Performance Evaluation of Genetic Algorithms for Flowshop Scheduling Problems

Tadahiko Murata and Hisao Ishibuchi
Department of Industrial Engineering, University of Osaka Prefecture
Gakuen-cho 1-1, Sakai, Osaka 593, JAPAN

Abstract — **The aim of this paper is to evaluate the performance of genetic algorithms for the flowshop scheduling problem with an objective of minimizing the makespan. First we examine various genetic operators for the scheduling problem. Next we compare genetic algorithms with other search algorithms such as local search, taboo search and simulated annealing. By computer simulations, it is shown that genetic algorithms are a bit inferior to the others. Finally, we show two hybrid genetic algorithms: genetic local search and genetic simulated annealing. Their high performance is demonstrated by computer simulations.**

I. 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]. 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 & Potts[2] proposed simulated annealing heuristics, and Widmer & Hertz[3] and Taillard[4] proposed taboo search heuristics.

General assumptions of the flowshop scheduling problem can be written as follows (see Dudek *et al.*[5]). 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 and m-machine sequencing problem.

Recently many authors applied genetic algorithms to traveling salesman problems (for example, Jog *et al.*[6], Starkweather *et al.*[7] and Ulder *et al.*[8]) and also to scheduling problems (for example, Fox & McMahon[9], Glass *et al.*[10], Ishibuchi *et al.*[11] and Syswerda[12]). Some empirical studies[8,10,11] showed that the ability of genetic algorithms to find near optimal solutions is inferior to other search algorithms. In this paper, we compare genetic algorithms with local search, taboo search and simulated annealing algorithms by applying these algorithms to the flowshop scheduling problem.

II. GENETIC OPERATORS

In this section, we examine various genetic operators for the flowshop scheduling problem before comparing genetic algorithms with other search algorithms in the next section.

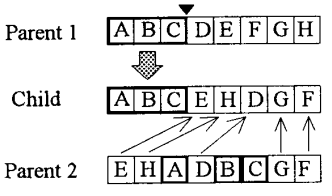
A. Crossover

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. Let us denote the sequence (*i.e.*, the order of processing) of n jobs by an n-dimensional vector $x = (x_1, x_2, \ldots x_n)$ where x_k denotes the k-th processing job. Crossover is an operation to generate a new sequence from two sequences.

In this paper, we examine various crossover operators. The following simple crossovers are effective for the flowshop scheduling problem as will be shown by computer simulations.

(1) One-point crossover.

This crossover is illustrated by the following figure:

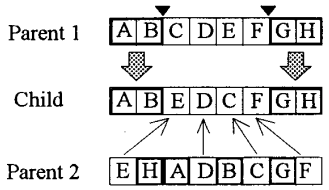


One point is randomly selected for dividing one parent. The jobs on one side (each side is chosen with the same probability) are inherited from the parent to the child, and the other jobs are placed in the order of them appeared in the other parent.

0-7803-1899-4/94 $4.00 ©1994 IEEE

(2) Two-point crossover (Version I)

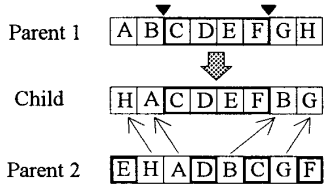
This crossover is illustrated by the following figure:



Two points is randomly selected for dividing one parent. The jobs outside the selected two points are always inherited from one parent to the child.

(3) Two-point crossover (Version II)

This crossover is illustrated by the following figure:



The jobs between randomly selected two points are always inherited from one parent to the child.

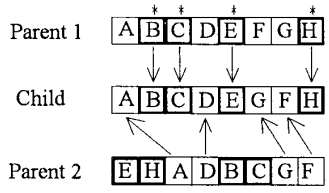
(4) Two-point crossover (Version III)

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

Position based crossovers proposed by Syswerda[12] are also examined in the following versions:

(5) Position based crossover (Version I)

This crossover is illustrated by the following figure:



The jobs at randomly selected positions marked by "*" are inherited from one parent to the child. The number of those positions is a random integer in [1, n].

(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.

The following four crossover operators, which were mainly proposed for traveling salesman problems, are also examined in this paper for the flowshop scheduling problem.

(7) Edge recombination crossover in Whitley [13]

(8) Enhanced Edge recombination crossover in Starkweather[7]

(9) Partially matched crossover in Goldberg[14]

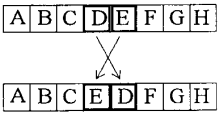
(10) Cycle crossover in Oliver[15]

B. Mutation

Mutation is an operation to change the order of n jobs in the generated child. This operation can be viewed as a transition from a current solution to its neighborhood solution in local search algorithms. We examine the following four mutations for the flowshop scheduling problem.

(1) Adjacent two-job change

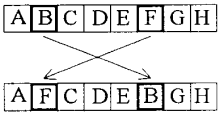
Adjacent two jobs are changed as follows:



The adjacent two jobs to be changed are randomly selected.

(2) Arbitrary two-job change

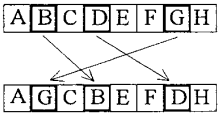
Arbitrary selected two jobs are changed as follows:



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

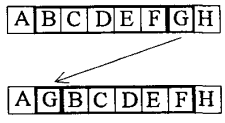
Arbitrary selected three jobs are changed as follows:



The three jobs to be changed are arbitrary and randomly selected, and the order of the selected jobs after mutation is randomly specified. This mutation includes the above two mutations as a special case.

(4) Shift change

In this mutation, a job at one position is removed and put at another position as follows:



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.

C. Selection

Selection is an operation to select two parents for generating a new child by one of the above crossover operators. 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 $\Psi_t = \{x_t^1, x_t^2, \ldots, x_t^{N_{pop}}\}$. Each solution x_t^i is selected as a parent according to the selection probability $P_S(x_t^i)$. We used the following selection probability in computer simulations:

$$P_S(x_t^i) = \frac{[f_M(\Psi_t) - f(x_t^i)]^2}{\sum_{x_t^j \in \Psi_t} [f_M(\Psi_t) - f(x_t^j)]^2}, \quad (1)$$

where $f(x_t^i)$ is the objective function to be minimized in the flowshop scheduling problem (*i.e.*, makespan) and

$$f_M(\Psi_t) = \max\{f(x_t^i) \mid x_t^i \in \Psi_t\}. \quad (2)$$

D. 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 including N_{pop} solutions.

Step 2 (Selection):
Select N_{pop} pairs of solutions 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_X. If the crossover operator is not applied to the selected pair, one of the selected solution remains as a new child.

Step 4 (Mutation):
Apply one of the above mutation operators to each of the generated N_{pop} solutions with the mutation probability P_m.

Step 5 (Elitist strategy):
Randomly remove one solution from the current population and add the best solution 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.

E. Comparison between Various Specifications

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].

By computer simulations on the test problems using genetic algorithms with various parameter specifications, we have the following specifications with high performance.

Population size: $N_{pop} = 10$,
Crossover probability: $P_X = 1$,
Crossover: Two-point crossover (Version I),
Mutation probability: $P_m = 1$,
Mutation: Shift change mutation.

As a stopping condition, we used the total number of evaluations of the objective function in order to compare various parameter specifications under the same computation load. In computer simulations, we used the following two stopping conditions: The total number of evaluations of the objective function is 10000 or 50000.

Table 1 Comparison of various crossover operators

Number of evaluations	10000	50000
One-point crossover	1571.6	1561.4
Two-point: Ver.I	1565.9**	1558.9**
Two-point: Ver.II	1572.2	1562.0
Two-point: Ver.III	1569.3*	1559.5*
Position based: Ver.I	1576.1	1563.6
Position based: Ver.II	1571.6	1562.8
Edge Recombination	1636.1	1598.3
Enhanced E.R.	1639.2	1605.6
Partially Matched	1569.7	1561.7
Cycle crossover	1572.1	1561.3

"**" and "*" shows the best result and the second best result in each column

First we show simulation results with various crossover operators in Table 1. From this table, 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.

Next we show simulation results with various mutation operators in Table 2. From this table, we can observe the high performance of the shift change mutation.

Table 2 Comparison of various mutation operators

Number of evaluations	10000	50000
Adjacent 2-job change	1657.0	1641.0
Arbitrary 2-job change	1582.8	1566.3
Arbitrary 3-job change	1577.9*	1565.7*
Shift change	1565.9**	1558.9**

In order to specify the other parameters, we also examined the following values of the population size, the mutation probability and the crossover probability:

N_{pop} = 5, 10, 20, 30, 40, 50,
P_X = 0.5, 0.6, 0.7, 0.8, 0.9, 1,
P_m = 0.5, 0.6, 0.7, 0.8, 0.9, 1.

Among these values, we obtained the best result by the genetic algorithm with the above mentioned parameter specifications.

III. 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.

A. Local Search Algorithm

As local search, we used the following algorithm in computer simulations.

Step 1 (Initialization):
Randomly generate an initial solution x.

Step 2 (Neighborhood search):
Examine the solutions in the neighbor of the current solution x in random order. Let y^* be the first solution that improves the current one. If there is no solution in the neighborhood that improves the current one, then let y^* be ϕ where ϕ shows that y^* is empty.

Step 3 (Termination test):
If a prespecified stopping condition is satisfied, then stop this algorithm. Otherwise, return to Step 1 if y^* is empty. If y^* is not empty, let x:=y^* and return to Step 2.

B. Taboo Search Algorithm

In computer simulations, we used the following taboo search algorithm, which is basically the same as one algorithm mentioned in Taillard[4].

Step 1 (Initialization):
Randomly generate an initial solution x, and set the taboo list as ϕ where ϕ shows that the taboo list is empty.

Step 2 (Neighborhood search):
Examine the neighborhood solutions that are not included in the taboo list 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.

Step 3 (Termination test):
If a prespecified stopping condition is satisfied, then stop this algorithm. Otherwise, let x:=y^*, renew the taboo list, and return to Step 2.

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.

C. Simulated Annealing Algorithm

In simulated annealing, the transition from a current solution x to a neighborhood solution y is accepted by the following probability:

$$P(x \to y) = \min\{1,\ \exp[-(f(y) - f(x))/c_i]\}, \quad (3)$$

where c_i is a control parameter called temperature. In simulated annealing algorithms, the value of the control parameter is gradually decreased from a large initial value to a small final value. If the sequence of c_i is given as $c_1, c_2, \ldots, c_N$ (N is the total iteration number of the algorithm), a simulated annealing algorithm can be written as follows.

Step 1 (Initialization):
Let i:=0, and randomly generate an initial solution x.

Step 2 (Neighborhood search):
Let $i := i + 1$, and randomly choose a solution y from the neighborhood of the current solution x. Replace x by y with the acceptance probability defined by (3).

Step 3 (Termination test):
If $i \geq N$, then stop this algorithm. Otherwise, return to Step 2.

In computer simulations, we specified the sequence of c_i as

$$c_{i+1} = c_i / (1+\beta c_i), \quad i = 1,2,\ldots,N-1, \tag{4}$$

where β is a positive constant, and c_1 and c_N are initial and final values of c_i. As in Osman & Potts[2], these parameters were specified as follows in computer simulations.

$$\beta = (c_1 - c_N) / (c_1 c_N (N-1)), \tag{5}$$

$$c_1 = \sum_{j=1}^{m} \sum_{i=1}^{n} t_P(i,j) / 5mn \quad \text{and} \quad c_N = 1, \tag{6}$$

where $t_P(i,j)$ is the processing time of the job j on the machine i.

D. Simulation Results

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

Simulation results are shown in Table 3 and Table 4 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 200000 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.

Table 3 Comparison of the search algorithms for 20-job problems

Evaluations	10000	50000	200000
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

Table 4 Comparison of the search algorithms for 50-job problems

Evaluations	100000	50000	200000
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

IV. HYBRID ALGORITHMS

In this section, we show some hybrid genetic algorithms to improve the performance of the genetic algorithm.

A. 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.*[6], Ulcer *et al.*[8] and Glass *et al.*[10]). In computer simulations, we used the following genetic local search algorithm.

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. In order to reduce the computation time, 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 before searching all the neighborhood solutions.

B. Genetic Simulated Annealing

In the genetic local search algorithm, we can use the simulated annealing instead of the local search algorithm. In computer simulations, we applied the simulated annealing algorithm with the constant temperature to each of the N_{pop} solutions in the current population in Step 2. The simulated annealing algorithm was iterated 300 times for each solution in computer simulations.

C. Simulation Results

As in a similar manner to the last section, we applied the genetic local search algorithm (GLS) and the genetic simulated annealing algorithm (GSA) to the test problems. The simulation results are shown in Tables 5 and 6. In the GLS, the algorithm searched α% (α=100, 75, 50, 25, 10, 5) neighborhood solutions in each local

search procedure. In the GSA, the constant temperature was specified as c=5 or c=2. From the comparison between the results in Tables 3~6, we can see that the performance of the genetic algorithm is improved by being combined with local search and simulated annealing. Especially, the genetic local search algorithm (GLS) with α=75% is the best for 20-job problems among the examined algorithms in Tables 3 and 5.

Table 5 Performance of the genetic local search and the genetic simulated annealing for 20-job problems

Evaluations	10000	50000	200000
GLS(100%)	101.1	100.1*	99.9*
GLS (75%)	101.1	100.1**	99.8**
GLS (50%)	101.0*	100.2	99.9
GLS (25%)	101.0**	100.2	100.0
GLS (10%)	101.2	100.3	100.0
GLS (5%)	101.3	100.5	100.2
GSA (c=5)	101.5	100.5	100.2
GSA (c=2)	101.1	100.2	100.0

Table 6 Performance of the genetic local search and the genetic simulated annealing for 50-job problems

Evaluations	10000	50000	200000
GLS(100%)	102.0**	101.0	100.2
GLS (75%)	102.0*	101.1	100.2
GLS (50%)	102.1	101.0	100.2
GLS (25%)	102.3	100.9	100.1*
GLS (10%)	102.7	100.8*	100.1**
GLS (5%)	102.6	100.7**	100.2
GSA (c=5)	102.7	101.2	100.6
GSA (c=2)	102.2	100.8	100.3

V. CONCLUSION

In this paper, we examined the performance of the genetic algorithm 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. We also demonstrated the high performance of hybrid genetic algorithms with local search and simulated annealing.

REFERENCES

[1] S.M.Johnson, Optimal two- and three-stage production schedules with setup times included, *Navel Research Logistics Quarterly*, **1**(1) (1954) 61-68.

[2] I.H.Osman and C.N.Potts, Simulated annealing for permutation flow-shop scheduling, *OMEGA*, **17**(6) (1989) 551-557.

[3] M.Widmer and A.Hertz, A new heuristic method for the flowshop sequencing problem, *European J. of Operational Research*, **41**(2) (1990) 186-193.

[4] E.Taillard, Some efficient heuristic methods for the flowshop sequencing problem, *European J. of Operational Research*, **47**(1) (1990) 65-74.

[5] R.A.Dudek, S.S.Panwalkar and M.L.Smith, The lessons of flowshop scheduling research, *Operations Research*, **40**(1) (1992) 7-13.

[6] 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* (1989) 110-115.

[7] T.Starkweather, S.McDaniel, K.Mathias, D.Whitley and C.Whitley, A comparison of genetic sequencing operators, *Proc. of the Fourth ICGA* (1991) 69-76.

[8] 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 H.-P.Schwefel and R.Manner (Eds.) *Parallel Problem Solving from Nature* (Springer-Verlag, Berlin, 1991) 109-116.

[9] B.R.Fox and M.B.McMahon, Genetic operations for sequencing problems, in G.J.E.Rawlins (ed.) *Foundations of Genetic Algorithms* (Morgan Kaufmann Publishers, San Mateo, 1991) 284-300.

[10] 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).

[11] H.Ishibuchi, N.Yamamoto, T.Murata and H.Tanaka, Genetic Algorithms and Neighborhood Search Algorithms for Fuzzy Flowshop Scheduling Problems, *Fuzzy Sets and Systems* (in press).

[12] G.Syswerda, Scheduling optimization using genetic algorithms, in L.Davis (ed.) *Handbook of Genetic Algorithms* (Van Nostrand Reinhold, New York, 1991) 332-349.

[13] D.Whitley, T.Starkweather and D.Fuquay, Scheduling problems and traveling salesmen: The genetic edge recombination operator, *Proc. of the Third ICGA* (1989) 133-140.

[14] D.E.Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, 1989), 147-215.

[15] I.Oliver, D.Smith and J.Holland, A study of permutation crossover operators on the traveling salesman problem, *Proc. of the Second ICGA* (1987) 224-230.