CHAPTER 4

GENETIC ALGORITHMS FOR MULTI-OBJECTIVE FLOWSHOP SCHEDULING PROBLEMS

4.1 INTRODUCTION

In Chapter 3, we considered genetic algorithms for single-objective flowshop scheduling problems. As shown in the previous chapter, GAs have been mainly applied to single-objective optimization problems. Many real-world problems, however, have multiple objectives. Since Johnson's work [56], various scheduling criteria have been considered (see, for example, reviews by Baker & Scudder [3] and Dudek *et al.*[10]). Among them are makespan, maximum tardiness, total tardiness, and total flowtime. Several researchers extended single-objective flowshop scheduling problems to multi-objective problems. For example, Ho & Chang [26] proposed a heuristic method for flowshop scheduling with bicriteria, Gangadharan *et al.*[20] proposed a simulated annealing heuristic for flowshop scheduling with bicriteria. Daniels & Chambers [7] considered the trade-off between the makespan and the maximum tardiness. Rajendran [95] proposed a branch-and-bound algorithm and two heuristic algorithms to minimize the total flowtime with a constraint condition on the makespan. Morizawa *et al.*[74] proposed a random sampling method for obtaining a set of non-dominated solutions of a flowshop scheduling problem with two objectives: to minimize the makespan and the maximum tardiness. A three-objective flowshop scheduling problem was considered in Morizawa *et al.*[75] where the makespan, the maximum tardiness and the total flowtime were used as scheduling criteria. Morizawa *et al.*[76] also proposed an interactive approach for searching a preferred schedule of multi-objective problems.

Since Schaffer's work [98], extensions of GAs to multi-objective optimization were proposed in several manners (*e.g.*, see Fonseca & Fleming [14,15], Gen *et al.* [21], Horn *et*

al.[30], Kita *et al.*[60], Kursawe [63], Murata & Ishibuchi [77,78,81], and Tamaki *et al.* [111,112]). We have already compared the multi-objective genetic algorithm described in Section 2.3 with the VEGA [98] and the NPGA [30].

This chapter addresses the application of GAs to multi-objective flowshop scheduling problems. We demonstrate the effectiveness of the MOGA on flowshop scheduling problems with two objectives and problems with three objectives. By two-objective flowshop scheduling problems, we compare the MOGA with single-objective genetic algorithms where one of two objectives is used as a fitness function. Next we examine the relation between the number of elite solutions to be inherited and the performance of the MOGA. Then we compare the MOGA with the VEGA and a constant weight genetic algorithm (CWGA). In the CWGA, two objectives are combined into a single scalar fitness function using constant weights. Last we apply the MOGA to three-objective flowshop scheduling problems. We hybridize the MOGA with a local search algorithm in the same manner as in Chapter 3. The effectiveness of the hybrid algorithm is shown by computer simulations.

4.2 GENETIC ALGORITHMS FOR MULTI-OBJECTIVE FLOWSHOP SCHEDULING PROBLEMS

In this section, first we compare the multi-objective genetic algorithm with single-objective genetic algorithms (SOGA) where one of two objectives is used for the fitness function. Next we examine the relation between the number of elite solutions to be inherited and the performance of the MOGA. Then we compare the MOGA with the VEGA and a constant weight genetic algorithm (CWGA). In the CWGA, two objectives are combined into a single scalar fitness function using constant weights. Last we apply the MOGA to a three-objective flowshop scheduling problem. In the SOGA, the VEGA, and the CWGA, we store two sets of solutions as in the MOGA: the population to be governed by genetic operators and the set of non-dominated solutions. In order to compare these algorithms, we use the final set of non-dominated solutions obtained by each genetic algorithm for multi-objective optimization.

4.2.1 Parameter specifications

As we have already explained flowshop scheduling problems in Chapter 3, there are many criteria for scheduling problems. In the previous chapter, we treated only one criterion out of four criteria: the makespan in (3.5), the total flowtime in (3.6), the maximum tardiness in (3.7), and the total tardiness in (3.8) (see Subsection 3.2). In this section, we treat two or three objectives out of the four objectives.

In this section, we specified parameters in flowshop scheduling problems with multiple objectives as follows. The processing time of each job at each machine was randomly specified as an integer in the closed interval [1, 99]. We specified the duedate of each job by the following procedure:

Step 1: Randomly generate a permutation of n jobs.

Step 2: Calculate the completion time $t_C(m,x_k)$ of each job, $k = 1, 2, ..., n$.

Step 3: Add a random integer in the closed interval [-100, 100] to each $t_C(m,x_k)$. That is, the duedate $d(x_k)$ of the k-th job is specified as follows:

$$d(x_k) = t_C(m, x_k) + random_k\,, \quad k = 1, 2, ..., n\,. \tag{4.1}$$

where $random_k$ is a random integer in the closed interval [-100, 100].

It is known that there is no correlation between the three objectives: the makespan, the total flowtime, and the maximum tardiness (or the total tardiness). That is, we considered two objectives of the makespan and the total tardiness or of the makespan and the maximum tardiness in two-objective flowshop scheduling problems. And we considered either set of three objectives in three-objective flowshop scheduling problems.

In this section, we generated 20-job and 10-machine problems. Because the total number of feasible solutions (*i.e.*, all permutations of 20 jobs) is over 10^{18}, we can not apply enumeration methods to the problems. We apply the multi-objective genetic algorithm (MOGA) to two-objective flowshop scheduling problems and three-objective flowshop scheduling problems. We employed the two-point order crossover and the shift change mutation as genetic operators in the GA, and we specified the population size N_{pop} as $N_{\text{pop}} = 10$.

4.2.2 Two-objective flowshop scheduling problems

In this section, we apply the MOGA to a randomly generated 20-job and 10-machine flowshop scheduling problem with two objectives: to minimize the makespan and to minimize the total tardiness. Therefore we employ the following fitness function in the MOGA:

$$f(\boldsymbol{x}) = -w_1 f_1(\boldsymbol{x}) - w_4 f_4(\boldsymbol{x})\,, \tag{4.2}$$

where $f_1(\boldsymbol{x})$ and $f_4(\boldsymbol{x})$ are the objective functions described in (3.5) and (3.8), respectively, and w_1 and w_4 are non-negative weights which satisfy the relations in (2.6) and (2.7).

A. Comparison of the MOGA and the SOGA

Non-dominated solutions obtained by the MOGA are shown by ○ in Fig. 4.1 where the horizontal and vertical axes are the makespan and the total tardiness, respectively. In Fig. 4.1, non-dominated solutions obtained by the SOGA where either the makespan or the total tardiness is used for the fitness function. In Fig. 4.1, non-dominated solutions obtained by the

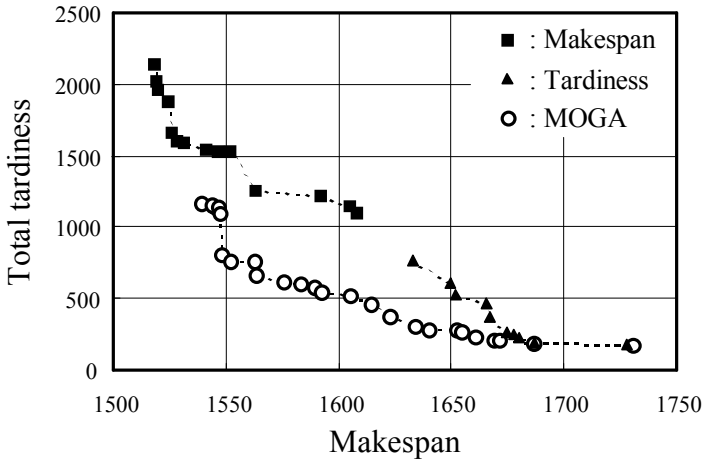


Fig. 4.1 Comparison of the MOGA with two trials of the SOGAs.

SOGA for minimizing the makespan are shown by ■, and those obtained by the SOGA for minimizing the total tardiness are shown by ▲. In order to compare the non-dominated solutions obtained by the MOGA with those obtained by the SOGAs, we specified the number of evaluations of the fitness function as 100,000 in the MOGA and as 50,000 in each trial of the SOGA. Therefore 100,000 solutions were evaluated by each of the MOGA and the SOGAs.

From Fig. 4.1, we can see that the set of the non-dominated solutions obtained by the MOGA (○) is superior to those obtained by the SOGAs (■ and ▲). That is, many solutions denoted by ■ and ▲ are dominated by solutions denoted by ○. This demonstrates the high performance of the MOGA.

B. Effectiveness of the elitist strategy in the MOGA

The effectiveness of the elitist strategy described in Subsection 2.3.4 is demonstrated in Fig. 4.2. In Fig. 4.2, "no elite", "2 elite", and "3 elite" indicate that no elite solutions, two elite solutions, and three elite solutions are inherited to the current population from the tentative set of non-dominated solutions, respectively. In the elitist strategy, we used the following heuristic: in the "2 elite" algorithm, only the elite solutions with respect to the two objective functions were preserved. In the "3 elite" algorithm, that is the MOGA, a solution which was randomly selected from the tentative set of non-dominated solutions was added to the two elite solutions in the "2 elite" algorithm. From Fig. 4.2, we can see that the MOGA could find better solutions

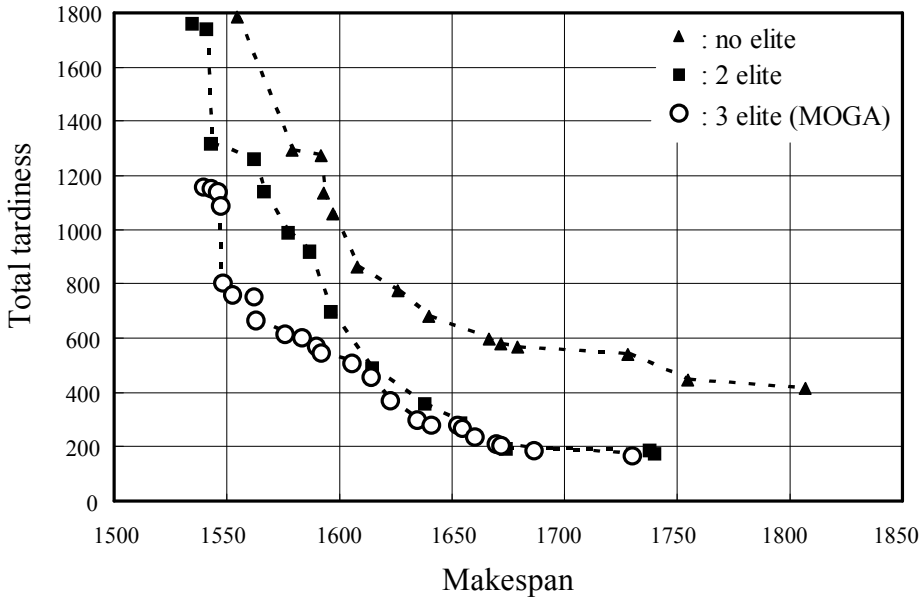
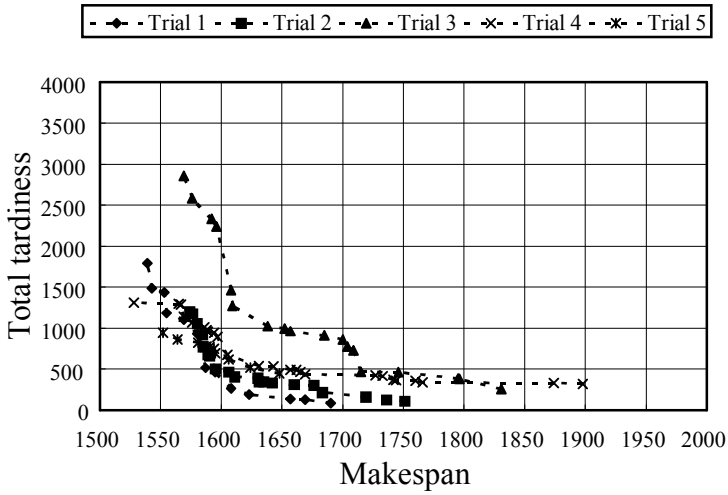


Fig. 4.2 Effect of the number of elite solutions in the MOGA.

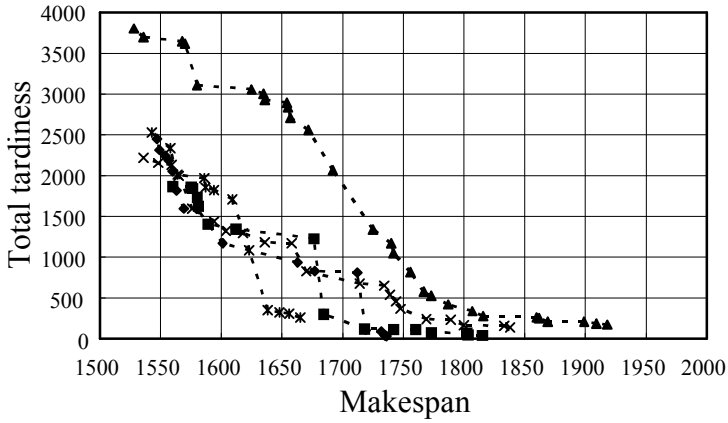
than the "2 elite" and "no elite" algorithms. This means that the elitist strategy of the MOGA is effective.

C. Comparison with the MOGA, the VEGA, and the CWGA

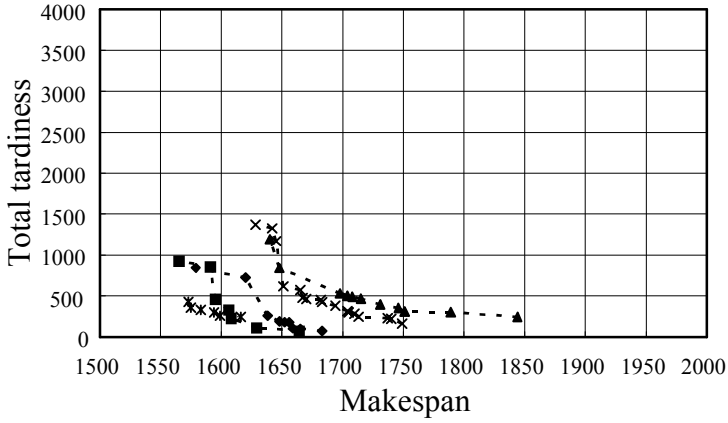
We also applied Schaffer's VEGA [98] and the constant weight genetic algorithm (CWGA) to the same flowshop scheduling problem. In the CWGA, we used the weights $w_1 = w_{\text{makespan}} = 5$ and $w_4 = w_{\text{tardiness}} = 2$ to calculate the fitness function in (4.2). As a stopping condition, we used the total number of evaluations of strings (*i.e.*, solutions). When 100,000 solutions were evaluated in each algorithm, the algorithm was terminated. It is noted that a tentative set of non-dominated solutions was also stored and updated in the VEGA and the CWGA. Simulation results by the MOGA, the VEGA and the CWGA are shown in Fig. 4.3 (a), (b), and (c), respectively. We applied each algorithm five times to the same flowshop scheduling problem. Each algorithm began to search a set of non-dominated solutions from the same initial population. From Fig. 4.3, we can see that better solutions were obtained by the MOGA. That is, many solutions obtained by the VEGA in Fig. 4.3 (b) are dominated those obtained by the MOGA solutions in Fig. 4.3 (a). The CWGA could find some better solutions than our MOGA, but the CWGA failed to find a large set of non-dominated solutions.



(a) Non-dominated solutions obtained by the MOGA.



(b) Non-dominated solutions obtained by the VEGA.



(c) Non-dominated solutions obtained by the CWGA.

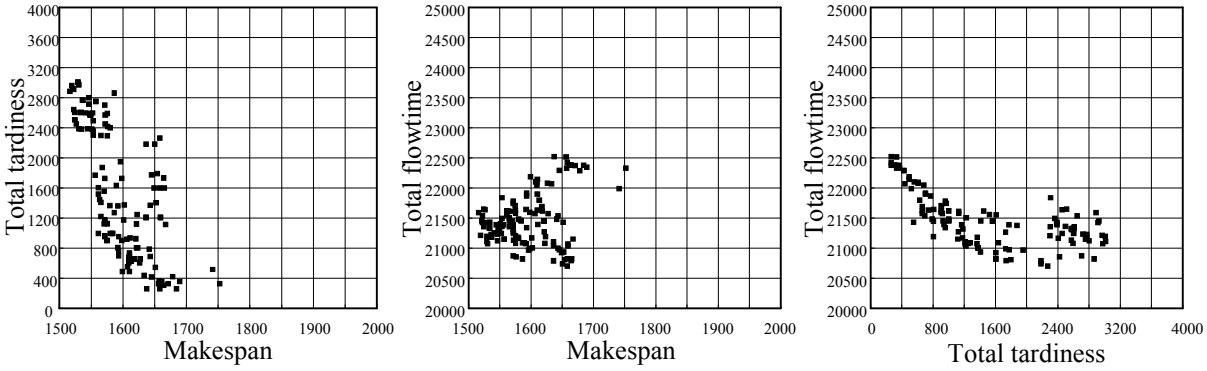
Fig. 4.3 Comparison of the MOGA, the VEGA, and the CWGA.

4.2.3 Three-objective flowshop scheduling problem

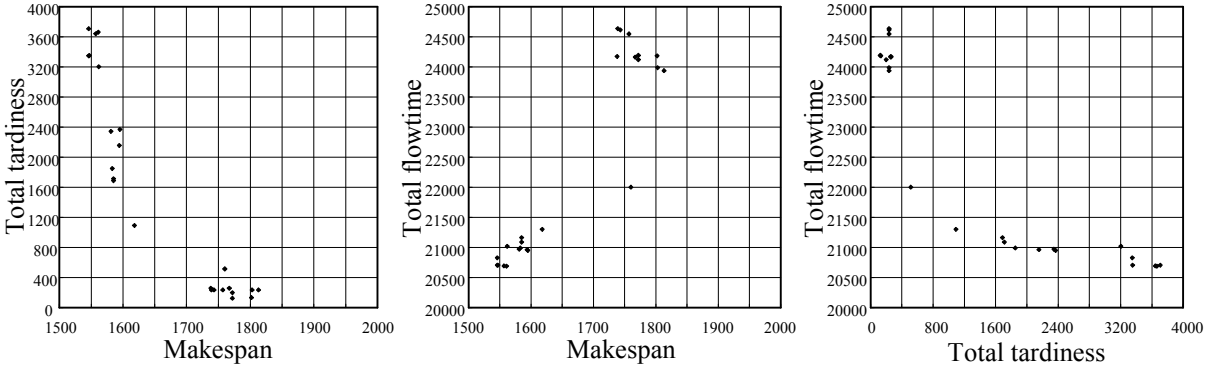
We also applied the MOGA to a flowshop scheduling problem with three objectives: to minimize the makespan, to minimize the total tardiness, and to minimize the total flowtime. Therefore we employ the following fitness function in the MOGA:

$$f(\boldsymbol{x}) = -w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}) - w_4 f_4(\boldsymbol{x}), \tag{4.3}$$

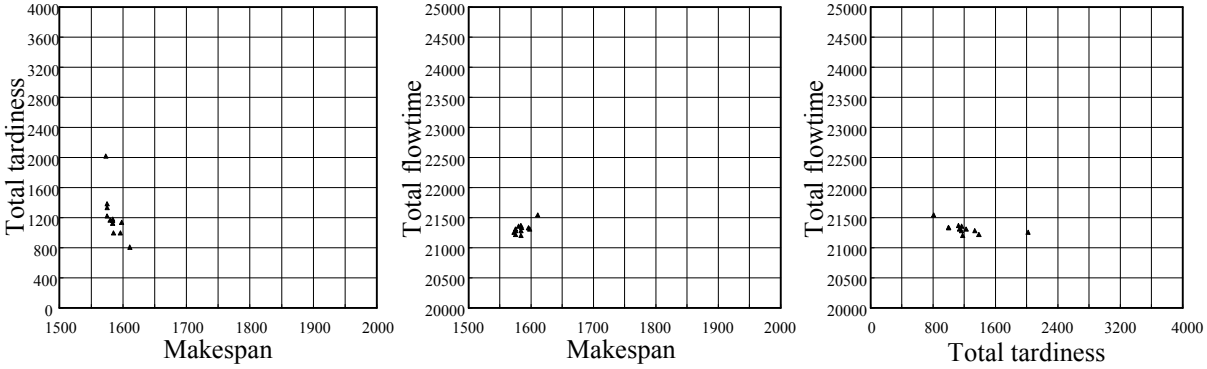
where $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$, and $f_4(\boldsymbol{x})$ are the objective functions described in (3.5), (3.6), and (3.8), respectively, and w_1, w_2, and w_4 are non-negative weights which satisfy the relations in (2.6) and (2.7). We compare the MOGA with the VEGA and the CWGA in this subsection. In the CWGA, we used the weights $w_1 = w_{\text{makespan}} = 5$, $w_2 = w_{\text{flowtime}} = 1$, and $w_4 = w_{\text{tardiness}} = 2$ to calculate the fitness function in (4.3). Because it is difficult to show obtained solutions in the three-dimensional objective space, we show the solutions by projecting them on two-dimensional objective spaces: (Makespan, Total tardiness), (Makespan, Total flowtime), and (Total tardiness, Total flowtime). Fig. 4.4 shows the simulation results obtained by the MOGA, the VEGA, and the CWGA. From Fig. 4.4, we can observe that the MOGA could find a better set of non-dominated solutions.



(a) Non-dominated solutions obtained by the MOGA.



(b) Non-dominated solutions obtained by the VEGA.



(c) Non-dominated solutions obtained by the CWGA.

Fig. 4.4 Comparison of the MOGA, the VEGA, and the CWGA.

4.3 MULTI-OBJECTIVE GENETIC LOCAL SEARCH ALGORITHM

In the same manner as in Section 3.6, we hybridize the MOGA with a local search algorithm. In this section, first we show how to hybridize the MOGA with a local search algorithm. Each solution generated by the genetic operators for multi-objective optimization (see Section 2.3) has information of weight values which were used for the selection of its parent solutions. A local search procedure is applied to the new solution to maximize its fitness value using those weight values. We also introduce the modified local search algorithm described in Subsection 3.6.1. In the modified local search procedure, only a few solutions in the neighborhood are examined. Next, we apply the multi-objective genetic local search algorithm (MOGLS) to flowshop scheduling with multiple objectives. Computer simulations show the effectiveness of the MOGLS.

4.3.1 Multi-objective genetic local search algorithm

In a multi-objective genetic local search algorithm (MOGLS) in this section, we use the same idea as in the previous section. That is, we specify the weight values by (2.8) whenever a pair of parent solutions are selected. These randomly specified weight values are also used in a local search procedure because the local search is performed to maximize the fitness function in (2.5). In our hybrid algorithm, the local search is applied to each new solution generated by the genetic operators (*i.e.*, selection, crossover, and mutation). The fitness function of the new solution is defined by the weight values that were used for selecting its parent solutions. Thus the search direction of the local search for each solution is determined by the fitness function used in the selection of its parent solutions. In this manner, each solution has its own direction of the local search. Thus both the selection operation and the local search have various search directions in the n-dimensional objective space of the multi-objective optimization problem.

Another issue to be addressed in the hybrid algorithm is how to divide the available computation time between searches by the local search and the genetic operators. If we simply combine the local search to the genetic operators, almost all the available computation time may be spent by the local search and only a few populations are generated by the genetic operators. This is because a time-consuming local search procedure is iterated for each solution generated by the genetic operators until a local optimum solution is found. In order to prevent

the local search from spending almost all the available computation time, we employ the modified local search procedure described in Subsection 3.6.1. In conventional local search procedures, the local search is terminated when a better solution is not found in the neighbor of the current solution. On the other hand, in the modified local search procedure, the local search is terminated when a better solution is not found in a pre-specified number (say, k) of randomly selected neighborhood solutions. That is, if there is no better solution among randomly selected k neighborhood solutions, the local search is terminated. When we assign a very small value to k (*e.g.*, $k = 2$), the local search may be terminated soon. Thus the local search does not spend long computation time and the generation update by the genetic operators can be iterated many times. On the contrary, when we assign a large value to k (*e.g.*, $k = 100$), almost all the computation time may be spent by the local search, and only a few populations can be generated by the genetic operators. In this manner, we can adjust the computation time spent by the local search.

By incorporating the modified local search algorithm into the MOGA, we construct the multi-objective genetic local search algorithm (MOGLS) as follows:

Step 0 (Initialization): Randomly generate an initial population of N_{pop} solutions.

Step 1 (Evaluation): Calculate the values of the n objectives for each solution in the current population. Then update the tentative set of non-dominated solutions.

Step 2 (Selection): Repeat the following procedures to select a certain number of pairs of parent solutions.

(i) Randomly specify the weight values $w_1, w_2, \ldots, w_n$ in the fitness function (2.5) by (2.8).

(ii) Select a pair of parent solutions according to the following selection probability based on the linear scaling [23]:

$$P_{\text{s}}(\boldsymbol{x}) = \frac{f(\boldsymbol{x}) - f_{\min}(\Psi_t)}{\sum_{\boldsymbol{x}' \in \Psi_t} \{f(\boldsymbol{x}') - f_{\min}(\Psi_t)\}}, \tag{4.4}$$

where $f_{\min}(\Psi_t)$ is the minimum fitness value (*i.e.*, the worst fitness value) in the current population Ψ_t.

Step 3 (Crossover and mutation): Apply a crossover operation to the selected pairs of parent solutions. A new solution is generated from each pair of parent solutions. Then apply a mutation operation to the generated solutions.

Step 4 (Elitist strategy): Randomly remove N_{elite} strings from the generated N_{pop} strings, and add N_{elite} strings that are randomly selected from the tentative set of non-dominated solutions to the current population.

Step 5 (Local search): Apply the modified local search procedure in Subsection 3.6.1 to each of the N_{pop} solutions in the current population. The search direction of the local search for each solution is specified by the weight values in the fitness function by which its parent solutions were selected. The current population is replaced with the N_{pop} solutions improved by the local search.

Step 6 (Termination test): If a pre-specified stopping condition is satisfied, stop the algorithm. Otherwise return to Step 1.

Update of the current population and the tentative set of non-dominated solutions is illustrated in Fig. 4.5.

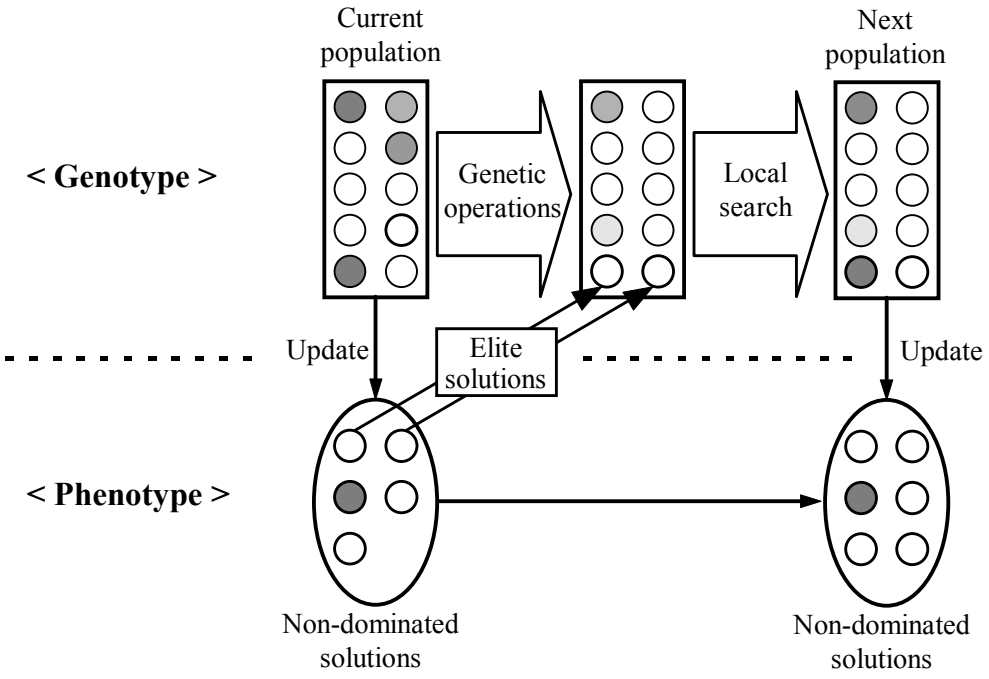


Fig. 4.5 Update of the two sets of solutions stored in the MOGLS.

4.3.2 Two-objective flowshop scheduling problems

In this section, we apply the MOGLS to a randomly generated 20-job and 10-machine flowshop scheduling problem with two objectives: to minimize the makespan and to minimize the maximum tardiness. Therefore we employ the following fitness function in the MOGLS:

$$f(\boldsymbol{x}) = -w_1 f_1(\boldsymbol{x}) - w_3 f_3(\boldsymbol{x}), \tag{4.5}$$

where $f_1(\boldsymbol{x})$ and $f_3(\boldsymbol{x})$ are the objective functions described in (3.5) and (3.7), respectively, and w_1 and w_3 are non-negative weights which satisfy the relations in (2.6) and (2.7). We applied the following four methods to the problem to compare their performance:

(i) The MOGLS with $k = 2$ and $N_{\text{elite}} = 3$.

(ii) The VEGA.

(iii) The CWGA with $w_1 = w_{\text{makespan}} = 5$ and $w_3 = w_{\text{tardiness}} = 2$.

(iv) A random sampling method. A large number of feasible schedules are randomly generated and each schedule is evaluated.

The first three methods were applied to this test problem with the same parameter specifications:

Populations size: $N_{\text{pop}} = 20$,

Crossover probability: 0.9,

Mutation probability: 0.3,

Stopping condition: Evaluation of 100,000 solutions.

In the random sampling method, we examined 2,000,000 feasible solutions, which are twenty times as many as in the other methods. Non-dominated solutions obtained by each method are shown in Fig. 4.6 and Fig. 4.7. From Fig. 4.6 and 4.7, we can see the following:

(1) Some solutions obtained by the VEGA have very small values of the makespan, and others have very small values of the maximum tardiness (see, Fig. 4.7). But no solutions obtained by the VEGA have very small values of both objectives if compared with non-dominated

solutions obtained by the hybrid algorithm and the CWGA (see Fig. 4.6).

(2) The variety of solutions obtained by the CWGA is not large (see Fig. 4.7).

(3) The quality of solutions obtained by the random sampling method is very poor while it examined much more solutions than the other three algorithms (see Fig. 4.6).

In order to clarify these observations, all the solutions obtained by the four algorithms were compared with each other and only non-dominated solutions among all the obtained solutions were selected. Some solutions obtained by one algorithm were dominated by other solutions obtained by the other algorithms. The number of the non-dominated solutions is shown in Table

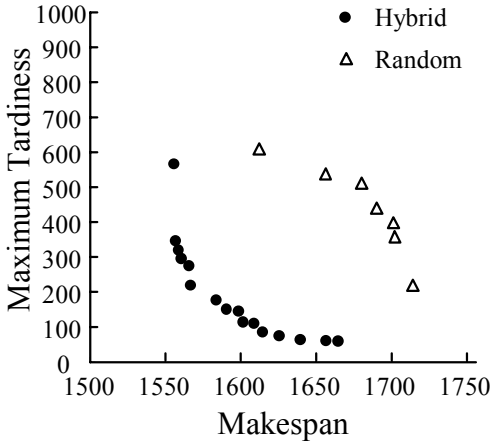


Fig. 4.6 Solutions obtained by the MOGLS and the random sampling method.

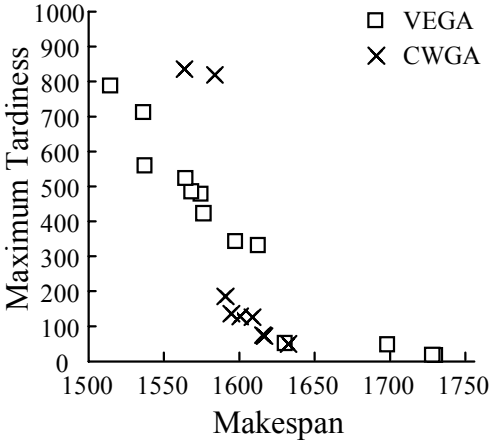


Fig. 4.7 Solutions obtained by the VEGA and the CWGA.

Table 4.1 Simulation results of a single trial of each algorithm for the two-objective flowshop scheduling problem.

Algorithm	The number of obtained solutions (A)	The number of non-dominated solutions (B)	Ratio: B/A
MOGLS	16	11	69%
VEGA	13	7	54%
CWGA	9	4	44%
Random	9	0	0%

Table 4.2 Average results over 20 trials of each algorithm for the two-objective flowshop scheduling problem.

Algorithm	The number of obtained solutions (A)	The number of non-dominated solutions (B)	Ratio: B/A
MOGLS	18.60	15.50	82.5%
VEGA	15.35	6.75	44.0%
CWGA	11.65	2.75	23.1%
Random	10.65	0.00	0.0%

4.1. From Table 4.1, we can see the high performance of the MOGLS because many solutions (*i.e.*, eleven solutions: 69% of the obtained solutions) are not dominated by any other solutions.

Because all the four algorithms are probabilistic search methods, their performance can not be evaluated by a single trial. Thus we applied each algorithm to the two-objective flowshop scheduling problem 20 times. In each trial, obtained solutions by the four algorithms were compared in the same manner as in Table 4.1. The average performance of each algorithm over the 20 trials is shown in Table 4.2. From Table 4.2, we can also see the high performance of the MOGLS.

The average CPU time of each algorithm is shown in Table 4.3. From Table 4.3, we can see that the average CPU time of the three GAs (*i.e.*, the MOGLS, the VEGA, and the CWGA) were almost the same. This is because these three algorithms used the same stopping condition (*i.e.*, evaluation of 100,000 solutions).

Table 4.3 Average CPU time of each algorithm for the two-objective flowshop scheduling problem.

MOGLS	VEGA	CWGA	Random
26.58(sec.)	26.23(sec.)	29.07(sec.)	82.7(sec.)

4.3.3 Three-objective flowshop scheduling problem

In this section, we apply the MOGLS to a randomly generated 20-job and 10-machine flowshop scheduling problem with three-objectives: to minimize the makespan, to minimize the total flowtime, and to minimize the maximum tardiness. Therefore we employ the following fitness function in the MOGLS:

$$f(\boldsymbol{x}) = -w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}) - w_3 f_3(\boldsymbol{x}), \tag{4.6}$$

where $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$, and $f_3(\boldsymbol{x})$ are the objective functions described in (3.5), (3.6), and (3.7), respectively, and w_1, w_2, and w_3 are non-negative weights which satisfy the relations in (2.6) and (2.7).

In the same manner as in Table 4.2, we applied the four algorithms to the three-objective flowshop scheduling problem 20 times. Average results of the twenty trials of each algorithm are shown in Table 4.4. From Table 4.4, we can see the high performance of the MOGLS because many solutions (*i.e.*, 92.8% of the obtained solutions by the hybrid algorithm) are not dominated by any other solutions.

Because it is not easy to compare non-dominated solutions by depicting them in the three-dimensional objective space, Lee *et al*.[66] defined four characteristic features of obtained solution sets as follows:

(i) the center of gravity of the final solution set is close to the ideal point,
(ii) the diversity of the non-dominated solutions is maximized,
(iii) the number of the non-dominated solutions is maximized,
(iv) the bounding volume of the set of the non-dominated solutions is maximized.

Esbensen [11] proposed a method to measure the quality of a set of non-dominated solutions. Let us denote a set of non-dominated solutions by Ω. Then the best solution $\boldsymbol{x}^*$ for a given weight vector $\boldsymbol{w} = (w_1, w_2, w_3)$ can be chosen from Ω as follows:

Table 4.4 Average results over 20 trials of each algorithm for the three-objective flowshop scheduling problems.

Algorithm	The number of obtained solutions (A)	The number of non-dominated solutions (B)	Ratio: B/A
MOGLS	93.75	86.85	92.8%
VEGA	59.45	27.10	45.7%
CWGA	38.30	8.10	23.5%
Random	29.70	0.00	0.0%

Table 4.5 Average quality of the solution set obtained by each algorithm.

MOGLS	VEGA	CWGA	Random
-9736.41	-9907.90	-9837.65	-10489.22

$$\begin{aligned} f(\boldsymbol{x}^*) &= -w_1 f_1(\boldsymbol{x}^*) - w_2 f_2(\boldsymbol{x}^*) - w_3 f_3(\boldsymbol{x}^*) \\ &= \max\{-w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}) - w_3 f_3(\boldsymbol{x}) \mid \boldsymbol{x} \in \Omega\} . \end{aligned} \tag{4.7}$$

Esbensen [11] proposed an idea of measuring the quality of a set of solutions by calculating the expected value of $f(\boldsymbol{x}^*)$ over possible weight vectors $\boldsymbol{w} = (w_1, w_2, w_3)$. In this section, we calculate the expected value of $f(\boldsymbol{x}^*)$ by randomly generating 10,000 weight vectors (say, $\boldsymbol{w}^1$, $\boldsymbol{w}^2$, ..., $\boldsymbol{w}^{10000}$) by (2.8). That is, the quality of the set of non-dominated solutions Ω is calculated in this section as follows:

$$q(\Omega) = \frac{1}{10000} \sum_{i=1}^{10000} \max\{-w_1^i f_1(\boldsymbol{x}) - w_2^i f_2(\boldsymbol{x}) - w_3^i f_3(\boldsymbol{x}) \mid \boldsymbol{x} \in \Omega\} , \tag{4.8}$$

where $q(\Omega)$ is the quality of the solution set Ω and $\boldsymbol{w}^i = (w_1^i, w_2^i, w_3^i)$, $i = 1,2,...,10000$.

For the set of non-dominated solutions obtained by each trial of each algorithm shown in Table 4.4, we calculated the quality of the solution sets by (4.8). We iterated this calculation 20 times for each algorithm to evaluate the average quality of the solution set obtained by each algorithm. Simulation results are summarized in Table 4.5. From Table 4.5, we can see that the best result (*i.e.*, the maximum average quality) was obtained by the MOGLS.

4.4 SUMMARY

This chapter dealt with the application of GAs to multi-objective flowshop scheduling problems. In the first section, we applied the MOGA to flowshop scheduling problems with two objectives and three objectives. By two-objective flowshop scheduling problems, we compared the MOGA with the SOGA. In the SOGA, one of two objectives was used for the fitness function. Next, we examined the relation between the number of elite solutions to be inherited and the performance of the MOGA. Then we compared the MOGA with the VEGA and the CWGA. In the CWGA, two objectives are combined into a single scalar fitness function using constant weights. Last in the first section, we applied the MOGA to a three-objective flowshop scheduling problem.

In the second section, we also hybridized the MOGA with a local search algorithm in the same manner as in Chapter 3. In this section, we described the multi-objective genetic local search algorithm (MOGLS). The MOGLS is an extension of the MOGA in [77] to a hybrid algorithm. In the MOGLS, a local search procedure is applied to each solution generated by the genetic operators. By computer simulations on flowshop scheduling problems, high performance of the MOGLS was demonstrated.

The characteristic features of the MOGLS can be summarized as follows:

(1) A weighted sum of multiple objectives is used as a fitness function in a selection of a pair of parent solutions. The weight values in the fitness function are randomly specified whenever a pair of parent solutions is selected.

(2) A local search procedure is applied to each new solution generated by the genetic operators (*i.e.*, crossover, and mutation). The local search for each new solution is performed to maximize the fitness function which was used for selecting its parent solutions. Thus each new solution has its own local search direction in the objective space.

(3) In the local search, all the neighborhood solutions of the current solution are not examined for each move. That is, the number of examined neighborhood solutions of the current solution is restricted in the local search. This is to prevent the local search from spending almost all the available computation time.

(4) A tentative set of non-dominated solutions is stored and updated at every generation. The tentative set is stored separately from a current population. A few solutions randomly selected from the tentative set are used as a kind of elite solutions.