Multiobjective Programming Using Uniform Design and Genetic Algorithm

Yiu-Wing Leung, *Senior Member, IEEE*, and Yuping Wang

***Abstract*—The notion of Pareto-optimality is one of the major approaches to multiobjective programming. While it is desirable to find more Pareto-optimal solutions, it is also desirable to find the ones scattered uniformly over the Pareto frontier in order to provide a variety of compromise solutions to the decision maker. In this paper, we design a genetic algorithm for this purpose. We compose multiple fitness functions to guide the search, where each fitness function is equal to a weighted sum of the normalized objective functions and we apply an experimental design method called *uniform design* to select the weights. As a result, the search directions guided by these fitness functions are scattered uniformly toward the Pareto frontier in the objective space. With multiple fitness functions, we design a selection scheme to maintain a good and diverse population. In addition, we apply the uniform design to generate a good initial population and design a new crossover operator for searching the Pareto-optimal solutions. The numerical results demonstrate that the proposed algorithm can find the Pareto-optimal solutions scattered uniformly over the Pareto frontier.**

***Index Terms*—Experimental design methods, genetic algorithms, multiobjective programming, Pareto-optimality, uniform array, uniform design.**

I. Introduction

We consider the following multiobjective programming problem:

$$\underset{x\in\Omega}{\text{Minimize}}\ f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x}) \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is a variable vector in a real and N-dimensional space, Ω is the feasible solution space, and there are M objective functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})$. Many real-world decision problems can be formulated as the above problem (e.g., see [1]–[3]). Very often, the objective functions are noncommensurable and they cannot be optimized simultaneously, and the decision maker has to find a compromise solution.

The notion of Pareto-optimality is one of the major approaches to multiobjective programming [1]–[7]. For any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in Ω, if the following conditions hold:

$$\begin{cases} f_i(\mathbf{x}_1) \le f_i(\mathbf{x}_2), & \text{for all } i \in \{1, 2, \ldots, M\} \\ f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2), & \text{for some } j \in \{1, 2, \ldots, M\} \end{cases} \tag{2}$$

then $\mathbf{x}_1$ is at least as good as $\mathbf{x}_2$ with respect to all the M objectives (the first condition), and $\mathbf{x}_1$ is strictly better than $\mathbf{x}_1$ with respect to at least one objective (the second condition). Therefore, $\mathbf{x}_1$ is strictly better than $\mathbf{x}_2$. If no other solution is strictly better than $\mathbf{x}_1$, then $\mathbf{x}_1$ is called a *Pareto-optimal solution*. A multiobjective programming problem may have multiple Pareto-optimal solutions, and these solutions can be regarded as the best compromise solutions. Different decision makers with different preference may select different Pareto-optimal solutions. It may be desirable to find all the Pareto-optimal solutions, so that the decision maker can select the best one based on his preference. The set of all possible Pareto-optimal solutions constitutes a *Pareto frontier* in the objective space. Fig. 1 shows an example.

Many multiobjective programming problems have very large or infinite numbers of Pareto-optimal solutions. When it is not possible to find all these solutions, it may be desirable to find as many solutions as possible in order to provide more choices to the decision maker.

Genetic algorithm (GA) is a promising approach to finding Pareto-optimal solutions [1]–[6], [8]–[10]. It evolves and improves a population of potential solutions iteratively using biologically inspired operators such as selection, crossover and mutation [11]–[14]. In this evolution, it uses a fitness function to guide the population members to converge toward the Pareto frontier. A well-known fitness function is the weighted sum of objective function

$$fitness = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \cdots + w_M f_M(\mathbf{x}) \tag{3}$$

where $w_1, w_2, \ldots, w_M$ are nonnegative weights such that $w_1 + w_2 + \cdots + w_M = 1$. We call $\mathbf{w} = (w_1, w_2, \ldots, w_M)$ a weight vector.

If a GA uses one weight vector to compose one fitness function, there is only one search direction. For example, if $\mathbf{w}' = (0.5, 0.5)$ is used for a two-objective programming problem, the search direction in the objective space is shown in Fig. 2. Along this search direction, it may be easy to find the Pareto-optimal solutions B and C, but it is difficult to find the other Pareto-optimal solutions such as A and D. To overcome this shortcoming, multiple weight vectors can be used to compose multiple fitness functions, so that there are multiple search directions [1], [9], [10]. Three specific methods were proposed in the literature.

1) Schaffer [9] proposed to divide the population into M sub-populations, and adopt M fitness functions where the first fitness function is $f_1(\mathbf{x})$, the second fitness function is $f_2(\mathbf{x})$, etc. Each sub-population is guided by one fitness function, and hence there are M fixed search directions. Fig. 2 shows the search directions in the objective

Manuscript received June 18, 1999; revised August 7, 2000. This work was supported by the HKBU FRG under Research Grant FRG/98-99/II-62.

Y.-W. Leung is with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong (http://www.comp.hkbu.edu.hk/~ywleung).

Y. Wang is with the Department of Applied Mathematics, Xidian University, Xi'an 710071, China.

Publisher Item Identifier S 1094-6977(00)09082-9.

1094–6977/00$10.00 © 2000 IEEE

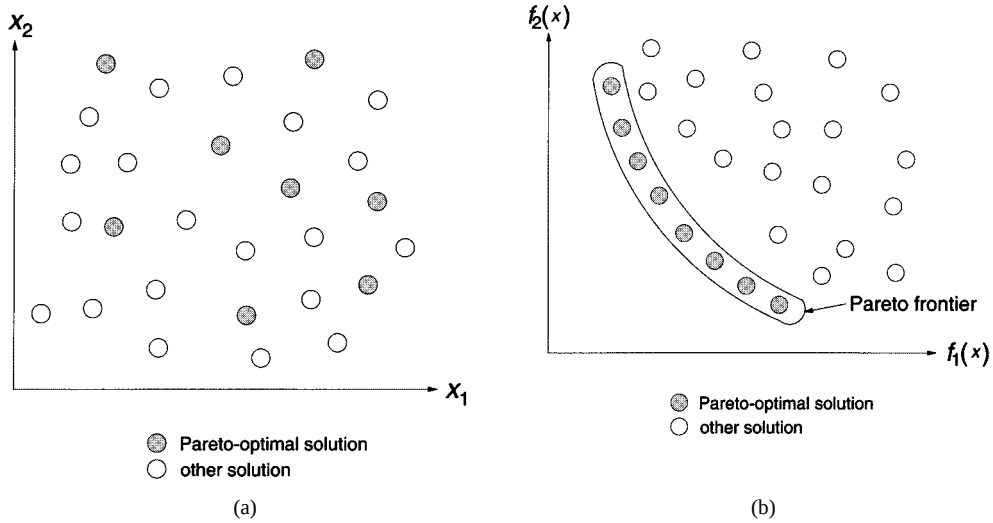

Fig. 1. Pareto-optimal solutions for a two-objective programming problem over a 2-D solution space.

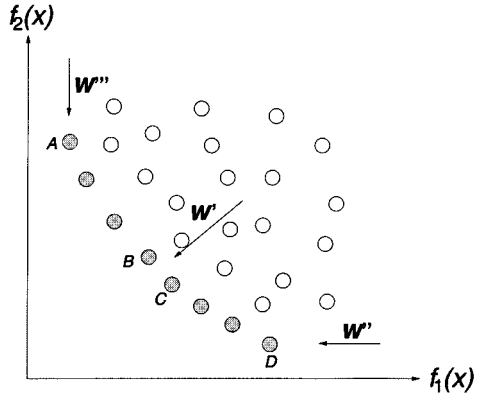

Fig. 2. Search directions toward the Pareto-optimal solutions.

space (denoted by $\mathbf{w}''$ and $\mathbf{w}'''$) for a two-objective programming problem. Along these two directions, it may be easy to find the Pareto-optimal solutions A and D, but it is difficult to find the other Pareto-optimal solutions such as B and C.

2) Kursawe [10] proposed a variant of the above method. The decision maker has to specify a probability for each objective function. When a fitness function is needed for selection, one of the objective functions is selected as the fitness function based on this probability distribution. Similar to the Schaffer's method, this method provides M fixed search directions.
3) Ishibuchi and Murata [1] recently proposed an interesting method. When a fitness function is needed for selection, a weighted sum of the objective functions is composed as the fitness function where the weights are randomly generated. Therefore, this method can provide multiple and randomly generated search directions toward the Pareto frontier.

The Ishibuchi–Murata method [1] can find the solutions that are randomly scattered over the Pareto frontier. It is possible that some solutions are close to each other in the objective space (e.g., see Fig. 3). If the solutions are close to each other, they are nearly the same choice. For example, if $f_1(\mathbf{x})$ is the cost and $f_2(\mathbf{x})$ is the reliability, then the solutions $\{f_1 = 10000, f_2 = 0.9990\}, \{f_1 = 10002, f_2 = 0.9991\}$ and $\{f_1 = 9999, f_2 = 0.9989\}$ are nearly the same choice. It is desirable to find the Pareto-optimal solutions scattered uniformly over the Pareto frontier, so that the decision maker can have a variety of choices (e.g., see Fig. 4).

In this paper, we design a genetic algorithm to determine the Pareto-optimal solutions scattered uniformly over the Pareto frontier. We apply an experimental design method called *uniform design* to compose multiple fitness functions and we design a selection scheme using these fitness functions, so that the resulting search directions are scattered uniformly toward the Pareto frontier. In addition, we apply the uniform design to generate a good initial population and design a new crossover operator. We demonstrate the effectiveness of the proposed algorithm by numerical experiments.

II. Uniform Design

Experimental design method is a sophisticated branch of statistics [15], [16]. In this section, we briefly describe an experimental design method called *uniform design*. The main objective of uniform design is to sample a small set of points from a given set of points, such that the sampled points are uniformly scattered. We describe the main features of uniform design in the following, and we refer the readers to [17]–[21] for more details.

Suppose the yield of a chemical depends on the temperature, the amount of catalyst, and the duration of the chemical process. These three quantities are called the *factors* of the experiment. If each factor has ten possible values, we say that each factor has ten *levels*. There are $10^3 = 59\,049$ combinations of levels. To find the best combination for a maximum yield, it is necessary to do 1000 experiments. When it is not possible or cost-effective to do all these experiments, it is desirable to select a small but representative sample of experiments. The uniform design was developed for this purpose [17]–[21].

Let there be n factors and q levels per factor. When n and q are given, the uniform design selects q combinations out of

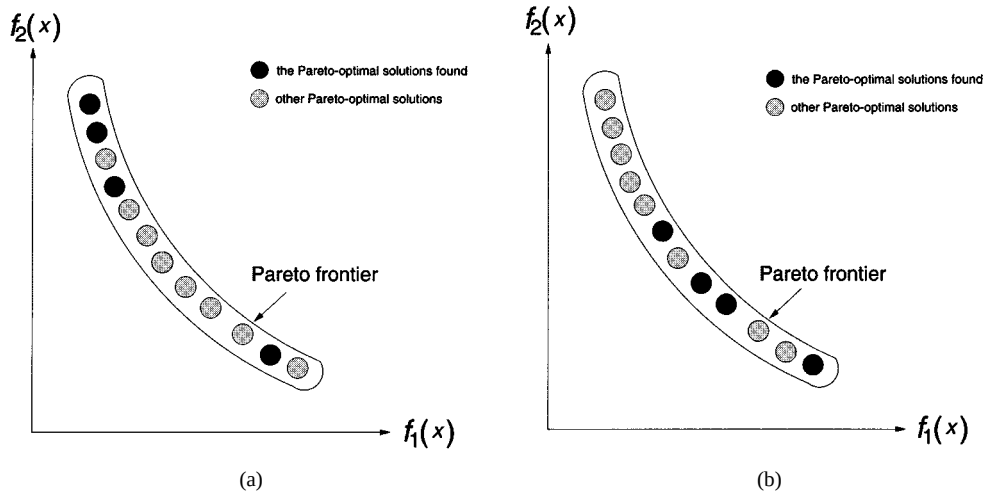

Fig. 3. When the weight vectors are randomly generated, the resulting Pareto-optimal solutions are scattered randomly over the Pareto frontier.

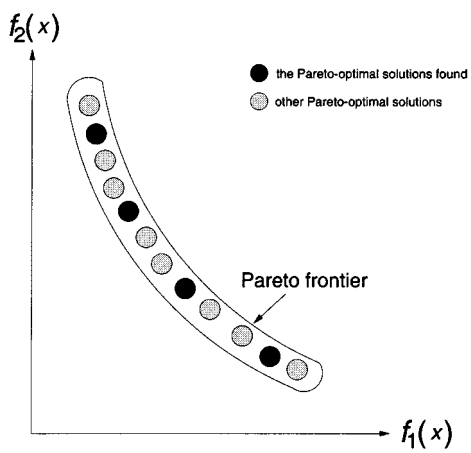

Fig. 4. Ideally, the Pareto-optimal solutions are scattered uniformly over the Pareto frontier.

q^n possible combinations, such that these q combinations are scattered uniformly over the space of all possible combinations. The selected q combinations are expressed in terms of a *uniform array* $\mathbf{U}(n,q) = [U_{i,j}]_{q\times n}$, where $U_{i,j}$ is the level of the jth factor in the ith combination.

Uniform arrays can be constructed as follows. Consider a unit hypercube over an n-dimensional space. We denote this hypercube by the set of points in it

$$\mathbf{C} = \{(c_1, c_2, \ldots, c_n) \,|\, 0 \le c_i \le 1 \text{ for } i = 1, 2, \ldots, n\}. \quad (4)$$

Consider any point in $\mathbf{C}$, say $\mathbf{r} = (r_1, r_2, \ldots, r_n)$. We form a hyper-rectangle between $\mathbf{0}$ and $\mathbf{r}$, and we denote it by the set of points in it

$$\mathbf{C}(\mathbf{r}) = \{(c_1, c_2, \ldots, c_n) \,|\, 0 \le c_i \le r_i \text{ for } i = 1, 2, \ldots, n\}. \quad (5)$$

We select a sample of q points such that they are scattered uniformly in the hypercube. Suppose $q(\mathbf{r})$ of these points are in the hyper-rectangle $\mathbf{C}(\mathbf{r})$. Then the fraction of points in the hyper-rectangle is $q(\mathbf{r})/q$. The volume of the unit hypercube is 1, and hence the fraction of volume of this hyper-rectangle is $r_1 r_2 \ldots r_n$. The uniform design is to determine q points in $\mathbf{C}$ such that the following *discrepancy* is minimized:

$$\sup_{\mathbf{r}\in\mathbf{C}} \left| \frac{q(\mathbf{r})}{q} - r_1 r_2 \ldots r_n \right|. \quad (6)$$

Then we map these q points in the unit hypercube to the space with n factors and q levels. When q is prime and $q > n$, it has been proved that $U_{i,j}$ is given by [17]–[21]

$$U_{i,j} = (i\sigma^{j-1} \bmod q) + 1 \quad (7)$$

where σ is a parameter given in Table I.

Example 1: We construct a uniform array with five factors and seven levels as follows. From Table I, we see that σ is equal to 3. We compute $\mathbf{U}(5,7)$ based on (7) and we get

$$\mathbf{U}(5,7) = \begin{bmatrix} 2 & 4 & 3 & 7 & 5 \\ 3 & 7 & 5 & 6 & 2 \\ 4 & 3 & 7 & 5 & 6 \\ 5 & 6 & 2 & 4 & 3 \\ 6 & 2 & 4 & 3 & 7 \\ 7 & 5 & 6 & 2 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (8)$$

In the first combination, the five factors have respective levels 2, 4, 3, 7, 5; in the second combination, the five factors have respective levels 3, 7, 5, 6, 2, etc.

III. Design of Genetic Algorithm for Multiobjective Programming

We let the feasible range of x_i be $[l_i, u_i]$, and we call this range the *domain* of x_i. We let $\mathbf{l} = (l_1, l_2, \ldots, l_N)$ and $\mathbf{u} = (u_1, u_2, \ldots, u_N)$, and we denote the feasible solution space by $[\mathbf{l}, \mathbf{u}]$. We define $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ to be a chromosome. The problem is to find a set of chromosomes that are scattered uniformly over the Pareto frontier in the objective space.

A. Fitness Functions

Here, we apply the uniform design to compose multiple fitness functions, such that their search directions are scattered uniformly toward the Pareto frontier in the objective space.

TABLE I
VALUES OF THE PARAMETER σ FOR DIFFERENT NUMBER OF FACTORS AND DIFFERENT NUMBER OF LEVELS PER FACTOR

Number of levels per factors	Number of factors	σ
5	2 – 4	2
7	2 – 6	3
11	2 – 10	7
13	2	5
	3	4
	4 – 12	6
17	2 – 16	10
19	2 – 3	8
	4 – 18	14
23	2, 13 – 14, 20 – 22	7
	8 – 12	15
	3 – 7, 15 – 19	17
29	2	12
	3	9
	4 – 7	16
	8 – 12, 16 – 24	8
	13 – 15	14
	25 – 28	18
31	2, 5 – 12, 20 – 30	12
	3 – 4, 13 – 19	22

The values of different objective functions may have different order of magnitude. If a fitness function is equal to a weighted sum of objective functions, it may be dominated by the objective functions with large values. For example, if $f_1(\mathbf{x})$ represents the cost with $10\,000 \le f_1(\mathbf{x}) \le 100\,000$ and $f_2(\mathbf{x})$ represents the reliability with $0 \le f_2(\mathbf{x}) \le 1$, then $w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x})$ may be dominated by $f_1(\mathbf{x})$. To overcome this problem, we normalize each objective function as follows:

$$h_i\mathbf{x} = \frac{f_i(\mathbf{x})}{\max_{y\in\Psi}\{|f_i(\mathbf{y})|\}} \tag{9}$$

where Ψ is a set of points in the current population and $h_i(\mathbf{x})$ is the normalized objective function.

We compose D fitness functions for any given D, where the ith fitness function is given by

$$fitness_i = w_{i,1}h_1(\mathbf{x}) + w_{i,2}h_2(\mathbf{x}) + \cdots + w_{i,M}h_M(\mathbf{x}), \quad 1 \le i \le D. \tag{10}$$

Let $\mathbf{w}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,M})$. We apply the uniform design to select the weight vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_D$ as follows. In the objective space, we treat each objective function as one factor and hence there are M factors. We need D weight vectors and hence there are D levels. We apply the uniform array $\mathbf{U}(M, D)$ to determine $w_{i,j}$ for any $1 \le i \le D$ and $1 \le j \le M$ as follows:

$$w_{i,j} = \frac{U_{i,j}}{U_{i,1} + U_{i,2} + \cdots + U_{i,M}} \tag{11}$$

where the denominator ensures that the weights for each fitness function must sum to one.

Example 2: We let $M = 5$ and $D = 7$. We apply the uniform array $\mathbf{U}(5, 7)$ [given by (8)] to select seven weight vectors, and then compose the seven fitness functions shown in the equation at the bottom of the page.

B. Generation of Initial Population

Before we solve a multiobjective optimization problem, we have no information about the location of the Pareto-optimal solutions in the solution space [see Fig. 1(a)]. We generate an initial population in which the population members are scattered uniformly over the feasible solution space, so that the genetic algorithm can explore the whole solution space evenly. For this purpose, we quantize the feasible solution space into a large number of points, and then apply the uniform design and a selection scheme to select G points as the initial population where G is a design parameter.

When the solution space is large, it is desirable to sample more points for a better coverage. In principle, we can apply the uniform array with a larger number of levels. However, only the uniform arrays with at most 37 levels have been tabulated in the literature [20], and it is very time consuming to compute the larger uniform arrays. To bypass this difficulty, we divide the solution space into multiple subspaces, and then apply the uniform array to sample some points in each subspace.

We divide $[\mathbf{l}, \mathbf{u}]$ into S subspaces $[\mathbf{l}(1), \mathbf{u}(1)], [\mathbf{l}(2), \mathbf{u}(2)], \ldots, [\mathbf{l}(S), \mathbf{u}(S)]$, where the design parameter S can assume the values 2, or 2^2, or 2^3, etc. First, we divide the solution space into two subspaces as follows. We select the dimension with the largest domain,

$$\begin{cases}
fitness_1 = \frac{2}{21}h_1(\mathbf{x}) + \frac{4}{21}h_2(\mathbf{x}) + \frac{3}{21}h_3(\mathbf{x}) + \frac{7}{21}h_4(\mathbf{x}) + \frac{5}{21}h_5(\mathbf{x}) \\
fitness_2 = \frac{3}{23}h_1(\mathbf{x}) + \frac{7}{23}h_2(\mathbf{x}) + \frac{5}{23}h_3(\mathbf{x}) + \frac{6}{23}h_4(\mathbf{x}) + \frac{2}{23}h_5(\mathbf{x}) \\
fitness_3 = \frac{4}{25}h_1(\mathbf{x}) + \frac{3}{25}h_2(\mathbf{x}) + \frac{7}{25}h_3(\mathbf{x}) + \frac{5}{25}h_4(\mathbf{x}) + \frac{6}{25}h_5(\mathbf{x}) \\
fitness_4 = \frac{5}{20}h_1(\mathbf{x}) + \frac{6}{20}h_2(\mathbf{x}) + \frac{2}{20}h_3(\mathbf{x}) + \frac{4}{20}h_4(\mathbf{x}) + \frac{3}{20}h_5(\mathbf{x}) \\
fitness_5 = \frac{6}{22}h_1(\mathbf{x}) + \frac{2}{22}h_2(\mathbf{x}) + \frac{4}{22}h_3(\mathbf{x}) + \frac{3}{22}h_4(\mathbf{x}) + \frac{7}{22}h_5(\mathbf{x}) \\
fitness_6 = \frac{7}{24}h_1(\mathbf{x}) + \frac{5}{24}h_2(\mathbf{x}) + \frac{6}{24}h_3(\mathbf{x}) + \frac{2}{24}h_4(\mathbf{x}) + \frac{4}{24}h_5(\mathbf{x}) \\
fitness_7 = \frac{1}{5}h_1(\mathbf{x}) + \frac{1}{5}h_2(\mathbf{x}) + \frac{1}{5}h_3(\mathbf{x}) + \frac{1}{5}h_4(\mathbf{x}) + \frac{1}{5}h_5(\mathbf{x})
\end{cases}$$

and divide the solution space into two equal subspaces along this dimension. Then we divide the two subspaces into four subspaces as follows. For any subspace, say $[\mathbf{l}(1), \mathbf{u}(1)]$, we select the dimension with the largest domain, and then divide the two subspaces along this dimension into four equal subspaces. We repeat this step in a similar manner, until the solution space has been divided into S subspaces. The details are as follows:

Algorithm 1: Dividing the Solution Space

Step 1) Let $\mathbf{a} = \mathbf{l}$ and $\mathbf{z} = \mathbf{u}$. Repeat the following computation $\log_2 S$ times: select the sth dimension such that $z_s - a_s = \max_{1 \le i \le N}\{z_i - a_i\}$, and then compute $z_s = (a_s + z_s)/2$.

Step 2) Compute $\Delta_i = z_i - a_i$ and $n_i = (u_i - l_i)/\Delta_i$ for all $i = 1, 2, \ldots, N$. Then compute the subspace $[\mathbf{l}(k), \mathbf{u}(k)]$ for all $1 \le j_i \le n_i$ and $1 \le i \le N$ as follows:

$$\begin{cases} \mathbf{l}(k) = \mathbf{l} + ((j_1 - 1)\Delta_1, (j_2 - 1)\Delta_2, \ldots, (j_N - 1)\Delta_N) \\ \mathbf{u}(k) = \mathbf{l} + (j_1\Delta_1, j_2\Delta_2, \ldots, j_N\Delta_N) \end{cases}$$

where $k = (j_1 - 1)n_2 n_3 \ldots n_N + (j_2 - 1)n_3 n_4 \ldots n_N + \cdots + (j_{N-1} - 1)n_N + j_N$.

After dividing the solution space into S subspaces, we select a sample of points from each subspace as follows. Consider any subspace, say the kth subspace, and denote it by

$$\begin{aligned} &[\mathbf{l}(k), \mathbf{u}(k)] \\ &\quad = [(l_1(k), l_2(k), \ldots, l_N(k)), (u_1(k), u_2(k), \ldots, u_N(k))]. \end{aligned}$$

In this subspace, we quantize the domain $[l_i(k), u_i(k)]$ of x_i into Q_0 levels $\alpha_{i,1}(k), \alpha_{i,2}(k), \ldots, \alpha_{i,Q_0}(k)$ where the design parameter Q_0 is prime and $\alpha_{i,j}(k)$ is given by

$$\alpha_{i,j}(k) = \begin{cases} l_i(k) & j = 1 \\ l_i(k) + (j-1)\left(\frac{u_i(k) - l_i(k)}{Q_0 - 1}\right) & 2 \le j \le Q_0 - 1 \\ u_i(k) & j = Q_0 \end{cases}. \tag{12}$$

In other words, the difference between any two successive levels is the same. We let $\alpha_i(k) = (\alpha_{i,1}(k), \alpha_{i,2}(k), \ldots, \alpha_{i,Q_0}(k))$. After quantization, the subspace consists of Q_0^N points. We apply the uniform array $\mathbf{U}(N, Q_0)$ to sample the following Q_0 points:

$$\begin{cases} \left(\alpha_{1,U_{1,1}}(k), \alpha_{2,U_{1,2}}(k), \ldots, \alpha_{N,U_{1,N}}(k)\right) \\ \left(\alpha_{1,U_{2,1}}(k), \alpha_{2,U_{2,2}}(k), \ldots, \alpha_{N,U_{2,N}}(k)\right) \\ \cdots \\ \left(\alpha_{1,U_{Q_0,1}}(k), \alpha_{2,U_{Q_0,2}}(k), \ldots, \alpha_{N,U_{Q_0,N}}(k)\right) \end{cases}. \tag{13}$$

We repeat the above steps for each of the S subspaces, so that we get a total of SQ_0 points.

Among the SQ_0 points, we select G of them to form the initial population. In this selection, we adopt D_0 fitness functions in order to realize D_0 search directions, where D_0 is a design parameter and it is prime. Based on each fitness function, we evaluate the quality of each of the SQ_0 points and then select the best $\lfloor G/D_0 \rfloor$ or $\lceil G/D_0 \rceil$ points. Overall, we select a total of G points to form the initial population. The details for generating an initial population are as follows:

Algorithm 2: Generation of Initial Population

Step 1) Execute Algorithm 1 to divide the feasible solution space $[\mathbf{l}, \mathbf{u}]$ into S subspaces $[\mathbf{l}(1), \mathbf{u}(1)], [\mathbf{l}(2), \mathbf{u}(2)], \ldots, [\mathbf{l}(S), \mathbf{u}(S)]$.

Step 2) Quantize each subspace based on (12), and then apply the uniform array $\mathbf{U}(N, Q_0)$ to sample Q_0 points based on (13).

Step 3) Based on each fitness function, evaluate the quality of each of the SQ_0 points generated in step 2, and then select the best $\lfloor G/D_0 \rfloor$ or $\lceil G/D_0 \rceil$ points. Overall, a total of G points are selected to form the initial population.

Example 3: Consider a three dimensional solution space. Suppose $0 \le x_1 \le 100, 0 \le x_2 \le 60$, and $0 \le x_3 \le 80$, and hence the feasible solution space $[\mathbf{l}, \mathbf{u}]$ is $[(0,0,0), (100, 60, 80)]$. We choose $S = 4, Q_0 = 5, D_0 = 5$, and $G = 8$. We execute *Algorithm* 1 to divide the solution space into four subspaces as follows.

Step 1) $\mathbf{a}$ and $\mathbf{z}$ are found to be $(0,0,0)$ and $(50, 60, 40)$, respectively.

Step 2) $\Delta_1 = 50, \Delta_2 = 60, \Delta_3 = 40$; and $n_1 = 2, n_2 = 1, n_3 = 2$. The four subspaces are found to be

$$\begin{cases} [\mathbf{l}(1), \mathbf{u}(1)] = [(0,0,0), (50, 60, 40)] \\ [\mathbf{l}(2), \mathbf{u}(2)] = [(0,0,40), (50, 60, 80)] \\ [\mathbf{l}(3), \mathbf{u}(3)] = [(50,0,0), (100, 60, 40)] \\ [\mathbf{l}(4), \mathbf{u}(4)] = [(50,0,40), (100, 60, 80)] \end{cases} \tag{14}$$

We execute *Algorithm 2* to generate an initial population as follows.

Step 1) Divide the solution space into four subspaces, which are given by (14).

Step 2) Quantize the first subspace $[\mathbf{l}(1), \mathbf{u}(1)] = [(0,0,0), (50, 60, 40)]$ based on (12) to get

$$\begin{cases} \alpha_1(1) = (0.0, 12.5, 25.0, 37.5, 50.0) \\ \alpha_2(1) = (0.0, 15.0, 30.0, 45.0, 60.0) \\ \alpha_3(1) = (0.0, 10.0, 20.0, 30.0, 40.0) \end{cases}$$

Adopt the uniform array $\mathbf{U}(3,5)$

$$\mathbf{U}(3,5) = \begin{bmatrix} 2 & 3 & 5 \\ 3 & 5 & 4 \\ 4 & 2 & 3 \\ 5 & 4 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

and select the following five points based on (13)

$$\begin{cases} (12.5, 30.0, 40.0) \\ (25.0, 60.0, 30.0) \\ (37.5, 15.0, 20.0) \\ (50.0, 45.0, 10.0) \\ (0.0, 0.0, 0.0). \end{cases}$$

Proceed in a similar manner for the other three subspaces.

Step 3) Based on each of the first three fitness functions, evaluate the quality of each of the 20 points and then

select the best $\lceil 8/5 \rceil = 2$ points. Based on each of the fourth and fifth fitness functions, evaluate the quality of each of the 20 points and then select the best $\lfloor 8/5 \rfloor = 1$ point. Overall, a total of 8 points are selected to form the initial population.

C. Crossover

We apply the uniform design to design a crossover operator. This operator acts on two parents. It quantizes the solution space defined by these parents into a finite number of points, and then applies the uniform design to select a small sample of uniformly scattered points as the potential offspring.

Consider any two parents $\mathbf{p}_1 = (p_{1,1}, p_{1,2}, \ldots, p_{1,N})$ and $\mathbf{p}_2 = (p_{2,1}, p_{2,2}, \ldots, p_{2,N})$. They define the solution space $[\mathbf{l}_{\text{parent}}, \mathbf{u}_{\text{parent}}]$ as shown in (15) at the bottom of the page. We quantize each domain of $[\mathbf{l}_{\text{parent}}, \mathbf{u}_{\text{parent}}]$ into Q_1 levels $\beta_{i,1}, \beta_{i,2}, \ldots, \beta_{i,Q_1}$, where Q_1 is a design parameter and $\beta_{i,j}$ is given by as shown in (16) at the bottom of the page. In other words, the difference between any two successive levels is the same. We denote $\beta_i = (\beta_{i,1}, \beta_{i,2}, \ldots, \beta_{i,Q_1})$. As the population is being evolved and improved, the population members are getting closer to each other, so that the solution space defined by two parents is becoming smaller. Since Q_1 is fixed, the quantized points are getting closer and hence we can get more and more precise results.

After quantizing $[\mathbf{l}_{\text{parent}}, \mathbf{u}_{\text{parent}}]$, we apply the uniform design to select a sample of points as the potential offspring. These potential offspring will undergo a selection process, and the details are described in Section III-D. Each pair of parents should not produce too many potential offspring in order to avoid a large number of function evaluations during selection. For this purpose, we divide the variables $x_1, x_2, \ldots, x_N$ into F groups where F is a small design parameter, and each group is treated as one factor. Consequently, the corresponding uniform array has a small number of combinations and hence a small number of potential offspring are generated. Specifically, we randomly generate $F-1$ integers $k_1, k_2, \ldots, k_{F-1}$ such that $1 < k_1 < k_2 < \cdots < k_{F-1} < N$, and then create the following F factors for any chromosome $\mathbf{x} = (x_1, x_2, \ldots, x_N)$:

$$\begin{cases} \mathbf{f}_1 = (x_1, \ldots, x_{k_1}) \\ \mathbf{f}_2 = (x_{k_1+1}, \ldots, x_{k_2}) \\ \cdots \\ \mathbf{f}_F = (x_{k_{F-1}+1}, \ldots, x_N) \end{cases} \tag{17}$$

TABLE II
SUMMARY OF DESIGN PARAMETERS AND THEIR FEASIBLE VALUES

	Design Parameter	Feasible Values
Population size	G	G is a positive integer.
Generation of Initial Population	S	$S = 2^i$ where i is a positive integer.
	D_0	D_0 is a prime number larger than M.
	Q_0	Q_0 is a prime number larger than N.
Population Evolution	F	F is a positive integer smaller than or equal to N.
	D_1	D_1 is a prime number larger than or equal to M; and $D_1 < D_0$ in practice.
	Q_1	Q_1 is a prime number larger than F.
	p_m	$0 \le p_m \le 1$ and p_m is small in practice.

Since $x_1, x_2, \ldots, x_N$ have been quantized, we define the following Q_1 levels for the ith factor $\mathbf{f}_i$:

$$\begin{cases} \mathbf{f}_i(1) = \left(\beta_{k_{i-1}+1,1}, \beta_{k_{i-1}+2,1}, \ldots, \beta_{k_i,1}\right) \\ \mathbf{f}_i(2) = \left(\beta_{k_{i-1}+1,2}, \beta_{k_{i-1}+2,2}, \ldots, \beta_{k_i,2}\right) \\ \cdots \\ \mathbf{f}_i(Q_1) = \left(\beta_{k_{i-1}+1,Q_1}, \beta_{k_{i-1}+2,Q_1}, \ldots, \beta_{k_i,Q_1}\right) \end{cases} . \tag{18}$$

We apply the uniform array $\mathbf{U}(F, Q_1)$ to select the following sample of Q_1 chromosomes as the potential offspring:

$$\begin{cases} (\mathbf{f}_1(U_{1,1}), \mathbf{f}_2(U_{1,2}), \ldots, \mathbf{f}_F(U_{1,F})) \\ (\mathbf{f}_1(U_{2,1}), \mathbf{f}_2(U_{2,2}), \ldots, \mathbf{f}_F(U_{2,F})) \\ \cdots \\ (\mathbf{f}_1(U_{Q_1,1}), \mathbf{f}_2(U_{Q_1,2}), \ldots, \mathbf{f}_F(U_{Q_1,F})) \end{cases} \tag{19}$$

The details of the proposed crossover operator are given as follows:

Algorithm 3: Crossover Operation

Step 1) Quantize $[\mathbf{l}_{\text{parent}}, \mathbf{u}_{\text{parent}}]$ based on (15).

Step 2) Randomly generate $F-1$ integers $k_1, k_2, \ldots, k_{F-1}$ such that $1 < k_1 < k_2 < \cdots < k_{F-1} < N$. Create F factors based on (17).

Step 3) Apply the uniform array $\mathbf{U}(F, Q_1)$ to generate Q_1 potential offspring based on (19).

Example 4: Consider a five-dimensional multiobjective programming problem. Let the two parents be $\mathbf{p}_1 = (7.5, 4.7, 2.3, 2.5, 1.0)$ and $\mathbf{p}_2 = (2.7, 0.2, -1.2, 7.5, 6.7)$. These parents define the solution space $[\mathbf{l}_{\text{parent}}, \mathbf{u}_{\text{parent}}] = [(2.7, 0.2, -1.2, 2.5, 1.0), (7.5, 4.7, 2.3, 7.5, 6.7)]$. Based on (7)

$$\begin{cases} \mathbf{l}_{\text{parent}} = [\min(p_{1,1}, p_{2,1}), \min(p_{1,2}, p_{2,2}), \ldots, \min(p_{1,N}, p_{2,N})] \\ \mathbf{u}_{\text{parent}} = [\max(p_{1,1}, p_{2,1}), \max(p_{1,2}, p_{2,2}), \ldots, \max(p_{1,N}, p_{2,N})] \end{cases} \tag{15}$$

$$\beta_{i,j} = \begin{cases} \min(p_{1,i}, p_{2,i}) & j = 1 \\ \min(p_{1,i}, p_{2,i}) + (j-1)\left(\frac{|p_{1,i} - p_{2,i}|}{Q_1 - 1}\right) & 2 \le j \le Q_1 - 1 \\ \max(p_{1,i}, p_{2,i}) & j = Q_1 \end{cases} \tag{16}$$

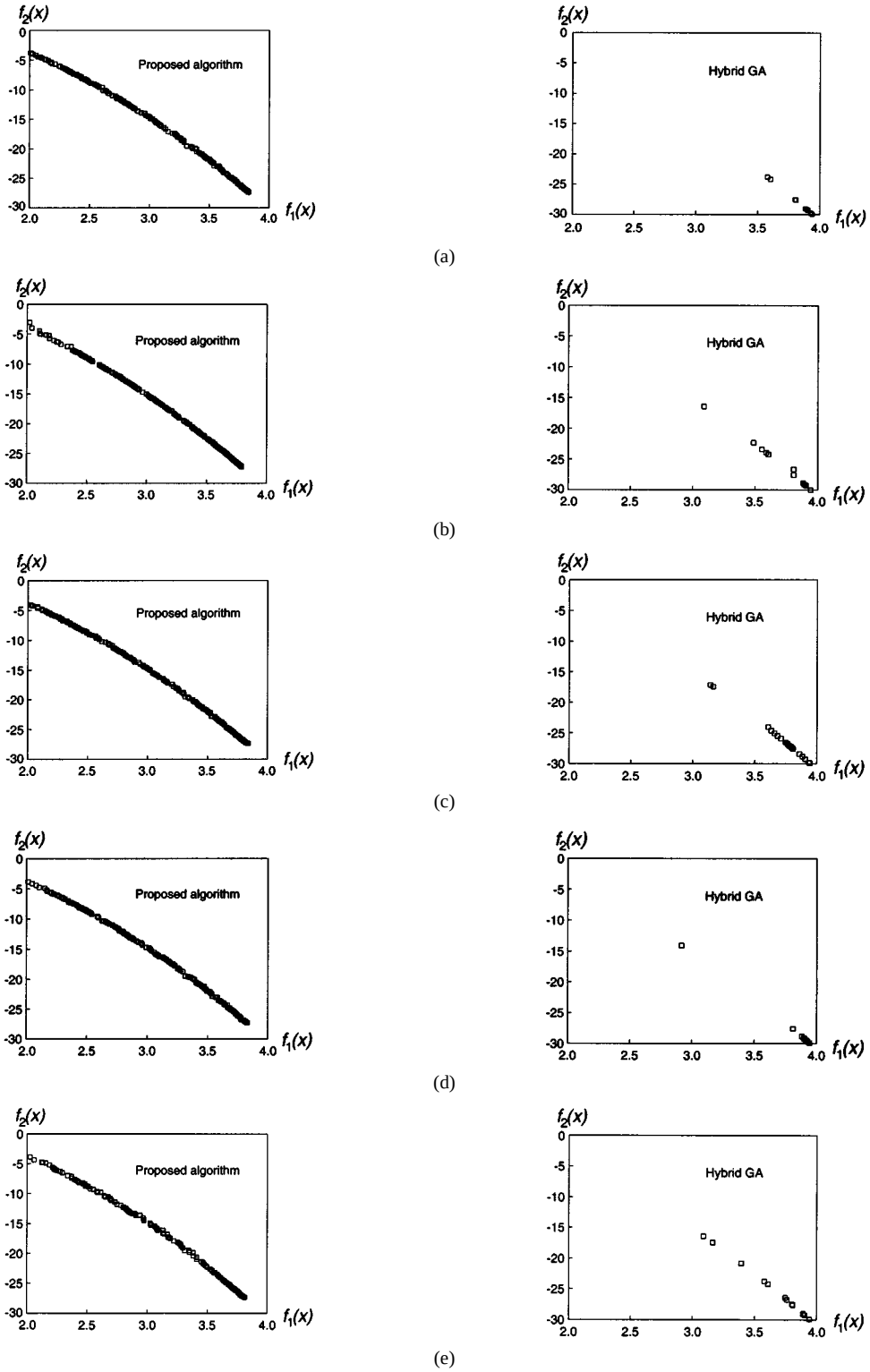

Fig. 5. Pareto-optimal solutions in the objective space for the first test problem.

and Table I, $\mathbf{U}(4,5)$ can be found to be

$$\mathbf{U}(4,5) = \begin{bmatrix} 2 & 3 & 5 & 4 \\ 3 & 5 & 4 & 2 \\ 4 & 2 & 3 & 5 \\ 5 & 4 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{20}$$

The execution of *Algorithm 3* is as follows.

Step 1) Quantize $[\mathbf{l}_{\text{parent}}, \mathbf{u}_{\text{parent}}]$ into

$$\begin{cases} \beta_1 = (2.700, 3.900, 5.100, 6.300, 7.500) \\ \beta_2 = (0.200, 1.325, 2.450, 3.575, 4.700) \\ \beta_3 = (-1.200, -0.325, 0.550, 1.425, 2.300) \\ \beta_4 = (2.500, 3.750, 5.000, 6.250, 7.500) \\ \beta_5 = (1.000, 2.425, 3.850, 5.275, 6.700). \end{cases}$$

Step 2) Suppose $k_1 = 1, k_2 = 3, k_3 = 4$, and $k_4 = 5$. Create the following four factors: $\mathbf{f}_1 = (x_1), \mathbf{f}_2 = (x_2, x_3), \mathbf{f}_3 = (x_4), \mathbf{f}_4 = (x_5)$.

Step 3) Apply $\mathbf{U}(4, 5)$ to get the following five potential offspring:

$$\begin{cases} (3.900, 2.450, 0.550, 7.500, 5.275) \\ (5.100, 4.700, 2.300, 6.250, 2.425) \\ (6.300, 1.325, -0.325, 5.000, 6.700) \\ (7.500, 3.575, 1.425, 3.750, 3.850) \\ (2.700, 0.200, -1.200, 2.500, 1.000). \end{cases}$$

D. Selection Scheme for Population Evolution

We evolve the population by crossover and mutation. To produce a new generation of population, it is necessary to select the parents for crossover and then select some of the potential offspring to form the new generation. In this subsection, we describe how to perform selection.

We adopt D_1 fitness functions to provide D_1 search directions, where D_1 is a design parameter. D_1 is smaller than D_0, because the solution space defined by two parents is usually much smaller than the feasible solution space and we adopt a smaller number of fitness functions to reduce the computation time.

To evolve a new generation, we execute the crossover operation D_1 times. In the ith execution, we select the best parent based on the ith fitness function, select another parent randomly, and then perform crossover on these two parents.

After performing crossover and mutation, a set of potential offspring are generated. Among these potential offspring and the parents, we select G of them to form the next generation. Based on each of the D_0 fitness functions, we select the best $\lfloor G/D_0 \rfloor$ or $\lceil G/D_0 \rceil$ chromosomes, such that the total number of selected chromosomes is G.

E. Genetic Algorithm for Multiobjective Programming

We execute *Algorithm 2* to generate a good initial population with G chromosomes. Then we evolve and improve the population iteratively using the proposed crossover operator, the mutation operator, and the proposed selection scheme.

We let $\mathbf{P}_{\text{gen}}$ be a population of chromosomes in the ith generation. The details of the genetic algorithm for multiobjective programming are as follows.

Genetic Algorithm for Multiobjective Programming

Step 1) **Generation of Initial Population**

Step 1.1) Determine the uniform arrays $\mathbf{U}(M, D_0)$ and $\mathbf{U}(N, D_0)$ based on (7).

Step 1.2) Compose D_0 fitness functions based on $\mathbf{U}(M, D_0)$ and (10)–(11).

Step 1.3) Execute *Algorithm 2* to generate an initial population $\mathbf{P}_0$. Initialize the generation number *gen* to 0.

Step 2) **Initialization for Population Evolution**

Step 2.1) Determine the uniform array $\mathbf{U}(M, D_1)$ and $\mathbf{U}(F, Q_1)$ based on (7).

Step 2.2) Compose D_1 fitness functions based on $\mathbf{U}(M, D_1)$ and (10)–(11).

TABLE III
NUMBER OF PARETO-OPTIMAL SOLUTIONS FOUND AND THE NUMBER OF FUNCTION EVALUATION REQUIRED FOR TEST PROBLEM 1

Execution	Number of Pareto-optimal solutions found		Number of function evaluations	
	UGA	HGA	UGA	HGA
1st	291	25	1674	11967
2nd	237	28	1683	11981
3rd	237	39	1680	11982
4th	252	26	1669	11959
5th	155	35	1636	11979

Step 3) **Population Evolution**

WHILE (stopping condition is not met) *DO*

BEGIN

Step 3.1) **Crossover**

Execute the crossover operation D_1 times. In the ith execution, select the best parent based on the ith fitness function, select another parent randomly, and execute *Algorithm 3* to perform crossover on these two parents.

Step 3.2) **Mutation**

Each chromosome in $\mathbf{P}_{\text{gen}}$ undergoes mutation with probability p_m. To perform mutation on a chromosome, randomly generate an integer $j \in [1, N]$ and a real number $r \in [l_j, u_j]$ and then replace the jth component of the chosen chromosome by r to get a new chromosome.

Step 3.3) **Selection**

Consider the chromosomes in $\mathbf{P}_{\text{gen}}$ and those generated by crossover and mutation. Adopt each of the D_0 fitness functions to select the best $\lfloor G/D_0 \rfloor$ or $\lceil G/D_0 \rceil$ chromosomes for the next generation, such that the total number of selected chromosomes is G.

Step 3.4) Increment the generation number *gen* by 1.

END

In step 3, the population is evolved and improved iteratively until a stopping condition is met. Similar to the other genetic algorithms, there can be many possible stopping conditions. For example, one possible stopping condition is to stop when the best chromosome based on each of the D_0 fitness functions cannot be further improved in a certain number of generation.

The above algorithm has several design parameters. In Table II, we summarize these design parameters and their feasible values. We remind that some of these design parameters correspond to the number of levels in the uniform design. Since only those uniform arrays with prime number of levels have been found and tabulated in the literature, these design parameters are prime.

IV. NUMERICAL RESULTS

We execute the proposed algorithm and the hybrid genetic algorithm [1] to solve three test problems, and we compare their performance.

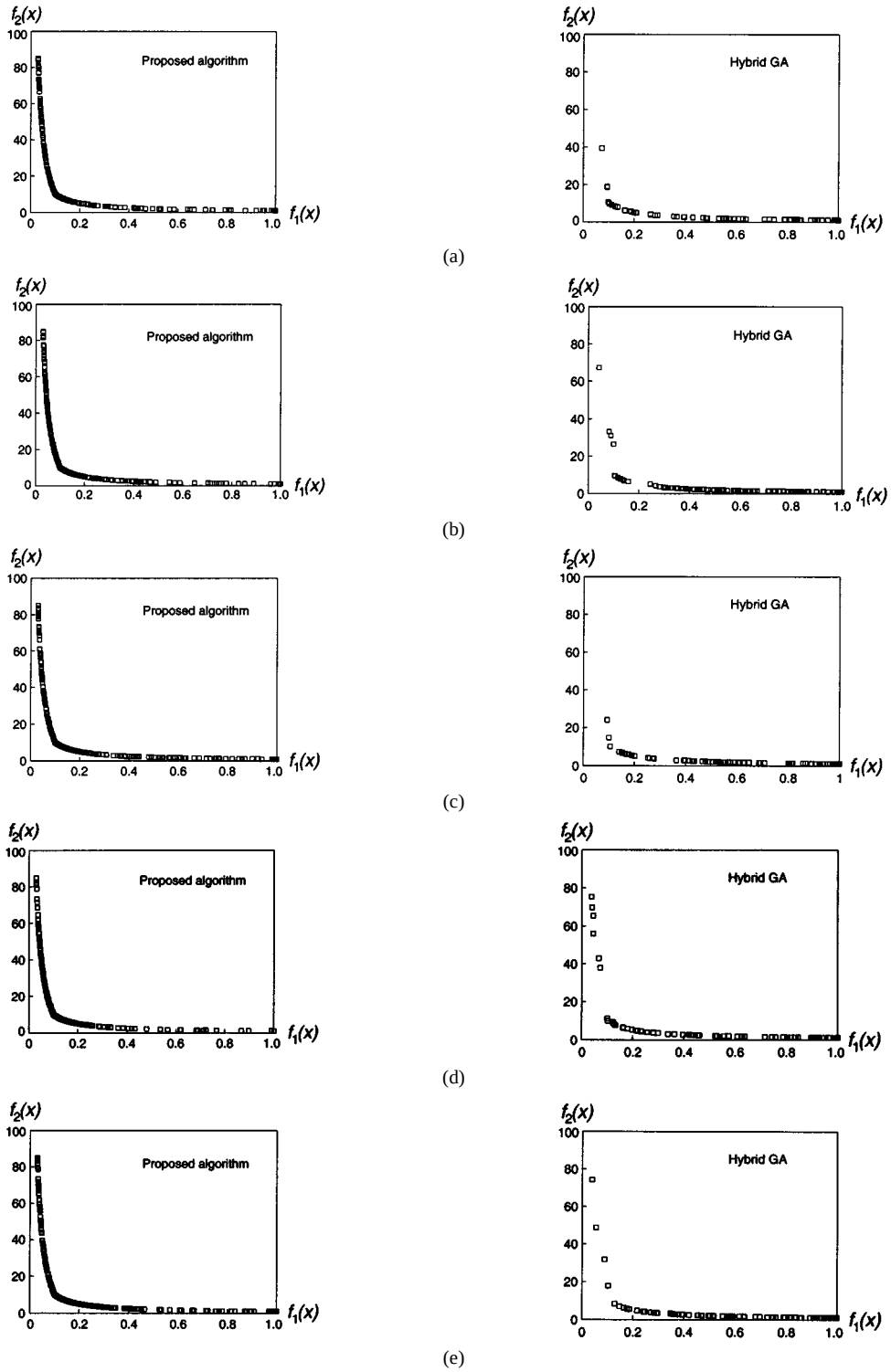

Fig. 6. Pareto-optimal solutions in the objective space for the second test problem.

A. Test Problems

1) Test Problem 1 [1]:

$$\begin{cases} \text{Minimize} & f_1(\mathbf{x}) = 2\sqrt{x_1} \\ \text{Minimize} & f_2(\mathbf{x}) = x_1(1 - x_2) + 5 \\ \text{Subject to:} & 1 \le x_1 \le 4 \\ & -20 \le x_2 \le 10 \end{cases}$$

where $\mathbf{x} = (x_1, x_2)$. This problem was tested in [1], but we enlarge the domain of x_2 from $[1, 2]$ to $[-20, 10]$ in order to increase the number of Pareto-optimal solutions. The resulting problem is more challenging.

2) Test Problem 2 [22]:

$$\begin{cases} \text{Minimize} & f_1(\mathbf{x}) = \frac{1}{x_1^2 + x_2^2 + 1} \\ \text{Minimize} & f_2(\mathbf{x}) = x_1^2 + 3x_2^2 + 1 \\ \text{Subject to:} & -3 \le x_1 \le 3 \\ & -5 \le x_2 \le 5 \end{cases}$$

where $\mathbf{x} = (x_1, x_2)$. This problem was tested in [22], but we enlarge the domain of x_2 from $[-3, 3]$ to $[-5, 5]$ in order to increase the number of Pareto-optimal solutions. The resulting problem is more challenging.

3) Test Problem 3 [7]: See the equation at the bottom of the page where $\mathbf{x} = (x_1, x_2)$. This problem is the well-known beam design problem. The first objective is to minimize the volume of the beam, and the second objective is to minimize the static compliance of the beam [7].

B. Parameter Values

We adopt the following parameter values.

- *Population size*: The population size G is 200.
- *Parameters for generating initial population and selection*: The feasible solution space is divided into $S = 16$ subspaces. We adopt $D_0 = 31$ fitness functions and $Q_0 = 31$ levels per domain.
- *Parameters for crossover and mutation*: We adopt $F = N, D_1 = 7, Q_1 = 5$, and $p_m = 0.02$.
- *Stopping condition*: The execution is stopped after 20 generations.

For the hybrid genetic algorithm, we adopt the following parameter values: the population size is 200, the number of examined neighborhood solutions per chromosome is 2, the number of elite solutions is 2, and the probability of mutation is 0.02. For more details about these parameters, see [1].

C. Results

For each test problem, we perform five independent executions. We record the following data for each execution:

- all Pareto-optimal solutions;
- number of Pareto-optimal solutions found;
- number of function evaluations required.

For convenience, the proposed genetic algorithm using uniform design is referred to as UGA, and the hybrid genetic algorithm [1] is referred to as HGA.

Fig. 5 shows the Pareto-optimal solutions in the objective space for the first test problem. Compared with the hybrid genetic algorithm, the proposed algorithm can find significantly more Pareto-optimal solutions and these solutions are scattered more uniformly over the entire Pareto frontier. This demonstrates that the proposed fitness functions using uniform design are effective in guiding the search toward the entire Pareto frontier. Nevertheless, Table III shows that the proposed algorithm requires significantly smaller number of function evaluations. On average for the five executions, the proposed algorithm requires 1668 function evaluations to find 234 Pareto-optimal solutions, while the hybrid genetic algorithm requires 11 974 function evaluations to find 31 Pareto-optimal solutions. This demonstrates that the proposed genetic algorithm using uniform design can effectively search the solution space.

TABLE IV
NUMBER OF PARETO-OPTIMAL SOLUTIONS FOUND AND THE NUMBER OF FUNCTION EVALUATION REQUIRED FOR TEST PROBLEM 2

Execution	Number of Pareto-optimal solutions found		Number of function evaluations	
	UGA	HGA	UGA	HGA
1st	290	63	1867	11966
2nd	298	112	1842	11975
3rd	303	78	1899	11982
4th	267	75	1878	11981
5th	284	80	1854	11975

Fig. 6 shows the Pareto-optimal solutions in the objective space for the second test problem. Compared with the hybrid genetic algorithm, the proposed algorithm can find significantly more Pareto-optimal solutions which are scattered more uniformly over the entire Pareto frontier. Nevertheless, Table IV shows that the proposed algorithm requires significantly smaller number of function evaluations. On average, the proposed algorithm requires 1868 function evaluations to find 288 Pareto-optimal solutions, while the hybrid genetic algorithm requires 11 976 function evaluations to find 82 Pareto-optimal solutions.

Fig. 7 and Table V show the results for the third test problem. These results confirm the competence of the proposed algorithm. Compared with the hybrid genetic algorithm, the proposed algorithm can find significantly more Pareto-optimal solutions using significantly fewer function evaluations, while these solutions are scattered more uniformly over the entire Pareto frontier.

V. CONCLUSION

We designed a genetic algorithm to find the Pareto-optimal solutions scattered uniformly over the Pareto frontier, so that it can provide a variety of compromise solutions to the decision maker. We applied the uniform design to compose multiple fitness functions and designed a selection scheme using these fitness functions, so that the resulting search directions are scattered uniformly toward the Pareto frontier in the objec-

$$\begin{cases} \text{Minimize} & f_1(\mathbf{x}) = 0.758\left[x_1\left(6.4\times10^3 - x_2^2\right) + \left(10^3 - x_1\right)\left(10^4 - x_2^2\right)\right] \\ \text{Minimize} & f_2(\mathbf{x}) = 3.298\times10^{-5}\left[\left(\frac{1}{4.096\times10^7 - x_2^4} - \frac{1}{10^8 - x_2^4}\right)x_1^3 + \frac{10^9}{10^8 - x_2^4}\right] \\ \text{Subject to:} & 40 \le x_2 \le 75.2 \\ & 0 \le x_1 \le \frac{180\times\left(4.096\times10^7 - x_2^4\right)}{9.78\times10^6} \end{cases}$$

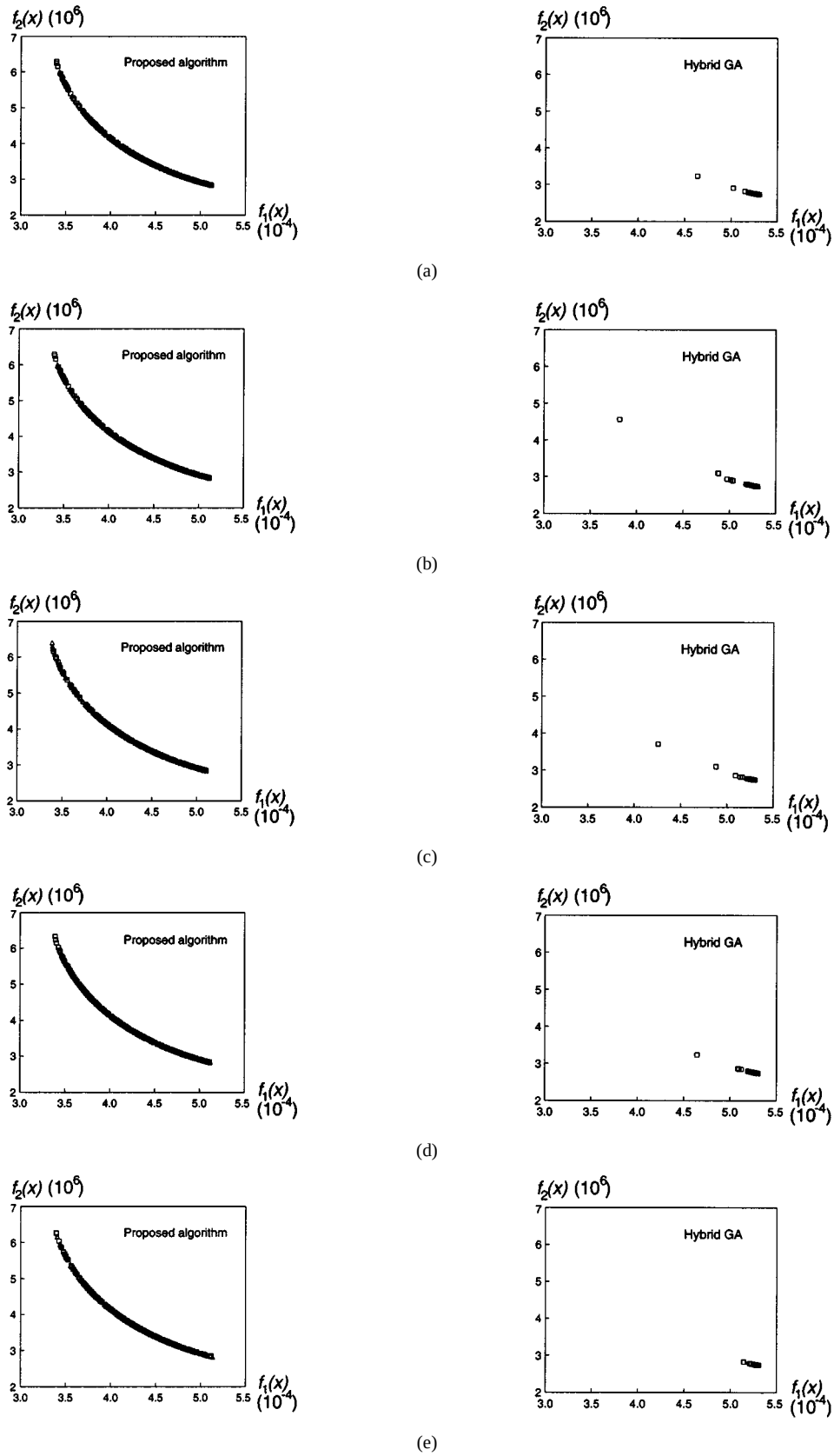

Fig. 7. Pareto-optimal solutions in the objective space for the third test problem.

tive space. In addition, we applied the uniform design to generate a good initial population and design a new crossover operator for searching the Pareto-optimal solutions. We executed the proposed algorithm and the hybrid genetic algorithm [1] to solve three test problems. The results demonstrated that the proposed algorithm can find larger numbers of Pareto-optimal so-

TABLE V
NUMBER OF PARETO-OPTIMAL SOLUTIONS FOUND AND THE NUMBER OF FUNCTION EVALUATION REQUIRED FOR TEST PROBLEM 3

Execution	Number of Pareto-optimal solutions found		Number of function evaluations	
	UGA	HGA	UGA	HGA
1st	864	35	1687	11970
2nd	858	54	1674	11982
3rd	865	49	1658	11969
4th	855	40	1684	11977
5th	849	27	1683	11983

lutions using smaller numbers of function evaluations, and these Pareto-optimal solutions are scattered more uniformly over the Pareto frontier.

REFERENCES

[1] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Trans. Syst., Man, Cybern. C*, vol. 28, pp. 392–403, Aug. 1998.

[2] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms—Part I: Unified formulation," *IEEE Trans. Syst., Man, Cybern. A*, vol. 28, pp. 26–37, Jan. 1998.

[3] ——, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms—Part II: Application example," *IEEE Trans. Syst., Man, Cybern. A*, vol. 28, pp. 38–47, Jan. 1998.

[4] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," in *Proc. 1st IEEE Int. Conf. Evolutionary Computation*, 1994, pp. 82–87.

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

[6] J. Horn and N. Nafpliotis, "Multiobjective optimization using the niched Pareto genetic algorithm," Univ. Illinois, Urbana-Champaign, IlliGAL Rep. 93 005, 1993.

[7] A. M. Sultan and A. B. Templeman, "Generation of Pareto solutions by entropy-based methods," in *Multiobjective Programming and Goal Programming: Theories and Applications*, M. Tamiz, Ed. Berlin, Germany: Springer-Verlag, 1996, pp. 164–195.

[8] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms for multiobjective optimization," *Evol. Comput.*, vol. 3, no. 1, pp. 1–16, 1995.

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

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

[11] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[12] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Networks*, vol. 5, pp. 3–14, Jan. 1994.

[13] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals," *Univ. Comput.*, vol. 15, pp. 58–69, 1993.

[14] ——, "An overview of genetic algorithms: Part 2, research topics," *Univ. Comput.*, vol. 15, pp. 170–181, 1993.

[15] D. C. Montgomery, *Design and Analysis of Experiments*, 3rd ed. New York: Wiley, 1991.

[16] C. R. Hicks, *Fundamental Concepts in the Design of Experiments*, 4th ed. New York: Saunders, 1993.

[17] Y. Wang and K. T. Fang, "A note on uniform distribution and experimental design," *KEXUE TONGBAO*, vol. 26, no. 6, pp. 485–489, 1981. In Chinese.

[18] K. T. Fang and J. K. Li, "Some New Uniform Designs," Hong Kong Baptist Univ., Hong Kong, Tech. Rep. Math-042, 1994.

[19] K. T. Fang and Y. Wang, *Number-Theoretic Methods in Statistics*, London, U.K.: Chapman & Hall, 1994.

[20] K. T. Fang, *Uniform Design and Design Tables*, Beijing, China: Science, 1994. in Chinese.

[21] P. Winker and K. T. Fang, "Application of threshold accepting to the evaluation of the discrepancy of a set of points," *SIAM J. Numer. Anal.*, vol. 34, pp. 2038–2042, 1998.

[22] V. R. Manuel and U. C. Eduardo, "A nongenerational genetic algorithm for multiobjective optimization," in *Proc. 7th Int. Conf. Genetic Algorithms*, 1997, pp. 658–665.

[23] Y. W. Leung and Q. Zhang, "Evolutionary algorithms+experimental design methods: A hybrid approach for hard optimization and search problems," *Res. Grant Prop.*, 1997.

[24] Q. Zhang and Y. W. Leung, "An orthogonal genetic algorithm for multimedia multicast routing," Dept. Comput. Sci., Hong Kong Baptist Univ., Tech. Rep., 1998.

[25] Y. W. Leung and Y. Wang, "An orthogonal genetic algorithm for global numerical optimization," Dept. Comput. Sci., Hong Kong Baptist Univ., Hong Kong, Tech. Rep., 1998.

Yiu-Wing Leung (M'92–SM'96) received the B.Sc. and Ph.D. degrees from the Chinese University of Hong Kong, Hong Kong, in 1989 and 1992, respectively.

He was with the Hong Kong Polytechnic University until 1997, when he joined the Department of Computer Science, Hong Kong Baptist University, where he is now an Associate Professor. His current research interests are in two major areas: information networks and systems and cybernetics. He has published more than 50 journal papers in these areas, most of which appear in various IEEE publications.

Yuping Wang received the B.Sc. degree in mathematics from Northwest University, China, in 1983, and the Ph.D. degree in computational mathematics from Xi'an Jiaotong University, China, in 1993.

He is currently a Professor with the Department of Applied Mathematics, Xidian University, Xi'an, China. He was a Visiting Research Scholar with the Chinese University of Hong Kong from January 1997 to July 1997, and at Hong Kong Baptist University from April 1998 to December 1998. His current research interests include evolutionary computation, optimization theory, algorithms, and applications.