Genetic Algorithm for searching parameters of Sobol sequence

by

Elena Kovaleva

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Prof. Christiane Lemieux

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

Sobol sequence is related to the class of low-discrepancy sequences, deterministic sequences constructed to have better uniformity properties than a random sample. The uniformity level of the point set generated by Sobol method is influenced by the choice of initial parameters, so-called Direction Numbers (DNs). The space of this parameters is odd integers and it is exponentially growing with dimension. In this aper, we propose to use Genetic Algorithms (GA) in order to explorer the space of Direction Numbers and find the optimal set of parameters in terms of uniformity quality. Experimental results show that GA based methods can be effective and perform at least on the same level as for example Kuo and Joe approach if the search of DNs is conducted for each dimension separately.

Acknowledgements

I would like to thank all the little people who made this possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

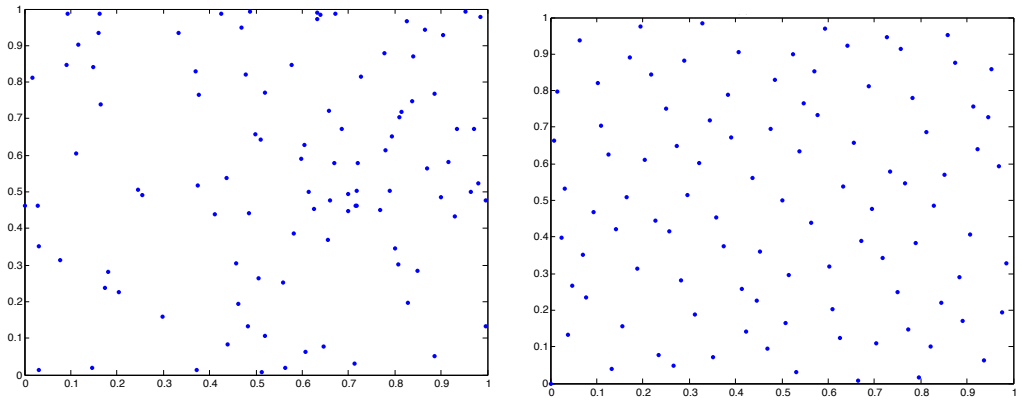
Introduction

In order to explore complex multidimensional spaces, several fields of research such as optimization, design of experiments, numerical integration use random samples. Often the standard random number generator can be replaced. As Niederreiter [16] says:

[...] instead of trying to cope with impalpable concept of randomness, one should select points according to a deterministic scheme that is well suited for the problem at hand. [...] For instance, in the area of numerical integration it turns out to be quite irrelevant whether the sample points or "nodes" are truly random; of primary importance is really the even distribution of the points over [integration domain].

The uniformity properties of generated samples are very important because there is a strong correlation between them and the accuracy of approximation [16]. In many cases, pseudo-random number generators produce distributions with unsatisfying uniformity quality as seen in Figure 1.0.1*a*. It is almost impossible for them to produce samples distributed as evenly as the ones generated from quasi-random set 1.0.1*b*.

These quasi-random techniques have already been applied in many areas of computer science and finance; see, for example, Niederreiter [18], L'Ecuyer and Lemieux [11] and Glasserman [6]. In some cases, construction of a quasi-random point set requires to define initial parameters. The choice of these parameters influences the uniformity properties of the constructed point set.



(a) (b)

Figure 1.0.1: Pseudo-randomly generated (a) and quasi-randomly generated (b) samples.

Genetic algorithms (GA) [9] are known as powerful meta-heuristic optimization methods. Multiple times they proved their ability to solve very difficult optimization problems. Nevertheless, these algorithms are rarely used to configure quasi-random (or low discrepancy) sequence generators. In our research, we apply GA-based methods to search for parameters than can produce very uniform Sobol sequences, which are popular quasi-random techniques.

The structure of this research paper is as follows. The next few sections provide basic definitions of Quasi-Monte Carlo methods and give the idea of the research goal. In Chapter 2 we talk in detail about the construction algorithm for the Sobol sequence, its parameters and its quality measures. The general concept of GA is introduced in the first section of Chapter 3; in the next two sections of the chapter we discuss methods for searching parameters of Sobol Sequence developed on the basis of GA. The results of our experiments are presented in Chapter 4. A conclusion is given in Chapter 5.

1.1 Monte Carlo and *quasi*-Monte Carlo methods

The Monte Carlo method is widely used to approximate integrals that have no closed-form solution. The Monte Carlo estimator for the integral

$$I(g) = \int_{[0,1)^s} g(\mathbf{u}) \mathrm{d}\mathbf{u},$$

where g is a real-valued function defined on the s-dimensional unit hypercube $[0,1)^s$ and $\mathbf{u} = (u_1, ..., u_s)$, is given by

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{u}_i),$$

where $\{\mathbf{u}_1, ..., \mathbf{u}_n\}$ is an independent and identically distributed sample from the uniform distribution over $[0,1)^s$. The integration error, by Central limit Theorem satisfies

$$I(g) - \hat{\mu} \approx N(0, \sigma/\sqrt{n}),$$

with $n \gg 1$ and σ^2 is the variance of g given by

$$\sigma^2 = \int_{[0,1)^s} (g(\mathbf{u}) - I(g))^2 \mathrm{d}\mathbf{u}.$$

It can be concluded that Monte Carlo integration error is in $O(1/\sqrt{n})$.

The convergence rate of the error does not depend on the dimension$s$, but the method is slow. For example, if we want to improve the accuracy by a factor 10, we need to generate 100 times more points. In order to speed up the performance of Monte Carlo, the random sample in the estimator can be replaced with a well-chosen deterministic point set, known as a $low-discrepancy$ point set. The approach is called *quasi*-Monte Carlo (QMC) integration. These low-discrepancy sets are constructed to be more uniformly distributed, i.e. contain less gaps and clusters than in random sample.

The concept of discrepancy provides a measure of non-uniformity of a point set placed in a unit hypercube $[0,1)^s$. The most widely studied distance measure is the star discrepancy defined by Niederreiter in [18] as:

$$D^*(\mathbf{u}_1, ..., \mathbf{u}_n) = \sup_{0 \le w_j < 1, j=1,...,s} \left| \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{s} 1_{0 \le u_{ij} < w_j} - \prod_{j=1}^{s} w_j \right| \tag{1.1}$$

In other words, we calculate the absolute difference between the volume of every subset $[0,1)^s$ of the form $[0, w_1) \times ... \times [0, w_s)$ and the number of points in the subset divided by n. The maximum difference is the star discrepancy D^*.

By definition, if a sequence $\mathbf{u}_1, ..., \mathbf{u}_n$ of points in $[0,1)^s$ for any $n \geqslant 1$ satisfies:

$$D^*(\mathbf{u}_1, ..., \mathbf{u}_n) \leq c(s)\frac{(\log n)^s}{n}, \tag{1.2}$$

it is called a low-discrepancy sequence. Here, the constant $c(s)$ depends only on the dimension s.

Two of the oldest constructions which are very popular among practitioners are the Halton [7] and Sobol sequences [**?**].

Assuming that points $\mathbf{u}_1, ..., \mathbf{u}_n$ are chosen from a low-discrepancy set, the Koksma-Hlawka inequality (1.3) establishes a deterministic bound on the QMC integration error, given by

$$\left|\frac{1}{n}\sum_{i=1}^{n} g(\mathbf{u}_i) - \int_{[0,1)^s} g(\mathbf{u})\mathrm{d}\mathbf{u}\right| \leq V(g)D^*(\mathbf{u}_1, ..., \mathbf{u}_n), \tag{1.3}$$

which means when $\{\mathbf{u}_1, ..., \mathbf{u}_n\}$ comes from a low-discrepancy sequence, the error is in $O((\log n)^s/n)$.

The quantity $V(g)$ is the variation of g and it is defined according to [17] as

$$V(g) = \sum_{k=1}^{s} \sum_{1\leq i_1<...<i_k\leq s} V^k(g; i_1, ..., i_k),$$

where $V^k(g; i_1, ..., i_k)$ is an application of

$$V^s(g) = \int_0^1 ... \int_0^1 \left|\frac{\partial^s g}{\partial u_1 ... \partial u_s}\right| \mathrm{d}u_1 ... \mathrm{d}u_s$$

to the restriction of g to the k-dimensional face $\{(u_1, .., u_s) \in [0, 1]^s : u_j = 1 \text{ for } j \neq i_1, ..., i_k\}$.

MC integration gives the error bound of $O(1/\sqrt{n})$ so for fixed dimension s Monte-Carlo method converges slower than *quasi*-Monte Carlo. However, even with relatively small s, for $1/\sqrt{n}$ to be larger than $(\log n)^s/n$ requires n to be huge. For instance, if $s = 10$, the inequality holds once $n \approx 10^{39}$. This limits the usefulness of the QMC error bound in practice since 10^{39} is an infeasible sample size.

Another practical limitation of the Koksma-Hlawka inequality is the condition that $V(g)$ must be finite. So it means that g should be bounded which for integrands in financial applications like option pricing is often not the case. Moreover, $V(g)$ as well as D^*, are

difficult to compute. In addition, the Koksma-Hlawka inequality gives only an upper bound, i.e. it does not provide a reliable error estimate. For these reasons, in practice, the performance of QMC methods is often tested and compared through numerical experiments and by using a randomization method, as discussed in the next section.

1.2 Randomized QMC

Referring to [14], let $P_n = \{\mathbf{u_1}, ..., \mathbf{u_n}\} \subseteq [0,1)^s$ be a deterministic low-discrepancy point set. By applying a randomization function to each point $\mathbf{u}_i \in P_n$ we obtain a randomized point set $\tilde{P}_n = \{\tilde{\mathbf{u}}_\mathbf{1}, ..., \tilde{\mathbf{u}}_\mathbf{n}\} \subseteq [0,1)^s$. Each point $\tilde{\mathbf{u}}_i \in \tilde{P}_n$ is $U([0,1)^s)$ (uniformly distributed over $[0,1)^s$) so that the estimator based on $\tilde{P}_n$ is unbiased:

$$E\left[\frac{1}{n}\sum_{i=1}^{n} g(\tilde{\mathbf{u}}_\mathbf{i})\right] = \frac{1}{n}\sum_{i=1}^{n} E\big[g(\tilde{\mathbf{u}}_\mathbf{i})\big] = \frac{1}{n}\sum_{i=1}^{n}\int_{[0,1)^s} g(\tilde{\mathbf{u}}_\mathbf{i})\mathrm{d}\tilde{\mathbf{u}}_\mathbf{i} = I(f)$$

The low-discrepancy of P_n should not be destroyed through randomization otherwise the advantage of using QMC over MC is lost. If randomization is appropriately chosen, we can create a random sample of QMC estimators

$$\hat{\mu}_{rqmc,t} = \frac{1}{n}\sum_{\tilde{\mathbf{u}}\in\tilde{P}_n,t} g(\mathbf{u}_i), t = 1..m$$

where $\tilde{\mathbf{P}}_{\mathbf{n},\mathbf{1}}, ..., \tilde{\mathbf{P}}_{\mathbf{n},\mathbf{m}}$ are m independent randomized copies of P_n. Then, the approximation of is the estimator

$$\hat{\mu}_{rqmc} = \frac{1}{m}\sum_{t=1}^{m}\hat{\mu}_{rqmc,t}$$

whose variance can be estimated by

$$\hat{\sigma}^2_{rqmc} = \frac{1}{m}\left(\frac{1}{m-1}\sum_{t=1}^{m}\big(\hat{\mu}_{rqmc,t} - \hat{\mu}_{rqmc}\big)^2\right).$$

Randomization methods are generally designed for specific types of low-discrepancy sequence, and there are several common methods.

One of the most simple randomization technique is to use *random shift*, also called a *rotation sampling*. The idea behind is to generate a uniform random vector $\mathbf{v} \in U([0,1)^s)$ and then let

$$\tilde{\mathbf{u}}_i = (\mathbf{u}_i + \mathbf{v}) mod(1)$$

for $i = 1, ..., n$, where the modulo 1 operation is taken coordinatewise. Because $\mathbf{v}$ is uniform, each point $\tilde{\mathbf{u}}_i$ is also uniformly distributed.

More information about QMC randomization can be found in [6, 14]

1.3 Motivation and goal of the research

As we already know, low-discrepancy sequences are constructed to have better uniformity properties than a random sample. But in addition, there exist some methods to improve "initial" uniformity of these sequences, i.e quality obtained in the basic construction algorithm with randomly chosen parameters.

Let us look at Halton sequence [7]. Referring to [14, p.153], the ith term in this sequence is given by:

$$\mathbf{u}_i = (\phi_{b_1}(i-1), ..., \phi_{b_s}(i-1)), i \leq 1,$$

where b_j is the jth prime number. The *radical-inverse* function ϕ_b is defined as

$$\phi_b(i) = \sum_{l=0}^{\infty} a_l(i) b^{-l-1},$$

where the coefficients $a_l(i)$ come from the expansion

$$i = \sum_{l=0}^{\infty} a_l(i) b^l$$

and where we assume infinitely many coefficients $a_l(i) = 0$.

Let us look at small example. Assume that we want to find the second ($j = 2$) coordinate of the eighth ($i = 8$) point of Halton sequence, i.e. we want to find $\phi_{b_2}(8-1) = \phi_3(7)$, where base b_2 is equal to the second prime number, which is 3. Now we can define $a_l(i)$ trough expansion of i : $7 = 1 \times 3^0 + 2 \times 3^1$, i.e. $a_0(7) = 1, a_1(7) = 2$. Then, substituting the

coefficients in the formula, we get the coordinate $\phi_3(7) = a_0(7) \times b^{-0-1} + a_1(7) \times b^{-1-1} = 1 \times \frac{1}{3} + 2 \times \frac{1}{9} = \frac{5}{9}$.

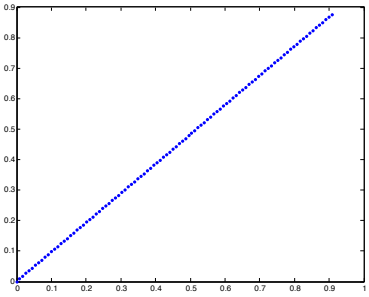
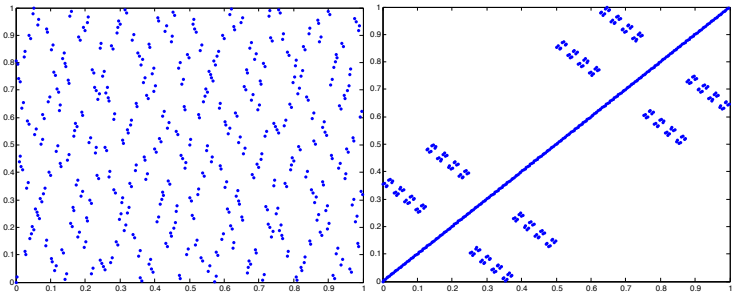


Figure 1.3.1: Projection of dimensions 29 and 30 of the first 100 points of Halton sequence

The Halton sequence suffers from the correlation between the points in high dimensions which affects its uniformity quality, as illustrated on Figure 1.3.1. One of the possibilities to improve the quality of the Halton sequence is to apply permutations to the coefficients $a_l(i)$[2]. In 2012, researchers Rainville, Gagné, Teytaud and Laurendeau came up with the idea to search for better permutations (in terms of uniformity) by using Evolutionary Algorithms, also know as Genetic Algorithms (GA) [20]. Briefly, they represent a permutation as a vector π_j of indices of coefficients $a_l(i)$ and *evolve* permutations dimension by dimension based on a chosen fitness criterion. The results they obtained in their experiments was an inspiration for this research.



(a) (b)

Figure 1.3.2: The 39:40 projections of the first 512 points of two Sobol sequences with different direction numbers

Let us look at two-dimensional projections of two different point sets. Both sets were generated by Sobol method. They have the same number of points, their projections were taken for the same dimensions. They differ from each other only by DNs. As we can see, the points of the projection on Figure 1.3.2*a* are much more evenly distributed than the points of the projection on Figure 1.3.2*b*.

The goal of this paper is to investigate the possibility of applying GA for searching parameters of Sobol sequence in order to improve its uniformity properties.

Chapter 2

Sobol Sequence

The Sobol sequence is a so-called (t, s)-sequence in base $b = 2$ where s is the dimension of the sequence and $t - value$ is a uniformity measure discussed in Section 2.2.2. The sequence was introduced by Russian mathematician I.M. Sobol in 1967 [21].

2.1 Construction

The detailed explanation for constructing the Sobol sequences is given in Bratley and Fox, Algorithm 659 [3]. To generate jth coordinate, it needs a primitive polynomial $P_j(z)$:

$$P_j(z) = z^{d_j} + a_{j,1}z^{d_j-1} + a_{j,2}z^{d_j-2} + ... + a_{j,d_j-1}z + 1, \tag{2.1}$$

where coefficients $a_{j,k} \in \{0, 1\}$ and where d_j is the degree of $P_j(z)$. According to the error bounds given by Sobol in [21], the degree d_j should be as low as possible.

The next step in the construction of the Sobol sequence is to choose initial values for d_j odd integers $m_{j,r}$ in $\{1,..., 2^r - 1\}$ for $r = 1, ...,d_j$. The rest of these numbers $(r > d_j)$ can be obtained recursively using

$$m_{j,r} = 2^1 a_{j,1} m_{j,r-1} \oplus 2^2 a_{j,2} m_{j,r-2} \oplus ... \oplus 2^{d_j-1} a_{j,d_j-1} m_{j,r-d_j+1} \oplus 2^{d_j} m_{j,r-d_j} \oplus m_{j,r-dj} \tag{2.2}$$

Now, the so-called direction numbers can be calculated from the formula:

$$v_{j,r} = \frac{m_{j,r}}{2^r} \tag{2.3}$$

Finally, to generate the sequence $\{u_{i,j}\}, i = 1, ..., N$ we can use:

$$u_{i,j} = i_1 v_{j,1} \oplus i_2 v_{j,2} \oplus ... \tag{2.4}$$

where $....i_1 i_2 i_3$ is the binary representation of the index i and $\oplus$ is the exclusive-or operation on binary vectors.

The last formula can be rewritten as a recursion:

$$u_{i+1,j} = u_{i,j} \oplus v_{j,c}, \tag{2.5}$$

where the index c is such that g_c is the leftmost zero-bit in the reverse binary representation $g(i) = g_1 g_2 g_3 ... = ...i_3 i_2 i_1$. The Gray code function $g(.)$ satisfies $g(0) = 0$ and the binary expansion of $g(i+1)$ differs from $g(i)$ only by one position, i.e if c is the smallest index s.t. $i_c \neq b - 1$, then $g_c(i) = g_c(i) + 1$ in the expansion of $g(i+1)$.This algorithm was proposed by Antonov and Saleev in 1979 [1] as an efficient implementation of Sobol's method.

Let us look at a simple example similar to those presented in [14, p.158]. Suppose that, for $j = 5$, we take $P_5 = z^3 + z^2 + 1$. Since the degree of P_5 is three, we need to choose three initial numbers $m_{5,r}, r = 1, 2, 3$. Suppose, we take $m_{5,1} = 1, m_{5,2} = 1, m_{5,3} = 5$. Then, from the definition of P_5, we have that $a_{5,1} = 1, a_{5,2} = 0, a_{5,3} = 1$ and so

$$m_{5,4} = 2m_{5,3} \oplus 0 \oplus 8m_{5,1} \oplus m_{5,1} = 10 \oplus 8 \oplus 1 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = 3$$

$$m_{5,5} = 6 \oplus 8 \oplus 1 = 15$$

$$m_{5,6} = 30 \oplus 40 \oplus 5 = 51$$

Now, we can calculate the first six direction numbers as:

$v_{5,1} = \frac{1}{2^1} = \frac{1}{2}$, $v_{5,2} = \frac{1}{2^2} = \frac{1}{4}$, $v_{5,3} = \frac{5}{2^3} = \frac{5}{8}$, $v_{5,4} = \frac{3}{16}$, $v_{5,5} = \frac{15}{32}$, $v_{5,6} = \frac{51}{64}$

Let C_5 be the 6×6 matrix containing their binary representation, where columns correspond to direction numbers in the given order. In our case we have:

$$C_5 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

For each s-dimensional sequence there exist s *generating matrices* $C_1, ..., C_s$ of unlimited size. As you can see, C is an upper triangular matrix. Because of oddness of values m, it has all ones the main diagonal. Based on these properties of generating matrix, we can represent DNs for given dimension as a single binary string, the form convenient to be implemented in GA. In our example, DNs can be represented as 010001011111100. convinent

Finally, we use the Gray code function to generate the 5th coordinate of the first eight points $u_{i,j=5}$ (for simplicity index j is omitted):

$Step(0) : u_0 = 0, i = 0, c = 1$

$Step(1) : u_1 = u_0 \oplus v_1 = \frac{1}{2}, i = 1 = (01) => g(i) = (10) => c = 2$

$Step(2) : u_2 = u_1 \oplus v_2 = \frac{1}{2} \oplus \frac{1}{4} = \frac{3}{4}, i = 2 = (10) => g(i) = (01) => c = 1$

$Step(3) : u_3 = u_2 \oplus v_1 = \frac{3}{4} \oplus \frac{1}{2} = \frac{1}{4}, i = 3 = (011) => g(i) = (110) => c = 3$

$Step(4) : u_4 = u_3 \oplus v_3 = \frac{1}{4} \oplus \frac{5}{8} = \frac{7}{8}, i = 4 = (100) => g(i) = (001) => c = 1$

$Step(5) : u_5 = u_4 \oplus v_1 = \frac{7}{8} \oplus \frac{1}{2} = \frac{3}{8}, i = 5 = (0101) => g(i) = (1010) => c = 2$

$Step(6) : u_6 = u_5 \oplus v_2 = \frac{3}{8} \oplus \frac{1}{4} = \frac{1}{8}, i = 6 = (110) => g(i) = (011) => c = 1$

$Step(7) : u_7 = u_6 \oplus v_1 = \frac{1}{8} \oplus \frac{1}{2} = \frac{5}{8}, i = 7 = (0111) => g(i) = (1110) => c = 4.$

It should be noted that when using the Gray code, we get the same points as those constructed by the original Sobol's method, but in a different order over the first 2^k points for each $k \geq 0$. The proof that the resulting sequence is still a (t, s)-sequence with the same value of t is given in [1, 22]. The direction numbers up to $s = 40$ were provided by Bratley and Fox [3] and come from Sobol, but most practical tasks require higher dimensional sequences. An interested reader can find direction numbers for 360-dimensional sequence

in the RandQMC library created by Lemieux [13]. To search for direction numbers in high dimensions Lemieux used *random walk* technique. In this research, for the same purpose, we want to apply *Evolutionary Algorithms*.

2.2 Quality measures

In this Section we discuss some criteria which help us to distinguish, in terms of uniformity, "bad" and "good" parameters of Sobol sequence. These criteria define the types of *fitness* functions we use in Genetic Algorithms.

2.2.1 Modified L_2- discrepancy norm

Replacing the *sup* norm byL_2 norm in (1.1), we obtain practical discrepancy measures. The modified version of $L_2 - discrepancy$ measure was introduced by Hickernell in [8]:

$$ML_2(\mathbf{U_N})^2 = \left(\frac{4}{3}\right)^s + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\prod_{k=1}^{s}[2 - \max(u_{i,k}, u_{j,k})] - \frac{2^{1-s}}{N}\sum_{i=1}^{N}\prod_{k=1}^{s}(3 - u_{i,k}^2), \quad (2.7)$$

where $u_{i,k}$ denote the kth coordinate of the i point $\in I^s$ of $\mathbf{U}_N$.

This was the only one criterion used to define fitness of Halton permutations in [20].

2.2.2 Parameter t-value

Before we define quality parameter t-value, the concept of $(q_1, .., q_s) - equidistribution$ should be explained. Following the definition given by Lemieux [14, *p.*76], we call a Sobol set Ψ_s of 2^k points $(q_1, .., q_s) - equidistributed$, if for non-negative integers $q_1, .., q_s$, every cell of the form

$$\prod_{j=1}^{s}\left[\frac{r_j}{2^{q_j}}, \frac{r_j + 1}{2^{q_j}}\right), \quad (2.8)$$

for $0 \leq r_j < 2^{q_j}, j = 1, .., s$, contains 2^{k-q} points from Ψ_s , where $q = \sum_{j=1}^{s} q_j$.

Partitioning the unit cube in 2^q congruent boxes of size 2^{-q_j} in dimension j, we verify that each box contains the same number of points. And this is true only if the number of points is at least the same as the number of boxes, i.e. we must have $q \leq k$.

Now, we can define the t-value of Ψ_s as the smallest integer t satisfying $q \leq k - t$ and such that for all $q_1, .., q_s \geq 0$, the point set Ψ_s is $(q_1, .., q_s) - equidistributed$.

The original idea of this criterion belongs to Sobol, who labeled $k-t$ as τ. The modern notation t, widely used to study QMC methods, was introduced by Niederreiter in [15]. The smaller t is, the more the point set is uniformly distributed.

A general algorithm to evaluate parameter t (Figure 2.2.1) is described in Pirsic [19]. As mentioned in this paper, the t-value can be determined in terms of properties of generating matrices $C_1, ..., C_s$; see [19] and the references therein for detailed explanation.

Data: matrices $C_1, ..., C_s$
Result: $t - value$ of point set Ψ_s
initialization;
for $d = 1$ *to* m **do**
  for *all partitions* d *in* s *parts* **do**
    compose subset $\{v'_1, ..., v'_d\}$ of vectors;
    if *the rank of the subset is smaller then* d **then**
      exit;
      return $t = m - d + 1$;
    end
  end
end

Figure 2.2.1: Algorithm of the t-value approach

Let us now look at the algorithm. For each d in the range from 1 to m, where m is the dimension of square matrices C (in the previous section matrix 2.6 has $m = 6$), the program tests whether or not the given matrices are a so-called (d, m, s)-system. This is achieved by checking the rank [1] of each subset constructed from the first d_1 row vectors of C_1, the first d_2 row vectors of C_2 and so on, where $d_1, ..., d_s$ is a partition of d into s. If the rank of a non-negative integers subset is smaller than d, the matrices do not form a (d, m, s)-system which means they form $(d-1, m, s)$-system and, therefore, t-value is equal $m - (d-1)$. It should be noted that t-value measures the uniformity of the first 2^m points.

[1] Number of linear independent vectors in the set

In order to avoid quite expensive computations, in this research the t-value was calculated using a *window* approach where we only consider pairs and triple of coordinates of the form $\{l, l+k\}, 1 \leq k < w$ and $\{l, l+k_1, l+k_2\}, 1 \leq k_1 < k_2 < w$ of matrices C , respectively. For instance, suppose that window size w is equal to 4 and $s = 6$, i.e generating matrices are $C_1, C_2, C_3, C_4, C_5, C_6$. Then, we compute the t-value for the following pairs and triples

$$\text{pairs: } \begin{matrix} \{1,2\} & \{1,3\} & \{1,4\} \\ \{2,3\} & \{2,4\} & \{2,5\} \\ \{3,4\} & \{3,5\} & \{3,6\} \\ \{4,5\} & \{4,6\} & \\ \{5,6\} & & \end{matrix} \qquad \text{triples: } \begin{matrix} \{1,2,3\} & \{1,2,4\} & \{1,3,4\} \\ \{2,3,4\} & \{2,3,5\} & \{2,4,5\} \\ \{3,4,5\} & \{3,4,6\} & \{3,5,6\} \\ \{4,5,6\} & & \end{matrix}$$

2.2.3 Resolution

The criterion *resolution* was introduced by L'Ecuyer in [12] . By definition, *resolution* of Ψ_s is the largest integer l_s such that Ψ_s is $(l_s, ..., l_s)$-equidistributed.

Geometrically speaking, a resolution of l_s means that the unit hypercube $[0,1)^s$ can be partitioned into 2^{sl_s} congruent cubic boxes, each of which contains l_s points from Ψ_s. The partitioning is done by dividing each axis into 2^{l_s} intervals of size 2^{-l_s}.

In order to evaluate the resolution, an approach similar to the one described for the t-value can be used. To illustrate how the algorithm actually works, let us look at a simple case. Suppose we want to get the resolution for three-dimensional Sobol sequence with $m = 12$, i.e we have matrices C_1, C_2 and C_3 all of dimension 12×12. If the subset of twelve rows, containing the first four rows from each matrix C, is linearly independent, then the resolution is equal to 4, otherwise we need to combine the subset of nine rows, including the first three rows from each matrix C, and check its independence. The stopping criterion for this algorithm is when we have linear independence of the constructed subset. The higher is the resolution, the better are the uniformity properties of the corresponding point set.

It should be taken into account that resolution is a less reliable quality measure than the t-value. The resolution algorithm checks equidistribution only for the subsets of the form $(l_s, ..., l_s)$, whereas the t-value algorithm verifies the equidistribution property for all subsets $(q_1, .., q_s)$. However, the obvious advantage of the resolution method is the computation speed.

Chapter 3

Genetic Algorithms

The ultimate goal of our research to design the method able to search for initial parameters of Sobol sequence, so-called Direction Numbers (DNs), that provide higher uniformity quality to the point set than randomly chosen. By construction DNs (as m_j, r in Section 2.1) are positive odd integers which makes difficult to apply for their search most of optimization techniques, for instance Gradient Descent, oriented on the search in continuous space. In this situation Genetic Algorithm seems like suitable tool that does not have listed limitations.

In this Chapter we discuss general concept of Genetic Algorithm and talk about two GA based methods developed to search for Direction Numbers of Sobol sequence.

3.1 Concept of GA

Genetic Algorithms (GA), often called Evolutionary Algorithms, have been introduced by Holland in 1975 [9]. The inspiration for these population-based optimization algorithms was the Darwinian evolution theory.

Each generation is produced through variations (crossover/recombination and mutation) of solutions from the current population as shown on Figure 3.1.1. Then, the algorithm selects the fittest solutions to be reproduced in the next generation; good characteristics (building blocks) are passed over and recombined in the next solutions in order to produce even fitter solutions. In genetic computation, the solutions are called individuals, and a set of solutions present at the same time in the algorithm is called population.

The value which indicates how well the proposed solution addresses the problem is called fitness.

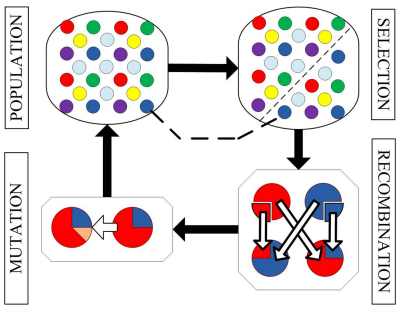


Figure 3.1.1: Scheme of GA [23]

Let us now look at the pseudocode for a very simple genetic algorithm given by Figure 3.1.2.

initialize $P^{(1)} \leftarrow [(x_i, f(x_i)), i = 1, ..., \mu]$;
for $g = 1$ *to* ω **do**
  new_generation$\leftarrow$ empty;
  while $l < \lambda$ **do**
    if *crossover* **then**
      parents$\leftarrow$ select_random($P^{(g)}$,2);
      $o_l \leftarrow$ recombine(parents);
      $l \leftarrow l + 1$;
      if *mutation* **then**
        $o_l \leftarrow$ mutate(o_l);
      end
    end
    add($[o_l, f(o_l)]$, new_generation);
  end
  $P^{(g+1)} \leftarrow$ select($P^{(g)} \cup$ new_generation, μ);
end
return best($P^{(\omega)}$,1)

Figure 3.1.2: Genetic Algorithm

The first step is to generate, at random, an initial population $P^{(1)}$ consisting of μ

feasible solutions x_i. The function $f(\cdot)$ defines the fitness of each initial solution. For every generation from 1 to ω, the algorithm produces λ pairs of offspring. In the inner loop, with predefined probability of crossover, the algorithm selects two individuals as parents: the better is the fitness the higher is the probability to be chosen. Via recombination of parent genes the algorithm generates a pair of new solutions, each of which can mutate with given probability. The resulting solutions are evaluated and stored in the new_generation array. After the new generation is fully produced, the algorithm selects the next μ sized parental population $P^{(g+1)}$ from the union of parental and offspring population, according to the fitness of the individuals. Finally, when a predefined number (ω) of generations are completed, the algorithm returns the best solution in terms of fitness from the last population $P^{(\omega)}$.

3.2 Local Evolution

By construction, Direction Numbers for the first two dimensions of Sobol sequence are constant.

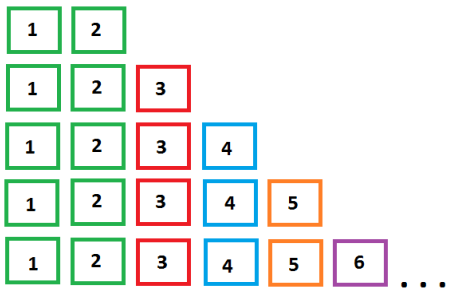


Figure 3.2.1: Scheme of Local Evolution

The idea for Local Evolution (LE) approach is to 'grow' the sequence dimension by dimension. For each new dimension we run Genetic Algorithm to find the DNs which are the better fit for the existing DNs in terms of uniformity.

```
le_fitness (C, C_j, w, opt);
s ← length(C);
up_window ← w;
if w > s then
    up_window ← s;
end
else
    C ← C[s − w + 1 : s];
end
if opt = 1 then
    C3 ← C_j;
    k ← 1;
    for i = 1 to up_window−1 do
        C1 ← C[i];
        for t = i + 1 to up_window do
            C2 ← C[t];
            Qm3[k] ← resolution(C1, C2, C3);
            k ← k + 1;
        end
    end
    C2 ← C_j;
    for i = 1 to up_window do
        C1 ← C[i];
        Qm2[i] ← resolution(C1, C2);
    end
    fitness ← 0.6 × min_i(Qm2) + 0.4 × min_k(Qm3);
end
else
    C2 ← C_j;
    for i = 1 to up_window do
        C1 ← C[i];
        Qm[i] ← t_value(C1,C2);
    end
    fitness ← max_i(Qm);
end
return fitness
```

Figure 3.2.2: LE fitness function

In this method DNs up to 160 dimensions are represented as binary strings (explanation is given in the comments for Equation 2.6), while for dimensions from 161 to 1000, they are represented as initial values m (Equation 2.3 shows their correlation). The reason for that is the storage limitation of software we used in our research. The binary representation of DNs for dimension 160 and 161 requires 45 and 55 bits respectively, whereas a maximum binary string stored in MatLab is of 52 bits.

The fitness function $f(\cdot)$ in LE estimates the uniformity quality of the union of DNs of evolved dimensions $C = \{C_i\}, i = 1...j-1$ and an evolving dimension C_j. We use two variations of fitness function in this method: the resolution and the t -value approach. In the pseudocode (Figure 3.2.2), the approach is determined by the parameter *opt*, which is set equal to 1 in the case of resolution, and any other number for the t-value. As you can see in the code, the algorithm chooses the "worst case" value of a quality measure among all estimations of pairs/triplets as a fitness criterion . The smallest t-value ($t = 0$) corresponds to the best uniformity properties so t-value based fitness is equal the maximum of all 2D estimations, stored in array Qm, in the given window w (see explanation in the Section 2.2.2). In the second variation, the algorithm minimizes 2D and 3D resolution (arrays $Qm3$ and $Qm2$) and returns as a fitness their combination with weights, 0.6 and 0.4 respectively. This approach allows to keep fitness function more objective, i.e not overestimate uniformity quality of the DNs.

To generate an offspring, GA randomly selects two parents using the Roulette Wheel method illustrated on Figure 3.2.3; the individual probabilities of being selected as a parent are correlated to their fitness.

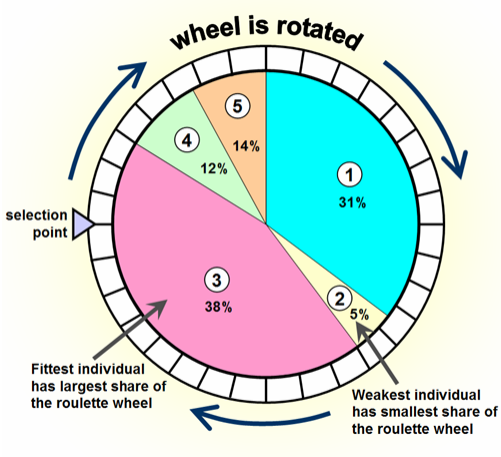


Figure 3.2.3: Roulette Wheel Selection [4]

Through the crossover process, the algorithm produces two new solutions. Figure 3.2.4 shows the recombination process for the binary case.

Figure 3.2.4: LE Crossover

Both of these solutions can mutate (with predefined probability) i.e., a randomly selected gene (bit) of the solution is flipped over.

Figure 3.2.5: LE Mutation

To summarize, let us now look at the simplified version of the LE code presented in Figure 3.2.6.

initialize $chosenDNs \leftarrow [DNs^{(1)}, DNs^{(2)}]$;
for $j = 3$ *to* 1000 **do**
  $DNs^{(j)} \leftarrow \text{GA}(chosenDNs, j)$;
  $chosenDNs \leftarrow chosenDNs \cup DNs^{(j)}$;
end
return DNs

Figure 3.2.6: LE algorithm

We initialize the array DNs with Direction Numbers for the first two dimensions which are constant. Then, for dimensions from 3 to 1000 we run GA to find the best Direction Numbers in terms of uniformity. The final result is the array of DNs for 1000 dimensions.

The pseudocode of GA can be found in Section 3.1. For the given dimension j the system randomly generates an initial set of μ Direction Numbers and estimates their fitness $P^{(1)} = [(x_i, f(x_i))], i = 1, ..., \mu$. To evaluate the fitness of a candidate DNs for the next dimension j, GA converts $chosenDNs$ to C and the candidate to C_j and calls $le\_fitness()$. At each step GA creates new DNs by recombining and mutating genes of the DNs selected by Roulette Wheel from the parental population $P^{(g)}$. GA returns the fittest DNs from

the last population $P^{(\omega)}$. The parameters of GA for the LE method are presented in the second column of Table 3.1.

Parameter	LE	GE
μ	40	500
ω	20	200
λ	16	100
crossover Pr	0.8	0.8
mutation Pr	0.05	0.05
window	10	10
dimension of C	12	12

Table 3.1: Parameters of LE and GE

3.3 Global Evolution

The Global Evolution (GE) method uses a Genetic Algorithm to search parameters for the Sobol sequence (DNs) for all dimensions simultaneously, i.e. at each generation the whole 1000-dimensional sequence is evolving.

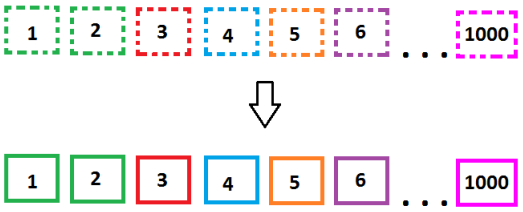


Figure 3.3.1: Scheme of Global Evolution

DNs in the GE method are represented as an array of 1000 generating matrices C as defined in (2.6). Matrix $C[j], j = 1, ..., 1000$ corresponds the jth gene. The fitness of an individual is evaluated by a weighted quality measure in the given window (see explanation in the Section 2.2.2). The weights for 2D and 3D projections are given by the formulas:

$$weight3D(C_i, C_j, C_k) = \frac{0.9^{|i-j|+|i-k|+|j-k|}}{\sum\limits_{i,j,k} 0.9^{|i-j|+|i-k|+|j-k|}} \tag{3.1}$$

$$weight2D(C_i, C_j) = \frac{0.9^{|i-j|}}{\sum_{i,j,k} 0.9^{|i-j|}} \tag{3.2}$$

Since s is fixed, the projections with the window w and corresponding weights are predefined in the arrays $Proj2D$ and $Proj3D$ of the form $(\{l_{p,k}\}, weight^{(p)})$ where p is the index of projections and $p = 1, ..., n_2$, $k = 1, 2$ for pairs and $p = 1, .., n_3$, $k = 1, 2, 3$ for triples. The pseudocode of the function $ge\_fitness()$ defining the fitness of the candidate solution, array C, is presented in Figure 3.3.2.

ge_fitness $(C, Proj2D, Proj3D, opt)$;
if $opt = 1$ **then**
  for $p = 1$ *to* n_2 **do**
    $projection \leftarrow Proj2D[p, 1]$;
    $weight \leftarrow Proj2D[p, 2]$;
    $Qm2[p] \leftarrow$ resolution$(projection) \times weight$;
  end
  for $p = 1$ *to* n_3 **do**
    $projection \leftarrow Proj3D[p, 1]$;
    $weight \leftarrow Proj3D[p, 2]$;
    $Qm3[p] \leftarrow$ resolution$(projection) \times weight$;
  end
  $fitness \leftarrow 0.6 \times \sum_{p=1}^{n_2}(Qm2[p]) + 0.4 \times \sum_{p=1}^{n_3}(Qm3[p])$;
end
else
  for $p = 1$ *to* n_2 **do**
    $projection \leftarrow Proj2D[p, 1]$;
    $weight \leftarrow Proj2D[p, 2]$;
    $Qm[p] \leftarrow t$-value$(projection) \times weight$;
  end
  $fitness \leftarrow \sum_{p=1}^{n_2}(Qm[p])$;
end
return $fitness$

Figure 3.3.2: Fitness function of GE

The process of producing a new generation in GE is involved Roulette Wheel selection of parents based on a fitness criterion. Genes of the parents are recombined in the crossover process,

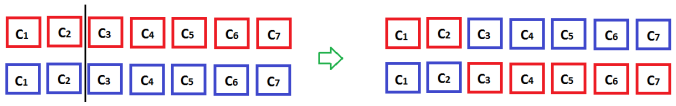


Figure 3.3.3: GE Crossover

and, then, resulting solutions have a small chance to be randomly changed by mutation.

Figure 3.3.4: GE Mutation

The mutating gene (C_7 in the Figure 3.3.4), is regenerated randomly, i.e the system randomly generates new DNs for selected dimension, converts them to a generating matrix and replaces the mutating matrix with the obtained one.

Let us now briefly go through the algorithm of Global Evolution. First of all, we generate μ initial 1000-dimensional arrays C. Then, for each generation we run crossover and mutation processes in order to produce λ offsprings. We select the fittest individuals from the union of parental and offspring population to reproduce in the new generation. The desired set of DNs is the fittest array C from the ωth population.

Parameters of GE method are presented in the third column of Table 3.1

Chapter 4

Numerical Results

In total, we conducted four experiments: Local Evolution with fitness based on resolution (rLE) and t-value (tLE) and Global Evolution with the same variations of fitness function (rGE and tGE). The sets of DNs obtained by these methods are denoted as FINALs. We call INITIALs the sets of DNs we use to evaluate the improvement achieved by the developed methods. For both variations of GE method we pick as the INITIAL the fittest candidate (array of generating matrices C) from the initial population. For LE methods the INITIAL is just a randomly generated set of DNs. For objectivity, we also compare the quality of FINALs with the set of DNs obtained by Kuo and Joe [10] which we denote KJ.

In this chapter we talk about the performance of the developed methods in four numerical tests.

4.1 Direct estimation of uniformity quality

In this test, we evaluate the t-value for each 2D projection and resolution for each 2D and 3D projections of DNs (in the form of 12×12 generating matrices) for 1000-dimensional sequence in the given window w (for explanation see Section 2.2.2) . Then, for tLE and tGE we calculate the proportion of 2D projections with $t = 0...11$. The same way we estimate the percentage of 2D projections with $resolution = 0...6$ and the percentage of 3D projections with $resolution = 0...4$, for rLE and rGE methods. We estimate KJ set using both approaches. For convenience, the results are presented in the form of diagrams.

The quality measure t-value is equal to 0 when uniformity properties are good and m (dimension of a generating matrix) when they are poor.

From the Figures 4.1.1 and 4.1.2, we can see that GE with t-value based fitness function does not demonstrate any improvement in terms of uniformity, whereas the quality of DNs obtained by tLE is noticeably better than randomly generated DNs.

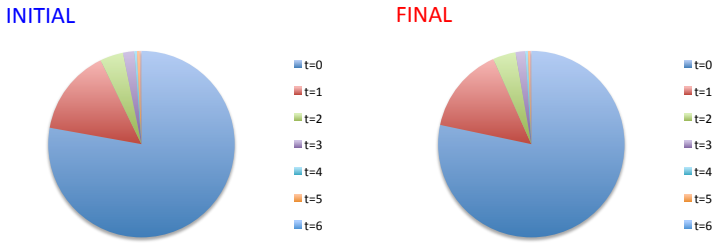


Figure 4.1.1: tGE

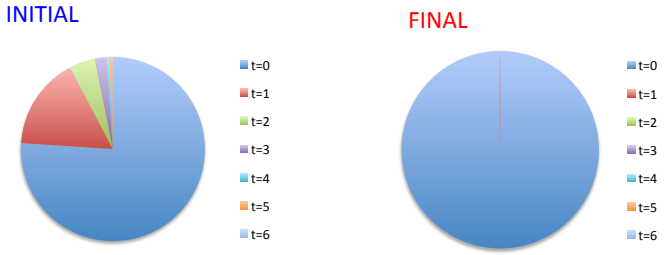


Figure 4.1.2: tLE

The uniformity quality of KJ estimated by t-value is at the same level as the quality of both INITIALs.

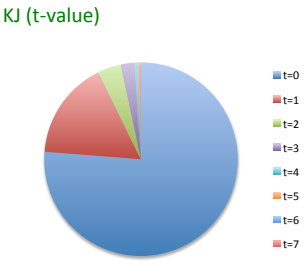


Figure 4.1.3: KJ estimated by t-value

Recall from Section 2.2.3, the larger resolution value the better uniformity properties of DNs.

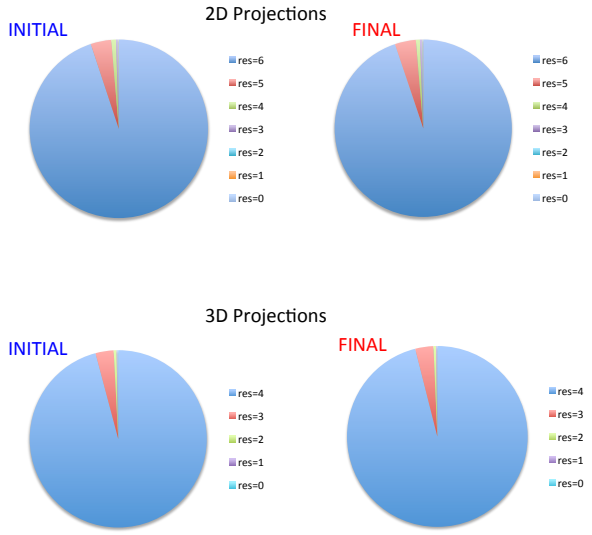


Figure 4.1.4: rGE

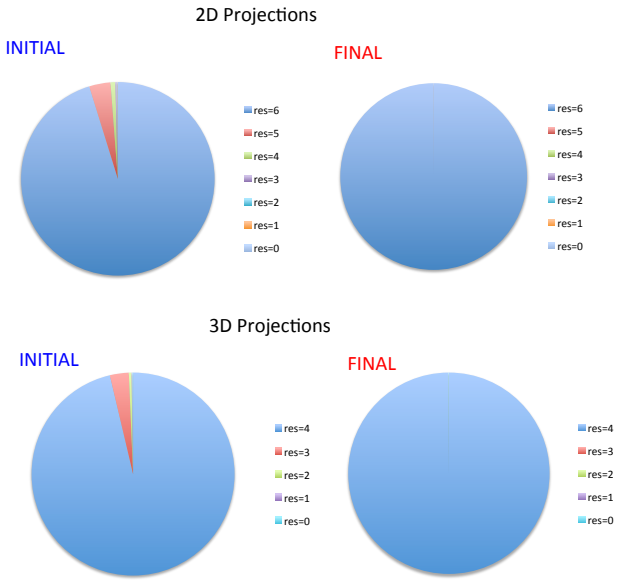


Figure 4.1.5: rLE

From Figures 4.1.4 and 4.1.5 we can observe that rGE does not perform well, the proportion of projections with high uniformity quality does not increase from INITIAL to FINAL. At the same time, we can see improvement in uniformity properties achieved by rLE. The DNs obtained by rLE also overperform KJ estimated by resolution.

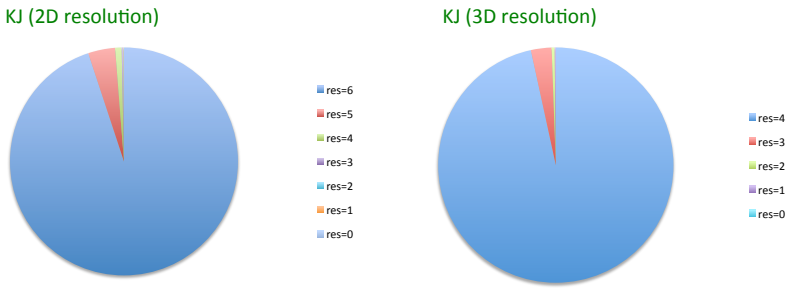


Figure 4.1.6: KJ estimated by resolution

To summarize, in this test, Local Evolution definitely overperform Global Evolution method and DNs evolved by LE have better uniformity quality than KJ parameters.

4.2 Test function g_1

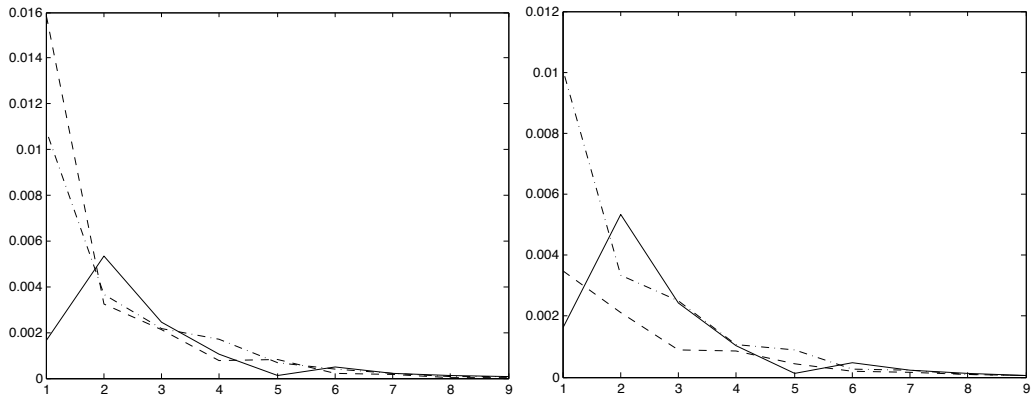
The test function g_1 is given by formula:

$$g_1(\boldsymbol{x}) = \prod_{j=1}^{s} \frac{|4x_j - 2| + (s-j+1)^2}{1 + (s-j+1)^2}, \tag{4.1}$$

where $\boldsymbol{x} = (x_1, ..., x_s)$ and $I(g_1) = 1$. In the function, the order of coordinates $\boldsymbol{x}$ is reversed, and so the influence of x_j grows as j grows. The function is useful for testing sensitivity of low-discrepancy constructions to the effective dimension in the truncation sense of the integrand [5].

In order to estimate the quality of the parameters of Sobol sequence for this function, for each DNs array we generate a set of 2^{16} points. Then, we approximate $I(g_1)$ by each of obtained sequences with 64×2^k points for $k = 1..9$. Finally, since the exact value for this function is known and equals 1, we plot the absolute error of approximation.

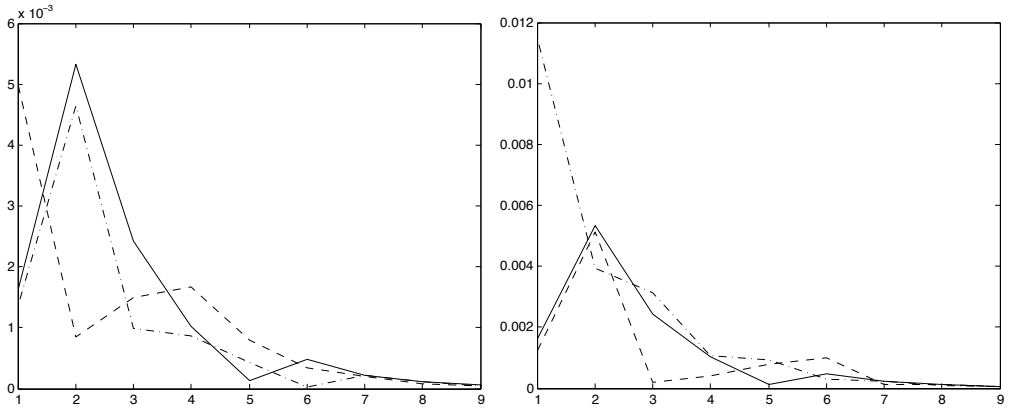
(Note: Error is on the y-axis and k is on the x-axis, KJ is $-$, INITIAL is $-\cdot-$, and $---$ represents FINAL)



(a) (b)

Figure 4.2.1: tGE & KJ (a) and tLE & KJ (b) absolute errors of approximation

Figures 4.2.1a and 4.2.1b show how well INITIAL and FINAL t-value point sets approximate g_1. It is easy to see that both point sets of GE performs roughly with the same level of accuracy. However, for LE method FINAL approximation slightly overperforms INITIAL one up to $k = 7$. The performance of KJ is better than GE, but almost the same as rLE and worse than tLE.



(a) (b)

Figure 4.2.2: rGE & KJ (a) and rLE & KJ (b) absolute errors of approximation

From Figures 4.2.2a and 4.2.2b we can see that in the given test the behavior of both FINAL point sets is unstable.

4.3 Asian call variance test

Referring to [14, p.236], let us assume that the value of one share of Google stock at time t, denoted $S(t)$, is lognormally distributed, i.e. $\ln(S(t)$ has a normal distribution with mean $\ln(S(0) + (r - \sigma^2/2)t$ and variance $\sigma^2 t$ where r is the risk-free interest rate and σ is the volatility of the stock price.

The payoff of Asian call option on this stock at time T is defined as:

$$C(T) = \max\left(\frac{1}{s}\sum_{j=1}^{s} S(t_j) - K, 0\right), \tag{4.2}$$

where K is the strike price and $0 \leq t_1 \leq ... \leq t_s = T$ are observation dates where the price of the stock $S(t)$ is recorded. The price of this option at time 0 is given by the expected value of the option's discounted payoff (Equation 4.3).

$$C(0) = \mathbb{E}(\exp^{-rT} C(T)) \tag{4.3}$$

An analytical formula for Asian call does not exist.

The asset price at time t_j has a lognormal distribution and is generated under risk-neutral measure as follows

$$S(t_j) = S(t_{j-1}) \exp^{(r-\sigma^2/2)T/s + \sigma\sqrt{T/s}\Phi^{-1}(u_j)} = S(0) \exp^{(r-\sigma^2/2)T/s + \sigma\sqrt{T/s}\sum_{i=1}^{j}\Phi^{-1}(u_i)}, \tag{4.4}$$

where $\mathbf{u} = (u_1, ..., u_s) \sim U([0,1)^s$. In quasi-Monte Carlo setting, the point $\mathbf{u}$ is instead taken from a low-discrepancy point set.

The pseudocode for the Asian call given in [14] is presented in Figure 4.3.1.

AsianCall $(r, \sigma, S(0), s, T, K, u_1, u_2, ..., u_s)$;
$a \leftarrow (r - \frac{\sigma^2}{2}) \times (\frac{T}{s})$;
$b \leftarrow \sigma \times \sqrt{\frac{T}{s}}$;
$S[0] \leftarrow S(0)$;
$sum \leftarrow 0$;
for $t = 1$ *to* s **do**
  $z \leftarrow Norm01(u_t)$;
  $S[t] \leftarrow S[t-1] \times \exp^{a+bz}$;
  $sum \leftarrow sum + S[t]$;
end
$sum \leftarrow \frac{sum}{s}$;
$C \leftarrow \exp(-rT) \times (sum - K)$;
if $C > 0$ **then**
  return C
end
else
  return 0
end

Figure 4.3.1: Asian Call algorithm

RunAllSim (U, n, m, s);
for $k = 1$ *to* m **do**
  $v \leftarrow Random01(1, s)$;
  for $i = 1$ *to* n **do**
    $u \leftarrow U[i]$;
    $w \leftarrow (u + v) \bmod 1$;
    $result[i] \leftarrow AsianCall(w)$;
  end
  $x[k] \leftarrow ave(result)$;
end
return $var(x)$

Figure 4.3.2: Simulation algorithm

In this test, we run simulations (pseudocode in Figure 4.3.2) of the Asian call option

based on shifted Sobol point sets (i.e., initial Sobol set U is shifted by random vector v) obtained in the experiments with m randomizations and estimate the total variance

$$var(x) = \sum_{k=1}^{m} \frac{(x[k] - ave(x))^2}{(m-1) \times m} \tag{4.5}$$

Table 4.1 contains the parameters for Asian call simulation.

Parameter	Value
T	1
r	0.05
σ	0.3
$S(0)$	50
K	50
s	1000
m	50
n	{1024, 4096, 1892}

Table 4.1: Parameters for Asian call simulation

n	tGE		rGE		KJ
	INITIAL	FINAL	INITIAL	FINAL	
1024	2.09e-4	1.36e-4	2.62e-4	2.66e-4	1.70e-4
4096	5.46e-5	4.22e-5	8.23e-5	8.29e-5	4.7e-5
8192	2.34e-5	2.19e-5	2.42e-5	2.28e-5	2.50e-5

Table 4.2: GE and KJ variances estimated in the Asian call test

n	tLE		rLE	
	INITIAL	FINAL	INITIAL	FINAL
1024	2.19e-4	1.18e-4	2.24e-4	2.09e-4
4096	7.06e-5	4.02e-5	7.17e-5	6.36e-5
8192	3.21e-5	1.52e-5	2.87e-5	1.90e-5

Table 4.3: LE variances estimated in the Asian call test

The variances estimated in the simulations are given in the Tables 4.2 and 4.3. We can conclude that in this test tLE performs the best since its variance dropped the most, basically cut in half from INITIAL to FINAL.

4.4 Total fitness

The quality measure L_2-discrepancy norm described in Section 2.2.1 was not used in the fitness function of the developed methods. So we can apply this measure to estimate independently the total fitness of the obtained point sets. The pseudocode of the total fitness estimation of a Sobol point set U is presented by Figure 4.4.1.

```
total_fitness (U, window);
for j = 2 to s do
    start ← j − window;
    if start < 1 then
        start ← 1;
    end
    Uw ← U[start : j − 1];
    up_window ← window;
    if j − 1 < window then
        up_window ← j − 1;
    end
    mult ← 0.9;
    for i = up_window to 1 do
        subset ← Uw[i] ∪ U[j];
        L[i] ← mult × L2norm(subset);
    end
    result[j] ← max(L);
end
return ave(result)
```

Figure 4.4.1: Algorithm of total fitness function

Tables 4.4 and 4.5 show the total fitness of Sobol point sets (with s=1000) estimated by modified L_2 discrepancy norm. For n=1024 we can observe the deterioration of uniformity properties of rGE from INITIAl to FINAL. At the same time, the first 1024 points of Kuo

and Joe sequence (KJ) are the best uniformly distributed among presented point sets. But for n=4096 KJ is the least "fit" point set based on the estimation. However, we can see that for n=4096 and n=8192 the tLE point set demonstrates the largest improvement according to L_2 discrepancy norm.

n	tGE		rGE		KJ
	INITIAL	FINAL	INITIAL	FINAL	
1024	9.82e-5	9.01e-5	8.97e-5	1.06e-4	7.95e-5
4096	4.92e-5	4.12e-5	4.59e-4	5.36e-5	8.54e-5
8192	1.77e-5	1.67e-5	3.61e-5	2.42e-5	3.19e-5

Table 4.4: Total fitness of GE and KJ point sets

n	tLE		rLE	
	INITIAL	FINAL	INITIAL	FINAL
1024	1.00e-4	8.62e-5	1.08e-4	9.23e-5
4096	5.93e-5	3.47e-5	5.61e-5	5.18e-5
8192	2.81e-5	1.45e-5	3.02e-5	1.97e-5

Table 4.5: Total fitness of LE point sets

Chapter 5

Conclusion

This research shows that Genetic Algorithms can be successfully applied for the optimization of the Sobol sequence generator. It proposes that the sequence produced by Local Evolution method based on t-value quality measure is competitive against the most recent high-dimensional sequences presented in the literature, such as Kuo and Joe sequences, for a wide range of functions. Thus, we conclude that evolutionary optimization can be considered when configuring low-discrepancy sequences, as this approach is able to walk its way through the fitness landscape of the problem and eventually produce efficient quasi-random sequence generators.

There are several ways to expand the current work. Taking into account the differences in accuracy and complexity of the quality measures discussed in the paper, someone can construct the fitness function based not on a single estimator but on the combination of them and apply it in the evolution turn of the Sobol sequence. The other possibility would be to apply the methods developed in this research for optimization of other widely used parametrized sequences such as Faure sequences. Finally, a promising research work can be done in the analysis of Sobol parameters domain. In this case someone may use Genetic Algorithms as a tool to explore the changes in the proportion of 'optimal' parameters with the growth of dimension.

Bibliography

[1] I.A. Antonov and V.M. Saleev. *An economic method of computing LP_τ-sequences.* .S.S.R Comput. Maths. Math. Phys. 8, p.252-256, 1979.

[2] E. Braaten and G. Weller. *An improved low-discrepancy sequence for multidimensional QMC integration.* J. Comput. Phys., 33, 2, p. 249-258, 1979.

[3] P. Bratley and B. L. Fox. *Algorithm 659: Implementing Sobol quasirandom sequence generator.* ACM Trans. Math. Software 14, p.88-100, 1988.

[4] Newcastle University Engineering Design Centre. Roulette wheel approach: based on fitness. http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php. article "Roulette wheel selection".

[5] H. Faure and C. Lemieux. *Generalized Halton sequences in 2008: A comparative study.* ACM Transactions on Modeling and Computer Simulation, 19 (4), 2009.

[6] P. Glasserman. *Monte-Carlo Methods in Financial Enginerring.* Springer -Verlag, New York, 2004.

[7] J .H. Halton. *On the efficiency of certain quasi-random sequences of points in evaluating multidimensional integrals.* Numerische Mathematik, 2, p. 84-90, 1960.

[8] F.J. Hickernell. *A generalized discrepancy and quadrature error bound.* Math. Comput., 67(221), p.299-322, 1998.

[9] J .H. Holland. *Adaptation in Natural and artificial Systems.* University of Michigan Press, An Arbor, MI, 1975.

[10] S. Joe and F.Y. Kuo. Sobol sequence generator. http://web.maths.unsw.edu.au/~fkuo/sobol/, 2008. (Recommended choice of parameters).

[11] P L'Ecuyer and C. Lemieux. *Variance reduction via lattice rules.* Manag. Sci. 46(9), p.1214-1235, 2000.

[12] P. L'Ecuyer and W.C. Shmidt. *Maximally equidistributed combined Tausworthe generators.* Mathematics of Computation, 65(213), p.203-213, 1996.

[13] C. Lemieux. Randomqmc. http://www.math.uwaterloo.ca/~clemieux/randqmc.html, 2004. Online, accessed on August 4th 2013.

[14] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling.* Springer, 2009.

[15] H. Niederreiter. *Point sets and sequences with small discrepancy.* Monatshefte fur Mathematik, 104, p.237-337, 1987.

[16] H. Niederreiter. *Quasi-Monte Carlo methods and pseudo-random numbers.* Bull. Amer. Math. Soc. 84 (6), p.957-1041, 1987.

[17] H. Niederreiter. *Low discrepancy point sets obtained by digital constructions over finite fields.* CBMS-NSF Regional Conference Series in Applied Mathematics, 63, SIAM, 1992.

[18] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* Czechoslovak Mathematical Journal, 42, p.143-166, 1992.

[19] G. Pirsic and W.C. Shmidt. *Calculation of the Quality Parameter of Digital nets and Application to Their Construction.* Journal of complexity, 17, p.827-839, 2001.

[20] Teytaud Rainville, Gagné and Laurendeau. *Evolutionary optimization of low-discrepancy sequences.* ACM Transactions on Modeling and Computer Simulation, 2 (22), Article 9, 2012.

[21] I.M. Sobol. *Distribution of points in a cube and approximate evaluation of integrals.* U.S.S.R Comput. Maths. Math. Phys. 7, p.86-112, 1967.

[22] S. Tezuka. *Uniform Random Numbers: Theory and Practice.* Kluwer Academic Publishers, Norwell, MA, 1995.

[23] Lancaster university Renewable Energy Group. A diagram of the operation of a simple genetic algorithm. http://www.engineering.lancs.ac.uk/lureg/group_research/wave_energy_research/Collector_Shape_Design.php. article "Optimization of Collector: Form and Response".