Electrical Engineering in Japan, Vol. 139, No. 1, 2002
Translated from Denki Gakkai Ronbunshi, Vol. 120-C, No. 6, June 2000, pp. 831–842

Proposal of a Cannibalism Bug-Based Search Strategy Using Genetic Algorithms (C-BUGS) and Its Application to Multiobjective Optimization Problems

KEIICHIRO YASUDA, OSAMU YAMAZAKI, and TAKAO WATANABE
Tokyo Metropolitan University, Japan

SUMMARY

For decision support under a multiobjective environment, it is effective to offer a Pareto optimal solution set with uniform distribution to the decision-maker. In this paper, a new optimization method for obtaining a Pareto optimal solution set with such uniform distribution is proposed. In order to overcome the difficulty of realizing this goal, the concept of cannibalism is introduced in BUGS (a bug-based search strategy using genetic algorithms). Introducing the concept of cannibalism achieves the uniform distribution of Pareto optimal solutions. A numerical experiment using typical continuous and discrete multiobjective optimization problems clarifies the usefulness of the proposed method. © 2002 Scripta Technica, Electr Eng Jpn, 139(1): 51–64, 2002; DOI 10.1002/eej.1146

Key words: Multiobjective optimization, genetic algorithm, bug-based search algorithm, knapsack problem.

1. Introduction

The diversified requirements of modern society are increasing the importance of solving multiobjective optimization problems in which multiple objects must be optimized simultaneously. In a multiobjective optimization problem, there rarely exists a complete optimal solution that optimizes all objective functions. In most cases, the solution is obtained as a set of "Pareto optimal solutions," which are defined as solutions that are at least not inferior to any other solutions. The final decision in the multiobjective optimization problem is made subjectively by the decision-maker, who selects a solution from the Pareto optimal solution set. This is a powerful form of decision-making support that provides the decision-maker with a well-balanced set consisting of an adequate number of Pareto optimal solutions [1].

In the genetic algorithm (GA), which is a mathematical model for optimization that simulates the biological evolutionary process, one of the features is set-based multipoint simultaneous search [2, 7]. There has been an effort to utilize the multipoint simultaneous search to determine well-balanced Pareto optimal solutions directly as a set for the multiobjective optimization problem. Some difficulties have been pointed out, for example, that the balance of the Pareto optimal solutions is adversely affected by the genetic fluctuation of the GA, and that the introduction of the sharing process to avoid such an effect increases the computational complexity [3–5].

BUGS (a bug-based search strategy using genetic algorithms) proposed by Iniwa and Sato [6, 7], on the other hand, is a mathematical optimization model based on the genetic algorithm and a model of the predatory behavior of insects (bugs). The method is based on the correspondence between the objective function to be optimized and the concentration of the bacteria that are the food of the bug. In the evolution of bugs, they have tended to concentrate at the points of maximum concentration of the food bacteria, that is, the maximum of the objective function, in the search for the optimal value. The method can be considered as a combination of adaptive search and hill-climbing. Two of its advantages are that the lack of a local search function in the conventional genetic algorithm is remedied, and that the biological concepts of resource competition and partitioning are introduced in a natural form. In other words, BUGS as an algorithm has the feature that a well-balanced solution distribution can be easily realized.

This paper notes such features of BUGS, and examines the possibility of applying the method to the multiobjective optimization problem, which has not been investigated. It is shown that the algorithm must be improved in order to be applied to the multiobjective optimization problem. In order to achieve a better balance of the solution distribution, it is proposed to introduce a new "concept of cannibalism" in which bugs within a certain neighborhood try to eat each other. Introducing this concept

makes it possible to determine a set of well-balanced Pareto optimal solutions that is adequate in number, while limiting the increase of computational complexity.

In the following, Section 2 outlines BUGS. BUGS is applied to the maximization problem and the region search problem. The behavior of the algorithm is investigated from the viewpoint of the region search. In Section 3, BUGS is applied to the continuous and discrete multiobjective optimization problem, and difficulties when BUGS is applied to the multiobjective optimization problem are pointed out. Based on the discussion up to Section 3, Section 4 proposes C-BUGS, in which the concept of cannibalism is introduced. Section 5 applies C-BUGS to typical continuous and discrete multiobjective optimization problems, and the usefulness of the method is demonstrated.

2. The Bug Search System and Its Behavior

This section outlines the BUGS algorithm proposed by Iniwa and Sato [6, 7]. The algorithm is applied to a maximization problem and a region search problem, and the behavior of the algorithm is investigated from the viewpoint of region search. By region search is meant the search of a region defined by an equality/inequality constraint. We attempt to derive a well-balanced solution set composed of a finite number of search points.

2.1 The BUGS algorithm

In BUGS, an individual performing the search is called a "bug," and the search is realized by a set of bugs. The number of individuals N_t in the set of bugs in each generation changes in the course of the search, with the maximum number of individuals N_{MAX} as the upper limit. The bug in BUGS is described by three parameters, the position, the direction, and the energy:

$$
\begin{aligned}
&Bug_i(t):\\
&\quad \text{position } \vec{X}_i(t) = (x_1^i(t), ..., x_n^i(t))\\
&\quad \text{direction } \vec{DX}_i(t) = (dx_1^i(t), ..., dx_n^i(t))\\
&\quad \text{energy } e_i(t)
\end{aligned}
$$

Here t represents the number of generations and x_i is the value of the i-th element in the search space. $\vec{DX}_i$ is the information used to control the movement of the bug. The position of the bug—the search point—is updated as follows:

$$\vec{X}_i(t+1) = \vec{X}_i(t) + G(\vec{DX}_i(t)) \tag{1}$$

where G is a function that maps the direction control vector $\vec{DX}_i$ to the n-dimensional direction vector. The energy of the i-th bug at generation t is determined by the cumulative value of the degree-of-match function and the energy decay with growth. The gene of the bug is the real vector $\vec{DX}$, and the solution (search point) is represented by $\vec{X}$. The detailed algorithm of BUGS is given in the Appendix.

2.2 Examination of behavior of BUGS by numerical experiment

In this section, BUGS is applied to the maximization problem and the inequality region search problem for a continuous function, and the behavior of the search process is investigated. In applying BUGS, it is necessary to set the coding, movement, degree-of-match function, parameters, initial generation, and auxiliary (exchange) operator. In this paper, when BUGS is also applied to other continuous problems, the following basic set is defined and used, other than the degree-of-match function.

2.2.1 Basic set of continuous BUGS

In this paper, the following setting is defined as the basic set of continuous BUGS. The auxiliary operator of Eq. (7) in the following basic set is the operator, which is newly introduced in this paper. It performs the action of exchanging the parent and child, and is called the "exchange operator." The effect of this operator is discussed in the region search problem in the next section.

[Basic set of continuous BUGS]

(1) Coding

The position of the bug can be used directly as a solution. In other words,

$$\text{position } \vec{X}_i(t) = (x_1^i(t), ..., x_n^i(t))$$

(2) Move

The search point is moved by the following formula:

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \vec{DX}_i(t) \tag{2}$$

In other words, $G(\vec{DX}_i(t)) = \vec{DX}_i(t)$. The direction control vector $\vec{DX}_i$ of the bug is given as

$$\text{direction } \vec{DX}_i(t) = (dx_1^i(t), ..., dx_n^i(t))$$

(3) Degree-of-match function

The degree of match function is defined as

$$F(\vec{X}_i) = \max\{f(x_1^i(t), ..., x_n^i(t)), 0\} \tag{3}$$

where $f(x_1^i(t), \ldots, x_n^i(t))$ is the objective function to be maximized. It is possible to define the degree-of-match function to take negative values.

(4) Parameters

The parameters are set as in Table A.1 of the Appendix.

(5) Composition of initial generation

The initial generation is composed as follows:

set at random in the range $\vec{X}_i(t) = (x_1^i(t), \ldots, x_n^i(t))$: $x_0 \le x_j^i(t) \le x_0, j = 1, \ldots, n$.

set at random in the range $\vec{DX}_i(t) = (dx_1^i(t), \ldots, dx_n^i(t))$: $-x_0/100 \le x_j^i(t) \le x_0/100, j = 1, \ldots, n$.

$e_i(t)$: set as 1.

(6) Crossover and mutation

The crossover is specified as uniform. In the two-dimensional case, for example, when the direction control vectors of parent bugs 1 and 2 are $\vec{DX}_1 = (dx_1^1(t), dx_2^1(t))$ and $\vec{DX}_2 = (dx_1^2(t), dx_2^2(t))$ and the mask pattern is (1,0), the direction control vectors $\vec{DX}_1'$ and $\vec{DX}_2'$ of the child bugs 1 and 2 are given by

$$\vec{DX}_1'(t) = (dx_1^1(t), dx_2^2(t))$$
$$\vec{DX}_2'(t) = (dx_1^2(t), dx_2^1(t))$$

Mutation is applied for each gene. The value is modified in mutation as follows. When $dx_1^1(t)$ is to be modified by mutation, the value is set at random in the range $-x_0/100 \le dx_j^i(t) \le x_0/100, j = 1, \ldots, n$.

(7) Exchange operator

In Eq. (4), the radius of reproduction RR is set as infinity. Consequently, bugs proliferate very rapidly, and the number of individuals soon reaches the upper limit N_{MAX}. Thus, a child cannot join the set when it is born, and a new individual is seldom generated. In order to prevent such a situation, when the number of individuals reaches the maximum and a new child is not added, a parent is deleted from the set and the child is added to the set.

2.2.2 Application of BUGS to continuous function maximization problem

BUGS is now applied to a two-variable nonlinear optimization problem, which is formulated as follows. The optimal solution for this problem is $(x_1, x_2) = (0, 0)$. The parameters are set as in the basic set of continuous BUGS above:

$$\max f(x_1, x_2) = -x_1^2 - x_2^2 + 1 \tag{4}$$

Figure 1 shows an example of the result when BUGS is applied. It is seen that a large number of bugs exist in the area where the degree-of-match function is large, namely, the area where the objective function takes large values, with few bugs in the area with small values. Since the bugs are distributed not only in the neighborhood of the maximum point, but also in a circular form with the origin as the center, it is seen that the search point is easily scattered, which is a basic property of BUGS.

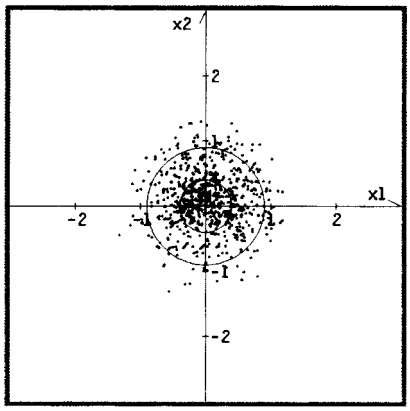

Fig. 1. Application result of BUGS for max-problem.

Thus, it is judged that BUGS is suited to the region search problem, in which a region is searched with good balance by multiple search points, rather than searching the region by a single point. In the following, we attempt to demonstrate this property by a numerical experiment. BUGS is applied to an inequality region search problem, and the condition for well-balanced region search is discussed.

2.2.3 Application of BUGS to inequality region search problem

BUGS is applied to a region search problem in which the region satisfying the following constraint is sought:

$$g(x_1, x_2) = x_1^2 + x_2^2 - 1 \le 0 \tag{5}$$

The previous basic set of continuous BUGS is used to set the parameters. The degree-of-match function, the exchange operator, and the reproduction radius are set as follows.

(1) Definition of degree-of-match function

In order to perform region search, a situation must be created in which bugs exist in the region, but not out of the region. The simplest method of doing so is to define the degree-of-match function in a step form, so that the bacteria can be obtained in the region but not out of the region. In this paper, however, the degree-of-match function is defined to take a negative value outside the permissible region of the problem (executable region), based on the results of several trial simulations:

$$F(\vec{X}) = F(x_1, x_2) = \begin{cases} 1 & (\text{if } g(x_1, x_2) = x_1^2 + x_2^2 - 1 \le 0) \\ -2 & (\text{otherwise}) \end{cases} \tag{6}$$

By defining the degree-of-match function in this way, not only are the bacteria unavailable, but energy is lost outside the permissible region. This makes the bug less likely to survive, which is a desirable situation in the region search.

(2) Exchange operator

In order to see the effect of exchanging the parent and the child, two cases are examined, in which the operator is introduced or not introduced.

(3) Other items

In order to see the dependence of the behavior of BUGS on the reproduction radius *RR*, the cases of $RR = \infty$, 1.0, and 0.1 are examined. The other settings are the same as in the basic set of continuous BUGS. Figure 2 is the result of the conventional BUGS application in which the parent–child exchange operator is not introduced. Figures 3 to 5 are the results in which the parent–child exchange operator is introduced and *RR* is varied.

Comparing Figs. 2 and 3, it is seen that the parent–child exchange operator is useful in maintaining the balance of the solution distribution. Comparing Figs. 3 to 5, the distribution is more uniform when *RR* is larger. When $RR = \infty$, the effect of the reproduction radius need not be considered, which helps to improve the computation speed.

Thus, by considering both the uniformity of the distribution and the reduction of computational complexity, it seems adequate in the region search to set $RR = \infty$ and to introduce the parent–child operator.

2.2.4 Consideration in application to region search problem

By the numerical experiments up to this stage, it has been shown that BUGS has basic properties which are suited to region search by a set of bugs. Since the bacterial concentration corresponds to the value of the objective

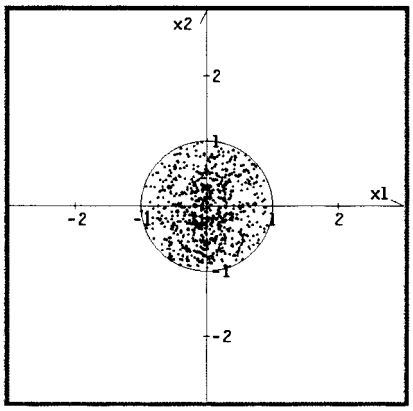

Fig. 3. BUGS with exchange operator ($RR = \infty$).

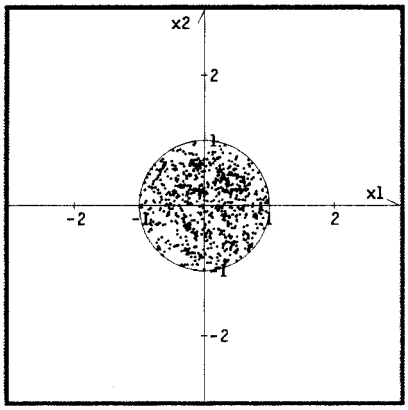

Fig. 4. BUGS with exchange operator ($RR = 1.0$).

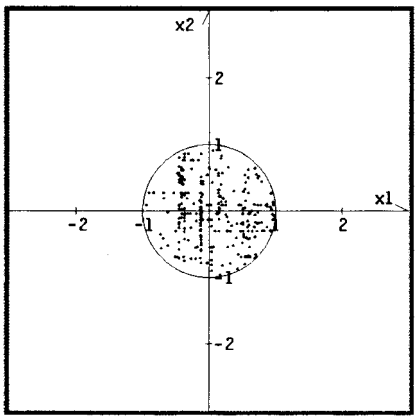

Fig. 2. BUGS without exchange operator ($RR = \infty$).

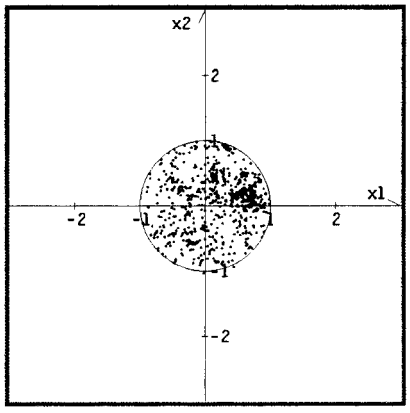

Fig. 5. BUGS with exchange operator ($RR = 0.1$).

function, it is judged that the above property is the effect of the bugs living in different regions.

It is also seen that, in order to achieve a better-balanced region search, the reproduction radius and the degree-of-match function must be adequately defined. This suggests that a subset of the Pareto optimal solution set for the multiobjective optimization problem can be derived with a good balance.

3. Application of BUGS to Multiobjective Optimization Problem and Discussion

It was suggested in the previous section that a subset of the Pareto optimal solution set for the multiobjective optimization problem can be derived with good balance by using BUGS. In this section it is shown that BUGS can be applied to the multiobjective optimization problem by adequately defining the degree-of-match function. Then, the method is applied to a typical two-objective continuous function optimization problem and a two-objective combinatorial optimization problem. Problems in the approach are pointed out.

3.1 Application of BUGS to 2-objective continuous function optimization problem

BUGS is next applied to the following multiobjective optimization problem:

$$\min \begin{cases} f_1(x_1, x_2) = & x_1^2 + x_2^2 \\ f_2(x_1, x_2) = & (x_1 - 1)^2 + (x_2 - 1)^2 \end{cases} \tag{7}$$

The Pareto optimal solution for this problem is the segment, which connects $(x_1, x_2) = (0, 0)$ and $(x_1, x_2) = (1, 1)$, the optimal solutions of the two objective functions f_1 and f_2, respectively.

3.1.1 Application procedure

The procedure for application of BUGS to the two-objective optimization problem is described below.

(1) Degree-of-match function

The degree-of-match function is defined as follows, with $f_i(x_1, x_2)$, $i = 1, 2$ as the evaluation functions:

$$F(\vec{X}) = F(x_1, x_2) = \begin{cases} 1 & \text{(if the bug is a Pareto optimal individual)} \\ -2 & \text{(otherwise)} \end{cases} \tag{8}$$

By a Pareto optimal individual is meant an individual which is not inferior to any other individual in the individual set. The above definition of the degree-of-match function is called the Pareto bug preservation strategy.

(2) Other items

The maximum number N_{MAX} of bugs is set as 200, and x_0 is set as 3. For case 1, the mutation is defined as $-x_0/100 \le dx_j^i(t) \le x_0/100$, $j = 1, \ldots, n$, as in the basic set of continuous BUGS. For case 2, the mutation is applied by setting $-4|dx| \le dx \le 4|dx|$. Other items are set as in the basic set of continuous BUGS.

3.1.2 Result of application

Figures 6 and 7 show the results of application of BUGS to the two-objective continuous function optimization problem in cases 1 and 2, respectively. It is seen in Fig. 6 that the bugs concentrate less near the optimal point of the respective objective functions than in the central area of the segment. This seems due to the fact that the probability that the Pareto optimality is lost by the movement of the bug is higher near the optimal point of the respective objective function than in the central area of the segment.

The Pareto optimal solutions form a segment, while bugs concentrate in an expanding area in the central area of the segment. This seems due to the fact that the bugs concentrate in the central area in which the Pareto optimality is less likely to be lost, since there is no penalty for concentration of bugs.

In order to resolve these two points and to distribute bugs with a good balance on the segment providing Pareto optimality, it is necessary to limit the movement of a bug once it reaches the Pareto optimal solution, so that the bug stays at the Pareto optimal solution as long as possible, and to provide a certain penalty for the concentration of bugs.

Figure 7 shows the result when movement is restricted. It is seen that although the distribution of the bugs is not made uniform, bugs stay on the segment providing the Pareto optimal solutions. This is because the number of

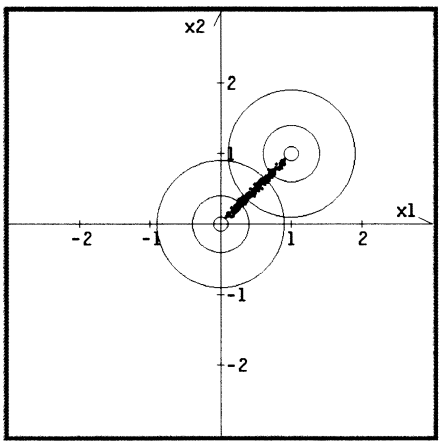

Fig. 6. Application result of BUGS to two-objective problem for case 1 (x-plane).

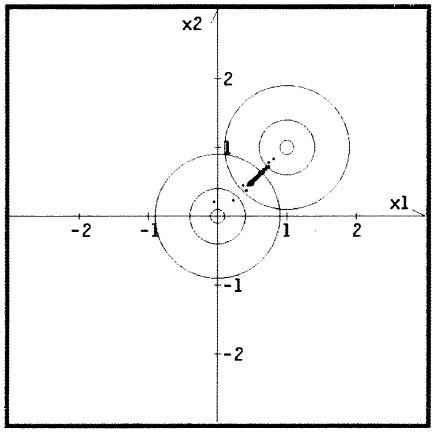

Fig. 7. Application result of BUGS to two-objective problem for case 2 (x-plane).

bugs soon reaches the maximum. In addition, the movement dx is decreased with the progress of the search, which restricts the movement of the bugs as a result. Thus, the bugs congregate in the central area where it is easier to survive, reducing movement and arriving at a situation in which the bugs concentrate in the central area.

Based on this reasoning, the concept of cannibalism is introduced into BUGS in the next section in order to restrict the movement of the bugs and to resolve the problem of concentration of bugs.

3.2 Two-objective combinatorial optimization problem

The knapsack problem is a problem of determining the combination of goods so that the total value is maximized, when goods with various values and weights are put into a knapsack with a constraint on the total weight. In the case of the two-objective knapsack problem, there are two kind of evaluations, with value 1 and value 2, for the same kind of goods.

The problem is formulated as follows. Suppose that there exist n goods. Let the weight of good j be a_j, and let value 1 be c_{1j} and value 2 be c_{2j}. Let the maximum weight that can be contained in the knapsack be b. Let x_j be the 0-1 variable, which takes the value 1 when good j is selected, and 0 when not. Then, the two-objective knapsack problem consists of determining the combination of goods so that the total value of the contained goods $\sum_{j=1}^{n} c_{ij}x_j (i = 1, 2)$ is maximized, under the constraint that the total weight of the contained goods $\sum_{j=1}^{n} a_j x_j$ does not exceed the maximum weight b of the knapsack.

The problem is formulated as follows:

$$\left.\begin{array}{ll} \max & f_i(x_1, ..., x_n) = \sum_{j=1}^{n} c_{ij}x_j,\ i = 1, 2 \\ \text{subj.to} & \sum_{j=1}^{n} a_j x_j \le b \\ & x_j = 0 \text{ or } 1,\ j = 1, ..., n \end{array}\right\} \quad (9)$$

This paper considers a 20-variable two-objective knapsack problem as expressed by Eq. (9) with the numerical values shown in Table 1. For this problem, there exist 10 Pareto optimal solutions as shown in Table 2.

3.2.1 Application procedure and basic set of combinatorial BUGS

The following application procedure forms the basis for the application of BUGS to the combinatorial problem in this paper. Consequently, it is called the basic set of combinatorial BUGS. Other settings are the same as in the basic set of continuous BUGS.

[Basic set of combinatorial BUGS]

(1) Degree-of-match function

The Pareto bug preservation strategy is used. The evaluation function is defined as

$$g_i(x_1, ..., x_n) = \begin{cases} \sum_{j=1}^{n} c_{ij}x_j, & i = 1, 2 \\ & (\sum_{j=1}^{n} a_j x_j \le b) \\ 0 & \text{(otherwise)} \end{cases} \quad (10)$$

(2) Coding

$\vec{X}$ is set the same as the solution x_j of Eq. (9). The outline of the move is defined as follows. Let the number of goods be n. G reverses stochastically the value of $x_i + 1$, according to the value of dx_i of the input $\vec{DX}$. When $i = n$, inversion is not applied. The above procedure is iterated twice, provided that different values are used for dx_i in the

Table 1. Twenty-variable two-objective knapsack problem

a	222,	832,	491,	949,	799,	230,	119,	713,	108,	435
	584,	386,	861,	454,	365,	969,	11,	153,	78,	86
c_1	354,	526,	927,	659,	68,	301,	252,	617,	268,	14
	292,	21,	263,	605,	5,	866,	452,	293,	564,	199
c_2	571,	69,	467,	419,	317,	199,	765,	977,	257,	641
	332,	542,	529,	161,	924,	386,	71,	957,	608,	846
b	4599									

Table 2. Pareto optimal solution set of knapsack problem

f_1	f_2	$\boldsymbol{x}$
1189	3354	10110111100001011111
1551	2517	10100111101001111111
1758	2484	10110111101001101111
1822	2307	10100111100101111111
1829	2208	10100111110001111111
2036	2175	10110111110001101111
2109	1865	10100011110101111111
2316	1832	10110011110101101111
2382	1720	10100111111101101111
3016	1684	10100111110110101111

first and second processes. (n+1) values of dx_i are needed in a single operation, and $\overrightarrow{DX}$ is a 2(n+1)-dimensional variable.

In this case, the number of goods is n = 20, and $\overrightarrow{DX}$ is a 42-dimensional variable. The element takes an integer value of 1, . . . , 9. Thus,

$$\overrightarrow{X}(t) = (x_0(t), ..., x_{19}(t)), x_i = 0 \text{ or } 1 \quad (11)$$

$$\overrightarrow{DX}(t) = (dx_0(t), ..., dx_{41}(t)), dx_i = 1, ..., 9 \quad (12)$$

(3) Movement

Let the number of goods be n. G in Eq. (1) realizes the mechanism given by the following algorithm.

Step 1: The bug is moved by BUGS-MOVE.

Step 2: The values of the objective functions before and after the movement are compared.

Step 3: If the value of the objective function after the movement is superior to the value before the movement, movement ends.

Step 4: If the value of the objective function after the movement is not inferior to the value before the movement, movement ends with probability P_p. If movement does not end, go to step 6.

Step 5: If the value of the objective function after the movement is inferior to the value before the movement, movement ends with probability P_i. If movement does not end, go to Step 6.

Step 6: The position of the bug is returned to the position before the movement. Movement ends.

"BUGS-MOVE"

Step 1: Let the moving bug be Bug_a. If x_k is reversed in the previous move, we set $N^a_{move} = k$. At the first move, k is determined at random in the range from 0 to n. Then, the number i is determined in the range from 0 to n. The determination is by roulette selection. More precisely, the probability p_i that the number i is selected is given by $e^{2dx_i}/\sum_{j=0}^{n} e^{2dx_j}$.

Step 2: If $i = n$ is selected, go to step 3. Otherwise, let $N^a_{move} = (N^a_{move} + i)\%n$, where $\%n$ implies that the remainder after dividing by n should be used. Then, the value of $x_{N^a_{move}}$ is reversed.

Step 3: A number from 0 to n is selected as in step 1. The selection is made with the probability p_i that the number i is selected is $e^{2dx_{i+21}}/\sum_{j=0}^{n} e^{2dx_{j+21}}$.

Step 4: As in step 2, if $i = n$ is selected, go to step 5. If otherwise, the value of N^a_{move} is updated and the value of $x_{N^a_{move}}$ is reversed, as in step 2.

Step 5: BUGS-MOVE ends.

(4) Other items

Mutation is performed for each chromosome. The value is changed by selecting an integer from 1 to 9 at random. The parameters are set as $P_p = 0.5$, $P_l = 0.2$, and $N_{MAX} = 200$.

(5) Composition of initial generation

0 or 1 is selected at random for x_i, $i = 1, \ldots, n$, and an integer from 1 to 9 is selected at random for dx_i, $i = 1, \ldots, n$.

3.2.2 Result of application

Figure 8 shows the number of Pareto optimal solutions obtained up to the 2000th generation. Among 10 Pareto optimal solutions, only 2 to 3 are obtained on average. This is because the bugs concentrate at a particular Pareto optimal solution, from which the search does not progress further.

Thus, in order to derive a larger number of Pareto optimal solutions, a provision must be made so that bugs,

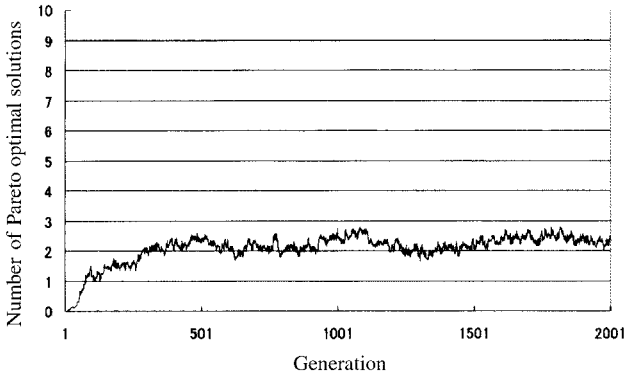

Fig. 8. Application result of BUGS for knapsack problem.

as the search points, do not concentrate at a particular solution, as in the case of the two-objective continuous function optimization problem considered in Section 3.1. For this purpose, in the next section we consider the introduction of the concept of cannibalism.

4. C-BUGS with cannibalism

Cannibalism is the phenomenon in which a bug obtains energy by eating another bug, instead of bacteria. By introducing this concept, the solution distribution is made more uniform in the continuous multiobjective optimization problem, and the concentration of bugs at a particular solution is eliminated. Thus, it is expected that a larger number of Pareto optimal solutions will be derived. In the following, the C-BUGS algorithm with cannibalism is described, and the behavior of this algorithm is examined by applying it to a simple region search problem. The method is applied in the next section to the multiobjective optimization problem.

4.1 C-BUGS algorithm

In order to introduce the concept of cannibalism into BUGS, BUGS-GA in the BUGS algorithm shown in the Appendix must be modified. More precisely, sexual reproduction is controlled by probability P_{msex}. When sexual reproduction does not occur, the bugs engage in cannibalism. When bugs come into a certain cannibalism radius (*CR*), they try to eat each other. A bug obtains a certain energy by eating another bug. The energy which is obtained is controlled by a constant α.

BUGS-GA after modification is called C-BUGS-GA in this paper. The algorithm is as follows.

[C-BUGS-GA]

Step 1: For two bugs Bug_a and Bug_b, sexual reproduction occurs with probability P_{msex}. If it occurs, go to step 2. If not, go to step 5.

Step 2: If the reproducibility condition is satisfied by Bug_a and Bug_b, sexual reproduction occurs. If they are capable of reproduction, go to step 3. If not, go to step 7. By the reproducibility condition is meant the situation in which each of the two bugs has an energy above a certain threshold (reproducibility energy, *RE*), and the distance between the two bugs is within a certain reproduction radius (reproducibility radius, *RR*).

Step 3: From two parent bugs Bug_a and Bug_b, child bugs Bug_c and Bug_d are generated.

Crossover with probability P_c and mutation with probability P_m are applied to the direction control vector $\vec{DX}$ of the parent bug, as in the GA, to obtain that of the child bug. The position vector $\vec{X}$ of the child bug is set near the parent bug. The energies E_a and E_b of the parent bugs are halved, $E_a/2$ and $E_b/2$, respectively. The energies of the child bugs are set as $(E_a + E_b)/2$, respectively.

Step 4: Go to step 7.

Step 5: For Bug_a and Bug_b, if they are at a distance within a certain *CR*, they try to eat each other. If they do not, go to step 7.

Step 6: Bug_b is deleted from the set. Bug_a obtains the energy αE_b, where α is a positive constant.

Step 7: If the above procedure has been performed for all pairs of bugs, go to step 8. If not, go back to step 1.

Step 8: If the energy of Bug_a exceeds a certain threshold value (producibility energy, *PE*), autoreproduction occurs with probability P_{asex}. If autoreproduction occurs, go to step 9. If not, go to step 10.

Step 9: Parent bug Bug_a disappears, being replaced by two child bugs. The direction control vector $\vec{DX}$ of the child bug is obtained by applying mutation to the direction control vector $\vec{DX}$ of the parent bug with probability P_m. The position vector $\vec{X}$ of the child bug is placed near the parent bug. The energy of the child bug is set as half the energy of the parent bug.

Step 10: If the above procedure has been performed for all bugs, end. If not, go to step 8.

4.2 Behavior of C-BUGS with cannibalism

In order to examine the behavior of C-BUGS with introduced cannibalism, the algorithm is applied to the region search problem considered in Section 2.2. It is a problem of determining the region satisfying the constraint of Eq. (5).

(1) Application procedure

The cannibalism parameters are newly set as $P_{msex} = 0.9$, $\alpha = 0.25$, and *CR* is set in two ways, as 0.1 and 0.05. The same degree-of-match function as in the region search problem of Section 2.2.2 is used. The other items are set the same as in the basic set of continuous BUGS, provided that the range of $-4|dx| \le dx \le 4|dx|$ is used in the mutation.

(2) Result of application

Figures 9 and 10 show the results of application. Comparing these to Fig. 3, it is seen that the density of bugs near the region boundary was low before introducing cannibalism, while the bugs are distributed with constant density in almost all areas in the region after introducing the cannibalism. It is also seen that the distance between bugs differs between Figs. 9 and 10. In other words, the density of bugs, that is, the density of the solution distribution, can be controlled by the cannibalism radius *CR*.

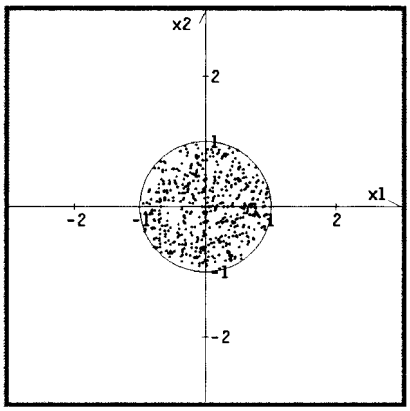

Fig. 9. Application result of C-BUGS to region search problem ($CR = 0.05$).

5. Application of C-BUGS to Multiobjective Optimization Problem

This section presents the results when C-BUGS with the proposed concept of cannibalism is applied to typical continuous and discrete multiobjective optimization problems. The results are shown and compared to the results given by RSGA (rank sharing genetic algorithm) [5]. RSGA is a method of multiobjective GA which is obtained by introducing the sharing concept into multiobjective GA.

5.1 Two-objective continuous function optimization problem

In the following, the result obtained when C-BUGS is applied to the two-objective continuous function optimization problem is presented and is compared to the result when RSGA is applied. The Pareto optimal solutions for this problem form the segment connecting $(x_1, x_2) = (0, 0)$ and $(x_1, x_2) = (1, 1)$, which are the optimal solutions of objective functions f_1 and f_2, respectively. The parameters of RSGA are set as in Table 3. Gray code is used for the coding. The selection is the tournament selection. The crossover is two-point crossover. The mutation is applied to each gene.

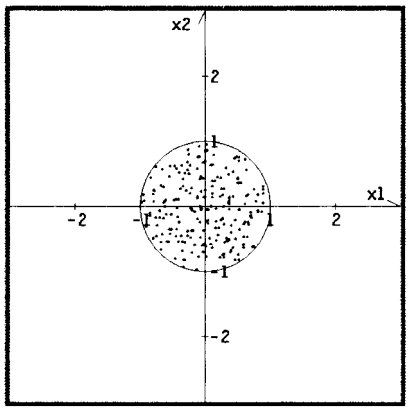

Fig. 10. Application result of C-BUGS to region search problem ($CR = 0.10$).

5.1.1 Application procedure of BUGS

(1) Degree-of-match function

The Pareto bug preservation strategy is used.

(2) Parameters

The cannibalism radius is set as $CR = 0.05$. We set $P_{msex} = 0.9$ and $\alpha = 0.25$.

(3) Mutation

The variables are set at random in the range of $-x_0/100 \le dx_j^i(t) \le x_0/100$, $j = 1, \ldots, n$ for case 1, and $-4|dx| \le dx \le 4|dx|$ for case 2. The other settings are the same as in the basic set of continuous BUGS.

5.1.2 Result of application

Figures 11 to 13 show the results of C-BUGS application. Figures 14 and 15 show the result of RSGA application. It is seen that the density of the bug distribution can be controlled by introducing cannibalism, resulting in a better-balanced solution set than that given by RSGA.

Comparing the results given by the two mutation procedures in cases 1 and 2, it is seen that the bugs as the search points are distributed with better balance and with a certain distance on the segment representing the Pareto optimal solutions in case 2, where the variables are selected in the range $-4|dx| \le dx \le 4|dx|$. This is because the move of the bug, that is, the value of dx, is reduced with the progress of the search in the conventional method, finally resulting in a stationary state. But by introducing the concept of cannibalism, the distribution of bugs settles at a final sta-

Table 3. Parameters of RSGA

Parameter	Continuous-type problem	Discrete-type problem
Size of set of individuals	200	200
Code string length	24	20
Crossover ratio	0.4	0.6
Mutation ratio	0.1	0.1
End generation no.	100	100

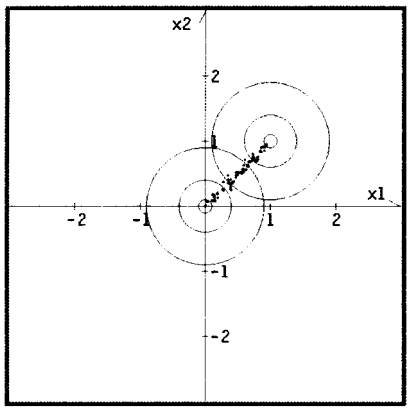

Fig. 11. Application result of C-BUGS to two-objective problem for case 1 (x-plane).

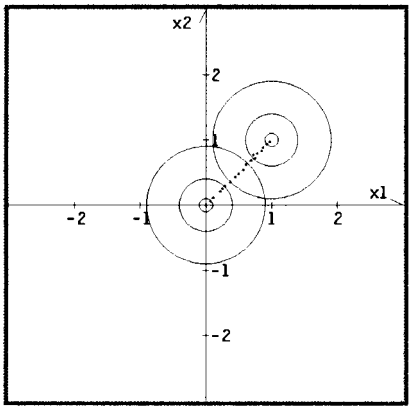

Fig. 12. Application result of C-BUGS to two-objective problem for case 2 (x-plane).

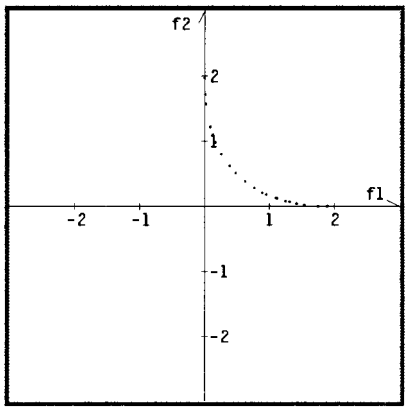

Fig. 13. Application result of C-BUGS to two-objective problem for case 2 (f-plane).

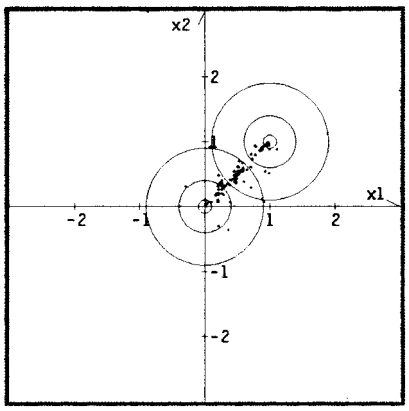

Fig. 14. Application result of RSGA to two-objective problem (x-plane).

tionary state with a spacing such that the bugs do not eat each other.

By the above mechanism, the bugs as search points are distributed with a good balance on the segment representing the Pareto optimal solutions, while maintaining the distance determined by the cannibalism radius. It is expected that a subset of the Pareto optimal solution set with a good balance will be obtained.

5.2 Three-objective continuous function optimization problem

In this section, the proposed method is applied to the following problem. The Pareto optimal solutions for this problem are the triangular region connecting

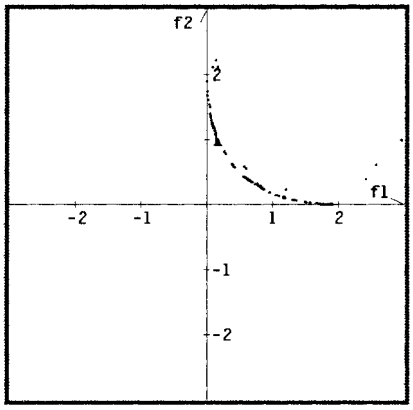

Fig. 15. Application result of RSGA to two-objective problem (f-plane).

$(x_1, x_2) = (0, 0)$, $(x_1, x_2) = (1, 1)$, and $(x_1, x_2) = (1, 0)$, which are the optimal solutions of f_1, f_2, and f_3, respectively. The parameters for RSGA are set the same as in the previous section:

$$\min \begin{cases} f_1(x_1, x_2) = & x_1^2 + x_2^2 \\ f_2(x_1, x_2) = & (x_1 - 1)^2 + (x_2 - 1)^2 \\ f_3(x_1, x_2) = & (x_1 - 1)^2 + x_2^2 \end{cases} \quad (13)$$

5.2.1 Application procedure of BUGS

(1) Degree-of-match function
The Pareto bug preservation strategy is used.
(2) Parameters
We set

$$CR = 0.05, P_{msex} = 0.9, \alpha = 0.25$$

(3) Mutation
The variable is set at random in the range of $-4|dx| \le dx \le 4|dx|$.

The other settings are the same as in the basic set of continuous BUGS.

5.2.2 Results of application

Figure 16 shows the results given by the proposed C-BUGS, and Fig. 17 the results given by RSGA. It is seen that the proposed C-BUGS determines a better-balanced subset of the Pareto optimal solution set than RSGA.

5.3 Two-objective combinatorial optimization problem

C-BUGS and RSGA are next applied to the 20-variable two-objective knapsack problem considered in Section 3.2.

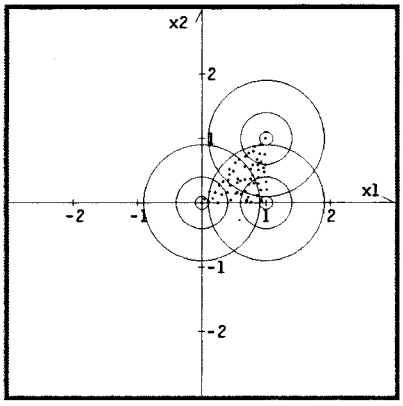

Fig. 16. Application result of C-BUGS to three-objective problem (x-plane).

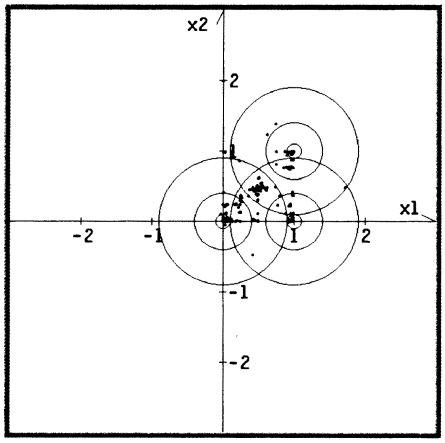

Fig. 17. Application result of RSGA to three-objective problem (x-plane).

5.3.1 Application procedure

The parameters in the knapsack problem are shown in Table 2. The parameters of RSGA are shown in Table 3. The selection type is tournament selection. The crossover is uniform crossover. Mutation is applied to each gene. The Pareto preservation strategy is used.

(1) Degree-of-match function
The Pareto bug preservation strategy is used.
(2) Parameters
We set $CR = 0$, $P_{msex} = 0.9$, and $\alpha = 0.25$, provided that the Hamming distance is used. The other items are the same as in the basic set of combinatorial BUGS.

5.3.2 Result of application

Figure 18 shows the numbers of Pareto optimal solutions obtained by RSGA and C-BUGS. The value shown in the figure is the average of 20 trials. Figure 19 shows the distribution of the number of Pareto optimal solutions obtained by 20 trials at the 500th, 1000th, and 2000th generations.

The following observations are made from the results of simulation. In RSGA, a larger number of Pareto optimal solutions can be derived in the early generation, but the probability is low that all Pareto optimal solutions are derived. In C-BUGS, the speed of search is lower than in RSGA, but the probability is high that all Pareto optimal solutions are derived.

5.4 Discussion of simulation results

A simulation was performed using C-BUGS with cannibalism as proposed in this paper, and the result was

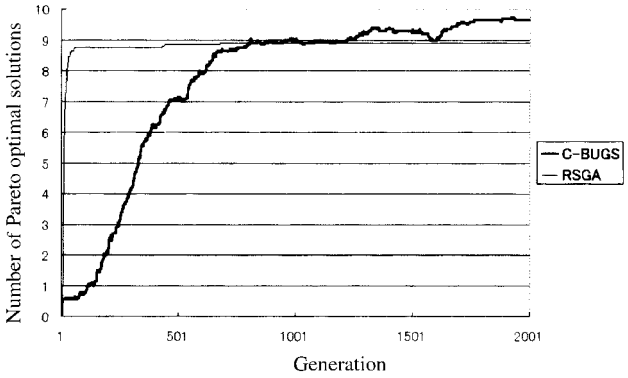

Fig. 18. Application results of C-BUGS and RSGA.

Table 4. Comparison of computation times (s)

Problem	C-BUGS	RSGA	Computer
2-objective continuous	121 (2000 generations)	41 (100 generations)	RS/6000
3-objective continuous	227 (2000 generations)	67 (100 generations)	RS/6000
2-objective discrete (knapsack)	1.7 (500 generations)	6 (500 generations)	Windows machine Pentium II 350 MHz
	5.8 (1000 generations)	11 (1000 generations)	
	16 (2000 generations)	23 (2000 generations)	

compared to RSGA. Table 4 compares the computation time and the number of generations until the process is completed for the two methods and for each problem.

It is seen from the simulation results that the introduction of cannibalism has the same effect as sharing in the genetic algorithm, and is very effective in making the solution distribution more uniform. It is also seen that the density of the solution distribution can be controlled by adjusting the cannibalism radius.

C-BUGS, which is the proposed method, realizes a more uniform distribution of solutions in the continuous multiobjective optimization problem than RSGA. In the discrete multiobjective optimization problem (20-variable knapsack problem), the number of Pareto optimal solutions derived is approximately 10% greater than with RSGA.

For the two-objective continuous optimization problem, the computation time of BUGS is 127 s (for 2000 generations).

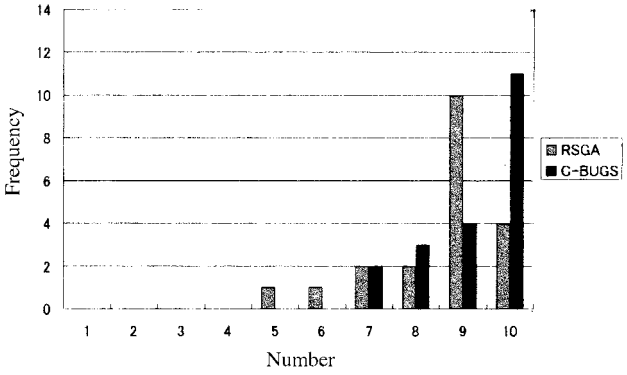

Fig. 20. The number of obtained Pareto optimal solutions (1000th generation).

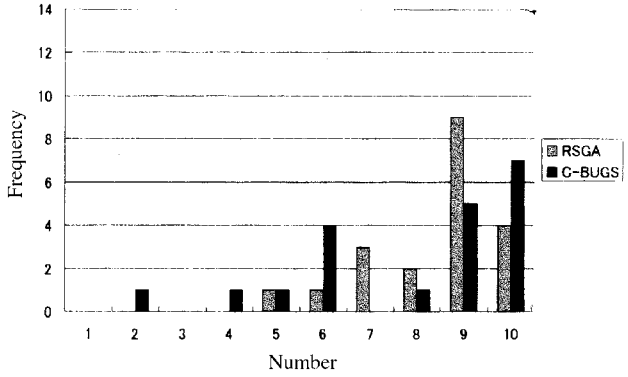

Fig. 19. The number of obtained Pareto optimal solutions (500th generation).

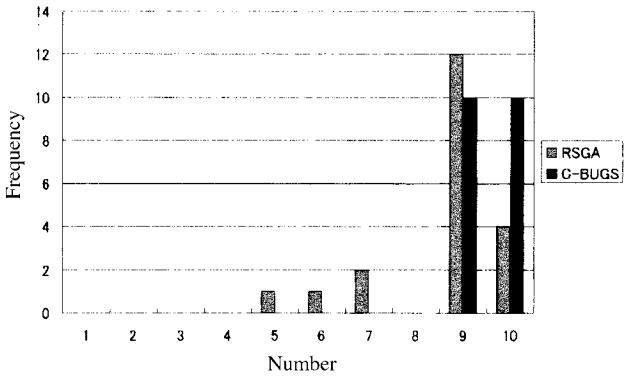

Fig. 21. The number of obtained Pareto optimal solutions (2000th generation).

6. Conclusions

This paper has examined the behavior of the BUGS algorithm proposed by Iniwa and Sato by numerical experiment. An algorithm has been proposed in which the Pareto optimal solutions for the multiobjective optimization problem are derived with good balance by introducing the concept of cannibalism.

As a result of applying the method to typical continuous and discrete multiobjective optimization problems, the following properties were revealed.

(1) BUGS contains the concept of separated lift in its algorithm itself, and thus is suited to the region search problem and the multiobjective optimization problem. In order to realize a more uniform distribution of the derived solutions, however, improvement by introducing a new concept is needed.

(2) The introduction of the cannibalism concept has an effect which corresponds to the sharing in the genetic algorithm, and has a marked effect of making the distribution of the derived solutions more uniform. By adjusting the cannibalism radius, the density of the solution distribution can be controlled.

(3) The proposed C-BUGS achieves a more uniform distribution of the derived solutions than RSGA in the continuous-type multiobjective optimization problem. In the discrete multiobjective optimization problem (20-variable knapsack problem), the number of Pareto optimal solutions derived is approximately 10% larger than with RSGA.

The following problems are noted for the future.

(1) In application to the continuous multiobjective optimization problem, there must be a quantitative measure of the balance (diversity) of the derived Pareto optimal solutions. Using such a measure, each procedure must be evaluated quantitatively.

(2) In application to the discrete multiobjective optimization problem, the application to a larger-scale problem should be attempted and the computation efficiency should be improved.

Iniwa and Sato investigated the use of BUGS in the higher-dimensional optimization problem, and reported that the performance of BUGS is equal to or better than that of GA. It is consequently expected, in the application to the multiobjective optimization problem, that BUGS will become better than the GA when applied to higher-dimensional problems. This paper has discussed problems of order up to the two-dimensional three-objective problem as basic investigations, but it is left as an important issue for the future to apply the proposed method to higher-dimensional problems, and to examine the performance by using a statistical approach or other means of analysis.

REFERENCES

1. Ichikawa (editor). Theory and methods of multiobjective decision. Society of Instrument and Control Engineers; 1980.
2. Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Addison–Wesley; 1989.
3. Fonseca CM, Fleming PJ. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. Proc Fifth Int Conference on Genetic Algorithms. Morgan Kaufmann; 1993. p 416–423.
4. Tamaoki, Mori, Araki. Generation of Pareto optimal solution set by genetic algorithm. Trans Soc Instrum Control Eng 1995; 31:1185–1192.
5. Kondo, Watanabe, Yasuda, Yokoyama. Composition of genetic algorithm designed to maintain the diversity of Pareto optimal solutions. Proc 35th Conv Soc Instrum Control Eng 309 A-2, p 883–884, 1996.
6. Iniwa, Sato. A bug type search system involving an extension of the genetic algorithm. J Soc Artif Intell 1993;8:797–809.
7. Iniwa. Fundamentals of genetic algorithms: Deciphering GAs. Ohm Press; 1994.

APPENDIX

[The BUGS algorithm]

Step 1. The set of bugs of the initial generation (set size N_0) is created at random and a constant energy is assigned. The generation number is initialized.

Step 2. The bug is moved according to Eq. (1).

Step 3. Based on the information given by $\vec{X}_i$, the degree of match of each bug $F(\vec{X}_i)$ is calculated and the energy of each bug is determined as follows:

$$e_i(t) := e_i(t-1) + F(\vec{X}_i) - C_{age}$$

where C_{age} is the decay factor of the energy with growth.

Step 4. If the energy of a bug reaches 0, the bug dies. Dead bugs are removed from the list of bugs. If, as a result of the death of a bug, the number of individuals in the set of bugs becomes less than X_f, a new bug is added to the set. The parameters of the newly added bug are determined by the same procedure as in the creation of the initial generation of bugs.

Step 5. BUGS-GA is called.

Step 6. The generation number is incremented by 1. Go to step 2.

[Genetic algorithm BUGS-GA]

Step 1. For two bugs Bug_a and Bug_b, if the proliferation condition is satisfied, sexual reproduction occurs. If

Table A.1. Parameters of BUGS

P_c	0.6	PE	2.0
P_m	0.03	RR	∞
P_{asex}	0.1	N_{MAX}	800
C_{age}	0.2	N_0	100
RE	2.0	N_f	10
End generation no.	2000		

it is possible, go to step 2. If not, go to step 3. The proliferation condition is that the energies of the two bugs both exceed a certain threshold (the reproducibility energy, *RE*) and the distance between the two bugs is within a certain reproduction radius *RR*.

Step 2. Child bugs Bug_c and Bug_d are generated from the two parent bugs Bug_a and Bug_b. Crossover with probability P_c and mutation with probability P_m are applied to the direction control vector $\overrightarrow{DX}$ of the parent bug to obtain that of the child bug, as in the GA. The position vector $\vec{X}$ of the child bug is set near that of the parent bug. The energies E_a and E_b of the parent bugs are halved, $E_a/2$ and $E_b/2$, respectively. The energies of the child bugs are set as $(E_a + E_b)/2$ respectively.

Step 3. If the above procedure has been performed for all pairs of bugs, go to step 4. If not, go to step 1.

Step 4. If the energy of bug Bug_a exceeds some threshold value (the producibility energy *PE*), autoreproduction occurs with probability P_{asex}. If autoreproduction occurs, go to step 5. Otherwise go to step 6.

Step 5. The parent bug Bug_a disappears and is replaced by two child bugs. The direction control vector $\overrightarrow{DX}$ of the child bug is obtained by applying mutation to the direction vector of the parent bug with probability P_m. The position vector $\vec{X}$ of the child bug is placed near the parent bug. The energy of the child bug is set as half that of the parent bug.

Step 6. If the above procedure has been performed for all bugs, end. Otherwise go to step 4.

AUTHORS (from left to right)



Keiichiro Yasuda (member) completed the doctoral program in electrical engineering at Hokkaido University in 1989. He became a research associate in 1989 and an associate professor in 1991 in the Faculty of Engineering, Tokyo Metropolitan University. He is now an associate professor in the Graduate School of Engineering. His research interests include system optimization and electrical power system engineering. He holds a D.Eng. degree, and is a member of the Society of Instrumentation and Control Engineers and the Electrical Installation Society.

Osamu Yamazaki (nonmember) graduated from the Department of Electrical Engineering of Tokyo Metropolitan University in 1999 and then joined Hitachi Systems Technology Corporation. His graduate research dealt with genetic algorithms.

Takao Watanabe (member) completed the M.E. program in electrical engineering at Tokyo Metropolitan University in 1994. He then became a research associate on the Faculty of Engineering, and is now a research associate in the Graduate School of Engineering. His research interests include robust control. He holds a D.Eng. degree, and is a member of IEE and the Society of Instrumentation and Control Engineers.