Use of biased neighborhood structures in multiobjective memetic algorithms

Hisao Ishibuchi, Yasuhiro Hitotsuyanagi, Noritaka Tsukamoto, and Yusuke Nojima

Department of Computer Science and Intelligent Systems,

Graduate School of Engineering, Osaka Prefecture University

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

hisaoi@cs.osakafu-u.ac.jp, hitotsu@ci.cs.osakafu-u.ac.jp, nori@ci.cs.osakafu-u.ac.jp, nojima@cs.osakafu-u.ac.jp

Phone: +81-72-254-9350 Fax: +81-72-254-9915

Abstract. In this paper, we examine the use of biased neighborhood structures for local search in multiobjective memetic algorithms. Under a biased neighborhood structure, each neighbor of the current solution has a different probability to be sampled in local search. In standard local search, all neighbors of the current solution usually have the same probability because they are randomly sampled. On the other hand, we assign larger probabilities to more promising neighbors in order to improve the search ability of multiobjective memetic algorithms. In this paper, we first explain our multiobjective memetic algorithm, which is a simple hybrid algorithm of NSGA-II and local search. Then we explain its variants with biased neighborhood structures for multiobjective 0/1 knapsack and flowshop scheduling problems. Finally we examine the performance of each variant through computational experiments. Experimental results show that the use of biased neighborhood structures clearly improves the performance of our multiobjective memetic algorithm.

Keywords: Multiobjective memetic algorithms, multiobjective optimization, evolutionary computation, knapsack problems, flowshop scheduling problems.

1. Introduction

Search ability of evolutionary algorithms for single-objective optimization problems is often significantly improved by the hybridization with local search. Such a hybrid algorithm is referred to as a memetic algorithm (Moscato 1999). High search ability of memetic algorithms has been reported in the literature (Krasnogor and Smith 2005; Ong et al. 2006; Smith 2007). An interesting issue is the adaptation of local search (Krasnogor et al. 2002; Ong and Keane 2004; Jakob 2006; Caponio et al. 2007). Adaptive memetic algorithms with multiple local search strategies are often called multimeme algorithms (Krasnogor et al. 2002).

Memetic algorithms have been applied to not only single-objective but also multiobjective problems (Knowles and Corne 2005). The main difficulty in the design of multiobjective memetic algorithms is the implementation of local search. This is because local search is basically a single-objective optimization technique. In its application to multiobjective problems, the comparison between the current solution and its neighbor should be based on multiple objectives. They are often non-dominated with each other. In this case, we can not say which is better between the current solution and its neighbor.

Some multiobjective memetic algorithms (MOMAs) use a weighted sum fitness function in local search to determine which is better between the current solution and its neighbor (Ishibuchi and Murata 1998; Jaszkiewicz 2002a; Ishibuchi et al. 2003; Jaszkiewicz 2004). Other MOMAs use Pareto dominance-based local search together with a secondary criterion for tiebreak (Knowles and Corne 2000a; Murata et al. 2003). These two types have been compared with each other (Knowles and Corne 2000b; Jaszkiewicz 2001; Ishibuchi and Narukawa 2004). Experimental results in these studies showed that better convergence to the Pareto front was obtained by Pareto dominance-based local search while better diversity along the Pareto front was obtained by weighted sum-based local search. It has also been reported in the literature (Jaszkiewicz 2002b; 2004) that MOMAs outperform pure multiobjective evolutionary algorithms (MOEAs) in their applications to knapsack problems.

As in the case of single-objective memetic algorithms (SOMAs), the adaptation of local search is an interesting issue in the design of MOMAs. This issue, however, has not been discussed in the literature for multiobjective optimization with only a few exceptions (Guo et al. 2006). This is because the implementation of local search itself is still a challenging issue in the design of MOMAs. In this paper, we empirically examine the use of biased neighborhood structures for local search in MOMAs. In almost all of the above-mentioned studies on MOMAs, a neighbor of the current solution is randomly generated from its neighborhood. Thus each neighbor has the same probability to be sampled in local search. It would, however, improve the search ability of MOMAs if we sample more promising neighbors with larger probabilities in local search. This is the basic idea behind the use of biased neighborhood structures for local search in MOMAs. In this paper, we implement this idea for multiobjective 0/1 knapsack problems (Zitzler and Thiele 1999) and multiobjective flowshop scheduling problems (Ishibuchi et al. 2003).

This paper is organized as follows. In Section 2, we explain a simple multiobjective genetic local search (S-MOGLS) algorithm. Our S-MOGLS is implemented by combining local search into NSGA-II of Deb et al. (2002) in a straightforward manner. When local search frequency is zero (i.e., the local search application probability is zero), our S-MOGLS is exactly the same as NSGA-II. We use such a simple MOMA in order to examine the effect of local search on the performance of MOMAs. In Section 3, we implement two variants of S-MOGLS for multiobjective 0/1 knapsack problems. Different problem-specific knowledge is incorporated into local search in each variant. In these variants, some neighbors have higher probabilities to be sampled in local search than others. We also implement a variant of S-MOGLS for multiobjective flowshop scheduling problems in Section 3 by incorporating problem-specific knowledge into local search. In Section 4, we examine the performance of each algorithm (i.e., NSGA-II, S-MOGLS and its variants) through computational experiments. Experimental results show that the performance of S-MOGLS is significantly improved by the incorporation of problem-specific knowledge into local search (i.e., by the use of biased neighborhood structures). It is also shown that the performance of NSGA-II is significantly improved by the hybridization with local search. Finally we conclude this paper in Section 5.

2. Multiobjective Memetic Algorithms

In this section, we first explain some basic concepts of multiobjective optimization. Next we explain the outline of our multiobjective memetic algorithm called S-MOGLS. Then we implement our S-MOGLS algorithm for multiobjective 0/1 knapsack and flowshop scheduling problems.

2.1 Multiobjective Optimization

Let us consider the following k-objective maximization problem:

$$\text{Maximize } \mathbf{z} = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x})), \tag{1}$$

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

where **z** is the objective vector, $f_i(\mathbf{x})$ is the *i*-th objective to be maximized, **x** is the decision vector, and **X** is the feasible region in the decision space.

Let **x** and **y** be two feasible solutions (i.e., feasible decision vectors) of the *k*-objective maximization problem in (1)-(2). If the following conditions hold, **y** can be viewed as being better than **x**:

$$\forall i,\ f_i(\mathbf{x}) \le f_i(\mathbf{y}) \quad \text{and} \quad \exists j,\ f_j(\mathbf{x}) < f_j(\mathbf{y}). \tag{3}$$

In this case, we say that **y** dominates **x** (equivalently **x** is dominated by **y**).

When **x** is not dominated by any other feasible solutions (i.e., when there exists no feasible solution **y** that dominates **x**), the solution **x** is referred to as a Pareto-optimal solution of the *k*-objective maximization problem in (1)-(2). The set of all Pareto-optimal solutions forms the tradeoff surface in the objective space. This tradeoff surface in the objective space is referred to as the Pareto front. Various EMO algorithms have been proposed to efficiently search for Pareto-optimal solutions as many as possible along the entire Pareto front by their single run (Deb 2001; Coello et al. 2002; Coello and Lamont 2004).

2.2 Outline of S-MOGLS

We use a simple multiobjective genetic local search (S-MOGLS) algorithm, which is a hybrid version of NSGA-II (Deb et al. 2002) with local search, in order to examine the effect of local search in comparison with the performance of the well-known base algorithm: NSGA-II. The outline of our S-MOGLS can be written as follows:

```
Step 1: $P$ := Initialize ($P$)
Step 2: while a termination condition is not satisfied, do
Step 3:         $P'$ := Selection ($P$)
Step 4:         $P''$ := Genetic Operations ($P'$)
Step 5:         $P'''$ := Local Search ($P''$)
Step 6:         $P$ := Generation Update ($P \cup P'' \cup P'''$)
Step 7: end while
Step 8: return (Non-dominated solutions ($P$))
```

This is a general framework of MOMAs where local search is invoked in every generation. In some MOMAs, local search is not used in every generation (e.g., it is used in every ten generations). In other MOMAs, local search is used in only the initial or final generation.

In Step 1, an initial population *P* is randomly generated. In Step 3, a prespecified number of pairs of parent solutions (i.e., a parent population *P*') are selected from the current population *P* using binary tournament selection with replacement. The number of pairs of parent solutions is the same as the population size when a single offspring is generated from each pair of parent solutions. Each solution is evaluated in the same manner as NSGA-II for parent selection. That is, the primary criterion is the rank of each solution, which is assigned based on Pareto-dominance relation. The best rank is assigned to solutions that are not dominated by any other solutions in *P*. The second rank is assigned to solutions that are not dominated by any other solutions in *P* except for the best rank solutions. In this manner, all solutions in *P* are sorted based on Pareto-dominance relation. When two solutions have the same rank, they are compared with each other using a secondary criterion called a crowding distance. Roughly speaking, the crowding distance of a solution is the sum of axis-wise distances between its adjacent two solutions with respect to the *k* axes of the objective space (for details, see Deb 2001 and Deb et al. 2002).

In Step 4, an offspring solution is generated from each pair of parent solutions by crossover and mutation. As a result, an offspring population *P*'' is generated. In Step 5, local search is probabilistically applied to each solution in the offspring population *P*''. Only when a solution in *P*'' is improved by local search, the obtained solution by local search is inserted into the improved population *P*'''. The next population in Step 6 is chosen from the current population *P*, the offspring population *P*'', and the improved population *P*''' in the same manner as NSGA-II (i.e., using the rank and the crowding distance of each solution). When the local search application probability is zero in Step 5, our S-MOGLS is exactly the same as NSGA-II.

2.3 Local Search in S-MOGLS

In local search, we use the following weighted sum fitness function:

$$f(\mathbf{x}) = \lambda_1 f_1(\mathbf{x}) + \lambda_2 f_2(\mathbf{x}) + \cdots + \lambda_k f_k(\mathbf{x}), \qquad (4)$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_k)$ is a non-negative weight vector. In this paper, we first generate a set of uniformly distributed weight vectors using the following formulation as in Murata et al. (2001):

$$\lambda_1 + \lambda_2 + \cdots + \lambda_k = d, \tag{5}$$

$$\lambda_i \in \{0, 1, ..., d\} \quad \text{for} \quad i = 1, 2, ..., k. \tag{6}$$

For example, we have six weight vectors (2, 0, 0), (1, 1, 0), (1, 0, 1), (0, 2, 0), (0, 1, 1), (0, 0, 2) when the value of d is specified as $d=2$ for $k=3$. In our computational experiments, the value of d is specified as $d=100$ for $k=2$ (i.e., 101 weight vectors), $d=13$ for $k=3$ (i.e., 105 weight vectors), $d=7$ for $k=4$ (i.e., 120 weight vectors), and $d=7$ for $k=6$ (i.e., 792 weight vectors).

When local search is employed, a weight vector is randomly drawn from the weight vector set generated by the above-mentioned method. Then an initial solution for local search is selected from the offspring population P'' using tournament selection with replacement. In order to choose a good initial solution for local search, we use a large tournament size (20 in our computational experiments). Solutions are compared with each other based on the weighted sum fitness function with the current weight vector in tournament selection. The same weighted sum fitness function is used for local search from the chosen initial solution. That is, the current solution and its neighbor are compared with each other in local search based on the weighted sum fitness function with the current weight vector. A neighbor is randomly generated from the neighborhood of the current solution. When a better neighbor is found, the current solution is replaced with that neighbor. This means that we adopt the first move strategy (i.e., the strategy to move to a neighbor that is first found to be better than the current solution). When a better neighbor is not found among a prespecified number of randomly drawn neighbors (say, among L_{fail} neighbors), the local search procedure is terminated. That is, L_{fail} is the upper bound on the number of successive failure attempts. In our computational experiments, L_{fail} is specified as $L_{\text{fail}}=5$. The local search procedure is also terminated by the total number of examined neighbors in a series of local search from the initial solution (say, L_{search} neighbors). That is, L_{search} is the upper bound on the total number of attempted local moves in a series of local search from a single initial solution. In our computational experiments, L_{search} is specified as $L_{\text{search}}=20$. We simultaneously use these two termination conditions. That is, local search is terminated when at least one of these two conditions is satisfied. If the initial solution is improved by local search, the final solution is added to the improved population *P*'''.

Local search is applied with a local search application probability P_{LS}, which is specified as $P_{\text{LS}}=0.1$ in our computational experiments. The selection of an initial solution and the probabilistic application of local search are iterated N_{pop} times in each generation where N_{pop} is the population size. Almost the same local search procedure can be more efficiently executed by calculating the number of solutions to which local search is applied in each generation as $\lfloor P_{\text{LS}} \times N_{\text{pop}} \rfloor$ where $\lfloor X \rfloor$ shows the largest integer that is smaller than or equal to *X*. This efficient implementation, however, is not used in our computational experiments since it slightly changes the original algorithm of S-MOGLS.

It should be noted that a weight vector is randomly drawn whenever an initial solution for local search is to be selected. This means that local search from each initial solution is governed by the weighted sum fitness function with a different weight vector. This is to search for a variety of Pareto-optimal solutions with a wide range of objective values along the entire Pareto front.

2.4 Implementation for Knapsack Problems

In this subsection, we explain the implementation of S-MOGLS for multi-objective 0/1 knapsack problems of Zitzler and Thiele (1999). In general, a *k*-objective *n*-item 0/1 knapsack problem (*k*-*n* problem) in Zitzler and Thiele (1999) is written as follows:

$$\text{Maximize } \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x})), \tag{7}$$

$$\text{subject to } \sum_{j=1}^{n} w_{ij} x_j \le c_i, \quad i = 1, 2, ..., k, \tag{8}$$

$$x_j = 0 \text{ or } 1, \quad j = 1, 2, ..., n, \tag{9}$$

where

$$f_i(\mathbf{x}) = \sum_{j=1}^{n} p_{ij} x_j, \quad i = 1, 2, ..., k. \tag{10}$$

In this formulation, **x** is an n-dimensional binary vector, p_{ij} is the profit of item j according to knapsack i, w_{ij} is the weight of item j according to knapsack i, and c_i is the capacity of knapsack i. Each solution **x** is handled as a binary string of length n. In Zitzler and Thiele (1999), nine test problems were examined. Each test problem has two, three or four objectives and 250, 500 or 750 items (i.e., 2-250, 2-500, 2-750, 3-250, 3-500, 3-750, 4-250, 4-500, and 4-750 test problems).

In the following, we explain problem-specific procedures in S-MOGLS for multiobjective 0/1 knapsack problems: repair, genetic operations, and local search.

In S-MOGLS, initial solutions are randomly generated. Such an initial solution does not always satisfy the constraint conditions in (8). Genetic operations and local moves also generate infeasible solutions even when their parents are feasible. We use a repair procedure based on a maximum profit/weight ratio as in Zitzler and Thiele (1999). When an infeasible solution is generated, a feasible solution is created by removing items in ascending order of the following maximum profit/weight ratio until the constraint conditions are satisfied:

$$q_j = \max\{p_{ij}/w_{ij} \mid i = 1, 2, ..., k\}, \quad j = 1, 2, ..., n. \tag{11}$$

The repair of infeasible solutions is implemented in the Lamarckian manner. That is, repaired strings are used in genetic operations in the next generation. For the comparison between the two repair schemes (i.e., Lamarckian and Baldwinian), see Ishibuchi et al. (2005).

As a crossover operator, we apply one-point crossover with a prespecified crossover probability to each pair of parent solutions. Whereas two offspring are generated from each pair of parent solutions by this crossover operation, only a single offspring is randomly chosen. The chosen offspring is used in mutation (the other offspring is discarded). When crossover is not applied, one of the two parents is randomly chosen and used in mutation. Bit-flip mutation is applied to each bit value of the chosen offspring (or parent) with a prespecified mutation probability. After mutation, the above-mentioned repair procedure is applied to the generated solution if it is infeasible. These genetic operations are iterated N_{pop} times to construct an offspring population P'' of size N_{pop}.

Then local search is applied to the generated offspring population P''. As we have already explained, local search is probabilistically applied to a solution selected from *P*'' by tournament selection of size 20. A neighbor is generated by flipping each bit value of the current solution with a prespecified probability (1/*n* in our computational experiments where *n* is the number of items, i.e., *n* is the string length). When the generated neighbor is infeasible, the above-mentioned repair procedure is used to generate a feasible neighbor. The probabilistic bit-flip operation and the repair procedure are exactly the same as those in mutation. The generated neighbor is accepted only when it is better than the current solution in local search. On the other hand, mutated offspring are always accepted after repair independently of their fitness values in mutation.

2.5 Implementation for Flowshop Scheduling Problems

In this subsection, we explain the implementation of S-MOGLS for multi-objective flowshop scheduling problems in Ishibuchi et al. (2003). Each solution of a flowshop scheduling problem with *n* jobs is represented by a permutation of the given *n* jobs {J1, J2, ..., J*n*}. A three-objective test problem with *n* jobs in Ishibuchi et al. (2003) is written as

$$\text{Minimize} \quad f_1(\mathbf{x}) = \max\{C_j : j = 1, 2, ..., n\}, \tag{12}$$

$$\text{Minimize} \quad f_2(\mathbf{x}) = \max\{\max\{C_j - d_j, 0\} : j = 1, 2, ..., n\}, \tag{13}$$

$$\text{Minimize} \quad f_3(\mathbf{x}) = \sum_{j=1}^{n} C_j, \tag{14}$$

where **x**, C_j and d_j are a permutation of the given *n* jobs, the completion time of the *j*th job, and its due-date, respectively. The first objective is to minimize the makespan, the second objective is to minimize the maximum tardiness, and the third objective is to minimize the total flow time. Two-objective test problems in Ishibuchi et al. (2003) have only the first two objectives: $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$.

Each test problem has two or three objectives and 20, 40, 60 or 80 jobs. All test problems have 20 machines. We denote the *k*-objective flowshop scheduling problem with *n* jobs as the *k*-*n* problem (i.e., 2-20, 2-40, 2-60, 2-80, 3-20, 3-40, 3-60 and 3-80 problems).

In the following, we explain problem-specific procedures in S-MOGLS for multiobjective flowshop scheduling: genetic operations and local search.

In S-MOGLS, initial solutions are randomly generated as permutations of the given *n* jobs. No repair procedure is needed because all permutations are feasible solutions in flowshop scheduling. As a crossover operator, we use two-point order crossover in Fig. 1 to generate a single offspring. In this crossover, two crossover points are randomly chosen to divide each parent into three parts. All jobs (A, B, G and H in Fig. 1) outside the crossover points are inherited from one parent (Parent 1 in Fig. 1) to the offspring with no changes. The other jobs (C, D, E and F in Fig. 1) are sorted in the same order as in the other parent (Parent 2 in Fig. 1). As a mutation operator, we use insertion in Fig. 2. This mutation is often referred to as shift. In this mutation, a randomly chosen job (F marked by * in Fig. 2) is inserted to a randomly chosen position (the 2nd locus marked by ** in Fig. 2). We use two-point order crossover and insertion mutation because good results were obtained by this combination of genetic operations in Murata et al. (1996) for flowshop scheduling to minimize the makespan. Insertion is also used to generate a neighbor from the current solution in local search.

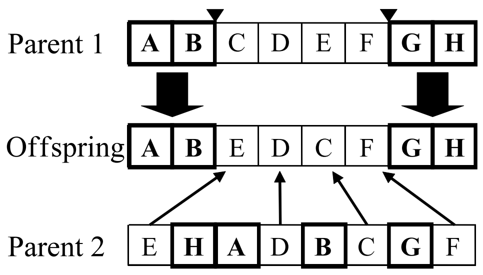


Fig. 1. Two-point order crossover.

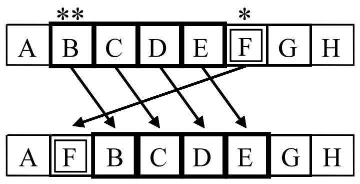


Fig. 2. Insertion (shift) mutation.

3. MOGLS with Biased Neighborhood Structures

In this section, we explain the implementation of biased neighborhood structures using problem-specific knowledge for the multiobjective 0/1 knapsack and flowshop scheduling problems. Effects of the use of biased neighborhood structures on the performance of S-MOGLS are examined in the next section.

3.1 Weighted Ratio Repair: MOGLS-WR

First we show that the use of the weighted sum fitness function in repair indirectly biases the neighborhood structure in local search for multiobjective 0/1 knapsack problems.

In an MOMA of Jaszkiewicz (2001; 2002b), the weighted sum fitness function in (4) was used for repair in the following manner: An infeasible solution is repaired by removing items in ascending order of the weighted profit/weight ratio:

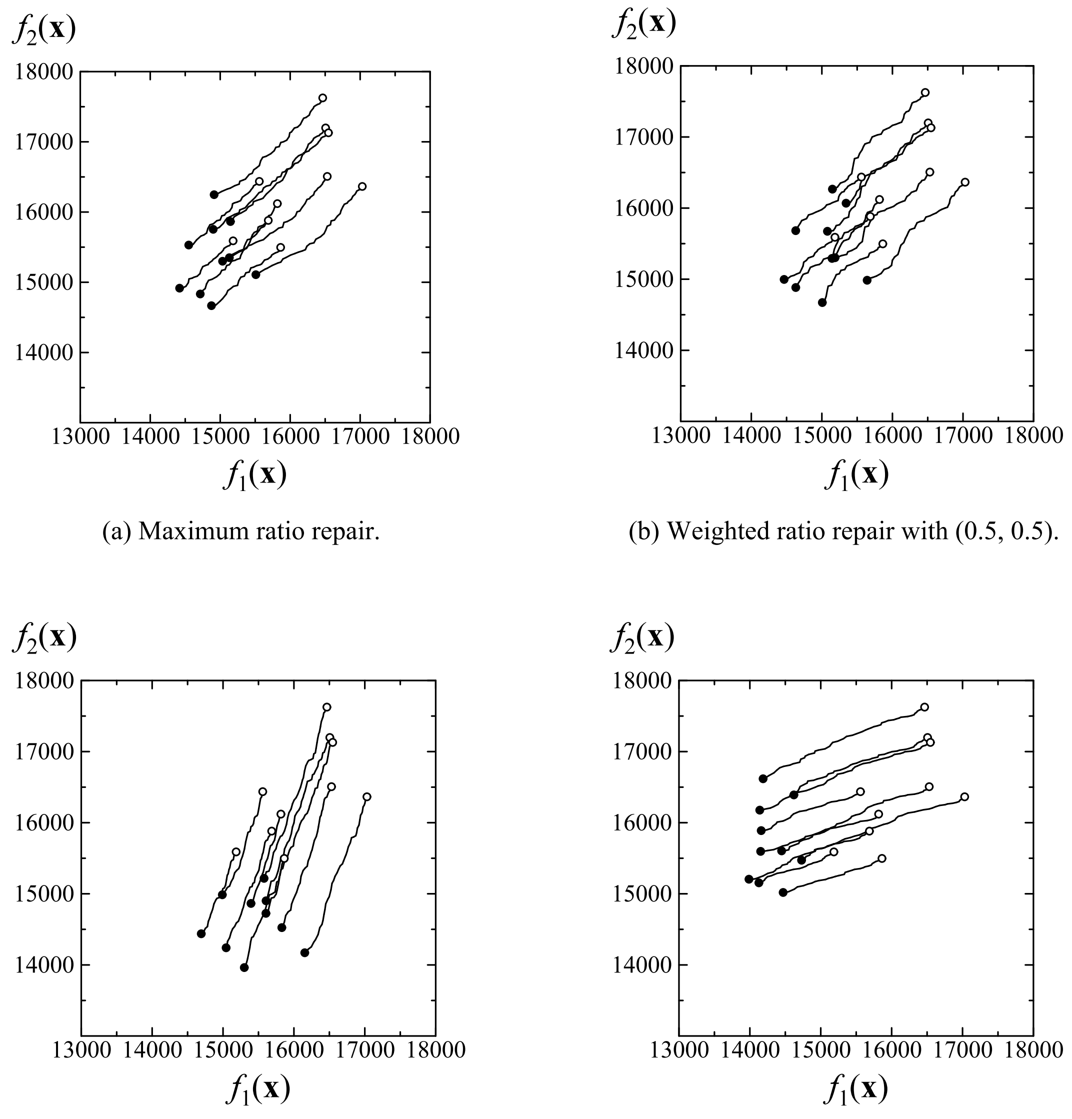
$$q_j = \sum_{i=1}^{k} \lambda_i p_{ij} \Bigg/ \sum_{i=1}^{k} w_{ij}\,, \quad j = 1, 2, ..., n\,, \tag{15}$$

where $\lambda = (\lambda_1, \lambda_2, ..., \lambda_k)$ is the current weight vector used in local search.

It should be noted that the weighted ratio repair is applicable to infeasible solutions only in local search with the weighted sum fitness function. Thus we still use the maximum ratio repair for infeasible solutions generated by genetic operations (and for infeasible solutions in an initial population).

For illustrating the difference between the maximum ratio repair in (11) and the weighted ratio repair in (15), we randomly generated an n-dimensional binary vector $\mathbf{x} = (x_1, ..., x_n)$ by assigning 0 with the probability 0.4 and 1 with the probability 0.6 to each x_j. Then we repaired the generated binary vector using one of the two repair methods. If the generated binary vector was feasible, we randomly generated another binary vector in the same manner. The random generation of an infeasible solution and the application of a repair method to the generated infeasible solution were iterated to obtain a prespecified number of feasible solutions. Then we drew the trajectory from each infeasible solution to its repaired one in the objective space.

In Fig. 3 (a), we show the trajectories from ten infeasible solutions by the maximum ratio repair for the 2-500 knapsack problem. In this figure, infeasible and feasible solutions are denoted by open and closed circles, respectively. It should be noted that the same order of items specified by (11) was always used for all the ten infeasible solutions in the case of the maximum ration repair. Thus the directions of the trajectories are similar to each other in Fig. 3 (a). On the other hand, we show the trajectories from the same ten infeasible solutions by the weighted ratio repair in Fig. 3 (b)-(d). The weight vector (λ_1, λ_2) was specified as (0.5, 0.5) in Fig. 3 (b), (0.9, 0.1) in Fig. 3 (c), and (0.1, 0.9) in Fig. 3 (d) just for illustration purpose. In Fig. 3 (b) with the same weight for the two objectives, the directions of the trajectories are similar to those in Fig. 3 (a) with the maximum ratio repair. The weighted ratio repair in Fig. 3 (c) tried to find feasible solutions with large values of the first objective because the weight for the first objective was large (i.e., 0.9). On the contrary, the weighted ratio repair in Fig. 3 (d) tried to find feasible solutions with large values of the second objective.



(a) Maximum ratio repair.

(b) Weighted ratio repair with (0.5, 0.5).

(c) Weighted ratio repair with (0.9, 0.1).

(d) Weighted ratio repair with (0.1, 0.9).

Fig. 3. Infeasible solutions (open circles) and repaired solutions (closed circles).

From Fig. 3, we can see that the probability of each feasible neighbor to be sampled from the current solution in local search strongly depends on the specification of the weight vector used in the weighted ratio repair. This means that the neighborhood structure is indirectly biased by the weight vector in local search.

Jaszkiewicz (2004) used the weighted profit/weight ratio in (15) not only for repair but also for item inclusion. An infeasible solution was first repaired by the weighted ratio repair in local search. Then the inclusion of each item into the repaired solution was examined in descending order of the weighted profit/weight ratio. We use this weighted ratio inclusion strategy as well as the weighted ratio repair in local search. When a generated neighbor in local search is feasible (i.e., when repair is not needed for the generated neighbor), we apply only the weighted ratio inclusion to the generated neighbor. In this paper, the S-MOGLS algorithm with the weighted ratio repair and inclusion is referred to as MOGLS-WR because the weighted ratio repair has a dominant effect on biasing the neighborhood structure.

3.2 Different Bit-Flip Probabilities: MOGLS-BF

In the previous subsection, we indirectly biased the neighborhood structure using the weighted ratio repair. In this subsection, we directly bias the neighborhood structure.

A good approximate solution can be obtained for a single-objective 0/1 knapsack problem by simply choosing items in descending order of the profit/weight ratio. For convenience of explanation, let us assume that n items in the single-objective 0/1 knapsack problem have already been sorted in descending order of the profit/weight ratio. That is, small item numbers close to 1 mean good items while large item numbers close to n mean bad items. Let us assume that the first N_{item} items are included in an approximate solution generated by the above-mentioned heuristic manner (the other items are not included in the solution). Now we consider local search from this approximate solution. This solution is not likely to be improved by removing good items with small item numbers. It is not likely to be improved by including bad items with large item numbers, either. On the contrary, promising neighbors seem to be generated by exchanging items with intermediate item numbers close to N_{item}. These discussions suggest the use of a different bit-flip probability for each item in local search: larger bit-flip probabilities for items with intermediate item numbers close to N_{item} than good items with small item numbers and bad items with large item numbers.

In the case of multiobjective 0/1 knapsack problems, it is not easy to sort items based on their profit/weight ratios because multiple profits are involved in each item. It is not the case, however, in weighted sum-based local search because multiple objectives are combined into a single weighted sum fitness function. That is, we can use the weighted profit/weight ratio in (15) for sorting items in local search.

When an initial solution is chosen for local search based on the weighted sum fitness function, we sort items (i.e., we tentatively renumber items) in descending order of the weighted profit/weight ratio using the current weight vector. Since we use a prespecified number of uniformly distributed weight vectors, the sorting of items can be preformed for each weight vector before the execution of our S-MOGLS algorithm.

Let N_{item} be the number of items in the current solution. We assign larger bit-flip probabilities to items with intermediate item numbers close to N_{item} than good items with small item numbers and bad items with large item numbers. In our computational experiments, we assign a larger bit-flip probability to 20 items from the $(N_{\text{item}} - 9)$*th* item to the $(N_{\text{item}} + 10)$*th* item than the other items. More specifically, the bit-flip probability for these 20 items is 1/20 while it is zero for the other items. That is, we concentrate local search on the examination of these 20 items. In this manner, we directly bias the neighborhood structure in local search for sampling more promising neighbors with larger probabilities.

It should be noted that the value of N_{item} is recalculated when the current solution is replaced with its neighbor in local search. It should also be noted that the *n* items are tentatively renumbered by the current weight vector when a new initial solution is selected for local search. We bias the neighborhood structure in the above-mentioned manner using different bit-flip probabilities in MOGLS-WR (i.e., we use different bit-flip probabilities explained in this subsection together with the weighted ratio repair and inclusion explained in the previous subsection). In this paper, the MOGLS-WR algorithm with different bit-flip probabilities is referred to as MOGLS-BF in order to clearly show that the neighborhood structure is directly biased by the use of different bit-flip probabilities.

3.3 Maximum Tardiness Insertion: MOGLS-MT

We have already explained two variants of S-MOGLS for multiobjective 0/1 knapsack problems (i.e., MOGLS-WR and MOGLS-BF). In this subsection, we explain a variant of S-MOGLS for multiobjective flowshop scheduling problems where the neighborhood structure is biased by assigning a different insertion probability to each job.

From the definition of the second objective (i.e., maximum tardiness) in (13) of our two-objective and three-objective flowshop scheduling problems, we can see that its objective value is determined by a single job with the largest delay. Thus the second objective is likely to be improved by moving this job to an earlier position in the current solution by an insertion move. Let us assume that the jth processing job has the largest delay in the current solution. In this case, the second objective is likely to be improved by inserting this job to one of the first $(j-1)$ positions. The inserted position is randomly chosen from these $(j-1)$ positions with the same probability (i.e., each position is chosen with the probability $1/(j-1)$). Since this heuristic insertion move is effective only for the second objective, we also use the standard insertion move (i.e., insertion of a randomly chosen job into a randomly chosen position).

In our computational experiments, the heuristic insertion for improving the maximum tardiness is used with the probability P_{MT} while the standard insertion is used with the probability $(1-P_{\text{MT}})$. It should be noted that we always use the standard insertion in mutation (i.e., the heuristic insertion is used only in local search). In this paper, the S-MOGLS algorithm with the heuristic insertion is referred to as MOGLS-MT in order to clearly indicate that the neighborhood structure is biased for improving the maximum tardiness (i.e., the second objective).

4. Computational Experiments

In this section, we examine the performance of each algorithm (i.e., NSGA-II, S-MOGLS and its three variants) through computational experiments. Experimental results clearly show the effectiveness of the use of biased neighborhood structures.

4.1 Performance Measures

In the field of evolutionary multiobjective optimization, a number of performance measures have already been proposed to evaluate the quality of an obtained non-dominated solution set (e.g., see Deb 2001). The performance of MOEAs and MOMAs has been evaluated by such a performance measure. It is, however, also shown that no single performance measure can simultaneously evaluate various aspects of an obtained non-dominated solution set such as its diversity along the Pareto front and its convergence to the Pareto front (see Zitzler et al. 2003). That is, the use of only a single performance measure is often misleading.

In this paper, we use four performance measures. Let S be a set of non-dominated solutions obtained by an MOEA or MOMA. The proximity of the solution set S to the Pareto front is evaluated by the generational distance (GD) as follows (Van Veldhuizen 1999):

$$\mathrm{GD}(S) = \frac{1}{|S|} \sum_{\mathbf{x}\in S} \min\{d_{\mathbf{xy}} \mid \mathbf{y}\in S^*\} , \tag{16}$$

where S^* is the set of all Pareto-optimal solutions and $d_{\mathbf{xy}}$ is the distance between a solution $\mathbf{x}$ and a Pareto optimal solution $\mathbf{y}$ in the k-dimensional objective space:

$$d_{\mathbf{xy}} = \sqrt{(f_1(\mathbf{x}) - f_1(\mathbf{y}))^2 + \cdots + (f_k(\mathbf{x}) - f_k(\mathbf{y}))^2} . \tag{17}$$

When the Pareto-optimal solution set S^* is not known, we construct S^* by choosing non-dominated solutions from the set of all the obtained solutions for each test problem in our computational experiments (including some preliminary experiments for each test problem).

Because each objective in our flowshop scheduling problems has a different order of magnitude, we use a normalized objective space when we calculate the generational distance for each flowshop scheduling problem. More specifically, we normalize the objective space of each test problem so that the minimum and maximum objective values in S^* become 0 and 100, respectively. Other performance measures, which are explained below, are calculated after the same normalization for the flowshop scheduling problems. On the other hand, we do not use such a normalization procedure for the knapsack problems where each objective has objective values of the same magnitude.

While the generational distance can evaluate only the convergence of the solution set *S* to the Pareto front, the following measure called $\mathrm{D1_R}$ (Knowles and Corne 2002) can evaluate both the convergence and the diversity of *S*:

$$\mathrm{D1_R}(S) = \frac{1}{|S^*|} \sum_{\mathbf{y}\in S^*} \min\{d_{\mathbf{xy}} \mid \mathbf{x}\in S\} \ . \tag{18}$$

This measure was used in Ishibuchi et al. (2003). Almost the same measure was also used in Czyzak and Jaszkiewicz (1998).

It should be noted that $\mathrm{D1_R}(S)$ is the average distance from each Pareto-optimal solution **y** in S^* to its nearest solution in *S* while GD(*S*) is the average distance from each solution **x** in *S* to its nearest Pareto-optimal solution in S^*.

A simple way to directly evaluate the diversity of solutions in the solution set *S* is to sum up the range of objective values over the *k* objectives as follows:

$$\mathrm{Range}(S) = \sum_{i=1}^{k} [\max_{\mathbf{x}\in S}\{f_i(\mathbf{x})\} - \min_{\mathbf{x}\in S}\{f_i(\mathbf{x})\}] . \tag{19}$$

Multiple solution sets can be directly compared with each other using Pareto dominance relation. Let us assume that we have *h* non-dominated solution sets S_1, S_2, ..., S_h. First all solutions in these solution sets are compared with each other using Pareto dominance relation. Next we remove all solutions that are dominated by other solutions. Let S_j^{ND} be the set of remaining solutions in the solution set S_j. That is, no solutions in S_j^{ND} are dominated by any other solutions in the *h* solution sets. Then we calculate the percentage of remaining solutions (i.e., the percentage of overall non-dominated solutions) over all solutions in each solution set as follows:

$$\mathrm{PND}(S_j) = \frac{|S_j^{\mathrm{ND}}|}{|S_j|} \times 100, \quad j = 1, 2, ..., h, \tag{20}$$

where $|S_j|$ is the cardinality of S_j (i.e., the number of solutions in S_j).

For a two-objective test problem, we can depict a 50% attainment surface (Fonseca and Fleming 1996) over multiple runs in order to visually show the performance of each algorithm in the two-dimensional objective space. Roughly and informally speaking, the 50% attainment surface can be viewed as a kind of median of multiple non-dominated solution sets in the two-dimensional objective space. See Fonseca and Fleming (1996) and Deb (2001) for the calculation of the 50% attainment surface.

4.2 Results on Knapsack Problems

We applied each of the four algorithms (i.e., NSGA-II, S-MOGLS, MOGLS-WR and MOGLS-BF) to each knapsack problem 30 times. The population size and the termination condition (i.e., the total number of examined solutions) were specified as in Table 1. The specifications in Table 1 are the same as those in Zitzler and Thiele (1999). The other parameters in the four algorithms were specified for all the nine knapsack problems as follows:

Crossover probability: 0.8 (One-point crossover),
Mutation probability: 1/*n* where *n* is the number of items (Bit-flip mutation),
Local search application probability: 0.1.

Table 1. Specifications of the population size and the termination condition for each of the nine knapsack problems.

Problem	Population size	Total number of examined solutions
2-250	150	75,000
2-500	200	100,000
2-750	250	125,000
3-250	200	100,000
3-500	250	125,000
3-750	300	150,000
4-250	250	125,000
4-500	300	150,000
4-750	350	175,000

Experimental results are summarized in Tables 2-5 for the above-mentioned four performance measures. In each row of these tables, the best result is highlighted by bold face with underline. We can see from these tables that the best results were obtained by MOGLS-WR for the two-objective test problems (i.e., 2-250, 2-500 and 2-750) and by MOGLS-BF for the three-objective and four-objective test problems (i.e., 3-250, 3-500, 3-750, 4-250, 4-500 and 4-750) with respect to all the four performance measures. These results show that the use of the biased neighborhood structures improved the search ability of S-MOGLS. We can also see that MOGLS-WR and MOGLS-BF consistently outperformed NSGA-II on all the nine test problems for all the four performance measures.

Table 2. Generational distance for knapsack problems. The termination condition is a prespecified number of examined solutions.

Problem	NSGA-II	MOGLS		
		S	WR	BF
2-250	94.8	112.2	**14.6**	31.4
2-500	273.6	345.7	**37.5**	91.9
2-750	470.1	553.1	**46.1**	116.1
3-250	245.2	268.2	85.4	**73.5**
3-500	478.4	534.6	175.1	**155.5**
3-750	845.5	936.9	223.3	**198.5**
4-250	355.2	352.9	153.0	**128.1**
4-500	1146.	1074.	377.1	**262.8**
4-750	1676.	1551.	551.2	**389.7**

Table 3. $D1_R$ measure for knapsack problems.

Problem	NSGA-II	MOGLS		
		S	WR	BF
2-250	390.4	354.4	**21.9**	41.5
2-500	819.9	786.5	**44.5**	114.8
2-750	1627.	1527.	**60.4**	173.0
3-250	508.6	525.1	147.9	**135.0**
3-500	1290.	1271.	293.5	**267.8**
3-750	2082.	2076.	364.5	**331.7**
4-250	597.3	595.1	281.5	**256.4**
4-500	1600.	1565.	576.2	**494.1**
4-750	2784.	2723.	864.6	**712.4**

Table 4. Range measure for knapsack problems.

Problem	NSGA-II	MOGLS		
		S	WR	BF
2-250	1288.3	1496.2	**4520.0**	4208.6
2-500	1654.2	1957.4	**7314.5**	6352.5
2-750	2224.5	2815.4	**11822.0**	10581.9
3-250	3601.5	3520.5	7489.7	**8056.4**
3-500	4410.5	4632.0	13382.9	**13975.6**
3-750	4466.4	4713.1	18439.1	**19367.9**
4-250	5691.2	5580.8	10180.1	**10847.4**
4-500	8763.7	8473.7	19192.3	**20372.4**
4-750	9705.1	9469.3	27707.7	**30656.3**

Table 5. Percentage of non-dominated solutions for knapsack problems.

Problem	NSGA-II	MOGLS		
		S	WR	BF
2-250	0.0	0.0	**92.6**	20.4
2-500	0.0	0.0	**99.4**	2.6
2-750	0.0	0.0	**98.3**	4.2
3-250	5.8	3.0	77.0	**82.5**
3-500	5.8	2.3	73.3	**82.1**
3-750	0.0	0.0	73.7	**81.1**
4-250	19.5	18.5	77.5	**87.1**
4-500	1.0	1.7	70.5	**89.4**
4-750	0.7	1.1	69.7	**90.7**

One may think that the performance improvement in the two variants of S-MOGLS with the biased neighborhood structures was achieved at the cost of a heavy computational overhead. In order to examine this issue, we also performed the same computational experiments using 15 seconds as the termination condition of each algorithm on a PC with a Pentium 4, 3.6GHz and 1GB of RAM. Experimental results are shown in Table 6. Because the same observations were obtained from the two specifications of the termination condition (i.e., the total number of examined solutions and the CPU time), we only show experimental results for the $D1_R$ measure in Table 6. It should be noted that the $D1_R$ measure can evaluate both the convergence of solutions to the Pareto front and their diversity. As in Table 3, the use of the biased neighborhood structures improved the performance of S-MOGLS in Table 6. We can also see that MOGLS-WR and MOGLS-BF consistently outperformed NSGA-II in Table 6.

Table 6. $D1_R$ measure for knapsack problems. The termination condition is a prespecified CPU time (15 seconds).

Problem	NSGA-II	MOGLS		
		S	WR	BF
2-250	339.9	273.5	**<u>16.7</u>**	30.5
2-500	880.1	768.9	**<u>41.5</u>**	110.9
2-750	1951.	1729.	**<u>95.2</u>**	203.8
3-250	545.8	507.0	144.7	**<u>132.7</u>**
3-500	1657.	1444.	308.6	**<u>264.3</u>**
3-750	2808.	2596.	470.5	**<u>357.2</u>**
4-250	694.3	641.1	278.8	**<u>251.6</u>**
4-500	2141.	1953.	606.1	**<u>496.3</u>**
4-750	3777.	3526.	1049.	**<u>770.8</u>**

In order to visually demonstrate the performance improvement of NSGA-II by the hybridization with local search, we show 50% attainment surfaces obtained by NSGA-II, S-MOGLS and MOGLS-WR for the 2-500 knapsack problem in Fig. 4. Each algorithm was terminated using 15 seconds as the termination condition. For comparison, we also show the Pareto front in Fig. 4. From this figure, we can see that the performance of NSGA-II was improved by the hybridization with local search in terms of both the diversity of solutions and their convergence to the Pareto front.

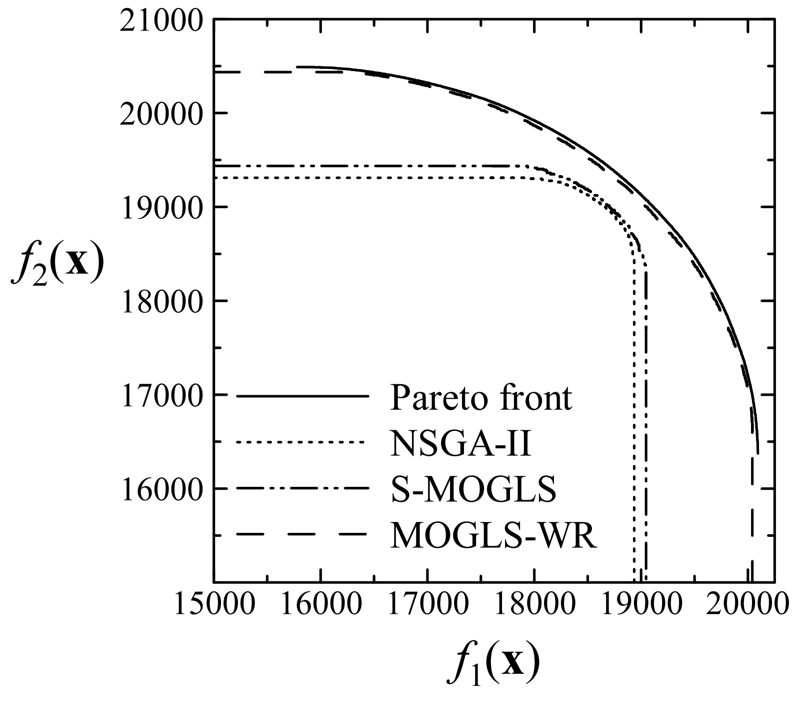


Fig. 4. Comparison of the three algorithms (NSGA-II, S-MOGLS and MOGLS-WR) using the 50% attainment surface by each algorithm for the 2-500 knapsack problem. The termination condition is a prespecified CPU time (15 seconds).

As shown in Tables 2-6, MOGLS-BF was outperformed by MOGLS-WR for the two-objective knapsack problems whereas the best results were obtained by MOGLS-BF for the other knapsack problems with three or four objectives. This means that the concentration of local search on only promising 20 items was not effective for the two-objective knapsack problems while it worked very well on the other knapsack problems. These observations suggest that broader local search is needed for the two-objective knapsack problems. In order to examine this issue, we performed computational experiments using various specifications of the number of promising items on which local search were concentrated.

In MOGLS-BF, the bit-flip probability in local search is 1/20 only for promising 20 items and zero for the other items. We generalize MOGLS-BF as MOGLS-BF(n_{BF}) where the bit-flip probability in local search is $1/n_{\mathrm{BF}}$ only for promising n_{BF} items and zero for the other items. The promising n_{BF} items are from the $(N_{\mathrm{item}} + 1 - n_{\mathrm{BF}}/2)th$ item to the $(N_{\mathrm{item}} + n_{\mathrm{BF}}/2)th$ item. We applied MOGLS-BF(n_{BF}) to each test problem using the total number of examined solutions in Table 1 as the termination condition. Experimental results are summarized in Table 7 for the $D1_R$ measure. We can see from Table 7 that the performance of MOGLS-BF (i.e., BF(20) in Table 7) was improved by the use of broader local search (i.e., BF(40) and BF(100)) for the two-objective knapsack problems. On the other hand, the best results for the four-objective knapsack problems were obtained from the use of narrower local search (i.e., BF(10)). These observations suggest that local search strategies can be more sophisticated by adjusting them not only to the problem type (e.g., knapsack problems) but also to the problem size (e.g., the number of objectives and the number of items).

Table 7. $D1_R$ measures obtained by MOGLS-BF(n_{BF}) for knapsack problems where local search is concentrated on promising n_{BF} items. The bit-flip probability for each of those items is $1/ n_{BF}$. BF(20) in this table is the same as MOGLS-BF in Tables 2-6.

Problem	BF(10)	BF(20)	BF(40)	BF(100)
2-250	48.5	41.5	31.3	**17.6**
2-500	108.7	114.8	106.1	**72.5**
2-750	173.8	173.0	149.9	**103.5**
3-250	133.9	135.0	135.5	**133.6**
3-500	**256.1**	267.8	271.1	268.2
3-750	**318.7**	331.7	346.5	344.8
4-250	**254.4**	256.4	258.5	262.8
4-500	**484.9**	494.1	508.2	514.0
4-750	**704.7**	712.4	725.9	735.2

We also performed the same computational experiments as in Table 7 after increasing the bit-flip probability for the promising items from $1/n_{BF}$ to $2/n_{BF}$. Experimental results are summarized in Table 8. We can see that experimental results in Table 8 are similar to those in Table 7. Actually we can obtain almost the same observations from these two tables. That is, except for the 3-250 test problems in Table 7, the best results were obtained by the broadest local search for the two-objective problems, and by the narrowest local search for the three-objective and four-objective knapsack problems in both tables.

Table 8. $D1_R$ measures obtained by MOGLS-BF(n_{BF}) for knapsack problems where local search is concentrated on promising n_{BF} items. The bit-flip probability for each of those items is $2/n_{BF}$.

Problem	BF(10)	BF(20)	BF(40)	BF(100)
2-250	45.2	40.0	29.9	**16.8**
2-500	101.2	107.1	101.6	**68.7**
2-750	145.7	148.2	135.3	**100.5**
3-250	**128.8**	130.5	132.4	129.3
3-500	**239.2**	246.4	256.9	252.8
3-750	**291.3**	300.9	324.9	328.1
4-250	**248.3**	250.5	252.6	260.2
4-500	**463.0**	473.3	486.5	503.6
4-750	**654.9**	671.8	679.2	708.1

4.3 Results on Flowshop Scheduling Problems

We applied NSGA-II and MOGLS-MT to each of the eight flowshop scheduling problems 30 times using the following parameter specifications:

Population size: 200,
Termination condition: Examination of 100,000 solutions,
Crossover probability: 0.9 (Two-point order crossover in Fig. 1),
Mutation probability: 0.6 for each string (Insertion mutation in Fig. 2),
Local search application probability: 0.1,
Probability of the heuristic insertion in local search: $P_{MT} = 0.0, 0.1, 0.2, 0.4, 0.8$,
Probability of the standard insertion in local search: $1 - P_{MT}$.

The same parameter specifications were used in NSGA-II and MOGLS-MT for all the eight flowshop scheduling problems. It should be noted that MOGLS-MT is exactly the same as S-MOGLS when $P_{MT} = 0.0$. In this case, the heuristic insertion for improving the maximum tardiness is never used in MOGLS-MT.

Experimental results are summarized in Tables 9-12. We can see that MOGLS-MT with $P_{MT} > 0.0$ outperformed S-MOGLS (i.e., MOGLS-MT with $P_{MT} = 0.0$) in many cases. This observation shows that the use of the biased neighborhood structure improved the performance of S-MOGLS. We can also see that the performance of NSGA-II was improved by the hybridization with local search on almost all test problems with respect to all the four measures.

Table 9. Generational distance for flowshop scheduling problems. The termination condition is a prespecified number of examined solutions.

Problem	NSGA-II	P_{MT} in MOGLS-MT				
		0.0	0.1	0.2	0.4	0.8
2-20	2.6	2.7	2.6	**<u>2.4</u>**	2.8	2.9
2-40	16.6	17.2	15.3	14.5	**<u>14.2</u>**	14.6
2-60	18.3	17.6	14.4	**<u>12.2</u>**	13.7	12.9
2-80	117.4	103.6	89.3	**<u>88.6</u>**	93.3	100.0
3-20	2.7	**<u>2.4</u>**	2.5	2.6	2.5	2.6
3-40	14.2	15.0	12.9	13.4	12.5	**<u>12.1</u>**
3-60	19.9	20.1	18.3	16.1	15.2	**<u>14.0</u>**
3-80	21.9	20.6	17.8	15.9	15.5	**<u>13.9</u>**

Table 10. $D1_R$ measure for flowshop scheduling problems.

Problem	NSGA-II	P_{MT} in MOGLS-MT				
		0.0	0.1	0.2	0.4	0.8
2-20	6.9	**<u>6.3</u>**	7.1	7.6	6.6	6.9
2-40	16.9	16.4	14.4	**<u>14.3</u>**	14.6	15.1
2-60	21.1	21.0	20.5	**<u>20.4</u>**	20.6	22.9
2-80	104.1	88.3	67.6	69.6	**<u>63.0</u>**	65.3
3-20	**<u>5.3</u>**	5.5	5.4	5.9	5.7	5.9
3-40	16.6	16.9	**<u>15.6</u>**	16.1	16.6	16.6
3-60	23.8	23.3	22.4	**<u>21.5</u>**	22.6	23.6
3-80	29.0	27.8	26.2	**<u>25.6</u>**	26.6	26.6

Table 11. Range measure for flowshop scheduling problems.

Problem	NSGA-II	P_{MT} in MOGLS-MT				
		0.0	0.1	0.2	0.4	0.8
2-20	75.0	**<u>81.7</u>**	74.6	68.0	74.5	77.4
2-40	89.0	**<u>109.7</u>**	98.6	102.0	96.0	93.1
2-60	**<u>66.8</u>**	59.4	62.3	59.1	64.5	**<u>66.8</u>**
2-80	237.8	223.0	197.5	246.0	265.2	**<u>306.8</u>**
3-20	**<u>272.0</u>**	264.5	267.0	261.1	264.0	264.4
3-40	228.9	232.6	235.0	**<u>235.3</u>**	221.0	222.2
3-60	181.8	194.6	**<u>195.0</u>**	187.1	189.0	176.5
3-80	154.8	**<u>160.2</u>**	144.4	138.7	144.0	139.8

Table 12. Percentage of non-dominated solutions for flowshop scheduling problems.

Proble m	NSGA-II	P_{MT} in MOGLS-MT				
		0.0	0.1	0.2	0.4	0.8
2-20	50.6	**52.6**	52.3	50.3	50.6	47.3
2-40	18.9	18.7	23.4	25.9	28.2	**30.9**
2-60	19.1	15.6	23.7	**40.9**	28.1	28.0
2-80	8.3	17.3	26.6	30.5	**39.3**	35.0
3-20	43.3	47.0	**47.1**	44.4	45.9	43.9
3-40	27.7	21.7	31.0	23.9	**32.1**	31.3
3-60	15.0	16.4	16.7	31.2	37.8	**38.4**
3-80	8.8	16.3	23.1	30.6	29.7	**49.9**

Experimental results under the termination condition specified by the CPU time of 15 seconds are summarized in Table 13 for the $D1_R$ measure. Since local search is faster than genetic search (i.e., more solutions can be examined by local search than genetic search in the same CPU time), the performance improvement of NSGA-II by the hybridization with local search became much clearer in Table 13 under the same CPU time than Table 10 under the same number of examined solutions.

Table 13. $D1_R$ measure for flowshop scheduling problems. The termination condition is a prespecified CPU time (15 seconds).

Proble m	NSGA-II	P_{MT} in MOGLS-MT				
		0.0	0.1	0.2	0.4	0.8
2-20	6.4	**5.5**	6.5	6.5	6.0	6.1
2-40	17.0	14.2	12.5	**12.3**	12.8	13.5
2-60	22.2	19.8	19.5	19.6	**19.4**	21.3
2-80	114.2	83.9	63.7	66.4	**61.1**	62.8
3-20	5.2	4.9	**4.8**	5.1	5.0	5.2
3-40	17.8	15.9	**14.9**	15.0	15.6	15.7
3-60	26.5	23.0	22.1	**20.9**	22.3	23.4
3-80	33.5	28.1	26.4	**25.8**	26.6	26.9

The performance of NSGA-II, S-MOGLS (i.e., MOGLS-MT with $P_{\mathrm{MT}} = 0.0$) and MOGLS-MT with $P_{\mathrm{MT}} = 0.1$ are visually compared using 50% attainment surfaces for the 2-80 flowshop scheduling problem in Fig. 5 where the termination condition is 15 seconds. We can see from this figure that the use of the biased neighborhood structure improved the performance of S-MOGLS with respect the maximum tardiness. This is because the neighborhood structure is biased only for improving the maximum tardiness. We can also see from Fig. 5 that the performance of NSGA-II was improved by the hybridization with local search with respect to both objectives.

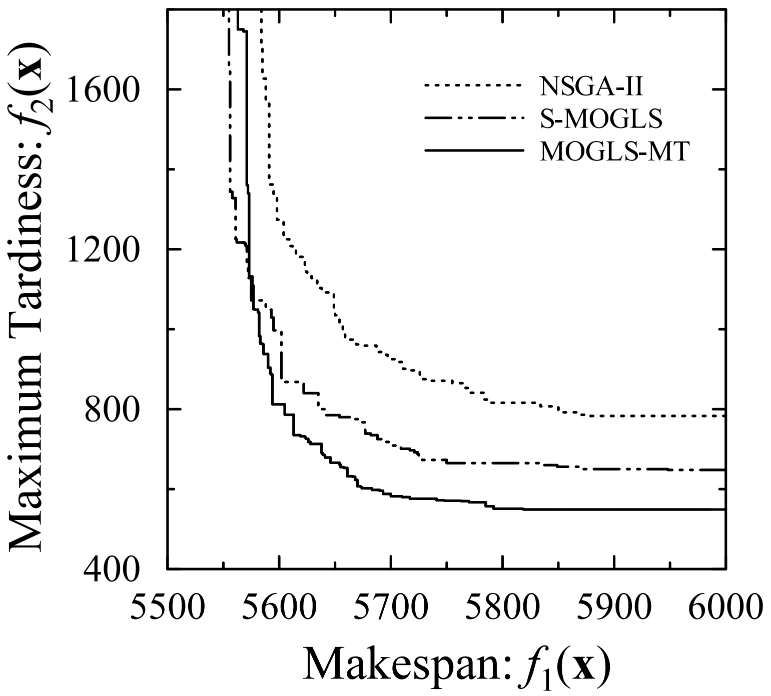


Fig. 5. Comparison of the three algorithms (NSGA-II, S-MOGLS and MOGLS-MT with $P_{\mathrm{MT}} = 0.1$) using 50% attainment surfaces for the 2-80 flowshop scheduling problem.

In Fig. 6, we show the effect of the specification of P_{MT} on the behavior of MOGLS-MT. We can see from Fig. 6 that the increase in the probability P_{MT} of the heuristic insertion for improving the maximum tardiness drove the search by MOGLS-MT toward smaller values of the second objective. As its side-effect, good results with respect to the first objective (i.e., makespan) were not obtained by MOGLS-MT when P_{MT} was large (e.g., $P_{\mathrm{MT}} = 0.8$).

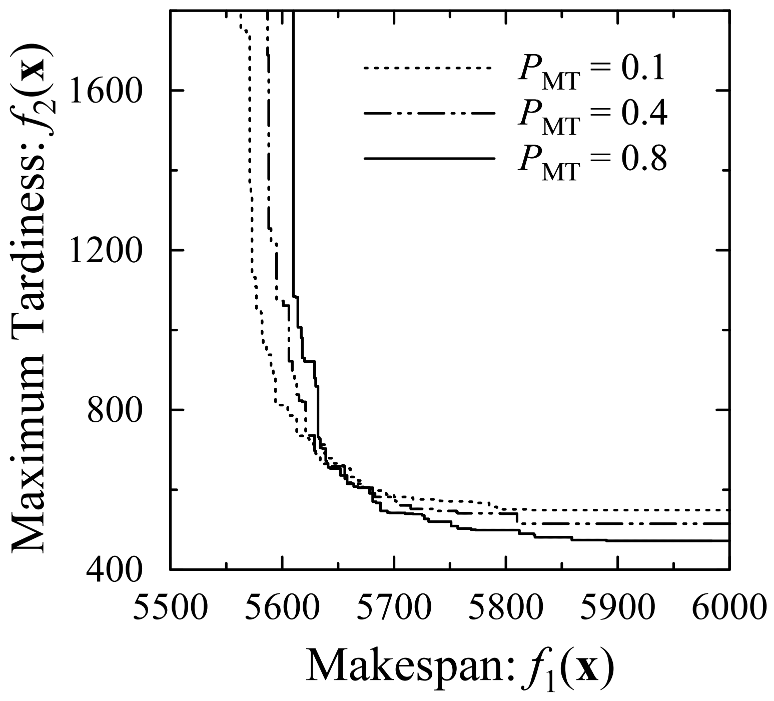


Fig. 6. Effect of the specification of P_{MT} on the search behavior by MOGLS-MT on the 2-80 flowshop scheduling problem.

4.4 Experimental Results on Large Test Problems

In this subsection, we examine the effect of the hybridization with local search on the performance of NSGA-II through computational experiments on large knapsack problems. In Subsection 4.2, we used the same knapsack problems as those in Zitzler and Thiele (1999). The test problems had two, three and four objectives and 250, 500 and 750 items. In the same manner as in Zitzler and Thiele (1999), we generated larger test problems with 1500 items (i.e., 2-1500, 3-1500 and 4-1500). We also generated six-objective test problems (i.e., 6-250, 6-500, 6-750, 6-1500). We applied NSGA-II, S-MOGLS, MOGLS-WR and MOGLS-BF (i.e., BF(20)) to these newly generated larger test problems. We used the same parameter specifications as those for the 4-750 problem in Table 1.

Experimental results are summarized in Table 14 for the $D1_R$ measure. We can see that the performance of NSGA-II and S-MOGLS was drastically improved by biasing the neighborhood structure in MOGLS-WR for all test problems. We can also see that the performance of MOGLS-WR was further improved by concentrating local search on only 20 promising items in MOGLS-BF for almost all test problems (except for 2-1500). From Table 14, we can conclude that MOGLS-WR and MOGLS-BF worked very well on large knapsack problems with many items and many objectives in comparison with NSGA-II.

Table 14. $D1_R$ measure for large knapsack problems where the same parameter specifications as those for the 4-750 test problem in Table 1 are used.

Problem	NSGA-II	MOGLS		
		S	WR	BF
2-1500	3541.1	3414.2	**76.8**	286.9
3-1500	5248.5	5251.0	784.0	**501.2**
4-1500	6106.4	6022.6	1476.6	**1085.2**
6-250	816.9	766.3	395.7	**377.1**
6-500	1901.5	1819.3	776.2	**716.2**
6-750	3258.5	3032.1	1140.8	**1044.7**
6-1500	7870.4	7232.5	2065.2	**1813.9**

4.5 Parameter Specifications in Local Search

In our previous computational experiments, we always used the following parameter specifications in local search:

Local search application probability: $P_{LS} = 0.1$,

Upper bound on the number of successive failure attempts: $L_{fail} = 5$,

Upper bound on the total number of attempted local moves: $L_{search} = 20$.

In this subsection, we examine the sensitivity of the performance of S-MOGLS and MOGLS-WR on the specifications of these parameters through computational experiments on the 2-500 knapsack problem.

We examined $5 \times 8 \times 8$ combinations of the following parameter values:

$P_{LS} = 0.0, 0.1, 0.2, 0.5, 1.0,$

$L_{fail} = 0, 1, 2, 5, 10, 20, 50, 100,$

$L_{search} = 0, 10, 20, 50, 100, 200, 500, 1000.$

It should be noted that S-MOGLS and MOGLS-WR are the same as NSGA-II when at least one of these three parameters is zero. It should be also noted that L_{fail} is never activated as the stopping condition when L_{fail} is larger than L_{search}.

Experimental results by S-MOGLS with $P_{\text{LS}} = 0.1$ are summarized in Fig. 7 where the average value of the D1_R measure was calculated over 30 runs for each combination of L_{fail} and L_{search}. We can see from Fig. 7 that the performance of NSGA-II (i.e., S-MOGLS with $L_{\text{fail}} = 0$ and/or $L_{\text{search}} = 0$) was improved by the hybridization with local search when L_{fail} and L_{search} are appropriately specified (e.g., $L_{\text{fail}} \geq 5$ and $L_{\text{search}} = 10$, $L_{\text{fail}} = 10$ and $L_{\text{search}} \geq 10$). The combination of $L_{\text{fail}} = 5$ and $L_{\text{search}} = 20$ in our previous computational experiments is not an optimal setting but a good one in Fig. 7. We can also observe a negative effect of the hybridization with local search when L_{fail} and L_{search} are too large (i.e., around the top-right corner with $L_{\text{fail}} = 100$ and $L_{\text{search}} = 1000$). In this case, too much local search is performed in S-MOGLS. For example, at least 100 neighbors are examined in a series of local search from a single initial solution when $L_{\text{fail}} = 100$ and $L_{\text{fail}} \leq L_{\text{search}}$.

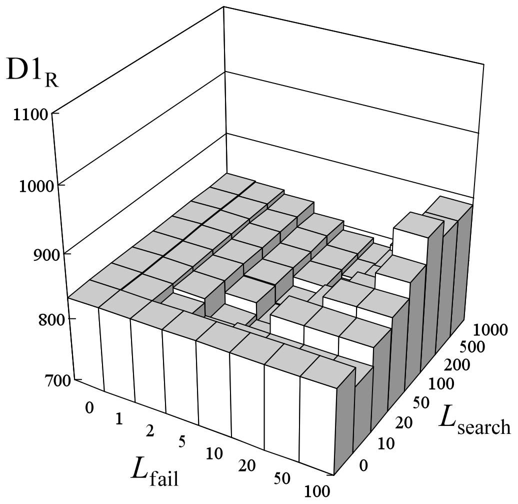


Fig. 7. D1_R measure by S-MOGLS with $P_{\text{LS}} = 0.1$ for the 2-500 knapsack problem.

The negative effect of the hybridization with local search becomes more prominent when we use a larger value of the local search application probability

P_{LS}. For example, we show experimental results by S-MOGLS with $P_{\mathrm{LS}} = 0.2$ in Fig. 8 (whereas $P_{\mathrm{LS}} = 0.1$ in Fig. 7). We can observe the negative effect of the hybridization more clearly in Fig. 8. From the comparison between Fig. 7 and Fig. 8, we can also see that the range of appropriate specifications of L_{fail} and L_{search} moved toward their smaller values in Fig. 8 from Fig. 7. This is because we used a larger value of the local search application probability P_{LS} in Fig. 8 than Fig. 7. Since local search was more frequently used in Fig. 8, its execution should be terminated earlier than the case of Fig. 7 in order to strike a good balance between local search and genetic search.

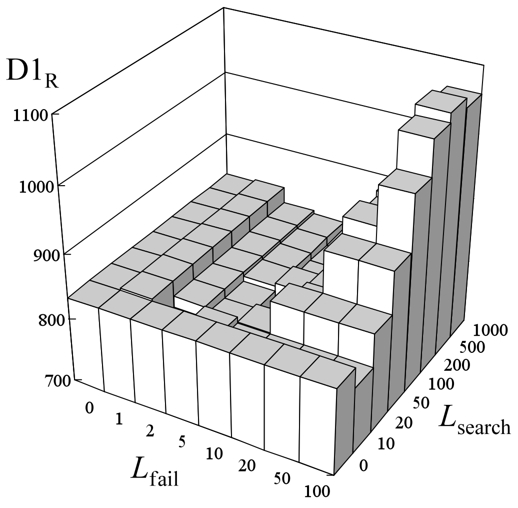


Fig. 8. $\mathrm{D1_R}$ measure by S-MOGLS with $P_{\mathrm{LS}} = 0.2$ for the 2-500 knapsack problem.

When we used MOGLS-WR, much better results were obtained from MOGLS-WR than NSGA-II independent of parameter specifications in local search. We show experimental results by MOGLS-WR in Fig. 9 for $P_{\mathrm{LS}} = 0.1$ and Fig. 10 for $P_{\mathrm{LS}} = 1.0$. It should be noted that the scale of the vertical axes of Fig. 9 and Fig. 10 is smaller than that of Fig. 7 and Fig. 8 by an order of magnitude. The average value of the $\mathrm{D1_R}$ measure by MOGLS-WR was always smaller than 120 in Fig. 9 and Fig. 10 whereas it was larger than 800 by NSGA-II in Fig. 7 and Fig. 8 (i.e., by S-MOGLS with $L_{\mathrm{fail}} = 0$ and/or $L_{\mathrm{search}} = 0$). Whereas much better results were always obtained by MOGLS-WR than NSGA-II, we can also observe the negative effect of too much local search in Fig. 10 around its top-right corner with $L_{\text{fail}} = 100$ and $L_{\text{search}} = 1000$. This observation suggests that a good balance between local search and genetic search is necessary for implementing MOMAs with high search ability.

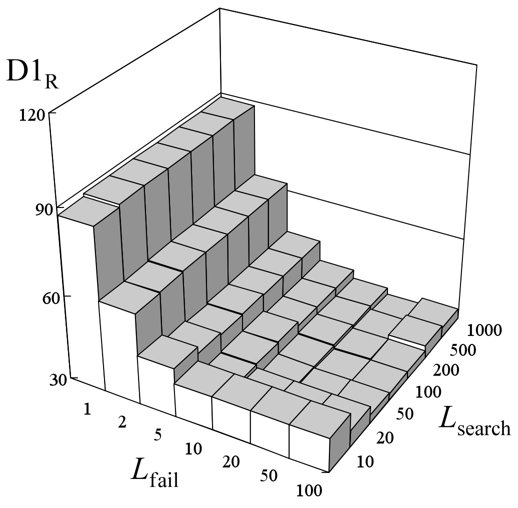


Fig. 9. $D1_R$ measure by MOGLS-WR with $P_{\text{LS}} = 0.1$ for the 2-500 knapsack problem.

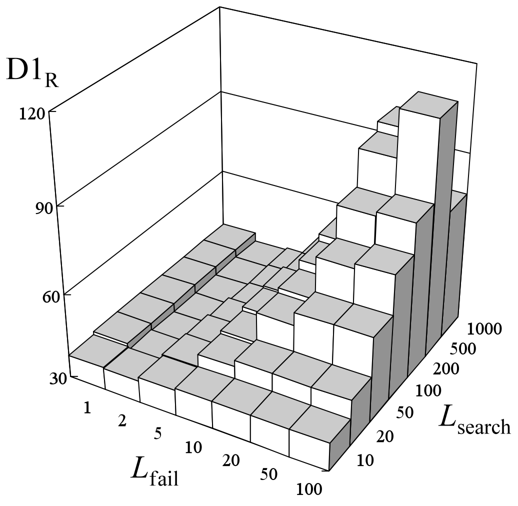


Fig. 10. $D1_R$ measure by MOGLS-WR with $P_{\text{LS}} = 1.0$ for the 2-500 knapsack problem.

5. Concluding Remarks

In this paper, we proposed the use of biased neighborhood structures in local search for improving the search ability of multiobjective memetic algorithms

(MOMAs). We implemented this idea for multiobjective 0/1 knapsack and flowshop scheduling problems. The effectiveness of this idea was demonstrated through computational experiments. That is, the use of biased neighborhood structure improved the performance of MOMAs. It was also shown that the performance of NSGA-II was improved by the hybridization with local search. This suggests that MOMAs can outperform multiobjective evolutionary algorithms (MOEAs).

Since this paper is the first attempt toward the use of biased neighborhood structures in local search for MOMAs, their implementation was not so sophisticated. As shown by computational experiments on multiobjective 0/1 knapsack problems, the performance of MOMAs can be improved by the use of an adjusted neighborhood structure for each problem (i.e., adjusted to the problem type and the problem size). In this sense, our implementation of biased neighborhood structures can be further improved. Especially, the biased neighborhood structure for multiobjective flowshop scheduling problems was implemented only for improving the second objective (i.e., the maximum tardiness). By using the other objectives to bias the neighborhood structure, we will obtain better MOMAs with higher search ability for multiobjective flowshop scheduling problems. The implementation of such a more sophisticated biased neighborhood structure is left for future research. This paper, however, clearly demonstrated the effectiveness of the use of biased neighborhood structures in MOMAs while their implementation was not so sophisticated.

As we explained in this paper, we need problem-specific knowledge to bias the neighborhood structure for each problem. Thus the effectiveness of the biased neighborhood structure depends on the usefulness of the utilized problem-specific knowledge. In this sense, it is difficult to generalize the performance improvement in our computational experiments on multiobjective knapsack and flowshop scheduling problems to other problems. However, there exist efficient heuristic techniques in many single-objective combinatorial optimization problems. Such a heuristic technique can be utilized to bias neighborhood structures for multiobjective optimization problems as we explained for flowshop scheduling problems in this paper.

Recently multiobjective optimization has been actively tackled by not only evolutionary algorithms but also other meta-heuristic approaches. For example, Rahimi-Vahed and Mirghorbani (2007) used a particle swarm optimization (PSO) algorithm for two-objective flow shop scheduling. In their computational experiments, better results were obtained from their multiobjective PSO than SPEA2 (Zitzler et al. 2001). On the other hand, Doerner et al. (2004) applied an ant colony optimization (ACO) algorithm to multiobjective portfolio selection problems with the same coding scheme as multiobjective knapsack problems. They demonstrated that better results were obtained by their multiobjective ACO than NSGA-II (Deb et al. 2002) and Pareto simulated annealing (Czyzak and Jaszkiewicz 1998). An important future research issue is the performance comparison of various meta-heuristic approaches to multiobjective optimization such as evolutionary algorithms, memetic algorithms, PSO and ACO.

This work was partially supported by Japan Society for the Promotion of Science (JSPS) through Grand-in-Aid for Scientific Research (B): KAKENHI (20300084).

References

Caponio A, Cascella GL, Neri F, Salvatore N, Sumner M (2007) A fast adaptive memetic algorithm for online and offline control design of PMSM drives. IEEE Trans. on Systems Man and Cybernetics - Part B: 37: 28-41

Coello CAC, Lamont GB (2004) Applications of Multi-Objective Evolutionary Algorithms. World Scientific, Singapore

Coello CAC, van Veldhuizen DA, Lamont GB (2002) Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, Boston

Czyzak P, Jaszkiewicz A (1998) Pareto simulated annealing - A metaheuristic technique for multiple objective combinatorial optimization. Journal of Multi-Criteria Decision Analysis 7: 34-47

Deb K (2001) Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Chichester

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation 6: 182-197

Doerner K, Gutjahr WJ, Hartl RF, Strauss C, Stummer C (2004) Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. Annals of Operations Research 131: 79-99

Fonseca CM, Fleming PJ (1996) On the performance assessment and comparison of stochastic multiobjective optimizers. Lecture Notes in Computer Science 1141: 584-593

Guo XP, Yang GK, Zhiming W, Huang ZH (2006) A hybrid fine-timed multi-objective memetic algorithm. IEICE Trans. on Fundamentals of Electronics Communications and Computer Sciences E89A: 790-797

Ishibuchi H, Kaige S, Narukawa K (2005) Comparison between Lamarckian and Baldwinian repair on multiobjective 0/1 knapsack problems. Lecture Notes in Computer Science 3410: 370-385

Ishibuchi H, Murata M (1998) A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans. on Systems, Man, and Cybernetics - Part C 28: 392-403

Ishibuchi H, Narukawa K (2004) Some issues on the implementation of local search in evolutionary multiobjective optimization. Lecture Notes in Computer Science 3102: 1246-1258

Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Trans. on Evolutionary Computation 7: 204-223

Jakob W (2006) Towards an adaptive multimeme algorithm for parameter optimisation suiting the engineers' needs. Lecture Notes in Computer Science 4193: 132-141

Jaszkiewicz A (2001) Comparison of local search-based meta-heuristics on the multiple objective knapsack problem. Foundations of Computing and Decision Sciences 26: 99-120

Jaszkiewicz A (2002a) Genetic local search for multi-objective combinatorial optimization. European Journal of Operational Research 137: 50-71

Jaszkiewicz A (2002b) On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - A comparative experiment. IEEE Trans. on Evolutionary Computation 6: 402-412

Jaszkiewicz A (2004) On the computational efficiency of multiple objective metaheuristics: The knapsack problem case study. European Journal of Operational Research 158: 418-433

Knowles JD, Corne DW (2000a) M-PAES: A memetic algorithm for multiobjective optimization. In: Proceedings of 2000 Congress on Evolutionary Computation. pp 325-332

Knowles JD, Corne DW (2000b) A comparison of diverse approaches to memetic multiobjective combinatorial optimization. In: Proceedings of 2000 Genetic and Evolutionary Computation Conference Workshop Program. pp. 103-108

Knowles JD, Corne DW (2002) On metrics for comparing non-dominated sets. In: Proceedings of 2002 Congress on Evolutionary Computation. pp. 711-716

Knowles JD, Corne DW (2005) Memetic algorithms for multi-objective optimization: Issues, methods and prospects. In Hart WE, Krasnogor N, Smith JE (eds) Recent Advances in Memetic Algorithms, Springer, Berlin, pp 313-352

Krasnogor N, Blackburnem B, Hirst JD, Burke EK (2002) Multimeme algorithms for protein structure prediction. Lecture Notes in Computer Science 2439: 769-778

Krasnogor N, Smith JE (2005) A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. IEEE Trans. on Evolutionary Computation 9: 474-488

Moscato P (1999) Memetic algorithms: A short introduction. In Corne D, Dorigo M, Glover F (eds) New Ideas in Optimization, McGraw-Hill, Maidenhead, pp 219-234

Murata T, Ishibuchi H, Gen M (2001) Specification of genetic search directions in cellular multi-objective genetic algorithm. Lecture Notes in Computer Science 1993: 82-95

Murata T, Ishibuchi H, Tanaka T (1996) Genetic algorithms for flowshop scheduling problems. Computer and Industrial Engineering 30: 1061-1071.

Murata T, Kaige S, Ishibuchi H (2003) Generalization of dominance relation-based replacement rules for memetic EMO algorithms. Lecture Notes in Computer Science 2723: 1233-1244

Ong YS, Keane AJ (2004) Meta-Lamarckian learning in memetic algorithms. IEEE Trans. on Evolutionary Computation 8: 99-110

Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: A comparative study. IEEE Trans. on Systems Man and Cybernetics - Part B 36: 141-152

Rahimi-Vahed AR, Mirghorbani SM (2007) A multi-objective particle swarm for a flow shop scheduling problem. Journal of Combinatorial Optimization 13: 79-102

Smith JE (2007) Coevolving memetic algorithms: A review and progress report. IEEE Trans. on Systems, Man, and Cybernetics - Part B: 37: 6-17

Van Veldhuizen DA (1999) Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations. Ph. D dissertation, Air Force Institute of Technology, Dayton, USA

Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Trans. on Evolutionary Computation 3: 257-271

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-Report 130, Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland

Zitzler E, Thiele L, Laumanns M, Fonseca CM, da Fonseca VG (2003) Performance assessment of multiobjective optimizers: An analysis and review. IEEE Trans. on Evolutionary Computation 7: 117-132