Two-Level 0-1 Programming Using Genetic Algorithms and a Sharing Scheme Based on Cluster Analysis

Keiichi Niwa * Ichiro Nishizaki† Masatoshi Sakawa‡

Abstract— **This paper deals with the two level 0-1 programming problems in which there are two decision makers (DMs); the decision maker at the upper level and the decision maker at the lower level. The authors modify the problematic aspects of a computation method for the Stackelberg solution which they previously presented, and thus propose an improved computation method. Specifically, a genetic algorithm (GA) is proposed with the objective of boosting the accuracy of solutions while maintaining the diversity of the population, which adopts a clustering method instead of calculating distances during sharing. Also, in order to eliminate unnecessary computation, an additional algorithm is included for avoiding obtaining the rational reaction of the lower level DM in response to upper level DM's decisions when necessary. In order to verify the effectiveness of the proposed method, it is intended to make a comparison with existing methods by performing numerical experiments into both the accuracy of solutions and the computation time.**

Keywords: Two-level 0-1 programming problem, Stackelberg solution, Genetic algorithm, Clustering algorithm

1 Introduction

Under actual decision making situations, there are multiple decision makers (DMs) in hierarchically structured organizations, and decisions may be taken serially or simultaneously in order to optimize each of the objectives. This kind of problem has been formulated as a two-level programming problem [11]. In two-level programming problems, upper level DMs take their decisions first, and then, with a knowledge of the decisions of the upper level DMs, the lower level DMs make their decisions in order to optimize their own objective functions. This combination of decisions is known as a Stackelberg solution. In this paper, both the upper level and the lower level have one DM, and the problem is treated as a two-level 0-1 programming problem in which both DMs treat all of their decision variables as 0-1 parameters.

As an overview of research dealing with two-level programming problems that include discrete variables, Bard *et al.* presented an algorithm based on the branch-and-bound approach in order to derive the Stackelberg solution for two-level 0-1 programming problems [4] and two-level mixed integer programming problems [3]. Wen *et al.* [12] have presented a computation method for obtaining the Stackelberg solution to two-level programming problems which have 0-1 parameters for the decision variables in the upper level and continuous parameters for the decision variables in the lower level.

Meanwhile, the adaptive process of systems in the natural world has been explained, and genetic algorithms (GAs) which imitate the evolution occurring in living organisms have been receiving attention at international conferences related to GAs, publications by Goldberg [6], as have methodologies for optimization, adaptation and learning. GAs have also been adopted for a variety of combinatorial optimization problems, and their effectiveness has been reported [10].

Regarding research utilizing GAs for two-level programming problems, Anandalingam *et al.* [1] have proposed a method for obtaining the Stackelberg solution for two-level linear programming problems. We have previously proposed a computation method for deriving the Stackelberg solution to 0-1 programming problems for two-level decentralized systems [9], which adopts the double string proposed by Sakawa *et al.* for individual representation. We have also proposed a computation method using sharing [5] in order to boost the computational precision of the Stackelberg solution [8]. However, while it was possible to increase the precision of the derived Stackelberg solution using this method, it prompted an increase in computation time as a result.

Having established such a background, in this paper we focus on two-level 0-1 programming problems, and by

*Faculty of Economics, Hiroshima University of Economics, 5-37-1 Gion, Asaminami-ku, Hiroshima 731-0192, Japan Tel/Fax: +81-82-871-1048/1005; Email: ki-niwa@hue.ac.jp

†Graduate School of Engineering, Hiroshima University 1-4-1, Kagamiyama, Higashi-Hiroshima, 739-8527, Japan, Tel/Fax: +81-82-424-7604/422-7195 Email: nisizaki@hiroshima-u.ac.jp

‡Graduate School of Engineering, Hiroshima University 1-4-1, Kagamiyama, Higashi-Hiroshima, 739-8527, Japan, Tel/Fax: +81-82-424-7694/422-7195 Email: sakawa@msl.sys.hiroshima-u.ac.jp

reforming the problematic aspects of the computation method for the Stackelberg solution we previously presented, propose an improved version of the computation method. Specifically, we propose a GA which introduces a clustering method to replace the computation of distances between individuals occurring during sharing, in order to reduce computation time while maintaining the accuracy of the approximate Stackelberg solution. Also, for duplications within populations of the GA which handles decision making in the upper level, the method does not generate $\boldsymbol{x}$ and for each value iterates the computation of the rational reaction of the lower level DM, $\boldsymbol{y}(\boldsymbol{x})$, using a GA. Rather, an algorithm is proposed which reduces the number of applications of the GA used to compute the rational reaction of the lower level, by placing the combinations of past decisions of the DM for the upper level, $\boldsymbol{x}$, and their corresponding rational reactions $\boldsymbol{y}(\boldsymbol{x})$ in constant memory, and extracting the stored $\boldsymbol{y}(\boldsymbol{x})$ whenever a $\boldsymbol{x}$ which is identical to a value occurring in a previous generation of the GA appears. In order to verify the effectiveness of the proposed method, it is intended to perform numerical experiments investigating and comparing the method with existing methods from the perspectives of both solution accuracy and computation time.

2 The two-level 0-1 programming problem

For the sake of brevity, we denote the upper and lower level DMs by DM1 and DM2, respectively. The objective functions of DM1 and DM2 are written $z_1(\boldsymbol{x}, \boldsymbol{y})$, and $z_2(\boldsymbol{x}, \boldsymbol{y})$. The vectors of decision variables for DM1 and DM2 are $\boldsymbol{x} = (x_1, \ldots, x_{n_1})^T$, and $\boldsymbol{y} = (y_1, \ldots, y_{n_2})^T$. The superscript T indicates transposition. The coefficient vectors of the objective functions are denoted by $\boldsymbol{c}_1 = (c_{11}, \ldots, c_{1n_1})$, $\boldsymbol{d}_1 = (d_{11}, \ldots, d_{1n_2})$, $\boldsymbol{c}_2 = (c_{21}, \ldots, c_{2n_1})$, and $\boldsymbol{d}_2 = (d_{21}, \ldots, d_{2n_2})$. The coefficient matrices in the constraint equation are the $m \times n_1$ matrix A, and the $m \times n_2$ matrix B. The vector of constants on the right hand side of the constraint equation is written as $\boldsymbol{b} = (b_1, \ldots, b_m)^T$. The two-level 0-1 programming problem may now be formulated in general with an equation such as the following.

$$\left.\begin{array}{ll} \underset{\boldsymbol{x}}{\text{maximize}} & z_1(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{c}_1\boldsymbol{x} + \boldsymbol{d}_1\boldsymbol{y} \\ \text{where } \boldsymbol{y} \text{ solves} & \\ \underset{\boldsymbol{y}}{\text{maximize}} & z_2(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{c}_2\boldsymbol{x} + \boldsymbol{d}_2\boldsymbol{y} \\ \text{subject to} & A\boldsymbol{x} + B\boldsymbol{y} \leqq \boldsymbol{b} \\ & \boldsymbol{x} \in \{0,1\}^{n_1}, \boldsymbol{y} \in \{0,1\}^{n_2} \end{array}\right\} \quad (1)$$

For the sake of simplicity, in this paper, it is assumed that each component of A, B, $\boldsymbol{b}$, $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, $\boldsymbol{d}_1$, and $\boldsymbol{d}_2$ is positive.

It is possible to express the process for choosing the Stackelberg solution for a two-level 0-1 programming problem in the following manner. Each decision maker completely knows objective functions and constraints of the opponent and self, and DM1 first makes a decision and then DM2 makes a decision so as to minimize the objective function with full knowledge of the decision of DM1. That is to say, when the decision by DM1 is denoted $\hat{\boldsymbol{x}}$, DM2 solves the 0-1 programming problem (2) with parameters $\hat{\boldsymbol{x}}$, choosing the optimal solution $\boldsymbol{y}(\hat{\boldsymbol{x}})$ as the rational reaction to $\hat{\boldsymbol{x}}$.

$$\left.\begin{array}{ll} \underset{\boldsymbol{y}}{\text{maximize}} & z_2(\hat{\boldsymbol{x}}, \boldsymbol{y}) = \boldsymbol{d}_2\boldsymbol{y} + \boldsymbol{c}_2\hat{\boldsymbol{x}} \\ \text{subject to} & B\boldsymbol{y} \leqq \boldsymbol{b} - A\hat{\boldsymbol{x}} \\ & \boldsymbol{y} \in \{0,1\}^{n_2} \end{array}\right\} \quad (2)$$

Under this premise, DM1 also determines $\boldsymbol{x}$ by choosing the value which minimizes its own objective function. For problems which adopt the Stackelberg solution to conceptualize their solution, it is assumed that there is no consensus among DMs that might mutually constrain decisions. Putting it another way, their relationship may be described as non-cooperative.

3 GA based computational method

In this section, we present the derivation of the Stackelberg solution to the two-level 0-1 programming problem. A computation method based on a genetic algorithm is explained.

3.1 Coding and decoding

When solving 0-1 programming problems using GAs, binary strings are usually adopted to express individuals [7, 6]. However, under this representation it is possible that infeasible solutions that do not satisfy the constraints may be generated, so there is a danger that the performance of the GA may degrade. Thus, in this paper, in order to derive only feasible solutions, a double string [10] is used which is composed of the substring corresponding to the decision of DM1, $\boldsymbol{x}$, and the substring corresponding to the decision of DM2, $\boldsymbol{y}$, as shown in Fig.1. The decisions of DM1 and DM2 are handled by applying genetic operators on each sub-individual. In this paper, the GA handling the decision of DM1 is called the upper level GA, and the GA handling the decision of DM2 is called the lower level GA.

← Individual for $\boldsymbol{x}$ →			← Individual for $\boldsymbol{y}$ →		
$i_x(1)$	$\cdots$	$i_x(n_1)$	$i_y(1)$	$\cdots$	$i_y(n_2)$
$S_{i_x(1)}$	$\cdots$	$S_{i_x(n_1)}$	$S_{i_y(1)}$	$\cdots$	$S_{i_y(n_2)}$

Figure 1: Double string

$s_{i_x(m)} \in \{0,1\}, i_x(m) \in \{1, \ldots, n_1\}$, and for $m \neq m'$ it is assumed that $i_x(m) \neq i_x(m')$. Similarly, $s_{i_y(m)} \in \{0,1\}, i_y(m) \in \{1, \ldots, n_2\}$, and for $m \neq m'$ it is assumed that $i_y(m) \neq i_y(m')$. Also, in the double string, $i_x(m)$,

$i_y(m)$ and $s_{i_x(m)}$, $s_{i_y(m)}$ express the indexes of the elements of each solution vector respectively, and their values.

In this paper a decoding algorithm proposed by the authors [9] is also applied to the upper level and lower level GAs, generating only feasible solutions.

3.2 Reproduction

We first describe the reproduction operator of the lower level GA. In the lower level GA the value of $\boldsymbol{y}$ obtained by decoding individuals in the lower level GA, and the given values of the decision variables in the upper level GA, $\boldsymbol{x}$, are substituted into the objective function of DM2, $z_2(\boldsymbol{x}, \boldsymbol{y})$, and the value of the evaluation function for each individual is thus obtained. Next, the fitness value for each individual is derived using linear scaling, and the individuals reproducing in the next generation are determined by applying elitist expected value selection.

We now describe the reproduction operator of the upper level GA. In the upper level GA, the value of $\boldsymbol{x}$ obtained by decoding individuals in the upper level GA and the value of the rational reaction obtained by applying the lower level GA, $\boldsymbol{y}(\boldsymbol{x})$, are substituted into the objective function of DM1, $z_1(\boldsymbol{x}, \boldsymbol{y}(\boldsymbol{x}))$, and the value of the evaluation function for each individual is thus obtained. Next, the fitness value for each individual is calculated by applying linear scaling and adopting a clustering method such as that described in the next section. The individuals reproducing in the next generation are determined by applying elitist expected value selection based on these fitness values.

3.3 Applying the cluster analysis method

In the existing computation method proposed by the authors [8], sharing was introduced in order to calculate the fitness value in accordance with the degree of convergence among individuals within populations. However, because introducing sharing made it necessary to compute the distances between all the individuals within a population for each generation, this encouraged an increase in computation time.

On the other hand, when Yin *et al.* applied sharing to multimodal function optimization problems, instead of computing the distances between strings, a GA utilizing a clustering method [2] was proposed. Numerical experiments revealed that it was possible to reduce computation time while maintaining roughly the same search performance with a GA using unmodified sharing. In this paper we therefore incorporate the method of Yin *et al.* into the reproduction operator for the upper level GA, thus reducing the computation time of the proposed method.

The method proposed by Yin *et al.* adopts the centroid method known as the Adaptive MacQueen's KMEAN Algorithm for clustering. The procedure for the centroid algorithm is stated below.

Procedure for the Centroid method

Step 1 Process of representation: Cluster C_c is given by its centroid $G(C_c)$.

$$G(C_c) = (\bar{x}_{c1}, \bar{x}_{c2}, \ldots, \bar{x}_{cj}, \ldots, \bar{x}_{cp}), c = 1, \ldots, k$$
$$\text{with} \quad \bar{x}_{cj} = \frac{1}{n_c} \sum_{i=1}^{n_c} x_{ij}$$

p expresses the number of variables in the data units, and k is the number of clusters. n_c expresses the number of data units in the c^{th} cluster, and $\bar{x}_{cj}$ is the mean value of the j^{th} variable in the c^{th} cluster. x_{ij} expresses the value of the j^{th} variable in the i^{th} data unit, and $\boldsymbol{x}_i$ is the vector of the i^{th} data unit.

Step 2 Process of assignment: Each data unit is assigned to the cluster with the nearest centroid.

$$\boldsymbol{x}_i \in C_c, \quad \text{if} \quad d(\boldsymbol{x}_i, G(C_c)) = \min_{l=1,\ldots,k} d(\boldsymbol{x}_i, G(C_l))$$

Under the centroid method, these two steps are repeated until the data units converge on a stable state.

Next, the process for the Adaptive MacQueen's KMEAN Algorithm is described.

Procedure for the Adaptive MacQueen's KMEAN Algorithm

Step 1 The initial value of k is chosen, and the first k data units are allocated as members of each of the k initial clusters. The value of each data unit is taken as the value for each centroid. Next, the distance between each of the k cluster's centroids is calculated. If this distance is less than d_{min}, then a new cluster is generated by combining them. In such a case, the new cluster's centroid is computed, and the distance between the new cluster and the remaining clusters is recomputed. These operations are repeated until the distance between the centroids of the nearest clusters is at least as large as d_{min}.

Step 2 Each of the remaining $N - k$ data units is allocated to the cluster with the nearest centroid. In each case, after allocating the data unit, the centroid of the cluster to which the data unit was added is revised, and the distance to the other clusters is recomputed. If the distance between two clusters' centroids falls below d_{min}, the two clusters are combined, generating a new cluster. In such a case, the new cluster's centroid is calculated, and the distance between the new cluster and each of the remaining

clusters is recomputed. These operations are repeated until the distance between the centroids of the nearest clusters is at least as large as d_{min}. If the distance to the cluster with the nearest centroid is larger than d_{max}, then a new cluster is generated with the data unit as a member.

Step 3 After allocating data units $k+1$ to N, the centroid of each of the existing clusters is taken as a fixed seed point, and each data unit is reallocated to the nearest seed point.

d_{min} and d_{max} are complementary parameters expressing the minimum and maximum distances used when the Adaptive MacQueen's KMEAN Algorithm is applied.

This paper adopts the method proposed by Yin *et al.* for distributing k data units among the initial clusters. Specifically, when selecting the k data units, they are chosen in order of decreasing fitness value.

By using such a cluster analysis method, each individual's degree of convergence is measured. The following calculation is subsequently performed in order to update the fitness values. Here, the fitness value of individual i is written f_i, and the updated fitness value is written f_i'. The population size is N. Also, α is a constant, and d_{ik} is the distance between individual i and the centroid of the cluster within which it is contained, $G(C_c)$.

$$f_i' := \frac{f_i}{m_i}, i = 1, \ldots, N$$
$$\text{with} \quad m_i = n_c - n_c * \left(\frac{d_{ic}}{2d_{max}}\right)^{\alpha}, \boldsymbol{x}_i \in C_c$$

3.4 Crossover and mutation

For double strings, if standard one-point or multi-point crossovers are performed then there is a possibility that infeasible solutions may be generated because the indexes occurring in the offspring, $i_x(m)$, $i_x(m^{'})$, $m \neq m^{'}$ or $i_y(m)$, $i_y(m^{'})$, $m \neq m^{'}$, may have the same number. This kind of difficulty has been identified as occurring when genetic algorithms are applied to problems such as that of the traveling salesman, or the scheduling problem. Partially matched crossovers (PMX) have been devised to overcome this difficulty. In this paper, a modified version of PMX is used in order to handle the double strings proposed by Sakawa *et al.*[10]. Also, when determining whether or not to apply the cross over operator, a probability p_c is used. Its value is set in advance.

The PMX procedure

Step 1 For two individuals expressed using double strings, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, two crossover points are set at random.

Step 2 According to PMX, the upper strings of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, along with the corresponding lower strings are reordered, generating $\boldsymbol{s}_1^{'}$ and $\boldsymbol{s}_2^{'}$.

Step 3 For double strings, the offsprings, $\boldsymbol{s}_1^{''}$ and $\boldsymbol{s}_2^{''}$, resulting from the application of the revised PMX are obtained by exchanging the lower strings between the two crossover points $\boldsymbol{s}_1^{'}$ and $\boldsymbol{s}_2^{'}$.

The mutation operator is thought to fulfill the role of a local random search in genetic algorithms. For double strings, the index string expresses the priority of the parameters. For 0-1 strings, since the value of the 0-1 parameters themselves are expressed, strings with differing properties coexist in a single string, and it is necessary to apply mutations to each string. In this paper, the mutation operator is applied to each string, and inversion is used for index strings. For 0-1 strings, bit-reverse is introduced. When applying the mutation operator to individuals, it is first determined whether or not mutation will be applied to an individual according to the mutation probability p_m. In the case that mutation is applied, it is then determined whether to apply inversion or bit-reverse according to the mutation selection constant M_{Pum}.

Mutation procedure

Step 1 For an individual $\boldsymbol{s}$, expressed using a double string, a random number r_m is generated. If $r_m \leqq M_{Pum}$, a point on the 0-1 string is chosen at random and bit-reverse is performed, yielding $\boldsymbol{s}_1^{'}$. Otherwise, step 2 is adopted.

Step 2 Two points on the index string are chosen at random, and inversion is applied to the substring between the two points, yielding $\boldsymbol{s}_2^{'}$.

3.5 Procedure for avoiding the lower level GA

It is possible to derive a good approximate Stackelberg solution in a comparatively short time using the computation method we have presented. However, there is a problem with the algorithm of the existing method, and by solving it, the accuracy of the approximate Stackelberg solution is improved, and unnecessary calculations may be eliminated.

Regarding the problem, it occurs when an individual $\boldsymbol{x}$, handled by the upper level GA, exists multiply in the population of a single generation, or when the same individual reappears in the population of another generation, and results in the same problem (2) being solved using the lower level GA, yielding the same rational reaction, $\boldsymbol{y}(\boldsymbol{x})$.

In order to avoid this problem, thinking naively, for every $\boldsymbol{x}$ obtained by operating the upper level GA the

rational reaction of the lower level GA, $\boldsymbol{y}(\boldsymbol{x})$, may be placed in memory, and then whenever a past individual $\boldsymbol{x}$ reappears, the application of the lower level GA may be avoided by extracting the recorded rational reaction. However, when the scale of the problem is increased, storing the rational reaction data for every $\boldsymbol{x}$ becomes problematic due to limitations on memory space and the time needed to search for $\boldsymbol{x}$.

A storage region with a data structure of the form shown in Fig.2 is therefore established, and a proportion of the individual and corresponding rational reaction data, $\boldsymbol{x}$ and $\boldsymbol{y}(\boldsymbol{x})$, is stored.

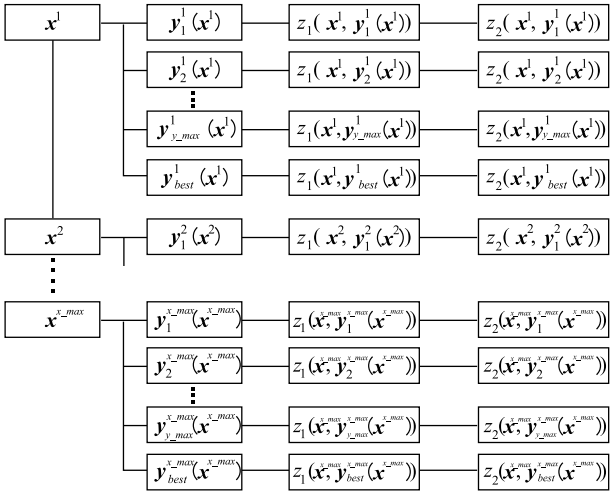

Figure 2: Storage region for $\boldsymbol{x}$ and $\boldsymbol{y}(\boldsymbol{x})$

$\boldsymbol{x}^i, i = 1, \ldots, x\_max$ express the values of $\boldsymbol{x}$ already handled as individuals by the upper level GA. $\boldsymbol{y}_j^i, i = 1, \ldots, x\_max, j = 1, \ldots, y\_max$ express the rational reaction values obtained from the lower level GA for each $\boldsymbol{x}^i$. x_max is the number of stored decisions $\boldsymbol{x}$. y_max is the number of stored rational reactions $\boldsymbol{y}(\boldsymbol{x})$. $z_1(\boldsymbol{x}^i, \boldsymbol{y}_j^i(\boldsymbol{x}^i))$ and $z_2(\boldsymbol{x}^i, \boldsymbol{y}_j^i(\boldsymbol{x}^i))$ are the values of the DM1 and DM2 objective functions for $\boldsymbol{x}^i$ and $\boldsymbol{y}_j^i(\boldsymbol{x}^i)$. $\boldsymbol{y}_{best}^i(\boldsymbol{x}^i)$ is the $\boldsymbol{y}_j^i(\boldsymbol{x}^i)$ which yields the largest value of $z_1(\boldsymbol{x}^i, \boldsymbol{y}_j^i(\boldsymbol{x}^i))$.

Using an algorithm such as the following reduces the number of times the lower GA is applied, and eliminates unnecessary computation.

Procedure for storing the rational reaction $\boldsymbol{y}(\boldsymbol{x})$ and avoiding the lower level GA

Step 1 If an individual $\bar{\boldsymbol{x}}$ of the upper level GA exists in a stored $\boldsymbol{x}^i$, then step 2 is adopted. If such an individual does not exist, it is checked whether the number of individuals $\boldsymbol{x}^i$ has reached x_max. If it has, then step 3 is adopted, if not, then step 4 is adopted.

Step 2 If $\boldsymbol{y}_j^i$ has reached y_max, then the saved $\boldsymbol{y}_{best}^i(\boldsymbol{x}^i)$ is returned to the upper level GA as the rational reaction and the algorithm terminates. If y_max has not been reached then step 4 is adopted.

Step 3 The element among the stored $\boldsymbol{x}^i$ for which the value of $z_1(\boldsymbol{x}^i, \boldsymbol{y}_{best}^i(\boldsymbol{x}^i))$ is smallest is selected, and that value, $\boldsymbol{x}$, is denoted $\boldsymbol{x}^k$. After obtaining the rational reaction $\boldsymbol{y}(\bar{\boldsymbol{x}})$ for $\bar{\boldsymbol{x}}$ by applying the lower level GA, the values are saved in the storage area of $\boldsymbol{x}^k$, and the algorithm terminates.

Step 4 After obtaining the rational reaction $\boldsymbol{y}(\bar{\boldsymbol{x}})$ for $\bar{\boldsymbol{x}}$ by applying the lower level GA, the values are saved and the algorithm terminates.

The algorithm of the previous method is thus refined by adopting the above methods.

3.6 Algorithm of the refined computation method

The algorithm of the refined computation method is summarized and presented below.

Step 1 The generation number of the upper level algorithm is set to $t := 0$, and N_u initial individuals are generated randomly.

Step 2 For each individual $\boldsymbol{x}$ in the upper level GA, it is judged whether or not to effect the procedure for avoiding the lower level GA. If it is effected, then a stored $\boldsymbol{y}(\boldsymbol{x})$ of the lower level is taken as the rational response, and step 3 is adopted. If it is not effected, the operations of the lower level GA from step 2-1 to step 2-3 are applied, and the rational response of the lower level, $\boldsymbol{y}(\boldsymbol{x})$, is obtained.

- **Step 2-1** N_l lower level GA individuals are randomly generated, and taken as the initial population of the lower level GA.
- **Step 2-2** Using a $\boldsymbol{x}$ given as an individual of the upper level GA, and a $\boldsymbol{y}$ generated by the lower level GA, the value of the objective function for DM2 is calculated, and an individual is reproduced using this value.
- **Step 2-3** If the current number of iterations exceeds a maximum number of generations set in advance, M_l, then step 3 is adopted. Otherwise, crossover operator and mutation operator are applied to each individual of the lower level GA and step 2-2 is repeated.

Step 3 The information related to the combination of the rational reaction $\boldsymbol{y}(\boldsymbol{x})$ obtained by operating the lower level GA and a given individual of the upper level GA, $\boldsymbol{x}$, is stored, and these values are used to calculate the value of the objective function for DM1.

Next, the convergence of the individual is measured using the clustering method, and the resulting degree is used as a basis for obtaining the fitness of each individual.

Step 4 If the current number of iterations in the upper level GA exceeds a maximum number of generations set in advance, M_u, then the algorithm is terminated. In such a case, the individual $(\boldsymbol{x}, \boldsymbol{y})$ with the largest fitness value among all those obtained until that point is regarded as the optimal individual. Otherwise, step 5 is adopted.

Step 5 Reproduction is performed using the fitness value of each individual in the upper level GA obtained using step 3, and crossover operator and mutation operator are applied. Next step 2 is repeated with $t := t + 1$.

4 Conclusion

In this paper, we considered the two level 0-1 programming problems in which there are two decision makers; the decision maker at the upper level and the decision maker at the lower level. By reforming a problem with the computation method for the Stackelberg solution previously presented by the authors, an improved computation method was presented. Specifically, the diversity of the population is maintained, and instead of computing the distances among all the individuals which occur during sharing, a GA adopting a clustering method is introduced in order to increase the precision of the derived solution. Also, for each decision $\boldsymbol{x}$, prompted by a duplicate among the population of the upper level GA, the rational response $\boldsymbol{y}(\boldsymbol{x})$ is not obtained by repeatedly applying the lower level GA. Instead, a proportion of $\boldsymbol{x}$ values already handled by the upper level GA and their rational responses $\boldsymbol{y}(\boldsymbol{x})$ are stored in memory, and when an identical $\boldsymbol{x}$ appears in a subsequent generation, the information regarding the stored $\boldsymbol{y}(\boldsymbol{x})$ is retrieved. An algorithm which reduces the number of times the lower level GA is applied was thus presented. In order to verify the effectiveness of the proposed method, it is intended to conduct numerical experiments into both the solution precision and computation time, and thus compare the method with previous methods.

References

[1] G. Anandalingam, R. Mathieu, C.L. Pittard, and N. Sinha: "Artificial intelligence based approaches for solving hierarchical optimization problems," in: Sharda, Golden, Wasil, Balci and Stewart (eds.), *Impacts of Recent Computer Advances on Operations Research,* North-Holland, pp. 289–301 (1989).

[2] M.R. Anderberg: *Cluster Analysis for Applications,* Academic press, (1975).

[3] J. Bard and J.Moore: "The mixed integer linear bilevel programming problem," *Operations Research*, Vol.38, pp.911-921 (1990).

[4] J. Bard and J.Moore: "An algorithm for the discrete bilevel programming problem," *Naval Research Logistics*, Vol.39, pp.419-435 (1992).

[5] D.E. Goldberg and J. Richardson: "Genetic algorithms with sharing for multimodal function optimization," *Proceedings of the Second International Conference on Genetic Algorithms,* Lawrence Erlbaum Associates, Hilsdale, New Jersey, pp. 41-49 (1987).

[6] D.E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Massachusetts (1989).

[7] D.E. Goldberg and R. Lingle: "Alleles, loci, and the traveling salesman problem," *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 154–159 (1985).

[8] K. Niwa: "Revised computational methods for using genetic algorithms for obtaining Stackelberg solutions to two-level 0-1 programming problems," *Essays and Studies in Commemoration of the 40th Anniversary of the Founding of Hiroshima University of Economics,* Hiroshima University Economics, pp. 771–794, in Japanese (2007)

[9] K. Niwa, I. Nishizaki and M. Sakawa: "Decentralized two-level 0-1 programming through genetic algorithms with double strings," *Proceedings of Second International Conference on Knowledge-Based Intelligent Electronic Systems*, Vol. 2, pp. 278–284 (1998).

[10] M. Sakawa, M. Tanaka: *Genetic Algorithms*, Asakura Publishing, in Japanese (1995).

[11] K. Shimizu, Y. Ishizuka and J.F. Bard: *Nondifferentiable and two-level mathematical programming*, Kluwer Academic Publishers (1997).

[12] W.P. Wen, and Y.H. Yang: "Algorithms for solving the mixed integer two-level linear programming problem," *Computers and Operations Research*, Vol. 17, pp. 133–142 (1990).

[13] X. Yin and N. Germay: "A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization," *Proceedings of the International Conference in Innsbruck, Austria, 1993,* Springer-Verlag, Wien, pp. 450–457 (1993).