Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling

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

* Department of Industrial Engineering, Osaka Prefecture University

1-1 Gakuen-cho, Sakai, Osaka 599-8531, Japan

Fax +81-72-254-9915, E-mail: {hisaoi, yossy}@ie.osakafu-u.ac.jp

** Department of Informatics, Faculty of Informatics, Kansai University

2-1-1 Ryozenji-cho, Takatsuki, Osaka 569-1095, Japan

Fax +81-72-690-2491, E-mail: murata@res.kutc.kansai-u.ac.jp

***Abstract* -** This paper shows how the performance of evolutionary multiobjective optimization (EMO) algorithms can be improved by the hybridization with local search. The main positive effect of the hybridization is the improvement in the convergence speed to the Pareto-front. On the other hand, the main negative effect is the increase in the computation time per generation. Thus the number of generations is decreased when the available computation time is limited. As a result, the global search ability of EMO algorithms is not fully utilized. These positive and negative effects are examined by computational experiments on multiobjective permutation flowshop scheduling problems. Results of our computational experiments clearly show the importance of striking a balance between genetic search and local search. In this paper, we first modify our former multiobjective genetic local search (MOGLS) algorithm by choosing only good individuals as initial solutions for local search and assigning an appropriate local search direction to each initial solution. Next we demonstrate the importance of striking a balance between genetic search and local search through computational experiments. Then we compare the modified MOGLS with recently developed EMO algorithms: SPEA and NSGA-II. Finally, we demonstrate that local search can be easily combined with those EMO algorithms for designing multiobjective memetic algorithms.

***Index Terms* -** Multiobjective optimization, evolutionary multiobjective optimization, memetic algorithms, genetic local search, permutation flowshop scheduling.

I. Introduction

Since Schaffer's study [1], evolutionary algorithms have been applied to various multiobjective optimization problems for finding their Pareto-optimal solutions. Evolutionary algorithms for multiobjective optimization are often referred to as EMO (evolutionary multiobjective optimization) algorithms. For review of this field, see [2]-[5]. The task of EMO algorithms is to find Pareto-optimal solutions as many as possible. In early studies on EMO algorithms (e.g., [6]-[8]), emphasis was mainly placed on the diversity of solutions in order to find uniformly distributed Pareto-optimal solutions. Thus several concepts such as niching, fitness sharing, and mating restriction were introduced into EMO algorithms. In recent studies (e.g., [9]-[13]), emphasis was placed on the convergence speed to the Pareto-front as well as the diversity of solutions. In those studies, some form of elitism was used as an important ingredient of EMO algorithms. It was shown that the use of elitism improved the convergence speed to the Pareto-front [12].

One promising approach for improving the convergence speed to the Pareto-front is the use of local search in EMO algorithms. Hybridization of evolutionary algorithms with local search has already been investigated for single-objective optimization problems in many studies (e.g., [14], [15]). Such a hybrid algorithm is often referred to as a memetic algorithm. See Moscato [16] for an introduction to this field and [17]-[19] for recent developments. The hybridization with local search for multiobjective optimization was first implemented in [20], [21] as a multiobjective genetic local search (MOGLS) algorithm where a scalar fitness function with random weights was used for the selection of parents and the local search for their offspring. Jaszkiewicz [22] improved the performance of the MOGLS by modifying its selection mechanism of parents. While his MOGLS still used the scalar fitness function with random weights in selection and local search, it did not use the roulette wheel selection over the entire population. A pair of parents was randomly selected from a pre-specified number of the best solutions with respect to the scalar fitness function with the current weights. This selection scheme can be viewed as a kind of mating restriction in EMO algorithms. Knowles & Corne [23] combined their Pareto archived evolution strategy (PAES [9], [11]) with a crossover operation for designing a memetic PAES (M-PAES). In their M-PAES, the Pareto-dominance relation and the grid-type partition of the objective space were used for determining the acceptance (or rejection) of new solutions generated in genetic search and local search. The M-PAES had a special form of elitism inherent in the PAES. The performance of the M-PAES was examined in [24] for multiobjective knapsack problems and [25] for degree-constrained multiobjective MST (minimum-weight spanning tree) problems. In those studies, the M-PAES was compared with the PAES, the MOGLS of Jaszkiewicz [22], and an EMO algorithm. In the above-mentioned hybrid EMO algorithms (i.e., multiobjective memetic algorithms [20]-[25]), local search was applied to individuals in every generation. In some studies [26], [27], local search was applied to individuals only in the final generation. While Deb and Goel [26] used local search for decreasing the number of non-dominated solutions (i.e., for decreasing the diversity of final solutions), Talbi et al. [27] intended to increase the diversity of final solutions by the application of local search. In this paper, we apply local search to solutions in every T generations. While T is implicitly assumed as $T = 1$ in many computational experiments of this paper as in [20]-[25], other values of T (e.g., $T = 10$, 100) are also examined in some computational experiments.

In many combinatorial optimization problems, local search can be much more efficiently executed than genetic search. Jaszkiewicz [22] mentioned that local search performed almost 300 times more function evaluations per second than genetic search in the application of his MOGLS to multiobjective traveling salesperson problems (TSPs). This is mainly because local search only needs the difference in the objective values (i.e., $\Delta f = f(\mathbf{x}) - f(\mathbf{x}'))$ between the current solution $\mathbf{x}$ and its neighbor $\mathbf{x}'$ instead of the objective value $f(\mathbf{x}')$ of $\mathbf{x}'$. In the case of TSPs, the complexity of the calculation of Δf is $O(1)$ while that of $f(\mathbf{x}')$ is $O(n)$ where n is the number of cities (for details, see [28], [29]). For example, let us consider Fig. 1 where a new tour is generated by removing the edges (1, 2) and (6, 7) and adding the edges (1, 6) and (2, 7). The difference in the objective values between the two tours can be calculated from only those four edges. On the other hand, when a new tour is generated by genetic operations, we usually have to consider much more edges for evaluating the new tour. In addition to the efficient evaluation of new solutions (i.e., neighbors), they can be much more efficiently generated in local search than genetic search. This is because genetic search uses three steps (i.e., selection, crossover and mutation) for generating new solutions while local search uses a single step.

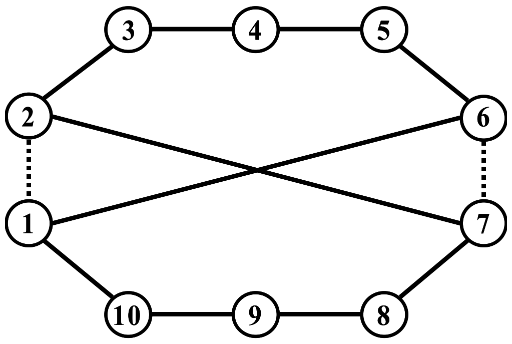


Fig. 1 An example of a new tour generated by a local search operation.

We use some variants of the MOGLS in [20], [21] for multiobjective permutation flowshop scheduling. Flowshop is one of the most frequently studied scheduling problems in the literature (see [30] for an introduction to this field). Permutation flowshop scheduling is to find an optimal permutation of n jobs processed on m machines. Thus the size of the search space is $n!$. Many objectives have been studied in the literature such as the makespan, total flow time, maximum tardiness, and total tardiness. Except for some special cases (e.g., two-machine flowshop scheduling for minimizing the makespan), m-machine n-job permutation flowshop scheduling problems are $\mathcal{NP}$-hard (see Brucker [31] for the complexity of scheduling problems). In flowshop scheduling, new solutions can be much more efficiently generated in local search than genetic search as in the case of TSPs. The evaluation of new solutions in local search for flowshop scheduling, however, is not much faster than genetic search. This is because the calculation of the difference in the objective values cannot be efficiently performed for commonly used neighborhood structures. For example, let us consider a schedule in Fig. 2 for a three-machine ten-job problem. From the schedule in Fig. 2, we generate a new schedule in Fig. 3 by the same local search operation as Fig. 1 for TSPs. We can see that the completion time of each job is different between Fig. 2 and Fig. 3 except for the first job. This means that the recalculation of the completion time of each job is necessary for evaluating a new schedule generated by the local search operation. Thus the computation time for evaluating a new schedule in local search is the same order of magnitude as that in genetic search. For the use of approximate evaluation of solutions in scheduling problems in order to speed up the search, see Watson et al. [32] where fast low-resolution and slow high-resolution simulations were compared with each other.

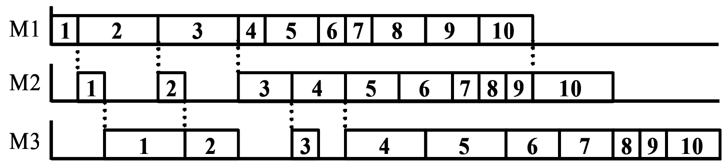


Fig. 2 An example of a schedule for a three-machine ten-job problem.

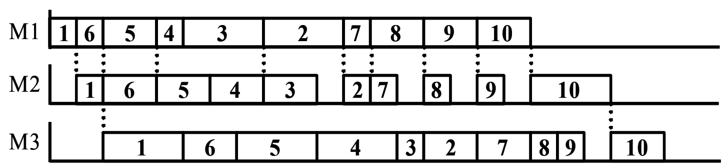


Fig. 3 An example of a new schedule generated by the same local search operation as Fig. 1.

In the former MOGLS [20], [21], we used an early termination strategy for decreasing the computation time spent by local search. In this strategy, neighbors of the current solution are examined in a random order. Then the current solution is replaced with the first neighbor that is better than the current solution (i.e., not the best improvement but the first improvement). The execution of local search was terminated when no better solution is found among k neighbors randomly generated from the current solution where k is a user-definable parameter. The same early termination strategy was used in the M-PAES [23]. On the other hand, all neighbors were examined in the MOGLS of Jaszkiewicz [22]. In Knowles and Corne [24], the early termination strategy was used in Jaszkiewicz's MOGLS as well as the M-PAES in their computational experiments on multiobjective knapsack problems.

In this paper, we introduce a local search probability p_{LS} to the former MOGLS [20], [21] for decreasing the computation time spent by local search. In the modified MOGLS, local search is not applied to all solutions in the current population but probabilistically applied to selected solutions with the probability p_{LS}. We used a different parameter N_{LS} (i.e., the number of solutions selected for local search) in our previous study [33]. While these two parameters have the same effect on the computation time spent by local search, we use the local search probability p_{LS} in this paper because the specification of N_{LS} depends on the population size (e.g., $N_{\text{LS}} = 50$ for the population size 50 has a totally different meaning from $N_{\text{LS}} = 50$ for the case of the population size 100). We try to strike a balance between genetic search and local search using the two parameters k and p_{LS} in local search. We also use another parameter T in some computational experiments where local search is applied to solutions in every T generations.

This paper is organized as follows. In Section II, we briefly describe the former MOGLS [20], [21] where local search was applied to all solutions in every generation. We show that the performance of the former MOGLS can be improved by applying local search to not all solutions but only good ones. We also discuss other implementation issues such as the specification of an objective function used in local search and the choice of a neighborhood structure. In Section III, we demonstrate the importance of striking a balance between genetic search and local search. Through computational experiments with various specifications of the three parameters in local search (i.e., k, p_{LS} and T), we show positive and negative effects of the hybridization with local search on the performance of EMO algorithms. We also examine the necessity of genetic operations in our MOGLS through computational experiments with various specifications of the crossover and mutation probabilities. In Section IV, we compare our MOGLS with the strength Pareto evolutionary algorithm (SPEA [10]) and the revised non-dominated sorting genetic algorithm (NSGA-II [13]). Then we show that local search is easily combined with those EMO algorithms for designing multiobjective memetic algorithms. We conclude this paper in Section V where some topics for future research are also suggested.

II. MOGLS ALGORITHMS

The outline of our MOGLS can be written in a generic form as Fig. 4. This figure shows a basic structure of simple memetic algorithms. For other types of memetic algorithms, see Krasnogor [34] where taxonomy of memetic algorithms was given using an index number *D*. Our MOGLS is a $D = 4$ memetic algorithm in his taxonomy (for details, see [34]).

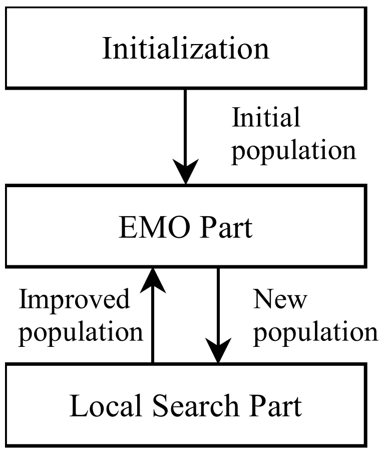


Fig. 4 Generic form of our MOGLS.

A. Former MOGLS

We explain the former MOGLS [20], [21] using the following *N*-objective minimization problem:

$$\text{Minimize } \mathbf{z} = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_N(\mathbf{x}))\,, \tag{1}$$

$$\text{subject to } \mathbf{x} \in \mathbf{X}\,, \tag{2}$$

where **z** is the objective vector, **x** is the decision vector, and **X** is the feasible region in the decision space.

One issue to be considered in the hybridization of EMO algorithms with local search is the specification of an objective function to be optimized by local search. In the former MOGLS, the following scalar fitness function to be minimized was used in both the selection of parents and the local search for their offspring.

$$f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \cdots + w_N f_N(\mathbf{x}) . \tag{3}$$

The weight w_i ($w_i \geq 0$, $i = 1,2,...,N$ and $\sum_i w_i = 1$) was randomly specified whenever a pair of parents was to be selected. That is, each selection was governed by a different weight vector. A local search procedure was applied to each offspring using the same scalar fitness function (i.e., the same weight vector) as in the selection of its parents.

Another issue is the balance between genetic search and local search. For decreasing the computation time spent by local search, only a small number of neighbors of the current solution were examined. It was shown in [21] that the performance of the former MOGLS was deteriorated when all neighbors were examined. The former MOGLS used a simple form of elitism where all non-dominated solutions obtained during its execution were stored in a secondary population separately from the current population. A few non-dominated solutions were randomly selected from the secondary population and their copies were added to the current population. The former MOGLS is written as follows:

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

Step 1) Evaluation: Calculate the N objectives for each solution in the current population. Then update the secondary population where non-dominated solutions are stored separately from the current population.

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

(a) Randomly specify the weights $w_1, w_2, ..., w_N$ where $w_i \geq 0$ for $i = 1,2,...,N$ and $\sum_i w_i = 1$.

(b) Select a pair of parents based on the scalar fitness function in (3). The selection probability $p_{\text{S}}(\mathbf{x})$ of each solution $\mathbf{x}$ in the current population Ψ is specified by the following roulette wheel selection scheme with the linear scaling:

$$p_{\text{S}}(\mathbf{x}) = \frac{f_{\max}(\Psi) - f(\mathbf{x})}{\sum\limits_{\mathbf{y}\in\Psi} (f_{\max}(\Psi) - f(\mathbf{y}))}, \tag{4}$$

where $f_{\max}(\Psi)$ is the maximum (i.e., worst) fitness value among the current population Ψ.

Step 3) Crossover and mutation: Apply a crossover operation to each of the selected ($N_{\text{pop}} - N_{\text{elite}}$) pairs of parents with the crossover probability p_{C}. A new solution is generated from each pair. When the crossover operation is not applied, one parent is randomly chosen and handled as a new solution. Then apply a mutation operation to each new solution with the mutation probability p_{M}.

Step 4) Elitist strategy: Randomly select N_{elite} solutions from the secondary population. Then add their copies to the ($N_{\text{pop}} - N_{\text{elite}}$) solutions generated in Step 3 to construct a population of N_{pop} solutions.

Step 5) Local search: Apply a local search procedure to each of the N_{pop} solutions in the current population using the scalar fitness function in (3). For each solution, the weight vector used in the selection of its parents is also used in local search. Only for a solution with no parents (i.e., solution generated in the initial generation in Step 0), we use a random weight vector. Local search is terminated when no better solution is found among k neighbors that are randomly selected from the neighborhood of the current solution. After local search is applied to all solutions in the current population, the current population is replaced with the improved solutions (i.e., this algorithm is a Lamarckian multiobjective memetic algorithm).

Step 6) Return to Step 1.

This algorithm is terminated when a pre-specified number of solutions are examined during its execution. In the local search part (i.e., Step 5), a neighbor is randomly generated from the neighborhood of the current solution. If the neighbor is better than the current solution, the current solution is replaced. That is, the first improvement strategy is used in the local search part instead of the best improvement strategy. When the current solution is updated, local search continues for the new current solution in the same manner.

In this algorithm, all non-dominated solutions are stored in the secondary population with no restriction (i.e., no upper bound) on its size. In general, the restriction is necessary from the viewpoint of memory storage and computation time (e.g., see the SPEA [10]). We use, however, no restriction because we did not encounter any difficulties related to the maintenance of the secondary population in our computational experiments on permutation flowshop scheduling problems reported in this paper. Of course, there may be many application fields where the restriction on the size of the secondary population is necessary.

Randomly selected N_{elite} solutions from the secondary population in Step 4 work as elite solutions. It was shown in [21] that the performance of this algorithm was deteriorated by specifying the value of N_{elite} as $N_{\text{elite}} = 0$ (i.e., no elitism). It was also shown that the performance was not sensitive to the value of N_{elite} when $N_{\text{elite}} \geq 2$. In this paper, the value of N_{elite} is specified based on preliminary computational experiments as $N_{\text{elite}} = 10$ (see Subsection II.C).

B. Modified MOGLS

In the above-mentioned MOGLS, the scalar fitness function in local search for each solution was specified by the weight vector used in the selection of its parents. This specification of the scalar fitness function in local search is not always appropriate. Using Fig. 5, we illustrate the drawback of this specification method. Let us assume that two solutions ***a*** and ***b*** denoted by closed circles are selected as parents based on the scalar fitness function with the weight vector $\mathbf{w} = (0.1,\ 0.9)$ for a two-objective minimization problem. This scalar fitness function is also used in local search. Since the two objectives in Fig. 5 should be minimized, $-\mathbf{w} = (-0.1,\ -0.9)$ can be viewed as the local search direction for a new solution generated from the selected parents. In this paper, the local search direction means the direction with the steepest improvement of the objective function in the objective space, which is $-\mathbf{w} = (-w_1,\ \ldots,\ -w_N)$ for the scalar fitness function in (3). When an offspring is generated around the parents (e.g., solution A in Fig. 5), $-\mathbf{w} = (-0.1,\ -0.9)$ is appropriate as the local search direction for the offspring. On the contrary, when an offspring is far from its parents (e.g., solution B in Fig. 5), $-\mathbf{w} = (-0.1,\ -0.9)$ is not appropriate as its local search direction. As we can see from Fig. 5, an appropriate local search direction for each offspring depends on its location in the objective space. For example, $(-0.9,\ -0.1)$ seems to be much more appropriate for the solution B than $(-0.1,\ -0.9)$ as its local search direction. These discussions suggest the importance of the specification of an appropriate local search direction for each offspring according to its location in the objective space.

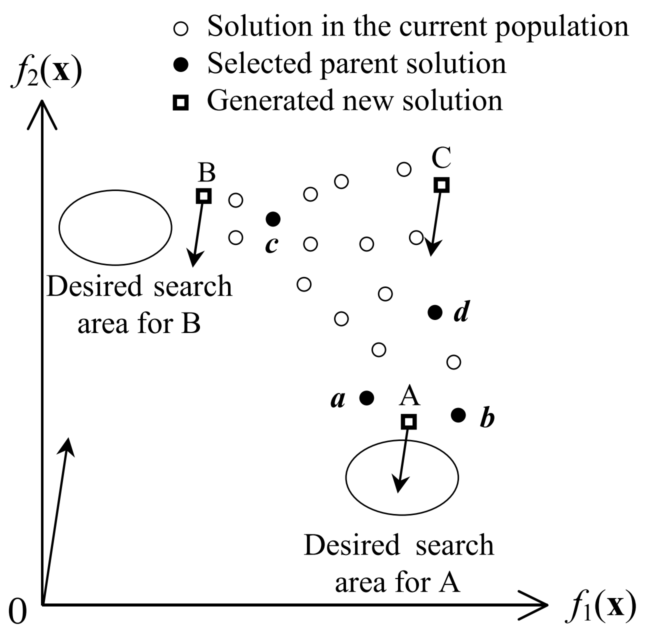

○ Solution in the current population
● Selected parent solution
□ Generated new solution
$f_2(\mathbf{x})$
$f_1(\mathbf{x})$
0
B
C
A
c
d
a
b
Desired search area for B
Desired search area for A

Fig. 5 Specification of a local search direction for an offspring.

When the quality of an offspring is very poor (e.g., solution C in Fig. 5), the application of local search seems to be waste of the computation time. Thus local search should be applied to only good offspring. That is, the choice of offspring, to which local search is applied, is also important in the MOGLS.

When two parents are similar to each other (e.g., ***a*** and ***b*** in Fig. 5), their offspring are usually similar to the parents. Thus appropriate initial solutions (e.g., A) are likely to be generated from good parents that are similar to each other. On the other hand, when two parents are not similar to each other (e.g., ***c*** and ***d***), inappropriate solutions are much more likely to be generated than the case of similar parents with high fitness values (e.g., ***a*** and ***b***). These discussions suggest that the use of parent selection schemes with high selection pressure may improve the performance of the former MOGLS with the roulette wheel selection. Such an approach to the modification of the former MOGLS will be further discussed in later through computational experiments.

In this subsection, we modify the former MOGLS by introducing a probabilistic selection scheme of initial solutions for local search. For choosing only good offspring and specifying an appropriate local search direction for each offspring, we modify Step 5 of the former MOGLS as follows:

Step 5) Local search: Iterate the following three steps N_{pop} times. Then replace the current population with N_{pop} solutions obtained by the following steps.

(a) Randomly specify the weights $w_1, w_2, ..., w_N$ where $w_i \geq 0$ for $i = 1,2,...,N$ and $\sum_i w_i = 1$.

(b) Select a solution from the current population using tournament selection with replacement based on the scalar fitness function with the current weights specified in (a). A copy of the selected solution is used in (c). Thus no solution is removed from the current population. In our computational experiments, the tournament size for the selection of an initial solution for local search is specified as five (See Subsection II.C).

(c) Apply local search to a copy of the selected solution using the current weights with the local search probability p_{LS}. The local search procedure is the same as in the former MOGLS. When local search is applied to a copy of the selected solution, the final solution where local search is terminated is included in the next population. On the other hand, when local search is not applied, a copy of the selected solution is included in the next population.

The basic idea is not to try to specify an appropriate local search direction to each solution but to choose an appropriate solution for a randomly specified local search direction. Moreover local search is not applied to all the selected solutions. We use the local search probability p_{LS} for decreasing the number of solutions to which local search is applied. Our idea is illustrated in Fig. 6 where local search is applied to only three solutions. As shown in this figure, the proposed algorithm chooses a good initial solution in Step 5 (b) with respect to the current local search direction specified in Step 5 (a). While the local search direction is randomly specified, the search is not a random walk because different solutions are chosen as initial solutions for different local search directions (see Fig. 6). It should be noted that the current solution does not move to any dominated neighbors because the weights are specified as $w_i \geq 0$ for $i = 1,2,...,N$ in the scalar fitness function. That is, local search does not degrade the current solution in the sense of the Pareto-dominance relation. This issue will be further discussed later.

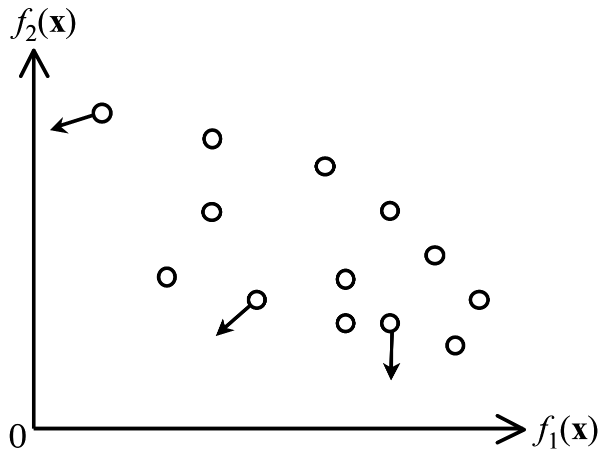


Fig. 6 Illustration of the selection of initial solutions for local search.

C. Test Problems and Performance Measures

Before demonstrating how the performance of the former MOGLS can be improved by the modification in its local search part, we explain test problems and performance measures used in this paper. In the same manner as in [21], we generated eight m-machine n-job permutation flowshop scheduling problems. 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 by adding a random integer in the interval [-100, 100] to its actual completion time in a randomly generated schedule. All the eight test problems have 20 machines (i.e., $m = 20$). Using the number of objectives (N) and the number of jobs (n), we denote each test problem as N/n where $N = 2$, 3 and $n = 20$, 40, 60, 80. Four test problems have two objectives (i.e., $N = 2$): to minimize the makespan and the maximum tardiness. The other four test problems are three-objective problems (i.e., $N = 3$) with an additional objective: to minimize the total flow time. Details of each test problem are available from the first author's homepage (http://www.ie.osakafu-u.ac.jp/~hisaoi/ci_lab_e/index.html).

Our three-objective test problems can be written in the format of Ausiello et al. [35] as follows (two-objective test problems can be also written in the same manner):

INSTANCE: n jobs $\{J_1, J_2, ..., J_n\}$, m machines $\{M_1, M_2, ..., M_m\}$, an $n \times m$ matrix whose (i, j) element is the processing time of the i-th job on the j-th machine, and an n-dimensional vector whose i-th element d_i is the due date of the i-th job.

SOLUTION: A set of non-dominated solutions with respect to the given objectives. Each solution is a permutation of $\{J_1, J_2, ..., J_n\}$.

OBJECTIVES: $\max\{C_i \mid i = 1,2,...,n\}$, $\max\{\max\{(C_i - d_i), 0\} \mid i = 1,2,...,n\}$, and $\sum_{i=1}^{n} C_i$ where C_i is the completion time of the i-th job, which is calculated from the $n \times m$ matrix. All the three-objectives are to be minimized.

As in [21], we used the two-point crossover in Fig. 7 and the insertion mutation in Fig. 8. The insertion mutation is often referred to as the shift mutation in the literature. We also used the insertion mutation as a local search operation for generating a neighbor of the current solution. The choice of a local search operation will be discussed later through computational experiments. Good results were reported in [36] where the insertion mutation was used in tabu search for minimizing the makespan. Good results were also reported by simulated annealing with the insertion mutation [37], [38]. Several crossover and mutation operations were examined in genetic algorithms for flowshop scheduling problems in [39] where good results were obtained from the combination of the two-point crossover and the insertion mutation. Moreover, the simultaneous use of different mutation operations with adaptive mutation probabilities was examined for two-objective flowshop scheduling problems in the framework of multiobjective memetic algorithms in Basseur et al. [40]. See Bagchi [41] for applications of multiobjective genetic algorithms to shop scheduling problems including flowshop, jobshop and openshop.

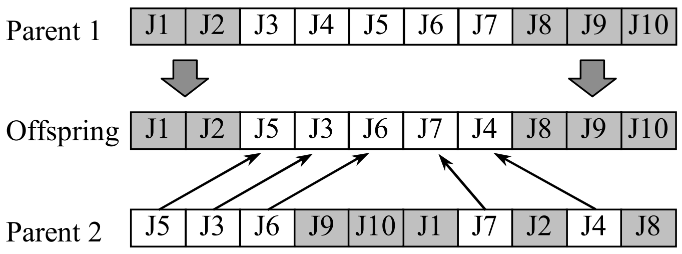


Fig. 7 Two-point crossover.

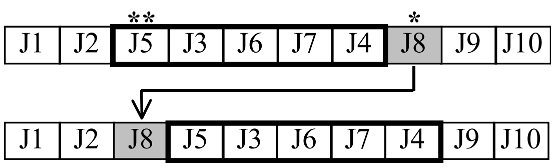


Fig. 8 Insertion mutation.

Next we briefly describe performance measures used in this paper for comparing many solution sets obtained from different algorithms or different parameter specifications. We use performance measures that are applicable to simultaneous comparison of many solution sets. Let S_j be a solution set ($j = 1,2,...,J$). For comparing J solution sets (S_1, S_2, ..., S_J), we use several performance measures because it is impossible to evaluate all aspects of each solution set using a single performance measure (see [4], [5], [42] for a number of performance measures).

We mainly use a performance measure based on the distance from a reference solution set (i.e., the Pareto-optimal solution set or a near Pareto-optimal solution set) for evaluating the solution set S_j. More specifically, we use the average distance from each reference solution to its nearest solution in S_j. This measure was used in Czyzak and Jaszkiewicz [43] and referred to as $\mathrm{D1_R}$ in Knowles and Corne [42]. Let S^* be the reference solution set. The $\mathrm{D1_R}$ measure can be written as

$$\mathrm{D1_R}(S_j) = \frac{1}{|S^*|} \sum_{\mathbf{y} \in S^*} \min\{d_{\mathbf{xy}} \mid \mathbf{x} \in S_j\} , \tag{5}$$

where $d_{\mathbf{xy}}$ is the distance between a solution $\mathbf{x}$ and a reference solution $\mathbf{y}$ in the N-dimensional normalized objective space:

$$d_{\mathbf{xy}} = \sqrt{(f_1^*(\mathbf{y}) - f_1^*(\mathbf{x}))^2 + \cdots + (f_N^*(\mathbf{y}) - f_N^*(\mathbf{x}))^2} , \tag{6}$$

where $f_i^*(\cdot)$ is the i-th objective that is normalized using the reference solution set S^*. We will explain the normalization of the objective space later. The smaller the value of $\mathrm{D1_R}(S_j)$ is, the better the solution set S_j is.

It should be noted that the $\mathrm{D1_R}$ measure in (5) is not the average distance from each solution in S_j to its nearest reference solution in S^*, which is referred to as the generation distance (GD) in the literature [4], [5], [42]. While the GD can only evaluate the proximity of the solution set S_j to S^*, $\mathrm{D1_R}(S_j)$ can evaluate the distribution of S_j as well as the proximity of S_j to S^*. See Czyzak and Jaszkiewicz [43] for characteristic features of the $\mathrm{D1_R}$ measure.

In any multiobjective optimization problem, it is reasonable for the decision maker (DM) to choose a final single solution $\mathbf{x}^*$ from the Pareto-optimal solution set. The final solution $\mathbf{x}^*$ is the best solution with respect to the DM's preference. When the true Pareto-optimal solution set is not given, the DM will choose a final solution $\mathbf{x}$ from an available solution set S_j. When S_j is a good approximation of the true Pareto-optimal solution set, the chosen solution $\mathbf{x}$ may be close to the best solution $\mathbf{x}^*$. In this case, the loss due to the choice of $\mathbf{x}$ instead of $\mathbf{x}^*$ can be approximately measured by the distance between $\mathbf{x}$ and $\mathbf{x}^*$ in the objective space. Since $\mathbf{x}$ and $\mathbf{x}^*$ are unknown, we cannot directly measure the distance between $\mathbf{x}$ and $\mathbf{x}^*$. The expected value of the distance, however, can be roughly estimated by the average value of the distance from each Pareto-optimal solution to its nearest available solution. The $\mathrm{D1_R}$ measure corresponds to this approximation. In addition to the $\mathrm{D1_R}$ measure, we also use the following performance measures for evaluating the solution set S_j.

Let *S* be the union of the *J* solution sets (i.e., $S = S_1 \cup \cdots \cup S_J$). A straightforward performance measure of the solution set S_j with respect to the *J* solution sets is the ratio of solutions in S_j that are not dominated by any other solutions in *S*. This measure is written as follows:

$$R_{\mathrm{NDS}}(S_j) = \frac{|\, S_j - \{\mathbf{x} \in S_j \mid \exists \mathbf{y} \in S : \mathbf{y} \prec \mathbf{x}\} \,|}{|\, S_j \,|}, \tag{7}$$

where $\mathbf{y} \prec \mathbf{x}$ means that the solution $\mathbf{x}$ is dominated by the solution $\mathbf{y}$. In the numerator of (7), dominated solutions $\mathbf{x}$ by other solutions $\mathbf{y}$ in *S* are removed from the solution set S_j. The higher the ratio $R_{\mathrm{NDS}}(S_j)$ is, the better the solution set S_j is. In some computation experiments of this paper, we also use the number of obtained solutions (i.e., $|\, S_j \,|$) as a performance measure.

The reference solution set S^* of each test problem was found using the SPEA [10], the NSGA-II [13], and our MOGLS (i.e., the modified MOGLS in Subsection II.B). Each algorithm was applied to each test problem with much longer computation time and larger memory storage than the other computational experiments in this paper. More specifically, we used the following parameter specifications in all the three algorithms for finding the reference solution set of each test problem:

Population size (N_{pop}): 200,

Crossover probability: 0.9,

Mutation probability per string: 0.6,

Stopping conditions: Evaluation of 5 000 000 solutions.

In the SPEA, the size of the secondary population was specified as 200. In our MOGLS, we used the following parameter specifications:

Number of elite solutions (N_{elite}): 10,

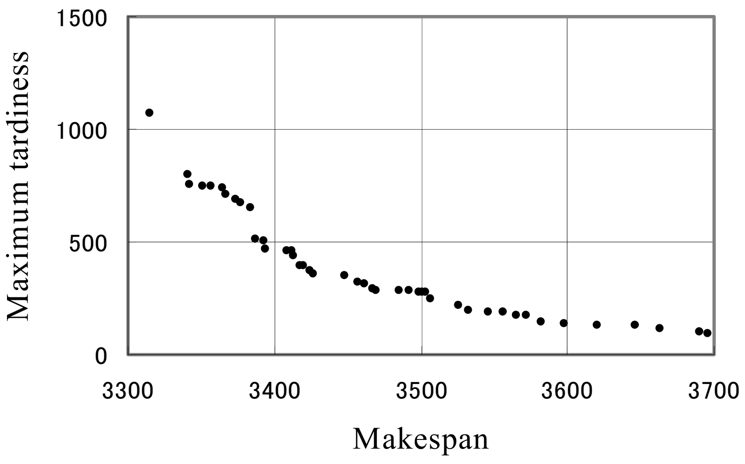
Number of neighbors to be examined (k): 2,

Tournament size in the selection of initial solutions: 5,

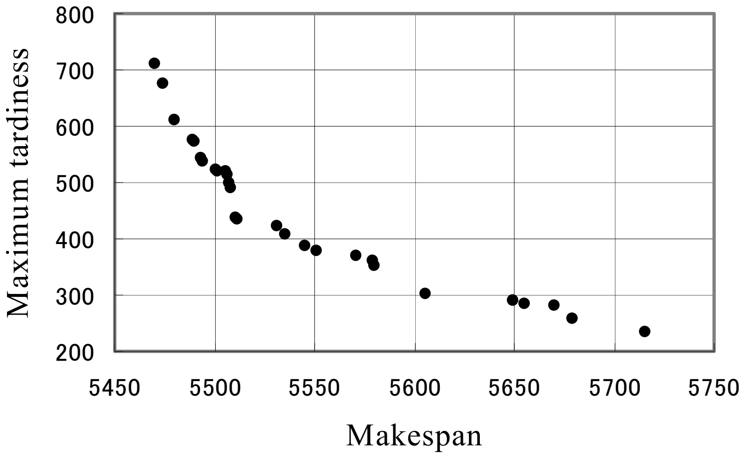
Local search probability (p_{LS}): 0.8.

The computation load in the search for reference solutions was 50 times as much as the other computational experiments in this paper where the stopping condition was the evaluation of 100 000 solutions. We used the two-point crossover and the insertion mutation in all the three algorithms. The insertion mutation was also used in local search of our MOGLS. The above parameter values were specified from preliminary computational experiments on the two-objective 40-job test problem (i.e., 2/40 problem). One may think that the value of *k* is too small. The effect of k and p_{LS} on the performance of our MOGLS will be discussed in Section III where $k = 2$ and $p_{\text{LS}} = 0.8$ are shown to be one of their good combinations. In computational experiments on multiobjective knapsack problems by Knowles and Corne [24], the value of *k* (i.e., *l_fails* in their notation) was specified as 5 for the M-PAES [23] and the MOGLS of Jaszkiewicz [22]. The effect of the other parameters on the performance of each algorithm will be discussed later in this paper.

We chose only non-dominated solutions as reference solutions from 30 solution sets obtained by 10 runs of the three algorithms for each test problem. We show the obtained reference solution sets for the two-objective 40-job and 80-job test problems in Fig. 9 (a) and Fig. 9 (b), respectively. We can observe the existence of a clear tradeoff between the two objectives in each figure. We can also see that the obtained reference solution set for each test problem has a good distribution (i.e., somewhat similar to a uniform distribution) on the tradeoff front in the objective space.



(a) Two-objective 40-job test problem.



(b) Two-objective 80-job test problem.

Fig. 9 Reference solutions obtained from the three EMO algorithms.

The number of the obtained reference solutions for each test problem is summarized in Table 1 for the two-objective problems and Table 2 for the three-objective problems. In these tables, we also show the width of the range of each objective where $f_1(\mathbf{x})$, $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ are the makespan, the maximum tardiness and the total flow time, respectively. The width of the range of the *i*-th objective $f_i(\cdot)$ over the reference solution set S^* is defined as

$$width_{f_i}(S^*) = \max\{f_i(\mathbf{y}) \mid \mathbf{y} \in S^*\} - \min\{f_i(\mathbf{y}) \mid \mathbf{y} \in S^*\}. \quad (8)$$

From the comparison between Table 1 and Table 2, we can see that much more reference solutions were obtained for the three-objective problems than the two-objective problems. We can also see that the reference solutions of each test problem locate over the wide range of each objective except for the case of the two-objective 80-job test problem (i.e., 2/80 in Table 1). In this case, it seems that the three algorithms did not find extreme solutions with very good values of one objective and poor values of the other objective. As we have already mentioned, we applied the three algorithms to each test problem 10 times (i.e., 10 runs). In each run, five million solutions were examined. This means that 150 million solutions were examined for each test problem in total. Thus we did not further perform the search for reference solutions.

Table 1 The number of obtained reference solutions for the two-objective test problems and the width of their range for each objective.

Test problem	# of solutions	Width of the range	
		$f_1(\mathbf{x})$	$f_2(\mathbf{x})$
2/20	38	284	834
2/40	44	381	978
2/60	54	473	2632
2/80	28	245	478

Table 2 The number of obtained reference solutions for the three-objective test problems and the width of their range for each objective.

Test problem	# of solutions	Width of the range		
		$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
3/20	548	351	1032	4115
3/40	580	446	1916	10663
3/60	381	507	3298	19309
3/80	508	463	4262	32105

The objective space of each test problem was normalized so that the minimum and maximum values of each objective among the reference solutions were 0 and 100, respectively. For example, the rectangle $[3315, 3696] \times [97, 1075]$ specified by the reference solutions in Fig. 9 (a) was normalized into the square $[0,\ 100] \times [0,\ 100]$. Using the normalized objective space, the $\mathrm{D1_R}$ measure is calculated.

D. Effect of Modification of the Local Search Part

For examining the effect of the modification of the local search part in Subsection II.B, we applied the former MOGLS [20], [21] and the modified MOGLS to the eight test problems using the following parameter specifications:

Population size (N_{pop}): 60,

Stopping conditions: Evaluation of 100 000 solutions.

The other parameter values were the same as those in Subsection II.C for finding the reference solution set of each test problem.

Each algorithm was applied to each test problem 20 times (i.e., 20 runs) using different initial populations. Multiple solutions were simultaneously obtained from a single run of each algorithm. In Fig. 10, we show 20 solution sets obtained from each algorithm for the two-objective 40-job test problem. We can see from Fig. 10 that all solutions obtained from the former algorithm (i.e., open circles) are dominated by many solutions from the modified one (i.e., closed circles). We can also see that no solutions from the modified algorithm are dominated by any solutions from the former one.

For each of 20 runs of the two algorithms for each test problem, we calculated the ratio of non-dominated solutions (i.e., $R_{\mathrm{NDS}}(\cdot)$) for the solution set S_{F} from the former algorithm and the solution set S_{M} from the modified one by specifying S in (7) as $S = S_{\mathrm{F}} \cup S_{\mathrm{M}}$. Then we calculated the average value of $R_{\mathrm{NDS}}(\cdot)$ over 20 runs. For all the eight test problems, we obtained the following average results: $R_{\mathrm{NDS}}(S_{\mathrm{F}}) = 0$ and $R_{\mathrm{NDS}}(S_{\mathrm{M}}) = 1$. These results show that all solutions obtained from the former algorithm were dominated by solutions from the modified one. Moreover no solutions from the modified algorithm were dominated by any solutions from the former one. That is, the modified algorithm clearly outperformed the former one for all the eight test problems as visually shown in Fig. 10 for the two-objective 40-job test problem.

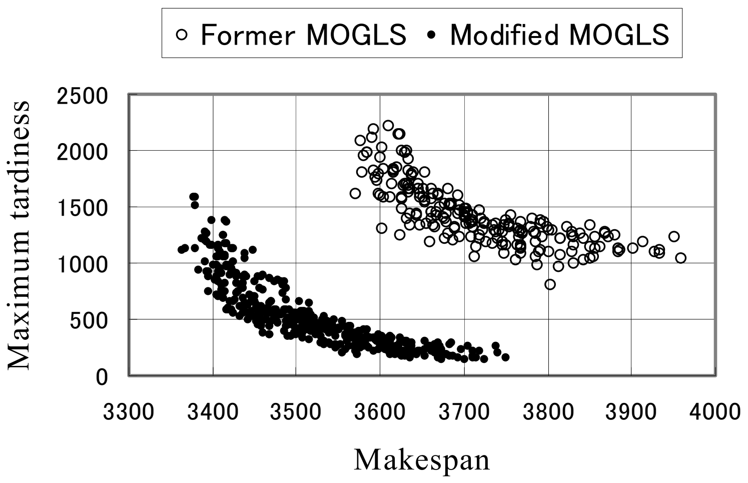


Fig. 10 Comparison between the former MOGLS and its modified version. All solutions obtained by 20 runs of each algorithm for the two-objective 40-job test problem are shown.

In Subsection II.B, we explained the motivation for modifying the former MOGLS using Fig. 5. More specifically, we pointed out the possibility that the genetic operations generate an inappropriate initial solution for the current weight vector (e.g., B and C in Fig. 5). For examining the validity of this motivation, we measured the distance between two parents of each solution in the normalized objective space during each of 20 runs of the former MOGLS for each test problem. We also measured the distance between each solution and its nearest parent. As we have already mentioned, we used the roulette wheel selection in (4) for parent selection in the former MOGLS. For comparison, we also examined the use of the tournament selection of the tournament size 2, 5 and 10 instead of the roulette wheel selection. Moreover, the use of the random selection from the best 10%, 20% and 50% solutions of the current population was also examined. Jaszkiewicz [22] used the latter selection scheme for parent selection.

Average results with respect to the distance between two parents are summarized in Table 3. From this table, we can see that the distance between two parents was much larger in the case of the roulette wheel selection than the other selection schemes. This observation means that dissimilar parents (e.g., ***c*** and ***d*** in Fig. 5) were often selected in the execution of the former MOGLS with the roulette wheel selection. The dissimilarity of parents may be the main cause of the poor performance of the former MOGLS. When we used the other selection schemes with higher selection pressure for parent selection, similar parents were selected more frequently as shown in Table 3.

Table 3 Average distance between two parents of each solution in the normalized objective space of each test problem.

Test	Roulette	Tournament			Best solutions		
problem	wheel	2	5	10	10%	20%	50%
2/20	49.6	15.5	11.1	3.0	6.9	11.8	20.4
2/40	47.8	18.0	8.8	2.0	5.3	11.5	20.1
2/60	31.5	9.2	5.3	1.8	3.2	5.7	11.5
2/80	109.4	22.6	12.2	3.9	9.8	13.9	31.0
3/20	56.8	13.4	10.3	4.2	8.8	7.2	23.7
3/40	43.2	8.4	6.6	3.0	5.3	5.3	17.4
3/60	39.3	6.4	5.0	2.4	4.1	4.2	14.1
3/80	40.4	5.8	3.8	2.0	3.6	3.4	14.3

Average results with respect to the distance between each solution and its nearest parent are summarized in Table 4. From the comparison between Table 4 and the second column of Table 3, we can see that each solution was similar to its nearest parent in all the seven MOGLS variants for all the eight test problems. This observation together with the above-mentioned observation on Table 3 suggests that good initial solutions (e.g., A in Fig. 5) were often generated from good parents with high similarity (e.g., ***a*** and ***b*** in Fig. 5) in the six variants with the tournament selection and the random selection from the best solutions. Thus we expect that the improvement of the former MOGLS would be achieved by the use of those selection schemes for parent selection.

Table 4 Average distance between each solution and its nearest parent in the normalized objective space of each test problem.

Test	Roulette	Tournament			Best solutions		
problem	wheel	2	5	10	10%	20%	50%
2/20	16.2	23.4	23.3	21.8	22.7	23.6	22.1
2/40	13.5	22.4	21.3	20.2	20.6	21.3	21.0
2/60	9.9	12.5	12.6	12.2	12.4	12.6	12.4
2/80	27.5	52.1	54.3	52.6	52.6	52.7	52.2
3/20	18.7	21.8	21.8	20.2	21.1	21.3	22.0
3/40	14.4	13.5	13.4	12.9	13.4	13.2	14.9
3/60	12.9	9.7	9.8	9.4	9.6	9.6	11.8
3/80	13.7	9.6	9.6	9.1	9.7	9.8	11.9

Average values of the $D1_R$ measure are summarized in Table 5 where smaller values mean better solution sets. As expected from Table 3 and Table 4, the six variants with the tournament selection and the random selection from the best solutions outperformed its original version with the roulette wheel selection. More specifically, all results by the six variants for the eight test problems in Table 5 are significantly better (i.e., smaller) than the corresponding results by their original version with the 99% confidence level (the Mann-Whitney U test).

Table 5 Performance evaluation of each variant of the former MOGLS using the $D1_R$ measure. Smaller values mean better solution sets.

Test	Roulette	Tournament			Best solutions		
problem	wheel	2	5	10	10%	20%	50%
2/20	21.4	6.8	6.2	6.9	6.8	7.3	8.0
2/40	48.5	17.6	20.5	22.4	20.9	19.6	20.4
2/60	45.7	21.7	21.1	21.6	21.3	23.7	24.6
2/80	267.9	72.7	69.8	72.5	70.8	72.2	76.9
3/20	17.4	10.8	9.5	9.6	9.4	11.1	9.6
3/40	41.3	23.6	24.3	22.8	23.4	26.0	21.5
3/60	58.5	32.1	33.7	28.1	32.8	32.3	30.1
3/80	70.2	39.5	41.3	40.6	40.2	42.4	34.9
Average	71.4	28.4	28.9	29.3	28.2	29.4	28.3

In the same manner as Table 5, we performed computational experiments using the modified MOGLS. The tournament selection with the tournament size five was used for selecting initial solutions for local search in the modified MOGLS. Average values of the $D1_R$ measure are summarized in Table 6 where the seven variants with different selection schemes for parent selection are compared. It is interesting to note that the best results were obtained from the roulette wheel selection in Table 6 on the average (especially for the three-objective test problems) while it was the worst in Table 5. When the roulette wheel was used for parent selection, the improvement by the modification of the local search part from Table 5 to Table 6 is significant for all the eight test problems with the 99% confidence level (the Mann-Whitney U test). On the other hand, the same modification significantly degraded the performance of the other six variants for all the four three-objective test problems with the 99% confidence level. The deterioration in the performance may be due to the negative effect of the selection of initial solutions for local search. When our MOGLS has a parent selection scheme with high selection pressure, the selection of initial solutions for local search makes the overall selection pressure too strong. Too strong selection pressure leads to the decrease in the diversity of solutions (i.e., undesired convergence to a small number of solutions). As a result, the performance of our MOGLS with high selection pressure in the parent selection was deteriorated by the combination with high selection pressure in the selection of initial solutions for local search in computational experiments on the three-objective test problems with many reference solutions.

Table 6 Performance evaluation of each variant of the modified MOGLS using the $D1_R$ measure.

Test	Roulette	Tournament			Best solutions		
problem	wheel	2	5	10	10%	20%	50%
2/20	6.0	7.5	8.3	7.4	7.1	7.5	8.9
2/40	17.8	19.7	20.6	22.0	18.7	19.4	20.3
2/60	22.9	22.8	22.8	22.7	23.1	23.5	25.2
2/80	77.3	77.4	72.1	67.2	72.1	76.3	82.3
3/20	9.3	14.4	14.1	13.5	13.7	15.2	15.8
3/40	21.6	33.9	33.1	30.5	29.3	32.8	34.6
3/60	29.5	37.5	39.1	38.5	38.9	39.9	40.8
3/80	35.7	48.1	48.6	48.9	48.0	51.2	50.7
Average	27.5	32.9	33.0	32.6	31.4	33.3	34.9

Among the 14 variants of the MOGLS in Table 5 and Table 6, good results were obtained by seven variants (i.e., the six variants of the former MOGLS with the tournament selection and the random selection from the best solutions in Table 5 and the modified MOGLS with the roulette wheel selection in Table 6). Hereafter we mainly use the modified MOGLS with the roulette wheel for parent selection (i.e., the second column of Table 6) for examining the balance between genetic search and local search through computational experiments using the local search probability p_{LS}. Multiobjective memetic algorithms with no selection scheme of initial solutions for local search will be examined again in Section IV in the context of the hybridization of popular EMO algorithms.

E. Choice of a Neighborhood Structure

In the above computational experiments, we used the insertion mutation as a local search operation. In this subsection, we examine other local search operations (i.e., other neighborhood structures): exchange of adjacent two jobs, exchange of arbitrary two jobs, and exchange of arbitrary three jobs. The number of neighbors of the current solution (i.e., the size of the neighborhood structure) is $(n-1)$ when we exchange adjacent two jobs for an n-job permutation flowshop scheduling problem. It is ${}_nC_2 = n(n-1)/2$ and $2 \cdot {}_n C_3 = n(n-1)(n-2)/3$ when we exchange arbitrary two and three jobs, respectively. The number of neighbors is $(n-1)^2$ in the case of the insertion operation. It should be noted that these four neighborhood structures are not mutually exclusive. For example, the adjacent two-job exchange neighbors are included in the arbitrary two-job exchange and insertion neighbors. The insertion neighbors partially overlap with the arbitrary two-job and three-job exchange neighbors. Many neighborhood structures were explained in a more general manner in Krasnogor [34].

The performance of the four local search operations was compared using the $\mathrm{D1_R}$ measure. For evaluating each local search operation, the modified MOGLS with the roulette wheel selection was applied to each test problem 20 times in the same manner as the previous computational experiments. The average value of the $\mathrm{D1_R}$ measure over 20 runs is shown together with the standard deviation (in parentheses) in Table 7. We can see from this table that the best (i.e., smallest) results were obtained from the insertion operation for all the eight test problems.

Table 7 Performance evaluation of each algorithm using the $\mathrm{D1_R}$ measure. Standard deviations are shown in parentheses.

Test	Local Search Operation			
problem	Adjacent	Two-job	Three-job	Insertion
2/20	6.9 (1.6)	6.1 (1.0)	6.9 (1.4)	6.0 (1.7)
2/40	26.9 (4.7)	22.4 (4.4)	28.0 (4.5)	17.8 (3.3)
2/60	28.9 (4.4)	24.7 (2.9)	27.1 (2.9)	22.9 (2.7)
2/80	156.9(24.5)	101.1(21.1)	125.1(14.8)	77.3(13.8)
3/20	11.5 (1.8)	10.0 (1.1)	10.1 (1.0)	9.3 (1.4)
3/40	26.3 (2.6)	22.6 (2.5)	25.4 (3.3)	21.6 (2.7)
3/60	37.8 (3.2)	34.1 (4.0)	35.0 (3.0)	29.5 (3.2)
3/80	44.5 (6.8)	38.4 (4.3)	41.2 (6.1)	35.7 (4.1)
Average	42.5 (6.2)	32.4 (5.2)	37.3 (4.6)	27.5 (4.1)

F. Choice of an Acceptance Rule in Local Search

In the local search part of the modified MOGLS, the scalar fitness function in (3) was used for making the decision on the replacement of the current solution with its neighbor. That is, the neighbor was accepted only when it had a better (i.e., smaller) value of the scalar fitness function than the current solution. It is possible to use other acceptance rules in the local search part. In this subsection, we examine three acceptance rules in addition to the scalar fitness function in (3).

One rule is to accept neighbors that are not dominated by the current solution. Let us consider Fig. 11 where the current solution and its neighbors are denoted by a closed circle (i.e., A) and open circles (i.e., B, C, D, E, F and G), respectively. The current solution A can move to the five neighbors except for G because only G is dominated by A. A drawback of this acceptance rule is that the current solution can be degraded by multiple moves. For example, the current solution A can move to the neighbor B, from which the current solution can further move to G. Another acceptance rule is to accept only better neighbors that dominate the current solution. In this case, the current solution A can move only to the neighbor D in Fig. 11. A drawback of this acceptance rule is that the movable area is very small especially when the number of objectives is large.

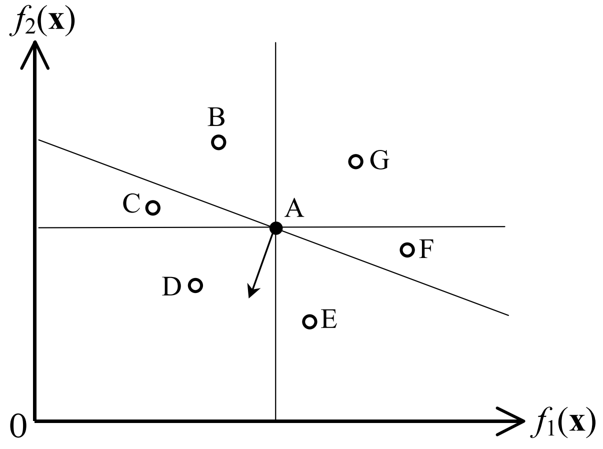


Fig. 11 Illustration of each acceptance rule.

The other acceptance rule is the use of the pseudo-weight vector [4]. The pseudo-weight w_i for the i-th objective is defined for the current solution $\mathbf{x}$ as

$$w_i = \frac{f_i^{\max} - f_i(\mathbf{x})}{f_i^{\max} - f_i^{\min}} \Bigg/ \sum_{j=1}^{N} \frac{f_j^{\max} - f_j(\mathbf{x})}{f_j^{\max} - f_j^{\min}}, \quad i = 1,2,\ldots,N, \tag{9}$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum values of the *i*-th objective in the current population, respectively. The scalar fitness function with the pseudo-weight vector $\mathbf{w} = (w_1, ..., w_N)$ determined by (9) is used in the third acceptance rule. Let us assume in Fig. 11 that the arrow shows the weight vector $-\mathbf{w}$ and the inclined line is orthogonal with this arrow. In this case, the current solution A can move to the three neighbors C, D and E. The determination of the weight vector by (9) is illustrated in Fig. 12 where all solutions in the current population are shown by open circles. The arrow attached to each open circle shows the weight vector $-\mathbf{w}$ for the corresponding solution. From this figure, we can see that an appropriate weight vector is assigned to each solution by (9). Note that each arrow in Fig. 12 is not the exact direction of the move by local search. Since we use the first improvement strategy for combinatorial optimization problems with discrete search spaces, the move by local search is not the same as the direction of the weight vector $-\mathbf{w}$. For example, A in Fig. 11 will move to the first examined neighbor among C, D and E. It should be noted that the local search direction specified by the weight vector $-\mathbf{w}$ in the objective space is a totally different concept from the local search direction in the continuous decision space (e.g., see Salomon [44]).

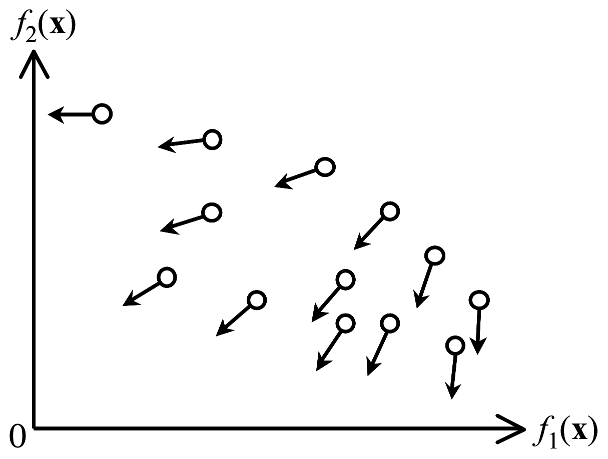


Fig. 12 Pseudo-weight vector.

In the calculation of the pseudo-weight vector for each solution, we need the maximum and minimum values of each objective over the current population. Thus this approach has some computational overhead. The overhead, however, is not large because the maximum and minimum values are calculated just once for the current population in each generation. The calculated values are used for all solutions in the current population. Moreover, the pseudo-weight vector is calculated only for each initial solution of local search (i.e., the pseudo-weight vector is not updated unless local search restarts from a new initial solution). A possible drawback of this approach is that the distribution of weight vectors directly depends on the distribution of solutions in the objective space. Thus the distribution of weight vectors is not uniform when the distribution of solutions in the current population is not uniform. For example, similar weight vectors are assigned to many solutions when they are closely located in the objective space.

We compared the three acceptance rules using the modified MOGLS with the insertion neighborhood. The same parameter values as in Subsection II.D were used. Since the three acceptance rules do not have any selection mechanism of initial solutions, we chose an initial solution in the same manner as the modified MOGLS. Then local search with each acceptance rule was applied to the selected initial solution with the local search probability. Average results over 20 runs with each acceptance rule are summarized in Table 8. We also show average results by the modified MOGLS in the same table. From this table, we can see that almost the same results were obtained from the modified MOGLS and the pseudo-weight approach. This is because these two approaches are based on the scalar fitness function. We can also see that the performance of the first two approaches based on the dominance relation were not bad for many cases while they were outperformed by the other approaches based on the scalar fitness function for all the eight test problems (small values in Table 8 mean better solution sets). That is, the above-mentioned drawbacks of the acceptance rules based on the dominance relation were not clear in Table 8. This is because the value of k (i.e., the maximum number of examined neighbors of the current solution) was very small (i.e., $k = 2$). We also performed the same computational experiments by specifying the value of k and p_{LS} as $k = 100$ and $p_{\mathrm{LS}} = 0.02$. Average results over 20 runs are summarized in Table 9. While good results were still obtained from the two approaches based on the scalar fitness function in the last two columns of Table 9, the performance of the first acceptance rule based on the dominance relation was severely deteriorated for all the eight test problems as shown in the second column of Table 9. The drawback of this acceptance rule (i.e., possible deterioration of the current solution by multiple moves) became clear by increasing the value of k in Table 9. The performance of the second acceptance rule based on the dominance relation (i.e., move to better solutions) was slightly deteriorated by increasing the value of k from Table 8 to Table 9 (see the third column of these tables).

Table 8 Comparison among the four acceptance rules using the $D1_R$ measure for the case of $k = 2$ and $p_{LS} = 0.8$.

Test	Acceptance Rule			
problem	Non-D	Better	Pseudo	MOGLS
2/20	6.7 (1.1)	7.2 (2.1)	6.4 (1.5)	6.0 (1.7)
2/40	23.5 (2.0)	21.4 (3.4)	18.3 (2.8)	17.8 (3.3)
2/60	28.6 (4.1)	26.0 (3.6)	23.9 (2.9)	22.9 (2.7)
2/80	116.0(21.0)	91.6(16.5)	82.9(21.3)	77.3(13.8)
3/20	9.6 (1.1)	11.1 (1.6)	9.8 (1.2)	9.3 (1.4)
3/40	30.2 (4.0)	23.3 (3.5)	23.0 (2.3)	21.6 (2.7)
3/60	42.9 (4.2)	32.4 (4.5)	30.1 (4.1)	29.5 (3.2)
3/80	50.4 (6.0)	38.4 (4.2)	36.6 (3.5)	35.7 (4.1)
Average	38.5 (5.5)	31.4 (4.9)	28.9 (5.0)	27.5 (4.1)

Table 9 Comparison among the four acceptance rules using the $D1_R$ measure for the case of $k = 100$ and $p_{LS} = 0.02$.

Test	Acceptance Rule			
problem	Non-D	Better	Pseudo	MOGLS
2/20	92.1(13.4)	6.8 (1.3)	7.3 (1.9)	4.4 (0.7)
2/40	163.9(15.0)	20.3 (3.1)	16.6 (3.9)	19.2 (3.1)
2/60	137.8(13.1)	25.8 (3.3)	21.9 (3.4)	20.1 (1.6)
2/80	699.6(75.6)	101.7(24.3)	67.2(12.6)	69.5 (8.8)
3/20	108.7(12.0)	11.5 (1.8)	10.9 (1.9)	7.8 (1.0)
3/40	142.2(16.3)	24.6 (2.6)	24.4 (2.4)	20.3 (1.9)
3/60	145.3(15.6)	33.4 (4.3)	32.3 (3.5)	26.8 (2.7)
3/80	175.0(21.0)	39.5 (3.9)	38.5 (3.4)	31.2 (3.2)
Average	208.1(22.8)	33.0 (5.6)	27.4 (4.1)	24.9 (2.9)

III. BALANCE BETWEEN GENETIC AND LOCAL SEARCH

In this section, we examine the effect of the balance between genetic search and local search on the search ability of our MOGLS (i.e., the modified MOGLS in Subsection II.B). The problem is how to allocate the available computation time wisely between genetic search and local search. This problem has been studied in the field of single-objective hybrid (i.e., memetic) algorithms [45]. For example, Orvosh and David [46] reported that the best results in their computational experiments were obtained from their memetic algorithm when individuals were improved by local search with a probability 0.05 (i.e., when the local search probability p_{LS} was specified as $p_{\mathrm{LS}} = 0.05$). Goldberg and Voessner [45] presented a theoretical framework for discussing the balance between genetic search and local search. Hart [47] investigated the following four questions for designing efficient memetic algorithms for continuous optimization:

(a) How often should local search be applied?

(b) On which solutions should local search be used?

(c) How long should local search be run?

(d) How efficient does local search need to be?

The first and second questions are related to the local search probability p_{LS} and the local search application interval T while the third question is related to the parameter k (i.e., the maximum number of examined neighbors of the current solution) in our MOGLS. Hart's study was extended to the case of combinatorial optimization by Land [48] where the balance between genetic search and local search was referred to as the local/global ratio. The balance can be also adjusted by the use of different neighborhood structures. Krasnogor [34] investigated how to change the size and the type of neighborhood structures dynamically in the framework of multimeme memetic algorithms where each meme had a different neighborhood structure, a different acceptance rule and a different number of iterations of local search.

All the above-mentioned studies investigated the balance between local search and genetic search for single-objective optimization. Since the aim of EMO algorithms is not to find a single final solution but to simultaneously find a variety of Pareto-optimal (or near Pareto-optimal) solutions, an appropriate balance for multiobjective optimization may be different from the case of single-objective optimization. For example, the diversity of solutions in the final generation is very important in multiobjective optimization while it is usually not important in single-objective optimization. Thus more emphasis should be placed on the maintenance of the diversity of solutions in each generation in the case of multiobjective optimization than single-objective optimization. In this section, we examine the balance between local search and genetic search using the three parameters (i.e., k, p_{LS} and T) in the local search part of our MOGLS. We also examine the necessity of genetic search using the crossover probability p_{C} and the mutation probability p_{M}.

A. Effect of Local Search

For examining the effect of local search on the search ability of our MOGLS, we performed computational experiments using various specifications of k and p_{LS}. More specifically, we examined 132 combinations of 11 values of *k* (i.e., $k = 1, 2, 4, 6, 8, 10, 20, 40, 60, 80, 100$) and 12 values of p_{LS} (i.e., $p_{\mathrm{LS}} = 0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0$). Using each combination of *k* and p_{LS}, our MOGLS was applied to each test problem 20 times in the same manner as Subsection II.D under the same stopping condition (i.e., evaluation of 100 000 solutions). The average value of the $\mathrm{D1_R}$ measure obtained from each combination of *k* and p_{LS} is shown in Fig. 13 for the two-objective 80-job test problem where shorter bars mean better solution sets. In this figure, we can observe a valley from the left-bottom corner to the right-top corner in the k - p_{LS} plane. That is, good results were obtained from combinations of *k* and p_{LS} that approximately satisfy the relation $k \cdot p_{\mathrm{LS}} = 1 \sim 10$. When the value of $k \cdot p_{\mathrm{LS}}$ was too small (i.e., the left-top corner), the search in our MOGLS was mainly driven by genetic operations. Thus the search ability of local search was not utilized well in our MOGLS. On the other hand, when the value of $k \cdot p_{\mathrm{LS}}$ was too large (i.e., the right-bottom corner), almost all computation time was spent by local search. Thus the search ability of genetic algorithms was not utilized well.

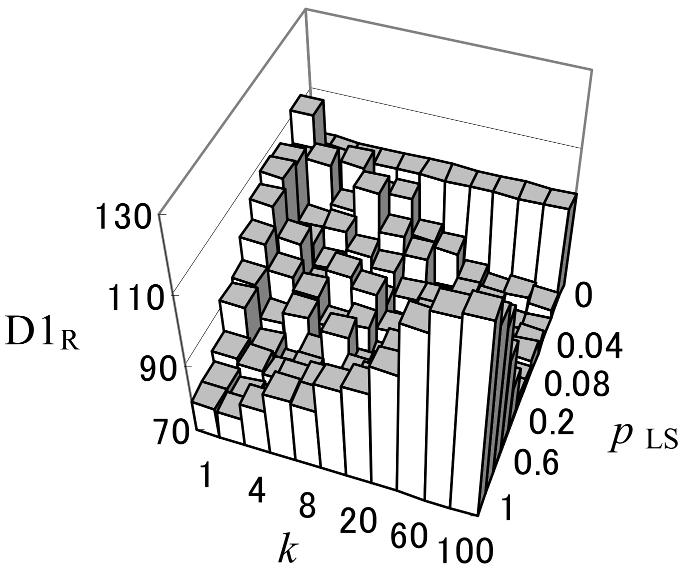


Fig. 13 Average value of the $\mathrm{D1_R}$ measure for the two-objective 80-job problem. Shorter bars mean better solution sets.

The best (i.e., smallest) average value of the $\mathrm{D1_R}$ measure was obtained from the combination of $k = 80$ and $p_{\mathrm{LS}} = 0.02$ as 67.2. The worst average value was obtained from the combination of $k = 100$ and $p_{\mathrm{LS}} = 1$ as 125.7. When $p_{\mathrm{LS}} = 0$, local search was applied to no solutions. Thus the value of k has no effect on the performance of our MOGLS as shown by the flat region corresponding to $p_{\mathrm{LS}} = 0$ in Fig. 13 (i.e., the top-most row). In this case, the average value of the $\mathrm{D1_R}$ measure was 97.8. We further examined solution sets obtained from these three specifications: $(k,\ p_{\mathrm{LS}}) = (80,\ 0.02)$, $(100,\ 1)$ and $p_{\mathrm{LS}} = 0$. In addition to the $\mathrm{D1_R}$ measure, we also calculated the ratio of non-dominated solutions (i.e., $R_{\mathrm{NDS}}(S_j)$) and the number of obtained solutions (i.e., $|\,S_j\,|$) for each run of our MOGLS using the three parameter specifications. Average results over 20 runs for each parameter specification are summarized in Table 10 together with standard deviations (in parentheses). In Table 10, we also show the average number of generations updated by the EMO part. When $k = 80$ and $p_{\mathrm{LS}} = 0.02$, the average number of obtained solutions was 12.7. Among those solutions, 90% were not dominated by any other solutions in each run. The EMO part of our MOGLS was iterated for 358 generations on the average. On the other hand, the EMO part was iterated for only a few generations when $k = 100$ and $p_{\mathrm{LS}} = 1$. In this case, the average number of obtained solutions was small (i.e., 9.0) and the quality of each solution was not good. Actually all the obtained solutions from this combination of k and p_{LS} were dominated by other solutions (i.e., the average ratio of non-dominated solutions was 0 in Table 10). That is, both the diversity of solutions and the convergence speed to the Pareto-front were degraded by the use of large values of k and p_{LS} in Table 10. When the local search probability p_{LS} was specified as $p_{\mathrm{LS}} = 0$, local search was not applied to any solutions. In this case, the quality of each solution was not high while the average number of obtained solutions was large. Actually only 23% of obtained solutions were not dominated by any other solutions in each run.

Table 10 Comparison of the three cases of $(k,\ p_{\mathrm{LS}})$ for the two-objective 80-job problem. Average values over 20 runs are shown together with standard deviations in parentheses. Larger values of $R_{\mathrm{NDS}}(S_j)$ and $|S_j|$ mean better solution sets while smaller values of $\mathrm{D1_R}$ mean better solution sets.

Measure	Specification of $(k,\ p_{\mathrm{LS}})$		
	(80, 0.02)	(100, 1)	$p_{\mathrm{LS}} = 0$
$\mathrm{D1_R}$	67.2 (12.1)	125.7 (9.3)	97.8 (21.1)
$R_{\mathrm{NDS}}(S_i)$	0.90 (0.22)	0.00 (0.00)	0.23 (0.33)
$\vert S_i \vert$	12.7 (3.3)	9.0 (2.0)	14.4 (4.7)
Generations	358 (38.0)	3.9 (0.3)	1667 (0.0)

For all the eight test problems, we observed the improvement in the $\mathrm{D1_R}$ measure by the hybridization with local search when the values of k and p_{LS} were appropriate. We also observed the negative effect of the hybridization with local search for all the eight test problems when both k and p_{LS} were large (i.e., the right-bottom corner of Fig. 13). The negative effect, however, was small for small-size test problems. For example, we show the average value of the $\mathrm{D1_R}$ measure for the two-objective 20-job test problem in Fig. 14 where the deterioration in the $\mathrm{D1_R}$ measure at the right-bottom corner is not clear. The best result in Fig. 14 was obtained from the combination of $k = 100$ and $p_{\mathrm{LS}} = 0.02$. In the same manner as Table 10, we compare the three specifications: $(k,\ p_{\mathrm{LS}}) = (100,\ 0.02)$, (100, 1) and $p_{\mathrm{LS}} = 0$ in Table 11. From this table, we can see that the performance deterioration by the use of large values of k and p_{LS} was small for the two-objective 20-job test problem (i.e., the negative effect of the hybridization with local search was small). This may be because the number of examined solutions (i.e., 100 000 solutions) during the execution of our MOGLS was large relative to the problem size in the case of the two-objective 20-job test problem. On the other hand, the positive effect of the hybridization with local search was still clear for small-size test problems as shown in Table 11.

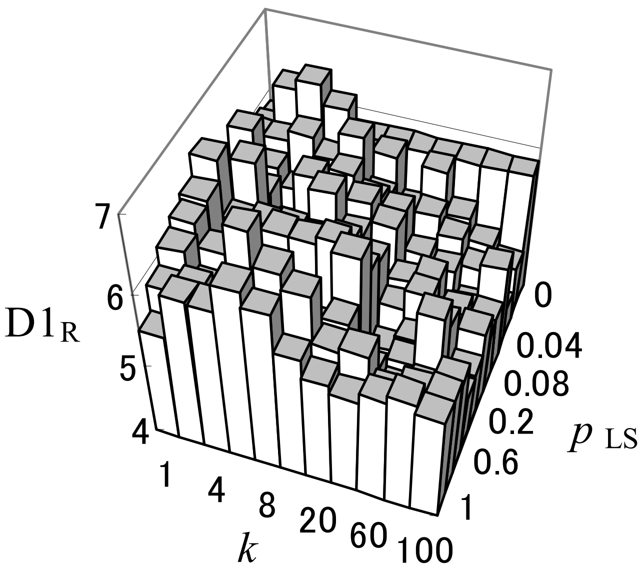


Fig. 14 Average value of the $\mathrm{D1_R}$ measure for the two-objective 20-job problem.

Table 11 Comparison of the three specifications of $(k,\ p_{\mathrm{LS}})$ for the two-objective 20-job problem.

Measure	Specification of $(k,\ p_{\mathrm{LS}})$		
	(100, 0.02)	(100, 1)	$p_{\mathrm{LS}}=0$
$\mathrm{D1_R}$	4.4 (0.65)	5.4 (0.92)	5.8 (1.34)
$R_{\mathrm{NDS}}(S_j)$	0.68 (0.13)	0.50 (0.17)	0.44 (0.15)
$\lvert S_j \rvert$	22.3 (3.2)	19.7 (2.4)	21.1 (3.4)
Generations	358 (16.5)	9.3 (0.4)	1667 (0.0)

We further examined the positive and negative effects of the hybridization with local search for the other test problems using the $\mathrm{D1_R}$ measure. Average results over 20 runs are summarized in Table 12 where standard deviations are shown in parentheses. In this table, the second column labeled as “Tuned” shows the results obtained from the best combination of k and p_{LS} for each test problem (e.g., $k=80$ and $p_{\mathrm{LS}}=0.02$ for the 2/80 problem). In this table, we can observe both the positive and negative effects in all the eight test problems while their strength depends on the problem.

Table 12 Effect of the parameter values in the local search part on the $D1_R$ measure.

Test problem	Specification of (k, p_{LS})		
	Tuned	(100, 1)	$p_{LS} = 0$
2/20	4.4 (0.7)	5.4 (0.9)	5.8 (1.3)
2/40	17.3 (3.1)	26.7 (1.7)	22.9 (5.3)
2/60	19.3 (1.8)	27.3 (2.5)	23.8 (2.6)
2/80	67.2(12.1)	125.7 (9.3)	97.8(21.1)
3/20	7.7 (1.1)	9.7 (1.2)	8.7 (0.9)
3/40	19.6 (2.4)	26.2 (3.0)	21.5 (2.1)
3/60	25.4 (2.9)	35.5 (3.1)	31.5 (4.6)
3/80	31.2 (3.2)	47.1 (5.5)	35.8 (4.4)

From the above experimental results, one may think that the negative effect of the hybridization with local search can be reduced by the increase in computation load. This may be the case for all test problems. We need, however, much more computation load for large test problems because the size of the search space exponentially increases with the number of jobs (i.e., $n!$ for n-job problems). We performed computational experiments with more computation load (i.e., evaluation of 500 000 solutions) for the two-objective 80-job test problem in the same manner as Fig. 13. In experimental results, we still observed a clear negative effect of the hybridization with local search when both k and p_{LS} were large as in Fig. 13.

In the above computational experiments, we adjusted the balance between genetic search and local search using the two parameters k and p_{LS}. We can also adjust the balance by invoking the local search part every T generations (not every generation). When the local search part is invoked, we still use the local search probability p_{LS}. Thus the overall local search probability can be viewed as p_{LS}/T over the whole execution of the MOGLS. The local search application interval T was implicitly assumed as $T = 1$ in all the above computational experiments.

In the same manner as Fig. 13, we examined 132 combinations of p_{LS} and T (i.e., $p_{LS} = 0$, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0 and $T = 1$, 2, 4, 6, 8, 10, 20, 40, 60, 80, 100) for the two-objective 80-job problem. The value of k was fixed as $k = 80$, which was the value of k in the best combination of k and p_{LS} in Fig. 13. While we examined various values of T, we did not observe any improvement in the $\mathrm{D1_R}$ measure by the specification of T as $T > 1$. That is, we obtained the best result from $T = 1$. We also examined the effect of T for the other test problems in the same manner. The best results were obtained from $T = 1$ for all the eight test problems. This may be because the selection of initial solutions for local search plays a very important role in our MOGLS as shown in Table 5 and Table 6. We will further examine the effect of T in the context of the hybridization of other EMO algorithms with local search in Section IV.

B. Effect of Genetic Search

For examining the effect of the crossover probability p_{C} and the mutation probability p_{M} on the performance of our MOGLS, we performed computational experiments using 121 combinations of 11 values of p_{C} and p_{M} (i.e., $p_{\mathrm{C}} = 0.0$, 0.1, ..., 1.0 and $p_{\mathrm{M}} = 0.0$, 0.1, ..., 1.0). When $p_{\mathrm{C}} = 0.0$ and $p_{\mathrm{M}} = 0.0$, the evolution is driven by local search and selection. In this case, our MOGLS can be viewed as a population-based multiobjective local search algorithm. Using the best parameter values in Fig. 13 for the local search part (i.e., $k = 80$, $p_{\mathrm{LS}} = 0.02$ and $T = 1$), we applied our MOGLS with each combination of p_{C} and p_{M} to the two-objective 80-job test problem 20 times. The other parameter values were the same as the above-mentioned computational experiments. Average results over 20 runs are summarized in Fig. 15 where the performance of the MOGLS is evaluated using the $\mathrm{D1_R}$ measure as in Fig. 13. From Fig. 15, we can see that the performance of the MOGLS was less sensitive to p_{C} and p_{M} than k and p_{LS} (compare Fig. 15 with Fig. 13).

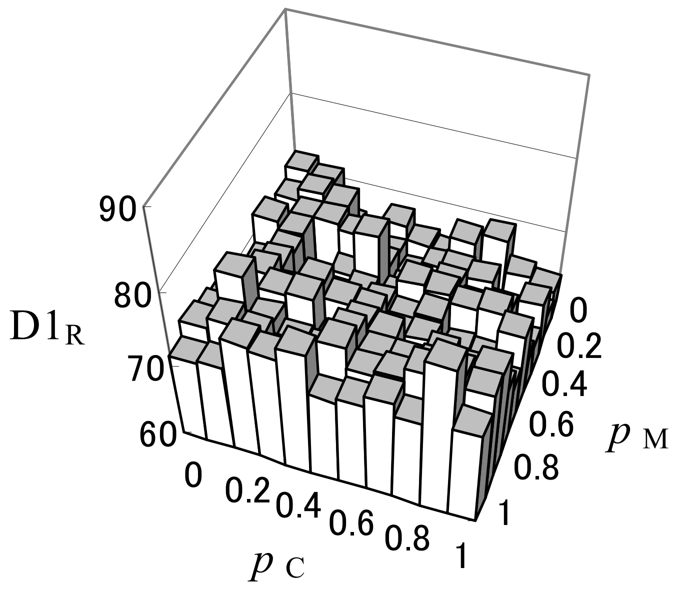


Fig. 15 Average value of the $D1_R$ measure for the two-objective 80-job problem by our MOGLS with various specifications of the crossover probability p_C and the mutation probability p_M.

In Fig. 15, the best (i.e., smallest) average result 61.6 was obtained from $p_C = 0.9$ and $p_M = 0.1$ among the 121 combinations of p_C and p_M. When the crossover probability p_C was specified as $p_C = 0$ (i.e., no crossover: the left-most row of Fig. 15), the best average result 68.6 was obtained from $p_M = 0.2$. On the other hand, the best average result 63.8 was obtained from $p_C = 1.0$ when the mutation probability p_M was specified as $p_M = 0$ (i.e., no mutation: the top-most row). Furthermore, the average result was 72.3 in the case of $p_C = 0$ and $p_M = 0$ (no genetic search: the left-top corner). These four cases are compared in Table 13. From this table, we can see that the crossover and the mutation improved the search ability of our MOGLS. When we did not use the genetic operations, the average number of obtained solutions was small (i.e., 8.2). Moreover, only 26% of them were not dominated by other solutions in each run on the average. In Table 13, the crossover seems to be more important than the mutation because better results were obtained from our MOGLS with only the crossover operation than that with only the mutation operation.

Table 13 Comparison of the four cases with respect to the parameter specifications in genetic search for the two-objective 80-job test problem. Larger values of $R_{\mathrm{NDS}}(S_j)$ and $|S_j|$ mean better solution sets while smaller values of $\mathrm{D1_R}$ mean better solution sets.

Measure	Specification of (p_C, p_M)			
	(0.9, 0.1)	(0, 0.2)	(1.0, 0)	(0, 0)
$\mathrm{D1_R}$	61.6(10.5)	68.6 (9.6)	63.8(12.0)	72.3(14.9)
$R_{\mathrm{NDS}}(S_i)$	0.53(0.36)	0.28(0.23)	0.52(0.31)	0.26(0.23)
$\lvert S_i \rvert$	10.2 (3.5)	11.0 (3.4)	11.1 (5.3)	8.2 (2.6)
Generations	410(68.4)	461(44.9)	443(45.5)	518(40.2)

In the same manner as Table 13, we further examined the effect of genetic search for the other test problems. Experimental results are summarized in Table 14 using the $\mathrm{D1_R}$ measure where we used the tuned parameter values of k, p_{LS} and T for each test problem. In this table, the column labeled as (p_C, p_M) shows the best result among the 121 combinations of p_C and p_M for each test problem. On the other hand, (0, p_M) and (p_C, 0) mean the best specification of p_M when $p_C = 0$ (i.e., no crossover) and the best specification of p_C when $p_M = 0$ (i.e., no mutation), respectively. For the results in Table 14, we examined the statistical significance using the Mann-Whitney U test for three confidence levels 95%, 97.5% and 99%. More specifically, we compared each result in the four columns in Table 14 obtained from the four variants of our MOGLS: LS (the population-based multiobjective local search algorithm with no genetic operations where $p_C = 0$ and $p_M = 0$), C (MOGLS with no mutation where $p_C > 0$ and $p_M = 0$), M (MOGLS with no crossover where $p_C = 0$ and $p_M > 0$) and CM (MOGLS with both genetic operations where $p_C > 0$ and $p_M > 0$). We examined the confidence level with which one algorithm can be viewed as being better than another algorithm for each test problem based on the $\mathrm{D1_R}$ measure. Results are summarized in Table 15 where $\mathrm{A} \prec \mathrm{B}$ means that the algorithm A outperforms the algorithm B. In this table, "-" means that the confidence level is less than 95%. From the fourth column of Table 15, we can see that our MOGLS with both genetic operations (i.e., CM) significantly outperformed its variant with no genetic operations (i.e., LS) for all the eight test problems. We can also see from the last two columns of Table 15 that the use of at least one genetic operation (i.e., C or M) significantly improved the performance of our MOGLS with no genetic operations (i.e., LS) for many test problems. These results suggest that at least one genetic operation is necessary in our MOGLS. The necessity of both genetic operations was clearly shown in the second and third columns of Table 15 for some test problems (e.g., 2/40 and 3/60) while it was not clear for other test problems (e.g., 2/60 and 3/40). Moreover the best result for the 3/80 test problem was obtained from the case of $p_C > 0$ and $p_M = 0$ (see Table 14).

Table 14 Comparison of the four cases with respect to the parameter specifications in genetic search for each of the eight test problems. The average value of the $D1_R$ measure and the corresponding standard deviation are shown for each case.

Test problem	Specification of (p_C, p_M)			
	(p_C, p_M)	(0, p_M)	(p_C, 0)	(0, 0)
2/20	4.4 (0.7)	4.6 (0.7)	5.9 (1.1)	8.5 (1.7)
2/40	17.3 (3.1)	21.8 (3.0)	22.2 (4.7)	28.5 (5.3)
2/60	19.2 (2.4)	19.7 (1.6)	20.4 (2.4)	22.5 (2.9)
2/80	61.6(10.5)	68.6 (9.6)	63.8(12.0)	72.3(14.9)
3/20	7.4 (0.6)	7.7 (1.0)	8.3 (1.0)	11.6 (1.7)
3/40	18.9 (2.5)	19.5 (2.1)	19.5 (2.6)	21.3 (2.8)
3/60	29.5 (3.6)	39.6 (3.6)	42.7 (3.9)	53.4 (4.7)
3/80	28.2 (3.3)	29.6 (4.3)	28.2 (3.3)	31.5 (3.3)

Table 15 Comparison of the four algorithms based on the results in Table 14. In the first row, $A \prec B$ means that the algorithm A outperforms the algorithm B.

Test Problem	CM $\prec$ M	CM $\prec$ C	CM $\prec$ LS	M $\prec$ LS	C $\prec$ LS
2/20	-	99	99	99	99
2/40	99	99	99	99	99
2/60	-	-	99	99	97.5
2/80	95	-	99	-	95
3/20	-	99	99	99	99
3/40	-	-	99	95	-
3/60	99	99	99	99	99
3/80	-	-	99	-	99

IV. Comparison with other EMO algorithms

A. Comparison with SPEA and NSGA-II

We compare our MOGLS with the SPEA [10] and the NSGA-II [13] through computational experiments on the eight test problems under the same stopping condition (i.e., evaluation of 100 000 solutions). Fair comparison among different algorithms is not easy especially when they involve many parameters. Since different parameter values may be appropriate for each of the three algorithms (i.e., MOGLS, SPEA and NSGA-II), we examined 27 combinations of the following parameter values:

Population size (N_{pop}): 30, 60, 120,

Crossover probability (p_{C}): 0.6, 0.8, 1.0,

Mutation probability per string (p_{M}): 0.4, 0.6, 0.8.

In the SPEA, the size of the secondary population was specified as 60 independent of the size of the primary population. The values of *k*, p_{LS} and *T* tuned in Section III were used for each test problem in our MOGLS. We used the two-point crossover in Fig. 7 and the insertion mutation in Fig. 8 for all the three algorithms. The insertion mutation was also used for local search in our MOGLS.

Each algorithm was applied to each test problem 20 times for each of the 27 combinations of the parameter values. Thus 540 solution sets were obtained by each algorithm for each test problem. Table 16 summarizes the best, average and worst values of the D1_{R} measure over those 540 solution sets. From this table, we can see that the performance of the NSGA-II strongly depends on the parameter specifications. While there are no large differences in the best results among the three algorithms except for the results on the 2/80 and 3/20 test problems, the worst results by the NSGA-II are much inferior to those by the other algorithms for all the eight test problems. The worst results by the MOGLS are better than those by the SPEA for six test problems except for 3/60 and 3/80. This means that the performance of our MOGLS is less sensitive to the parameter specifications of N_{pop}, p_{C} and p_{M} in the EMO part than the SPEA and the NSGA-II. The best results for the two-objective test problems in Table 16 were obtained by our MOGLS on the average while the SPEA was the best for the three-objective test problems.

Table 16 The best, average and worst values of the $D1_R$ measure over 540 solution sets obtained by each algorithm for each test problem.

Test	SPEA			NSGA-II			MOGLS		
problem	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
2/20	2.9	6.0	12.5	3.3	12.1	40.8	2.0	5.3	9.4
2/40	7.9	18.3	33.4	9.1	28.4	76.8	10.7	20.0	31.5
2/60	12.4	22.6	34.1	13.1	28.1	72.5	13.1	21.3	31.4
2/80	40.6	84.9	145.4	38.0	99.9	220.8	26.4	72.2	116.7
3/20	4.2	7.7	15.4	9.0	24.0	54.1	5.9	8.1	12.3
3/40	11.7	17.8	31.5	13.2	45.0	89.6	14.2	20.8	29.0
3/60	15.1	24.8	38.4	15.6	46.8	80.0	18.2	27.2	38.8
3/80	19.9	28.8	42.4	21.2	49.8	90.5	20.1	32.2	49.3

We further examined the performance of each algorithm using the best values of the three parameters N_{pop}, p_C and p_M for each test problem. That is, we chose the parameter values from which the best solution set was obtained by each algorithm for each test problem in Table 16. Using those parameter values, we applied each algorithm to each test problem 20 times. Experimental results were summarized in Table 17 for the $D1_R$ measure, Table 18 for the ratio of non-dominated solutions, and Table 19 for the number of obtained solutions. We can see from Table 18 that our MOGLS outperformed the other algorithms for six test problems in terms of the ratio of non-dominated solutions. On the other hand, our MOGLS was inferior to the other algorithms in terms of the number of obtained solutions for the two-objective test problems in Table 19. These results suggest that our MOGLS tends to find fewer solutions with higher quality than the SPEA and the NSGA-II (we have similar results by the hybridization of the SPEA and the NSGA-II with local search in the next subsection). As a result, our MOGLS is comparable to the other algorithms for many test problems with respect to the $D1_R$ measure in Table 17.

In Table 20, we compare the three algorithms in terms of the computation time. All the three algorithms were coded in C and executed on a PC with a Pentium 4 CPU (2.2 GHz). We used the same code in the three algorithms for calculating the objective functions of each solution. Since the number of evaluated solutions was used as the stopping condition, the three algorithms spent the same computation time for solution evaluations. Thus the difference in the total computation time among the three algorithms stemmed from the difference in their fitness calculation mechanisms and the generation update mechanisms. While the SPEA and NSGA-II used sophisticated fitness calculation mechanisms based on the Pareto-dominance relation and the concept of crowding, our MOGLS used a simple scalar fitness function. Moreover, the local search operation for generating new solutions is simpler than the genetic operations with selection, crossover and mutation. From Table 20, we can expect that experimental results may favor our MOGLS when the three algorithms are compared under the same computation time instead of the same number of examined solutions. In Table 21, we show the average values of the $D1_R$ measure obtained by each algorithm when the same computation time was used as the stopping condition. In computational experiments in Table 21, the execution of each algorithm was iterated for 10 seconds. From this table, we can see that better results were obtained from our MOGLS than the SPEA and the NSGA-II for seven test problems in Table 21 (except for the 2/40 problem).

Table 17 Comparison of the three algorithms using the $D1_R$ measure (smaller values mean better solution sets).

Test problem	SPEA	NSGA-II	MOGLS
2/20	5.1 (1.3)	5.8 (1.5)	4.6 (1.1)
2/40	21.7 (3.7)	15.2 (3.2)	21.3 (4.2)
2/60	19.1 (3.0)	17.8 (1.8)	20.2 (2.7)
2/80	97.3(19.2)	71.3(14.7)	60.4(12.2)
3/20	10.8 (1.1)	10.6 (0.9)	8.2 (0.8)
3/40	16.2 (1.9)	20.2 (2.7)	19.2 (2.2)
3/60	24.9 (3.9)	35.9 (6.7)	24.5 (3.3)
3/80	25.9 (2.8)	27.8 (3.4)	29.8 (3.4)

Table 18 Comparison of the three algorithms using the ratio of non-dominated solutions (larger values mean better solution sets).

Test problem	SPEA	NSGA-II	MOGLS
2/20	0.60 (.14)	0.57 (.15)	0.64 (.11)
2/40	0.18 (.23)	0.70 (.29)	0.35 (.24)
2/60	0.37 (.25)	0.42 (.22)	0.56 (.27)
2/80	0.14 (.20)	0.38 (.36)	0.73 (.27)
3/20	0.33 (.14)	0.41 (.14)	0.85 (.10)
3/40	0.60 (.18)	0.48 (.26)	0.64 (.19)
3/60	0.61 (.25)	0.21 (.12)	0.78 (.22)
3/80	0.58 (.36)	0.53 (.30)	0.54 (.28)

Table 19 Comparison of the three algorithms using the number of obtained solutions (larger values mean better solution sets).

Test problem	SPEA	NSGA-II	MOGLS
2/20	23.5 (3.0)	19.6 (2.1)	22.7 (3.0)
2/40	21.8 (3.8)	23.1 (2.6)	20.1 (5.4)
2/60	21.5 (5.2)	19.9 (4.0)	13.7 (3.7)
2/80	12.0 (4.6)	16.0 (3.7)	11.9 (3.6)
3/20	30.5 (0.8)	48.9 (4.8)	104.9(11.9)
3/40	60.3 (2.1)	59.2 (3.6)	73.8 (14.2)
3/60	61.0 (1.1)	41.5 (5.7)	71.4 (14.6)
3/80	60.6 (1.3)	53.6 (7.8)	53.2 (11.1)

Table 20 Comparison of the three algorithms using the computation time (seconds).

Test problem	SPEA	NSGA-II	MOGLS
2/20	4.6 (0.12)	8.1 (0.02)	3.3 (0.04)
2/40	6.3 (0.05)	10.6 (0.03)	7.5 (0.03)
2/60	9.6 (0.23)	13.2 (0.05)	7.0 (0.05)
2/80	11.1 (0.10)	15.7 (0.04)	9.8 (0.11)
3/20	7.5 (0.19)	8.9 (0.06)	4.0 (0.10)
3/40	10.9 (0.78)	11.2 (0.02)	5.1 (0.04)
3/60	16.0 (0.97)	11.4 (0.04)	8.7 (0.17)
3/80	15.5 (1.11)	16.4 (0.04)	9.8 (0.14)

Table 21 Comparison of the three algorithms using the $D1_R$ measure under the same computation time: 10 seconds.

Test problem	SPEA	NSGA-II	MOGLS
2/20	4.4 (1.1)	5.7 (1.3)	3.3 (1.0)
2/40	17.4 (2.9)	15.6 (3.1)	19.3 (4.0)
2/60	18.8 (2.9)	19.8 (2.3)	18.4 (3.0)
2/80	101.2(19.1)	91.2(16.8)	59.9(12.1)
3/20	11.1 (1.2)	10.7 (1.1)	6.3 (1.0)
3/40	16.4 (1.8)	20.6 (2.7)	16.0 (2.1)
3/60	30.2 (4.9)	35.3 (5.1)	23.5 (3.5)
3/80	30.1 (3.8)	31.9 (3.5)	29.4 (3.4)

B. Hybridization of EMO Algorithms

In our MOGLS, the local search direction (i.e., the weight vector in the scalar fitness function) for each solution is not inherited from the EMO part. This means that the local search part is independent of the EMO part. Thus the local search part can be combined with other EMO algorithms such as the SPEA and the NSGA-II. We implemented a hybrid SPEA by combining the SPEA with the local search part of our MOGLS. A hybrid NSGA-II was also implemented in the same manner. In those hybrid algorithms, the SPEA and the NSGA-II are used as the EMO part of Fig. 4 with no modifications. The local search part is applied to the new population generated by the EMO part. The improved population is returned to the EMO part as the current population. In the hybrid SPEA, local search is not applied to the secondary population as in our MOGLS. The secondary population is updated using the primary population improved by local search. We also implemented another version (say Ver.2) of hybrid algorithms where the scalar fitness function was not used for selecting initial solutions for local search. In the Ver.2 hybrid algorithms, local search is applied to each solution with the local search probability p_{LS} independent of its quality as an initial solution. The local search direction of each solution is specified by the pseudo-weight vector in (9). A similar idea to the Ver.2 hybrid algorithms has already been used in Table 5 for avoiding too much selection pressure.

These two versions of the hybrid SPEA and the hybrid NSGA-II were compared with their non-hybrid versions (i.e., pure EMO algorithms). Each algorithm was applied to each test problem 20 times using the same stopping condition: evaluation of 100 000 solutions. In the EMO part of each hybrid algorithm, we used the same parameter values as its non-hybrid version in Tables 17-20. That is, the parameter values in the EMO part were tuned not for each hybrid algorithm but for its original pure EMO algorithm. In the local search part of each hybrid algorithm, the best combination of k , p_{LS} and T was chosen for each test problem from their 18 combinations (i.e., $k = 1$, 10, 100, $p_{\mathrm{LS}} = 0.01$, 0.1 and $T = 1$, 10, 100). The average value of the $\mathrm{D1_R}$ measure, the average ratio of non-dominated solutions, the average number of obtained solutions and the average computation time are summarized in Tables 22, 23, 24 and 25, respectively. We examined whether each hybrid algorithm outperformed its original pure EMO algorithm for each test problem. When we can confirm that a hybrid algorithm outperformed its non-hybrid version with the 95% confidence level by the Mann-Whitney U test, the corresponding result by the hybrid algorithm is highlighted by boldface in each table. From Table 22, we can see that the performance of the SPEA and the NSGA-II was significantly improved for some test problems by the hybridization with local search. Such improvement is also observed in Table 23. The difference between the two versions of the hybridization was not large for many test problems in Table 22. On the other hand, the hybridization with local search severely decreased the number of obtained solutions for some test problems (i.e., 2/80 and 3/60) while there exist some counter-examples in Table 24. Moreover, the hybridization with local search significantly decreased the computation time for many test problems as shown in Table 25. Thus the experimental results in Table 22 and Table 23 will more favor the hybrid algorithms if computational experiments are performed under the same computation time as in Table 21.

Table 22 Average value of the $D1_R$ measure. Each boldface result by a hybrid algorithm can be viewed as being better than the corresponding result by its non-hybrid version with the 95% confidence level.

Test problem	Pure SPEA	Hybrid SPEA		Pure NSGA-II	Hybrid NSGA-II	
		Ver.1	Ver.2		Ver.1	Ver.2
2/20	5.1	4.7	5.0	5.8	5.1	5.1
2/40	21.7	**15.7**	**19.6**	15.2	14.6	14.4
2/60	19.1	18.4	18.5	17.8	18.0	17.5
2/80	97.3	**50.5**	**71.7**	71.3	**60.5**	64.7
3/20	10.8	**9.6**	**9.3**	10.6	**9.7**	**8.8**
3/40	16.2	15.9	15.9	20.2	**18.2**	**18.2**
3/60	24.9	23.6	23.6	35.9	**28.8**	**28.2**
3/80	25.9	26.0	25.4	27.8	27.2	27.1

Table 23 Average ratio of non-dominated solutions. Each boldface result by a hybrid algorithm can be viewed as being better than the corresponding result by its non-hybrid version with the 95% confidence level.

Test problem	Pure SPEA	Hybrid SPEA		Pure NSGA-II	Hybrid NSGA-II	
		Ver.1	Ver.2		Ver.1	Ver.2
2/20	0.48	**0.61**	0.50	0.50	0.53	0.53
2/40	0.05	**0.34**	0.02	0.31	0.39	0.32
2/60	0.24	0.29	0.22	0.25	0.19	0.24
2/80	0.04	**0.53**	0.11	0.11	**0.32**	**0.26**
3/20	0.23	**0.58**	**0.47**	0.26	**0.36**	**0.54**
3/40	0.29	**0.39**	0.36	0.24	0.31	0.28
3/60	0.42	0.44	0.47	0.18	**0.28**	0.15
3/80	0.26	0.32	0.34	0.35	0.23	0.22

Table 24 Average number of obtained solutions. Each boldface result by a hybrid algorithm can be viewed as being better than the corresponding result by its non-hybrid version with the 95% confidence level.

Test problem	Pure SPEA	Hybrid SPEA		Pure NSGA-II	Hybrid NSGA-II	
		Ver.1	Ver.2		Ver.1	Ver.2
2/20	23.5	22.3	23.2	19.6	20.3	**21.0**
2/40	21.8	23.7	22.5	23.1	21.5	24.2
2/60	21.5	20.7	23.4	19.9	20.8	**22.0**
2/80	12.0	9.7	10.1	16.0	16.5	16.0
3/20	30.5	30.6	30.8	48.9	47.0	**53.6**
3/40	60.3	60.3	60.8	59.2	56.7	58.7
3/60	61.0	30.8	30.8	41.5	31.2	37.4
3/80	60.6	60.6	59.5	53.6	51.1	53.3

Table 25 Average computation time (seconds). Each boldface result by a hybrid algorithm can be viewed as being better than the corresponding result by its non-hybrid version with the 95% confidence level.

Test problem	Pure SPEA	Hybrid SPEA		Pure NSGA-II	Hybrid NSGA-II	
		Ver.1	Ver.2		Ver.1	Ver.2
2/20	4.6	**3.1**	**4.3**	8.1	**6.9**	**4.0**
2/40	6.3	7.3	6.4	10.6	10.7	10.6
2/60	9.6	**9.0**	9.6	13.24	**13.21**	**13.19**
2/80	11.1	**9.7**	**11.0**	15.7	**14.0**	**13.6**
3/20	7.5	**2.7**	**2.6**	8.9	**4.3**	**2.7**
3/40	10.9	**9.5**	**10.4**	11.2	11.4	11.3
3/60	16.0	**13.1**	**12.4**	11.4	**6.8**	**7.4**
3/80	15.5	15.3	15.5	16.4	**16.3**	**16.3**

In Table 22, the largest improvement was achieved for the 2/80 test problem by the hybrid SPEA Ver.1 algorithm. Using this algorithm, we examined the effect of the parameters *k* and $p_{\rm LS}$ on the performance in the same manner as Fig. 13 in Section III. Experimental results are summarized in Fig. 16. In this figure, we can observe the negative effect of the hybridization with local search when both *k* and $p_{\rm LS}$ were large (i.e., the right-bottom corner). We can also observe the positive effect of the hybridization when *k* and $p_{\rm LS}$ were appropriately specified. That is, smaller values of the $D1_R$ measure were obtained by the hybrid algorithm than the case of $p_{LS} = 0$ with no local search (i.e., the top-most row of Fig. 16). In Fig. 16, the best result 43.8 was obtained when $k = 60$ and $p_{LS} = 0.02$. Using $k = 60$, we examined the effect of p_{LS} and T on the performance of the hybrid SPEA Ver.1 algorithm in the same manner as Fig. 16. Experimental results are summarized in Fig. 17. As in Fig. 16, we can observe the negative effect of the hybridization with local search in Fig. 17 when p_{LS} was large and T was small (i.e., the right-bottom corner). Moreover, we can observe a valley from the left-bottom corner to the right-top corner in Fig. 16 and Fig. 17 as in Fig. 13 by our MOGLS in Section III.

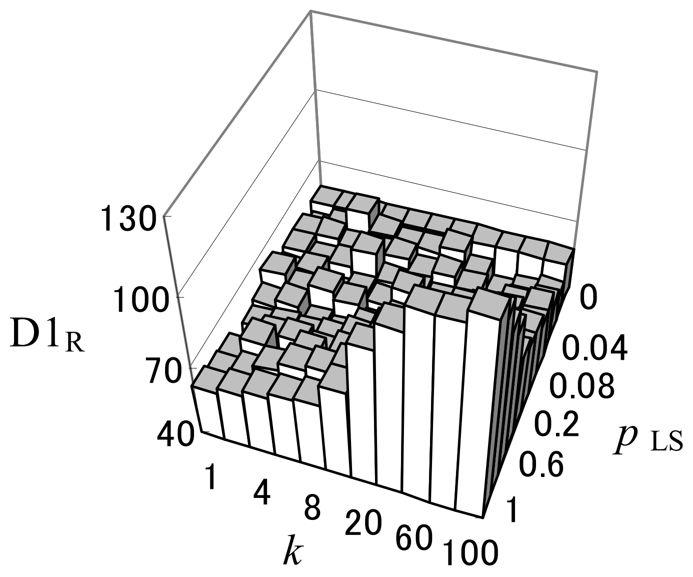


Fig. 16 Average value of the $D1_R$ measure obtained by the hybrid SPEA Ver.1 algorithm using various values of k and p_{LS} for the 2/80 test problem.

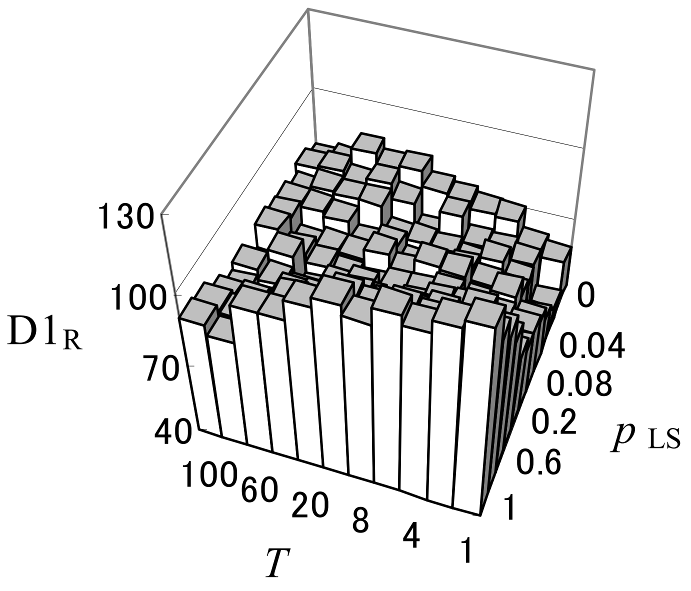


Fig. 17 Average value of the $\mathrm{D1_R}$ measure obtained by the hybrid SPEA Ver.1 algorithm using various values of p_{LS} and T for the 2/80 test problem.

While better results were obtained from the Ver.2 hybrid algorithms than their original pure EMO versions for all the eight test problems in Table 22, the Ver.1 hybrid algorithms did not always outperform their original versions (i.e., 3/80 for SPEA and 2/60 for NSGA-II). One possible reason is the use of the tuned parameter values for the SPEA and the NSGA-II. In Subsection IV.A, the best combination was chosen for the SPEA and the NSGA-II among 27 combinations of the population size N_{pop} (30, 60, 120), the crossover probability p_{C} (0.6, 0.8, 1.0) and the mutation probability p_{M} (0.4, 0.6, 0.8). The chosen combination for each EMO algorithm was also used for its hybrid versions in Tables 22-25. When the best combination among those 27 combinations was used for each hybrid algorithm, experimental results were improved for some test problems as shown in Table 26 (compare Table 26 with Table 22). Even in Table 26, the Ver.1 hybrid algorithms slightly deteriorated the performance of their original pure EMO algorithms (i.e., 3/80 for SPEA and 2/60 for NSGA-II). This may be due to the negative effect of the selection of initial solutions for local search. As we have already shown in Table 3 and Table 4, the proposed selection scheme of initial solutions for local search can degrade EMO algorithms. It should be noted, however, that much larger improvement was achieved in Table 26 for some test problems (e.g., 2/80) by the Ver.1 hybridization with the proposed selection scheme than the Ver.2 hybridization with no selection of initial solutions. The Ver.1 hybridization significantly improved the convergence speed to the Pareto-front of the SPEA in Table 23 for much more test problems than the Ver.2 hybridization. On the other hand, the number of obtained solutions by the Ver.1 hybrid NSGA-II was smaller than that by the Ver.2 hybrid NSGA-II for seven test problems in Table 24 (except for 2/80). These observations suggest that the proposed selection scheme of initial solutions for local search used in the Ver.1 hybridization tends to improve the convergence speed to the Pareto-front while it tends to degrade the diversity of solutions.

Table 26 Average value of the $D1_R$ measure obtained from the tuned parameter values of the population size, the crossover probability and the mutation probability. Each boldface result by a hybrid algorithm can be viewed as being better than the corresponding result by its non-hybrid version with the 95% confidence level.

Test problem	Pure SPEA	Hybrid SPEA		Pure NSGA-II	Hybrid NSGA-II	
		Ver.1	Ver.2		Ver.1	Ver.2
2/20	5.1	4.5	5.0	5.8	5.1	5.1
2/40	21.7	**14.9**	**15.2**	15.2	14.6	14.4
2/60	19.1	18.3	18.5	17.8	18.0	17.5
2/80	97.3	**46.0**	**71.7**	71.3	**54.9**	**61.8**
3/20	10.8	**8.2**	**7.8**	10.6	**9.3**	**8.8**
3/40	16.2	15.6	15.8	20.2	**18.2**	**18.2**
3/60	24.9	**21.7**	**22.0**	35.9	**27.3**	**27.1**
3/80	25.9	26.0	25.4	27.8	27.2	27.1

V. Conclusion and future research

In this paper, first we improved the performance of the former MOGLS [20], [21] by modifying its local search part for choosing only good individuals from the current population as initial solutions of local search and for appropriately specifying a local search direction of each initial solution. Next we examined positive and negative effects of the hybridization with local search on the performance of our MOGLS. Then we demonstrated the importance of striking a balance between genetic search and local search. We also examined the role of genetic search in our MOGLS. Moreover our MOGLS was compared with the SPEA and the NSGA-II. Finally we demonstrated that the local search part of our MOGLS could be easily combined with other EMO algorithms such as the SPEA and the NSGA-II. It was shown through computational experiments that the performance of the SPEA and the NSGA-II was significantly improved for some test problems by the hybridization with local search. It was also shown that the hybridization significantly decreased the computation time of those EMO algorithms for many test problems.

The main contribution of this paper is that the importance of striking a balance between genetic search and local search was clearly demonstrated through computational experiments on multiobjective permutation flowshop scheduling problems. For adjusting the balance, we used three parameters that can decrease the number of solutions examined by local search. The values of those three parameters were constant during the execution of our computational experiments. Dynamic control of those parameters is a future research topic. Tan et al. [49] proposed an idea of adjusting the number of solutions examined in local search in their multiobjective memetic algorithm. Many issues related to dynamic parameter control have already been studied for single-objective memetic algorithms [34], [47], [48], [50], [51]. Those studies can be extended to the case of multiobjective memetic algorithms where more emphasis should be placed on the diversity of solutions than the case of single-objective optimization.

The performance evaluation of our MOGLS in this paper is not complete. We compared our MOGLS with a population-based multiobjective local search (MOLS) algorithm, which was implemented by specifying the crossover probability p_{C} and the mutation probability p_{M} as $p_{\mathrm{C}} = 0$ and $p_{\mathrm{M}} = 0$. As summarized in Jaszkiewicz [52], a number of MOLS algorithms have been proposed in the field of multicriteria decision making such as multiobjective simulated annealing (MOSA [43], [53]) and multiobjective tabu search (MOTS [54]). Comparison of our MOGLS with those MOLS algorithms is a future research topic. It is also left for future research to compare our MOGLS with other multiobjective memetic algorithms such as the MOGLS of Jaszkiewicz [22] and the M-PAES of Knowles & Corne [23]. Jaszkiewicz [55] compared these two algorithms with three MOSA algorithms through computational experiments on multiobjective knapsack problems. He obtained the best results from his MOGLS [22] and an MOSA of Czyzak and Jaszkiewicz [43]. Jaszkiewicz's MOGLS and the M-PAES were also compared with each other on multiobjective knapsack problems by Knowles and Corne [24] where better results were obtained from the M-PAES than Jaszkiewicz's MOGLS.

In our MOGLS, simple hill climbing was used as local search. It is worth examining the use of other local search algorithms (e.g., simulated annealing and tabu search) in multiobjective memetic algorithms. Such a future study will motivate us to design adaptive multiobjective memetic algorithms that can dynamically control the balance between genetic search and local search through the choice of local search algorithms and neighborhood structures in addition to the adaptation of parameter values in a similar manner to multimeme memetic algorithms [34].

ACKNOWLEDGEMENT

The authors would like to thank the financial support from Japan Society for the Promotion of Science (JSPS) through Grand-in-Aid for Scientific Research (B): KAKENHI (14380194). Special thanks are due to the anonymous reviewers for their valuable comments.

REFERENCES

[1] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," *Proc. of 1st International Conference on Genetic Algorithms and Their Applications*, pp. 93-100, Carnegie-Mellon University, Pittsburgh, July 24-26, 1985.

[2] C. A. Coello Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information Systems*, vol. 1, no. 3, pp. 269-308, August 1999.

[3] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithms: Analyzing the state-of-the-art," *Evolutionary Computation*, vol. 8, no. 2, pp. 125-147, Summer 2000.

[4] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, Chichester, 2001.

[5] C. A. Coello Coello, D. A. van Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic Publishers, Boston, 2002.

[6] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," *Proc. of 5th International Conference on Genetic Algorithms*, pp. 416-423, University of Illinois at Urbana-Champaign, July 17-21, 1993.

[7] J. Horn, N. Nafpliotis and D. E. Goldberg, "A niched Pareto genetic algorithm for multi-objective optimization," *Proc. of 1st IEEE International Conference on Evolutionary Computation*, pp. 82-87, Orlando, June 27-29, 1994.

[8] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221-248, Fall 1994.

[9] J. D. Knowles and D. W. Corne, “The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimization,” *Proc. of 1999 Congress on Evolutionary Computation*, pp. 98-105, Washington D.C., July 6-9, 1999.

[10] E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach,” *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 4, pp. 257-271, November 1999.

[11] J. D. Knowles and D. W. Corne, “Approximating the nondominated front using Pareto archived evolution strategy,” *Evolutionary Computation*, vol. 8, no. 2, pp. 149-172, Summer 2000.

[12] E. Zitzler, K. Deb, and L. Thiele, “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results,” *Evolutionary Computation*, vol. 8, no. 2, pp. 173-195, Summer 2000.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, April 2002.

[14] P. Merz and B. Freisleben, “Genetic local search for the TSP: New results,” *Proc. of 4th IEEE International Conference on Evolutionary Computation*, pp. 159-164, Indianapolis, USA, April 13-16, 1997.

[15] N. Krasnogor and J. Smith, “A memetic algorithm with self-adaptive local search: TSP as a case study,” *Proc. of 2000 Genetic and Evolutionary Computation Conference*, pp. 987-994, Las Vegas, July 10-12, 2000.

[16] P. Moscato, “Memetic algorithms: A short introduction,” in D. Corne, F. Glover, and M. Dorigo (eds.), *New Ideas in Optimization*, McGraw-Hill, pp. 219-234, Maidenhead, 1999.

[17] W. E. Hart, N. Krasnogor, and J. Smith (eds.), *First Workshop on Memetic Algorithms* (WOMA I), in *Proc. of 2000 Genetic and Evolutionary Computation Conference Workshop Program,* pp. 95-130, Las Vegas, July 8, 2000.

[18] W. E. Hart, N. Krasnogor, and J. Smith (eds.), *Second Workshop on Memetic Algorithms* (WOMA II), in *Proc. of 2001 Genetic and Evolutionary Computation Conference Workshop Program,* pp. 137-179, San Francisco, July 7, 2001.

[19] W. E. Hart, N. Krasnogor, and J. Smith (eds.), *Proc. of Third Workshop on Memetic Algorithms* (WOMA III), Granada, Spain, September 7, 2002.

[20] H. Ishibuchi and T. Murata, “Multi-objective genetic local search algorithm,” *Proc. of 3rd IEEE International Conference on Evolutionary Computation*, pp. 119-124, Nagoya, Japan, May 20-22, 1996.

[21] H. Ishibuchi and T. Murata, “A multi-objective genetic local search algorithm and its

application to flowshop scheduling," *IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 28, no. 3, pp. 392-403, August 1998.

[22] A. Jaszkiewicz, "Genetic local search for multi-objective combinatorial optimization," *European Journal of Operational Research*, vol. 137, no. 1, pp. 50-71, February 2002.

[23] J. D. Knowles and D. W. Corne, "M-PAES: A memetic algorithm for multiobjective optimization," *Proc. of 2000 Congress on Evolutionary Computation*, pp. 325-332, San Diego, July 16-19, 2000.

[24] J. D. Knowles and D. W. Corne, "A comparison of diverse approaches to memetic multiobjective combinatorial optimization," *Proc. of 2000 Genetic and Evolutionary Computation Conference Workshop Program,* pp. 103-108, Las Vegas, July 8, 2000.

[25] J. D. Knowles and D. W. Corne, "A comparative Assessment of memetic, evolutionary, and constructive algorithms for the multiobjective *d*-MST problem," *Proc. of 2001 Genetic and Evolutionary Computation Conference Workshop Program,* pp. 162-167, San Francisco, July 7, 2001.

[26] K. Deb and T. Goel, "A hybrid multi-objective evolutionary approach to engineering shape design," *Proc. of 1st International Conference on Evolutionary Multi-Criterion Optimization*, pp. 385-399, Zurich, Switzerland, March 7-9, 2001.

[27] E. Talbi, M. Rahoual, M. H. Mabed, and C. Dhaenens, "A hybrid evolutionary approach for multicriteria optimization problems: Application to the flow shop," *Proc. of 1st International Conference on Evolutionary Multi-Criterion Optimization*, pp. 416-428, Zurich, Switzerland, March 7-9, 2001.

[28] P. Merz, "On the performance of memetic algorithms in combinatorial optimization," *Proc. of 2001 Genetic and Evolutionary computation Conference Workshop Program*, pp.168-173, San Francisco, July 7, 2001.

[29] P. Merz, "Memetic algorithms for combinatorial optimization problems: Fitness landscape and effective search strategy," Ph. D. Thesis, University of Siegen, December 2000.

[30] R. A. Dudek, S. S. Panwalkar, and M. L. Smith, "The lessons of flowshop scheduling research," *Operations Research*, vol. 40, no. 1, pp. 7-13, January/February 1992.

[31] P. Brucker, *Scheduling Algorithms*, Springer, Berlin, 1998.

[32] J. P. Watson, S. Rana, L. D. Whitley, and A. E. Howe, "The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling," *Journal of Scheduling*, vol. 2, no. 2, pp. 79-98, March/April 1999.

[33] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in hybrid evolutionary multi-criterion optimization algorithms," *Proc. of 2002 Genetic and*

Evolutionary Computation Conference, pp. 1301-1308, New York, July 9-13, 2002.

[34] N. Krasnogor, "Studies on the theory and design space of memetic algorithms," Ph. D. Thesis, University of the West of England, Bristol, June 2002.

[35] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 1999.

[36] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problem," *European Journal of Operational Research*, vol. 47, no. 1, pp. 65-74, July 1990.

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

[38] H. Ishibuchi, S. Misaki, and H.Tanaka, "Modified simulated annealing algorithms for the flow shop sequencing problem," *European Journal of Operational Research*, vol. 81, no. 2, pp. 388-398, March 1995.

[39] T. Murata, H. Ishibuchi, and H. Tanaka, "Genetic algorithms for flowshop scheduling problems," *Computer and Industrial Engineering*, vol. 30, no. 4, pp. 1061-1071, October 1996.

[40] M. Basseur, F. Seynhaeve, E. G. Talbi, "Design of multi-objective evolutionary algorithms: Application to the flow-shop scheduling problem," *Proc. of 2002 Congress on Evolutionary Computation*, pp. 1151-1156, Honolulu, May 12-17, 2002.

[41] T. P. Bagchi, *Multiobjective Scheduling by Genetic Algorithms*, Kluwer Academic Publishers, Boston, 1999.

[42] J. D. Knowles and D. W. Corne, "On metrics for comparing non-dominated sets," *Proc. of 2002 Congress on Evolutionary Computation*, pp. 711-716, Honolulu, May 12-17, 2002.

[43] P. Czyzak and A. Jaszkiewicz, "Pareto-simulated annealing – A metaheuristic technique for multi-objective combinatorial optimization," *Journal of Multi-Criteria Decision Analysis*, vol. 7, no.1, pp. 34-47, January 1998.

[44] R. Salomon, "Evolutionary algorithms and gradient search: Similarities and differences," *IEEE Trans. on Evolutionary Computation*, vol. 2, no. 2, pp. 45-55, July 1998.

[45] D. E. Goldberg and S. Voessner, "Optimizing global-local search hybrids," *Proc. of 1999 Genetic and Evolutionary Computation Conference*, pp. 220-228, Orlando, July 13-17, 1999.

[46] D. Orvosh and L. David, "Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints," *Proc. of 5th International Conference on Genetic Algorithms*, p.650, University of Illinois at Urbana-Champaign, July 17-21, 1993.

[47] W. E. Hart, "Adaptive global optimization with local search," Ph. D. Thesis, University of

California, San Diego, 1994.

[48] M. W. S. Land, “Evolutionary algorithms with local search for combinatorial optimization,” Ph. D. Thesis, University of California, San Diego, 1998.

[49] K. C. Tan, T. H. Lee, E. F. Khor, “Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization,” *IEEE Trans. on Evolutionary Computation*, vol. 5, no. 6, pp. 565-588, December 2001.

[50] N. Krasnogor and J. Smith, “Emergence of profitable search strategies based on a simple inheritance mechanism,” *Proc. of 2001 Genetic and Evolutionary Computation Conference*, pp. 432-439, San Francisco, July 7-11, 2001.

[51] B. Carr, W. Hart, N. Krasnogor, J. Hirst, E. Burke, J. Smith, “Alignment of protein structures with a memetic evolutionary algorithm,” *Proc. of 2002 Genetic and Evolutionary Computation Conference*, pp. 1027-1034, New York, July 9-13, 2002.

[52] A. Jaszkiewicz, “Multiple objective metaheuristic algorithms for combinatorial optimization,” Habilitation Thesis, 360, Poznan University of Technology, Poznan, 2001.

[53] E. L. Ulungu, J. Teghem, P. H. Fortemps, and D. Tuyttens, “MOSA method: a tool for solving multiobjective combinatorial optimization problems,” *Journal of Multi-Criteria Decision Analysis*, vol. 8, no.4, pp. 221-236, July 1998.

[54] M. P. Hansen, “Tabu search for multiobjective optimization: MOTS,” *Proc. of 13th International Conference on Multiple Criteria Decision Making*, Cape Town, South Africa, January 6-10, 1997.

[55] A. Jaszkiewicz, “Comparison of local search-based metaheuristics on the multiple objective knapsack problem,” *Foundations of Computing and Decision Sciences*, vol. 26, no. 1, pp. 99-120, 2001.

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.

Since 1987, he has been with Department of Industrial Engineering, Osaka Prefecture University, where he is currently a Professor. He was a Visiting Research Associate at the University of Toronto from August 1994 to March 1995 and from July 1997 to March 1998. His research interests include fuzzy systems, genetic algorithms, and neural networks. He is currently an Associate Editor for *IEEE Trans. on Systems, Man, and Cybernetics - Part B*, and *Mathware and Soft Computing*.

Tadashi Yoshida received the B.S. degree in industrial engineering from Osaka Prefecture University, Osaka, Japan, in 2001. He is currently working toward the M.S. degree at the same university.

His research interests include evolutionary multi-objective optimization and hybrid algorithms.

Tadahiko Murata (S'96 – M'97) received the B.S., M.S., and Ph.D. degrees from Osaka Prefecture University, Osaka, Japan, in 1994, 1996, and 1997 respectively.

He joined the Department of Industrial and Information Engineering, Ashikaga Institute Technology as a Research Associate in 1997, and became an Assistant Professor in 1998. Since 2001, he has been an Associate Professor in the Department of Informatics, Faculty of Informatics, Kansai University. His research interests include multiobjective optimization, scheduling, pattern classification, fuzzy systems, machine learning, and genetic algorithms. He received the 1997 Award from Institute of Systems, Control and Information Engineers for his paper on flowshop scheduling with interval processing time. He has been on the program committees of GECCO 2000-2003, EMO 2001, EMO 2003 and other conferences. He has been a member of the Soft Computing Committee of the IEEE SMC Technical Committee since 2002. He is a member of SOFT, JIMA, ISCIE, and ISGEC.