A Multi-objective Evolutionary Algorithm Based on Decomposition

Technical Report CSM-450

Qingfu Zhang
Department of Computer Science
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ, U.K.
qzhang@essex.ac.uk

Hui Li
Department of Computer Science
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ, U.K.
hlil@essex.ac.uk

May, 2006

Abstract

Decomposition is a basic strategy in traditional multiobjective optimization. However, this strategy has not yet widely used in multiobjective evolutionary optimization. This paper proposes a multiobjective evolutionary algorithm based on decomposition (MOEA/D). It decomposes a MOP into a number of scalar optimization subproblems and optimizes them simultaneously. Each subproblem is optimized by using information from its several neighboring subproblems, which makes MOEA/D have lower computational complexity at each generation than MOGLS and NSGA-II. Experimental results show that it outperforms or performs similarly to MOGLS and NSGA-II on multiobjective 0-1 knapsack problems and continuous multiobjective optimization problems.

Index Terms

multiobjective optimization, decomposition, evolutionary algorithms, memetic algorithms, Pareto optimality, computational complexity.

I. Introduction

A multiobjective optimization problem (MOP) can be stated as follows:

$$\begin{aligned} \text{maximize} \quad & F(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T \\ \text{subject to} \quad & x \in S \end{aligned} \tag{1}$$

where S is the decision (variable) space, $F : S \rightarrow R^m$ consists of m real-valued objective functions and R^m is called the objective space. Very often, since these objectives contradict each other, no point in S minimizes all the objectives simultaneously. One has to balance these objectives. The best tradeoffs among these objectives can be defined in terms of Pareto optimality.

Let $u, v \in R^m$, u is said to *dominate* v if and only if $u_i \geq v_i$ for every $i \in \{1, 2, \ldots, m\}$ and $u_j > v_j$ for at least one index $j \in \{1, 2, \ldots, m\}$[1]. A point $x^* \in S$ is *Pareto optimal* to (1) if there is no point x such that $F(x)$ dominates $F(x^*)$. $F(x^*)$ is then called a *Pareto optimal (objective) vector*. The set of all the Pareto optimal points is called the *Pareto set (PS)* and the set of all the Pareto optimal objective vectors is the *Pareto front (PF)* [1].

In many real-life applications of multiobjective optimization, an approximation to the PF is required by a decision maker for selecting a final preferred solution. Most MOPs may have many or even infinite Pareto optimal vectors. It is very time-consuming, if not impossible, to obtain the complete PF. On the other hand, the decision maker may not be interested to have an unduly huge number of Pareto optimal vectors to deal with due to overflow of information. Therefore, a number of current MOP algorithms are to find a manageable number of Pareto optimal vectors which are uniformly distributed along the PF and thus good representatives of the entire PF [2] [3] [4] [5] [6]. Some researchers have also made an effort to approximate the PF by using a mathematical model [7] [8] [9] [10].

It is well-known that a Pareto optimal solution for a MOP, under mild conditions, could be an optimal solution of a scalar optimization problem in which the objective is an aggregation of all the f_i's. Therefore, approximation of the PF can be decomposed into a number of scalar objective optimization subproblems. This is a basic idea behind many traditional mathematical programming methods for approximating the PF. Several methods for constructing aggregation functions can be found in the literature [1]. The most popular ones among them include the weight sum approach and Tchebycheff approach [11] [12] [13]. Recently, the normal-boundary intersection method has also attracted a lot of attention [14].

There is no decomposition involved in the majority of the current multiobjective evolutionary algorithms (MOEAs). These algorithms treat a MOP as a whole. They do not associate each individual solution with any particular scalar optimization problem. In scalar objective optimization problems, all the solutions can be compared based on their objective function values and the task of a scalar objective EA is to find one optimal solution. In MOPs, however, domination does not define a complete ordering among the solutions in the objective space and MOEAs aim at producing a number of Pareto optimal solutions as diverse as possible for representing the whole PF. Therefore, conventional selection operators, which were originally designed for scalar optimization evolutionary algorithms (EAs), cannot be directly used in non-decomposition MOEAs. Arguably, if there is a fitness assignment scheme for assigning an individual solution a relative fitness value to reflect its utility for selection, then scalar optimization EAs can be readily extended for dealing with MOPs, although other techniques such as mating restriction, diversity maintaining and external populations may also be needed for enhancing the performances of these extended algorithms [15] [16]. For this reason, fitness assignment has been a major issue in current MOEA research. The popular fitness assignment strategies include alternating objectives based fitness assignment such as VEGA [17], and domination-based fitness assignment such as NSGA [18], SPEA [3] and PAES [19].

The idea of decomposition has been used to certain extent in several metaheuristics for MOPs [20] [21]. For example, the two-phase local search (TPLS) considers a set of scalar optimization problems, in which the objectives are aggregations of the objectives in the MOP under consideration, a scalar optimization algorithm is applied to these scalar optimization problems in

[1]This definition of domination is for maximization. All the inequalities should be reversed if the goal is to minimize the objectives in (1). "dominate" means "be better than".

a sequence based on aggregation coefficients, a solution obtained in the previous problem is set as a starting point for solving the next problem since its aggregation objective is just slightly different from that in the previous one. The multiobjective genetic local search (MOGLS) aims at simultaneous optimization of all aggregations constructed by the weight sum approach or Tchebycheff approach [12]. At each iteration, it optimizes a random selected aggregation objective.

In this paper, we propose a new multiobjective evolutionary algorithm based on decomposition (MOEA/D). MOEA/D explicitly decomposes the MOP (1) into N scalar optimization subproblems. It solves these subproblems simultaneously by evolving a population of solutions. The procedures for solving these subproblems collaborate with each other by exchanging information at each generation. MOEA/D has the following features:

- Each scalar optimization problem (i.e, scalar aggregation function) has its best solution found so far in the current population. Each scalar optimization problem has several neighboring subproblems. The optimal solutions to two neighboring subproblems should be similar. Each subproblem is optimized in MOEA/D by using information from its neighboring subproblems.
- MOEA/D has lower computational complexity at each generation than NSGA-II and MOGLS.
- It is very natural to use scalar optimization methods in MOEA/D since each solution is associated with a scalar optimization problem.

The paper is organized as follows. Section II introduces two popular decomposition approaches for MOPs. Section III presents a framework of MOEA/D. Section IV and V compare MOEA/D with MOGLS and NSGA-II and show that MOEA/D outperforms or performs similarly to NSGA-II and MOGLS on a set of MOP test instances. Section VI concludes this paper.

II. Decomposition of Multi-objective Optimization Problems

There are several approaches for converting the problem of approximation of the PF into a number of scalar optimization problems. In the following, we introduce two very popular and simple approaches, which are used in our experimental studies.

A. Weight Sum Approach [1]

This approach considers a convex combination of the different objectives. Let $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)^T$ be a weight vector, i. e., $\lambda_i \geq 0$ for all $i = 1, 2, \ldots, m$ and $\sum_{i=1}^{m} \lambda_i = 1$. Then the optimal solution to the following scalar optimization problem

$$\begin{array}{ll} \text{maximize} & g^{ws}(x|\lambda) = \sum_{i=1}^{m} \lambda_i f_i(x) \\ \text{subject to} & x \in S \end{array} \tag{2}$$

is a Pareto optimal point for (1)[2], where we use $g^{ws}(x|\lambda)$ to emphasize that λ is a coefficient vector in this objective function while x is the variables to be optimized. To generate a set of different Pareto optimal vectors, one can use different weight vectors λ in the above scalar optimization problem. If the PF is concave (convex in the case of minimization), this approach could work well. However, not every Pareto optimal vector can be obtained by this approach in the case of nonconcave PFs. To overcome these shortcomings, some effort has been made to incorporate other techniques such as ε-constraint into this approach.

B. Tchebycheff Approach [1]

In this approach, the scalar optimization problem is in the form

$$\begin{array}{ll} \text{minmize} & g^{te}(x|\lambda, z^*) = \max_{1 \leq i \leq m}[\lambda_i |f_i(x) - z_i^*|], \\ \text{subject to} & x \in S, \end{array} \tag{3}$$

where $z^* = (z_1^*, \ldots, z_m^*)^T$ is the ideal vector, i. e., $z_i^* = \max\{f_i(x)|x \in S\}$ [3] for each $i = 1, \ldots, m$. For each Pareto optimal point x^* there exists a weight vector λ such that x^* is the optimal solution of (3) and each optimal solution of (3) is a Pareto optimal solution of (1). Therefore, one is able to obtain different Pareto optimal solutions by altering the weight vector. One weakness with this approach is that its aggregation function is not smooth for a continuous MOP.

The weight L_p-approach considers:

$$\begin{array}{ll} \text{minimize} & g^{L_p}(x|\lambda, z^*) = \|\lambda^T(F(x) - z^*)\|_p \\ \text{subject to} & x \in S, \end{array} \tag{4}$$

where $1 \leq p \leq +\infty$. It will become Tchebycheff approach in the case of $p = +\infty$.

Both the weight sum approach and Tchebycheff approach may decompose the approximation of the PF into a number of scalar optimization problems. Under mild conditions, a reasonable large number of evenly distributed weight vectors usually lead to a set of Pareto optimal vectors, which may not be evenly spread but could approximate the PF very well.

[2] If (1) is for minimization, "maximize" in (2) should be changed to "minimize"

[3] In the case when the goal of (1) is minimization, $z_i^* = \min\{f_i(x)|x \in S\}$)

III. The Framework of Multiobjective Evolutionary Algorithm Based on Decomposition

A. General Framework

Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D), the algorithm proposed in this paper, needs to decompose the MOP under consideration. Any decomposition approaches, such as the weight sum approach and Tchebycheff approach, can serve for this purpose. In the following, we suppose that the Tchebycheff approach is employed.

Let $\lambda^1, \ldots, \lambda^N$ be a set of even spread weight vectors and z^* be the ideal vector. As shown in Section II, the problem of approximation of the PF of (1) can be decomposed into N scalar optimization subproblems by using the Tchebycheff approach and the objective function of the j-th subproblem is:

$$g^{te}(x|\lambda^j, z^*) = \max_{1 \le i \le m} [\lambda_i^j | f_i(x) - z_i^*|] \tag{5}$$

where $\lambda^j = (\lambda_1^j, \ldots, \lambda_m^j)^T$. MOEA/D minimizes all these N objective functions simultaneously in a single run.

Note that g^{te} is continuous of λ, the optimal solution of $g^{te}(x|\lambda^i, z^*)$ should be close to that of $g^{te}(x|\lambda^j, z^*)$ if λ^i and λ^j are close to each other. Therefore, any information about these g^{te}'s with weight vectors close to λ^i should be helpful for optimizing $g^{te}(x|\lambda^i, z^*)$. This is a major motivation behind MOEA/D.

At each generation t, MOEA/D with the Tchebycheff approach maintains:

- a population of N points $x^1, \ldots, x^N \in S$, where x^i is the current solution to the i-th subproblem;
- $FV^1, \ldots, FV^N$, where FV^i is the F-value of x^i, i.e., $FV^i = F(x^i)$;
- $z = (z_1, \ldots, z_m)^T$, where z_i is the best value found so far for objective f_i;
- an external population EP, which is used to store nondominated solutions found during the search.

The algorithm works as follows:

Input:
- MOP (1);
- a stopping criterion;
- N : the number of the subproblems considered in MOEA/D;
- an uniform spread of N weight vectors: $\lambda^1, \ldots, \lambda^N$;
- T : the number of the weight vectors in the neighborhood of each weight vector.

Output: EP.

Step 1 Initialization

Step 1.1 Set $EP = \emptyset$.

Step 1.2 For each $i = 1, \ldots, N$, set $B(i) = \{i_1, \ldots, i_T\}$ where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closet weight vectors to λ^i.

Step 1.3 Generate an initial population $x^1, \ldots, x^N$ randomly or by a problem-specific method and initialize $z = (z_1, \ldots, z_m)^T$. Set $FV^i = F(x^i)$.

Step 2 Update

For $i = 1, \ldots, N$, do

Step 2.1 Reproduction : Randomly select two indexes k, l from $B(i)$, and then generate a new solution y from x^k and x^l by using genetic operators.

Step 2.2 Improvement: Apply a repair/improvement heuristic on y to produce y'.

Step 2.3 Update of z: For each $j = 1, \ldots, m$, if $f_j(y') > z_j$, then set $z_j = f_j(y')$.

Step 2.4 Update of Neighboring Solutions: For each index $j \in B(i)$, if $g^{te}(y'|\lambda^j, z) \le g^{te}(x^j|\lambda^j, z)$, then set $x^j = y'$ and $FV^j = F(y')$.

Step 2.5 Update of EP:

Remove from EP all the vectors dominated by $F(y')$.

Add $F(y')$ to EP if no vectors in EP dominates $F(y')$.

Step 3 Stopping Criteria If stopping criteria is satisfied, then stop and output EP. Otherwise go to **Step 2**.

In initialization, $B(i) = \{i_1, \ldots, i_T\}$ is computed for each index i. We always let $i \in B(i)$ and use the Euclidean distance to measure the closeness between any two weight vectors. If $j \in B(i)$, the j-th subproblem can be regarded as a neighbor of the i-th subproblem.

In the i-th pass of the loop in Step 2, T neighboring subproblems of the i-th subproblem are considered. Since x^k and x^j in Step 2.1 are the current solutions to neighbors of the i-th subproblem, their offspring y should hopefully be a good solution for the i-th subproblem. In Step 2.2, a heuristic is used to repair y in case when y invalidates any constraints, and/or optimize the i-th g^{te}. Therefore, the resultant solution y' is feasible and very likely to have a lower function value for the neighbors of i-th subproblem, Step 2.3 considers all the neighbors of the i-th subproblems, it replaces x^j with y' if y' performs better than x^j with regard to the j-th subproblem. Since it is often very time-consuming to find the actual ideal vector z^*, we use z, which is initialized in Step 1 and updated in Step 2.4, as a substitute for z^* in g^{te}. The external population EP is updated by the new generated solution y' in Step 2.5.

In the case when the goal in (1) is to minimize $F(x)$, the inequality in Step 2.3 should be reversed.

B. Variants of MOEA/D

It is obvious that we can use other decomposition methods such as the weight sum approach in MOEA/D. When the weight sum approach is used, MOEA/D does not need to maintain z.

Step 2.2 allows MOEA/D to be able to make use of a scalar optimization method very naturally. One can take the $g^{te}(x|\lambda^i, z)$ or $g^{ws}(x|\lambda^i)$ as the objective function in Step 2.2. Although it is one of the major features of MOEA/D, Step 2.2 is not a must in MOEA/D, particularly if Step 2.1 can produce a feasible solution.

Using the external population EP is also an option, although it is often very helpful for improving the performance of the algorithm. An alternative is to return the final population as an approximation to the PF when EP is not maintained. Of course, other sophisticated strategies [5] [6] for updating EP can also be easily adopted in the framework of MOEA/D.

IV. Comparison with MOGLS

In the following, we discuss the difference and similarity between MOEA/D and MOGLS. We also compare the performances of these two algorithms on a set of test instances of the multiobjective 0/1 knapsack problem. The major reason for choosing MOGLS for comparison is that it is also based on decomposition and performs better than a number of popular algorithms on the multiobjective 0/1 knapsack problem [12].

A. MOGLS

MOGLS was first proposed by Ishibuchi and Murata in [11], and further improved by Jaszkiewicz [12]. The basic idea is to reformulate the MOP (1) as simultaneous optimization of all weighted Tchebycheff functions or all weighted sum functions. In the following, we give a brief description of Jaszkiewicz's version of MOGLS.

At each iteration, MOGLS maintains:

- a set of current solutions CS, and the F-values of these solutions;
- an external population EP, which is used to store nondominated solutions.

If MOGLS optimizes weighted Tchebycheff functions, it also should maintain:

- $z = (z_1, \ldots, z_m)^T$, where z_i is the biggest value found so far for objective f_i.

MOGLS needs two control parameters K and S. K is the size of its temporary elite population and S is the number of initial solutions.

S solutions are initialized randomly or by a problem-specific heuristic and form the initial set CS. The nondominated solutions in CS constitute the initial EP. At each iteration, MOGLS with the Tchebycheff approach works as follows:

Step 1 Uniformly randomly choose a weight vector λ,
From CS select the K best solutions, with regard to the Tchebycheff aggregation function g^{te} with the weight vector λ, to form a temporary elite population TEP.
Draw at random two solutions from TEP and then generate a new solution y from these two solutions by genetic operators.
Step 2 Apply a repair/improvement heuristic on y to generate y'.
Step 3 For each $j = 1, \ldots, m$, if $f_j(y') > z_j$, then set $z_j = f_j(y')$.
Step 4 If y' is better than the worst solution in TEP with regard to g^{te} with the weight vector λ and different from any solutions in TEP with regard to F-values, then add it to the set CS.
If the size of CS is larger than $K \times S$, delete the oldest solution in CS.
Step 5 Remove from EP all the points dominated by $F(y')$.
Add $F(y')$ to EP if no point in EP dominates $F(y')$.

In the case of MOGLS with the weight sum approach, g^{te} should be replaced by g^{ws} and there is no need to store z. Therefore, Step 3 should be removed.

B. Difference and similarity between MOEA/D and MOGLS

1) Weight Vector: Both MOEA/D and MOGLS optimize an aggregation function at each iteration. A weight vector used in MOEA/D is one of the N preselected weight vectors, MOEA/D spends about the same amount of effort on each of the N aggregation functions, while MOGLS randomly chooses a weight vector at each iteration, aiming at optimizing all the aggregation functions.

2) Populations: During the search, MOEA/D needs to maintain an internal population of N solutions, and its external population EP, while MOGLS stores the set of current solutions CS and its external population EP. The size of CS increases until it reaches its upper bound, which is suggested to be set as $K \times S$. Therefore, if $K \times S$ in MOGLS is much larger than N in MOEA/D and both algorithms produce about the same number of nondominated solutions, then the space complexity of MOEA/D is lower than that of MOGLS.

3) Computational Complexity: The major computational cost in MOEA/D is involved in Step 2. Let us compare the computational complexity between a single pass of Step 2 in MOEA/D and an iteration in MOGLS:

- Step 2.1 in MOEA/D vs. Step 1 in MOGLS: MOGLS has to generate TEP, which needs $O(|CS| \times K)$ basic operations if a naive selection method is employed, while Step 2.1 in MOEA/D only needs to randomly select two solutions for genetic operators. Note that the size of CS, $|CS|$, could be very large (e.g, it was set from $3,000$ to $7,000$ in [3]), the computational cost of Step 1 in MOGLS is much higher than that of Step 2.1 in MOEA/D.
- Steps 2.2 and 2.3 in MOEA/D are the same as Steps 2 and 3 in MOGLS.
- Step 2.4 in MOEA/D vs Step 4 in MOGLS: Step 2.4 in MOEA/D needs $O(T)$ basic operations while Step 4 in MOGLS needs $O(K)$ basic operation. In the case when T and K are close as in our experiments, there is no significant difference in computational costs between them.

Therefore, we can conclude that each pass of Step 2 in MOEA/D involves less computational cost than each iteration in MOGLS does.

C. Multiobjective 0-1 Knapsack Problem

Given a set of n items and a set of m knapsacks, the multi-objective 0-1 knapsack problem (MOKP) can be stated as:

$$\begin{aligned} \text{maximize} \quad & f_i(x) = \sum_{j=1}^{n} p_{ij} x_j, \ i = 1, \ldots, m \\ \text{subject to} \quad & \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \ i = 1, \ldots, m \\ & x = (x_1, \ldots, x_n)^T \in \{0,1\}^n \end{aligned} \tag{6}$$

where $p_{ij} \geq 0$ is the profit of item j in knapsack i, $w_{ij} \geq 0$ is the weight of item j in knapsack i, and c_i is the capacity of knapsack i. $x_i = 1$ means that item i is selected and put in all the knapsacks.

The MOKP is NP-hard and can model a variety of applications in resource allocation. A set of nine test instances of the above problem have been proposed in [3] and widely used in testing multiobjective heuristics. MOGLS outperforms a number of MOEAs on these test instances [12]. In this paper, we will also use these nine instances for comparing the performances of MOEA/D and MOGLS.

D. Implementations of MOEA/D and MOGLS for MOKP

1) Repair Method: To apply an EA for the MOKP, one needs a heuristic for repairing infeasible solutions. Several repair approaches have been proposed for this purpose [22] [3] [12].

Let $y = (y_1, \ldots, y_n)^T \in \{0,1\}^n$ be an infeasible solution to (6). Note that w_{ij} and p_{ij} in (6) are nonnegative, one can remove some items from it (i.e., change the values of some y_i from 1 to 0) for making it feasible. Recently, Jaszkiewicz proposed and used the following greedy repair method in this latest work [4]:

Input:
- the constraints in (6);
- a solution: $y = (y_1, \ldots, y_n)^T$;
- an objective function: $g : \{0,1\}^n \rightarrow R$.

Output: a feasible solution $y' = (y'_1, \ldots, y'_n)^T$.

Step 1: If y is feasible, then set $y' = y$ and return y'.

Step 2: Set $J = \{j | 1 \leq j \leq n \text{ and } y_j = 1\}$ and $I = \{i | 1 \leq i \leq m \text{ and } \sum_{j=1}^{n} w_{ij} y_j > c_i, \}$

Step 3: Select $k \in J$ such that

$$k = \arg\min_{j \in J} \frac{g(y) - g(y^{j-})}{\sum_{i \in I} w_{ij}}$$

where y^{j-} is different from y only in position j, i.e., $y_i^{j-} = y_i$ for all $i \neq j$ and $y_j^{j-} = 0$.

Set $y_k = 0$ and go to Step 1.

In this approach, items are removed one by one from y until y becomes feasible. An item with the heavy weights in the overfilled knapsacks and little contribution to $g(x)$ is more likely to be removed.

[4]This approach is used in the latest version of his implementation, which can be downloaded from his web and is slightly better than that used in his earlier paper [23].

TABLE I
PARAMETER SETTING OF MOEA/D AND MOGLS FOR THE TEST INSTANCES OF THE 0/1 KNAPSACK PROBLEM

Instance		S	N (H)
m: # of objectives	n: # of items	in MOGLS	in MOEA/D
2	250	150	150 (149)
2	500	200	200 (199)
2	750	200	250 (249)
3	250	200	351 (25)
3	500	250	351 (25)
3	750	300	351 (25)
4	250	250	455 (12)
4	500	300	455 (12)
4	750	350	455 (12)

2) Implementation of MOGLS: To have a fair comparison, we directly use Jaszkiewicz's latest implementation of MOGLS for the MOKP, the details of MOGLS with the Tchebycheff approach are given as follows:

- **Initialization of $z = (z_1, \ldots, z_m)^T$:** Taking each f_i as the objective function, apply the repair method on a randomly generated point and produce a feasible solution. Set z_i to be the f_i value of the resultant point.
- **Initialization of EP and CS:**

  Set $EP = \emptyset$ and $CS = \emptyset$. Then **Repeat** S times:

  1. Randomly draw a weight vector λ by using the sampling method described in [12].

  2. Randomly generate a solution $x = (x_1, \ldots, x_n)^T \in \{0,1\}^n$, where the probability of $x_i = 1$ equals to 0.5.

  3. Taking $g^{te}(x|\lambda, z)$ as the objective function, apply the repair method to x and obtain a feasible solution x'.

  4. Add x' to CS. Remove from EP all the vectors dominated by $F(y')$, then add $F(y')$ to EP if no vectors in EP dominates $F(y')$.
- **Reproduction operator in Step 1:** The reproduction operator used is a combination of the one-point crossover operator and the standard mutation operator. The one-point crossover is first applied to the two solutions and generates one child solution, then the standard mutation operator mutates it to produce a new solution y. The mutation mutates each position of the child solution independently with probability 0.01.
- **Heuristic in Step 2:** The repair method described in this section is used.

3) Implementation of MOEA/D: We use the same reproduction operator and the repair method as in the implementation of MOGLS. More particularly, in MOEA/D with the Tchebycheff approach,

- **Initialization in Step 1.3** Initialization of z: Initialize z in the same way as in the implementation of MOGLS for the MOKP.
  Initialization of x^i (the initial solution to the $i-$th subproblem): Taking $g^{te}(x|\lambda^i, z)$ as the objective function, apply the repair method to a randomly generated solution. Set x^i to be the resultant solution.
- **Reproduction operator in Step 2.1:** It is the same as that used in Step 1 of the implementation of MOGLS for the MOKP.
- **Heuristic in Step 2.2:** The repair method described in this section is used.

Tchebycheff aggregation function g^{te} is used in the above implementations of MOEA/D and MOGLS. In the case when g^{ws} is used in the repair method, there is no need to initialize and maintain z.

E. Parameter Setting

The setting of S and K in MOGLS is the same as in [12]. K is set to 20 for all the instances. The values of S for different instances are given in Table I. T in MOEA/D is set to 10 for all the test instances. The setting of N and $\lambda^1, \ldots, \lambda^N$ in MOEA/D is controlled by a parameter H. $\lambda^1, \ldots, \lambda^N$ are all the weight vectors in which each individual weight takes a value from:

$$\{\frac{0}{H}, \frac{1}{H}, \ldots, \frac{H}{H}\}.$$

Therefore, the number of such vectors is:

$$N = C_{H+m-1}^{m-1}.$$

Table I lists the value of N and H in MOEA/D for each test instance. For the instances with 2 objectives, the value N is the same as that of S in MOGLS. For all the instances with three objective, $H = 25$ and therefore $N = 351$. For all the instances with four objectives, $H = 12$ and then $N = 455$. Both of the algorithms stop after $500 \times S$ calls of the repair method.

In our experimental studies, both g^{ws} and g^{te} have been used in the repair method. In the following, W-MOEA/D (W-MOGLS) stands for MOEA/D (MOGLS) in which g^{ws} is used, while T-MOEA/D (T-MOGLS) represents MOEA/D (MOGLS) in which g^{te} is used.

TABLE II
SET COVERAGE BETWEEN MOEA/D (A) AND MOGLS (B)

C-Metric (%)			C(A, B)		C(B,A)	
Decomposition Methods			g^{ws}	g^{te}	g^{ws}	g^{te}
Instance	2	250	45.6	93.4	40.3	3.9
	2	500	56.8	96.1	28.6	3.0
	2	750	74.0	97.1	15.6	2.6
	3	250	45.2	78.2	21.7	8.2
	3	500	61.0	92.8	10.4	1.9
	3	750	78.1	98.0	4.3	0.4
	4	250	19.3	28.8	26.2	26.5
	4	500	26.1	36.8	16.5	14.1
	4	750	41.1	57.6	8.7	5.4

TABLE III
GENERATIONAL DISTANCES FROM THE REFERENCE SOLUTIONS TO THE NONDOMINATED FRONTS FOUND BY MOEA/D AND MOGLS

D-Metric			MOEA/D		MOGLS	
Decomposition Methods			g^{ws}	g^{te}	g^{ws}	g^{te}
Instance	2	250	36.9	53.8	37.9	95.5
	2	500	78.7	184.2	98.4	330.6
	2	750	165.5	437.8	275.2	770.1
	3	250	97.5	158.5	141.2	216.0
	3	500	271.2	488.5	420.7	705.9
	3	750	446.9	963.6	775.9	1380.8
	4	250	176.4	253.5	265.8	301.5
	4	500	433.8	765.2	724.9	969.0
	4	750	761.0	1547.3	1242.6	2002.1

F. Experimental Results

Both MOGLS and MOEA/D have been independently run for 20 times for each test instance on identical computers (Pentium(R) 3.2GHZ, 1.00 GB of RAM). Due to the nature of MOPs, multiple performance indices should be used for comparing the performances of different algorithms [24] [25]. In our experiments, the following performance indices are used.

- **Set Coverage (C-metric):** Let A and B be two approximations to the PF of a MOP, $C(A, B)$ is defined as the percentage of the solutions in B that are dominated by at least one solution in A, i.e.,

$$C(A, B) = \frac{|\{u \in B | \exists v \in A : v \text{ dominates } u\}|}{|B|}$$

  $C(A, B)$ is not necessarily equal to $1 - C(B, A)$. $C(A, B) = 1$ means that all solutions in B are dominated by some solutions in A, while $C(A, B) = 0$ implies that no solution in B is dominated by a solution in A.
- **Distance from Representatives in the PF (D-metric):** Let P^* be a set of uniformly distributed points along the PF. Let A be an approximation to the PF, the average distance from P^* to A is defined as:

$$D(A, P^*) = \frac{\sum_{v \in P^*} d(v, A)}{|P^*|}$$

  where $d(v, A)$ is the minimum Euclidean distance between v and the points in A. If $|P^*|$ is large enough to represent the PF very well, $D(A, P^*)$ could measure both the diversity and convergence of A in a sense. To have a low value of $D(A, P^*)$, A must be very close to the PF and cannot miss any part of the whole PF.
  In the case when we don't know the actual PF, we can set P^* to be an upper approximation of the PF. Jaszkiewicz has produced a very good upper approximation to each 0/1 knapsack test instance by solving the linear programming relaxed version of (3) with a number of uniformly distributed λ's [12]. The number of the points in the upper approximation is 202 for each of the bi-objective instances, 1326 for the 3-objective instances, and 3276 for the 4-objectives. In our experiments, P^* is set as such an approximation.

Table IV gives the average CPU time of these algorithms for each instance.

Figures 1-3 show the evolution of the average D-value of EP from P^* in 20 runs with the number of the calls of the repair method in each algorithm for each of the test instance. Sine the ranges of D-values are large, we use the logarithmic scale for the axes of the average D-values in these figures.

Figures 4-5 illustrate the distributions of EP with the lowest D-metric value in a single run of each algorithm for the three bi-objective instances.

Table II presents the mean value of C-metric of the final approximations (EP obtained by the two algorithms with two different repair methods.

We can make the following observations:

TABLE IV
AVERAGE CPU TIME (S) USED BY MOEA/D AND MOGLS

Average CPU Time			MOEA/D		MOGLS	
Decomposition Methods			g^{ws}	g^{te}	g^{ws}	g^{te}
Instance	2	250	3.60	3.90	28.05	23.40
	2	500	9.30	10.00	73.20	68.10
	2	750	17.70	19.45	132.90	128.30
	3	250	6.45	6.75	76.45	78.00
	3	500	15.05	16.75	136.15	145.65
	3	750	26.40	30.40	202.80	217.95
	4	250	18.85	16.60	183.85	185.50
	4	500	43.00	40.25	288.95	290.70
	4	750	68.30	70.30	396.90	418.55

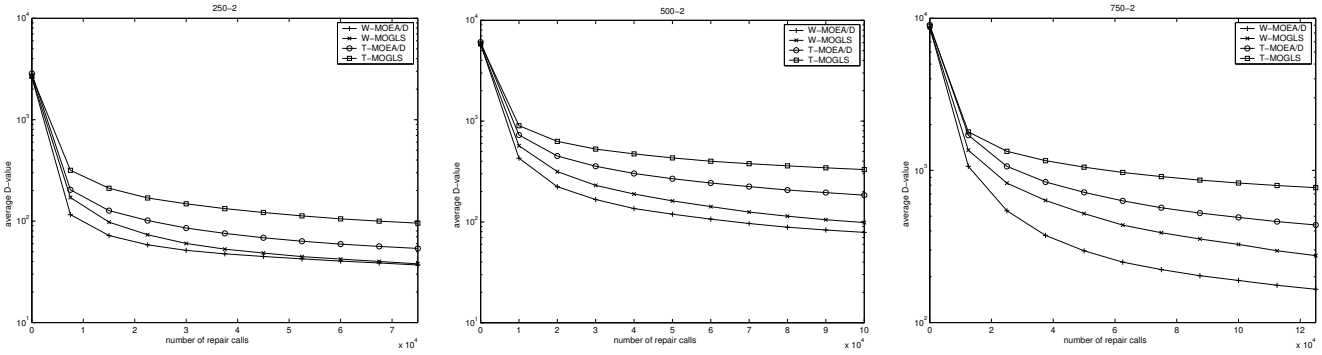

Fig. 1. The evolution of the average D-value obtained by MOEA/D and MOGLS for three 2-objective instances

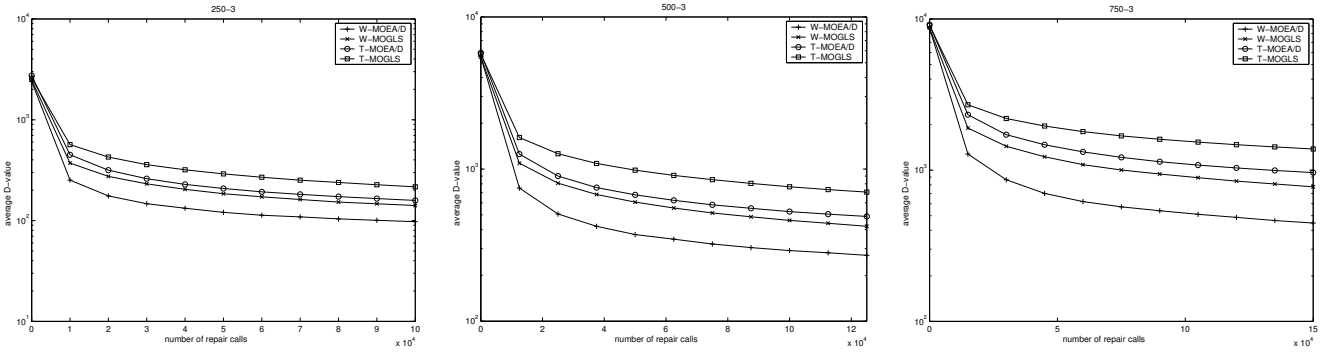

Fig. 2. The evolution of the average D-value obtained by MOEA/D and MOGLS for three 3-objective instances

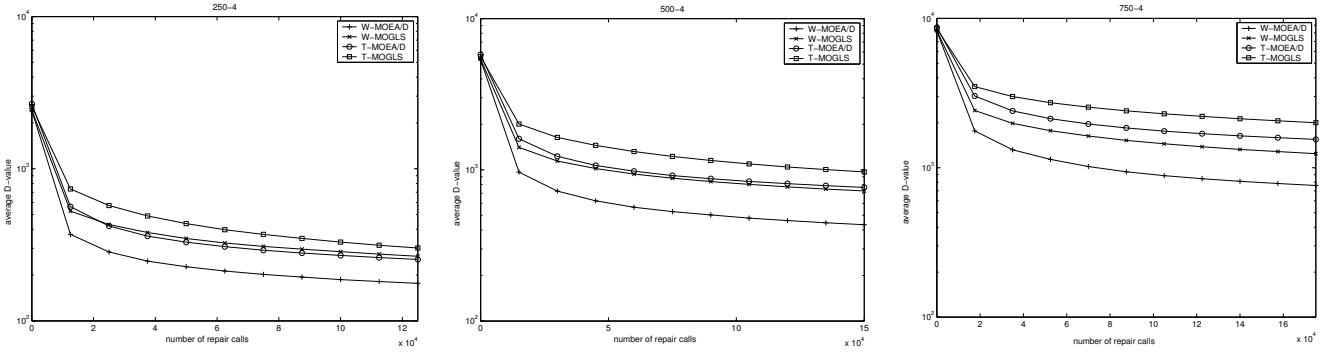

Fig. 3. The evolution of the average D-value obtained by MOEA/D and MOGLS for three 4-objective instances

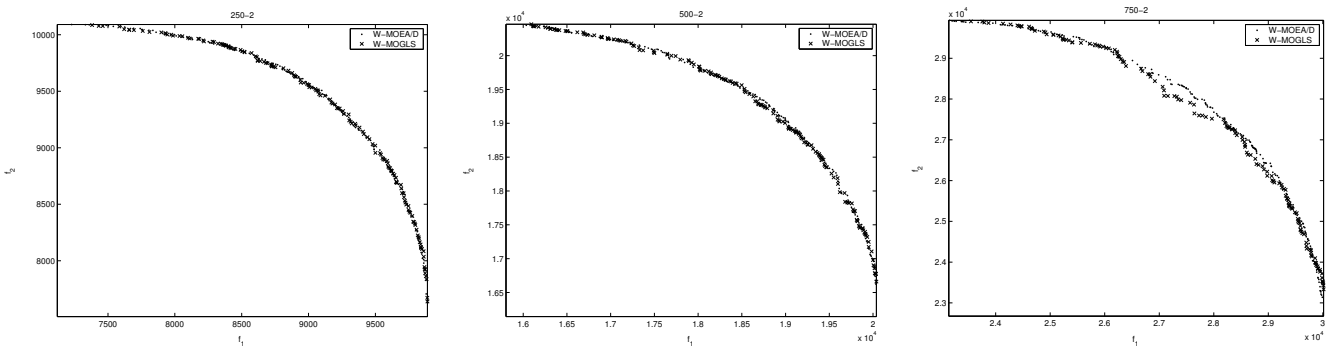

Fig. 4. Plots of the nondominated solutions found by MOEA/D and MOGLS with the weight sum approach

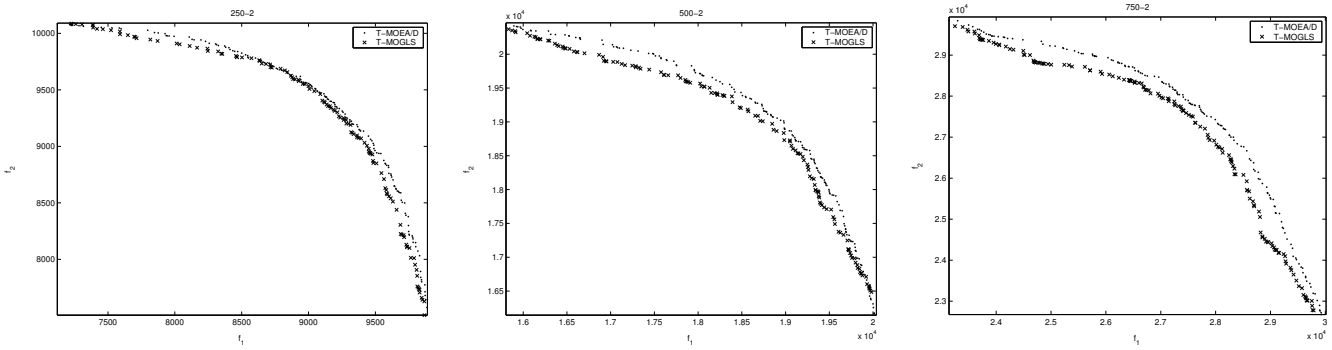

Fig. 5. Plots of the nondominated solutions found by MOEA/D and MOGLS with the Tchebycheff approach

- With the same number of the calls of repair method (i.e., the same number of trial solutions), it is evident from Table IV that MOEA/D needs less computational time than MOGLS does. Taking the instance 250-3 as an example, W-MOEA/D (T-MOEA/D) requires about 15% of the CPU time that W-MOGLS (T-MOGLS) needs. This observation agrees with our analysis of the computational complexity of MOEA/D and MOGLS in Section IV-B.
- Figures 1-3 clearly indicate that for all the test instances, W-MOEA/D (T-MOEA/D) is more efficient and effective than W-MOGLS (T-MOGLS) in minimizing the D-metric, which suggests that MOEA/D needs fewer calls of repair method than MOGLS for a knapsack problem.
- Table II and III show that the final EP obtained by W-MOEA/D (T-MOEA/D) is better than that obtained by W-MOGLS (T-MOGLS), in terms of both D-metric and C-metric, for all the test instances except the instance 250-4 in which W-MOEA/D is slightly worse than W-MOGLS in C-metric. The difference between the approximations by T-MOEA/D and T-MOGLS on the instances 250-2, 500-2 and 750-2 can be visually detected from Figure 5, while difference between W-MOEA/D and W-MOGLS in middle part of the fronts on the instances $500-2$ and $750-2$ can be spotted from Figure 4.
- Tables III and Figures 1-3 reveal that the weight sum approach outperforms the Tchebycheff approach in both MOEA/D and MOGLS, which suggests that different decomposition approaches in MOEA/D and MOGLS can have different performances.

V. Comparison with NSGA-II on Continuous MOPs

A. Multi-objective Continuous Test Suites

We use widely-used ZDT test instances of the continuous MOP [15] in comparing MOEA/D with NSGA-II [6], one of the most popular MOEAs. All these test instances are minimization of the objectives.

- ZDT1

$$\begin{aligned} f_1(x) &= x_1, \\ f_2(x) &= g(x)\left[1-\sqrt{f_1(x)/g(x)}\right] \end{aligned}$$

where

$$g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$$

and $x = (x_1, \ldots, x_n) \in [0,1]^n$. Its PF is convex. $n = 30$ in our experiments.

- ZDT2

$$\begin{aligned} f_1(x) &= x_1, \\ f_2(x) &= g(x)\left[1 - (f_1(x)/g(x))^2\right] \end{aligned}$$

where $g(x)$ and the range and dimensionality of x are the same as in ZDT1. The PF of ZDT2 is nonconvex.

- ZDT3

$$\begin{aligned} f_1(x) &= x_1, \\ f_2(x) &= g(x)\left[1 - \sqrt{f_1(x)/g(x)} - \frac{f_1(x)}{g(x)}\sin(10\pi x_1)\right] \end{aligned}$$

where $g(x)$ and the range and dimensionality of x are the same as in ZDT1. Its PF is disconnected.

- ZDT4

$$\begin{aligned} f_1(x) &= x_1, \\ f_2(x) &= g(x)\left[1 - \sqrt{f_1(x)/g(x)}\right] \end{aligned}$$

where

$$g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}\left[x_i^2 - 10\cos 4\pi x_i\right],$$

and $x_1 \in [0,1]$, and $x_i \in [-5,5], i = 2, \ldots, n$. Its PF is nonconvex and there are many local PFs. $n = 10$ in our experiments.

- ZDT6

$$\begin{aligned} f_1(x) &= 1 - \exp(-4x_1)\sin^6(6\pi x_1), \\ f_2(x) &= g(x)\left[1 - (f_1(x)/g(x))^2\right] \end{aligned}$$

where

$$g(x) = 1 + 9\left[(\sum_{i=2}^{n} x_i)/(n-1)\right]^{0.25},$$

and $x = (x_1, \ldots, x_n)^T \in [0,1]^n$. Its PF is nonconvex. The distribution of the Pareto solutions in the Pareto front is very nonuniform, i.e., For a set of uniformly distributed points in the Pareto set in the decision space, their images crowd in a corner of the Pareto front in the objective space. $n = 10$ in our experiments.

B. NSGA-II [6]

NSGA-II does not use an external population. NSGA-II maintains a population P_t of size N at generation t and generate P_{t+1} from P_t in the following way:

Step 1: Use selection, crossover and mutation to create an offspring population Q_t from P_t.

Step 2: Choose N best solutions from $P_t \bigcup Q_t$ to form P_{t+1}.

The characteristic feature of NSGA-II is that it uses a fast non-dominated sorting and crowded distance estimation procedure for comparing qualities of different solutions in **Step 2** and selection in **Step 1**. The computational complexity of each generation in NSGA-II is $O(mN^2)$, where m is the number of the objectives and N is its population size.

C. Variant of MOEA/D used in Comparison

We use the following variant of MOEA/D in comparison:

- There is no external population EP. Instead, the final internal population is returned as an approximation to the PF.
- No repair/improvement method is used, therefore, Step 2.2 is not needed.
- g^{ts} is used in the repair method in Step 2.4.

g^{ts} is used for decomposition. We do not use g^{ws} mainly because the weight sum approach are unable to deal with nonconvex PFs as it is the case in the test instances except ZDT1. Since NSGA-II has no external population, we do not use external population in MOEA/D to have a fair comparison. In comparison with NSGA-II, the only extra memory requirement in the variant of MOEA/D used in our comparison is for storing z, which is not large.

In a similar way as in Section IV-B, we can easily work out that the computational complexity of each pass of Step 2 in the above variant of MOEA/D is $O(N(T+m))$, where N is the size of its external population. Since T and m are smaller than N, this variant of MOEA/D has lower computational complexity than NSGA-II at each generation if they use the same size of population.

TABLE V
SET COVERAGE BETWEEN NSGA-II AND MOEA/D

C-Metric (%)		C(MOEA/D,NSGA-II)	C(NSGA-II, MOEA/D)
Instance	ZDT1	12.7	3.8
	ZDT2	16.1	3.4
	ZDT3	12.5	4.6
	ZDT4	16.0	17.8
	ZDT6	97.7	0.3

TABLE VI
AVERAGE GENERATIONAL DISTANCES FROM THE PARETO-OPTIMAL FRONTS TO THE NONDOMINATED FRONTS FOUND BY NSGA-II AND MOEA/D

D-Metric		NSGA-II	MOEA/D
Instance	ZDT1	0.0050	0.0057
	ZDT2	0.0049	0.0071
	ZDT3	0.0084	0.0233
	ZDT4	0.0239	0.0080
	ZDT6	0.0238	0.0067

D. Experimental Setting

In our experimental studies, the implementation of NSGA-II follows [6]. The population size N in both NSGA-II and MOEA/D is set to be 100 for all the test instances. Both algorithms stop after $25,000$ F-function evaluations.

Initial populations are generated by uniformly randomly sampling from the feasible search space in both algorithms. z^i in MOEA/D is initialized as the lowest value of f_i found in the initial population. The simulated binary crossover (SBX) and polynomial mutation are used in both NSGA-II and MOEA/D. More precisely, in Step 2.1 of MOEA/D, the crossover operator generates one offspring, which is then modified by the mutation operator. The control parameters of these two operators are the same in both algorithms. The distribution indexes in both SBX and the polynomial mutation are set to be 20. The crossover rate is 1.00 while the mutation rate is $1/n$ where n is the number of decision variables.

In MOEA/D, the setting of weight vectors $\lambda^1, \ldots, \lambda^N$ is the same as in Section IV-E, T is set to be 20.

E. Experimental Results

Table VII presents the average CPU time of the two algorithms for each test instance. It is clear from Table VII that MOEA/D runs at least twice fast as NSGA-II does with the same number of function evaluations for all the test instances. This observation is consistent with our analysis in Section V-D.

Both MOEA/D and NSGA-II have been run 20 times independently for each test instance. As in the knapsack problem, we use both the C-metric and D-metric to compare the two algorithms. To compute D-metric, P^* is chosen to be a set of 500 uniformly distributed points in the PF.

Figure 6 illustrates the evolution of the average D-value of the current population to P^* with the number of function evaluations in two algorithms for each test instance. These figures indicate that MOEA/D converges a little bit faster than NSGA-II in minimizing the D-metric for ZDT4 and ZDT6, but a little bit slower for ZDT1-3, in terms of the number of the function evaluations.

Table V presents the mean of D-metric of the final solutions obtained by two algorithms for each test instance. This table reveals that, in terms of D-metric, the final solutions obtained by MOEA/D is slightly better than NSGA-II in ZDT4 and ZDT6, and slightly worse in ZDT1-3.

Table VI shows that, in terms of C-metric, the final solutions obtained by MOEA/D is a little bit better than those obtained by NSGA-II for all the test instances but ZDT4.

Figures 7-8 show, in the objective space, the distribution of the final solutions obtained in the run with the lowest D-value of two algorithms for each test instance. It is evident that, as to the uniformness of final solutions, MOEA/D is better than NSGA-II for ZDT1, ZDT2, and ZDT6, is about the same as NSGA-II for ZDT4. It is also clear that MOEA/D found fewer solutions than NSGA-II in the two left segments of the PF in ZDT3. This suggests that the decomposition by Tchebycheff

TABLE VII
AVERAGE CPU TIME (S) USED BY NSGA-II AND MOEA/D

Average CPU Time		NSGA-II	MOEA/D
Instance	ZDT1	1.25	0.45
	ZDT2	1.25	0.45
	ZDT3	1.20	0.45
	ZDT4	1.15	0.25
	ZDT6	1.05	0.20

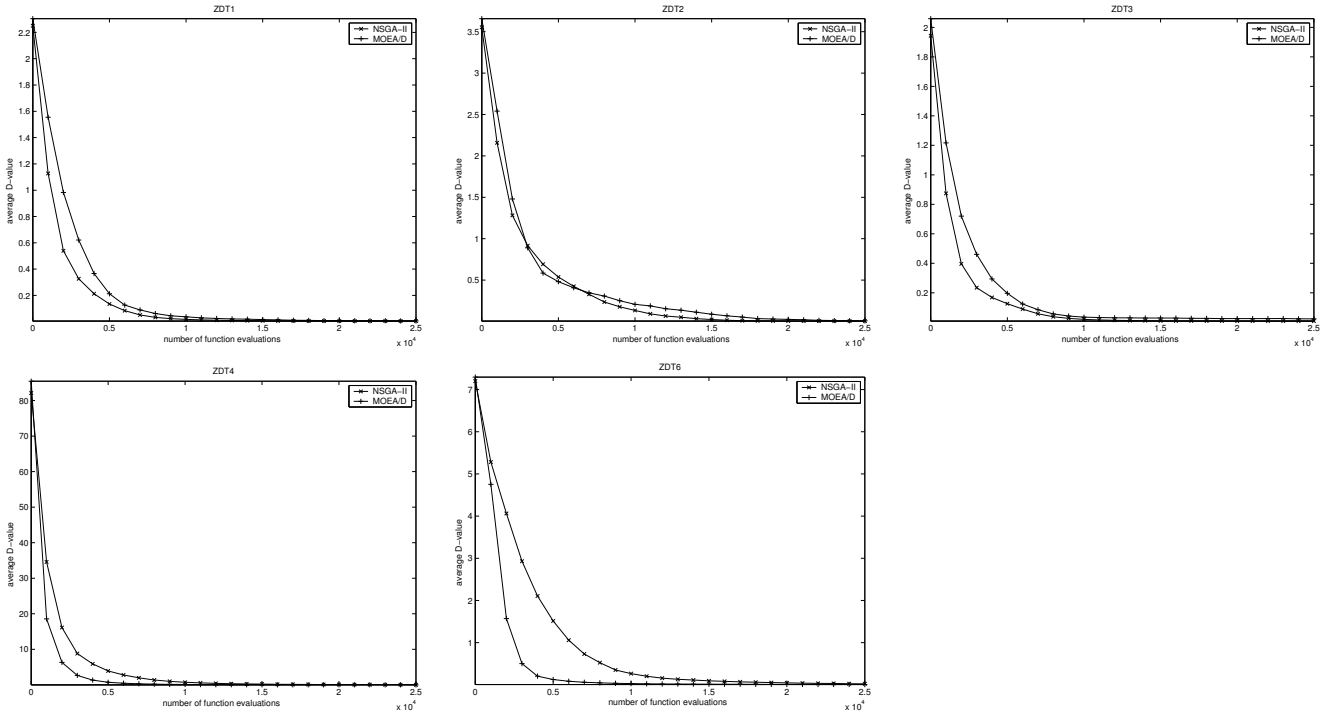

Fig. 6. The evolution of D-value of NSGA-II and MOEA/D for five ZDT test instances

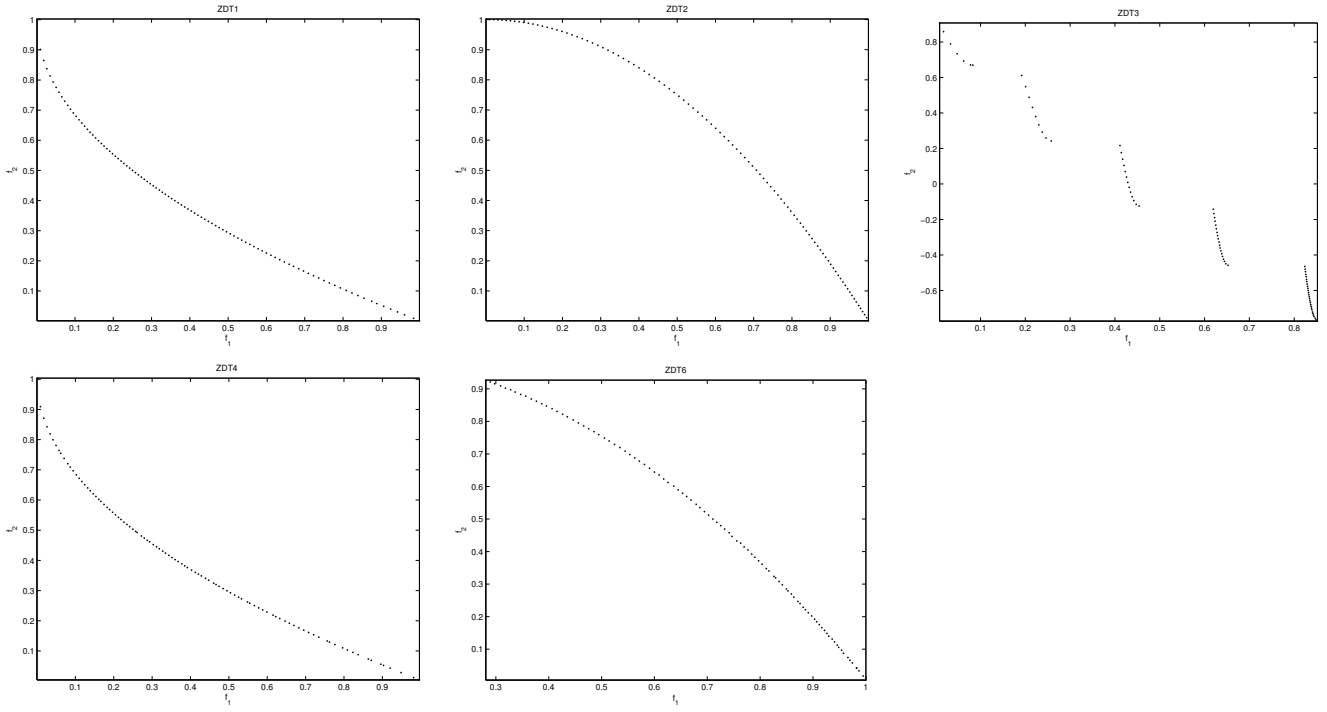

Fig. 7. Plot of the nondominated front with the lowest D-value of MOEA/D for five ZDT test instances

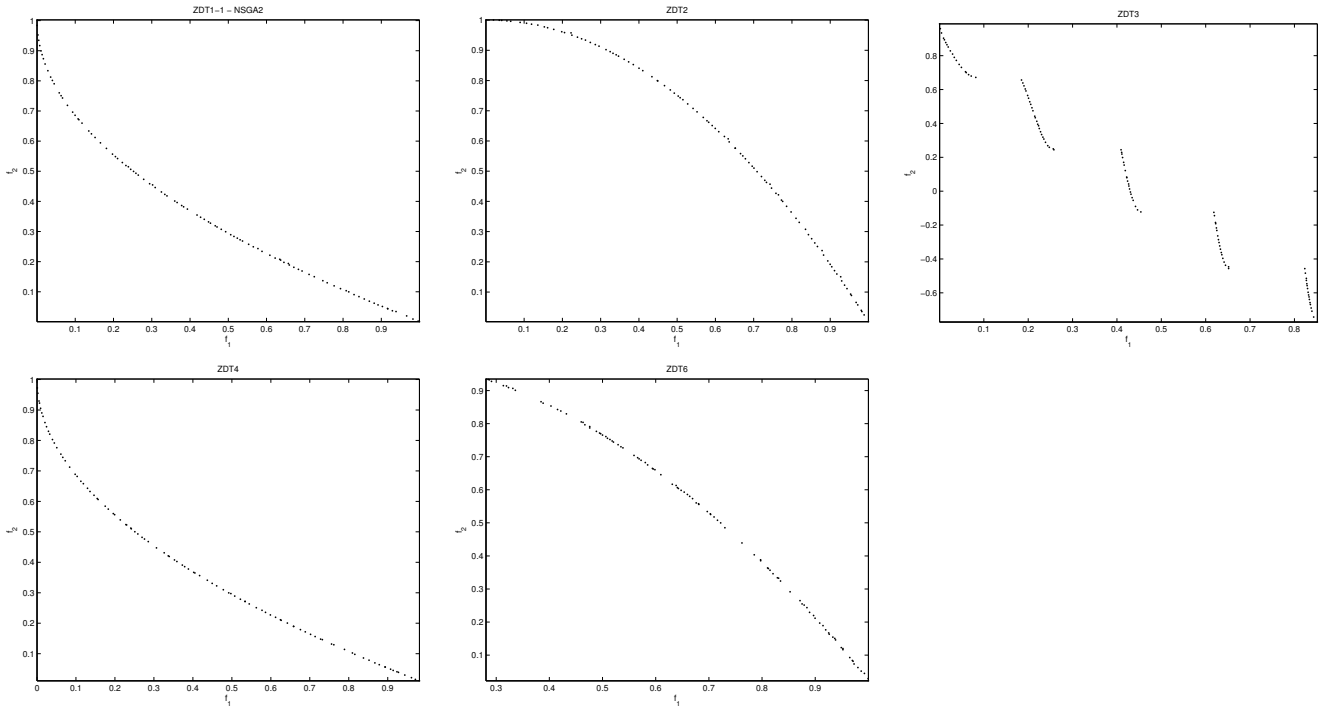

Fig. 8. Plot of the nondominated front with the lowest D-value of NSGA-II for five ZDT test instances

approach with uniformly distributed weight vectors may not lead to uniform distributed Pareto optimal solutions in the PF. This shortcoming should be overcome by adaptively adjusting the weight vectors in MOEA/D to encourage the uniformness in the objective space.

We can conclude from the above results that MOEA/D needs less CPU time than NSGA-II with the same number of function evaluations for these test problems. the solution qualities of the solutions generated by these two algorithms are about the same given the same number of function evaluations.

VI. CONCLUSIONS

Decomposition has widely used in traditional mathematical programming methods for solving MOPs. In contrast, most MOEAs treat a MOP as a whole and mainly rely on domination for measuring the solution quality during their search.

This paper has proposed a simple and generic evolutionary multiobjective optimization algorithm based on decomposition, called MOEA/D. It first uses a decomposition method to decompose the MOP into a number of scalar optimization problems. Then an evolutionary algorithm is employed for optimizing these subproblems simultaneously. Each individual solution in the population of MOEA/D is associated with a subproblem. A neighborhood relationship among all the subproblems is defined based on the distances of their weight vectors. In MOEA/D, optimization of a subproblem uses the current information of its neighboring subproblems since two neighboring subproblems should have close optimal solutions. We compared MOEA/D with MOGLS and NSGA-II on multiobjective knapsack problems and continuous multiobjective problems, respectively. Our analysis shows that MOEA/D has lower computational complexity than MOGLS and NSGA-II, and the experimental results also confirm it. In terms of solution quality, MOEA/D outperforms or performs similarly to MOGLS and NSGA-II for most test instances.

Our experimental studies on the multiobjective knapsack problem suggest that different decomposition strategies may lead to different performances in MOEA/D. Therefore, one of our major research issues in the future to study the effects of other decomposition strategies such as normal boundary intersection method [2] in MOEA/D. Our experimental results also indicate that a decomposition method with uniform weight vectors, in MOEA/D, may not generate a set of uniformly-distributed Pareto solutions in the objective space. A strategy for adaptively adjusting weight vectors is worthwhile studying for overcoming this disadvantage.

Combination of mathematical programming methods and evolutionary algorithms has been proved to be very successful in scalar optimization problems. Our work in this work provides a very natural way for introducing decomposition strategies, which have been studied in the community of mathematical programming for long time, into evolutionary algorithms for multiobjective optimization. New developments in decomposition strategies can be readily applied in MOEA/D.

REFERENCES

[1] K. Miettinen, *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[2] Das, "Normal-bounday intersection: A new method for generating pareto optimal points in multicriteria optimization problems," *SIAM J. on Optimization*, vol. 8, no. 3, pp. 631–657, August 1998.

[3] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach." *IEEE Trans. Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[4] Y.-W. Leung and Y. Wang, "Multiobjective programming using uniform design and genetic algorithm." *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 30, no. 3, pp. 293–, 2000.

[5] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, Eds., Athens, Greece, pp. 95–100.

[6] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II." *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[7] E. Polak, "On the approximation of solutions to multiple criteria decision making problems," in *Multiple Criteria Decision Making*, ser. Lecture Notes in Economics and Mathematical Systems, M. Zeleny, Ed., vol. 123. Springer, 1976, pp. 271–282.

[8] S. Helbig, "On a constructive approximation of the efficient outcomes in bicriterion vector optimization," *Journal of Global Optimization*, vol. 5, pp. 35–48, 1994.

[9] M. Wiecek, W. Chen, and J. Zhang, "Piecewise quadratic approximation of the non-dominated set for bi-criteria programs," *Journal of Multi-Criteria Decision Analysis*, vol. 10, no. 1, pp. 35–47, 2001.

[10] T. Okabe, Y. Jin, B. Sendhoff, and M. Olhofer, "Voronoi-based estimation of distribution algorithm for multi-objective optimization," in *Congress on Evolutionary Computation (CEC)*, Y. Shi, Ed. Portland: IEEE, 2004, pp. 1594–1602.

[11] H. Ishibuchi and T. Murata, "Multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, no. 3, pp. 392–403, 1998.

[12] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment." *IEEE Trans. Evolutionary Computation*, vol. 6, no. 4, pp. 402–412, 2002.

[13] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling." *IEEE Trans. Evolutionary Computation*, vol. 7, no. 2, pp. 204–223, 2003.

[14] C. A. Mattson, A. A. Mullur, and A. Messac, "Smart pareto filter: Obtaining a minimal representation of multiobjective design space," vol. 36, no. 6, pp. 721–740, 2004.

[15] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results." *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.

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

[17] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Inc., 1985, pp. 93–100.

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

[19] J. D. Knowles and D. W. Corne, "The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimisation," in *Congress on Evolutionary Computation (CEC'99)*. Washington D.C.: IEEE Service Center, July 1999, pp. 98–105.

[20] E. Ulungu, J. Teghem, P. Fortemps, and D. Tuyttens, "Mosa method: A tool for solving multiobjective combinatorial optimization problem." *Journal of Multi-Criteria Decision Analysis*, vol. 8, no. 4, pp. 221–236, 1999.

[21] L. Paquete and T. Stützle, "A two-phase local search for the biobjective traveling salesman problem." in *EMO*, 2003, pp. 479–493.

[22] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[23] A. Jaszkiewicz, "Genetic local search for mulitple objective combinatorial optimization," Institute of Computing Science, Poznañ University of Technology, Tech. Rep. RA-014/98, 1998.

[24] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., 2001.

[25] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review." *IEEE Trans. Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.