A Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling

Hisao Ishibuchi, *Member, IEEE*, and Tadahiko Murata, *Member, IEEE*

***Abstract*— In this paper, we propose a hybrid algorithm for finding a set of nondominated solutions of a multi-objective optimization problem. In the proposed algorithm, a local search procedure is applied to each solution (i.e., each individual) generated by genetic operations. Our algorithm uses a weighted sum of multiple objectives as a fitness function. The fitness function is utilized when a pair of parent solutions are selected for generating a new solution by crossover and mutation operations. A local search procedure is applied to the new solution to maximize its fitness value. One characteristic feature of our algorithm is to randomly specify weight values whenever a pair of parent solutions are selected. That is, each selection (i.e., the selection of two parent solutions) is performed by a different weight vector. Another characteristic feature of our algorithm is not to examine all neighborhood solutions of a current solution in the local search procedure. Only a small number of neighborhood solutions are examined to prevent the local search procedure from spending almost all available computation time in our algorithm. High performance of our algorithm is demonstrated by applying it to multi-objective flowshop scheduling problems.**

***Index Terms*—Genetic algorithms (GA's), local search, multi-objective optimization, scheduling, search direction.**

I. Introduction

GENETIC algorithms (GA's) [1] have been successfully applied to various optimization problems (see, for example, Goldberg [2] and Davis [3]). The extension of GA's to multi-objective optimization was proposed in several manners (for example, see Schaffer [4], Kursawe [5], Horn *et al.* [6], Fonseca and Fleming [7], [8], Murata and Ishibuchi [9], and Tamaki *et al.* [10]). In this paper, we propose a multi-objective genetic local search algorithm, which is a hybrid algorithm of a multi-objective GA [9] and a modified local search procedure. Many hybrid algorithms [11]–[15] of GA's and neighborhood search algorithms (e.g., local search, simulated annealing, and tabu search) were proposed for single-objective optimization problems to improve the search ability of GA's, and their high performance was reported in the literature. In those studies, it was clearly shown that the performance of GA's for traveling salesman problems and scheduling problems was significantly improved by combining neighborhood search algorithms. While we can expect significant improvement of the performance of multi-objective GA's by such hybridization, multi-objective hybrid GA's have not been proposed. When we try to implement multi-objective hybrid GA's, one difficulty lies in determining appropriate search directions for neighborhood search algorithms. This paper proposes a multi-objective genetic local search algorithm in which a simple but efficient idea for coping with this difficulty is employed.

Manuscript received July 5, 1996; revised October 22, 1997.

H. Ishibuchi is with the Department of Industrial Engineering, Osaka Prefecture University, Sakai, Osaka, 599-8531 Japan.

T. Murata is with the Department of Industrial and Systems Engineering, Ashikaga Institute of Technology, Ashikaga, Tochigi, 326-8558 Japan (e-mail: murata@ashitech.ac.jp).

Publisher Item Identifier S 1094-6977(98)03906-6.

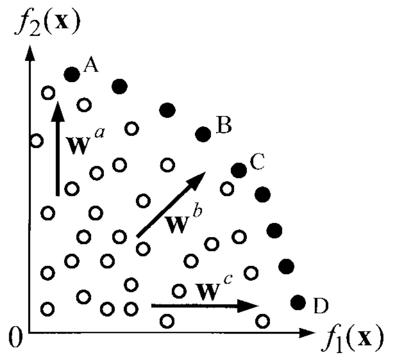

Fig. 1. Nondominated solutions (closed circles) and dominated solutions (open circles).

Our multi-objective genetic local search algorithm tries to find all nondominated solutions of an optimization problem with multiple objectives. Let us consider the following multi-objective optimization problem with n objectives:

$$\text{Maximize} \quad f_1(\boldsymbol{x}),\ f_2(\boldsymbol{x}),\ \cdots,\ f_n(\boldsymbol{x}) \tag{1}$$

where $f_1(\cdot), f_2(\cdot), \cdots, f_n(\cdot)$ are n objectives to be maximized. When the following inequalities hold between two solutions $\boldsymbol{x}$ and $\boldsymbol{y}$, the solution $\boldsymbol{y}$ is said to dominate the solution $\boldsymbol{x}$:

$$\forall i{:}\ f_i(\boldsymbol{x}) \le f_i(\boldsymbol{y}) \quad \text{and} \quad \exists j{:}\ f_j(\boldsymbol{x}) < f_j(\boldsymbol{y}). \tag{2}$$

If a solution is not dominated by any other solutions of the multi-objective optimization problem, that solution is said to be a nondominated solution. Examples of nondominated solutions are shown in Fig. 1, where dominated solutions and nondominated solutions are depicted by open circles and closed circles in a two-dimensional (2-D) objective space, respectively. The 2-D objective space in Fig. 1 corresponds to the following two-objective optimization problem:

$$\text{Maximize} \quad f_1(\boldsymbol{x}) \quad \text{and} \quad f_2(\boldsymbol{x}). \tag{3}$$

As is shown in Fig. 1, multi-objective optimization problems usually have several nondominated solutions.

The aim of our hybrid algorithm is not to determine a single final solution but to try to find all nondominated solutions of the multi-objective optimization problem in (1). When we apply GA's to the multi-objective optimization problem, we have to evaluate a fitness value of each solution. We define a fitness function of the solution $\boldsymbol{x}$ by the following weighted sum of the n objectives:

$$f(\boldsymbol{x}) = w_1 f_1(\boldsymbol{x}) + w_2 f_2(\boldsymbol{x}) + \cdots + w_n f_n(\boldsymbol{x}) \tag{4}$$

where $w_1, \cdots, w_n$ are nonnegative weights for the n objectives, which satisfy the following relations:

$$w_i \geq 0 \quad \text{for} \quad i = 1, 2, \cdots, n \tag{5}$$

$$w_1 + w_2 + \cdots + w_n = 1. \tag{6}$$

If we use constant weight values, the search direction by GA's is fixed. For example, the search direction $\mathbf{w}^b$ in Fig. 1 corresponds to the weight vector $\mathbf{w}^b = (w_1, w_2) = (0.5, 0.5)$ in the 2-D objective space. When the search direction is fixed, it is not easy to obtain a variety of nondominated solutions. In Fig. 1, GA's with the constant weight vector $\mathbf{w}^b = (w_1, w_2) = (0.5, 0.5)$ may easily find the solutions B and C, but it is very difficult to find the solutions A and D.

An alternative approach is to choose one of the n objectives as a fitness function of each solution. For example, Schaffer [4] divided a population (i.e., a set of solutions) into n subpopulations, each of which was governed by one of the n objectives. Kursawe [5] suggested an idea to choose one of the n objectives according to the user definable probability assigned to each objective. Thus, GA's had n search directions in Schaffer [4] and Kursawe [5]. We show the search directions $\mathbf{w}^a$ and $\mathbf{w}^c$ of these approaches in Fig. 1 for the case of the two-objective optimization problem in (3). As we can expect from Fig. 1, these approaches with the search directions $\mathbf{w}^a$ and $\mathbf{w}^c$ can easily find the solutions A and D, but it is not easy to find the solutions B and C.

From the above discussions, we can see that neither the constant weight value approach nor the choice of one objective is appropriate for finding all of the nondominated solutions of the multi-objective optimization problem in (1). This is because various search directions are required to find a variety of nondominated solutions. In order to realize various search directions, we suggested an idea of randomly specified weight values in our former work [9]. The weight values were determined as

$$\begin{aligned} w_i &= random_i/(random_1 + \cdots + random_n) \\ i &= 1, 2, \cdots, n \end{aligned} \tag{7}$$

where $random_1, random_2, \cdots, random_n$ are nonnegative random real numbers (or nonnegative random integers). It should be noted that the weight values are specified by (7) whenever a pair of parent solutions are selected for generating a new solution by a crossover operation. For example, when N pairs of parent solutions are selected for generate a new population, N different weight vectors are specified by (7). This means that N search directions are utilized in a single generation of GA's. In other words, each selection (i.e., the selection of two parent solutions) is governed by its own fitness function.

In the multi-objective genetic local search algorithm in this paper, we use the same idea as in our former work [9]. That is, we specify the weight values by (7) 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 (4). In our hybrid algorithm, the local search is applied to each new solution generated by the genetic operations (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 in (1).

Another issue to be addressed in the hybrid algorithm is how to divide the available computation time between the local search and the genetic operations. If we simply combine the local search with the genetic operations, almost all available computation time may be spent by the local search and only a few populations are generated by the genetic operations. This is because a time-consuming local search procedure is iterated for each solution generated by the genetic operations until a locally optimum solution is found. In order to prevent the local search from spending almost all available computation time, we propose an idea to restrict the number of solutions examined for each move in the local search. In conventional local search procedures, the local search is terminated when a better solution than the current one is not found by examining all neighborhood solutions. On the other hand, in our local search procedure, the local search is terminated when a better solution is not found by examining a prespecified 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 a long computation time and the generation update by the genetic operations can be iterated many times. On the contrary, when we assign a large value to k (e.g., $k = 100$), almost all computation time may be spent by the local search and only a few populations can be generated by the genetic operations. In this manner, we can adjust the computation time spent by the local search.

The proposed hybrid algorithm is applied to multi-objective flowshop scheduling problems. Flowshop scheduling is one of the most well-known scheduling problems. Since Johnson's work [16], various scheduling criteria have been considered (see, for example, reviews by Baker and Scudder [17] and Dudek *et al.* [18]). Among them are makespan, maximum tardiness, total tardiness, maximum flowtime, and total flowtime. Several researchers extended single-objective flowshop scheduling problems to multi-objective problems. For example, Daniels and Chambers [19] considered the tradeoff between the makespan and the maximum tardiness. Rajendran [20] 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.* [21]

proposed a modified random sampling method for obtaining a set of nondominated 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.* [22], where the makespan, the maximum tardiness, and the total flowtime were used as scheduling criteria. In this paper, we apply our hybrid algorithm to the two-objective and three-objective flowshop scheduling problems in Morizawa *et al.* [21], [22]. By computer simulations on randomly generated test problems, we compare our hybrid algorithm with other multi-objective genetic algorithms and a random sampling technique. While various approaches have been proposed for multi-objective flowshop scheduling problems, they are special-purpose algorithms. That is, each algorithm is only applicable to a special flowshop scheduling problem because it was tailored by utilizing domain knowledge. For example, some algorithms are only applicable to two-machine problems. Other algorithms can handle only the makespan and the total tardiness as objective functions. One advantage of our hybrid algorithm over those approaches is its generality. That is, it is a general-purpose algorithm applicable to any multi-objective flowshop scheduling problems with many machines and many objectives. Actually, our hybrid algorithm is applicable to not only flowshop scheduling problems, but also to any other multi-objective optimization problems by adjusting genetic operations and a local search procedure.

The organization of this paper is as follows. Section II explains each step of the proposed hybrid algorithm. Section III illustrates the proposed algorithm by small-size multi-objective optimization problems. Section III also demonstrates that the proposed algorithm can find nondominated solutions of a multi-objective optimization problem with a nonconvex feasible region in the objective space. Section IV compares the proposed algorithm with other multi-objective GA's by applying them to multi-objective flowshop scheduling problems. High performance of the proposed algorithm is demonstrated by various computer simulations. Section V concludes this paper.

II. Proposed Algorithm

In this section, we propose a hybrid algorithm to find all nondominated solutions of the n-objective optimization problem in (1): Maximize $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$, $\cdots$, $f_n(\boldsymbol{x})$.

A. Selection Operation

When a pair of parent solutions are to be selected from a current population Ψ for generating an offspring by a crossover operation, first the n weight values $(w_1, w_2, \cdots, w_n)$ are randomly specified by (7) and then a fitness value of each solution $\boldsymbol{x}$ in the current population Ψ is calculated as the weighted sum of the n objectives by (4). The selection probability $P(\boldsymbol{x})$ of each solution $\boldsymbol{x}$ is defined by the roulette wheel selection using the linear scaling (see Goldberg [2]) as

$$P(\boldsymbol{x}) = \frac{f(\boldsymbol{x}) - f_{\min}(\Psi)}{\sum_{\boldsymbol{x}\in\Psi} \{f(\boldsymbol{x}) - f_{\min}(\Psi)\}} \tag{8}$$

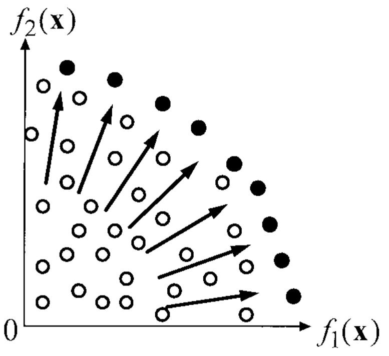

Fig. 2. Various search directions of our hybrid algorithm.

where $f_{\min}(\Psi)$ is the fitness value of the worst solution in the current population Ψ. That is, $f_{\min}(\Psi) = \min\{f(\boldsymbol{x}) | \boldsymbol{x} \in \Psi\}$. According to this selection probability, a pair of parent solutions are selected from the current population Ψ.

An offspring (i.e., a new solution) is generated by a crossover operation from the selected pair of parent solutions. Then a mutation operation is applied to the new solution. A local search procedure is applied to the new solution after the mutation. The local search tries to maximize the fitness value [i.e., the weighted sum of the n objectives defined by (4)] of the new solution. This means that the direction of the local search of the new solution is defined by the weight values used in the selection of its parent solutions.

When another pair of parent solutions are selected, we randomly specify the n weight values $(w_1, w_2, \cdots, w_n)$ again. That is, we use a different weight vector for the selection of each pair of parent solutions. Because the local search for a new solution uses the same weight values as in the selection of its parent solutions, each new solution has its own local search direction. Thus, the selection and the local search in our hybrid algorithm have various search directions, as shown in Fig. 2.

B. Local Search Procedure

As is explained in the above, a local search procedure is applied to each new solution generated by the genetic operations (i.e., selection, crossover, and mutation) to maximize its fitness value $f(\boldsymbol{x})$ in (4). The local search is also applied to elite solutions inherited from previous populations.

Generally, a local search procedure can be written as follows.

Local Search Procedure:

Step 0) Specify an initial solution $\boldsymbol{x}$.

Step 1) Examine a neighborhood solution $\boldsymbol{y}$ of the current solution $\boldsymbol{x}$.

Step 2) If $\boldsymbol{y}$ is a better solution than $\boldsymbol{x}$ [i.e., $f(\boldsymbol{x}) < f(\boldsymbol{y})$], replace the current solution $\boldsymbol{x}$ with $\boldsymbol{y}$ (i.e., let $\boldsymbol{x}\colon = \boldsymbol{y}$) and return to Step 1).

Step 3) If all of the neighborhood solutions of the current solution $\boldsymbol{x}$ have been already examined (i.e., if there is no neighborhood solution that improves $\boldsymbol{x}$), end this procedure. Otherwise, return to Step 1) [i.e., another neighborhood solution is examined in Step 1)].

As we can see from Step 3), this local search procedure is terminated when there is no better solution in the neighborhood of the current solution $\boldsymbol{x}$. This means that all of the neighborhood solutions of the current solution $\boldsymbol{x}$ should be examined before the procedure is terminated. Therefore, the total number of solutions examined by this local search procedure for a single initial solution is more than or equal to the number of neighborhood solutions. For example, if we define the neighborhood solutions by exchanging arbitrarily two jobs for a flowshop scheduling problem with 20 jobs, the number of the neighborhood solutions is ${}_{20}C_2 = 190$. This means that at least 190 solutions are examined before the local search procedure is terminated for a single initial solution. Therefore, almost all available computation time is spent by the local search procedure if we apply this local search procedure to each new solution generated by the genetic operations in our hybrid algorithm.

If we want to efficiently utilize the global search ability of GA's in our hybrid algorithm, we have to reduce the computation time spent by the local search. This can be realized by restricting the number of neighborhood solutions examined by the local search procedure. In our hybrid algorithm, we use the following modified local search procedure.

Modified Local Search Procedure:

Step 0) Specify an initial solution $\boldsymbol{x}$.

Step 1) Examine a neighborhood solution $\boldsymbol{y}$ of the current solution $\boldsymbol{x}$.

Step 2) If $\boldsymbol{y}$ is a better solution than $\boldsymbol{x}$, replace the current solution $\boldsymbol{x}$ with $\boldsymbol{y}$ and return to Step 1).

Step 3) If randomly chosen k neighborhood solutions of the current solution $\boldsymbol{x}$ have been already examined (i.e., if there is no better solution among the examined k neighborhood solutions of $\boldsymbol{x}$), end this procedure. Otherwise, return to Step 1).

This algorithm is terminated if no better solution is found among k neighborhood solutions that are randomly selected from the neighborhood of the current solution. Therefore, if we use a very small value of k (e.g., $k = 2$), the local search procedure may be terminated soon. On the contrary, if we use a large value of k (e.g., $k = 100$), the local search procedure examines many solutions. In this manner, we can adjust the computation time spent by the local search procedure in our hybrid algorithm.

C. Elitist Strategy

Our hybrid algorithm stores two sets of solutions: a current population and a tentative set of nondominated solutions. After the local search, the current population is replaced with the improved population by the local search (i.e., the current population is improved by the local search) and then the tentative set of nondominated solutions is updated by the new current population. That is, if a solution in the current population is not dominated by any other solutions in the current population and the tentative set of nondominated solutions, this solution is added to the tentative set. Then all solutions dominated by the added one are eliminated from the tentative set. In this manner, the tentative set of nondominated solutions is updated at every generation in our hybrid algorithm.

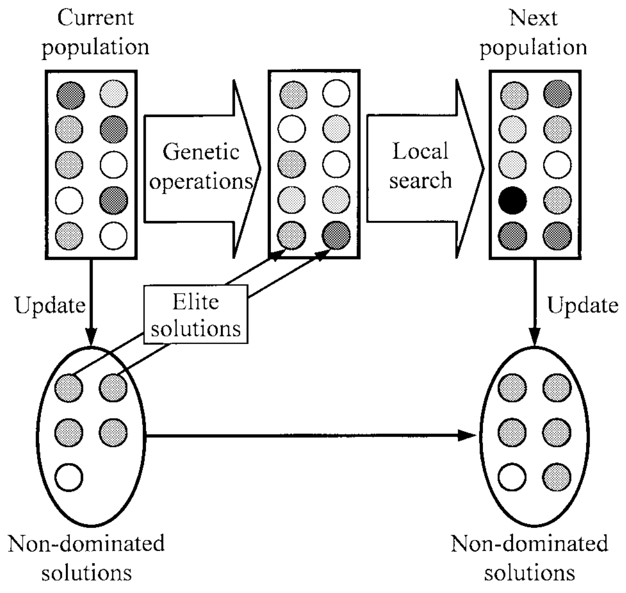

Fig. 3. Update of the two sets of solutions stored in our hybrid algorithm.

From the tentative set of nondominated solutions, a few solutions are randomly selected as initial solutions of the local search. That is, the local search is applied to the selected nondominated solutions as well as new solutions generated by the genetic operations. The direction of the local search for each nondominated solution is determined by the fitness function (i.e., the weighted sum of the n objectives) used in the selection of its parent solutions. If a nondominated solution does not have parent solutions (i.e., if a nondominated solution is a randomly generated initial solution), random weight values are assigned to that nondominated solution to perform the local search. The randomly selected nondominated solutions may be viewed as kinds of elite solutions because they are added to the current population with no genetic operations. Update of the current population and the tentative set of nondominated solutions are illustrated in Fig. 3.

D. Multi-Objective Genetic Local Search Algorithm

Let us denote the population size by N_{pop}. We also denote the number of nondominated solutions added to the current population by N_{elite} (i.e., N_{elite} is the number of elite solutions, see Fig. 3). Our hybrid algorithm can be written 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, and then update the tentative set of nondominated solutions.

Step 2) Selection: Repeat the following procedures to select $(N_{\text{pop}} - N_{\text{elite}})$ pairs of parent solutions.

a) Randomly specify the weight values $w_1, w_2, \cdots, w_n$ in the fitness function (4) by (7).

b) According to the selection probability in (8), select a pair of parent solutions.

TABLE I
PROCESSING TIME OF EACH JOB ON EACH MACHINE IN THE TEN-JOB AND FIVE-MACHINE TEST PROBLEM

	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10
Machine 1	32	1	61	42	62	61	3	97	26	9
Machine 2	21	27	87	45	59	24	71	34	20	28
Machine 3	10	42	66	75	41	24	3	36	85	74
Machine 4	51	19	23	85	86	81	93	31	75	23
Machine 5	33	45	58	97	91	85	30	38	17	51

Step 3) Crossover and Mutation: Apply a crossover operator to each of the selected ($N_{\rm pop}$–$N_{\rm elite}$) pairs of parent solutions. A new solution is generated from each pair of parent solutions. Then apply a mutation operator to the generated new solutions.

Step 4) Elitist Strategy: Randomly select $N_{\rm elite}$ solutions from the tentative set of nondominated solutions, and then add the selected $N_{\rm elite}$ solutions to the ($N_{\rm pop}$–$N_{\rm elite}$) solutions generated in Step 3) to construct a population of $N_{\rm pop}$ solutions.

Step 5) Local Search: Apply the modified local search procedure in Section II-B to all $N_{\rm pop}$ 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 $N_{\rm pop}$ solutions improved by the local search.

Step 6) Termination Test: If a prespecified stopping condition is satisfied, end the algorithm. Otherwise, return to Step 1).

III. ILLUSTRATION OF THE PROPOSED ALGORITHM

In this section, we illustrate the proposed hybrid algorithm by applying it to a small size flowshop scheduling problem and a simple test problem with a nonconvex feasible region.

A. Application to a Flowshop Scheduling Problem

We generated a ten-job and five-machine flowshop scheduling problem as a small-size test problem for illustrating our hybrid algorithm. The processing time of each job on each machine was specified as a random integer in the interval [1, 99]. The due date of each job was specified as follows.

1) Randomly generate a sequence of the ten jobs.
2) Calculate the completion time of each job when the given jobs are processed in the sequence specified in 1).
3) Specify the due date of each job by

$$d_j = c_j + \text{random}[-100, 100] \tag{9}$$

where d_j is the due date of job j, c_j is the completion time of job j, and random[-100, 100] is a random integer in the interval [-100, 100].

We show the processing time and the due date of each job specified in this manner in Tables I and II, respectively.

For the test problem in Tables I and II, we use two scheduling criteria: the makespan and the maximum tardiness, which

TABLE II
DUE DATE OF EACH JOB IN THE TEN-JOB AND FIVE-MACHINE TEST PROBLEM

Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10
674	396	431	369	626	597	790	437	656	780

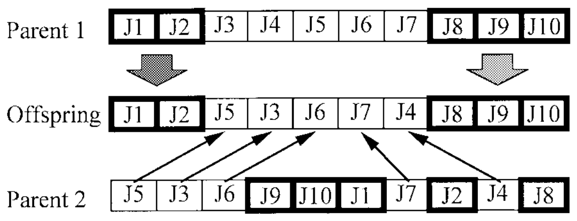

Fig. 4. Two-point crossover.

were considered in Morizawa *et al.* [21]. Because the variance of the makespan is much smaller than that of the maximum tardiness, we normalize these two scheduling criteria and specify the following two objectives:

$$f_1(\boldsymbol{x}) = 5 \times (\text{makespan}) \tag{10}$$

$$f_2(\boldsymbol{x}) = 2 \times (\text{maximum tardiness}) \tag{11}$$

where $\boldsymbol{x}$ is a feasible solution of the flowshop scheduling (i.e., $\boldsymbol{x}$ is a permutation of the given jobs). Constant multipliers in (10) and (11) were introduced to handle the two scheduling criteria equally. Using (10) and (11), our test problem can be written as follows.

Test Problem 1:

$$\text{Minimize} \quad f_1(\boldsymbol{x}) \quad \text{and} \quad f_2(\boldsymbol{x}).$$

Because these two objectives should be minimized, we define the fitness function as

$$f(\boldsymbol{x}) = -w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}). \tag{12}$$

This fitness function is used in the selection and the local search of our hybrid algorithm.

As a crossover operator, we used a two-point crossover illustrated in Fig. 4. A single offspring is generated from two parent solutions, as shown in Fig. 4. As a mutation operator, we use a shift mutation illustrated in Fig. 5, where a randomly selected job is removed and inserted into a randomly selected position. High performance of these genetic operators was demonstrated in Murata and Ishibuchi [15] for single-objective flowshop scheduling problems to minimize the makespan. For the local search, we defined neighborhood solutions by the shift mutation. That is, neighborhood solutions of the current solution $\boldsymbol{x}$ are generated by a single application of the shift mutation to $\boldsymbol{x}$. The total number of the neighborhood solutions of $\boldsymbol{x}$ is $(n-1)^2$ for n-job flowshop scheduling problems (i.e., 81 for our ten-job test problem). The neighborhood structure defined by the shift mutation was often used for tabu search algorithms and simulated annealing algorithms (see, for example, Taillard [23], Osman and Potts [24], and Ishibuchi *et al.* [25]).

We applied the proposed hybrid algorithm to Test Problem 1 with the following parameter specifications: population size:

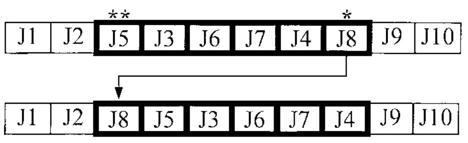

Fig. 5. Shift mutation.

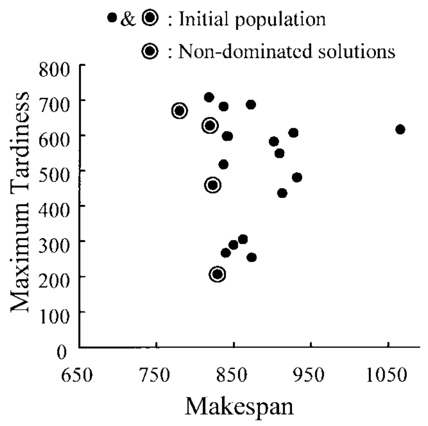

Fig. 6. Initial population and the tentative set of nondominated solutions.

$N_{\rm pop} = 20$, crossover probability: 0.9, mutation probability for each string: 0.3, number of elite solutions: $N_{\rm elite} = 3$, number of neighborhood solutions examined for each move in the local search: $k = 2$, and stopping condition: evaluation of 10 000 solutions. It should be noted that the mutation probability is assigned not to each position of a string but to each string (i.e., to each solution), as in many GAs' applications to permutation problems (e.g., TSP).

Now we illustrate our hybrid algorithm in Section II-D by our test problem. In Step 0), our hybrid algorithm randomly generates 20 initial solutions. Then each solution is evaluated in Step 1). We show each solution in the 2-D objective space in Fig. 6. Our hybrid algorithm also updates the tentative set of nondominated solutions in Step 1). Nondominated solutions in the initial population, which are indicated in Fig. 6, are stored as the initial tentative set of nondominated solutions.

In Step 2), $N_{\rm pop} - N_{\rm elite} = 17$ pairs of parent solutions are selected from the current population by randomly specifying the weight values (w_1, w_2) 17 times. For example, a pair of parent solutions are selected with the weight values $w_1 = 0.503$ and $w_2 = 0.497$. Those parent solutions are shown in Fig. 7. In Step 3), the two-point crossover and the shift mutation are applied to the selected 17 pairs of parent solutions with the prespecified crossover and mutation probabilities. Thus, 17 new solutions are generated. In Step 4), $N_{\rm elite}$ solutions (i.e., three solutions) are randomly selected from the tentative set of nondominated solutions and added to the set of the 17 new solutions to form a population of 20 solutions. In Step 5), the modified local search procedure in Section II-B is applied to each solution with $k = 2$. In Fig. 7, we show a new solution generated by the crossover and the mutation. Because the parent solutions of this new solution in Fig. 7 were selected with the weight values $w_1 = 0.503$ and $w_2 = 0.497$, these weight values are also used in the

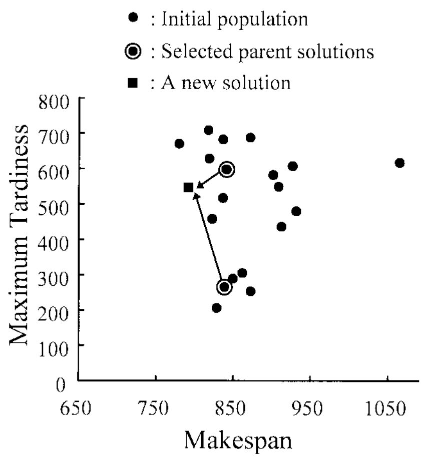

Fig. 7. Pair of selected solutions and their offspring generated by the genetic operations.

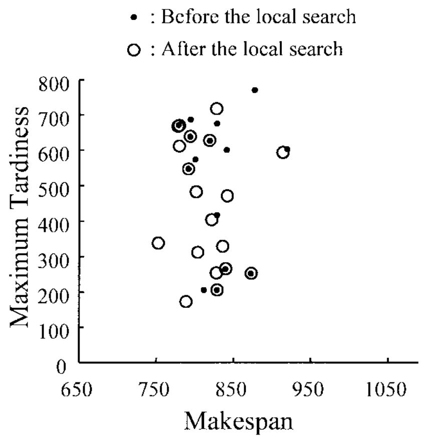

Fig. 8. Population before the local search (i.e., after the genetic operations) and after the local search.

local search in Step 5) for the new solution. That is, the local search is applied to the new solution in Fig. 7 to maximize the following fitness function:

$$\begin{aligned} f(\boldsymbol{x}) &= -w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}) \\ &= -0.503 f_1(\boldsymbol{x}) - 0.497 f_2(\boldsymbol{x}). \end{aligned} \tag{13}$$

In Fig. 8, we show the population of 20 solutions before the local search (i.e., closed circles) and after the local search (i.e., open circle). That is, the 20 solutions denoted by closed circles are used as initial solutions of the local search and the 20 solutions denoted by open circles are obtained by the local search. From the comparison between Figs. 6 and 8, we can see that the quality of the solutions was improved by the genetic operations and the local search.

The above procedures [i.e., steps 1)–5)] were iterated until the prespecified stopping condition was satisfied (i.e., until 10 000 solutions were examined). In our computer simulation, the hybrid algorithm was terminated after 158 iterations of the above procedure. Because each generation consisted of 20 solutions, we can see that $20 \times 158 = 3016$ solutions were examined in the generation update by the genetic operations. The other solutions were examined during the local search.

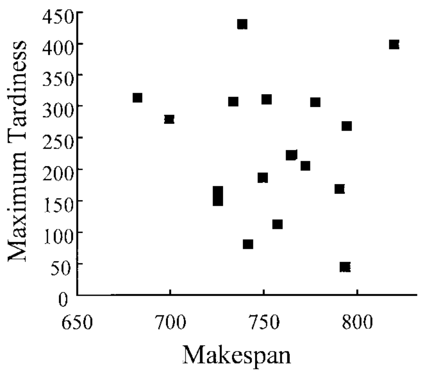

Fig. 9. Final population.

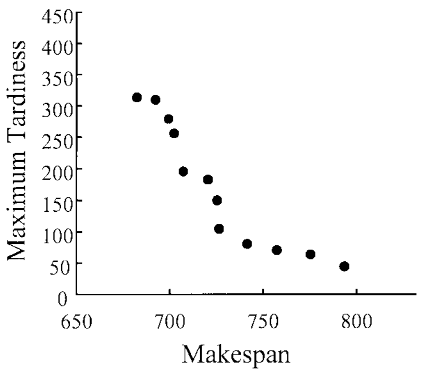

Fig. 10. Final set of nondominated solutions.

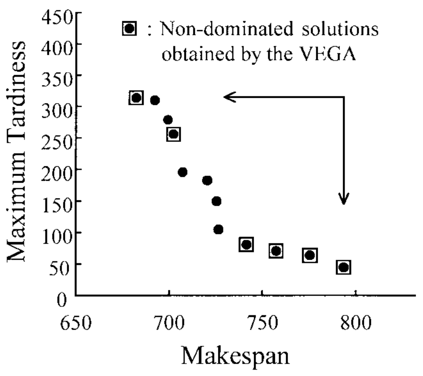

Fig. 11. Search directions of the VEGA and the obtained nondominated solutions.

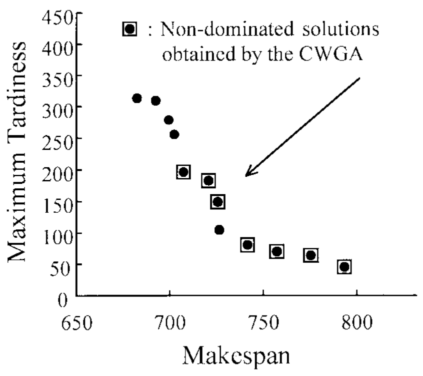

Fig. 12. Search direction of the CWGA with $w_1 = w_2 = 0.5$ and the obtained nondominated solutions.

We show the final population in Fig. 9 and the final set of nondominated solutions in Fig. 10. Because our test problem is small, we can examine all feasible solutions (i.e., all permutations of ten jobs: $10! = 3\,628\,800$ solutions). By such an enumeration method, we confirmed that all nondominated solutions of our test problem were obtained by the proposed hybrid algorithm. That is, 12 solutions in Fig. 10 are all nondominated solutions of our test problem.

In order to demonstrate the effect of the modification of the local search, we also applied the hybrid algorithm with the original local search procedure in Section II-B to our test problem. In the original local search procedure, all 81 neighborhood solutions of a current solution were examined for each move in the local search. The hybrid algorithm was iterated four times while 10 000 solutions were examined. This means that the generation update was iterated only four times. Thus, we can see that almost all of the computation time was spent by the local search. Three solutions out of the 12 nondominated solutions in Fig. 10 were not found by the hybrid algorithm with the original local search. From this result, we can see that the modification of the local search has a good effect on the performance of the hybrid algorithm.

For comparison, we also applied the vector evaluated genetic algorithm (VEGA) by Schaffer [4] and a constant weight genetic algorithm (CWGA). In Fig. 11, we show the search directions of the VEGA and the obtained nondominated solutions. We can see from Fig. 11 that all nondominated solutions were not found by the VEGA. This is because each of its search directions is parallel to one axis of the objective space, as shown in Fig. 11. On the other hand, we show the search direction of the CWGA with $w_1 = w_2 = 0.5$ and the obtained nondominated solutions in Fig. 12. In the execution of the CWGA, we stored the tentative set of nondominated solutions in the same manner as in the proposed hybrid algorithm. From Figs 11 and 12, we can see that neither the VEGA nor the CWGA found all nondominated solutions. These results show the effectiveness of variable weight values in the proposed hybrid algorithm.

B. Application to a Test Problem with a Nonconvex Feasible Region

Weighted-sum approaches usually do not work well for multi-objective optimization problems with nonconvex feasible regions in objective spaces. In this subsection, we demonstrate that the proposed hybrid algorithm can handle such a multi-objective optimization problem. As a test problem, let us consider the following two-objective optimization problem.

Test Problem 2:

$$\text{Minimize} \quad f_1(\boldsymbol{x}) = 2\sqrt{x_1} \quad \text{and} \quad f_2(\boldsymbol{x}) = x_1(1 - x_2) + 5 \tag{14}$$

$$\text{subject to} \quad 1 \le x_1 \le 4 \quad \text{and} \quad 1 \le x_2 \le 2 \tag{15}$$

where $\boldsymbol{x} = (x_1, x_2)$. This test problem was used for examining some multi-objective GA's in Tamaki *et al.* [10]. We show

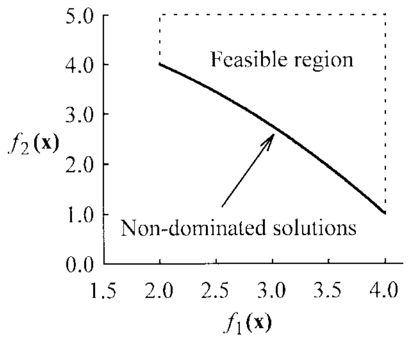

Fig. 13. Feasible region and nondominated solutions of Test Problem 2.

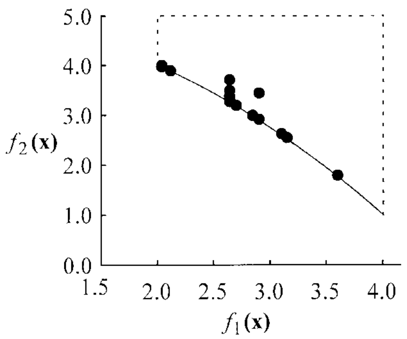

Fig. 14. Final population.

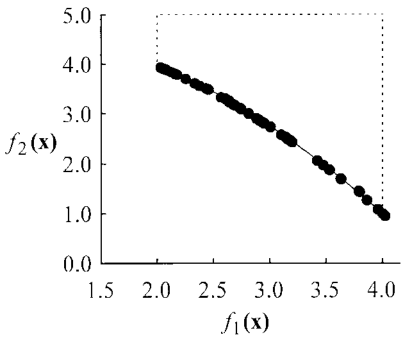

Fig. 15. Final set of nondominated solutions.

the feasible region of this test problem and the nondominated solutions in the 2-D objective space in Fig. 13. From Fig. 13, we can see that this test problem has the nonconvex feasible region.

Because the two objectives of Test Problem 2 should be minimized, we specified the fitness function of each solution $\boldsymbol{x}$ by (12), as in Test Problem 1. In the same manner as in Tamaki *et al.* [10], we denoted each decision variable x_i by a 10-bit string. For example

$$x_1 = 1010101000 \quad \text{and} \quad x_2 = 1111010100. \tag{16}$$

Thus, each solution $\boldsymbol{x} = (x_1, x_2)$ of the test problem was denoted by a 20-bit string. For example

$$\boldsymbol{x} = 10101010001111010100. \tag{17}$$

For 20-bit strings, we used a two-point crossover and the standard bit mutation (i.e., $0 \rightarrow 1$ and $1 \rightarrow 0$) in the same manner as in Tamaki *et al.* [10]. We applied the proposed algorithm to the test problem with the following parameter specifications: population size: $N_{\text{pop}} = 100$; crossover probability: 0.9; mutation probability for each bit: 0.01; number of elite solutions: $N_{\text{elite}} = 5$; number of neighborhood solutions examined for each move in the local search: $k = 0$; and stopping condition: 20 generations. These parameter specifications are similar to those in Tamaki *et al.* [10]. Because this test problem is simple, we did not use the local search in the proposed method. Thus, we specified k as $k = 0$.

We show the final population and the final set of nondominated solutions in Figs. 14 and 15, respectively. From Fig. 15, we can see that nondominated solutions on the nonconvex surface of the feasible region were obtained by the proposed algorithm.

IV. Comparison with Other Approaches

In this section, we examine the performance of the proposed hybrid algorithm by computer simulations on two-objective and three-objective flowshop scheduling problems.

A. Test Problems

In the same manner as in Section III-A, we generated a 20-job and ten-machine flowshop scheduling problem. The size of this problem is much larger than that of Test Problem 1 in Section III-A. The total number of feasible solutions (i.e., all permutations of 20 jobs) is over 10^{18}. Thus, we cannot apply enumeration methods to this problem.

As scheduling criteria, we used the makespan, the maximum tardiness, and the total flowtime as in Morizawa *et al.* [21], [22]. Using these scheduling criteria, we specified the following three objectives:

$$f_1(\boldsymbol{x}) = 5 \times (\text{makespan}) \tag{18}$$
$$f_2(\boldsymbol{x}) = 2 \times (\text{maximum tardiness}) \tag{19}$$
$$f_3(\boldsymbol{x}) = 1 \times (\text{total flowtime}). \tag{20}$$

In computer simulations, we examined the following test problems.

Test Problem 3:

$$\text{Minimize} \quad f_1(\boldsymbol{x}) \quad \text{and} \quad f_2(\boldsymbol{x})$$

Test Problem 4:

$$\text{Minimize} \quad f_1(\boldsymbol{x}), \quad f_2(\boldsymbol{x}), \quad \text{and} \quad f_3(\boldsymbol{x}).$$

Test Problem 3 seems to be the same as Test Problem 1 in Section III-A, but the problem size is different. Test Problem 3 is a 20-job and ten-machine problem, while Test Problem 1 was a ten-job and five-machine problem. For Test Problem 4, we used the following fitness function:

$$f(\boldsymbol{x}) = -w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}) - w_3 f_3(\boldsymbol{x}). \tag{21}$$

We applied the following four methods to these two test problems to compare their performance:

1) proposed hybrid algorithm with $k = 2$ and $N_{\text{elite}} = 3$;
2) VEGA in Schaffer [4];

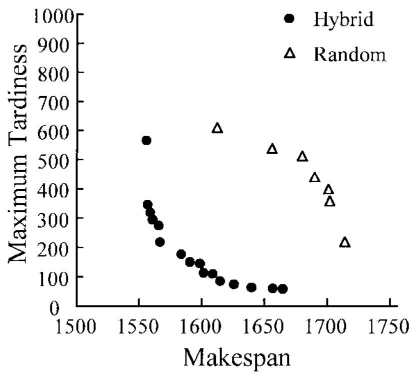

Fig. 16. Solutions obtained by the proposed hybrid algorithm and the random sampling method.

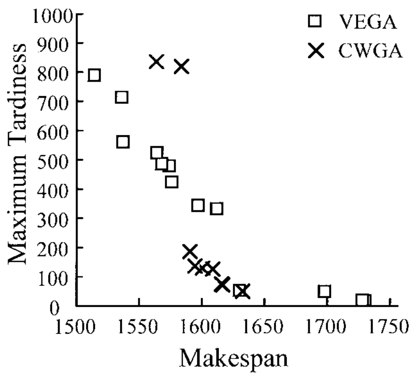

Fig. 17. Solutions obtained by the VEGA and the CWGA.

3) CWGA (the weight values were specified as $w_1 = w_2 = 0.5$ in Test Problem 3 and $w_1 = w_2 = w_3 = 1/3$ in Test Problem 4);
4) random sampling method (a large number of feasible schedules are randomly generated and each schedule is evaluated in this method).

The first three methods were applied to the test problems with the same parameter specifications: population size: $N_{\text{pop}} = 20$; crossover probability: 0.9; mutation probability for each string: 0.3; and stopping condition: evaluation of 100 000 solutions. In the random sampling method, we examined 2 000 000 feasible solutions of each test problem, which are 20 times as many as in the other three methods.

B. Simulation Results for Two-Objective Flowshop Scheduling Problems

We applied the proposed hybrid algorithm, VEGA in Schaffer [4], CWGA, and the random sampling method to Test Problem 3. Nondominated solutions obtained by each method are shown in Figs. 16 and 17. From Figs. 16 and 17, 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. 17). But no solutions obtained by the VEGA have very small values of both objectives if compared with nondominated solutions obtained by the proposed hybrid algorithm (see Fig. 16).
2) The variety of solutions obtained by the CWGA is not large (see Fig. 17).
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. 16).

In order to clarify these observations, all solutions obtained by the four algorithms were compared with each other and only nondominated solutions among all obtained solutions were selected. Some solutions obtained by one algorithm were dominated by other solutions obtained by other algorithms. The number of the nondominated solutions is shown in Table III. From Table III, we can see the high performance of the proposed hybrid algorithm because many solutions (i.e., 11 solutions: 69% of the obtained solutions) are not dominated by any other solutions.

TABLE III
SIMULATION RESULTS OF A SINGLE TRIAL OF EACH ALGORITHM FOR TEST PROBLEM 3

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

TABLE IV
AVERAGE RESULTS OVER 20 TRIALS OF EACH ALGORITHM FOR TEST PROBLEM 3

Algorithm	The number of obtained solutions (A)	The number of non-dominated solutions (B)	Ratio: B/A
Hybrid	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%

TABLE V
AVERAGE CPU TIME OF EACH ALGORITHM FOR TEST PROBLEM 3

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

Because all four algorithms are probabilistic search methods, their performance cannot be evaluated by a single trial. Thus, we applied each algorithm to Test Problem 3, 20 times. In each trial, obtained solutions by the four algorithms were compared in the same manner as in Table III. The average performance of each algorithm over the 20 trials is shown in Table IV. From Table IV, we can also see the high performance of the proposed hybrid algorithm.

The average CPU time of each algorithm is shown in Table V. From Table V, we can see that the average CPU times of the three GA's (i.e., the proposed hybrid algorithm, 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).

C. Simulation Results for Three-Objective Flowshop Scheduling Problems

In the same manner as in Table IV, we applied the four algorithms to Test Problem 4, 20 times. Average results of

TABLE VI
AVERAGE RESULTS OF OVER 20 TRIALS OF EACH ALGORITHM FOR TEST PROBLEM 4

Algorithm	The number of obtained solutions (A)	The number of non-dominated solutions (B)	Ratio: B/A
Hybrid	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 VII
AVERAGE QUALITY OF THE SOLUTION SET OBTAINED BY EACH ALGORITHM

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

the 20 trials of each algorithm are shown in Table VI. From Table VI, we can see the high performance of the proposed hybrid algorithm 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 nondominated solutions of Test Problem 4 by depicting them in the three-dimensional (3-D) objective space, we use an evaluation method in Esbensen [26] to measure the quality of a set of nondominated solutions obtained by each algorithm. Let us denote a set of nondominated solutions by Ω. Then the best solution $\boldsymbol{x}^*$ for a given weight vector $\mathbf{w} = (w_1, w_2, w_3)$ can be chosen from Ω as follows:

$$\begin{aligned} f(\boldsymbol{x}^*) &= -w_1 f_1(\boldsymbol{x}^*) - w_2 f_2(\boldsymbol{x}^*) - w_3 f_3(\boldsymbol{x}^*) \\ &= \max\{-w_1 f_1(\boldsymbol{x}) - w_2 f_2(\boldsymbol{x}) - w_3 f_3(\boldsymbol{x}) | \boldsymbol{x} \in \Omega\}. \end{aligned} \tag{22}$$

Esbensen [26] 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 $\mathbf{w} = (w_1, w_2, w_3)$. In this paper, we calculate the expected value of $f(\boldsymbol{x}^*)$ by randomly generating 10 000 weight vectors (say, $\mathbf{w}^1, \mathbf{w}^2, \cdots, \mathbf{w}^{10\,000}$) by (7). That is, the quality of the set of nondominated solutions Ω is calculated as follows:

$$\begin{aligned} q(\Omega) = \frac{1}{10\,000} \sum_{i=1}^{10\,000} \max\{-w_1^i f(\boldsymbol{x}) - w_2^i f(\boldsymbol{x}) \\ - w_3^i f(\boldsymbol{x}) | \boldsymbol{x} \in \Omega\} \end{aligned} \tag{23}$$

where $q(\Omega)$ is the quality of the solution set Ω and $\mathbf{w}^i = (w_1^i, w_2^i, w_3^i)$, $i = 1, 2, \cdots, 10\,000$ are randomly specified weight values.

For the set of nondominated solutions obtained by each trial of each algorithm, we calculated the quality of the solution set by (23). 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 VII. From Table VII, we can see that the best result (i.e., the maximum average quality) was obtained by the proposed hybrid algorithm.

D. Discussions on Parameter Specifications

We have already demonstrated high performance of our multi-objective genetic local search algorithm by computer simulations on multi-objective flowshop scheduling problems. In this subsection, we discuss the dependency of its performance on parameter specifications. Our hybrid algorithm has the following additional parameters that are not used in standard single objective GA's:

1) number of neighborhood solutions examined for each move in the local search procedure (i.e., k);
2) number of nondominated solutions added to the current population at each generation (i.e., N_{elite});
3) constant multipliers introduced for normalizing different objectives [i.e., five for the makespan in (10) and (18), two for the maximum tardiness in (11) and (19), and one for the total flowtime in (20)].

First, we examined the effect of the choice of k on the performance of our hybrid algorithm by computer simulations on Test Problem 1. Because each solution of Test Problem 1 with ten jobs has 81 neighborhood solutions, we can chose any integer in the closed interval [0, 81] for k. If we specified k as $k = 0$, our hybrid algorithm reduced to a multi-objective GA with no local search procedure. On the other hand, $k = 81$ means that the standard local search procedure was used in our hybrid algorithm. In the same manner as in Section III, we applied our hybrid algorithm to Test Problem 1 using various values of k (i.e., $k = 0, 1, 2, 3, 4, 5, 10, 20, 40, 81$). The value of N_{elite} was specified as $N_{\text{elite}} = 4$. We iterated this computer simulation 100 times for each value of k. In each trial, we examined whether all 12 nondominated solutions of this problem in Fig. 10 were obtained (i.e., how many solutions among the 12 nondominated solutions were obtained). We also calculated the average number of generation updates in our hybrid algorithm for each value of k. Simulation results are summarized in Table VIII, where the successful trial means that all 12 nondominated solutions were obtained in that trial. From these results, we can see that good results were obtained by $k = 1$–10. From Table VIII, we can also see that only a small number of generations were updated in our hybrid algorithm in the case of a large value of k. Because at least $N_{\text{pop}} \cdot (k + 1)$ solutions are examined at each generation of our hybrid algorithm, the number of generation updates can be roughly approximated by the following formulation:

$$N_{\text{generation}} \cong \frac{N_{\text{stop}}}{N_{\text{pop}} \cdot (k+1)} \tag{24}$$

where $N_{\text{generation}}$ is the number of generation updates, N_{stop} is the total number of examined solutions used as the stopping condition, and N_{pop} is the population size (i.e., the number of solutions at each generation). In Table VIII, N_{stop} and N_{pop} were specified as $N_{\text{stop}} = 10\ 000$ and $N_{\text{pop}} = 20$, respectively. From Table VIII, we can see that the above formulation is a good approximation for the relation between the value of k and the number of generation updates. In general, a certain number of generation updates are required for obtaining good results by GA's. Thus, we can specify the value of k by (24) when the number of required generation updates is given.

Next we examine the dependency of the performance of our hybrid algorithm on the choice of N_{elite}. In the same manner as in Section III, we applied our hybrid algorithm to Test Problem

TABLE VIII
SIMULATION RESULTS WITH VARIOUS VALUES OF k FOR TEST PROBLEM 1

The value of k	0	1	2	3	4	5	10	20	40	81
The number of successful trials	24	36	26	45	29	33	25	20	8	0
The number of obtained solutions	10.26	10.88	10.61	10.76	10.73	10.70	10.57	10.24	9.85	8.48
The number of generation updates	500.0	218.7	138.6	101.7	79.0	64.8	33.1	16.1	8.0	4.1

TABLE IX
SIMULATION RESULTS WITH VARIOUS VALUES OF N_{elite} FOR TEST PROBLEM 1

The value of N_{elite}	0	1	2	3	4	5	6	7	8	9
The number of successful trials	0	2	16	33	45	43	42	32	39	35
The number of obtained solutions	7.44	9.20	10.06	10.67	10.76	10.88	11.05	10.76	11.11	10.92

1 using various values of N_{elite} (i.e., $N_{\text{elite}} = 0, 1, \cdots, 9$). The value of k was specified as $k = 3$. For each value of N_{elite}, we iterated the computer simulation 100 times. Simulation results are summarized in Table IX. From Table IX, we can see that the performance of our hybrid algorithm was significantly deteriorated by specifying N_{elite} as $N_{\text{elite}} = 0$. This means that elite solutions play a significant role in our hybrid algorithm. We can also see that the performance of our hybrid algorithm is not sensitive to the choice of N_{elite} if $N_{\text{elite}} > 3$.

Finally, we examine the dependency of the performance of our hybrid algorithm on the choice of the normalization factors. Because the weight value w_i for each objective in the fitness function (4) is randomly specified whenever a pair of parent solutions are selected in our hybrid algorithm, its performance is not sensitive to the choice of the normalization factors. In the computer simulations of the previous sections, we used the normalization factors (5, 2, 1) for the makespan, the maximum tardiness, and the total flowtime, respectively. We performed the same computer simulations without the normalization factors. Simulation results by our hybrid algorithm without the normalization factors were almost the same as those with the normalization factors. Only the simulation results by CWGA were directly affected by the values of the normalization factors. This is because the fixed search direction in the CWGA is specified by the values of the normalization factors. While almost the same results were obtained by our hybrid algorithm without the normalization factors for the test problems in this paper, they may remedy the difficulty in obtaining nondominated solutions of multi-objective flowshop scheduling problems when the difference of the variance of each objective is very large. In such a case, we can determine the value of each normalization factor by the inverse of the standard deviation of the corresponding objective. The standard deviation of each objective can be estimated by randomly generating a number of schedules.

V. CONCLUSION

In this paper, we proposed a multi-objective genetic local search algorithm. The proposed algorithm is an extension of our multi-objective GA in [9] to a hybrid algorithm. In the proposed algorithm, a local search procedure is applied to each solution generated by genetic operations. By computer simulations on flowshop scheduling problems, high performance of the proposed algorithm was demonstrated. It was also shown that the proposed algorithm can handle a multi-objective optimization problem with a nonconvex feasible region in the objective space.

The characteristic features of the proposed hybrid algorithm can be summarized as follows.

1) A weighted sum of multiple objectives is used as a fitness function in a selection operation. The weight values in the fitness function are randomly specified whenever a pair of parent solutions are selected. That is, each selection is performed with a different weight vector.
2) A local search procedure is applied to each new solution generated by the genetic operations (i.e., selection, crossover, and mutation). The local search for each new solution is performed to maximize the fitness function that 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 neighborhood solutions of a current solution are not examined for each move. That is, the number of examined neighborhood solutions of a current solution is restricted in the local search. This is to prevent the local search from spending almost all available computation time.
4) A tentative set of nondominated 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.

The proposed algorithm is simple, and its computation time is almost the same as that of a CWGA with no local search, as shown in computer simulations in Section IV. This simplicity is also an advantage of the proposed hybrid algorithm.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: Univ. of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[3] L. Davis, Ed., *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

[4] J. D. Schaffer, “Multi-objective optimization with vector evaluated genetic algorithms,” in *Proc. 1st Int. Conf. Genetic Algorithms,* 1985, pp. 93–100.

[5] F. Kursawe, “A variant of evolution strategies for vector optimization,” in *Parallel Problem Solving from Nature,* H.-P. Schwefel and R. Männer, Eds. Berlin, Germany: Springer-Verlag, 1991, pp. 193–197.

[6] J. Horn, N. Nafpliotis, and D. E. Goldberg, “A niched Pareto genetic algorithm for multi-objective optimization,” in *Proc. 1st IEEE Int. Conf. Evolutionary Computat.,* 1994, pp. 82–87.

[7] C. M. Fonseca and P. J. Fleming, “Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization,” in *Proc. 5th Int. Conf. Genetic Algorithms,* 1993, pp. 416–423.

[8] ——, “An overview of evolutionary algorithms in multiobjective optimization,” *Evolutionary Computat.,* vol. 3, no. 1, pp. 1–16, 1995.

[9] T. Murata and H. Ishibuchi, “MOGA: Multi-objective genetic algorithms,” in *Proc. 2nd IEEE Int. Conf. Evolutionary Computat.,* 1995, pp. 289–294.

[10] H. Tamaki, M. Mori, and M. Araki, “Generation of a set of Pareto-optimal solutions by genetic algorithms,” *Trans. Soc. Instrum. Contr. Eng.,* vol. 31, no. 8, pp. 1185–1192, 1995 (in Japanese).

[11] 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,” in *Proc. 3rd Int. Conf. Genetic Algorithms,* 1989, pp. 110–115.

[12] N. L. J. Ulder, E. H. L. Aarts, H.-J. Bandelt, P. J. M. von Laarhoven, and E. Pesch, “Genetic local search algorithms for the traveling salesman problem,” in *Parallel Problem Solving from Nature,* H.-P. Schwefel and R. Männer, Eds. Berlin, Germany: Springer-Verlag, 1991, pp. 109–116.

[13] C. A. Glass, C. N. Potts, and P. Shade, “Genetic algorithms and neighborhood search for scheduling unrelated parallel machines,” Univ. Southampton, Southampton, U.K., Preprint Series OR47, 1992.

[14] H. Ishibuchi, N. Yamamoto, T. Murata, and H. Tanaka, “Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems,” *Fuzzy Sets Syst.,* vol. 67, pp. 81–100, 1994.

[15] T. Murata and H. Ishibuchi, “Performance evaluation of genetic algorithms for flowshop scheduling problems,” in *Proc. 1st IEEE Int. Conf. Evolutionary Computat.,* 1994, pp. 812–817.

[16] S. M. Johnson, “Optimal two- and three-stage production schedules with setup times included,” *Naval Res. Logistics Quart.,* vol. 1, no. 1, pp. 61–68, 1954.

[17] K. R. Baker and G. D. Scudder, “Sequencing with earliness and tardiness penalties: A review,” *Oper. Res.,* vol. 38, no. 1, pp. 22–36, 1990.

[18] R. A. Dudek, S. S. Panwalkar, and M. L. Smith, “The lessons of flowshop scheduling research,” *Oper. Res.,* vol. 40, no. 1, pp. 7–13, 1992.

[19] R. L. Daniels and R. J. Chambers, “Multiobjective flow-shop scheduling,” *Naval Res. Logistics Quart.,* vol. 37, pp. 981–995, 1990.

[20] C. Rajendran, “Two-stage flowshop scheduling problem with bicriteria,” *J. Oper. Res. Soc.,* vol. 43, no. 9, pp. 871–884, 1992.

[21] K. Morizawa, H. Nagasawa, and N. Nishiyama, “A new procedure for generating initial solutions in a multiobjective scheduling method using complex random sampling,” *J. Jpn. Ind. Manage. Assoc.,* vol. 44, no. 6, pp. 510–516, 1994 (in Japanese).

[22] ——, “Two-machine flowshop scheduling to minimize makespan, total flow time and maximum tardiness,” *J. Jpn. Ind. Manage. Assoc.,* vol. 43, no. 3, pp. 186–192, 1992 (in Japanese).

[23] E. Taillard, “Some efficient heuristic methods for the flow shop sequencing problem,” *Eur. J. Oper. Res.,* vol. 47, no. 1, pp. 65–74, 1990.

[24] I. H. Osman and C. N. Potts, “Simulated annealing for permutation flow-shop scheduling,” *OMEGA,* vol. 17, no. 6, pp. 551–557, 1989.

[25] H. Ishibuchi, S. Misaki, and H. Tanaka, “Modified simulated annealing algorithms for the flow shop sequencing problem,” *Eur. J. Oper. Res.,* vol. 81, no. 2, pp. 388–398, 1995.

[26] H. Esbensen, “Defining solution set quality,“ Elect. Res. Lab., College Eng., Univ. California, Berkeley, Memo. UCB/ERL M96/1, Jan. 1996.

Hisao Ishibuchi (M’93) received the B.S. and M.S. degrees in precision mechanics from Kyoto University, Kyoto, Japan, in 1985 and 1987, respectively, and the Ph.D. degree from Osaka Prefecture University, Osaka, Japan, in 1992.

He was a Visiting Research Associate at the University of Toronto, Toronto, Ont., Canada, from August 1994 to March 1995 and from July 1997 to March 1998. He is currently an Associate Professor in the Department of Industrial Engineering, Osaka Prefecture University. His current research interests include fuzzy rule-based classification systems, fuzzy neural networks, genetic algorithms, fuzzy scheduling, and evolutionary games.

Dr. Ishibuchi is a member of INNS and IFSA.

Tadahiko Murata (M’96) received the B.S., M.S., and Ph.D. degrees from the Department of Industrial Engineering, Osaka Prefecture University, Osaka, Japan, in 1994, 1996, and 1997, respectively.

He is currently an Assistant Professor in the Department of Industrial and Systems Engineering, Ashikaga Institute of Technology, Tochigi, Japan. His research interests include genetic algorithms, scheduling problems, and pattern classification problems.