Trans. of the Society of Instrument and Control Engineers
Vol.E-6, No.1, 16/25 (2007)

Improvement of the Performances of Genetic Algorithms by Using Low-discrepancy Sequences

Shuhei KIMURA* and Koki MATSUMURA*

The random number generator is one of the important components of evolutionary algorithms. Therefore, when we try to solve function optimization problems using the evolutionary algorithms, we must carefully choose a good pseudo-random number generator. In the evolutionary algorithms, the pseudo-random number generator is often used for creating uniformly distributed individuals. In this study, as the low-discrepancy sequences allow us to create individuals more uniformly than the random number sequences, we apply the low-discrepancy sequence generator, instead of the pseudo-random number generator, to the evolutionary algorithms. Since it was difficult for some evolutionary algorithms, such as binary-coded genetic algorithms, to utilize the uniformity of the sequences, the low-discrepancy sequence generator was applied to real-coded genetic algorithms. The numerical experiments show that the low-discrepancy sequence generator improves the search performances of the real-coded genetic algorithms.

Key Words: genetic algorithm, real-coded GA, low-discrepancy sequence, random number generator, function optimization

1. Introduction

Evolutionary algorithms (EAs), such as genetic algorithms (GAs), are function optimizers inspired by the process of natural evolution. Because of their powerfulness and flexibility, a number of researchers have taken an interest in EAs. In order to enhance the probability of finding a reasonable solution with a low computational effort, a great number of EAs have been proposed [1), 5), 6), 11), 19)]. The random number generator is a component that most of these algorithms use. As it is difficult for computers to generate actual random numbers, EAs generally utilize pseudo-random numbers.

Although the random number generator is an essential component for EAs, researchers paid less attention to it. Recently, however, several studies have showed that the performances of EAs depend on the pseudo-random number generator applied [3), 4), 14), 15)]. Therefore, when we try to apply EAs to function optimization problems, we should use a pseudo-random number generator that has an ability to generate "good" pseudo-random numbers.

One of the goodness measures of random number sequences is uniformity. The most common measure of uniformity is discrepancy [22)]. For the point set $P_N = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ in $[0,1]^s$, the discrepancy is defined as

$$T_N^*(P_N) = \sqrt{\int_{[0,1]^s} \left[\frac{A(J,P_N)}{N} - V(J) \right]^2 d\mathbf{u}}, \qquad (1)$$

where $\mathbf{u} = (u_1, u_2, \cdots, u_s)$, J is a hyper-brick defined by $[\mathbf{0}, \mathbf{u}]$, $A(J,P_N)$ is the number of points landed inside J, and $V(J)$ is the volume of J. The discrepancy represents the averaged difference between the true volume of J and that estimated by a Monte Carlo method. This measure has been developed in the field of the Monte Carlo methods of numerical integration.

It is known that, when the discrepancy is used to measure the uniformities of sequences, a uniform random number sequence does not have the best uniformity. As the sequences that give the lower discrepancy than the uniform random number sequence, low-discrepancy sequences have been proposed [23)]. While the discrepancy of the uniform random number sequence is in the order of

$$\sqrt{\frac{\log\log N}{N}}, \qquad (2)$$

those of the low-discrepancy sequences are in the order of [16)]

$$\frac{(\log N)^s}{N}. \qquad (3)$$

The low-discrepancy sequences are less random, but have a better uniformity than the random number sequence.

In EAs, a pseudo-random number sequence is often used for creating new sampling points. Since we generally have no information about the search space before sampling, these sampling points should be created uniformly. However, the points generated by the pseudo-random number sequence have a worse uniformity than those generated by the low-discrepancy sequence. Therefore, when trying to create new sampling points in EAs, this study uses the low-discrepancy sequence generator, instead of the pseudo-random number generator. The use of the low-discrepancy sequence should make the search processes of EAs efficient. However, since the use of the low-discrepancy sequence would not improve all EAs, this study applies the low-discrepancy sequence generator into real-coded GAs [19)]. As

* Faculty of Engineering, Tottori University, 4-101 Koyama-Minami, Tottori

Table 1 Sample observation sites for the van der Corput sequences in base 2,3,4 and 5.

n	S_2	S_3	S_4	S_5
0	0.0000	0.0000	0.0000	0.0000
1	0.5000	0.3333	0.2500	0.2000
2	0.2500	0.6667	0.5000	0.4000
3	0.7500	0.1111	0.7500	0.6000
4	0.1250	0.4444	0.0625	0.8000
5	0.6250	0.7778	0.3125	0.0400
6	0.3750	0.2222	0.5625	0.2400
7	0.8750	0.5556	0.8125	0.4400
8	0.0625	0.8889	0.1250	0.6400
9	0.5625	0.0370	0.3750	0.8400
⋮	⋮	⋮	⋮	⋮

chromosomes of the real-coded GAs are vectors of real numbers, these GAs easily utilize the uniformity of the low-discrepancy sequence.

In the next section, we will present the low-discrepancy sequence used in this study. The section 3 will describe how to apply the low-discrepancy sequence generator into real-coded GAs. Then, in the section 4, we will verify the effectiveness of the use of the low-discrepancy sequence through numerical experiments on several benchmark problems. The sections 5 and 6 are the discussion and the conclusion, respectively.

2. Low-discrepancy Sequence

Although a number of low-discrepancy sequences have been proposed [7), 12), 17), 22), 23)], this study uses canonical one. The sequence used in this study is described below.

2.1 Van der Corput sequence

The van der Corput sequence is a one-dimensional low-discrepancy sequence [22)]. For an integer $b \geq 2$, the van der Corput sequence in base b is the sequence $S = \{t_0, t_1, t_2, \cdots\}$, where

$$t_n = \phi_b(n). \tag{4}$$

$\phi_b(n)$ is a radical inverse function, given by

$$\phi_b(n) = \sum_{j=0}^{\infty} \frac{a_j}{b^{j+1}}, \tag{5}$$

where a_j $(j = 0, 1, 2, \cdots)$ is a coefficient of a digit expansion of the integer n in base b, i.e.,

$$n = \sum_{j=0}^{\infty} a_j b^j. \tag{6}$$

Different values of base b provide us with different van der Corput sequences. **Table 1** shows the first ten observation sites for four van der Corput sequences in base 2, 3, 4 and 5, respectively.

2.2 Halton sequence

The van der Corput sequence described above has an ability to generate points that are uniformly distributed only in a one-dimensional space. In order to generate points uniformly distributed in a multi-dimensional space, Halton proposed the Halton sequence [7)].

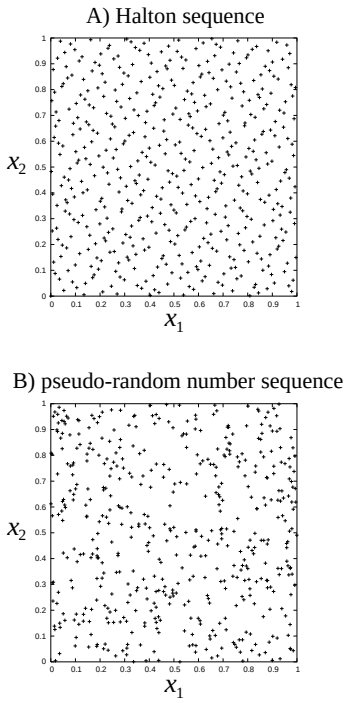


Fig. 1 2-D plots of 500 points generated by A) the Halton sequence, and B) the pseudo-random number sequence, respectively.

The Halton sequence is an extension of the van der Corput sequence to a multi-dimensional space. The s-dimensional Halton sequence is defined as $S_H = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots\}$, where

$$\mathbf{x}_n = \left(\phi_{b_1}(n), \phi_{b_2}(n), \cdots, \phi_{b_s}(n)\right). \tag{7}$$

$b_1, b_2, \cdots, b_s$ are integers that are greater than one and pairwise prime. In practice, they are often chosen to be the first s prime numbers. This sequence enables us to obtain points uniformly distributed in a multi-dimensional space. **Fig. 1** shows 2-dimensional plots of 500 points generated by the Halton sequence and the pseudo-random number sequence (Mersenne Twister). As shown in the figure, the points generated by the Halton sequence seem to have a better uniformity.

2.3 Digit-scrambling

As the Halton sequence is deterministic, it is difficult to use statistical techniques for analyzing it. This nature is inconvenient when we try to compare the performances of GAs with and without the Halton sequence. In order to introduce randomness into the Halton sequence, this study utilizes a digit-scrambling [12)]. Even when the scrambling is applied, the sequence still possesses a good uniformity [22)].

When we apply the digit-scrambling into the Halton sequence, the n-th point in the sequence is $\mathbf{x}_n = (\phi'_{b_1}(n), \phi'_{b_2}(n), \cdots, \phi'_{b_s}(n))$,

where

$$\phi'_{b_i}(n) = \sum_{j=0}^{J_{max}} \frac{\pi_j^{(i)}(a_j)}{b_i^{j+1}} + \sum_{j=J_{max}+1}^{\infty} \frac{a_j}{b_i^{j+1}}, \quad (8)$$

$\pi_j^{(i)}$ is a randomly generated permutation of $\{0,1,\cdots,b_i-1\}$, and J_{max} is a positive integer.

This study applies the low-discrepancy sequence described here into several real-coded GAs.

3. Application of Low-discrepancy Sequences to EAs

In order to confirm whether the use of the low-discrepancy sequence improves GAs, this study applies the low-discrepancy sequence described in the previous section into several real-coded GAs. We should note here that, as mentioned in the section 1, we use the low-discrepancy sequence only for generating new sampling points (individuals). Therefore, it is insufficient even when we simply substitute the low-discrepancy sequence generator for the random number generator in EAs. In this section, through the application of the low-discrepancy sequence into the real-coded GA used in this study, we will describe how to apply the low-discrepancy sequences into EAs.

This study uses MGG (a Minimal Generation Gap model) [21] as a generation alternation model of our GAs because it is relatively simple. The followings are an algorithm of MGG. In the algorithm described below, a recombination operator requires $m \geq 2$ parents to generate offsprings.

[Algorithm: MGG]

(1) **Initialization**

As an initial population, create n_p individuals. Set *Generation* $= 0$.

(2) **Selection for reproduction**

Select m individuals without replacement from the population. The selected individuals, that are expressed here as $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$, are used as the parents for the recombination operator in the next step.

(3) **Generation of offsprings**

Generate n_c children by applying some recombination operator to the parents selected in the previous step.

(4) **Selection for survival**

Select two individuals from the family containing the two parents ($\mathbf{p}_1$ and $\mathbf{p}_2$) and their children. One is the individual that has the best objective value, and the other is selected randomly. Then, replace the two parents with the selected individuals.

(5) **Termination**

Stop if the halting criteria are satisfied. Otherwise, *Generation* $\leftarrow$ *Generation* $+1$, and then return to the step 2.

As mentioned above, this study uses the low-discrepancy sequence only when trying to create new individuals. Therefore, we can apply it only to the steps 1 and 3 of MGG. Noting this point, the reminder of this section will describe each of the steps of MGG in greater detail.

3.1 Step: Initialization

As an initial population, create n_p individuals. In real-coded GAs, individuals are represented as s-dimensional real number vectors, where s is the dimension of the search space. To make the initial population uniformly distributed, we use the low-discrepancy sequence generator in this study.

3.2 Step: Selection for reproduction

Select m parents, $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$, randomly without replacement from the population. For the random selection of this step, the pseudo-random number generator is used.

3.3 Step: Generation of offsprings

Generate n_c children by applying a recombination operator into the parents selected in the previous step. As the recombination operator, this study uses any of ENDX (an Extended Normal Distribution Crossover) [9], UNDX (a Unimodal Normal Distribution Crossover) [18] and SPX (a Simplex Crossover) [8]. This section however describes the technique of applying the low-discrepancy sequence into ENDX, for example.

ENDX utilizes m parents, $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$, and generates a child $\mathbf{c}$ according to the following equation.

$$\mathbf{c} = \mathbf{p} + \xi \mathbf{d} + \sum_{i=3}^{m} \eta_i \mathbf{p}'_i, \quad (9)$$

where

$$\mathbf{p} = (\mathbf{p}_1 + \mathbf{p}_2)/2, \quad (10)$$

$$\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1, \quad (11)$$

$$\mathbf{p}'_i = \mathbf{p}_i - \frac{1}{m-2} \sum_{j=3}^{m} \mathbf{p}_j. \quad (12)$$

ξ and η_i are random numbers drawn from normal distributions $N(0,\alpha^2)$ and $N(0,\beta^2)$, respectively. This study uses the following recommended values for the parameters; $\alpha = 0.434$, $\beta = 0.35/\sqrt{m-3}$ and $m = s+2$.

In this study, ξ and η_i are generated using the low-discrepancy sequence generator, instead of using the pseudo-random number generator. In order to generate normally distributed numbers from uniformly distributed ones, this study uses the Box-Muller transformation (see Appendix A) [2]. Although we can also use the rejection method or the method based on the central limit theorem to generate normally distributed numbers, these methods may destroy the uniformity in the sequence.

3.4 Step: Selection for survival

Choose two individuals from the family that includes the two parents ($\mathbf{p}_1$ and $\mathbf{p}_2$) and their children. One is the individual that has the best objective value, and the other is selected randomly. Then, replace the two parents (i.e., $\mathbf{p}_1$ and $\mathbf{p}_2$) with the selected

Table 2 Benchmark functions.

Objective function		Search region	Optimum
Sphere	$f_{sp}(\mathbf{x}) = \sum_{i=1}^{s} x_i^2$	$-5.12 \le x_i \le 5.12$	$f_{sp}(0,\cdots,0) = 0$
Rosenbrock	$f_{ro}(\mathbf{x}) = \sum_{i=1}^{s-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$-2.048 \le x_i \le 2.048$	$f_{ro}(1,\cdots,1) = 0$
Rastrigin	$f_{ra}(\mathbf{x}) = 10s + \sum_{i=1}^{s}[x_i^2 - \cos(2\pi x_i)]$	$-5.12 \le x_i \le 5.12$	$f_{ra}(0,\cdots,0) = 0$
Wide & shifted Rastrigin	$f_{ws}(\mathbf{x}) = 10s + \sum_{i=1}^{s}[x_i^2 - \cos(2\pi x_i)]$	$-90 \le x_i \le 110$	$f_{ws}(0,\cdots,0) = 0$
Griewangk	$f_{gr}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{s} x_i^2 - \prod_{i=1}^{s} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-512 \le x_i \le 512$	$f_{gr}(0,\cdots,0) = 0$

individuals. We use the pseudo-random number generator for the random selection of this step.

4. Numerical Experiments

In order to confirm the effectiveness of the use of the low-discrepancy sequence, this section applies the real-coded GAs with and without the low-discrepancy sequence generator into several benchmark functions.

4.1 Objective functions

Five benchmark functions listed in **Table 2** were minimized in our experiments. The experiments were performed on 10, 15, and 20 dimensional functions (i.e., $s = 10, 15$ and 20). In each trial, we generated an initial population uniformly in the search region given in the table, and considered no explicit treatment of the search region during the search.

Sphere and Rosenbrock functions are unimodal. The Rosenbrock function is also non-separable since the optimum resides at the deep and curved valley. Rastrigin function is multimodal, and it has a number of local optima around a global optimum. Although the search region of the original Rastrigin function is $[-5.12, 5.12]^s$, we enlarged and shifted it to $[-90, 110]^s$ in a wide and shifted Rastrigin function. This treatment makes the optimization problem difficult. Griewangk function is also multimodal. However, this function is close to the Sphere function when the dimension of the search space is high. Therefore, the Griewangk function becomes easy as the dimension increases.

4.2 Experimental setup

We used three real-coded GAs, i.e., ENDX/MGG, UNDX/MGG and SPX/MGG, that use ENDX, UNDX (see Appendix B) [18] and SPX (see Appendix C) [8], respectively, as a recombination operator, and use MGG as a generation alternation model. In order to confirm the effectiveness of the use of the low-discrepancy sequence, the performances of the GAs with the low-discrepancy sequence were compared with those without the low-discrepancy sequence. This study used the scrambled Halton sequence described in the section 2.3 as a low-discrepancy sequence generator, and the Mersenne Twister [13] as the a pseudo-random number generator.

The population sizes of ENDX/MGG, UNDX/MGG and SPX/MGG were $n_p = 15s, 10s$ and $20s$, respectively. The number of the children generated by a recombination operator per selection n_c was 100 in all of the trials. We did not always use the recommended values for these parameters because the use of the recommended values made us difficult to confirm the differences between the performances of the GAs with and without the low-discrepancy sequence. For the other parameters of the GAs, we used their recommended values.

We performed 300 trials for the unimodal functions and 500 trials for the multimodal functions. Each trial was continued until the best fitness value reached less than 1.0×10^{-6}, the population was converged within the range of 1.0×10^{-6} in each coordinate, the number of function evaluation reached 1.0×10^9, or the best fitness value did not improve for 2.0×10^7 function evaluations. The optimum was considered to be found only when the best fitness value reached less than 1.0×10^{-6}.

4.3 Results

The experimental results of the GAs with the low-discrepancy sequence (with LDS) and those without the low-discrepancy sequence (w/o LDS) are shown in **Table 3**. The performances were compared using two standards, the number of trials where the algorithm succeeds in finding an optimum (SUC) and the average number of function evaluations required for finding an optimum (AVG). The standard deviations of the number of function evaluations required (SD) are also shown in the table.

Except for the trials where SPX/MGG was applied to the higher-dimensional Rosenbrock functions, the GAs succeeded in finding the optimum solutions in the unimodal functions. The GAs with the low-discrepancy sequence optimized these functions with the smaller number of function evaluations than those without the low-discrepancy sequence. The difference of the number of function evaluations was statistically significant at the significance level $\alpha = 5\%$, except for the experiments where SPX/MGG was applied to the Sphere functions.

In most of the multimodal functions, on the other hand, the GAs with the low-discrepancy sequence found the optimum solutions with a higher probability than those without the low-discrepancy sequence. The difference of the probability was statistically significant at the significance level $\alpha = 5\%$ when the GAs were applied to the Rastrigin functions, except for the trials where UNDX/MGG was applied to the 20-dimensional function. In addition, the GAs with the low-discrepancy sequence required the smaller number of function evaluations for the optimization

Table 3 Summary of results.

Objective function	Dimension s	ENDX/MGG		UNDX/MGG		SPX/MGG	
		with LDS	w/o LDS	with LDS	w/o LDS	with LDS	w/o LDS
		SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD
Sphere	10	300/300 2.2674×10^5 $\pm 9.7978 \times 10^3$	300/300 2.2900×10^5 $\pm 8.9727 \times 10^3$	300/300 1.4677×10^5 $\pm 6.7206 \times 10^3$	300/300 1.4841×10^5 $\pm 6.7950 \times 10^3$	300/300 2.5733×10^5 $\pm 1.5567 \times 10^4$	300/300 2.5810×10^5 $\pm 1.4999 \times 10^4$
	15	300/300 4.0780×10^5 $\pm 1.2850 \times 10^4$	300/300 4.0958×10^5 $\pm 1.2864 \times 10^4$	300/300 2.7776×10^5 $\pm 8.6498 \times 10^3$	300/300 2.7912×10^5 $\pm 8.8160 \times 10^3$	300/300 4.8032×10^5 $\pm 1.9569 \times 10^4$	300/300 4.8197×10^5 $\pm 1.9578 \times 10^4$
	20	300/300 6.0907×10^5 $\pm 1.5674 \times 10^4$	300/300 6.1450×10^5 $\pm 1.5969 \times 10^4$	300/300 4.2934×10^5 $\pm 1.1957 \times 10^4$	300/300 4.3293×10^5 $\pm 1.3071 \times 10^4$	300/300 7.3653×10^5 $\pm 2.3307 \times 10^4$	300/300 7.3882×10^5 $\pm 2.2777 \times 10^4$
Rosenbrock	10	300/300 9.8427×10^5 $\pm 6.6814 \times 10^4$	300/300 1.0103×10^6 $\pm 8.3740 \times 10^4$	300/300 1.0395×10^6 $\pm 4.8351 \times 10^4$	300/300 1.0511×10^6 $\pm 5.4197 \times 10^4$	300/300 8.7382×10^5 $\pm 7.8563 \times 10^4$	300/300 8.9910×10^5 $\pm 8.9111 \times 10^4$
	15	300/300 3.2853×10^6 $\pm 2.2863 \times 10^5$	300/300 3.4639×10^6 $\pm 2.5642 \times 10^5$	300/300 3.2363×10^6 $\pm 1.0344 \times 10^5$	300/300 3.2818×10^6 $\pm 1.1180 \times 10^5$	3/300 6.6822×10^6 $\pm 9.0693 \times 10^5$	0/300 — —
	20	300/300 9.7032×10^6 $\pm 7.4792 \times 10^5$	300/300 1.0601×10^7 $\pm 1.0719 \times 10^6$	300/300 7.4567×10^6 $\pm 2.1838 \times 10^5$	300/300 7.5397×10^6 $\pm 2.2955 \times 10^5$	0/300 — —	0/300 — —
Rastrigin	10	451/500 5.5360×10^5 $\pm 6.1959 \times 10^4$	388/500 5.6773×10^5 $\pm 7.0424 \times 10^4$	407/500 4.9821×10^5 $\pm 8.7026 \times 10^4$	378/500 5.1056×10^5 $\pm 9.5134 \times 10^4$	409/500 4.8443×10^5 $\pm 3.0292 \times 10^4$	381/500 4.8543×10^5 $\pm 3.3277 \times 10^4$
	15	442/500 8.9325×10^5 $\pm 8.8003 \times 10^4$	359/500 9.3697×10^5 $\pm 1.0337 \times 10^5$	366/500 8.7400×10^5 $\pm 1.1868 \times 10^5$	339/500 9.0513×10^5 $\pm 1.3172 \times 10^5$	402/500 8.7912×10^5 $\pm 3.6783 \times 10^4$	353/500 8.8518×10^5 $\pm 3.9989 \times 10^4$
	20	453/500 1.2381×10^6 $\pm 1.0924 \times 10^5$	371/500 1.3021×10^6 $\pm 1.7820 \times 10^5$	370/500 1.2664×10^6 $\pm 1.3927 \times 10^5$	347/500 1.3160×10^6 $\pm 1.4448 \times 10^5$	402/500 1.3436×10^6 $\pm 5.0985 \times 10^4$	377/500 1.3469×10^6 $\pm 4.7696 \times 10^4$
Wide & shifted Rastrigin	10	175/500 7.2372×10^5 $\pm 9.1184 \times 10^4$	179/500 7.3512×10^5 $\pm 1.0132 \times 10^5$	331/500 5.9066×10^5 $\pm 9.2806 \times 10^4$	327/500 6.1706×10^5 $\pm 1.0293 \times 10^5$	304/500 5.9476×10^5 $\pm 3.7113 \times 10^4$	300/500 5.9648×10^5 $\pm 3.6087 \times 10^4$
	15	16/500 1.2868×10^6 $\pm 1.6509 \times 10^5$	14/500 1.2181×10^6 $\pm 1.4141 \times 10^5$	186/500 1.1009×10^6 $\pm 1.5911 \times 10^5$	172/500 1.1191×10^6 $\pm 1.5806 \times 10^5$	206/500 1.0728×10^6 $\pm 4.5707 \times 10^4$	171/500 1.0729×10^6 $\pm 4.0789 \times 10^4$
	20	1/500 1.8751×10^6 $\pm 0.0000 \times 10^0$	0/500 — —	89/500 1.6741×10^6 $\pm 2.2611 \times 10^5$	78/500 1.7691×10^6 $\pm 2.7190 \times 10^5$	132/500 1.6114×10^6 $\pm 4.5525 \times 10^4$	124/500 1.6236×10^6 $\pm 5.4931 \times 10^4$
Griewangk	10	361/500 5.8344×10^5 $\pm 7.8820 \times 10^4$	337/500 5.9290×10^5 $\pm 8.6214 \times 10^4$	351/500 4.5742×10^5 $\pm 9.8646 \times 10^4$	335/500 4.6706×10^5 $\pm 8.9377 \times 10^4$	419/500 4.7076×10^5 $\pm 2.6541 \times 10^4$	425/500 4.7514×10^5 $\pm 2.8373 \times 10^4$
	15	487/500 6.7858×10^5 $\pm 4.3654 \times 10^4$	486/500 6.9400×10^5 $\pm 6.0286 \times 10^4$	495/500 4.8974×10^5 $\pm 2.1687 \times 10^4$	492/500 4.9386×10^5 $\pm 2.3570 \times 10^4$	499/500 7.0786×10^5 $\pm 2.4850 \times 10^4$	497/500 7.0972×10^5 $\pm 2.5102 \times 10^4$
	20	491/500 9.2828×10^5 $\pm 7.0790 \times 10^4$	489/500 9.4666×10^5 $\pm 7.1118 \times 10^4$	500/500 7.3605×10^5 $\pm 2.3879 \times 10^4$	496/500 7.4326×10^5 $\pm 2.4426 \times 10^4$	500/500 1.0471×10^6 $\pm 2.8843 \times 10^4$	500/500 1.0479×10^6 $\pm 2.6021 \times 10^4$

in most of the multimodal functions.

The experimental results therefore indicate that the application of the low-discrepancy sequence into real-coded GAs decreases the number of function evaluations required and increases a probability of finding an optimum solution. Note here that this section compared the performances of the GAs with and without the low-discrepancy sequence only. These GAs were not always use the recommended values for their parameters. Thus, we cannot use the experimental results to compare the performances of the GAs that utilize the different recombination operators from each other.

5. Discussion

5. 1 Steps where low-discrepancy sequences are applied

In the previous section, we showed that the use of the low-discrepancy sequence has an ability to improve the search performances of real-coded GAs. As described in the section 3, on

Table 5 Performances of the 4 GAs mentioned in the section 5. 1.

Objective function	Dimension s	ENDX/MGG (both)	ENDX/MGG (none)	ENDX/MGG (init)	ENDX/MGG (xover)
		SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD
Rosenbrock	10	300/300 9.8427×10^5 $\pm 6.6814 \times 10^4$	300/300 1.0103×10^6 $\pm 8.3740 \times 10^4$	300/300 1.0052×10^6 $\pm 7.8834 \times 10^4$	300/300 9.6852×10^5 $\pm 7.5416 \times 10^4$
	15	300/300 3.2853×10^6 $\pm 2.2863 \times 10^5$	300/300 3.4639×10^6 $\pm 2.5642 \times 10^5$	300/300 3.4570×10^6 $\pm 2.3935 \times 10^5$	300/300 3.2952×10^6 $\pm 2.2206 \times 10^5$
	20	300/300 9.7032×10^6 $\pm 7.4792 \times 10^5$	300/300 1.0601×10^7 $\pm 1.0719 \times 10^6$	300/300 1.0595×10^7 $\pm 9.6709 \times 10^5$	300/300 9.6765×10^6 $\pm 8.6192 \times 10^5$
Rastrigin	10	451/500 5.5360×10^5 $\pm 6.1959 \times 10^4$	388/500 5.6773×10^5 $\pm 7.0424 \times 10^4$	455/500 5.5970×10^5 $\pm 6.3677 \times 10^4$	395/500 5.7061×10^5 $\pm 7.2881 \times 10^4$
	15	442/500 8.9325×10^5 $\pm 8.8003 \times 10^4$	359/500 9.3697×10^5 $\pm 1.0337 \times 10^5$	443/500 9.0542×10^5 $\pm 9.4941 \times 10^4$	374/500 9.3388×10^5 $\pm 1.0421 \times 10^5$
	20	453/500 1.2381×10^6 $\pm 1.0924 \times 10^5$	371/500 1.3021×10^6 $\pm 1.7820 \times 10^5$	445/500 1.2515×10^6 $\pm 1.2454 \times 10^5$	361/500 1.2857×10^6 $\pm 1.2744 \times 10^5$

Table 4 GAs used in the section 5. 1. The low-discrepancy sequence generator (LDS) and the pseudo-random number generator (PRN) are differently applied to the steps of ENDX /MGG.

GAs	step 1 Initialization	step 3 Gen. of off.
ENDX /MGG (both)	LDS	LDS
ENDX /MGG (none)	PRN	PRN
ENDX /MGG (init)	LDS	PRN
ENDX /MGG (xover)	PRN	LDS

the other hand, we applied the low-discrepancy sequence generator into the two steps of the GAs, i.e., the "Initialization" step (step 1) and the "Generation of offsprings" step (step 3). This section investigates the effect that the application of the the low-discrepancy sequence into these steps has.

We applied four real-coded GAs listed in **Table 4** into the Rosenbrock and the Rastrigin functions. Although all of the GAs used here utilized ENDX as a recombination operator and MGG as a generation alternation model, they differently applied the low-discrepancy sequence generator and the pseudo-random number generator into the two steps. ENDX/MGG (both) and ENDX/MGG (none) are the GAs that use the low-discrepancy sequence generator and the pseudo-random number generator, respectively, for generating new individuals. ENDX/MGG (init) and ENDX/MGG (xover), on the other hand, apply the low-discrepancy sequence generator only to the "Initialization" step and the "Generation of offsprings" step, respectively. All of the other experimental conditions were the same as those used in the previous section.

Table 5 summarizes the experimental results for the four GAs. When we compared the GAs with respect to the number of function evaluations required for optimizing the Rosenbrock functions, no statistical difference between ENDX/MGG (none) and ENDX/MGG (init) was found at the significance level $\alpha = 1\%$. We could not also find any notable difference between ENDX/MGG (both) and ENDX/MGG (xover) in the 15 and 20 dimensional Rosenbrock functions (the significance level $\alpha = 1\%$). These results indicate that the application of the low-discrepancy sequence into the "Generation of offsprings" step decreases the number of function evaluations required for the optimization. The application of the low-discrepancy sequence into the "Initialization" step, on the other hand, may not improve the search performances of the GAs in the Rosenbrock function.

In the Rastrigin functions, the performances of ENDX/MGG (both) and ENDX/MGG (none) were almost the same as those of ENDX/MGG (init) and ENDX/MGG (xover), respectively, with respect to the probability of finding an optimum solution (the significance level $\alpha = 1\%$). The application of the low-discrepancy sequence into the "Initialization" step, therefore, may enhance a probability of finding an optimum in multimodal functions. On the contrary, even if the low-discrepancy sequence was applied to the "Generation of offsprings" step, a probability of finding an optimum was not improved in multimodal functions.

5. 2 Scrambling techniques

As described in the section 2. 3, this study uses the digit-scrambling in order to introduce randomness into the Halton sequence. However, some other techniques have been proposed for the same purpose [12), 22), 23)]. This section shows that the other techniques are not always appropriate for the application into EAs.

Table 6 Performances of ENDX/MGGs using the digit-scrambled, the random linear scrambled, and the random-start Halton sequences, respectively.

Objective function	Dimension s	Digit-scrambling		Random linear scrambling		Random-start	
		b: small	b: large	b: small	b: large	b: small	b: large
		SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD	SUC AVG STD
Rosenbrock	10	300/300 9.8427×10^5 $\pm6.6814\times10^4$	300/300 1.0036×10^6 $\pm8.6969\times10^4$	300/300 9.7380×10^5 $\pm7.8254\times10^4$	300/300 1.0293×10^6 $\pm3.1068\times10^5$	300/300 9.7361×10^5 $\pm6.8190\times10^4$	0/300 — —
	15	300/300 3.2853×10^6 $\pm2.2863\times10^5$	300/300 3.4626×10^6 $\pm2.4170\times10^5$	300/300 3.3177×10^6 $\pm2.4972\times10^5$	299/300 3.5301×10^6 $\pm5.9383\times10^5$	300/300 3.3113×10^6 $\pm2.4769\times10^5$	0/300 — —
	20	300/300 9.7032×10^6 $\pm7.4792\times10^5$	300/300 1.0673×10^7 $\pm1.0610\times10^6$	300/300 9.7074×10^6 $\pm9.6406\times10^5$	294/300 1.0844×10^7 $\pm3.6527\times10^6$	300/300 9.3812×10^6 $\pm7.6102\times10^5$	0/300 — —
Rastrigin	10	451/500 5.5360×10^5 $\pm6.1959\times10^4$	419/500 5.6453×10^5 $\pm6.6153\times10^4$	448/500 5.5189×10^5 $\pm5.7641\times10^4$	426/500 5.5680×10^5 $\pm5.7871\times10^4$	459/500 5.5802×10^5 $\pm6.6538\times10^4$	0/500 — —
	15	442/500 8.9325×10^5 $\pm8.8003\times10^4$	380/500 9.1908×10^5 $\pm1.0589\times10^5$	454/500 8.9508×10^5 $\pm8.2201\times10^4$	401/500 9.0596×10^5 $\pm1.0625\times10^5$	434/500 9.0318×10^5 $\pm9.2358\times10^4$	0/500 — —
	20	453/500 1.2381×10^6 $\pm1.0924\times10^5$	409/500 1.2728×10^6 $\pm1.3010\times10^5$	463/500 1.2399×10^6 $\pm1.0618\times10^5$	415/500 1.2432×10^6 $\pm9.5050\times10^4$	439/500 1.2759×10^6 $\pm1.3851\times10^5$	0/500 — —

In order to introduce randomness into the Halton sequence, this section uses three techniques, i.e., the digit-scrambling described in the section 2. 3, the random linear scrambling [22] and the random-start Halton sequence [23]. The random linear scrambling uses

$$\pi_j^{(i)}(\mathbf{a}) = \sum_{k=0}^{j} M_{jk}^{(i)} a_j + c_j^{(i)} \pmod{b_i}, \tag{13}$$

as $\pi_j^{(i)}$ given in the equation (8), where $M_{jk}^{(i)}$ and $c_j^{(i)}$ are integers randomly selected from $\{0,1,\cdots,b_i-1\}$. The random-start Halton sequence is, on the other hand, equivalent to the sequence $S_{rH} = \{\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \cdots\}$, where

$$\mathbf{y}_n = \left(\phi_{b_1}(n+m_1), \phi_{b_2}(n+m_2), \cdots, \phi_{b_s}(n+m_s)\right).$$

$m_1, m_2, \cdots, m_s$ ($m_i \geq 0$) are constant integers randomly selected.

In this section, we applied the Halton sequences with the different scrambling techniques to ENDX/MGG, and compared their performances. As mentioned in the section 2. 2, when we try to utilize the s-dimensional Halton sequence, the first s prime numbers are generally used as its bases b. In addition to using the Halton sequences with these prime numbers, we also used those with the first s prime numbers larger than 547 (the 101-th prime number) in this section. This experiment simulates the situation in which we try to solve higher-dimensional problems. All of the other experimental conditions were the same as those used in the section 4.

The experimental results are summarized in **Table 6**. When we applied the Halton sequence with the small bases into ENDX/MGG (b: small), its search performance was independent of the scrambling technique applied. The performance of ENDX/MGG, on the other hand, depended on the scrambling technique when the large prime numbers were used (b: large). Especially when we used the random-start Halton sequence with the large bases, the GA failed in optimizing the functions for all of the trials. Real-coded GAs generally create not more than hundreds of individuals at a time. When the random-start Halton sequence with the large bases is used to generate several hundreds of points, they are distributed in the biased area (**Fig. 2**). Since the GA using the random-start Halton sequence with the large bases could not create sampling points uniformly, it should fail in finding an optimum solution.

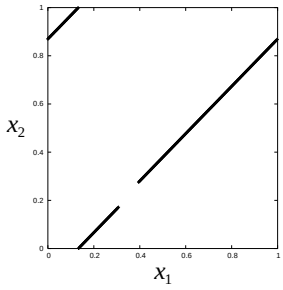


Fig. 2 2-D plot of 500 points generated by the random-start Halton sequence of large bases.

The performance of the GA using the digit-scrambled Halton sequence was not worse than that using the pseudo-random number sequence, even when the large prime numbers are used as its bases. As points generated by the digit-scrambled Hal-

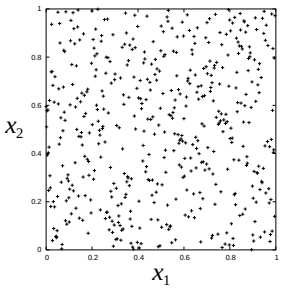


Fig. 3 2-D plot of 500 points generated by the digit-scrambled Halton sequence of large bases.

ton sequence are similar to those of the pseudo-random number sequence (**Fig. 3**), the performances of the GAs would resemble each other. The GA using the random linear scrambled Halton sequence with the large bases, on the other hand, sometimes failed in the optimization, even when it was applied to the unimodal function. The reason for the failure in the optimization is that the random linear scrambled Halton sequence with large bases often fails in generating points uniformly. The digit-scrambled Halton sequence with large bases scarcely creates points distributed in the biased area.

The experimental results indicate that, even when we try to apply GAs into high-dimensional functions, the use of the digit-scrambled Halton sequence should improve their search performance. The use of other scrambling techniques, on the other hand, may not be suitable for the optimization of high-dimensional functions.

6. Conclusion

This study applied the low-discrepancy sequence, instead of the pseudo-random number sequence, into the real-coded GAs. The experimental results showed that the use of the low-discrepancy sequence has an ability to enhance the search performances of the real-coded GAs. Through the additional experiments, we found that the application of the low-discrepancy sequence into the "Initialization" step enhances the probability of finding an optimum solution and the application of the low-discrepancy sequence into "Generation of offsprings" step decreases the number of function evaluations required for the optimization. We then showed that, even when we try to apply GAs into high-dimensional functions, the use of the digit-scrambled Halton sequence should improve their search performance. Even when we applied the low-discrepancy sequence to the GAs, the improvement of the performances was slight. However, since the application of the low-discrepancy sequence requires no modification on GAs, it should be a useful technique.

As described in the section 1, several studies have showed that the performances of GAs depend on the pseudo-random number sequence applied [3), 4), 14), 15)]. However, we do not know what nature of the sequences improves the performances of GAs. Moreover, Wiese and his colleagues have reported that the behavior and performance of the random number sequence in EAs are dependent on the different characteristics of the problems [24)]. On the other hand, this study improved the performances of the GAs using the low-discrepancy sequence. The low-discrepancy sequence is less random, but has a better uniformity than the random number sequence. The originality of this work is to utilize the uniformity of the low-discrepancy sequence for EAs.

The low-discrepancy sequences have been applied into EAs in few studies [20)]. In this study, we applied the low-discrepancy sequence only to the simple real-coded GAs. Therefore, we should confirm whether the low-discrepancy sequences improve the search performances of more complicated EAs. In addition, as this study only used the Halton sequence as the low-discrepancy sequence, we should test EAs that apply other low-discrepancy sequences.

References

1) T. Bäck, U. Hammel and H.P. Schwefel: Evolutionary Computation: Comments on the History and Current State, IEEE Trans. on Evolutionary Computation, **1**-1, 3/17 (1997)
2) G.E.P. Box and M.E. Muller: A Note on the Generation of Random Normal Deviates. Ann. Math. Stat., **29**, 610/611 (1958)
3) E. Cantú-Paz: On Random Numbers and the Performance of Genetic Algorithms, Proc. of Genetic and Evolutionary Computation Conference (GECCO) 2002, 754/761 (2002)
4) J.M. Daida, D.S. Ampy, M. Ratanasavetavadhana, H. Li and O.A. Chaudhri: Challenges with Verification, Repeatability, and Meaningful Comparison in Genetic Programming: Gibson's Magic, Proc. of GECCO 1999, 1851/1858 (1999)
5) K. Deb, D. Joshi and A. Anand: Real-coded Evolutionary Algorithms with Parent-Centric Recombination, Proc. of Congress on Evolutionary Computation (CEC) 2002, 61/66 (2002)
6) L.J. Eshelman and J.D. Schaffer: Real-coded Genetic Algorithms and Interval-Schemata, Proc. of Foundations of Genetic Algorithms (FOGA) 2, 187/202 (1993)
7) J.H. Halton: On the Efficiency of Certain Quasi-Random Sequences of Points in Evaluating Multi-Dimensional Integrals, Numerische Mathematik, **2**, 84/90 (1960)
8) T. Higuchi, S. Tsutsui and M. Yamamura: Simplex Crossover for Real-coded Genetic Algorithms, Trans. of the Japanese Society for Artificial Intelligence (JSAI), **16**-1, 147/155 (2001, in Japanese)
9) S. Kimura, I. Ono, H. Kita and S. Kobayashi: An Extension of UNDX based on Guidelines for Designing Crossover Operators: Proposition and Evaluation of ENDX, Trans. of the Society of Instrument and Control Engineers (SICE), **36**-12, 1162/1171 (2000, in Japanese)
10) S. Kimura and K. Matsumura: Genetic Algorithms using Low-Discrepancy Sequences, Proc. of GECCO 2005, 1341/1346 (2005)
11) H. Kita, I. Ono and S. Kobayashi: Multi-parental Extension of the Unimodal Normal Distribution Crossover for Real-coded Genetic Algorithms, Proc. of CEC 1999, 1581/1588 (1999)
12) J. Matoušek: Geometric Discrepancy: An Illustrated Guide, Springer (1999)
13) M. Matsumoto and T. Nishimura: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudorandom Number Generator, ACM Trans. on Modeling and Computer Simulation, **8**-

1, 3/30 (1998)
14) M.M. Meysenburg and J.A. Foster: Randomness and GA Performance, Revisited, Proc. of GECCO 1999, 425/432 (1999)
15) M.M. Meysenburg and J.A. Foster: Random Generator Quality and GP Performance, Proc. of GECCO 1999, 1121/1126 (1999)
16) W.J. Morokoff and R.E. Caflisch: Quasi-random sequences and their discrepancies, SIAM J. on Scientific Computing, **15**-6, 1251/1279 (1994)
17) H. Niederreiter: Random Number Generation and Quasi-Monte Carlo Methods, SIAM (1992)
18) I. Ono, H. Satoh and S. Kobayashi: A Real-coded Genetic Algorithm for Function Optimization Using the Unimodal Normal Distribution Crossover, J. of the JSAI, **14**-6, 1146/1155 (1999, in Japanese)
19) I. Ono, M. Yamamura and H. Kita: Real-coded Genetic Algorithms and Their Applications, J. of the JSAI, **15**-2, 259/266 (2000, in Japanese)
20) I.C. Parmee and C.R. Bonham: Improving Cluster Oriented Genetic Algorithms for High-performance Region Identification, Proceedings US United Engineering Foundation's 'Optimization in Industry' Conference, 14p (2001)
21) H. Satoh, I. Ono and S. Kobayashi: A New Generation Alteration Model of Genetic Algorithms and Its Assessment, Trans. of the JSAI, **12**-5, 734/744 (1997, in Japanese)
22) J. Wang, M. Taguri, S. Tezuka, Y. Kabashima and N. Ueda: Computational Statistics I: New Methods for the Computation of Probability, Iwanami Shoten (2003, in Japanese)
23) X. Wang and F.J. Hickernell: Randomized Halton Sequences, Mathematical and Computer Modelling, **32**, 887/899 (2000)
24) K.C. Wiese, A. Hendriks, A. Deschênes and B.B. Youssef: The Impact of Pseudorandom Number Quality on P-RnaPredict, a Parallel Genetic Algorithm for RNA Secondary Structure Prediction, Proc. of GECCO 2005, 479/480 (2005)

Appendix A. Box-Muller Transformation

The Box-Muller transformation is a method of generating pairs of independent normally distributed random numbers, given a source of uniformly distributed random numbers [2]. If x_1 and x_2 are uniformly and independently distributed between 0 and 1, then z_1 and z_2 defined below follow a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$.

$$z_1 = \sqrt{-2\ln(x_1)}\cos(2\pi x_2), \quad \text{(A. 1)}$$

$$z_2 = \sqrt{-2\ln(x_1)}\sin(2\pi x_2). \quad \text{(A. 2)}$$

When the low-discrepancy sequences are used, the Box-Muller transformation seems to give us "better" normally distributed points (**Fig.A.1**).

Appendix B. Application of Low-discrepancy Sequences into UNDX

UNDX (Unimodal Normal Distribution Crossover) [18] is a recombination operator that requires three parents ($\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$), and it generates offsprings according to the following equation.

$$\mathbf{c} = \mathbf{p} + \xi\mathbf{d} + D\sum_{i=1}^{s-1}\eta_i\mathbf{e}_i. \quad \text{(B. 1)}$$

where

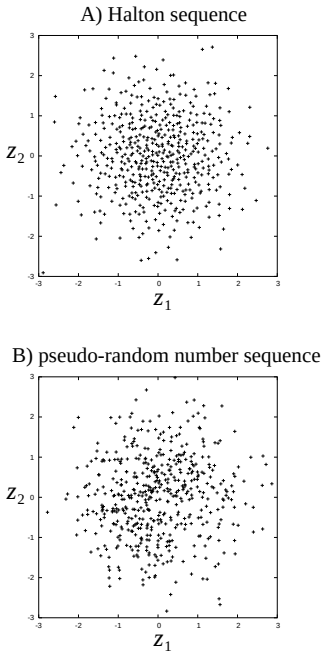


Fig. A.1 2-D plots of 500 normally distributed points generated by A) the Halton sequence, and B) the pseudo-random number sequence, respectively.

$$\mathbf{p} = (\mathbf{p}_1 + \mathbf{p}_2)/2, \quad \text{(B. 2)}$$

$$\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1, \quad \text{(B. 3)}$$

$$D = \sqrt{|\mathbf{p}_3 - \mathbf{p}_1|^2 - \frac{[\mathbf{d}\cdot(\mathbf{p}_3 - \mathbf{p}_1)]^2}{|\mathbf{d}|^2}}, \quad \text{(B. 4)}$$

$$\xi \sim N(0, \alpha^2), \quad \text{(B. 5)}$$

$$\eta_i \sim N(0, \beta^2). \quad \text{(B. 6)}$$

The vectors $\mathbf{e}_i$ ($i = 1, 2, \cdots, s-1$) are normalized orthogonal bases that span the subspace orthogonal to the vector $\mathbf{d}$. $\alpha = 0.5$ and $\beta = 0.35/\sqrt{s}$ are recommended, where s is the dimension of the search space.

When we try to apply the low-discrepancy sequence into UNDX, we use the numbers ξ and η_i generated from the low-discrepancy sequence.

Appendix C. Application of Low-discrepancy Sequences into SPX

SPX (Simplex Crossover) [8] is a multi-parental extension of BLX-α [6]. SPX requires m parents ($\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$) and generates offsprings according to the following procedure.

(1) Let the center of mass of the parents be $\mathbf{g} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{p}_i$.

(2) Compute $\mathbf{x}^1 = \mathbf{g} + \alpha(\mathbf{p}_1 - \mathbf{g})$.

(3) Compute $\mathbf{x}^k$ and $\mathbf{c}^k$ ($k = 2, 3, \cdots, m$) according to the following equations.

$$\mathbf{x}^k = \mathbf{g} + \alpha(\mathbf{p}_k - \mathbf{g}), \quad \text{(C. 1)}$$

$$\mathbf{c}^k = r_{k-1}(\mathbf{x}^{k-1} - \mathbf{x}^k + \mathbf{c}^{k-1}), \quad \text{(C. 2)}$$

where $\mathbf{c}^1 = \mathbf{0}$, and α is a constant parameter. r_k is a random number given by

$$r_k = u_k^{\frac{1}{k}}, \tag{C. 3}$$

where u_k is a uniform random number in $[0,1]$.

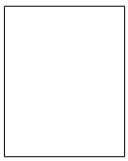
(4) Generate a offspring according to the following equation.

$$\mathbf{c} = \mathbf{x}_m + \mathbf{c}^m. \tag{C. 4}$$

As the parameters of SPX, $\alpha = \sqrt{s+2}$ and $m = s+1$ are recommended.

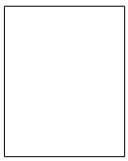
When we try to apply the low-discrepancy sequence into SPX, we simply substitute u_k generated by the low-discrepancy sequence generator for that generated by the pseudo-random number generator.

Shuhei KIMURA (Member)



Shuhei Kimura received his M.E. degree from Kyoto Univ. in 1998 and Ph.D degree from Tokyo Institute of Tech. in 2001. He had been in RIKEN Genomic Sciences Center as a research scientist since 2001. Since 2004, he has been in Tottori Univ. as an associate professor. His current research interests are evolutionary algorithms and bioinformatics.

Koki MATSUMURA



Koki Matsumura received his Ph.D degree from Osaka City Univ. in 1978. He had been in Ibaraki Univ. as an associate professor, and Konan Univ. as a professor. Since 2002, he has been in Tottori Univ. as a professor. His current research interests are evolutionary algorithms, knowledge engineering, study of management information and financial engineering.

Reprinted from Trans. of the SICE
Vol. 42 No. 6 659/667 2006