Genetic Algorithms using Low-Discrepancy Sequences

Shuhei Kimura
Department of Information and Knowledge Engineering,
Faculty of Engineering, Tottori University
4-101, Koyama Minami, Tottori, JAPAN
kimura@ike.tottori-u.ac.jp

Koki Matsumura
Department of Information and Knowledge Engineering,
Faculty of Engineering, Tottori University
4-101, Koyama Minami, Tottori, JAPAN
matumura@ike.tottori-u.ac.jp

ABSTRACT

The random number generator is one of the important components of evolutionary algorithms (EAs). Therefore, when we try to solve function optimization problems using EAs, we must carefully choose a good pseudo-random number generator. In EAs, the pseudo-random number generator is often used for creating uniformly distributed individuals. 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 EAs in this study. The numerical experiments show that the low-discrepancy sequence generator improves the search performances of EAs.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Genetic algorithm, Random number generator, Pseudo-random number sequence, Low-discrepancy sequence

1. INTRODUCTION

The random number generator is a basic component of evolutionary algorithms (EAs) as they are the stochastic search algorithms for function optimization problems. Recently, several studies showed that the performances of EAs can be affected by the choice of the pseudo-random number generator [3, 4, 11, 12]. In general, when we try to design new EAs, we suppose that "good" random number sequences are available for them. Therefore, when EAs are applied to the function optimization problems, we should choose a "good" pseudo-random number generator, such as the Mersenne Twister [10].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'05, June 25–29, 2005, Washington, DC, USA.
Copyright 2005 ACM 1-59593-010-8/05/0006 ...$5.00.

One of the goodness measures of random number sequences is uniformity. The most common measure of uniformity is discrepancy (see, e.g., [19]). 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 $\mathbf{u} = (u_1, \cdots, u_s)$, J is a hyper-brick defined by $[0, u_i]$ $(i = 1, 2, \cdots, s)$, $A(J, P_N)$ is the number of points landed inside J, and $V(J)$ is the volume of J.

It was shown that uniform random number sequences have the discrepancy in the order of

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

and they do not have the lowest discrepancy (see, e.g., [13]). As the sequences that give the lower discrepancy than the uniform random number sequences, the low-discrepancy sequences have been developed [7, 14, 19]. These sequences are also called quasi-random or sub-random sequences, and they have the discrepancy in the order of

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

The low-discrepancy sequences are less random, but more uniform than the random number sequences.

In EAs, pseudo-random number sequences are often used for creating new sampling points. These sampling points should be created uniformly, since we generally have no information about the search space before the sampling. However, the randomly generated points are less uniform than those generated by the low-discrepancy sequences. Therefore, in this study, we use the low-discrepancy sequence generator, instead of the pseudo-random number generator, to create new individuals in EAs. As it was difficult for binary-coded genetic algorithms (GAs) to utilize the uniformity of the sequences, the low-discrepancy sequence generator was applied to a real-coded GA (see, e.g., [6]). The effectiveness of the use of the low-discrepancy sequence was verified through numerical experiments on several benchmark problems.

2. LOW-DISCREPANCY SEQUENCES

Many low-discrepancy sequences have been proposed [7, 14, 19]. The sequence used in this study is described below.

2.1 Van Der Corput Sequence

The van der Corput sequence in base b is the one-dimensional low-discrepancy sequence and can be constructed as follows (see, e.g., [14]); for an integer $b \geq 2$, we put $Z_b = \{0, 1, \cdots, b-1\}$, i.e.,

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
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Z_b is the least residue system modulo b. Every integer $n \geq 0$ has a unique digit expansion

$$n = \sum_{j=0}^{m} a_j b^j \tag{4}$$

in base b, where $a_j \in Z_b$ for all $j \geq 0$, and m is the integral part of $\log_b n$, i.e., $m = [\log_b n]$. Also, let $\phi_b(n)$ be the radical-inverse function in base b for an integer $b \geq 2$, where

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

for all integers $n \geq 0$. Thus, $\phi_b(n)$ is obtained from n by a symmetric reflection of the expansion (4) in the decimal point. Then, for an integer $b \geq 2$, the van der Corput sequence in base b is the sequence $S_b = \{t_0, t_1, t_2, \cdots\}$, where

$$t_n = \phi_b(n). \tag{6}$$

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 in the one-dimensional space. The sequence, however, does not generate points uniformly distributed in the higher-dimensional space. To overcome this drawback, Halton proposed the Halton sequence [7].

The Halton sequence is an extension of the van der Corput sequence to the higher-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. Figure 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 Halton sequence allows us to generate points uniformly distributed in the higher-dimensional space.

It is, however, difficult to use the statistical techniques for analyzing the Halton sequence, since it is deterministic. This nature is inconvenient when we try to compare the performances of GAs with and without the low-discrepancy sequence. Therefore,

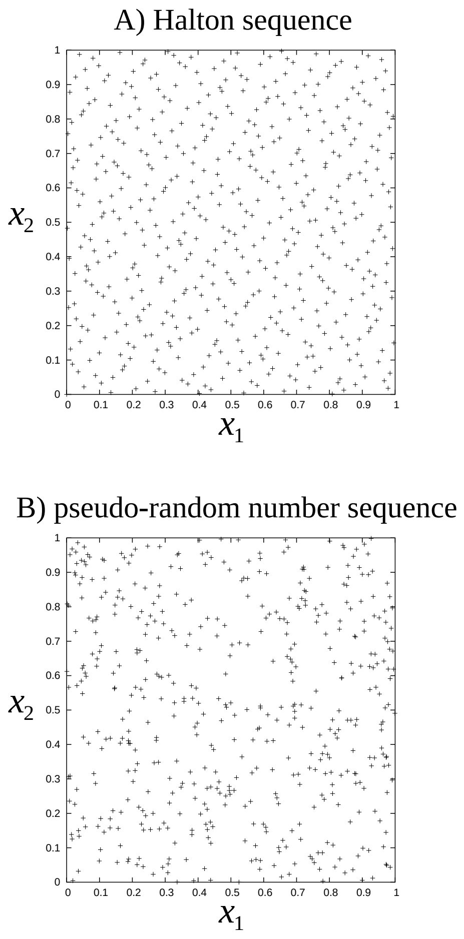

Figure 1: 2-D plots of 500 points generated by A)the Halton sequence, and B)the pseudo-random number sequence, respectively.

in this study, we use the random-start Halton sequence that introduces some randomness into the Halton sequence [19]. The random-start Halton sequence is equivalent to the sequence $S_{rH} = \{\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \cdots\}$, where

$$\mathbf{y}_n = \left(\phi_{b_1}(n+m_1), \phi_{b_2}(n+m_2), \cdots, \phi_{b_s}(n+m_s)\right), \tag{8}$$

and $m_1, m_2, \cdots, m_s$ ($m_i \geq 0$) are constant integers randomly selected. Even if the randomness is introduced, the sequence still possesses the uniformity. Readers can find more detailed information on this sequence in the paper written by Wang and Hickernell [19].

3. REAL-CODED GA USING LOW-DISCREPANCY SEQUENCE

In this section, we apply the low-discrepancy sequence generator described above into a GA. A real-coded GA is used in this study as it has an ability to easily utilize the uniformity of the sequences.

Many real-coded GAs have been proposed so far [5, 6, 9, 15, 18]. However, several real-coded GAs may be unsuitable for the use of the low-discrepancy sequences, since the treatment of some transformation disrupts the uniformity of the sequences. In this study, we use a relatively simple real-coded GA, ENDX /MGG,

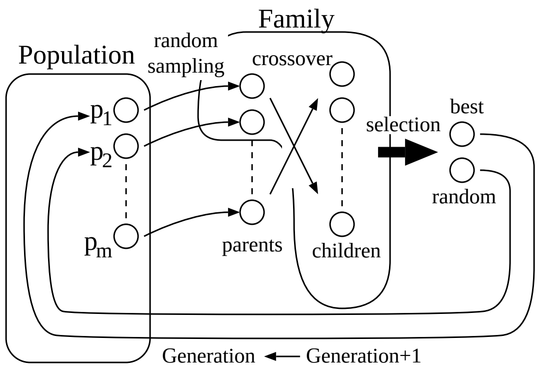

Figure 2: Framework of optimization process by ENDX /MGG.

because it requires no complicated transformation. ENDX /MGG uses ENDX (Extended Normal Distribution Crossover) [8] as a recombination operator, and MGG (Minimal Generation Gap model) [17] as a generation alternation model. The following is an algorithm of ENDX /MGG (see also Figure 2);

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

2. *Selection for reproduction*
   Select a pair of individuals randomly without replacing it from the population. The selected individuals are used as the parents for the recombination operator in the next step.

3. *Generation of offspring*
   Generate n_c children by applying the recombination operator, ENDX, to the selected pair of the individuals.

4. *Selection for survival*
   Select two individuals from the family containing the parents and their children. One has the best fitness value, and the other is selected randomly. Then, replace the parents with the selected individuals.

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

As the low-discrepancy sequence generator should be used to create individuals, we apply it only to the steps 1 and 3 in this study. To generate random numbers in the rest of the steps, the pseudo-random number generator, Mersenne Twister [10], is used. Each of the steps is described below in greater detail.

3.1 Step: Initialize

As an initial population, create n_p individuals. In real-coded GAs, individuals are represented as n-dimensional real number vectors, where n 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 from the population. As ENDX is the multi-parental recombination operator, m parents are selected here. For the random selection in this step, the pseudo-random number generator is used.

3.3 Step: Generation of Offspring

Generate n_c children by applying ENDX to the selected parents, $\mathbf{p}_1, \cdots, \mathbf{p}_m$. ENDX is a multi-parental extension of UNDX (Unimodal Normal Distribution Crossover) [15], and it generates children mainly along with the line connecting the two parents, $\mathbf{p}_1$ and $\mathbf{p}_2$, according to the following procedure;

1. Let the mid point of the parents $\mathbf{p}_1$ and $\mathbf{p}_2$ be $\mathbf{p} = (\mathbf{p}_1 + \mathbf{p}_2)/2$, and let the difference vector of these parents be $\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1$.

2. Let the mass center of the rest of the parents be
$$\mathbf{g} = \frac{1}{m-2}\sum_{j=3}^{m}\mathbf{p}_j,$$
   and let the difference vector of $\mathbf{p}_i$ and $\mathbf{g}$ be $\mathbf{q}_i = \mathbf{p}_i - \mathbf{g}$.

3. Generate a child $\mathbf{c}$ according to the following equation;
$$\mathbf{c} = \mathbf{p} + \xi\mathbf{d} + \sum_{i=3}^{m}\eta_i\mathbf{q}_i,$$
   where ξ and η_i are random variables that follow normal distributions $N(0,\alpha^2)$ and $N(0,\beta^2)$, respectively.

In this study, in order to generate the random variables, ξ and η_i, we use the low-discrepancy sequence generator. The Box-Muller transformation [2] (see also appendix) is used to generate normally distributed numbers from uniformly distributed ones, because the rejection method or the method based on the central limit theorem 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 has the best fitness value, and the other is selected randomly. Then, replace the parents (i.e., $\mathbf{p}_1$ and $\mathbf{p}_2$) with the selected individuals. We use the pseudo-random number generator in this step.

4. EXPERIMENTS

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

4.1 Benchmark Functions

Four benchmark functions were used in our experiments. These benchmark functions are given in Table 2. The experiments were performed on 10, 15, and 20 dimensional functions (i.e., $n = 10, 15$ and 20).

Rosenbrock function is unimodal. This function is non-separable since the optimum resides at the deep and curved valley. Rastrigin function is a multimodal one, but all of the local optima exist at equal intervals and these optima array parallel to the axes of the coordinate system. Hence, this function is well-scaled and separable. Griewangk function is also multimodal. However, this function is close to Sphere function when the dimension of the search space is high. Therefore, the Griewangk function becomes easy as the dimension increases. In order to compare the performances of the algorithms on another multimodal function, we used Ackley function. The Griewangk function and the Ackley function are also weakly non-separable.

Table 2: Benchmark functions.

	Objective function	Search region	Optimum
Rosenbrock	$f_{ro}(\mathbf{x}) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$-2.048 \leq x_i \leq 2.048$	$f_{ro}(1,\cdots,1) = 0$
Rastrigin	$f_{ra}(\mathbf{x}) = 10n + \sum_{i=1}^{n}[x_i^2 - \cos(2\pi x_i)]$	$-5.12 \leq x_i \leq 5.12$	$f_{sp}(0,\cdots,0) = 0$
Griewangk	$f_{gr}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-512 \leq x_i \leq 512$	$f_{gr}(0,\cdots,0) = 0$
Ackley	$f_{ac}(\mathbf{x}) = 20 + e$ $-20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n}\cos 2\pi x_i}{n}\right)$	$-20 \leq x_i \leq 30$	$f_{ac}(0,\cdots,0) = 0$

Table 3: GAs used in the experiments. 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 Initialize	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

4.2 Experimental Setup

We compared four GAs listed in Table 3. As described in the section 3, we applied the low-discrepancy sequence generator into the two steps of ENDX /MGG, i.e., the "Initialize" step and the "Generation of offspring" step. This GA is referred to as ENDX /MGG (both) in this study. We compared ENDX /MGG (both) with the original ENDX/MGG that uses no low-discrepancy sequence generator. This original ENDX /MGG is called ENDX /MGG (none) here. In addition, to study the effect of the use of the low-discrepancy sequence on the GA performance more precisely, we also carried out the experiments using ENDX /MGG (init) and ENDX /MGG (xover) that apply the low-discrepancy sequence generator only to the "Initialize" step and the "Generation of offspring" step, respectively. In all of the experiments, we used the Mersenne Twister [10] as the the pseudo-random number generator, and the random-start Halton sequence described in the section 2.2 as the low-discrepancy sequence generator.

We used the following GA parameters; the population size n_p was $15n$ where n is the dimension, and the number of the children generated by the crossover per selection n_c was 100. The recommended parameters in ENDX were used here; the number of parents m is $n+2$, and the standard deviations of random numbers α and β are 0.434 and $0.35/\sqrt{m-3}$, respectively.

100 runs were carried out for each benchmark function. Each run was continued until the best fi tness value reached less than 10^{-6}, or the population was converged within the range of 10^{-6} in each coordinate. The optimum was considered to be found only when the best fi tness value reached less than 10^{-6}.

4.3 Results and Discussions

The results are summarized in Table 4. The performance is compared using two standards, the number of trials where the algorithm succeeds in fi nding optimum (SUC) and the average number of function evaluations required for fi nding the optimum (EVAL). The standard deviations of the number of function evaluations required (SD) are also shown as the parenthesized values in the table.

As the Rosenbrock function is unimodal, the optimum solution was found in all of the trials. ENDX /MGG (both) and ENDX /MGG (xover), however, optimized this function with the smaller number of function evaluations than those of the others. The difference in the number of function evaluations was not disregardable at the signifi cance level $\alpha = 1\%$. This fact may indicate that the application of the low-discrepancy sequence into the "Generation of offspring" step decreases the number of function evaluations required for the optimization. On the other hand, the search performance of ENDX /MGG (init) that applies the low-discrepancy sequence only to the "Initialize" step resembled that of ENDX /MGG (none). The application of the low-discrepancy sequence into the "Initialize" step may not improve the search performance on this function.

On the multimodal functions, the search performances of ENDX /MGG (both) and ENDX /MGG (none) were almost the same as those of ENDX /MGG (init) and ENDX /MGG (xover), respectively. Since ENDX /MGG (both) and ENDX /MGG (init) found the optimum with higher probability than the others on the multimodal functions, the application of the low-discrepancy sequence into the "Initialize" step may play an important role in the optimization of multimodal functions. On the contrary, even if the low-discrepancy sequence was applied to the "Generation of offspring" step, the number of trials where the algorithm succeeds in fi nding optimum did not increase on these multi-modal functions.

The low-discrepancy sequences are often used by Monte Carlo methods in order to estimate the high-dimensional integral (see, e.g., [7]). In general, a Monte Carlo method with the low-discrepancy sequences is called a quasi-Monte Carlo method. In the estimation of the integral, the accuracy of the quasi-Monte Carlo method increases much faster than that of the original Monte Carlo method which uses no low-discrepancy sequence. Accordingly, the use of the the low-discrepancy sequences is considered a promising technique for resolving the curse of dimensionality in the high-dimensional integral. On the other hand, although EAs change the search space adaptively in contrast to Monte Carlo methods, EAs are related to Monte Carlo methods. Therefore, the low-discrepancy sequences may also improve the search performances of EAs in the high-dimensional space.

5. CONCLUSIONS

In this study, in order to generate individuals uniformly, we applied the low-discrepancy sequence generator, instead of the pseudo-random number generator, into the real-coded GA. The experimental results showed that the use of the low-discrepancy sequence enhances the search performance of the GA. When the low-discrepancy sequence was used to make the initial population uniformly distributed, the probability of fi nding the optimum solution was improved. Moreover, the low-discrepancy sequence decreased the number of function evaluations required for the optimization, when the recombination operator creates children utilizing the low-discrepancy sequence generator.

Table 4: Summary of results.

Objective function	Dim. n	ENDX/MGG(both)		ENDX/MGG(none)		ENDX/MGG(init)		ENDX/MGG(xover)	
		SUC	EVAL (SD)	SUC	EVAL (SD)	SUC	EVAL (SD)	SUC	EVAL (SD)
Rosenbrock	10	100/100	9.74×10^5 (7.08×10^4)	100/100	1.01×10^6 (6.85×10^4)	100/100	1.02×10^6 (7.57×10^4)	100/100	9.75×10^5 (8.33×10^4)
	15	100/100	3.29×10^6 (2.57×10^5)	100/100	3.49×10^6 (2.43×10^5)	100/100	3.48×10^6 (2.50×10^5)	100/100	3.33×10^6 (2.33×10^5)
	20	100/100	9.41×10^6 (7.77×10^5)	100/100	1.06×10^7 (9.96×10^5)	100/100	1.05×10^7 (9.53×10^5)	100/100	9.42×10^6 (7.71×10^5)
Rastrigin	10	90/100	5.57×10^5 (6.41×10^4)	84/100	5.76×10^5 (7.59×10^4)	94/100	5.61×10^5 (6.88×10^4)	84/100	5.89×10^5 (8.42×10^4)
	15	89/100	9.13×10^5 (1.04×10^5)	71/100	9.41×10^5 (1.24×10^5)	84/100	9.11×10^5 (1.20×10^5)	76/100	9.45×10^5 (1.23×10^5)
	20	88/100	1.27×10^6 (9.23×10^4)	73/100	1.33×10^6 (2.46×10^5)	89/100	1.26×10^6 (8.90×10^4)	72/100	1.32×10^6 (1.71×10^5)
Griewangk	10	72/100	5.74×10^5 (8.31×10^4)	60/100	5.86×10^5 (7.24×10^4)	74/100	5.91×10^5 (7.53×10^4)	71/100	6.00×10^5 (8.88×10^4)
	15	99/100	6.92×10^5 (5.53×10^4)	98/100	6.93×10^5 (5.95×10^4)	99/100	6.79×10^5 (3.91×10^4)	95/100	6.95×10^5 (6.18×10^4)
	20	97/100	9.58×10^5 (6.99×10^4)	98/100	9.42×10^5 (5.35×10^4)	99/100	9.39×10^5 (6.15×10^4)	98/100	9.98×10^5 (2.86×10^5)
Ackley	10	100/100	5.13×10^5 (1.51×10^4)	100/100	5.18×10^5 (1.49×10^4)	100/100	5.18×10^5 (1.54×10^4)	100/100	5.16×10^5 (1.42×10^4)
	15	100/100	8.84×10^5 (1.85×10^4)	100/100	8.81×10^5 (1.67×10^4)	100/100	8.83×10^5 (1.83×10^4)	100/100	8.80×10^5 (1.58×10^4)
	20	100/100	1.32×10^6 (3.29×10^4)	100/100	1.29×10^6 (2.10×10^4)	100/100	1.29×10^6 (2.55×10^4)	100/100	1.31×10^6 (2.92×10^4)

In this study, we applied the low-discrepancy sequence only to the simple real-coded GA, ENDX /MGG. Moreover, we tested the performances of the GAs with and without the low-discrepancy sequence only on the four benchmark functions. Therefore, further experiments should be required to investigate the effectiveness of the use of the low-discrepancy sequences more precisely.

The low-discrepancy sequences have been applied into EAs in very few studies [16]. The low-discrepancy sequences, however, may have an ability to enhance the performances of lots of EAs including real-coded GAs, evolution strategies (e.g., [1]), and so on. In future work, we should confirm whether the low-discrepancy sequences improve the search performances of more powerful EAs.

6. 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. In *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. In *Proc. of GECCO 1999*, 1851-1858, 1999.

[5] K. Deb, D. Joshi and A. Anand. Real-coded Evolutionary Algorithms with Parent-Centric Recombination. In *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. In *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] 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*, 36(12): 1162-1171, 2000 (in Japanese).

[9] H. Kita, I. Ono and S. Kobayashi. Multi-parental Extension of the Unimodal Normal Distribution Crossover for Real-coded Genetic Algorithms. In *Proc. of CEC 1999*, 1581-1588, 1999.

[10] 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.

[11] M.M. Meysenburg and J.A. Foster. Randomness and GA Performance, Revisited. In *Proc. of GECCO 1999*, 425-432, 1999.

[12] M.M. Meysenburg and J.A. Foster. Random Generator Quality and GP Performance. In *Proc. of GECCO 1999*, 1121-1126, 1999.

[13] W.J. Morokoff and R.E. Caflisch. Quasi-random sequences and their discrepancies. *SIAM J. on Scientific Computing*, 15(6): 1251-1279, 1994.

[14] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: SIAM, 1992.

[15] I. Ono and S. Kobayashi. A Real-coded Genetic Algorithm for Function Optimization Using Unimodal Normal Distribution Crossover. In *Proc. of 7th Int. Conf. on Genetic Algorithms*, 246-253, 1997.

[16] I.C. Parmee and C.R. Bonham. Improving Cluster Oriented Genetic Algorithms for High-perfomance Region Identifi cation. In *Proceedings US United Engineering Foundation's 'Optimization in Industry' Conference*, 2001.

[17] H. Satoh, M. Yamamura and S. Kobayashi. Minimal Generation Gap Model for GAs considering both Exploration and Exploitation. In *Proc. of the IIZUKA*, 494-497, 1996.

[18] S. Tsutsui, M. Yamamura and T. Higuchi. Multi-parent Recombination with Simplex Crossover in Real Coded Genetic Algorithms. In *Proc. of GECCO 1999*, 657-664, 1999.

[19] X. Wang and F.J. Hickernell. Randomized Halton Sequences. *Mathematical and Computer Modelling*, 32: 887-899, 2000.

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 as defi ned below have a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$.

$$\begin{cases} z_1 = \sqrt{-2\ln(x_1)}\cos(2\pi x_2), \\ z_2 = \sqrt{-2\ln(x_1)}\sin(2\pi x_2). \end{cases}$$

When the low-discrepancy sequences are used as the input of the Box-Muller transformation, it seems to give us "good" normally distributed points (Figure 3).

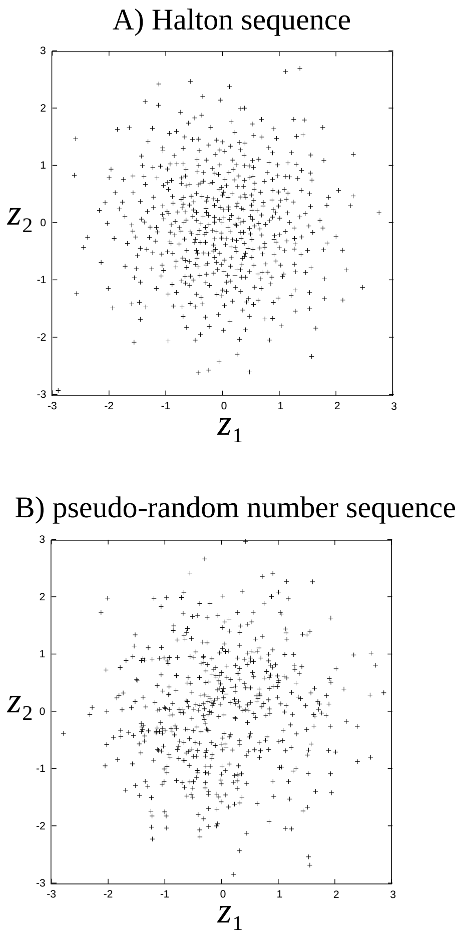

Figure 3: 2-D plots of 500 normally distributed points generated by A)the Halton sequence, and B)the pseudo-random number sequence, respectively.