1998 Second International Conference on Knowledge-Based Intelligent Electronic Systems, 21-23 April 1998, Adelaide, Australia. Editors, L.C. Jain and R.K.Jain

An Interactive Fuzzy Satisficing Method for Multiobjective Integer Programming Problems through Genetic Algorithms

†Masatoshi SAKAWA*1, ‡Toshihiro SHIBANO*2 and Kosuke KATO*1
*1Department of Industrial and Systems Engineering, Hiroshima University, JAPAN
*2System Products Division, Shinryo Corporation, JAPAN

Abstract **— This paper deals with multiobjective integer programming problems by considering fuzzy goals of the decision maker for objective functions. After determining the fuzzy goals of the decision maker, if the decision maker specifies the reference membership values, the corresponding Pareto optimal solution can be obtained by solving the augmented minimax problem which becomes an integer programming problem. For solving the problem, decoding algorithms for 0-1 programming problems are revised and ringed double strings are also introduced. Then an interactive fuzzy satisficing method is presented together with an illustrative numerical example.**

KeyWords **— Multiobjective integer programming problem, fuzzy goals, genetic algorithms, ringed double strings, interactive methods**

1. Introduction

In the 1970s, genetic algorithms (GAs) were proposed by Holland, his colleagues and his students at the University of Michigan as a new learning paradigm that models a natural evolution mechanism [2]. Although GAs were not much known at the beginning, after the publication of Goldberg's book [3], GAs have recently attracted considerable attention in a number of fields as a methodology for optimization, adaptation and learning. As we look at recent applications of GAs to optimization problems, especially to various kind of single-objective combinatorial optimization problems and/or to other difficult optimization problems with nonlinear multimodal functions, we can see continuing advances [1], [4], [8]. In recent years, Sakawa et al. have formulated multiobjective 0-1 programming problems incorporating the fuzzy goals of the decision maker (DM) and introduced a genetic algorithm with double strings for deriving a compromise solutions for the DM by adopting the fuzzy decision for combining the fuzzy goals of the DM [5]. Furthermore, Sakawa et al. incorporated interactive techniques into the proposed method [6], [7].

Under these circumstances, in this paper, we focus on multiobjective integer programming problems and propose an interactive fuzzy satisficing method. In the proposed method, after determining the linear membership functions for the fuzzy goals of the DM, a satisficing solution for the DM can be derived from (M-)Pareto optimal solutions which are close to reference points in the membership function space through interactive updates of reference points. As a method to obtain (M-)Pareto optimal solutions, we adopt GAs with double string representation [5–7]. For the GAs, we propose unbiased partially matched crossover (PMX) by looping double strings (ringed double strings) and a new decoding algorithm of double strings that can deal with integer decision variables. Moreover, for the purpose of more efficient search, individuals corresponding to optimal solution for each objective function are introduced into population.

2. Problem formulation

Consider a multiobjective integer programming problem formulated as:

$$\left.\begin{array}{ll} \text{minimize} & z_1(\boldsymbol{x}) = \boldsymbol{c}_1\boldsymbol{x} \\ \vdots & \vdots \\ \text{minimize} & z_k(\boldsymbol{x}) = \boldsymbol{c}_k\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \le \boldsymbol{b} \\ & x_j \in \{0,\ldots,\nu_j\};\ j = 1,\ldots,n \end{array}\right\} \quad (1)$$

where $\boldsymbol{c}_i = (c_{i1},\ldots,c_{in})$, $\boldsymbol{x} = (x_1,\ldots,x_n)^T$, $\boldsymbol{b} = (b_1,\ldots,b_m)^T$, $A = (a_{ij})$ is an $m \times n$ matrix and ν_j's are positive integers. In this paper, it is supposed that each element of the matrix A and the vector $\boldsymbol{b}$ is positive, i.e., we focus on multiobjective multidimensional integer knapsack problems.

In general, however, for multiobjective programming problems, a complete optimal solution which simultaneously minimizes all of the multiple objective functions does not always exist when the objective functions conflict with each other. Thus, instead of a complete optimal solution, Pareto optimality is introduced in multiobjective programming problems [9].

Definition 1 Pareto optimal solution
A feasible solution $\boldsymbol{x}^*$ in the feasible region X is said to be a Pareto optimal solution if and only if there does not exist another $\boldsymbol{x} \in X$ such that $z_i(\boldsymbol{x}) \le z_i(\boldsymbol{x}^*)$, $i = 1,\ldots,k$ and $z_l(\boldsymbol{x}) < z_l(\boldsymbol{x}^*)$ for some l, $1 \le l \le k$.

† Corresponding author
Tel:+81-824-24-7694, Fax:+81-824-24-7694
E-mail: sakawa@msl.sys.hiroshima-u.ac.jp
‡ Presenter, 2-2-1-1 Minato-Mirai, Nishi-Ku, Yokohama, 220-81, JAPAN
E-mail: shibano@shinryo.super-nova.co.jp

0-7803-4316-6/98/$10.00

As immediately understood from Definition 1, in general, there exist a lot of Pareto optimal solutions to a multiobjective integer programming problem. Thus, the DM must select a compromise or satisficing solution from the Pareto optimal solution set according to his subjective judgements.

For the problem (1), in order to consider the ambiguity of the DM's judgements as a human, fuzzy goals for objective functions such as "$z_i(\boldsymbol{x})$ should be substantially less than or equal to a certain value." are incorporated.

Linear membership function such that the objective function value z_i^0 and z_i^1 where the membership values are equal to 0 and 1 respectively in the range from the individual minimum and maximum for each objective function are connected by a straight line segment is often used (Fig. 1).

$$\mu_i(z_i(\boldsymbol{x})) = \begin{cases} 0 & ; z_i(\boldsymbol{x}) > z_i^0 \\ \dfrac{z_i(\boldsymbol{x}) - z_i^0}{z_i^1 - z_i^0} & ; z_i^1 < z_i(\boldsymbol{x}) \leq z_i^0 \\ 1 & ; z_i(\boldsymbol{x}) \leq z_i^1 \end{cases}$$

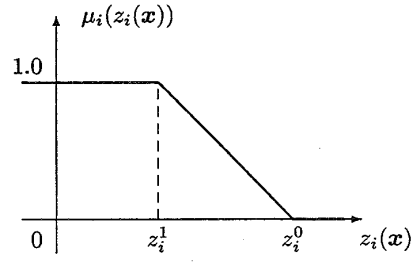

Figure 1. Linear membership function.

As one of the possible ways to help the DM determine z_i^0 and z_i^1, it is convenient to calculate the minimal value $z_i^{\min}$ and the maximal value $z_i^{\max}$ of each objective function under the given constraints. Then by taking account of the calculated individual minimum and maximum of each objective function, the DM is asked to assess z_i^0 and z_i^1 in the interval $[z_i^{\min}, z_i^{\max}]$, $i = 1, ..., k$.

Zimmermann suggested a way to determine the linear membership function $\mu_i(z_i(\boldsymbol{x}))$. To be more specific, using the individual minimum

$$z_i^{\min} = z_i(\boldsymbol{x}^{io}) = \min_{\boldsymbol{x} \in X} z_i(\boldsymbol{x}),\ i = 1, \ldots, k, \quad (2)$$

together with

$$z_i^{\mathrm{m}} = \max(z_i(\boldsymbol{x}^{1o}), \ldots, z_i(\boldsymbol{x}^{i-1,o}), z_i(\boldsymbol{x}^{i+1,o}), \ldots, z_i(\boldsymbol{x}^{ko})), \quad (3)$$

he determined the linear membership function by choosing $z_i^1 = z_i^{\min}$ and $z_i^0 = z_i^{\mathrm{m}}$.

Having elicited the linear membership functions $\mu_i(z_i(\boldsymbol{x}))$ from the DM for each of the objective functions $z_i(\boldsymbol{x})$, $i = 1, \ldots, k$, in the multiobjective integer programming problem (1), by using an aggregation function $\mu_D(\cdot)$ the decision making problem considering k conflicting membership functions can be formally defined as [9], [10]:

$$\underset{\boldsymbol{x} \in X}{\text{maximize}} (\mu_1(z_1(\boldsymbol{x})), \ldots, \mu_k(z_k(\boldsymbol{x}))) \quad (4)$$

where the aggregation function $\mu_D(\cdot)$ is supposed to be strictly monotone increasing and continuous in general. If the form of the function $\mu_D(\cdot)$ can be identified explicitly, this problem can be reduced to an ordinary single-objective programming problem. However, it is so difficult to identify $\mu_D(\cdot)$ globally that the DM is required to select a satisficing solution from a Pareto optimal solution set through interactions.

3. Augmented minimax problems

Assume that the reference membership levels $\bar{\mu}_i$, $i = 1, \ldots, k$, reflecting the aspiration level of the DM for each membership function $\mu_i(\boldsymbol{x})$ is subjectively specified by the DM. Then, if the reference membership levels are attainable, a better Pareto optimal solution than the reference membership levels is obtained. While, if not, it is desirable to obtain a Pareto optimal solution which is nearest to the reference membership levels in the minimax sense.

Such a Pareto optimal solution can be obtained by solving the following minimax problem [9], [10]:

$$\left.\begin{array}{ll} \text{minimize} & \max\limits_{i=1,\ldots,k} \{\bar{\mu}_i - \mu_i(z_i(\boldsymbol{x}))\} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & x_j \in \{0, \ldots, \nu_j\},\ j = 1, \ldots, n \end{array}\right\} \quad (5)$$

It must be noted here that, for generating Pareto optimal solutions by solving the minimax problem, if the uniqueness of the optimal solution is not guaranteed, it is necessary to perform the Pareto optimality test. To circumvent the necessity to perform the Pareto optimality test in the minimax problems, it is reasonable to use the following augmented minimax problem instead of the minimax problem (5):

$$\left.\begin{array}{ll} \text{minimize} & \max\limits_{i=1,\ldots,k} \left\{ (\bar{\mu}_i - \mu_i(z_i(\boldsymbol{x}))) + \rho \sum\limits_{i=1}^{k} (\bar{\mu}_i - \mu_i(z_i(\boldsymbol{x}))) \right\} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & x_j \in \{0, \ldots, \nu_j\},\ j = 1, \ldots, n \end{array}\right\} \quad (6)$$

where ρ is a sufficiently small positive number. By solving the above augmented minimax problem (6), a Pareto optimal solution which is nearest to the reference membership levels in the minimax sense can be obtained regardless of its uniqueness [9], [10].

Since the augmented minimax problem (6) is an integer programming problem whose objective function and constraints are linear and all coefficients in the constraints are positive, we attempt

to develop genetic algorithms using double string representation for multidimensional 0-1 knapsack problems proposed by Sakawa et al. [5–8].

4. Genetic algorithms using double string representation

4.1. coding

For multiobjective 0-1 programming problems where all elements of coefficients A and $\boldsymbol{b}$ in the linear constraints $A\boldsymbol{x} \leq \boldsymbol{b}$ are positive, coding by double string in which each element in the upper string denotes the index of a variable and each element in the lower string denotes the value of a variable as shown in Fig. 2 have been proposed [5], [6], [10]. where $s(i)$ corresponds to the index j

Indices	$s(1)$	$s(2)$	$\cdots$	$s(i)$	$\cdots$	$s(n)$
Values	$x_{s(1)}$	$x_{s(2)}$	$\cdots$	$x_{s(i)}$	$\cdots$	$x_{s(n)}$

Figure 2. Double string.

of a variable x_j and $x_{s(i)}$ denotes the value of the variable x_j. In the decoding algorithm for double strings proposed by M. Sakawa et al. [5], [6], [10] for 0-1 programming problems where all coefficients in the linear constraints are positive, starting from the left edge of the string, the value of a variable corresponding to each index $s(i)$ is fixed to $x_{s(i)}$ until the constraints are broken. For remaining variables, if the constraints are broken when $x_{s(i)} = 1$, the value of the variable with the index $s(i)$ is fixed to 0 by force. By doing so, only feasible solutions will be generated.

Now the following decoding algorithm for integer programming problems is proposed by developing the decoding algorithm mentioned above.

Decoding algorithm for double strings

Step 1 Let $i = 1$, $sum_j = 0$ $(j = 1, \ldots, m)$.

Step 2 The value of the variable with the index $s(i)$ is fixed to $p_{s(i)}$ determined by

$$p_{s(i)} = \min\left(\min_{\{j|a_{js(i)}\neq 0\}}\left[\frac{b_j - sum_j}{a_{js(i)}}\right], x_{s(i)}\right)$$

where $a_{js(i)}$'s are coefficients in the constraints.

Step 3 Let $sum_j = sum_j + a_{js(i)}p_{s(i)}$, $j = 1, \ldots, m$ and $i = i + 1$.

Step 4 If $i > n$, stop. Otherwise, return to step 2.

Note that, according to the above decoding algorithms for double strings, a gene has less influence on phenotype with progression of i. In order to consider the imbalance of the influence between genes, ringed double strings illustrated in Fig. 3 are introduced.

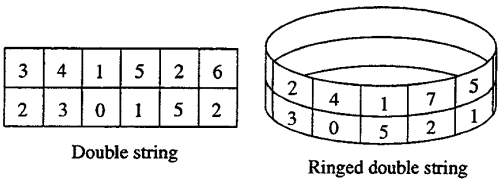

Figure 3. Double string and ringed double string.

4.2. Fitness

The fitness $f(s)$ for each individual s is defined as

$$f(s) = 1.0 + k\rho - \max_{i=1,\ldots,k}\Big\{(\bar{\mu}_i - \mu_i(z_i(\boldsymbol{x}))) + \rho\sum_{i=1}^{k}(\bar{\mu}_i - \mu_i(z_i(\boldsymbol{x})))\Big\}.$$

As a way of scaling of fitness, the linear scaling $f_i' = a \cdot f_i + b$ is adopted, where the constants a, b are determined so that the mean fitness of the population will be the fixed point and the max fitness will be mapped to the twice value of the mean fitness.

4.3. Reproduction

Various kinds of reproduction methods have been proposed. Among them, the authors have already investigated the performance of each of six reproduction operators, i.e., ranking selection, elitist ranking selection, expected value selection, elitist expected value selection, roulette wheel selection and elitist roulette wheel selection, and as a result, it was confirmed that elitist expected value selection is relatively efficient for multiobjective 0-1 programming problems incorporating the fuzzy goals of the decision maker [5], [6], [8]. In this paper, based mainly on our experience, as a reproduction operator, elitist expected value selection, elitism and expected value selection are combined together, is adopted.

4.4. Crossover

If a single-point crossover or multi-point crossover is directly applied to individuals of double string type, the index $s(i)$ in the ith gene of an offspring may take the same number that an index $s(i')$ $(i \neq i')$ takes. The same violation occurs in solving traveling salesman problem or scheduling problem through genetic algorithm as well. In order to avoid this violation, a crossover method called partially matched crossover (PMX) is modified to be suitable for double strings as follows.

PMX for double strings

Step 1 Choose two parent strings X and Y and let the ith column of X and Y be $(s_X(i), x_{s_X(i)})^T$

and $(s_Y(i), x_{s_Y(i)})^T$ respectively. Then, determine two crossover points h and k randomly and let $i = h$.

Step 2 According to the following procedures (1) and (2), make a new string X' by rearranging X so that the upper row of the substring of X between h and k will be identical with that of Y between h and k.

(1) After finding j satisfying $s_Y(i) = s_X(j)$, exchange the ith column $(s_X(i), x_{s_X(i)})^T$ of X for the jth column $(s_X(j), x_{s_X(j)})^T$ of X and let $i = i + 1$.

(2) If $i > k$, stop. Otherwise return to (1).

Then, make Y' by rearranging Y according to the same procedures as above.

Step 3 Make an offspring X^* by exchanging the substring of X' between h and k for the corresponding substring of Y. Then, make another offspring Y^* by the same operation for Y' as that for X'.

Figure 4 illustrates an example of PMX for double strings. Note the probability that the middle part

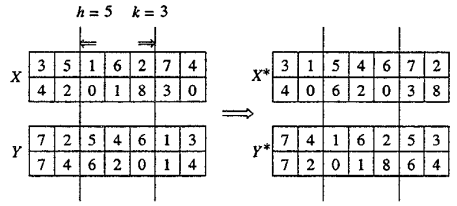

Figure 4. An example of PMX for double strings.

of a string will be exchanged is higher than other parts of a string. In this paper, the following PMX using ringed double strings, where every part of a string will be exchanged equally, is proposed.

PMX for ringed double strings

Step 1 Choose two parent strings X and Y and let the ith column of X and Y be $(s_X(i), x_{s_X(i)})^T$ and $(s_Y(i), x_{s_Y(i)})^T$ respectively.

Step 2 Determine two crossover points h and k randomly and let $i = h$.

Step 3 If $i >$ *length* (*length* denotes the length of the string), let $n = i -$ *length*. Otherwise, let $n = i$.

Step 4 After finding j satisfying $s_Y(n) = s_X(j)$ and exchange the ith column $(s_X(i), x_{s_X(i)})^T$ of X for the jth column $(s_X(j), x_{s_X(j)})^T$ of X. Then exchanging the substring of X between h and k for the corresponding substring of Y. Let $i = i + 1$.

Step 5 If $i > k$, stop. Otherwise, return to step 3.

Step 6 Perform the same operation for Y. The obtained strings X^* and Y^* are regarded as offsprings.

Figure 4 illustrates an example of PMX for ringed double strings.

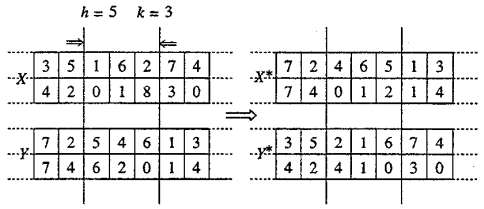

Figure 5. An example of PMX for ringed double strings.

4.5. Mutation and inversion

In the mutation operation, after choosing the mutation point randomly, determine the value of the variable $x_{s(i)}$ randomly. Here, $0 \le x_{s(i)} \le \nu_{s(i)}$ ($\nu_{s(i)}$ denotes the maximal value of $x_{s(i)}$ which satisfies all constraints). Furthermore, the inversion operation defined by the following algorithms is adopted in this paper.

Algorithm of mutation

Step 1 For each column of a double string, generate a real random number $rand() \in [0, 1]$.

Step 2 If $p_m \ge rand()$, determine the value of the lower element of the column. Here, $0 \le x_{s(i)} \le \nu_{s(i)}$ ($\nu_{s(i)}$ denotes the maximal value of $x_{s(i)}$ satisfies all constraints).

Step 3 Perform the procedures (1), (2) to all strings in the population.

Algorithm of inversion

Step 1 After determining two inversion points h and k (for ordinary double strings, the condition $h < k$ is added), pick out the part of the string from h to k.

Step 2 Arrange the substring in reverse order.

Step 3 Put the arranged substring back in the string.

5. An interactive fuzzy satisficing method

In general, the search space for an integer programming problem is much larger than that for a 0-1 programming problem. For example, if the number of variables is 20, the search space for 0-1 decision variables is 2^{20} while that for integer decision variables $\in \{0, 1, \ldots, 9\}$ is 10^{20}. In applying genetic algorithms to integer programming problems with so large search space, more efficient techniques are required than in case of 0-1 programming problems for genetic algorithms to work well. In this paper, the corresponding individual with the optimal solution for each objective function is forced to be included the population before

crossover after reproduction in every generation.

The fundamental algorithm incorporating the proposed genetic algorithm with double string representation into interactive fuzzy programming methods [9], [10] is summarized as follows.

Step 1 Set initial reference membership levels (if it is difficult to determine these values, set them to 1.0).

Step 2 Generate the initial population involving N individuals of (ringed) double string type at random.

Step 3 Calculate the fitness for each individual and apply reproduction operator based on the fitness.

Step 4 Insert the corresponding individual with the optimal solution for each objective function into the current population.

Step 5 Apply crossover operator according to the probability of crossover p_c.

Step 6 Apply mutation operator according to the probability of mutation p_m.

Step 7 Repeat a series of procedures from step 3 to step 6 until the termination condition is satisfied. If it is satisfied, regard the individual with the maximal fitness as the optimal individual and go to step 8.

Step 8 If the DM is satisfied with the current values of membership functions and objective functions given by the current optimal individual, stop. Otherwise, ask the DM to update reference membership levels by taking account of the current values of membership functions and objective functions and return to step 2.

Here, in the generation of the initial population at each interaction, the (approximate) optimal individual in the previous interaction is incorporated into the initial population. By using both the elitist expected value selection and the way of generating the initial population mentioned above, it is expected that the (approximate) optimal solution obtained in the current interaction will not be dominated by that in the previous interaction.

6. Comparison between double string and ringed double string

6.1. *The way of generating numerical examples*

As numerical examples, 15 dimensional integer knapsack problems with 20 variables were generated through the following procedures.

(1) Determine each of a_{ij}s by a random number according to Gaussian distribution with the mean $\mu = 300$ and the standard deviation $\sigma = 50$.

(2) Let each of b_is be a real number by multiplying a random number in $[1.25, 1.75]$ by $\sum_{j=1}^{n} a_{ij}$.

(3) Determine c_{ij}s in the same manner as a_{ij}s, where c_{1j}s and c_{2j}s are supposed to be all positive and negative respectively while c_{ij}s are supposed to be positive and negative half-and-half.

An example of problems generated by the above procedures is shown in Table 1.

6.2. *Results of simulations*

For the numerical example shown in Table 1, simulations were performed twenty times for both cases using the ordinary double string representation and the ringed double string representation. Here, the number of individuals 100, the probability of crossover $p_c = 0.8$, the probability of mutation $p_m = 0.02$, the maximal generation 300 and the maximal value of each integer decision variable is 10. The couples of objective function values (z_1^0, z_1^1), (z_2^0, z_2^1), (z_3^0, z_3^1) such as the linear membership function value becomes 0 or 1 were determined as $(9200, 0)$, $(0, -9600)$, $(3000, -9200)$ respectively on the basis of the Zimmermann's method [11].

Table 2 shows the best value, the worst value and the mean value of $\min(\mu_i(z_i(\boldsymbol{x})))$ obtained by two methods of twenty trails. As known from Table 2, the performance of the method using the ringed double string representation was better with respect to the worst value and the mean value.

Table 2. Results of 20 trials.

	Best	Worst	Mean
Ordinary DS	0.575521	0.563804	0.571542
Ringed DS	0.575521	0.567717	0.572829

(DS: double string)

6.3. *Interactive processes*

Based on the simulation results as shown in Table 2, an interactive method through genetic algorithms using the ringed double string representation was applied to the above problem. The same values of parameters in genetic algorithms and the same membership functions in the previous experiment were used. In this experiment, the hypothetical DM updated the reference membership levels $(\bar{\mu}_1, \bar{\mu}_2, \bar{\mu}_3)$ as $(1.0, 1.0, 1.0) \rightarrow (0.8, 1.0, 1.0) \rightarrow (0.8, 1.0, 0.9)$. Table 3 shows the (approximate) optimal solution for each interaction. In the first interaction, the membership values (0.5773, 0.5755, 0.5917) of the (approximate) optimal solution for the reference membership levels $(1.0, 1.0, 1.0)$ were obtained, but the DM was not satisfied with the solution. Then the DM considered that the membership function values μ_2 and μ_3 should be improved at the sacrifice of the membership function value μ_1 and updated the reference membership levels from $(1.0, 1.0, 1.0)$ to $(0.8, 1.0, 1.0)$. In the second interaction, the membership values

Table 1. An example of integer programming problems with 3 objectives and 15 constraints.

a_1	271	331	300	314	256	369	306	226	285	284
	333	379	307	182	312	323	264	288	325	192
a_2	316	202	324	346	328	348	286	263	362	311
	359	340	223	276	376	291	325	296	342	263
a_3	252	265	319	276	158	314	276	277	364	246
	240	316	311	395	371	339	411	300	266	360
a_4	188	288	285	313	306	227	284	273	317	254
	279	340	312	285	286	285	240	399	293	353
a_5	339	252	314	328	269	392	268	367	220	249
	281	295	323	283	286	242	295	270	313	312
a_6	302	267	340	239	377	287	279	282	215	264
	302	421	374	293	227	313	339	353	315	315
a_7	188	260	299	344	352	321	234	317	280	287
	374	353	330	272	297	227	332	291	311	273
a_8	246	349	350	230	297	370	252	265	290	276
	321	237	242	300	260	243	247	411	310	326
a_9	245	299	318	299	307	266	314	301	309	333
	341	315	324	297	344	327	250	367	309	296
a_{10}	211	361	248	217	197	249	300	231	373	320
	256	261	301	302	324	286	391	276	273	298
a_{11}	303	330	328	328	356	284	298	321	269	274
	301	346	289	239	403	252	230	315	192	391
a_{12}	232	277	315	331	262	286	314	369	434	290
	247	318	353	331	315	292	323	250	279	318
a_{13}	198	326	376	289	191	253	239	354	251	298
	315	329	272	255	293	368	320	371	396	293
a_{14}	248	264	275	225	353	310	329	423	306	248
	264	307	251	242	305	274	398	230	204	207
a_{15}	274	335	376	299	334	198	318	305	251	288
	286	273	310	293	226	344	352	236	305	306
c_1	281	327	305	344	371	247	335	336	364	392
	263	280	370	292	231	352	297	254	293	320
c_2	−271	−338	−282	−267	−319	−379	−245	−215	−187	−419
	−360	−302	−266	−324	−315	−307	−322	−366	−332	−273
c_3	−315	−293	−245	−291	282	316	226	297	360	182
	−330	394	−345	276	−321	−219	341	−372	250	324
b	8012	7871	8660	8297	9144	10289	8200	7670	10091	8522
	9934	10432	7959	9623	10108					

$(0.4917, 0.6872, 0.7373)$ of the (approximate) optimal solution for the reference membership levels $(0.8, 1.0, 1.0)$ were obtained. Since the DM hoped more improvement of μ_2, he updated the reference membership levels from $(0.8, 1.0, 1.0)$ to $(0.8, 1.0, 0.9)$. The DM was satisfied with the membership values $(0.4927, 0.6966, 0.6310)$ obtained in the third interaction and the interactive procedure was finished.

In this example, after updating the reference membership levels twice, at the third interaction, the satisficing solution which is also Pareto optimal was derived. Here, since it took about 69 seconds to solve the augmented minimax problem for each of reference membership levels, the proposed method is considered to be sufficiently fast as an interactive method.

Unfortunately, however, at each interaction, (approximate) Pareto optimal solutions were obtained about 2 times out of 10 trials. Concerning this point, further refinements of individual representation, decoding algorithm and so forth will be required.

7. Conclusions

In this paper, we have proposed an interactive fuzzy satisficing method through genetic algorithms for multiobjective integer programming problems. With respect to genetic algorithms, a decoding algorithm to deal with integer variables and a new PMX (partially matched crossover) operations using ringed double string representation were proposed. Furthermore, by inserting the individuals corresponding to the optimal solution for each objectives into the population, we gave a clue to solve an integer programming problem whose solution space may be much larger than that of a 0-1 programming problem. In interaction processes, however, since (approximate) Pareto optimal solutions were obtained about 2 times out of 10 trials, further refinements of individual representation, decoding algorithm and so forth will be expected.

Table 3. Results of interactions.

First interaction	μ_1	μ_2	μ_3	$z_1(x)$	$z_2(x)$	$z_3(x)$	Number of solutions
$\overline{\mu}_1 = 1.0$	0.5773	0.5755	0.5917	3889	−5525	−4219	2
$\overline{\mu}_2 = 1.0$	0.5763	0.5749	0.5882	3898	−5519	−4177	2
$\overline{\mu}_3 = 1.0$	0.5757	0.5742	0.5848	3907	−5513	−4135	1
	0.5740	0.5795	0.6522	3919	−5563	−4958	2
	0.5722	0.6378	0.5736	3923	−5494	−4781	1
	0.5827	0.5707	0.6331	3839	−5479	−4724	1
	0.5677	0.5762	0.5877	3977	−5532	−4170	1
Second interaction	μ_1	μ_2	μ_3	$z_1(x)$	$z_2(x)$	$z_3(x)$	Number of solutions
$\overline{\mu}_1 = 0.8$	0.4917	0.6872	0.7373	4676	−6598	−5939	2
$\overline{\mu}_2 = 1.0$	0.4908	0.6867	0.7292	4685	−6592	−5897	1
$\overline{\mu}_3 = 1.0$	0.4945	0.6899	0.6832	4651	−6623	−5335	2
	0.4829	0.6821	0.7422	4757	−6548	−6055	1
	0.4849	0.6768	0.7169	4727	−6497	−5746	1
	0.4753	0.6734	0.7488	4827	−6465	−6135	1
	0.4701	0.7227	0.6884	4875	−6938	−5398	1
	0.4824	0.6673	0.7579	4762	−6406	−6246	1
Third interaction	μ_1	μ_2	μ_3	$z_1(x)$	$z_2(x)$	$z_3(x)$	Number of solutions
$\overline{\mu}_1 = 0.8$	0.4927	0.6966	0.6310	4667	−6687	−4698	2
$\overline{\mu}_2 = 1.0$	0.4933	0.6900	0.6199	4622	−6624	−4563	2
$\overline{\mu}_3 = 0.9$	0.4945	0.6899	0.6832	4651	−6623	−5335	1
	0.4892	0.6926	0.7369	4699	−6649	−5990	1
	0.4908	0.6867	0.7293	4685	−6592	−5897	1
	0.4836	0.6845	0.6108	4751	−6571	−4452	1
	0.4827	0.6828	0.6824	4759	−6555	−5325	1
	0.4799	0.6935	0.7170	4785	−6658	−5747	1

References

1. M. Gen and R. Cheng, "*Genetic Algorithms and Engineering Design*," John Wiley & Sons, New York, 1997.
2. J.H. Holland, "*Adaptation in Natural and Artificial Systems*," University of Michigan Press, 1975, MIT Press, Cambridge, 1992.
3. D.E. Goldberg, "*Genetic Algorithms in Search, Optimization, and Machine Learning*," Addison Wesley, Massachusetts, 1989.
4. Z. Michalewicz, "*Genetic Algorithms + Data Structures = Evolution Programs*," Springer-Verlag, Berlin, 1992, Second, extended edition, 1994, Third, revised and extended edition, 1996.
5. M. Sakawa, K. Kato, H. Sunada and T. Shibano, "Fuzzy programming for multiobjective 0-1 programming problems through revised genetic algorithms," European Journal of Operational Research, Vol. 8, No. 6, pp. 149-158, 1997.
6. M. Sakawa and T. Shibano, "Interactive fuzzy programming for multiobjective 0-1 programming problems through genetic algorithms with double strings", in Da Ruan (ed.) *Fuzzy Logic Foundations and Industrial Applications*, Kluwer Academic Publishers, Boston, pp. 111-128, 1996.
7. M. Sakawa and T. Shibano, "Multiobjective fuzzy satisficing methods for 0-1 knapsack problems through genetic algorithms", in W. Pedrycz (ed.) *Fuzzy Evolutionary Computation*, Kluwer Academic Publishers, Boston, pp. 155-177, 1997.
8. M. Sakawa and M. Tanaka, "*Genetic algorithms*," Asakura Publishing, Tokyo, 1995 (in Japanese).
9. M. Sakawa, "*Fuzzy Sets and Interactive Multiobjective Optimization*," Plenum Press , New York, 1993.
10. M. Sakawa, H. Ishii and I. Nishizaki, "*Soft Optimization*," Asakura Publishing, Tokyo, 1995 (in Japanese).
11. H.-J. Zimmermann, "Fuzzy programming and linear programming with several objective functions," Fuzzy Sets and Systems, Vol. 1, No. 1, pp. 45-55, 1978.