Nonlinear Analysis 71 (2009) e679–e692

Contents lists available at ScienceDirect

Nonlinear Analysis

journal homepage: www.elsevier.com/locate/na

Random number generators in genetic algorithms for unconstrained and constrained optimization

Andrea Reese *

School of Mathematics, Daytona State College, 1200 W. International Speedway Blvd., Daytona Beach, FL 32114, United States

ARTICLE INFO

MSC:
90C30

Keywords:
Genetic algorithm
Pseudo-random number
Quasi-random number
Constrained optimization
Unconstrained optimization
Relative error

ABSTRACT

Presented here is a genetic algorithm that computes an approximate solution to constrained and unconstrained global optimization problems. This technique has been implemented using several pseudo- and quasi-random number generators and the results of several test examples are presented. The performance of this technique is based on a ranked comparison of relative error.

1. Introduction

A genetic algorithm (GA) is a search technique used to find approximate solutions to optimization and search problems. The search methods model some natural phenomena that are based on the mechanics of natural selection and natural genetics. John Holland is credited with the invention of GAs during the 1960s. His GA is a population-based algorithm and was seen as a major innovation. His method moves one population to a new population using natural selection together with crossover, mutation, and inversion [1].

Genetic algorithms differ from other optimization and search procedures in the following ways: they work with a coding of the parameter set (not the parameters themselves); they search from a population of points (not a single point); they use payoff information (not derivatives and other auxiliary information); and they use probabilistic transition rules (not deterministic rules) [2].

A genetic algorithm must have the following five components: a genetic representation for potential solutions to the problem, a way to create an initial population of potential solutions, an evaluation function that rates solutions in terms of their 'fitness' (this function is to be maximized),[1] genetic operators that alter the composition of future populations, and values for various parameters that the GA uses [3]. There are endless variations on this basic idea. A simple GA that yields good results in many practical problems is composed of the three operators: reproduction, crossover, and mutation [2].

There are many areas in which genetic algorithms have been utilized: biotechnology, computing, data mining, electrical engineering, finance, image processing, modeling, optimization, pattern recognition, and signal processing. This list is not exhaustive but shows the scope and importance of GAs.

The scope of this paper will be to highlight the effect of random number generators on the outcome of the GA. Several pseudo- and quasi-random number generators will be substituted as the parent population.

A comparison of the accuracy of the solution of the test functions is conducted. A ranking is attempted based on accuracy/relative error. An estimate of relative error is included for each test function.

* Tel.: +1 386 492 5681.
E-mail address: ReeseA@DaytonaState.edu.

[1] Throughout, optimization will refer to maximization without loss of generality. Minimization problems can easily be converted to a maximization problem since minimizing a function is the same as maximizing the negative of the function, i.e., $\min f = \max -f$.

0362-546X/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.na.2008.11.084

The outline of the genetic algorithm is given in Section 2, while the random number generators are briefly discussed in Section 3. Test examples and rankings are presented in Section 4 and 5, respectively. The conclusion appears in Section 6.

2. The algorithm

The first step of a genetic algorithm is to generate an initial population of candidate solutions (known as the parent population) by coding the parameter set as a finite-length binary string where each bit is assigned a meaning. The bit string is the concept definition language for the GA [4]. This binary string represents a real value of the variable.

How should one choose the population size? It can be set anywhere from 10 to 1000,000. It has been suggested that for smaller problems, a population size of 1000 should be used; however, as the problem becomes more difficult, the population size should increase [4].

To generate the parent population, an array of random numbers is generated and then converted to binary strings. Each binary string will contain 22 bits so as to assure the required precision of six places after the decimal point.[2]The binary strings are then converted to base 10 representation and then into a real number.

To illustrate this procedure, suppose the random number 0.9501 was generated. This value is then multiplied by 2^{22} to ensure 22 bits in the binary conversion. The decimal to binary conversion results in the string '1111001100111001110000'. The conversion from binary to base 10 results in 3985008. Lastly, the conversion to a real number requires the endpoints of the interval for the variable. Suppose a value between -1 and 2 is to be generated. Then the real number conversion is found by multiplying the base 10 representation by the length of the interval, dividing by $(2^{22} - 1)$, and then adding the left endpoint of the interval. In this case, $x = -1 + 3985008 \times 3/(2^{22} - 1) = 1.8503$.

Starting with the parent population of strings, the GA then generates successive populations of strings. This simulated evolution allows the 'good' strings to reproduce and the 'bad' strings to die off (through the reproduction operator). The search for better structures is based solely on the fitness values associated with the individual strings and is guided by probabilistic transition rules.

Reproduction (also known as selection) is a process in which individual strings from the parent population are copied according to their objective function values (i.e., payoff values) and is the second step of the GA. It is an asexual process in that only a single string is involved [5]. Holland's method uses fitness-proportionate selection with 'roulette wheel' [1]. In this method, the chance of an individual string being selected is proportional to the amount by which its fitness is greater or less than its competitor's fitness [6] causing stronger strings to be selected over weaker strings. This method will produce a new generation of the same size as the parent population.

Holland's method is conducted as follows. First, the absolute value[3] of the fitness value of each parent string is calculated. From this the total fitness of the entire population can be found by summing the individual fitness values. Then for each parent string, the probability of selection is calculated by dividing the individual fitness value by the total fitness. Lastly, the cumulative probability distribution is calculated. To select parent strings to copy, a random number between 0 and 1 (inclusive) is first generated. Using the cumulative probability distribution, if the generated random number lies within a parent string's cumulative probability range then that parent string is chosen for reproduction. For example, if the generated random number is less than the first value of the cumulative probability distribution, then the first chromosome will be selected. This random number generation is repeated until a new generation of the same size as the parent generation has been created.

The third step of the GA is crossover (also known as partial string exchange). Crossover combines the features of two parent strings to form similar (i.e., of the same size) offspring by swapping corresponding segments of the parents [3]. The expected number of parent strings to crossover can be found by multiplying the population size by the probability of crossover (which is chosen by the user). According to Mitchell [1], the probability of crossover can be set at the 'fairly typical' value of 0.7. However, Banzhaf et al. [4] suggest that the crossover probability should be set 'high', around 90%. Obitko and Slavik [7] also suggest that the crossover rate should be set high, about 80%–95%.

To generate the set of strings to be involved in crossover, a random number is generated for each string. If the random number is less than the probability of crossover, the chromosome is selected for crossover. If an odd number of strings are chosen to crossover, another string can be added or an existing string can be deleted. Strings are then paired together to undergo crossover. For each pair, a random number between 1 and (the total number of bits $-$ 1) inclusive is generated (in this case between 1 and 21 inclusive). Crossover will occur at the bit after the generated random number. This is called single point crossover and is the simplest method.

To illustrate, suppose we have the following two strings selected for crossover:

00111010011111100110111 and 11000000110101111000100.

If the generated random number was 5, crossover would occur after the fifth bit:

00111|01001111100110111 and 11000|00011010111000100.

[2] For the one variable test problem, the variable x lies in the interval $[-1, 2]$. Therefore, to obtain six places after the decimal point, the range should be divided into $3 \times 1000{,}000$ equal intervals. This will require 22 bits, as $2^{21} = 2097{,}152 < 3000{,}000 < 4194{,}304 = 2^{22}$.

[3] The absolute value allows functions resulting in negative values to be considered.

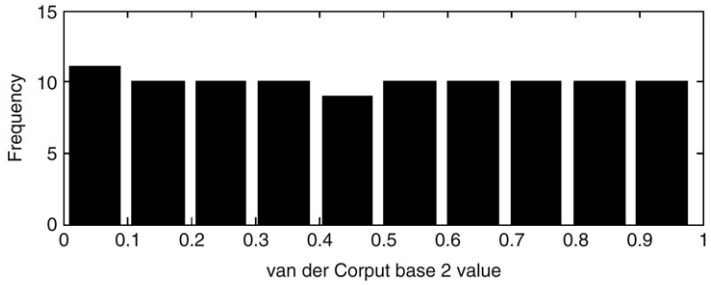

Fig. 3.1. Frequency distribution of the first 100 points generated by a QRNG.

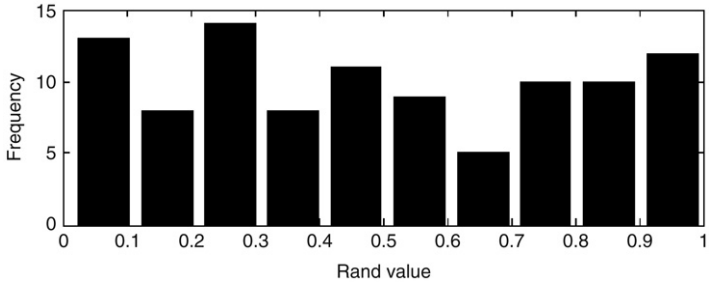

Fig. 3.2. Frequency distribution of the first 100 points generated by a PRNG.

Each part after the crossover position is exchanged to form two new offspring:

0011100011010111000100 and 11000**01001111100110111**.

Other crossover methods include: two point crossover (where two crossover points are selected and the binary string from the beginning of the chromosome to the first crossover point is copied from the first parent and the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent), uniform crossover (where bits are randomly copied from the first or from the second parent), and arithmetic crossover (where some arithmetic operation is performed to make a new offspring) [7].

The last step in the GA is mutation. Mutation plays a secondary role in the operation of GAs [2]. It is the occasional random alteration of the value of a string position. Mutation, like reproduction, is an asexual process. The expected number of mutated bits can be found by multiplying the probability of mutation (this is also chosen by the user) by the total number of bits (in this case, 22) by the population size. According to Mitchell [1], the probability of mutation can be set at a 'fairly typical' value of 0.001. However, Banzhaf et al. [4] suggest that the mutation probability should be set 'low', around 10%. Obitko and Slavik [7] also suggest the mutation rate should be set low, about 0.5%–1%.

The procedure for mutation is as follows. For every bit in the population, a random number between 0 and 1 (inclusive) is generated. If the generated random number is less than the probability of mutation, the bit is mutated. If the original bit was '0', it now becomes a '1', and vice versa. Mutation and crossover can be performed in either order. However, it has been shown that mutation before crossover is less efficient than mutation after crossover [8].

After crossover and mutation have been performed, the first generation (after the parent population) has been formed. The fitness values of this newly formed generation are then evaluated. The GA should be repeated starting with reproduction using this first generation as the new parent population. The total number of generations to evaluate is often based on previous results and can range anywhere from 50 to more than 500 [1].

One option for termination is to run the algorithm for a set number of generations (which is the approach used here). An optional approach is to end the algorithm after a certain number of generations pass with no improvement in the fitness of the best individual in the population [9].

3. Random numbers in GAs

There are two types of random number generators (RNGs): pseudo-random number generators (PRNGs) and quasi-random number generators (QRNGs). The objective of PRNGs is to look like a sequence of independent and identically distributed uniform random variables. The objective of QRNGs is to fill a unit hyper-cube as uniformly as possible (resulting in no clumps or gaps). Figs. 3.1 and 3.2 give a graphical comparison of the types of generators.

Within each test problem, four generators of each type will be used and compared. The PRNGs are called Matlab RAND, R250, Simscript, and Whlcg. The QRNGs are called Halton, Faure, Sobol, and Niederreiter. For further details on the generation of these sequences, see Sen et al. [10] and Lakshmikantham et al. [11].

The four components that are nondeterministic and use randomization in GAs are initialization, reproduction, crossover, and mutation. Through the experiments conducted by Cantu-Paz [12], it has been shown that the pseudo-random number generator used to initialize the population is critical; however, the PRNG used as input to other operations does not affect the performance significantly. However, the results by Meysenburg and Foster [13] suggest that the quality of the PRNG chosen for a GA has relatively little effect on the performance of the algorithm. In a follow-up study, Meysenburg and Foster [14] concluded that different PRNGs caused different levels of GA performance (both better and worse) depending on the test problem.

Maaranen et al. [15] found that the use of quasi-random number sequences as the parent population (specifically Sobol and Niederreiter) improved the final objective function value and the number of generations used. The results provided by Kimura and Matsumura [16] showed that the use of the QRNGs of van der Corput and Halton enhanced the search performance of the GA.

For each test problem considered here, the only randomized component that was altered was the initialization of the parent population. All other random components (reproduction, crossover, and mutation) use Matlab RAND. For the Halton and Faure QRNGs, the second dimension is used since the first dimension of these sequences is identical.

4. Numerical experiments

To evaluate the test problems (many of which are violently fluctuating, i.e., containing many local maxima), the n-dimensional bisection method will be implemented. This method is most commonly associated with root-finding, but it can also be used to show that a continuous function on a closed interval achieves its maximum [17]. Bisection is the division into two equal halves of a given curve, figure, or interval.

The domain for one dimension (i.e., one variable) is a line segment. The one-dimensional bisection procedure for iteratively converging on a solution which is known to lie inside some interval $[a, b]$ proceeds by evaluating the function at the midpoint of the original interval $x = (a + b)/2$ and testing to see in which of the subintervals $[a, (a + b)/2]$ or $[(a + b)/2, b]$ the solution lies. This procedure is then repeated with the new interval as often as needed to locate the solution to the desired accuracy.

For two dimensions (i.e., two variables), the domain is a regular hexagon. The bisection method will produce four search areas. For example, if the first variable lies between $[a, b]$ and the second variable lies between $[c, d]$, then the bisection method will produce two subintervals for each variable: $[a, (a + b)/2]$ and $[(a + b)/2, b]$ for the first variable, and $[c, (c + d)/2]$ and $[(c + d)/2, d]$ for the second variable. There are four possible combinations since there are two options for each variable. The combination resulting in the largest function value will be chosen for the next iteration. For three dimensions, there would be eight subintervals in which to choose one for the next iteration. This procedure can be generalized for higher dimensions.

The bisection method will terminate when the relative error is less than 0.005%. The relative error is calculated as the absolute value of the proposed solution minus the actual solution divided by the absolute value of the actual solution, i.e.,$(|f_{\text{proposed}} - f_{\text{actual}}|)/|f_{\text{actual}}|$.

The parameters of the GA are set at 512 for population size, 10 for the number of generations, 0.80 for the crossover rate, and 0.001 for the mutation rate. The GA was run 30 times and the maximum function value is reported.

4.1. Unconstrained global optimization problems

Unconstrained global optimization problems have the form:

$$\max f(x)$$
$$\text{subject to } x \in [a, b]$$

where $f(x)$ is the objective function and a and b are the endpoints of the search interval for each variable.

Test function 1: *A unimodal one variable problem*

Using a test problem obtained from *The GA Playground* (http://www.aridolan.com/ofiles/ga/gaa/gaa.aspx#Examples), the goal is to find x in the closed interval from -3 to 3 that maximizes $f(x) = -x^4 + 12x^3 - 15x^2 - 56x + 60$. Fig. 4.1.1 shows the graph of this function over the domain of x.

To obtain the precision requirement of six places after the decimal point, x requires 23 bits ($2^{22} = 4194{,}304 < 6 \times 1000{,}000 = 6000{,}000 < 8388{,}608 = 2^{23}$).

The known solution is found at $x = -0.8702$ resulting in a function value of 88.891568. The initial results using the different RNGs as the parent population are given in Table 4.1.1.

From Table 4.1.1, it is clear that the relative error requirement was met in all cases (as indicated in bold throughout). Therefore the bisection method does not need to be implemented. Using the relative error, the generators can be ranked as follows: Niederreiter, Halton, R250, Matlab RAND, Whlcg, Simscript, and Faure and Sobol (tied).

Test function 2: *A multimodal one variable problem with one global maximum*

Using a test problem provided by Goldberg [2], the goal is to find x in the closed interval from -1 to 2 that maximizes $f(x) = 1.0 + x \sin(10\pi x)$. Fig. 4.1.2 shows the graph of this function over the domain of x.

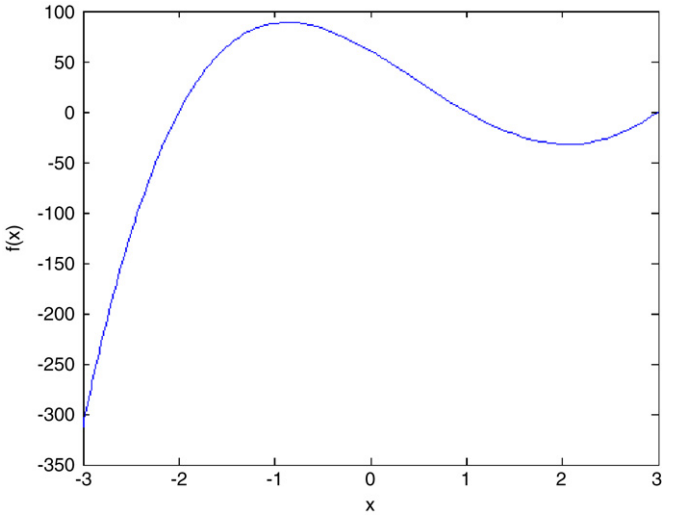

Fig. 4.1.1. Graph of $f(x) = -x^4 + 12x^3 - 15x^2 - 56x + 60$ over $[-3, 3]$.

Table 4.1.1
Initial GA results for $f(x) = -x^4 + 12x^3 - 15x^2 - 56x + 60$ over $[-3, 3]$.

RNG	x	$f(x)$	Relative error (%)	Ranking
Halton (dim = 2)	−0.869857	88.891563	**0.000006**	2
Faure (dim = 2)	−0.867187	88.891115	**0.000510**	7
Sobol	−0.867187	88.891115	**0.000510**	7
Niederreiter	−0.869992	88.891566	**0.000002**	1
Matlab RAND	−0.870583	88.891559	**0.000010**	4
R250	−0.870562	88.891560	**0.000009**	3
Simscript	−0.873081	88.891137	**0.000484**	6
Whlcg	−0.871128	88.891521	**0.000052**	5

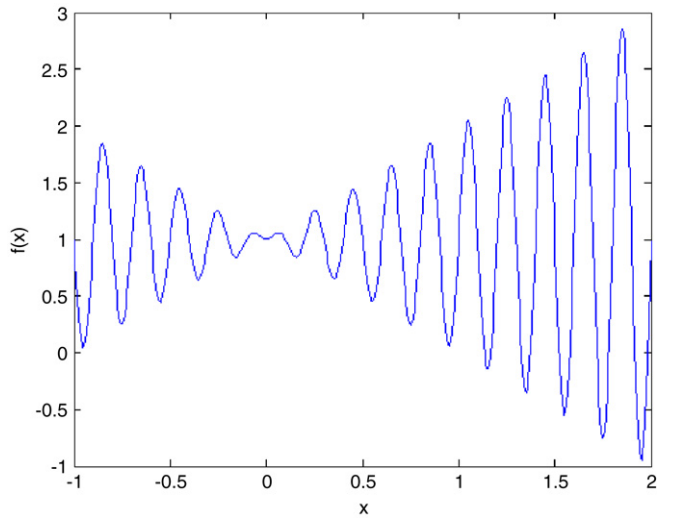

Fig. 4.1.2. Graph of $f(x) = 1.0 + x\sin(10\pi x)$ over $[-1, 2]$.

The known solution is found at $x = 1.8508$ resulting in a function value of 2.850227. The results for this test problem using the different RNGs as the parent population are given in Table 4.1.2.

It is clear from the table that the global maximum was found in all cases and therefore the bisection method is not necessary here. Further, the relative error is less than 0.005% in all cases. Based on the relative error, the ranking of the generators is: Faure and Sobol (tied), Niederreiter, R250 and Simscript (all tied), Halton and Whlcg (tied), and Matlab RAND. (The relative error has been rounded to six decimal places; however, the ranking is based on 14 decimal places.)

Table 4.1.2
Initial GA results for $f(x) = 1.0 + x\sin(10\pi x)$ over $[-1, 2]$.

RNG	x	$f(x)$	Relative error (%)	Ranking
Halton (dim = 2)	1.850548	2.850274	**0.001641**	6
Faure (dim = 2)	1.850586	2.850272	**0.001593**	1
Sobol	1.850586	2.850272	**0.001593**	1
Niederreiter	1.850547	2.850274	**0.001641**	3
Matlab RAND	1.850547	2.850274	**0.001641**	8
R250	1.850547	2.850274	**0.001641**	3
Simscript	1.850547	2.850274	**0.001641**	3
Whlcg	1.850548	2.850274	**0.001641**	6

Table 4.1.3
Initial GA results for $f(x_1, x_2) = x_1\sin(\sqrt{|x_1|}) + x_2\sin(\sqrt{|x_2|})$.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	420.569975	420.668662	837.934351	**0.003750**	5
Faure (dim = 2)	420.898438	−500.000000	599.571422	28.449170	8
Sobol	420.959474	−500.000000	599.572035	28.449102	7
Niederreiter	420.916932	421.096803	837.963366	**0.000287**	1
Matlab RAND	421.033968	420.493365	837.936727	**0.003466**	4
R250	420.738733	420.895100	837.958415	**0.000878**	2
Simscript	421.495918	420.956042	837.930680	**0.004188**	6
Whlcg	421.322110	420.999766	837.949895	**0.001895**	3

Table 4.1.4
Initial GA results for $f(x_1, x_2) = \cos(x_1) \times \cos(x_2) \times e^{-[(x_1-\pi)^2+(x_2-\pi)^2]}$.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	3.648768	3.058285	0.668861	33.113886	5
Faure (dim = 2)	5.078125	0.0000004	0.0000004	99.999957	8
Sobol	5.078125	0.195313	0.000001	99.999860	7
Niederreiter	2.858898	2.852631	0.781718	21.828187	3
Matlab RAND	3.086550	3.054154	0.984109	1.589074	1
R250	3.256237	2.714891	0.743976	25.602365	4
Simscript	2.595938	3.428048	0.560851	43.914938	6
Whlcg	3.037817	3.264571	0.961877	3.8122700	2

Test function 3: *A multimodal two variable problem with one global maximum*

This two variable test problem is known as Schwefel's function and was also obtained at *The GA Playground*. The function is characterized by a global maximum that is geometrically distant from the next best local maxima. Therefore, search algorithms are prone to convergence in the wrong direction. The function is defined as $f(x_1, x_2) = x_1\sin(\sqrt{|x_1|}) + x_2\sin(\sqrt{|x_2|})$ over the closed interval of $[-500, 500]$ for both x_1 and x_2.

To obtain the precision requirement of six places after the decimal point, x_1 and x_2 each require 30 bits ($2^{29} = 536{,}870{,}912 < 1000 \times 1000{,}000 = 1000{,}000{,}000 < 1073{,}741{,}824 = 2^{30}$). Therefore, the parent population will contain strings of $30 + 30 = 60$ bits.

The known solution for this test problem is found at $x_1 = x_2 = 420.9687$, resulting in a function value of $f(x_1, x_2) = 837.965775$. Table 4.1.3 shows the results of the initial run for the different RNGs.

From Table 4.1.3, it is clear that the random number generators Halton, Niederreiter, Matlab RAND, R250, Simscript, and Whlcg all produce a maximum function value close to the global maximum. Further, all generators except Faure and Sobol have an acceptable relative error less than 0.005%. The Faure and Sobol sequences produce results that are significantly lower and therefore are finding local maxima, as opposed to the global maximum. These generators perform poorly and are not recommended for use as the parent population. A ranking of the remaining generators based on relative error gives: Niederreiter, R250, Whlcg, Matlab RAND, Halton, and Simscript.

Test function 4: *A unimodal two variable problem*

Easom's function (obtained at *The GA Playground*) is unimodal; however, the area surrounding the global maximum is small in relation to the entire search space. The function is defined as $f(x_1, x_2) = \cos(x_1) \times \cos(x_2) \times e^{-[(x_1-\pi)^2+(x_2-\pi)^2]}$. Both x_1 and x_2 lie within the closed interval of $[-100, 100]$.

To obtain the precision requirement of six places after the decimal point, x_1 and x_2 each require 28 bits ($2^{27} = 134{,}217{,}728 < 200 \times 1000{,}000 = 200{,}000{,}000 < 268{,}435{,}456 = 2^{28}$). Therefore, the parent population will contain strings of $28 + 28 = 56$ bits.

The known solution is at the point (π, π), resulting in a function value of 1.000000. The initial results are shown in Table 4.1.4.

Table 4.1.5
Seventh bisection results for $f(x_1, x_2) = \cos(x_1) \times \cos(x_2) \times e^{-[(x_1-\pi)^2+(x_2-\pi)^2]}$.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	3.142094	3.141289	0.999999	**0.000052**	2
Faure (dim = 2)	3.140259	3.137207	0.999968	**0.003152**	8
Sobol	3.140640	3.137207	0.999970	**0.003021**	7
Niederreiter	3.139949	3.140378	0.999994	**0.000627**	6
Matlab RAND	3.140284	3.142056	0.999997	**0.000289**	5
R250	3.140895	3.140742	0.999998	**0.000181**	3
Simscript	3.141738	3.141972	0.9999998	**0.000025**	1
Whlcg	3.142797	3.141665	0.999998	**0.000218**	4

Table 4.1.6
Initial GA results for Branin's rcos function.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	−3.142254	12.229782	−0.400080	0.551236	3
Faure (dim = 2)	2.880860	3.750000	−2.317300	482.401515	7
Sobol	2.763672	3.750000	−2.425307	509.546693	8
Niederreiter	3.142586	2.274885	−0.397893	**0.001391**	1
Matlab RAND	−3.122581	12.251327	−0.400105	0.557546	4
R250	9.416882	2.467061	−0.398188	0.075728	2
Simscript	3.096712	2.356910	−0.409733	2.977148	6
Whlcg	9.442635	2.429075	−0.403143	1.320941	5

The results provided in Table 4.1.4 show that only the random number generators Matlab RAND and Whlcg come close to finding the global maximum, although the relative error requirement is not met in either of these cases. The bisection method was therefore implemented. The intermediate bisection results are omitted. The seventh bisection, which uses the search interval of [3.125, 4.6875] for both x_1 and x_2, allowed all of the generators to meet the 0.005% relative error condition. The variable, function value, relative error, and rankings for this bisection are given in Table 4.1.5. Based on the relative error, the random number generators are ranked as follows: Simscript, Halton, R250, Whlcg, Matlab RAND, Niederreiter, Sobol, and Faure.

Test function 5: *A two variable problem with multiple global maxima*

This two variable test problem is known as Branin's rcos function and was obtained at *The GA Playground*. This function contains multiple global maxima. The function is defined as $f(x_1, x_2) = -a(x_2 - bx_1^2 + cx_1 - d)^2 - e(1 - f)\cos(x_1) - e$, where $a = 1$, $b = \frac{5.1}{4\pi^2}$, $c = \frac{5}{\pi}$, $d = 6$, $e = 10$, and $f = \frac{1}{8\pi}$. The x_1 variable is defined on the closed interval $[-5, 10]$ and x_2 on $[0, 15]$.

To obtain the precision requirement of six places after the decimal point, x_1 and x_2 each require 24 bits ($2^{23} = 8388{,}608 < 15 \times 1000{,}000 = 15{,}000{,}000 < 16{,}777{,}216 = 2^{24}$). Therefore, the parent population will contain strings of 24+24 = 48 bits.

There are three global maxima located at the points $(-\pi, 12.275)$, $(\pi, 2.275)$, and $(9.4248, 2.475)$. Each point results in a function value of $f(x_1, x_2) = -0.397887$. Table 4.1.6 shows the results of the initial run for the different random number generators.

From Table 4.1.6, it is interesting to note that different global maxima were reached depending on the random number generator. Halton and Matlab RAND are aiming towards $(-\pi, 12.275)$, while Niederreiter and Simscript are closer to $(\pi, 2.275)$. R250 and Whlcg are closest to $(9.42478, 2.475)$. The remaining generators of Faure and Sobol are not initially reaching towards any of the global maxima. Only the generator of Niederreiter is below the required precision level, so the bisection method will be implemented.

The intermediate bisection results are omitted. The sixth bisection, which uses the search interval of [2.9688, 3.2031] for x_1 and [2.1094, 2.3438] for x_2, allowed all of the generators except Faure and Sobol to meet the 0.005% relative error condition. These generators should not be used as the parent population. The variable, function value, relative error, and rankings for this bisection are given in Table 4.1.7. Based on the relative error, the random number generators are ranked as follows: Niederreiter, Matlab RAND, Whlcg, Halton, R250, Simscript, Faure, and Sobol.

Test function 6: *An eight variable unimodal problem*

This eight variable test problem is known as the sum of different powers and was obtained at *The Genetic and Evolutionary Algorithm Toolbox* (http://www.geatbx.com). It is a commonly used unimodal function and is defined as $f(x) = \sum |x_i|^{i+1}$, $i = 1, \ldots, 8$. The x_i variables are defined on the closed interval from −1 to 1.

To obtain the precision requirement of six places after the decimal point, each x_i require 21 bits ($2^{20} = 1048{,}576 < 2 \times 1000{,}000 = 2000{,}000 < 2097{,}152 = 2^{21}$). Therefore, the parent population will contain strings of $21 \times 8 = 168$ bits.

The global maximum is located at the point $x_i = 0$ resulting in a function value of $f(x_i) = 0$. Initial and intermediate bisection results are omitted. The results of the fifth bisection which uses the search interval of [0, 0.0625] for each x_i are shown in Table 4.1.8 for the QRNGs and Table 4.1.9 for the PRNGs. Each table includes the variable values, function value, absolute relative error, and rankings for this bisection.

Table 4.1.7
Sixth bisection results for Branin's rcos function.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	3.142259	2.274014	−0.397890	**0.000681**	4
Faure (dim = 2)	3.112522	2.226600	−0.407011	2.293086	7
Sobol	3.185757	2.226600	−0.407452	2.403976	8
Niederreiter	3.141528	2.275720	−0.397888	**0.000208**	1
Matlab RAND	3.141987	2.273565	−0.397889	**0.000597**	2
R250	3.141607	2.2718752	−0.397897	**0.002532**	5
Simscript	3.142866	2.2708510	−0.397905	**0.004551**	6
Whlcg	3.140970	2.2748833	−0.397890	**0.000648**	3

Table 4.1.8
Fifth bisection results for sum of different powers' function for QRNGs.

QRNG	Halton	Faure	Sobol	Niederreiter
x_1	0.003344	0.001709	0.005005	0.004728
x_2	0.019805	0	0	0.004737
x_3	0.040380	0	0	0
x_4	0	0.003906	0	0
x_5	0	0.0000002	0	0
x_6	0	0	0	0
x_7	0	0	0	0
x_8	0	0	0	0
$f(x_i)$	0.000022	0.000003	0.000025	0.000022
Relative error	**0.002161**	**0.000292**	**0.002505**	**0.002246**
Ranking	2	1	4	3

Table 4.1.9
Fifth bisection results for sum of different powers' function for PRNGs.

PRNG	Matlab RAND	R250	Simscript	Whlcg
x_1	0.001498	0.000602	0.004818	0.001012
x_2	0.007772	0.024272	0.007817	0.030726
x_3	0.006683	0.053366	0.055061	0.005127
x_4	0	0	0	0
x_5	0	0	0	0
x_6	0	0	0	0
x_7	0	0	0	0
x_8	0	0	0	0
$f(x_i)$	0.000003	0.000023	0.000033	0.000030
Relative error	**0.000272**	**0.002277**	**0.003288**	**0.003003**
Ranking	1	2	4	3

The overall ranking of all of the random number generators is: Matlab RAND, Faure, Halton, Niederreiter, R250, Sobol, Whlcg, and Simscript.

Test function 7: *A ten variable unimodal problem*

This ten variable test problem is known as the axis parallel hyper-ellipsoid (or weighted sphere model) and was also obtained at *The Genetic and Evolutionary Algorithm Toolbox*. It is a unimodal function and is defined as $f(x) = \sum i \cdot x_i^2$, $i = 1, \ldots, 10$. The x_i variables are defined on the closed interval from −5.12 to 5.12.

To obtain the precision requirement of six places after the decimal point, each x_i require 24 bits ($2^{23} = 8388{,}608 < 10.24 \times 1000{,}000 = 10{,}240{,}000 < 16{,}777{,}216 = 2^{24}$). Therefore, the parent population will contain strings of $24 \times 10 = 240$ bits.

The global maximum is located at the point $x_i = 0$ resulting in a function value of $f(x_i) = 0$. Initial and intermediate bisection results are omitted. The results of the eleventh bisection which uses the search interval of [0, 0.005] for each x_i are shown in Table 4.1.10 for the QRNGs and Table 4.1.11 for the PRNGs. Each table includes the variable values, function value, absolute relative error, and rankings for this bisection.

The overall ranking of all of the RNGs is: Sobol, Faure, Niederreiter, Simscript, Whlcg, Halton, R250, and Matlab RAND.

4.2. Constrained global optimization problems

Constrained global optimization problems have the form:

$$\max f(x)$$
$$\text{subject to } a(x) < b$$
$$x \in [c, d]$$

Table 4.1.10
Eleventh bisection results for axis parallel hyper-ellipsoid function for QRNGs.

QRNG	Halton	Faure	Sobol	Niederreiter
x_1	0.002723	0.003076	0.002363	0.002731
x_2	0.002229	0	0	0.001577
x_3	0.002188	0	0	0
x_4	0	0	0	0
x_5	0	0	0	0
x_6	0	0	0	0
x_7	0	0	0	0
x_8	0	0	0	0
x_9	0	0	0	0
x_{10}	0	0	0	0
$f(x_i)$	0.000032	0.000009	0.000006	0.000012
Relative error	**0.003171**	**0.000946**	**0.000559**	**0.001244**
Ranking	4	2	1	3

Table 4.1.11
Eleventh bisection results for axis parallel hyper-ellipsoid function for PRNGs.

PRNG	Matlab RAND	R250	Simscript	Whlcg
x_1	0.003859	0.003551	0.001936	0.000502
x_2	0.000175	0.002411	0.002515	0.001263
x_3	0.002813	0.002031	0.001172	0.002813
x_4	0	0	0	0
x_5	0	0	0	0
x_6	0	0	0	0
x_7	0	0	0	0
x_8	0	0	0	0
x_9	0	0	0	0
x_{10}	0	0	0	0
$f(x_i)$	0.000039	0.000037	0.000021	0.000027
Relative error	**0.003869**	**0.003661**	**0.002052**	**0.002717**
Ranking	4	3	1	2

where $f(x)$ is the objective function, $a(x) < b$ are constraints (and $<$ can be replaced with any of the inequality symbols) and c and d are the endpoints of the search interval of each variable.

Test function 8: *A two variable problem*

This test problem has a nonlinear objective and both linear and nonlinear constraints. The problem was found in Winston [18] and is defined as:

$$\max z = 30x_1 + 35x_2 - 2x_1^2 - 3x_2^2$$
$$\text{subject to } x_1^2 + 2x_2^2 \leq 250$$
$$x_1 + x_2 \leq 20$$
$$0 \leq x_1, \quad x_2 \leq 10.$$

To obtain the precision requirement of six places after the decimal point, each x_i require 24 bits ($2^{23} = 8388{,}608 < 10.24 \times 1000{,}000 = 10{,}240{,}000 < 16{,}777{,}216 = 2^{24}$). Therefore, the parent population will contain strings of $24 \times 2 = 48$ bits.

The known solution is at $x_1 = 7.5$, $x_2 = 5.83$, and $z = 214.58$. The initial results are shown in Table 4.2.1

From Table 4.2.1, it is clear that all generators except Faure and Sobol have an acceptable relative error less than 0.005%. The Faure and Sobol sequences produce relative error results that are higher than the requirement and are not recommended for use as the parent population. A ranking of the remaining generators gives: Whlcg, Niederreiter, R250, Halton, Matlab RAND, and Simscript.

Test function 9: *A two variable problem*

This test problem has a nonlinear objective and a linear constraint. The problem was found in Hillier and Lieberman [19] and is defined as:

$$\max z = 5x_1 - x_1^2 + 8x_2 - 2x_2^2$$
$$\text{subject to } 3x_1 + 2x_2 \leq 6$$
$$0 \leq x_1 \leq 2$$
$$0 \leq x_2 \leq 3.$$

Table 4.2.1
Initial results for test function 8.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	7.540970	5.829869	214.579940	**0.000028**	4
Faure (dim = 2)	7.480469	5.625000	214.452363	0.059482	8
Sobol	7.500005	5.625000	214.453125	0.059127	7
Niederreiter	7.497130	5.866397	214.580037	**0.000017**	2
Matlab RAND	7.512426	5.799871	214.579665	**0.000156**	5
R250	7.468755	5.812269	214.580050	**0.000023**	3
Simscript	7.463716	5.890903	214.570757	**0.004307**	6
Whlcg	7.516693	5.863900	214.579973	**0.000013**	1

Table 4.2.2
Initial results for test function 9.

RNG	x_1	x_2	$f(x_1, x_2)$	Relative error (%)	Ranking
Halton (dim = 2)	0.998287	1.502401	11.499648	**0.003057**	3
Faure (dim = 2)	0.999512	1.500000	11.498535	0.012737	7
Sobol	0.999512	1.500000	11.498535	0.012737	7
Niederreiter	0.996288	1.505464	11.499718	**0.002452**	1
Matlab RAND	0.997296	1.503887	11.499623	**0.003282**	4
R250	0.998580	1.501978	11.499687	**0.002721**	2
Simscript	0.999792	1.500115	11.499606	**0.003426**	5
Whlcg	0.994099	1.508696	11.499502	**0.004326**	6

Table 4.2.3
Initial results for QRNGs for test function 10.

QRNG	Halton	Faure	Sobol	Niederreiter
x_1	0.999999	1.000000	0.999999	0.999997
x_2	0.999999	0.999990	0.999991	0.999999
x_3	0.0000005	0	0	0
x_4	0.999990	0.999990	0.999990	0.999990
x_5	0	0	0	0
$f(x_i)$	−16.999302	−16.998884	−16.998914	−16.999210
Relative error	**0.004104**	0.006565	0.0063864	**0.004646**
Ranking	1	4	3	2

To obtain the precision requirement of six places after the decimal point, x_1 requires 21 bits ($2^{20} = 1048{,}576 < 2 \times 1000{,}000 = 2000{,}000 < 2097{,}152 = 2^{21}$) and x_2 requires 22 bits ($2^{21} = 2097{,}152 < 3 \times 1000{,}000 = 3000{,}000 < 4194{,}304 = 2^{22}$). Therefore, the parent population will contain strings of $21 + 22 = 43$ bits.

The known solution is at $x_1 = 1$, $x_2 = 1.5$, and $z = 11.5$. The initial results are shown in Table 4.2.2.

From Table 4.2.2, it is clear that all generators except Faure and Sobol have an acceptable relative error less than 0.005%. The Faure and Sobol sequences produce relative error results that are higher than the requirement and are not recommended for use as the parent population. A ranking of the remaining generators based on relative error gives: Niederreiter, R250, Halton, Matlab RAND, Simscript, and Whlcg.

Test Function 10: *A five variable problem*

This test problem has a nonlinear objective and a linear constraint. The problem is defined as:

$$\max z = 42x_1 + 44x_2 + 45x_3 + 47x_4 + 47.5x_5 - 50(x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2)$$
$$\text{subject to } 20x_1 + 12x_2 + 11x_3 + 7x_4 + 5x_5 \leq 40$$
$$0 \leq x_i \leq 1.$$

To obtain the precision requirement of six places after the decimal point, each x_i requires 20 bits ($2^{19} = 524{,}288 < 1 \times 1000{,}000 = 1000{,}000 < 1048{,}576 = 2^{20}$). Therefore, the parent population will contain strings of $20 \times 5 = 100$ bits.

The known solution is at $(1, 1, 0, 1, 0)$ with $z = -17$. The initial results for the QRNGs and PRNGs are shown in Tables 4.2.3 and 4.2.4 respectively.

From Table 4.2.3, all of the QRNGs except Faure and Sobol have an acceptable relative error less than 0.005%. The Faure and Sobol sequences produce relative error results that are higher than the requirement (although close) and are not recommended for use as the parent population. A ranking of the remaining generators based on relative error gives: Niederreiter and then Halton. From Table 4.2.4, all of the PRNGs have an acceptable relative error less than 0.005%. A ranking of these generators gives R250, Matlab RAND, Whlcg, and Simscript. An overall ranking of all RNGs gives: R250, Halton, Niederreiter, Matlab RAND, Whlcg, Simscript, Sobol, and Faure.

Table 4.2.4
Initial results for PRNGs for test function 10.

PRNG	Matlab RAND	R250	Simscript	Whlcg
x_1	0.999998	0.999999	0.999997	0.999999
x_2	0.999999	1.000000	0.999998	0.999999
x_3	0.000004	0.000001	0.0000004	0.000003
x_4	0.999991	0.999990	0.999990	0.999990
x_5	0	0	0	0
$f(x_i)$	−16.999173	−16.999344	−16.999146	−16.999170
Relative error	**0.004862**	**0.003858**	**0.005025**	**0.004880**
Ranking	2	1	4	3

Table 4.2.5
Initial results for QRNGs for test function 11.

QRNG	Halton	Faure	Sobol	Niederreiter
x_1	0.0000009	0.0000005	0.0000004	0.0000008
x_2	0.999998	0.999990	0.999990	1.000000
x_3	0.00000004	0	0	0
x_4	0.999990	0.999990	0.999990	0.999990
x_5	0.999994	0.999993	0.999998	0.999993
x_6	19.999990	19.999990	19.999990	19.999990
$f(x_i)$	−361.498074	−361.497045	−361.497550	−361.498083
Relative error	**0.000533**	**0.000818**	**0.000678**	**0.000530**
Ranking	2	4	3	1

Table 4.2.6
Initial results for PRNGs for test function 11.

PRNG	Matlab RAND	R250	Simscript	Whlcg
x_1	0.0000001	0.0000002	0.00000004	0.0000006
x_2	1.000000	0.999995	0.999993	0.999994
x_3	0.00000006	0.0000003	0.00000009	0.0000002
x_4	0.999990	0.999990	0.999995	0.999990
x_5	0.999995	0.999993	0.999990	0.999993
x_6	19.999990	19.999990	19.999990	19.999990
$f(x_i)$	−361.498353	−361.497607	−361.497635	−361.497545
Relative error	**0.000456**	**0.000662**	**0.000654**	**0.000679**
Ranking	1	3	2	4

Test function 11: *A six variable problem*

This test problem has a nonlinear objective and only linear constraints. The problem is defined as:

$$\max z = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10x_6 - 50(x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2)$$
$$\text{subject to } 6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \le 6.5$$
$$10x_1 + 10x_3 + x_6 \le 20$$
$$0 \le x_1, x_2, x_3, x_4, x_5 \le 1$$
$$10 \le x_6 \le 20.$$

To obtain the precision requirement of six places after the decimal point, x_1, x_2, x_3, x_4, and x_5 each require 20 bits ($2^{19} = 524{,}288 < 1 \times 1000{,}000 = 1000{,}000 < 1048{,}576 = 2^{20}$) and x_6 requires 24 bits ($2^{23} = 8388{,}608 < 10 \times 1000{,}000 = 10{,}000{,}000 < 16{,}777{,}216 = 2^{24}$). Therefore, the parent population will contain strings of $20 \times 5 + 24 = 124$ bits.

The known solution is at (0, 1, 0, 1, 1, 20) with $z = -361.5$. The initial results for the QRNGs and PRNGs are shown in Tables 4.2.5 and 4.2.6, respectively.

From Table 4.2.5, all of the QRNGs have an acceptable relative error less than 0.005%. A ranking of these generators gives: Niederreiter, Halton, Sobol, and Faure. From Table 4.2.6, all of the PRNGs also have an acceptable relative error less than 0.005%. A ranking of these generators gives: Matlab RAND, Simscript, R250, and Whlcg. An overall ranking of all RNGs gives: Matlab RAND, Niederreiter, Halton, Simscript, R250, Sobol, Whlcg, and Faure.

Test function 12: *A ten variable problem*

This test problem has a nonlinear objective and a linear constraint. The problem is defined as:

$$\max z = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7 + x_7x_8 + x_8x_9 + x_9x_{10} + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6 + x_5x_7 + x_6x_8 + x_7x_9 + x_8x_{10} + x_1x_9 + x_1x_{10} + x_2x_{10} + x_1x_5 + x_4x_7$$
$$\text{subject to } x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \le 1$$
$$0 \le x_i \le 0.50.$$

Table 4.2.7
Initial results for QRNGs for test function 12.

QRNG	Halton	Faure	Sobol	Niederreiter
x_1	0.000002	0.000001	0.000007	0.0000009
x_2	0.000007	0	0	0.0000018
x_3	0.000001	0	0	0.0000076
x_4	0.249996	0.249996	0.249997	0.2499971
x_5	0.250000	0.249996	0.249996	0.2499962
x_6	0.249996	0.249996	0.249996	0.2499962
x_7	0.249996	0.249997	0.249996	0.2499962
x_8	0	0	0	0
x_9	0	0	0	0
x_{10}	0	0	0	0
$f(x_i)$	0.374992	0.374989	0.374989	0.374989
Relative error	**0.000023**	**0.000030**	**0.000030**	**0.000029**
Ranking	1	3	3	2

Table 4.2.8
Initial results for PRNGs for test function 12.

PRNG	Matlab RAND	R250	Simscript	Whlcg
x_1	0.000005	0.000002	0.000001	0.000002
x_2	0.000005	0.000004	0.0000005	0.000007
x_3	0.000002	0.000005	0.000002	0.0000008
x_4	0.249996	0.250000	0.250000	0.250000
x_5	0.249996	0.249996	0.249996	0.249996
x_6	0.249997	0.249996	0.249996	0.249996
x_7	0.249996	0.249996	0.249996	0.249996
x_8	0	0	0	0
x_9	0	0	0	0
x_{10}	0	0	0	0
$f(x_i)$	0.374989	0.374992	0.374991	0.374991
Relative error	**0.000029**	**0.000023**	**0.000023**	**0.000029**
Ranking	4	1	2	2

To obtain the precision requirement of six places after the decimal point, each x_i requires 19 bits ($2^{18} = 262{,}144 < 0.5 \times 1000{,}000 = 500{,}000 < 524{,}288 = 2^{19}$). Therefore, the parent population will contain strings of $19 \times 10 = 190$ bits.

The known solution is at $(0, 0, 0, 0.25, 0.25, 0.25, 0.25, 0, 0, 0)$ with $z = 0.375$. The initial results for the QRNGs and PRNGs are shown in Tables 4.2.7 and 4.2.8, respectively.

From Table 4.2.7, all of the QRNGs have an acceptable relative error less than 0.005%. A ranking of these generators based on relative error gives: Halton, Niederreiter, and Faure and Sobol (tied). From Table 4.2.8, all of the PRNGs have an acceptable relative error less than 0.005%. A ranking of these generators gives R250, Simscript and Whlcg (tied), and Matlab RAND. An overall ranking of all RNGs gives: Halton and R250 (tied), Simscript and Whlcg (tied), Niederreiter and Matlab RAND (tied), and Faure and Sobol (tied).

Test function 13: *A ten variable problem*

This test problem has a nonlinear objective and linear constraints. The problem is defined as:

$$\begin{aligned}
\max z \;=\; & 48x_1 + 42x_2 + 48x_3 + 45x_4 + 44x_5 + 41x_6 + 47x_7 + 42x_8 + 45x_9 + 46x_{10} \\
& - 50(x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2) \\
\text{subject to } & -2x_1 - 6x_2 - x_3 - 3x_5 - 3x_6 - 2x_7 - 6x_8 - 2x_9 - 2x_{10} <= -4 \\
& 6x_1 - 5x_2 + 8x_3 - 3x_4 + x_6 + 3x_7 + 8x_8 + 9x_9 - 3x_{10} <= 22 \\
& -5x_1 + 6x_2 + 5x_3 + 3x_4 + 8x_5 - 8x_6 + 9x_7 + 2x_8 - 9x_{10} <= -6 \\
& 9x_1 + 5x_2 - 9x_4 + x_5 - 8x_6 + 3x_7 - 9x_8 - 9x_9 - 3x_{10} <= -23 \\
& -8x_1 + 7x_2 - 4x_3 - 5x_4 - 9x_5 + x_6 - 7x_7 - x_8 + 3x_9 - 2x_{10} <= -12 \\
& 0 \le x_i \le 1.
\end{aligned}$$

To obtain the precision requirement of six places after the decimal point, each x_i requires 20 bits ($2^{19} = 524{,}288 < 1 \times 1000{,}000 = 1000{,}000 < 1048{,}576 = 2^{20}$). Therefore, the parent population will contain strings of $20 \times 10 = 200$ bits.

The known solution is at $(1, 0, 0, 1, 1, 1, 0, 1, 1, 1)$ with $z = 39$. The initial results for the QRNGs and PRNGs are shown in Tables 4.2.9 and 4.2.10, respectively.

From Table 4.2.9, all of the QRNGs have an acceptable relative error less than 0.005%. A ranking of these generators gives: Faure, Sobol, Halton, and Niederreiter. From Table 4.2.10, all of the PRNGs have an acceptable relative error less than 0.005%. A ranking of these generators gives: Simscript, Whlcg, R250, and Matlab RAND. An overall ranking of all RNGs gives: Faure, Sobol, Simscript, Halton, Niederreiter, Whlcg, R250, and Matlab RAND.

Table 4.2.9
Initial results for QRNGs for test function 13.

QRNG	Halton	Faure	Sobol	Niederreiter
x_1	1.000000	1.000000	1.000000	1.000000
x_2	0.0000004	0	0	0.0000005
x_3	0.0000009	0	0	0.0000005
x_4	0.999996	0.999996	0.999996	0.999996
x_5	0.999997	0.999997	0.999996	0.999996
x_6	0.999996	0.999996	0.999996	0.999996
x_7	0.0000001	0.0000001	0.	0.0000001
x_8	0.999996	0.999996	0.999996	0.999996
x_9	0.999996	0.999996	0.999996	0.999996
x_{10}	0.999996	0.999996	0.999996	0.999996
$f(x_i)$	−38.998671	−38.998719	−38.998708	−38.998662
Relative error	**0.000034**	**0.000033**	**0.000033**	**0.000034**
Ranking	3	1	2	4

Table 4.2.10
Initial results for PRNGs for test function 13.

PRNG	Matlab RAND	R250	Simscript	Whlcg
x_1	**1.000000**	**1.000000**	**0.999999**	**1.000000**
x_2	**0.0000003**	**0.000001**	**0.000000001**	**0.0000002**
x_3	**0.000002**	**0.000001**	**0.0000003**	**0.000002**
x_4	0.999996	0.999996	0.999996	0.999996
x_5	0.999996	0.999996	0.999996	0.999997
x_6	0.999996	0.999996	0.999997	0.999996
x_7	0	0.00000003	0	0.0000001
x_8	0.999996	0.999996	0.999996	0.999996
x_9	0.999996	0.999996	0.999996	0.999996
x_{10}	0.999996	0.999996	0.999996	0.999996
$f(x_i)$	**−38.998609**	**−38.998631**	**−38.998681**	**−38.998640**
Relative error	**0.000036**	**0.000035**	**0.000034**	**0.000035**
Ranking	4	3	1	2

Table 5.1
Comparison of rankings for each generator based on test problems.

RNG	TF 1	TF 2	TF 3	TF 4	TF 5	TF 6	TF 7	Overall ranking
Halton (d = 2)	2	6	5	2	4	3	6	2
Faure (d = 2)	7	1	8	8	7	2	2	7
Sobol	7	1	7	7	8	6	1	8
Niederreiter	1	3	1	6	1	4	3	1
Matlab RAND	4	8	4	5	2	1	8	4
R250	3	3	2	3	5	5	7	2
Simscript	6	3	6	1	6	8	4	6
Whlcg	5	6	3	4	3	7	5	5

5. Rankings of RNGs

The unconstrained test problems have shown that the best RNG to use is dependent on the type of the problem. Table 5.1 gives a comparison of the rankings for the seven test examples. Based on the average rank for all of the test problems, the generators are ranked overall as Niederreiter, Halton and R250 (tied), Matlab RAND, Whlcg, Simscript, Faure, and Sobol.

The constrained test problems have shown that the best RNG to use is dependent on the type of the problem. Table 5.2 gives a comparison of the rankings for the six test examples. Based on the average rank for all of the test problems, the generators are ranked overall as: Halton, Niederreiter, R250, Matlab RAND and Simscript (tied), Whlcg, Sobol, and Faure.

6. Conclusions

Which existing random number generator is the best under all circumstances is still an open problem. The answer will depend on numerous parameters, such as the number of variables, the choice of function, the search space, and desired accuracy.

Traditional mathematical techniques for optimization, such as Newton–Raphson and conjugate gradient techniques (both of which are derivative based), encounter difficulties with estimation of a large number of parameters that interact in

Table 5.2
Comparison of rankings for each generator based on test problems.

RNG	TF 1	TF 2	TF 3	TF 4	TF 5	TF 6	Overall ranking
Halton (d = 2)	4	3	2	3	1	4	1
Faure (d = 2)	8	7	8	8	7	1	8
Sobol	7	7	7	6	7	2	7
Niederreiter	2	1	3	2	5	5	2
Matlab RAND	5	4	4	1	5	8	4
R250	3	2	1	5	1	7	3
Simscript	6	5	6	4	3	3	4
Whlcg	1	6	5	7	3	6	6

highly nonlinear ways, such as objective functions characterized with many local optima or when the objective function is discontinuous [5]. The heuristic[4] method of GAs offers a powerful alternative to these types of problems.

The convergence of GAs is usually slower than traditional techniques. This is because of the constant testing of suboptimal solutions. Further, the solution will only be an estimate versus an exact answer.

Genetic algorithms have been successfully implemented on parallel computers. In a parallel GA, the population is divided into subpopulations and the GA is performed on each subpopulation. The parallelization of the population into several subpopulations has been shown to speed up the convergence of the GA as the size of the problem increases [20].

References

[1] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, Massachusetts, 1997.
[2] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, Massachusetts, 1989.
[3] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed., Springer, Germany, 1996.
[4] W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francome, Genetic Programming — An Introduction: On the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann Publishers, San Francisco, California, 1998.
[5] R. Baker, Genetic algorithms in search and optimization, Financial Engineering News 5 (1998).
[6] A. Marczyk, Genetic algorithms and evolutionary computation, 2004, The Talk Origins Archive: http://www.talkorigins/faqs/genalg/genalg.html.
[7] M. Obitko, P. Slavík, Visualization of genetic algorithms in a learning environment, Spring Conference on Computer Graphics, 1999.
[8] M. Vose, The Simple Genetic Algorithm: Foundations and Theory, MIT Press, Cambridge, Massachusetts, 1999.
[9] S.M. Thede, An introduction to genetic algorithms, JCSC 20 (2004) 1.
[10] S.K. Sen, T. Samanta, A. Reese, Quasi- versus Pseudo-random generators: Discrepancy, complexity and integration-error based comparison, International Journal of Innovative Computing, Information and Control 2 (3) (2006) 621–651.
[11] V. Lakshmikantham, S.K. Sen, T. Samanta, Comparing random number generators using Monte Carlo integration, International Journal of Innovative Computing, Information and Control 1 (2) (2005) 143–165.
[12] E. Cantu-Paz, On random numbers and the performance of genetic algorithms, in: Proc. Genetic and Evolutionary Computation Conference, 2002.
[13] M.M. Meysenburg, J.A. Foster, The quality of pseudo-random number generators and simple genetic algorithm performance, in: Proc. Seventh International Conference on Genetic Algorithms, pp. 276–281, 1997.
[14] M.M. Meysenburg, D. Hoelting, D. McElvain, J.A. Foster, How random generator quality impacts genetic algorithm performance, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2002.
[15] M. Maaranen, K. Miettinen, M.M. Makela, Quasi-random initial population for genetic algorithms, Computers and Mathematics with Applications 47 (2004) 1885–1895.
[16] S. Kimura, K. Matsumura, Genetic algorithms using low-discrepancy sequences, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2005.
[17] G.R. Wood, The bisection method in higher dimensions, Mathematical Programming 55 (1992) 319–337.
[18] W.L. Winston, Operations Research: Applications and Algorithms, 4th ed., Brooks/Cole, 2004.
[19] F. Hillier, G. Lieberman, Introduction to Operations Research, McGraw-Hill Co., New York, 1995.
[20] T. Marvidou, P.M. Pardalos, L. Pitsoulis, M.G.C. Resende, Parallel search for combinatorial optimization: genetic algorithms, simulated annealing, tabu search, and GRASP, in: Proceedings of the Parallel Algorithms for Irregularly Structured Problems, 1995.

[4] Heuristic methods are based on or involve trial and error. They are not rigorous, but still convincing.