Paper

Computational Methods for Two-Level 0-1 Programming Problems through Distributed Genetic Algorithms

Keiichi Niwa, Tomohiro Hayashida, and Masatoshi Sakawa

Abstract—In this paper, we consider a two-level 0-1 programming problem in which there is not coordination between the decision maker (DM) at the upper level and the decision maker at the lower level. We propose a revised computational method that solves problems related to computational methods for obtaining the Stackelberg solution. Specifically, in order to improve the computational accuracy of approximate Stakelberg solutions and shorten the computational time of a computational method implementing a genetic algorithm (GA) proposed by the authors, a distributed genetic algorithm is introduced with respect to the upper level GA, which handles decision variables for the upper level DM. Parallelization of the lower level GA is also performed along with parallelization of the upper level GA. The proposed algorithm is also improved in order to eliminate unnecessary computation during operation of the lower level GA, which handles decision variables for the lower level DM. In order to verify the effectiveness of the proposed method, we propose comparisons with existing methods by performing numerical experiments to verify both the accuracy of the solution and the time required for the computation.

Keywords—distributed genetic algorithm, Stackelberg solution, two-level 0-1 programming problem.

1. Introduction

In the real world, we can often encounter situations that 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 [1]. In two-level programming problems, the upper level DM makes his/her decision first, and then, with full knowledge of the decision of the upper level DM, the lower level DM makes his/her decision in order to optimize his/her own objective function. According to this rule, the upper level DM also makes a decision so as to optimize the objective function of self. The solution defined as the above mentioned procedure is 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 variables.

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 [2] and two-level mixed integer programming problems [3]. Wen *et al.* [4] have presented a computation method for obtaining the Stackelberg solution to two-level programming problems which have 0-1 variables for the decision variables in the upper level and real variables for the decision variables in the lower level.

On the other hand, 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 [5], 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 [6].

An example of research related to two-level programming problems using GAs is given by Anandalingam, *et al.* [7] which presents a method for deriving a Stackelberg solution for two-level linear programming problems. Also, Nishizaki, *et al.* presented an algorithm based on GAs in order to derive the Stackelberg solution for two-level integer programming problems [8] and two-level mixed integer programming problems [9]. In order to derive a Stackelberg solution for 0-1 programming problems related to two-level decentralized systems, the authors [10] have also proposed a computational method that adopts the double string proposed by Sakawa, et al as the individual representation. In order to improve the computational accuracy of approximate Stakelberg solutions, the authors have proposed computational methods that implement sharing [11] and cluster analysis [12] methods. Furthermore, the authors have proposed a computational method using parallel genetic algorithm [13]. Use of these methods allows for the derivation of approximate Stackelberg solutions with relatively high precision and in a relatively short time, but there is still room for improvement, particularly with regards to calculation times.

Therefore, this paper focuses on two-level 0-1 programming problems, and proposes an improved computational method that addresses problems related to the computational method proposed by the authors for deriving the Stackelberg solution. Specifically, a distributed genetic algorithm is introduced with respect to the upper level GA, which handles decision variables for the upper level DM, in order to improve the computational accuracy of approximate Stackelberg solutions and decrease the computational time of a computational method implementing a genetic algorithm proposed by the authors. Also, parallelization of the lower level GA is performed along with parallelization of the upper level GA. The proposed algorithm is also improved in order to eliminate unnecessary computation during operation of the lower level GA, which handles decision variables for the lower level DM. In order to verify the effectiveness of the proposed method, we propose comparisons with the existing method and the computational method using parallel GA by performing numerical experiments to verify both the accuracy of the solution and the time required for the computation.

2. 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 two-level 0-1 programming problem is expressed as

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

where $\mathbf{x} = (x_1,\dots,x_{n_1})^T$, and $\mathbf{y} = (y_1,\dots,y_{n_2})^T$ are the vectors of decision variables for DM1 and DM2; $z_1(\mathbf{x},\mathbf{y})$, and $z_2(\mathbf{x},\mathbf{y})$ respectively represent the objective functions of DM1 and DM2; $\mathbf{c}_1 = (c_{11},\dots,c_{1n_1})$, $\mathbf{d}_1 = (d_{11},\dots,d_{1n_2})$, $\mathbf{c}_2 = (c_{21},\dots,c_{2n_1})$, and $\mathbf{d}_2 = (d_{21},\dots,d_{2n_2})$ denote the coefficient vectors of the objective functions; A and B are $m \times n_1$ and $m \times n_2$ coefficient matrices in the constraints, respectively; $\mathbf{b} = (b_1,\dots,b_m)^T$ is a coefficient vector of the right hand side of the constraints; the superscript T means transposition of a vector.
For the sake of simplicity, in this paper, it is assumed that each component of A, B, $\mathbf{b}$, $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{d}_1$, and $\mathbf{d}_2$ 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 in order to maximize the objective function with full knowledge of the decision of DM1. That is to say, when the decision by DM1 is denoted $\hat{\mathbf{x}}$, DM2 solves the 0-1 programming problem (2) with parameters $\hat{\mathbf{x}}$, choosing the optimal solution $\mathbf{y}(\hat{\mathbf{x}})$ as the rational reaction to $\hat{\mathbf{x}}$.

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

Under this premise, DM1 also determines $\mathbf{x}$ by choosing the value which maximizes 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

We propose a computational method through GA in order to obtain Stackelberg solutions to the two-level 0-1 programming problems. In Subsections 3.1, 3.2 and 3.3, first, we describe fundamental elements of GA, which are coding procedure, a decoding procedure and genetic operators, used in the computational method using GA [10]. In this paper, we call the computational method using GA [10] normal GA (NGA). Furthermore, we show additional elements used in the proposed computational method using distributed genetic algorithm and the computational method using parallel genetic algorithm [13] in Subsections 3.4 and 3.5. Finally, the algorithm used in the proposed computational method using distributed genetic algorithm is described in Subsection 3.6.

3.1. Coding and decoding

When solving 0-1 programming problems using GAs, binary strings are usually adopted to express individuals [5], [14]. However, under this representation it is possible that infeasible individuals that do not satisfy the constraints may be generated, so there is a danger that the performance of the GAs may degrade. Thus, in this paper, a double string [6] is used which is composed of the substring corresponding to the decision of DM1, $\mathbf{x}$, and the substring corresponding to the decision of DM2, $\mathbf{y}$, as shown in Fig.1 in order to derive only feasible solutions. The decisions of DM1 and DM2 are handled by performing genetic operators on each sub-individual. We call the GA operating to the decision $\mathbf{x}$ of DM1 the upper level GA, and the GA operating to the decision $\mathbf{y}$ of DM2 the lower level GA.

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

Fig. 1. Double string.

In Fig. 1 $s_{i_x(m)} \in \{0,1\}, i_x(m) \in \{1,\dots,n_1\}$, and for $m \neq m'$ it is assumed that $i_x(m) \neq i_x(m')$. Similarly, $s_{i_y(m)} \in \{0,1\}, i_y(m) \in \{1,\dots,n_2\}$, 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 indexes of the elements of each solution vector respectively, and their values.

In order to generate only feasible solutions, a decoding algorithm proposed by the authors [10] is also applied to the upper level and the lower level GA.

3.2. Reproduction

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

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

3.3. Crossover and Mutation

For double strings, if single-point or multi-point crossover operators are performed then there is a possibility that infeasible individuals 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. When solving the traveling salesman problem or the scheduling problem through GAs, this kind of violation occurs. In order to circumvent such violation, partially matched crossovers (PMX) have been devised. In this paper, a modified version of PMX is used in order to handle the double strings proposed by Sakawa *et al.* [6]. Also, when determining whether or not to apply the crossover operator, a probability p_c is used. Its value is set in advance.

PMX procedure

Step 1: For two individuals expressed using double strings, s_1 and s_2, two crossover points are set at random.

Step 2: According to PMX, the upper strings of s_1 and s_2, along with the corresponding lower strings are reordered, generating s_1' and s_2'.

Step 3: For double strings, the offsprings, s_1'' and s_2'', resulting from the application of the revised PMX are obtained by exchanging the lower strings between the two crossover points s_1' and s_2'.

It is well recognized that the mutation operator plays a role of local random search in genetic algorithms. In this paper, the mutation operator is applied to each string, and inversion is used for index strings. For binary strings, mutation of bit-reverse type is adopted. When applying the mutation operator to individuals, it is first determined whether or not the mutation operator 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 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 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 s_2'.

3.4. Application of the parallel genetic algorithm

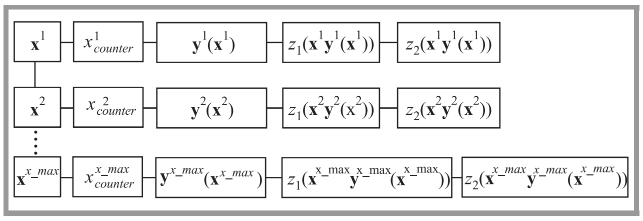
In genetic algorithms, it is possible to perform parallel processing in the greater part of the operations included in the algorithm. In reproduction operations, however, because it is necessary to calculate evaluation values for each individual in a population, and based on that value determine the fitness of each individual, direct application of parallel processing is difficult. Research related to the parallelization of GAs started with improvements to such barriers to the implementation of parallelization, and a variety of types of models have been proposed and their effectiveness noted by numerous researchers [15], [16], [17]. Today, GAs that implement parallel processing have come to be called parallel genetic algorithms.

The computational method proposed by the authors [13] divides the lower level GA operations and assigns them across multiple processors. Also, the computational method adopts the single-population master-slave GAs as the upper level GA. By assigning the calculation of individual fitness values, crossover operator and mutation operator to multiple processors, calculation times are reduced. We call the computational method proposed by the authors the computational method using parallel GA (PGA). However, while the computational method using parallel GA succeeds in obtaining good approximate solutions and reducing the amount of computational time, there is still likely much more room for improvement. In this study, therefore, we aim for further improvements of the precision of approximate solutions and further reductions in computational time, and consider parallelization of the upper level GA and the lower level GA implemented by the computational method using parallel GA.

The multiple-population genetic algorithms performs parallel processing by dividing the population and assigning the partial populations (sub-populations) to multiple processors. If we adopt this model as our computational method, it is possible to use the computational method using GA proposed by authors [12] without a lot of modifications. In that case, the multiple-population genetic algorithm is adopted as the upper level GA. The upper level GA operations proposed by authors are applied for partial population assigned to each processor. Additionally, migration operator is performed in every migration interval. Also, it is able to divide the lower level GA operations and assign them across multiple processor in the same way as PGA. In this paper, we employ a multiple-population genetic algorithms (distributed genetic algorithms) so as to obtain good approximate Stackelberg solutions and reduce calculation times. We call the proposed computational method using distributed GA (DGA).

3.5. Lower Level GA Avoidance Procedures

In this paper we introduce a storage region as shown in Fig. 2.



$\mathbf{x}^1$
$x^1_{counter}$
$\mathbf{y}^1(\mathbf{x}^1)$
$z_1(\mathbf{x}^1\mathbf{y}^1(\mathbf{x}^1))$
$z_2(\mathbf{x}^1\mathbf{y}^1(\mathbf{x}^1))$
$\mathbf{x}^2$
$x^2_{counter}$
$\mathbf{y}^2(\mathbf{x}^2)$
$z_1(\mathbf{x}^2\mathbf{y}^2(\mathbf{x}^2))$
$z_2(\mathbf{x}^2\mathbf{y}^2(\mathbf{x}^2))$
$\mathbf{x}^{x\_max}$
$x^{x\_max}_{counter}$
$\mathbf{y}^{x\_max}(\mathbf{x}^{x\_max})$
$z_1(\mathbf{x}^{x\_max}\mathbf{y}^{x\_max}(\mathbf{x}^{x\_max}))$
$z_2(\mathbf{x}^{x\_max}\mathbf{y}^{x\_max}(\mathbf{x}^{x\_max}))$

Fig. 2. Storage for saving $\mathbf{x}$ and $\mathbf{y}(\mathbf{x})$.

Here, $\mathbf{x}^i, i = 1, \ldots, x\_max$ indicate those values of $\mathbf{x}$ that were used in the past for handling individuals of the upper level GA, and $\mathbf{y}^i(\mathbf{x}^i), i = 1, \ldots, x\_max$ indicate the values of the rational reactions associated with $\mathbf{x}^i$ obtained by the lower level GA. $x^i_{counter} \in \{1, 2, \ldots, y\_max\}, i = 1, \ldots, x\_max$ indicate the number of times that the lower level GA was used to find the rational reaction $\mathbf{y}^i(\mathbf{x}^i)$ for $\mathbf{x}^i$. *x_max* indicates the maximum number of DM1 decisions $\mathbf{x}$ saved, and *y_max* indicates the maximum number of times that the lower level GA can be repeatedly used to find the rational reaction $\mathbf{y}^i(\mathbf{x}^i)$ for $\mathbf{x}^i$. $z_1(\mathbf{x}^i, \mathbf{y}^i(\mathbf{x}^i))$ and $z_2(\mathbf{x}^i, \mathbf{y}^i(\mathbf{x}^i))$ are stored $\mathbf{x}^i, \mathbf{y}^i(\mathbf{x}^i)$ values used in place of DM1 and DM2 objective functions. By using the following algorithm, the number of applications of the lower level GA is reduced, and unnecessary calculation times eliminated.

Storage of the rational reaction $\mathbf{y}(\mathbf{x})$ and lower level GA avoidance procedures

Step 1: If there exists in $\mathbf{x}^i$ an upper level GA individual $\bar{\mathbf{x}}$, proceed to Step 2. If one does not exist, then check if the number of $\mathbf{x}^i$ has reached *x_max*, and if so continue on to Step 3. If not, proceed to Step 4.

Step 2: If $x^i_{counter}$ has reached *y_max*, then the saved $\mathbf{y}^i(\mathbf{x}^i)$ is returned to the upper level GA as the rational reaction and the algorithm terminates. If not reached, proceed to Step 4.

Step 3: Select the least of the values $z_1(\mathbf{x}^i, \mathbf{y}^i(\mathbf{x}^i))$ from the saved $\mathbf{x}^i$, and take that $\mathbf{x}^i$ value as $\mathbf{x}^k$. After applying the lower level GA and thus obtaining the rational reaction $\mathbf{y}(\bar{\mathbf{x}})$ for $\bar{\mathbf{x}}$, if $z_1(\mathbf{x}^k, \mathbf{y}^k(\mathbf{x}^k)) \leq z_1(\bar{\mathbf{x}}, \mathbf{y}(\bar{\mathbf{x}}))$, save $\bar{\mathbf{x}}$, $\mathbf{y}(\bar{\mathbf{x}})$, $z_1(\bar{\mathbf{x}}, \mathbf{y}(\bar{\mathbf{x}}))$, $z_2(\bar{\mathbf{x}}, \mathbf{y}(\bar{\mathbf{x}}))$ in the storage region $\mathbf{x}^k$, and terminate the algorithm.

Step 4: After obtaining the rational reaction $\mathbf{y}(\bar{\mathbf{x}})$ for $\bar{\mathbf{x}}$ by applying the lower level GA, save $\bar{\mathbf{x}}$, $\mathbf{y}(\bar{\mathbf{x}})$, $z_1(\bar{\mathbf{x}}, \mathbf{y}(\bar{\mathbf{x}}))$, $z_2(\bar{\mathbf{x}}, \mathbf{y}(\bar{\mathbf{x}}))$, and terminate the algorithm.

Implementation of the algorithm described above improved upon previous methods.

3.6. The Algorithm for the Improved Computational Method

The algorithm used in the computational method after improvement can be described as follows, N_p denotes the number of processors.

Step 1: For each processor $q, q = 1, \ldots, N_p$, apply the upper level GA operations on Step1 through Step 7. Taking the generation of the upper level GA as $t_{uq} := 0$, N_u initial individuals are randomly generated.

Step 2: For each individual $\mathbf{x}$ in the upper level GA, determine whether or not to apply the lower level GA, and find the number of lower level GA to apply, N_{ul}. For those individuals N_{ul}, apply the lower level GA operations in Step 2-1 through Step 2-3, and obtain the rational reaction $\mathbf{y}(\mathbf{x})$. For those $N_u - N_{ul}$ individuals to which the lower level GA will not be applied, take the saved $\mathbf{y}(\mathbf{x})$ as the rational reaction, and proceed to Step 4.

Step 2-1: Set $t_l := 0$. Randomly generate N_l lower level GA individuals y, and take these as the initial population of the lower level GA . Proceed to Step 2-2.

Step 2-2: By using $\mathbf{x}$ given as the upper level GA individual and $\mathbf{y}$ generated by the lower level GA, the DM2 objective function value is calculated. After applying linear scaling to the value, the reproduction operator is applied. Proceed to Step 2-3.

Step 2-3: If t_l has exceeded the previously defined a maximum number of generation M_l, take the individual with the best fitness value as the optimal individual $\mathbf{y}(\mathbf{x})$, and proceed to Step 3. Otherwise, apply crossover operator and mutation operator to each lower level GA individual, let $t_l = t_l + 1$, and proceed to Step 2-2.

Step 3: By using the lower level rational reaction $\mathbf{y}(\mathbf{x})$ obtained by operation of the lower level GA and the individual $\mathbf{x}$ of the upper level GA, calculate the values for the DM1 and DM2 objective functions. Perform the procedures required to save $\mathbf{x}$ and its rational reactions $\mathbf{y}(\mathbf{x})$ to the storage region, and proceed to Step 4.

Step 4: Calculate the DM1 objective function for each upper level GA individual $\mathbf{x}$, and after performing linear scaling, apply the clustering method to measure the level of convergence of the individuals. Depending upon the degree of convergence, calculate the fitness value of each individual. Proceed to Step 5.

Step 5: If t_{uq} has exceeded the previously set a maximum number of generation M_u, then terminate the algorithm. In that case, the individual obtained up to that generation with the best fitness value is taken as the optimal individual $(\mathbf{x}, \mathbf{y})$. Otherwise, proceed to Step 6.

Step 6: Reproduction operator is performed using the fitness values of each individual of the upper level GA. Apply crossover operator and mutation operator to each upper level GA individual, and proceed to Step 7.

Step 7: If t_{uq} mod m_i (migration interval) = 0, after performing synchronization between the processors, apply migration. Return to Step 2 with $t_{uq} := t_{uq} + 1$.

4. Numerical Experiments

Numerical experiments are carried out in order to demonstrate the feasibility and the effectiveness of DGA. We apply DGA, PGA, and NGA to twelve types of two-level 0-1 programming problems as shown in Table 1. Each problem has five constraints.

Table 1
Problems used in the numerical experiments

Problem	DM1 variables	DM2 variables	Constraint strength
A	15	15	I (50%)
			II (70%)
			III (90%)
B	20	20	I (50%)
			II (70%)
			III (90%)
C	25	25	I (50%)
			II (70%)
			III (90%)
D	30	30	I (50%)
			II (70%)
			III (90%)

In this case, the elements A, B, $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{d}_1$, and $\mathbf{d}_2$ of the two-level 0-1 programming problem are selected at random from the closed interval $[10, 99]$, and the b_i element of $\mathbf{b}$ is set according to the equation

$$b_i = r_i \left(\sum_{j=1}^{n_1} a_{ij} + \sum_{k=1}^{n_2} b_{ik} \right), i = 1, \ldots, m. \qquad (3)$$

Furthermore, a_{ij} represents the ij element of matrix A, and b_{ik} represents the ik element of matrix B. Here, r_i represents the strength of the constraint. In the strong constraint problem (I), a random number is determined at random from the closed interval $[0.45, 0.55]$, in the middle constraint problem (II), a random number is selected at random from the closed interval $[0.65, 0.75]$, and in the weak constraint problem (III), a random number is determined at random from the closed interval $[0.85, 0.95]$, respectively. The decimal portion of the b_i value is rounded off, and the result is stored as an integer value.

Next, the GA parameters are set for NGA, PGA, and DGA as follows. First, there are parameters that are used in common by all three methods, namely the population sizes of the upper level GA and the lower level GA, the crossover rate, the mutation rate, and the maximum number of generation, and those values are set as 120 for the population size, 0.9 for the crossover rate, 0.02 for the mutation rate, and 300 for the maximum number of generation. Next, for PGA and DGA, α is set to 0.25. The initial number of clusters, k, is set to 5, and d_{max} and d_{min}, used to measure the distance between individuals, are set to 2.5 and 1.0, respectively. In addition to these parameters, x_max and y_max are set to 100 and 5, respectively. The number of processors is set to 3. Finally, for DGA, we apply the random ring model to the communication topology and adopt Best-Hole model as the selection of emigrants and immigrants. The number of migration interval m_i is set to 15. The migration rate is set to 2.5%.

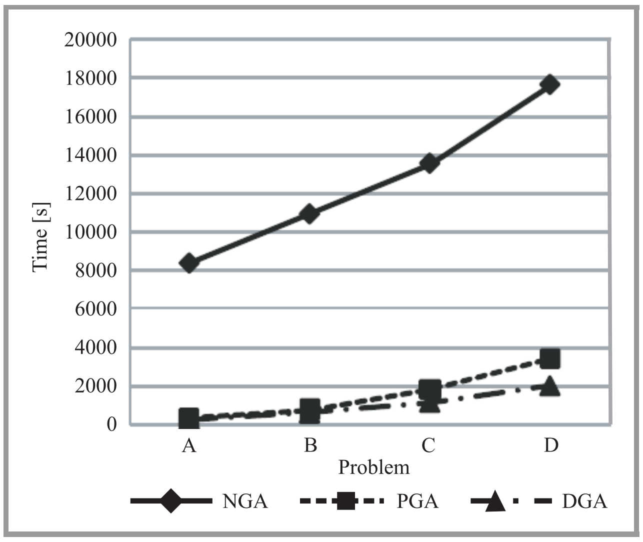


Fig. 3. Comparison of calculation times (the strong constraint problem (I)).

Table 2
Comparison of solution precisions

Problem	Constraint strength	DGA				PGA			
		best	worst	average	variance	best	worst	average	variance
A	I	1078	1078	1078.0	0.00	1078	1078	1078.0	0.00
	II	1377	1377	1377.0	0.00	1377	1377	1377.0	0.00
	III	1727	1727	1727.0	0.00	1727	1727	1727.0	0.00
B	I	1342	1342	1342.0	0.00	1342	1342	1342.0	0.00
	II	1764	1764	1764.0	0.00	1764	1764	1764.0	0.00
	III	2157	2157	2157.0	0.00	2157	2157	2157.0	0.00
C	I	1713	1713	1713.0	0.00	1713	1713	1713.0	0.00
	II	2221	2207	2214.6	43.84	2221	2207	2215.7	20.81
	III	2680	2680	2680.0	0.00	2680	2680	2680.0	0.00
D	I	1864	1857	1859.8	11.76	1864	1851	1856.8	12.36
	II	2480	2444	2469.6	253.44	2480	2439	2449.7	233.01
	III	2930	2930	2930.0	0.00	2930	2930	2930.0	0.00

Problem	Constraint strength	NGA				Enumeration
		best	worst	average	variance	
A	I	1078	1068	1077.0	9.00	1078
	II	1377	1377	1377.0	0.00	1377
	III	1727	1727	1727.0	0.00	1727
B	I	1342	1328	1340.4	17.24	-
	II	1764	1754	1763.0	9.00	-
	III	2157	2157	2157.0	0.00	-
C	I	1713	1713	1713.0	0.00	-
	II	2221	2207	2210.7	23.81	-
	III	2680	2672	2678.5	9.05	-
D	I	1864	1849	1856.9	11.29	-
	II	2480	2444	2470.0	80.80	-
	III	2930	2915	2927.0	20.00	-

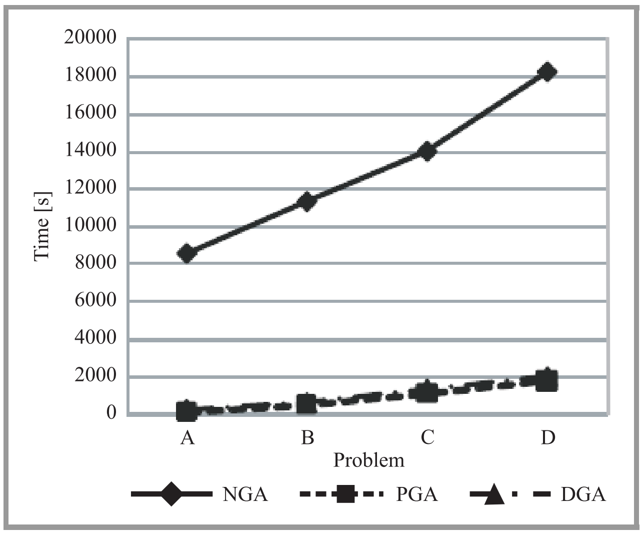


Fig. 4. Comparison of calculation times (the middle constraint problem (II)).

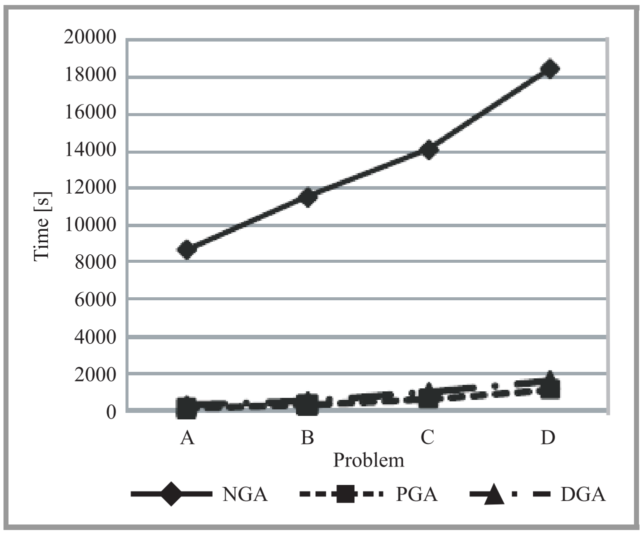


Fig. 5. Comparison of calculation times (the weak constraint problem (III)).

Next we will describe the experimental environment. The experiment is run on a personal computer with a 2.80 GHz CPU and running Windows XP. The compiler used is Microsoft Visual C++ 6.0.

For each problem, NGA, PGA, and DGA are run ten times each. The results are given in Table 2 through Table 5.
In Table 2, for all trials of Problem A, DGA, PGA, and NGA derive precise Stackelberg solutions. But, for the re-

Table 3
Comparison of calculation times

Problem	Constraint strength	DGA				PGA			
		best	worst	average	variance	best	worst	average	variance
A	I	220.78	351.25	269.82	1138.83	213.64	434.43	319.70	3611.94
	II	247.30	320.80	275.36	482.07	108.05	166.72	128.41	316.94
	III	215.46	281.27	253.94	427.14	62.20	99.46	82.38	191.60
B	I	394.09	801.84	599.33	12700.54	599.11	856.95	770.43	4550.59
	II	464.08	772.94	610.84	7149.91	419.83	569.50	504.90	2619.66
	III	345.83	633.73	473.91	5558.42	248.95	360.14	293.30	1186.14
C	I	975.14	1326.10	1149.18	10536.46	1728.47	1913.19	1824.04	4479.17
	II	1092.68	1569.18	1337.10	25015.18	1067.88	1199.30	1116.03	1888.83
	III	772.44	1191.86	965.45	16951.74	537.94	665.39	605.52	1484.93
D	I	1640.06	2426.27	2000.36	53181.01	3347.20	3547.55	3428.30	3115.52
	II	1568.27	2515.28	1980.26	81371.51	1753.85	1781.79	1767.90	97.18
	III	1012.63	2268.36	1619.37	89178.03	1073.29	1169.15	1125.58	848.64

Problem	Constraint strength	NGA				Enumeration
		best	worst	average	variance	
A	I	8379.75	8613.52	8414.59	4578.06	566.73
	II	8550.34	8668.06	8618.90	842.89	652.38
	III	8708.25	8714.92	8711.78	3.91	665.81
B	I	10882.38	11006.64	10954.14	1946.30	–
	II	11296.80	11467.94	11360.96	2888.93	–
	III	11447.75	11907.14	11520.65	17290.82	–
C	I	13467.95	13651.13	13573.35	1886.21	–
	II	14022.60	14067.24	14032.10	147.58	–
	III	14060.64	14090.92	14081.75	80.04	–
D	I	17532.97	17794.47	17646.13	4659.97	–
	II	18208.95	18263.34	18250.02	235.16	–
	III	18424.03	18541.39	18461.96	1912.68	–

Table 4
Lower level GA avoidance counts

Problem	Constraint strength	DGA				PGA			
		best	worst	average	variance	best	worst	average	variance
A	I	35558	34898	35283.7	46901.01	32008	27418	29742.4	1589311.44
	II	35539	35093	35291.8	24252.76	31094	26980	29362.6	1272588.24
	III	35634	35133	35371.4	24639.24	32006	29261	30431.2	948838.16
B	I	34983	33060	33995.6	265319.84	21852	14956	17355.0	3164457.60
	II	34757	32693	33959.8	317217.36	20845	14153	17304.8	4949162.76
	III	35274	34007	34667.5	121787.45	23258	17347	20721.5	3514280.45
C	I	32740	30118	31504.7	518636.01	8626	5577	6963.4	1213648.44
	II	32705	30017	31094.1	677375.69	5164	1599	3707.9	1823801.69
	III	33408	31591	32853.0	390069.20	16092	10170	13038.9	3552493.09
D	I	30696	26108	27780.8	1576362.56	1205	83	490.9	110786.89
	II	30819	26494	28875.6	2235743.84	847	114	292.5	46420.85
	III	32731	27108	30240.6	1688052.64	5939	3469	4376.3	514510.01

sults of Problem A-I, NGA is not possible to obtain precise Stackelberg solutions in some trials. Performance in deriving the best solutions to Problem B is equal for all three methods, but comparing the worst values and the average values shows that DGA and PGA present the best performance. Observe that the best values and the worst values obtained by DGA for Problem C and Problem D are equal or superior to the corresponding values obtained by NGA. Also, for Problem C and Problem D without Problem D-II, comparing the average values shows that DGA is superior to NGA. Finally, for Problem C and Problem D, DGA is superior or approximately equivalent to PGA in all results. When comparing the three methods with regards to the solution precision obtained, DGA is superior.

Table 5
Comparison of generations in which the best solution was found

Problem	Constraint strength	DGA				PGA			
		best	worst	average	variance	best	worst	average	variance
A	I	5	147	37.8	2210.96	3	42	13.9	130.09
	II	6	20	11.7	16.81	2	13	9.0	8.80
	III	5	20	11.0	20.80	3	16	9.3	18.81
B	I	5	158	54.4	1845.04	13	147	62.2	2204.76
	II	9	53	16.7	155.01	7	22	16.2	27.56
	III	9	29	16.6	35.24	9	30	18.9	35.69
C	I	15	71	23.0	258.60	15	59	38.3	180.01
	II	14	165	61.1	3067.89	50	292	141.1	6644.29
	III	13	247	75.6	6204.84	26	53	40.8	75.76
D	I	12	256	68.4	7015.44	62	291	157.3	5343.41
	II	23	299	117.7	9811.21	36	249	159.9	4262.09
	III	19	70	39.6	275.04	30	239	74.9	3441.69

Problem	Constraint strength	NGA				Enumeration
		best	worst	average	variance	
A	I	5	208	48.0	6130.80	–
	II	4	16	8.4	10.64	–
	III	3	126	22.4	1205.84	–
B	I	13	133	45.7	1728.41	–
	II	9	44	16.3	91.61	–
	III	9	63	19.5	223.25	–
C	I	11	35	20.5	34.85	–
	II	16	66	28.2	352.76	–
	III	13	26	18.8	17.36	–
D	I	19	108	34.7	666.61	–
	II	14	103	51.1	721.69	–
	III	23	284	64.1	5833.29	–

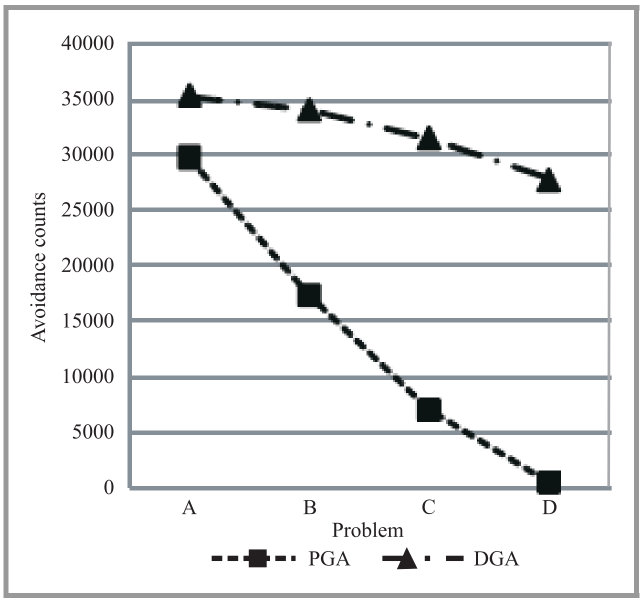


Fig. 6. Comparison of average avoidance counts (the strong constraint problem (I)).

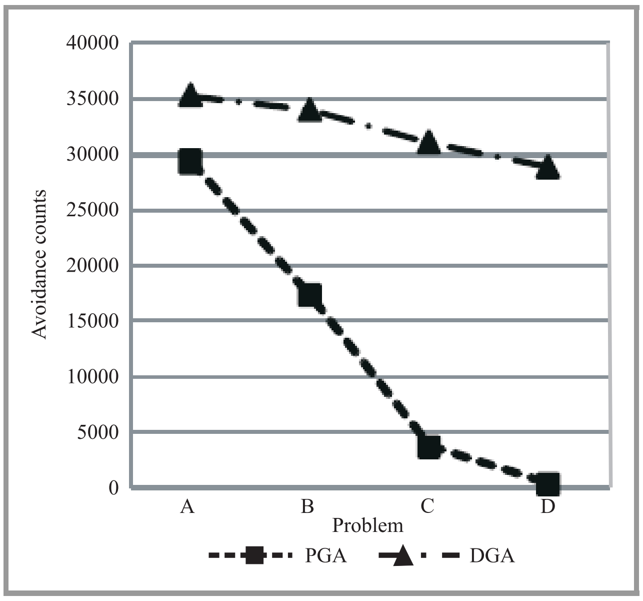


Fig. 7. Comparison of average avoidance counts (the middle constraint problem (II)).

Examining the calculation time results displayed in Table 3, for all trials of all problems, DGA and PGA are superior to NGA. Also, the calculation times of DGA are less than 15% of the calculation times of NGA.

Figure 3 shows the average calculation times for each size of the strong constraint problems (I). We can see that as compared to PGA, DGA is superior.

The average calculation times for each size of the middle constraint problems (II) are illustrated in Fig. 4. When comparing DGA and PGA, DGA is approximately equivalent or slightly inferior.

The average calculation times for each size of the weak constraint problems (III) are shown in Fig. 5. From Fig. 5, DGA is approximately equivalent or slightly inferior to PGA.

The results of the avoidance count for the lower level GA are shown in Table 4. From the results listed in this table, we see that with both DGA and PGA, as the scale of the problem increases the number of lower level GA avoidances is reduced.

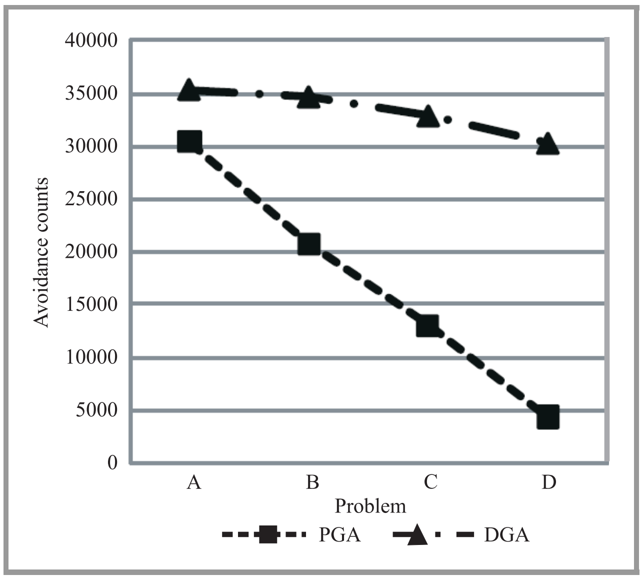


Fig. 8. Comparison of average avoidance counts (the weak constraint problem (III)).

The average avoidance counts for each size of the strong constraint problems (I), the middle constraint problems (II), and the weak constraint problems (III) are shown in Figs. 6–8. Examining the results displayed in these figures, we see that the avoidance counts of DGA gradually decrease as the size of the problem increases. On the other hand, the avoidance counts of PGA rapidly decrease.

Finally, the results of the comparison of generations in which the best solution was found are illustrated in Table 5. The results listed in Table 5 show that with both DGA and PGA, the generation deriving the best result is even later than with NGA, and so it is likely that by introducing cluster analysis methods the diversity of individuals within the population of the upper level GA would be maintained, and rapid population convergence avoided.

From the above results, compared to the other two computational methods, DGA is the superior computational method, both from a standpoint of solution precision and required calculation time.

5. Conclusion

This paper has focused on a two-level 0-1 programming problem in which there is not coordination between the decision maker at the upper level and the decision maker at the lower level. The authors have proposed a modified computational method that solves problems related to computational methods for obtaining the Stackelberg solution. Specifically, in order to improve the computational accuracy of approximate Stakelberg solutions and shorten the computational time of a computational method implementing GA proposed by the authors, a distributed genetic algorithm has been introduced with respect to the upper level GA, which handles decision variables for the upper level DM. Also, parallelization of the lower level GA has been performed along with parallelization of the upper level GA. The proposed algorithm has been improved in order to eliminate unnecessary computation during operation of the lower level GA, which handles decision variables for the lower level DM. In order to verify the effectiveness of the proposed method, numerical experiments have been carried out. From the results, we have shown that the proposed method is the superior to the other two computational methods, both from a standpoint of solution precision and required calculation time.

References

[1] K. Shimizu, Y. Ishizuka, and J. F. Bard, *Nondifferentiable and Two-Level Mathematical Programming*. Norwell: Kluwer, 1997.

[2] J. Bard and J.Moore, "An algorithm for the discrete bilevel programming problem", *Nav. Res. Log.*, vol. 39, pp. 419–435, 1992.

[3] J. Bard and J. Moore, "The mixed integer linear bilevel programming problem", *Oper. Res.*, vol. 38, pp. 911–921, 1990.

[4] W. P. Wen and Y. H. Yang, "Algorithms for solving the mixed integer two-level linear programming problem", *Comput. Oper. Res.*, vol. 17, pp. 133–142, 1990.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Norwell: Addison Wesley, 1989.

[6] M. Sakawa and M. Tanaka, *Genetic Algorithms*. Tokyo: Asakura Publishing, 1995.

[7] G. Anandalingam, R. Mathieu, C. L. Pittard, and N. Sinha, "Artificial intelligence based approaches for solving hierarchical optimization problems," in *Impacts of Recent Computer Advances on Operations Research,* R. Sharda, B. L. Golden, E. Wasil, O. Balci and W. Stewart, Eds. New York: Elsevier Science Publishing, 1989, pp. 289–301.

[8] I. Nishizaki and M. Sakawa, "Computational methods through genetic algorithms for obtaining Stackelberg solutions to two-level integer programming problems", *Cybernet. Syst. Int. J.*, vol. 36, pp. 565–579, 2005.

[9] I. Nishizaki and M. Sakawa, "Computational methods through genetic algorithms for obtaining Stackelberg solutions to two-level mixed zero-one programming problems", *Cybernet. Syst. Int. J.*, vol. 31, pp. 203–221, 2000.

[10] K. Niwa, I. Nishizaki, and M. Sakawa, "Decentralized two-level 0-1 programming through genetic algorithms with double strings", in *Proc. 2nd Int. Conf. Knowl. Int. Electron. Syst.*, 1998, vol. 2, pp. 278–284.

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

[12] K. Niwa, I. Nishizaki, and M. Sakawa, “Two-Level 0-1 Programming Using Genetic Algorithms and a Sharing Scheme Based on Cluster Analysis”, in *Proc. Int. MultiConf. Eng. Comput. Sci.*, Hong Kong, China, 2008, pp. 1931–1936.

[13] K. Niwa, I. Nishizaki, and M. Sakawa, “Two-level 0-1 programming through parallel genetic algorithms”, in *Proc. Int. MultiConf. Eng. Comput. Sci.*, Hong Kong, China, 2009, vol. 2, pp. 2000–2005.

[14] D. E. Goldberg and R. Lingle, “Alleles, loci, and the traveling salesman problem”, in *Proc. First Int. Conf. Genet. Algorithms Appl.*, Hillsdale, USA, 1985, pp. 154—159.

[15] D. Abramson and J. Abela, “A parallel genetic algorithm for solving the school timetabling problem”, in *Proc. 15th Austr. Comput. Sci. Conf.*, Hobart, Australia, 1992, vol. 14, pp. 1–11.

[16] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms.* Norwell, Massachusetts: Kluwer, 2000.

[17] T. C. Fogarty and R. Huang, “Implementing the genetic algorithm on transputer based parallel processing systems”, in *Proc. First Int. Conf. Parallel Prob. Solv. Nat.*, Dortmund, Germany, 1990, pp. 145–149.

Keiichi Niwa received the Ph.D. degree in systems engineering from Hiroshima University in 2000. Since 2000, he has jointed Hiroshima University of Economics where he is currently a Associate Professor at Department for Information Systems in Business. His main research interests include meta-heuristics and integer programming.

e-mail: ki-niwa@hue.ac.jp
Department for Information Systems in Business
Faculty of Economics
Hiroshima University of Economics
Hiroshima, 731-0192, Japan

Tomohiro Hayashida is currently Assistant Professor at Department of Artificial Complex Systems Engineering, Graduate School of Engineering of the Hiroshima University, Japan. His current research interests are agent-based simulation analysis, game theory, and decision making theory.

e-mail: hayashida@hiroshima-u.ac.jp
Department of Artificial Complex Systems Engineering
Graduate School of Engineering
Hiroshima University
Higashi-Hiroshima, 739-8527, Japan

Masatoshi Sakawa received B.Sc., M.Sc., and Ph.D. degrees in applied mathematics and physics at Kyoto University, in 1970, 1972, and 1975, respectively. From 1975 he was with Kobe University where, since 1981, he was an Associate Professor in the Department of Systems Engineering. From l987 to 1990, he was a Professor of Department of Computer Science at Iwate University. At present he is a Professor of Hiroshima University, Japan, and is working with the Department of Artificial Complex Systems Engineering, Hiroshima University. His research and teaching activities are in the area of systems engineering, especially, mathematical optimization, multiobjective decision making, fuzzy mathematical programming and game theory.

e-mail: sakawa@hiroshima-u.ac.jp
Department of Artificial Complex Systems Engineering
Graduate School of Engineering
Hiroshima University
Higashi-Hiroshima, 739-8527, Japan