda [15]) were also examined in computer simulations:

(5) Position based crossover (Version I). This crossover is illustrated in Fig. 1(d). The jobs at randomly selected positions marked by "*" are inherited from one parent to the child, and the other jobs are placed in the order of their appearance in the other parent. In Version I, the number of those positions is first determined as a random integer in $[1, n]$, then positions are randomly selected.

(6) Position based crossover (Version II). This crossover is basically the same as the above position based crossover (Version I) except for the choice of positions. In this crossover, each position is independently marked with the probability of 0.5, while the number of marked positions is first determined in the above crossover (Version I).



(a) One-point crossover (b) Two-point crossover Ver. I

(c) Two-point crossover Ver. II (d) Position based crossover

Fig. 1. Crossover operators.

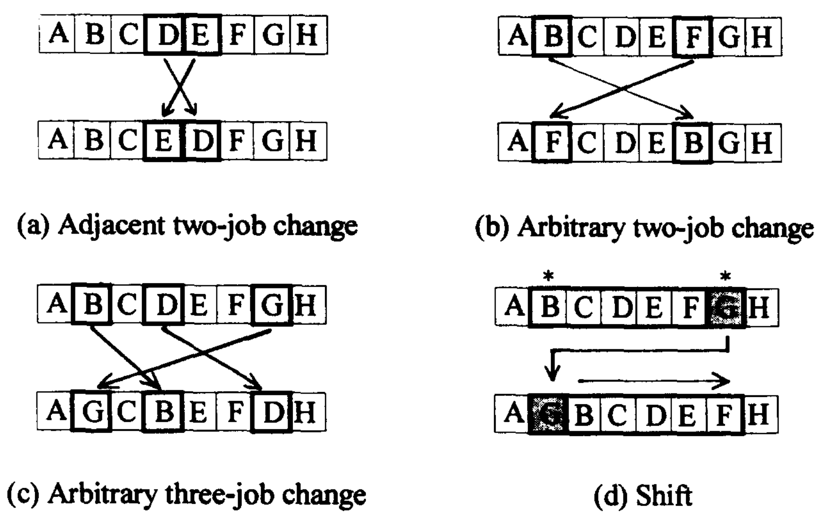

(a) Adjacent two-job change
(b) Arbitrary two-job change
(c) Arbitrary three-job change
(d) Shift

Fig. 2. Mutation operators.

The following four crossover operators, which have been mainly proposed for traveling salesman problems, are also examined in this paper for the flowshop scheduling problem. We omit a detailed explanation of these crossover schemes because they turned out to be inappropriate for the flowshop scheduling by computer simulations (simulation results will be shown later).

(7) Edge recombination crossover in Whitley [*17*].

(8) Enhanced Edge recombination crossover in Starkweather [*9*].

(9) Partially matched crossover in Goldberg [*6*].

(10) Cycle crossover in Oliver [*18*].

2.4. Mutation

Mutation is an operation to change the order of n jobs in each string generated by a crossover operator. Such a mutation operation can be viewed as a transition from a current solution to its neighborhood solution in local search algorithms. We examine the following four mutation operators for the flowshop scheduling problem.

(1) Adjacent two-job change. Adjacent two jobs are changed as shown in Fig. 2(a). The adjacent two jobs to be changed are randomly selected.

(2) Arbitrary two-job change. Arbitrary selected two jobs are changed as shown in Fig. 2(b). The two jobs to be changed are arbitrary and randomly selected. This mutation includes the adjacent two-job change as a special case.

(3) Arbitrary three-job change. Arbitrary selected three jobs are arbitrary changed as shown in Fig. 2(c). The three jobs to be changed are arbitrary and randomly selected, and the order of the selected jobs after the mutation is randomly specified. This mutation includes the above two mutation operaters as a special case.

(4) Shift change. In this mutation, a job at one position is removed and put at another position as shown in Fig. 2(d). The two positions are randomly selected. This mutation includes the adjacent two-job change as a special case and has an intersection with the arbitrary three-job change.

2.5. Genetic algorithms

By combining the above genetic operators, we can implement various genetic algorithms. The outline of the genetic algorithms can be written as follows.

Step 1 (Initialization). Randomly generate an initial population Ψ_1 of N_{pop} strings (i.e., N_{pop} solutions).

Step 2 (Selection). Select N_{pop} pairs of strings from a current population according to the selection probability.

Step 3 (Crossover). Apply one of the above crossover operators to each of the selected pairs in Step 2 to generate N_{pop} solutions with the crossover probability P_c. If the crossover operator is not applied to the selected pair, one of the selected solutions remains as a new string.

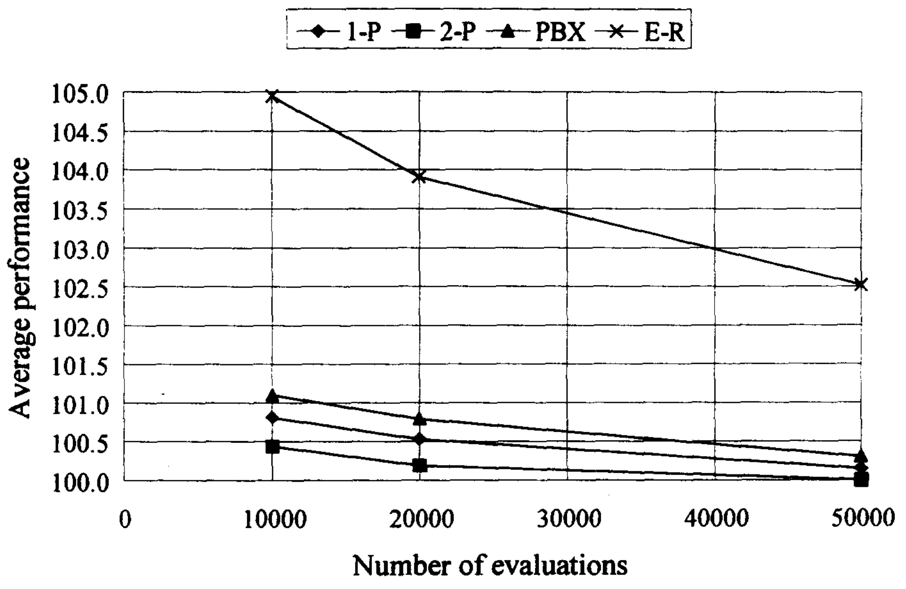


Fig. 3. Simulation results by the four crossover operators (1-P: One-point crossover, 2-P: two-point crossover Ver. I, PBX: Position based crossover Ver. I, E-R: Edge recombination).

Step 4 (Mutation). Apply one of the above mutation operators to each of the generated N_{pop} strings with the mutation probability P_m (we assign the mutation probability not to each bit but to each string).

Step 5 (Elitist strategy). Randomly remove one string from the current population and add the best string in the previous population to the current one.

Step 6 (Termination test). If a prespecified stopping condition is satisfied, stop this algorithm. Otherwise, return to Step 2.

2.6. Comparison between various specifications of genetic algorithms

As test problems, we randomly generated 100 flowshop scheduling problems with 20 jobs and 10 machines. The processing time of each job at each machine is randomly specified as an integer in the closed interval [1, 99].

In order to choose appropriate parameter values in our genetic algorithms for the flowshop scheduling with an objective of minimizing the makespan, we employed the above mentioned 10 crossover operators and four mutation operators with the following values of the population size N_{pop}, the crossover probability P_c and the mutation probability P_m:

$$N_{pop} = 5, 10, 20, 30, 40, 50,$$

$$P_c = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,$$

$$P_m = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0.$$

By computer simulations on the test problems using genetic algorithms with various parameter specifications, we found that the following specifications worked best.

Population size: $N_{pop} = 10$,
Crossover probability: $P_c = 1.0$,
Crossover: Two-point crossover (Version I),
Mutation probability: $P_m = 1.0$,
Mutation: Shift change mutation.

As we can see from Fig. 2, the mutation operators are applied to an entire string while a mutation operator is usually applied to each position of a string in the case of a binary coding. Then each string was mutated only once by the shift change mutation in each generation in computer simulations because we use the mutation probability: $P_m = 1.0$. As a stopping condition in computer simulations, we used the total number of evaluations of the objective function in order

Table 1. Comparison of various crossover operators

	Number of evaluations		
Crossover operator	10,000	20,000	50,000
One-point crossover	100.81	100.53	100.15
Two-point: Ver. I	100.44*	100.19*	100.00*
Two-point: Ver. II	100.85	100.55	100.19
Two-point: Ver. III	100.66†	100.35†	100.03†
Position based: Ver. I	101.10	100.79	100.30
Position based: Ver. II	100.81	100.50	100.24
Edge recombination	104.95	103.91	102.52
Enhanced E.R.	105.14	104.12	102.99
Partially matched	100.69	100.45	100.17
Cycle crossover	100.84	100.50	100.15

*·† show the best result and the second best result in each column.

Table 2. Comparison of four crossover operators

	50,000 evaluations	
Crossover operator	Average	Standard deviation
One-point crossover	100.05	0.6835
Two-point: Ver. I	100.00	0.7047
Position based: Ver. I	100.19	0.7214
Edge recombination	102.33	1.1952

to compare various parameter specifications under the same computation load. Genetic algorithms with various parameter specifications were compared with each other for the three stopping conditions: The total number of evaluations of the objective function was specified as 10,000, 20,000 or 50,000 in our computer simulations. Because the total number of evaluations was used as a stopping condition, the total number of generations was specified as to be inversely proportional to the population size N_{pop}. For example, when the total number of evaluations was 10,000, the total number of generations was specified as $10{,}000/N_{pop}$.

First we show simulation results with various crossover operators in Table 1 and Fig. 3. Table 1 is the result of a single simulation for each of 100 test problems by each crossover operator. Average makespans were normalized in Table 1 using the result by the genetic algorithm with the above mentioned parameter specifications and the stopping condition of 50,000 evaluations. We also plotted the results of four representative crossover operators (i.e., One-point crossover, Two-point crossover Version I, Position based crossover Version I and Edge recombination) in Fig. 3. From Table 1 and Fig. 3, we can observe the high performance of the two-point crossovers (Versions I and III) and the poor performance of the edge recombination and the enhanced edge recombination. We also examined the four representative crossover operators in detail by the same 100 problems. To get the average performance of each crossover operator and its standard deviation, we iterated the computer simulation 10 times on the 100 test problems for each crossover operator. The average performance of the ten trials and its standard deviation are shown in Table 2 for each crossover operator. In the same manner as in Table 1, each average value in Table 2 was normalized. Table 2 shows that the standard deviation of the average performance in Table 1 is not so large. From Table 2, we also observe the high performance of the two-point crossover Version I.

Next we show simulation results with various mutation operators in Tables 3 and 4. Figure 4 shows the same results as in Table 3. From these results, we can observe the high performance of the shift change mutation.

3. COMPARISON WITH OTHER SEARCH ALGORITHMS

In this section, we compare the genetic algorithm with other search algorithms such as local search, taboo search and simulated annealing. In computer simulations, the neighborhood structure based on the shift change mutation was used in all the search algorithms. The same stopping condition as in the genetic algorithm was also used in all the search algorithms.

Table 3. Comparison of various mutation operators

	Number of evaluations		
Mutation operator	10,000	20,000	50,000
Adjacent 2-job change	106.29	105.89	105.26
Arbitrary 2-job change	101.53†	101.06†	100.47†
Arbitrary 3-job change	101.69	101.21	100.82
Shift change	100.44*	100.19*	100.00*

*·† show the best result and the second best result in each column.

Table 4. Comparison of four mutation operators

	50,000 evaluations	
Mutation operator	Average	Standard deviation
Adjacent 2-job change	105.41	2.2647
Arbitrary 2-job change	100.41	0.7739
Arbitrary 3-job change	100.72	0.8492
Shift change	100.00	0.7047

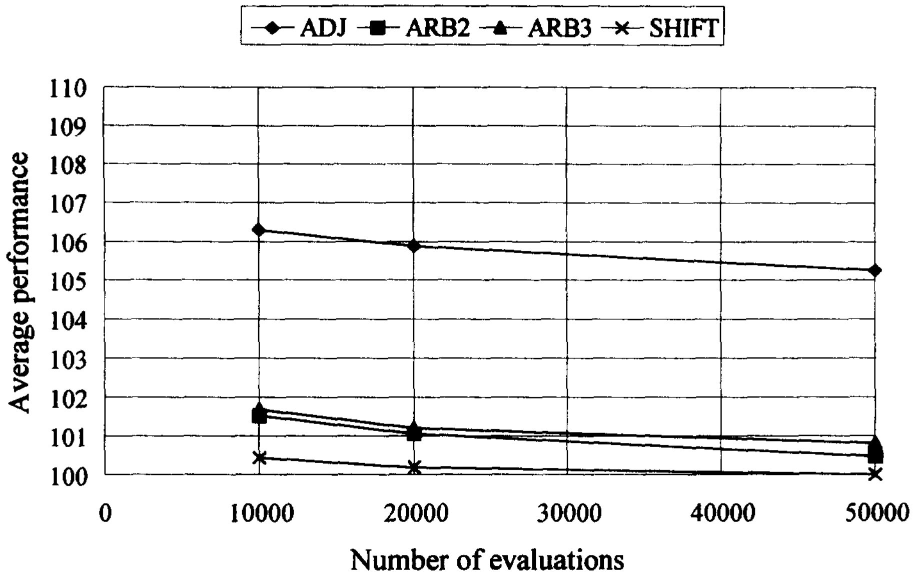


Fig. 4. Simulation results by the four mutation operators (ADJ: Adjacent 2-job change, ARB2: Arbitrary 2-job change, ARB3: Arbitrary 3-job change, SHIFT: Shift change).

3.1. Local search algorithm

In a local search algorithm, first an initial solution **x** is randomly generated. Then the solutions in the neighbor of the current solution **x** are examined in random order. When a better solution is found by this neighborhood examination, the current solution is immediately replaced with that solution. That is, the current solution is replaced with the first solution that improves the current one. On the other hand, when there is no solution that improves the current one in the neighborhood, the current solution **x** can be regarded as a local optimal solution. Then the search procedure restarts from another initial solution that is generated randomly again. This search procedure is iterated until a prespecified stopping condition is satisfied.

3.2. Taboo search algorithm

In computer simulations, we used the following taboo search algorithm, which is basically the same as an algorithm mentioned in Taillard [4]. First an initial solution **x** is randomly generated, and the taboo list is specified as ϕ where ϕ shows that the taboo list is empty. Next the neighborhood solutions that are not included in the taboo list are examined in random order. Let **y*** be the first solution that improves the current one. If no solution improves the current one, then let **y*** be the best solution in the neighborhood solutions that are not included in the taboo list. If a prespecified stopping condition is satisfied, then stop the search procedure. Otherwise, let **x**: = **y***, renew the taboo list, and continue the search procedure from the updated current solution **x**. In computer simulations, we used the taboo list defined by the pairs of positions and jobs. When the job x_j at the j-th position is removed and put at another position, the pair (j, x_j) is added to the taboo list. The length of the taboo list was specified as seven.

3.3. Simulated annealing algorithm

In this paper, we used the simulated annealing algorithm proposed by Osman and Potts [2]. First an initial solution **x** is randomly generated. The transition from a current solution **x** to a neighborhood solution **y** is accepted by the following probability:

$$P(\mathbf{x} \rightarrow \mathbf{y}) = \min\left\{1, \exp\left(-\frac{f(\mathbf{y}) - f(\mathbf{x})}{c_k}\right)\right\}, \quad k = 1, 2, \ldots, N, \tag{4}$$

where c_k is a control parameter called temperature and N is the total iteration number of the simulated annealing algorithm. In simulated annealing, the value of the control parameter is gradually decreased from a large initial value to a small final value. In computer simulations, we specified the sequence of c_k as

$$c_{k+1} = c_k/(1 + \beta \cdot c_k), \quad k = 1,2,\ldots,N-1, \tag{5}$$

where β is positive constant. We determined the positive constant β and the initial values of c_k according to Osman and Potts [2]. The current solution **x** is replaced with the candidate solution **y** with the acceptance probability in (4). This search procedure is iterated until a prespecified stopping conditions (e.g., the number of iterations $N = 200{,}000$) is satisfied.

3.4. *Simulation results*

As in a similar manner to the computer simulations in the last section, we applied the genetic algorithm (GA), the local search algorith (LS), the taboo search algorithm (TS) and the simulated annealing algorithm (SA) to the 100 test problems with 20 jobs and 10 machines (Fig. 5). For comparison, we also applied a random sampling technique with the same computation load as the other algorithms. We also applied these algorithms to randomly generated 100 test problems with 50 jobs and 10 machines.

Simulation results are shown in Tables 5 and 6 for 20-job problems and 50-job problems, respectively. In these tables, average makespans obtained by each algorithm are normalized using the results by the simulated annealing algorithm with 200,000 evaluations. From these tables, we can see that the genetic algorithm, which is much superior to the random sampling technique, is a bit inferior to the other search algorithms.

4. HYBRID ALGORITHMS

In this section, we examine two hybrid genetic algorithms (i.e., genetic local search and genetic simulated annealing) to improve the performance of the genetic algorithm.

4.1. *Genetic local search algorithm*

Genetic local search algorithms have been proposed by several authors for mainly traveling salesman problems (for example, see Jog *et al.* [8], Ulder *et al.* [10] and Glass *et al.* [12]). In computer simulations, we used the following genetic local search algorithm.

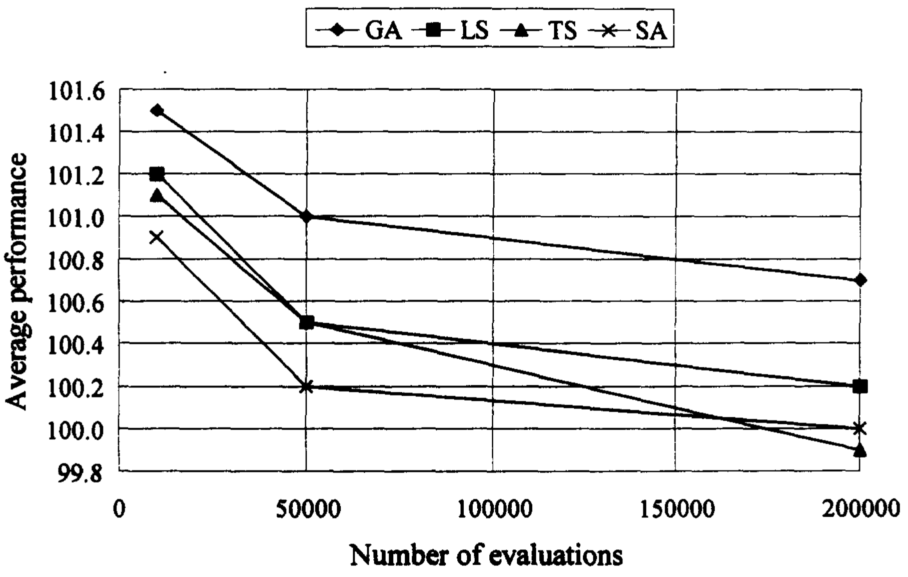


Fig. 5. Simulation results by the four search algorithms.

Table 5. Comparison of the search algorithms for 20-job problems

Evaluations	10,000	50,000	200,000
GA	101.5	101.0	100.7
LS	101.2	100.5	100.2
TS	101.1†	100.5†	100.0*
SA	100.9*	100.2*	100.0†
Random	109.6	108.4	107.5

*·† show the best result and the second best result in each column.

Table 6. Comparison of the search algorithms for 50-job problems

Evaluations	10,000	50,000	200,000
GA	102.3	101.4	101.1
LS	101.9	101.2	100.7
TS	101.4†	101.0†	100.5†
SA	101.2*	100.4*	100.0*
Random	111.4	110.5	109.8

*·† show the best result and the second best result in each column.

Step 1 (Initialization).

Step 2 (Local search and termination test). Apply the local search algorithm to the N_{pop} solutions in the current population. If a prespecified stopping condition is satisfied during the local search, stop the algorithm. If the local search is completed for all the N_{pop} solutions, let the set of the obtained N_{pop} local optimal solutions be the current population, and go to Step 3.

Step 3 (Selection).

Step 4 (Crossover).

Step 5 (Mutation).

Step 6 (Elitist strategy).

Step 7. Return to Step 2.

One difficulty of this genetic local search algorithm is its enormous computation time for finding N_{pop} local optimal solutions in each generation. When we applied the genetic local search algorithm to 100 test problems with 20 jobs, the average number of generations was 7.4 (in the case of $N_{pop} = 10$ and 50,000 evaluations). In order to reduce the computation time for finding local optimal solutions, we propose a strategy not to search all the neighborhood solutions but to search a part of them. For example, we can use the strategy to search randomly selected 10% neighborhood solutions in each local search procedure. If there are no solutions that improve the current one in the 10% neighborhood solutions, the local search procedure is terminated while all the neighborhood solutions are not examined.

4.2. Genetic simulated annealing

In the genetic local search algorithm, we can use the simulated annealing instead of the local search to construct a genetic simulated annealing algorithm. That is, we have the genetic simulated annealing algorithm only modifying Step 2 of the genetic local search algorithm in the last subsection. In computer simulations, we applied the simulated annealing algorithm with the constant temperature to each of the N_{pop} solutions in the current population. We used the constant temperature to avoid extreme deterioration of the current solution during the initial state of annealing with high temperatures. The simulated annealing algorithm was iterated 500 times for each solution of the current population in computer simulations. In order to improve the

Table 7. Performance of the genetic local search and the genetic simulated annealing for 20-job problems

Search	Number of evaluations		
algorithms	10,000	50,000	200,000
GA	101.48	101.03	100.71
GLS (100%)	101.14	100.14	99.85†
GLS (75%)	101.05	100.12*	99.82*
GLS (50%)	101.04†	100.18	99.92
GLS (25%)	101.02*	100.19	99.97
GLS (10%)	101.16	100.28	100.01
GLS (5%)	101.25	100.52	100.24
GSA ($k = 1$)	101.25	100.25	99.92
GSA ($k = 2$)	101.18	100.20	99.93
GSA ($k = 4$)	101.21	100.22	99.96
GSA ($k = 6$)	101.10	100.12†	99.87
GSA ($k = 8$)	101.26	100.23	99.92
GSA ($k = 10$)	101.27	100.17	99.92
GSA ($k = 20$)	101.51	100.20	99.94

*·† show the best result and the second best result in each column.

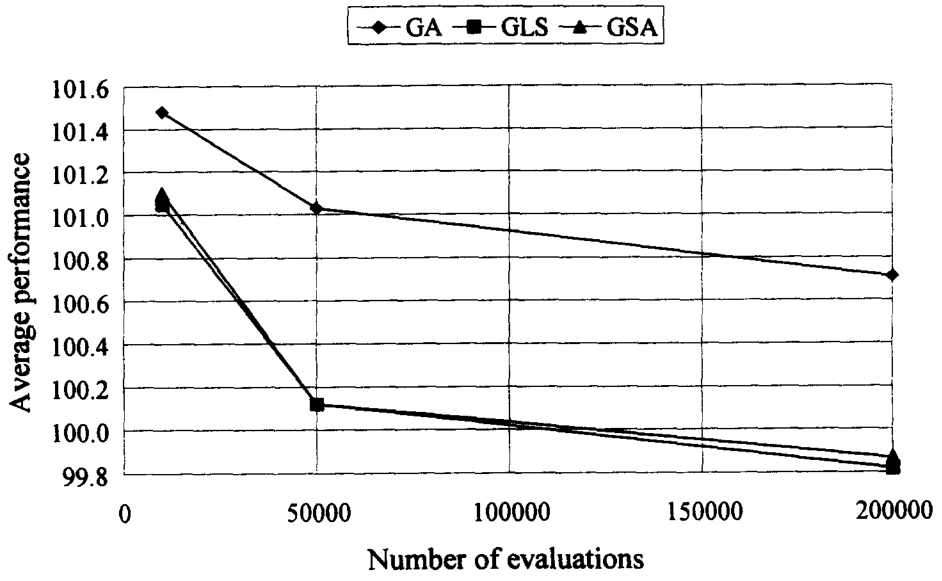


Fig. 6. Simulation results by GA, GLS ($\alpha = 75\%$) and GSA ($k = 6$).

performance of the genetic simulated annealing algorithm, we modify the simulated annealing algorithm by randomly selecting k neighborhood solutions of the current one and letting the best one be the candidate solution for the next transition in the simulated annealing (for such a modification of simulated annealing, see Ishibuchi *et al.* [19]).

4.3. *Simulation results*

As in a similar manner to the computer simulations in the last section, we applied the genetic local search algorithm (GLS) and the genetic simulated annealing algorithm (GSA) to the 100 test problems with 20 jobs and 10 machines. The simulation results are shown in Table 7. In the GLS, $\alpha\%$ ($\alpha = 100, 75, 50, 25, 10, 5$) neighborhood solutions were examined in each local search procedure. When $\alpha = 100$, the local search algorithm in the GLS is the same as the standard local search algorithm. In the GSA, k ($k = 1, 2, 4, 6, 8, 10, 20$) candidate solutions were selected from the neighbor of the current solution, and the transition to the best solution was examined with the acceptance probability (4) of the simulated annealing algorithm. When $k = 1$, the simulated annealing in the GSA is the same as the standard algorithm. In this paper, the constant temperature in the GSA was specified as $c = 2$. In Fig. 6 we plotted the results obtained by GA, GLS ($\alpha = 75$) and GSA ($k = 6$). From the comparison between the results in Tables 5 and 7, we can see that the performance of the genetic algorithm was improved by being combined with local search and simulated annealing. Table 7 shows that the reduction of searching space in the GLS is effective to improve the performance of the GLS. Table 7 also shows that the introduction of k into the simulated annealing improved the performance of the genetic simulated annealing algorithm. From Tables 5 and 7, we can see that the best result was obtained by the GLS with $\alpha = 75\%$. It should be noted that not only genetic algorithm but also the local search and thè simulated annealing were improved by the hybridization (compare Table 5 with Table 7).

5. CONCLUSION

In this paper, we examined the performance of genetic algorithms in order to specify some genetic operators (e.g., crossover and mutation) and parameters (e.g., population size, crossover probability and mutation probability) for the flowshop scheduling problem. By computer simulations, we showed that the genetic algorithm was much superior to a random sampling technique but it was a bit inferior to other search algorithms such as local search, taboo search and simulated annealing. In order to improve the performance of the genetic algorithm, we examined two hybrid algorithms: genetic local search and genetic simulated annealing. We also examined some variants of their hybrid algorithms. By computer simulations, we showed that the hybrid algorithms outperformed the other algorithms (i.e., the genetic algorithm, the local search algorithm, the taboo search algorithm and the simulated annealing algorithm).

Acknowledgement—This work was partially supported by Grant-in-Aid for International Scientific Research (No. 06045047) from the Ministry of Education, Science and Culture in Japan.

REFERENCES

1. S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logistics Q.* **1,** 61–68 (1954).
2. I. H. Osman and C. N. Potts. Simulated annealing for permutation flow-shop scheduling. *OMEGA* **17,** 551–557 (1989).
3. M. Widmer and A. Hertz. A new heuristic method for the flowshop sequencing problem. *Europ. J. Opnl Res.* **41,** 186–193 (1990).
4. E. Taillard. Some efficient heuristic methods for the flowshop sequencing problem. *Europ. J. Opnl Res.* **47,** 65–74 (1990).
5. J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor (1975).
6. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA (1989).
7. L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York (1991).
8. P. Jog, J. Y. Suh and D. V. Gucht. The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. *Proc. of the Third ICGA,* 110–115 (1989).
9. T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley. A comparison of genetic sequencing operators. *Proc. of the Fourth ICGA,* 69–76 (1991).
10. N. L. J. Ulder, E. H. L. Aarts, H.-J. Bandelt, P. J. M. van Laarhoven and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Parallel Problem Solving from Nature* (Edited by H.-P. Schwefel and R. Manner), pp. 109–116. Springer, Berlin (1991).
11. B. R. Fox and M. B. McMahon. Genetic operations for sequencing problems. In *Foundations of Genetic Algorithms* (Edited by G. J. E. Rawlins), pp. 284–300. Morgan Kaufmann Publishers, San Mateo (1991).
12. C. A. Glass, C. N. Potts and P. Shade. Genetic algorithms and neighborhood search for scheduling unrelated parallel machines. Preprint series, No. OR47, University of Southampton (1992).
13. H. Ishibuchi, N. Yamamoto, T. Murata and H. Tanaka. Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems. *Fuzzy Sets and Systems* **67,** 81–100 (1994).
14. B. Manderick and P. Spiessens. How to select genetic operators for combinatorial optimization problems by analyzing their fitness landscape. *Computational Intelligence Imitating Life,* pp. 170–181. IEEE Press, New York (1994).
15. G. Syswerda. Scheduling optimization using genetic algorithms. In *Handbook of Genetic Algorithms* (Edited by L. Davis), pp. 332–349. Van Nostrand Reinhold, New York (1991).
16. R. A. Dudek, S. S. Panwalkar and M. L. Smith. The lessons of flowshop scheduling research. *Ops Res.* **40,** 7–13 (1992).
17. D. Whitley, T. Starkweather and D. Fuquay. Scheduling problems and traveling salesmen: the genetic edge recombination operator. *Proc. of the Third ICGA,* 133–140 (1989).
18. I. Oliver, D. Smith and J. Holland. A study of permutation crossover operators on the traveling salesman problem. *Proc. of the Second ICGA,* 224–230 (1987).
19. H. Ishibuchi, S. Misaki and H. Tanaka. Modified simulated annealing algorithms for the flow shop sequencing problem. *Europ. J. Opnl Res.* **81,** 388–398 (1995).