Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 47 (2004) 1885–1895
An International Journal
computers & mathematics with applications
www.elsevier.com/locate/camwa

Quasi-Random Initial Population for Genetic Algorithms

H. MAARANEN*
Department of Mathematical Information Technology, University of Jyväskylä
P.O. Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland
K. MIETTINEN
Helsinki School of Economics
P.O. Box 1210, FIN-00101, Finland
M. M. MÄKELÄ
Department of Mathematical Information Technology, University of Jyväskylä
P.O. Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland

(Received June 2003; accepted July 2003)

Abstract—The selection of the initial population in a population-based heuristic optimization method is important, since it affects the search for several iterations and often has an influence on the final solution. If no *a priori* information about the optima is available, the initial population is often selected randomly using pseudorandom numbers. Usually, however, it is more important that the points are as evenly distributed as possible than that they imitate random points. In this paper, we study the use of quasi-random sequences in the initial population of a genetic algorithm. Sample points in a quasi-random sequence are designed to have good distribution properties. Here a modified genetic algorithm using quasi-random sequences in the initial population is tested by solving a large number of continuous benchmark problems from the literature. The numerical results of two implementations of genetic algorithms using different quasi-random sequences are compared to those of a traditional implementation using pseudorandom numbers. The results obtained are promising.

Keywords—Random numbers, Quasi-random sequences, Global continuous optimization, Genetic algorithms.

1. INTRODUCTION

Many practical optimization problems are nonconvex and contain several local optima. Without loss of generality, we may restrict our study to minimization problems. We consider *global optimization problems*, where a continuous real-valued objective function f is minimized over

---

*Author to whom all correspondence should be addressed.

This research was supported by the Academy of Finland Grant #102020 and the National Technology Agency of Finland. We also wish to thank Professor M. Mascagni and Y. Li, Florida State University, for providing the Niederreiter quasi-random sequence generator. We are also indebted to Dr. J. Toivanen, University of Jyväskylä, Finland, for providing the code for the original genetic algorithm.

0898-1221/04/$ - see front matter Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX
doi:10.1016/j.camwa.2003.07.011

a feasible region $S \subset \mathbf{R}^n$. The *feasible region* S is defined using upper and lower bounds for each of the variables $\mathbf{x}_i$, $i = 1, \dots, n$. A solution $\tilde{\mathbf{x}}$ is called a *local minimum*, if there exists a neighborhood of $\tilde{\mathbf{x}}$ such that $f(\tilde{\mathbf{x}}) \leq f(\mathbf{x})$, for all $\mathbf{x}$ in that neighborhood. A *global minimum* $\mathbf{x}^*$ is the smallest of all the local minima in S. Traditional optimization methods explore the neighborhood of a current solution and only select new solutions that strictly decrease the objective function value. This usually causes them to stagnate in a local minimum, which may be far from a global minimum. Therefore, global optimization methods are needed if one desires to search for a global minimum. In recent years, methods that are called metaheuristics [1] have become popular for solving computationally difficult global optimization problems. In this paper, we consider a genetic algorithm, see, for example, [2], which is a widely used metaheuristic.

Genetic algorithms imitate natural evolution in a population. They consider several solutions at the same time. In the genetic algorithm context, the solutions are called *individuals*. We use the term individual as a synonym for the terms solution and (feasible) point. The set of solutions is called a *population* and the iteration of a genetic algorithm is called a *generation*. The population evolves from one generation to another as the individuals are crossbred and mutated.

The purpose of this paper is twofold. First, we want to study whether different initial populations of a genetic algorithm have an effect on the final objective function value and the total number of generations used. Our hypothesis is that the distribution of the initial population is meaningful. Second, we apply quasi-random sequences in the initial population to find out whether this would improve the final objective function value.

The distribution of the population at different stages of the search is essential for genetic algorithms, see [2]. Often, a diverse population is preferred in the beginning and a more condensed population at the end of the search. We only consider the distribution of the initial population, since this issue is often overlooked in the literature.

Sometimes there is *a priori* information about the location of the minima. This information may include, for example, some knowledge about the region of attraction of the global minimum. The *region of attraction* of a global minimum $\mathbf{x}^*$ is defined as the largest set of points, such that for any starting point from that set, the infinitely small step steepest decent algorithm will converge to the global minimum $\mathbf{x}^*$ [3]. The region of attraction for a local minimum is defined in a similar manner.

When *a priori* information about the minima is available, then the initial population can be selected so that the attractive areas of the feasible region are covered with the sample points more elaborately than other areas. This, however, is not a standard feature in genetic algorithms. We concentrate on a more traditional case when it is assumed that no *a priori* information is available about the number, value, or location of the local or global minima or the size of their regions of attraction.

Uniformly distributed points are diverse and, therefore, they are particularly useful in the initial population when there is no *a priori* information about the minima available. Different aspects of uniform distributions of sequences are discussed, for example, in [4]. When studying different uniform distributions, it becomes apparent that not all uniformly distributed sequences are as evenly distributed as the others. There exist measures to evaluate the goodness of a uniform sequence. For simplicity, from now on, we say that points have a "good" uniform distribution if they are very evenly distributed over the feasible region. Correspondingly, we say that points have a "bad" uniform distribution, if they are just barely uniformly distributed.

Traditionally, the initial population of a genetic algorithm is said to be random. Yet, it is a well-known fact that random numbers cannot be generated algorithmically. The algorithmically generated numbers, which are commonly used, only try to imitate random numbers. They are more accurately called *pseudorandom numbers*. To make the distinction to pseudorandom numbers more clear, we use the term *genuine random numbers*, when we mean numbers that are truly independent.

There are also so-called quasi-random sequences. The points in a quasi-random sequence are designed to maximally avoid each other. In other words, while the points generated using pseudorandom numbers try to imitate genuine random points, the points generated using quasi-random sequences try to imitate points with a "perfect uniform distribution". The former is impossible and the latter is, if not impossible, at least extremely difficult.

Quasi-random sequences have been successfully applied in numerical integration [5–9] and in random search optimization methods [6,10,11]. The idea of a good initial population has also been used in genetic programming [12]. These results motivated us to apply quasi-random sequences to the generation of the initial population of a genetic algorithm for problems involving continuous variables. For short, we call this modification a *quasi-genetic algorithm* and the genetic algorithm with pseudorandom numbers the *original genetic algorithm*.

The goal of selecting the initial population for genetic algorithms is to gather as much information about the objective function as possible. This corresponds to small-scale random search with no *a priori* knowledge about minima. Since the use of quasi-random sequences has been advantageous in random search methods [6,10], we may assume that quasi-random sequences have a positive effect on the performance of genetic algorithms when applied to the initial population.

We evaluate the performance of the quasi-genetic algorithm by solving a large set of computationally difficult test problems and comparing the results with those of the original genetic algorithm. The main criterion in the evaluation is the accuracy of the final objective function value, although the number of generations is also considered.

The rest of the paper is organized as follows. In Section 2, we contrast the ideology of pseudorandom numbers and quasi-random sequences. We also define three common measures for distinguishing good and bad uniform distributions. In Section 3, we briefly describe the genetic algorithm used in terms of genetic operations and introduce the parameter values used. We also discuss how and where the quasi-random sequences are used in our implementation. In Section 4, we introduce the test problems and some numerical and graphical results. In Section 5, we discuss some general issues related to quasi-random sequence generation. Finally, we draw conclusions from the results and point out some future directions of our research.

2. PSEUDORANDOM NUMBERS AND QUASI-RANDOM SEQUENCES

In quasi-genetic algorithms, we partly substitute the pseudorandom numbers with quasi-random sequences. In this chapter, we discuss pseudorandom numbers and quasi-random sequences in general and point out some of their differences.

Pseudorandom numbers are deterministic, but they try to imitate an independent sequence of genuine random numbers. Common pseudorandom number generators include, among others, linear congruential, quadratic congruential, inversive congruential, parallel linear congruential, additive congruential, lagged Fibonacci, and feedback shift register generators (see, for example, [6,13,14]). In addition, there exist numerous modifications and combinations of the basic generators [13].

In contrast to pseudorandom numbers, the points in a quasi-random sequence do not imitate genuine random points but, instead, try to cover the feasible region in an optimal way. Quasi-random generators do not generate numbers, but sequences of points in the desired dimension. The points in a quasi-random sequence are interrelated in the way that they inherently assume to have some "knowledge" of the location of the previous points in the sequence. Therefore, quasi-random sequences should usually include all the points from the beginning of the sequence. Common quasi-random sequence generators include Hammersley, Faure, Halton, Sobol', Niederreiter, and SQRT generators. Good general references for quasi-random sequence generation are, for example, [6,13]. Here, we use Niederreiter [6,15] and Sobol' [5,16] generators since they have proven most preferable in our preliminary tests including Faure, Halton, Niederreiter, and two

Sobol' generators [17]. A theoretical comparison of the distribution properties of a Sobol' and some Niederreiter generators are given in [18]. For practical differences in generating Niederreiter and Sobol' sequences, we refer to [15] and [5], respectively.

We restrict our considerations to uniform distributions, because we assume to have no *a priori* knowledge about the minima. As mentioned earlier, within uniformly distributed sequences, there are, however, sequences with good distribution properties and sequences that are just barely uniform [4]. Next, we define discrepancy, star discrepancy, and dispersion [4,6] that can be used for distinguishing the good uniform sequences from the bad ones. For different modifications of the traditional discrepancy and dispersion, see [6,19] and [20], respectively. As a rule of thumb, it can be said that the lower the value for discrepancy, star discrepancy, or dispersion, the better the distribution.

Let $I^n \subset \mathbf{R}^n$ be an n-dimensional unit hypercube, let P be a point set consisting of $\mathbf{x}^1, \dots, \mathbf{x}^N \in I^n$, and let $\mathcal{B}$ be a nonempty family of Lebesgue-measurable subintervals of I^n and $B \in \mathcal{B}$. Furthermore, let $A(B; P)$ be a counting function defined as the number of terms $\mathbf{x}^k$, $1 \le k \le N$, for which $\mathbf{x}^k \in B$. Then, *discrepancy* D_N [6] with respect to P in I^n is defined as

$$D_N(\mathcal{B}; P) = \sup_{B \in \mathcal{B}} \left| \frac{A(B;P)}{N} - \lambda(B) \right|,$$

where λ is a Lebesgue-measure. Often, a simpler measure, which is called star discrepancy $D_N^*(P)$, is used. *Star discrepancy* [6] is defined as

$$D_N^*(\mathcal{B}, P) = D_N(\mathcal{B}^*, P),$$

where $\mathcal{B}^*$ is the family of intervals of I^n of the form $\prod_{i=1}^n [0, u_i)$, and $\mathbf{u} = (u_1, \dots, u_n) \in I^n$. A simple relationship between discrepancy and star discrepancy is given in [19]

$$D_N^*(\mathcal{B}, P) \le D_N(\mathcal{B}, P) \le 2^n D_N^*(\mathcal{B}, P).$$

Discrepancy and star discrepancy are commonly used in numerical integration. In numerical integration, the error to be minimized is the difference between the estimated and the actual value of the integral. It has been proven that a small integration error can be guaranteed if point sets with small discrepancies or star discrepancies are used [6].

In optimization, the error to be minimized is the difference between the estimated and the actual optimal value $f(\mathbf{x}^*)$. Low discrepancy sequences have good distribution properties, but often in optimization, another measure, which is called dispersion, is used. Let d be a metric, (X, d) be a bounded metric space, and $P = \{\mathbf{x}^1, \dots, \mathbf{x}^N\}$. Then the *dispersion* d_N [6] is defined as

$$d_N(P; X) = \sup_{\mathbf{x} \in X} \min_{1 \le k \le N} d\left(\mathbf{x}, \mathbf{x}^k\right).$$

The relationship between discrepancy and dispersion is given in [6], and it shows that every low-discrepancy sequence is also a low-dispersion sequence, but not vice versa. To obtain a tool for optimization error analysis using dispersion, let us define the *modulus of continuity* of f

$$\omega(f; t) = \sup_{\substack{\mathbf{x}, \mathbf{y} \in X \\ d(\mathbf{x}, \mathbf{y}) \le t}} |f(\mathbf{x}) - f(\mathbf{y})|, \qquad \text{for } t \ge 0.$$

It has been proven (see, for example, [6]) that

$$\min_{1 \le k \le N} f\left(\mathbf{x}^k\right) - f\left(\mathbf{x}^*\right) \le \omega\left(f; d_N(P; X)\right),$$

indicating that low-dispersion sequences are suitable for random search techniques [6].

Now that we have defined some relations between discrepancy and star discrepancy as well as discrepancy and dispersion, it is, for our purpose, sufficient to consider only the discrepancy. The discrepancy bounds for quasi-random sequences for a large sample size N are roughly of order of magnitude $C(\log N)^n N^{-1}$, where C is a generator specific coefficient depending only on the dimension n. This is also a minimum possible discrepancy size for large N, see [19]. Different values of C are compared for Niederreiter and Sobol' sequences in [18]. As far as the pseudorandom numbers are concerned, the discrepancy bound is often estimated with the respective value of the genuine random numbers, which is of order of magnitude $C(\log\log N)^{1/2} N^{-1/2}$, see [19].

By substituting some values of n and a relatively small N to the discrepancy bounds, we notice that the discrepancy bound for genuine random numbers gives smaller values than the discrepancy bound for quasi-random sequences. This is misleading, because these error bounds are meaningful for large sample sizes only. In fact, they give no information if N is small [19]. For large dimensions n, the sample size should be of order $N = O(e^n)$ before the discrepancy bounds start to be informative [19]. In [19], the authors conclude that the performance of sequences should be tested in practice. This is especially true for optimization, where the sample sizes are significantly smaller than in numerical integration.

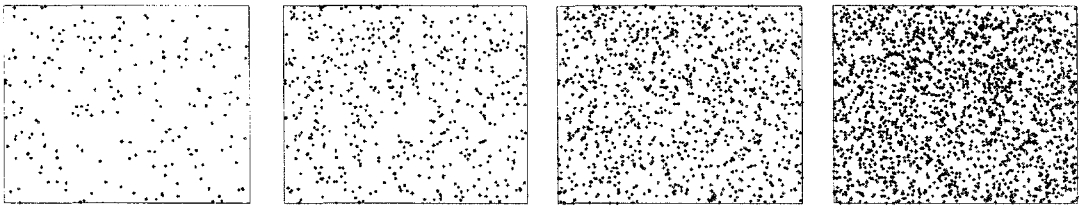

Figure 1. Two-dimensional sequence of 200, 500, 1000, and 2000 points generated using pseudorandom numbers.

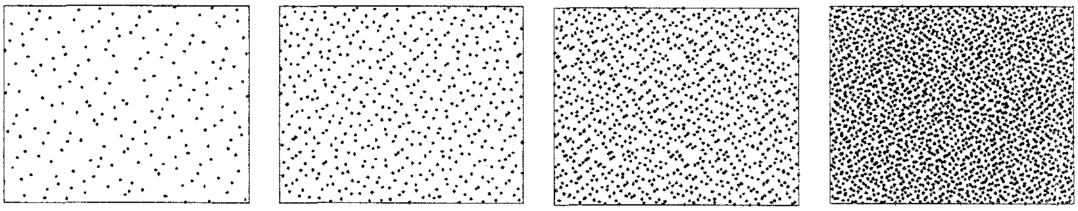

Figure 2. Two-dimensional quasi-random sequence of 200, 500, 1000, and 2000 points.

Figure 1 illustrates samples of 200, 500, 1000, and 2000 points generated using pseudorandom numbers. The figure illustrates one sequence, where new points are iteratively added among the existing ones. The pseudorandom numbers are generated using a combination of a lagged Fibonacci generator [13] and a shift register random integer generator [21].

Figure 2 illustrates samples of 200, 500, 1000, and 2000 points of a quasi-random sequence. The quasi-random sequence is generated using Sobol' generator, see, for example, [5,13]. If the values $n = 2$ and $N = 200$, 500, 1000, 2000 are substituted to the discrepancy bounds given above, we note that the discrepancy estimate is better for the pseudorandom numbers. However, we observe from Figure 2 that the points in the quasi-random sequence have the kind of distribution properties that we are interested in. This is a simple example showing that the actual discrepancy for small N may be smaller for the quasi-random sequences even though the discrepancy estimates derived using the given discrepancy bounds indicate otherwise.

Figures 1 and 2 are good examples of how the different generators are designed to work. The different goals become obvious. In Figure 1, there is no clear pattern to be detected in the distribution of the points generated using pseudorandom numbers. Thus, the goal to imitate genuine random points is well attained. On the other hand, the points generated using pseudorandom numbers tend to clump and form clusters leaving other areas relatively unexplored. This is characteristic of an independent sample [19]. As far as the quasi-random sequences are concerned, a certain pattern can be detected, especially when the sample size is large, see Figure 2. The points generated using quasi-random sequences are more evenly distributed and they do not clump. Hence, the replacing of pseudorandom numbers with a quasi-random sequence is well grounded in cases where the points do not need to be random, but a good uniform distribution is required.

3. QUASI-GENETIC ALGORITHM

In quasi-genetic algorithms, we use both pseudorandom numbers and quasi-random sequences where appropriate. Pseudorandom numbers, which imitate genuine random numbers, are used in all the genetic operations, while quasi-random sequences, which imitate points with perfect uniform distribution, are used in the initialization of the population. This division is natural, since the genetic operations (selection, crossover, and mutation) assume randomness, but the points in the initial population are desired to have a good uniform distribution.

The selection of the initial population of a genetic algorithm can be considered as one (initial) generation out of hundreds or thousands of generations. Moreover, selecting the initial population usually takes less time than a regular generation in a genetic algorithm. In any case, we assume that the initial population has a special role since all the populations in the iterative search process depend, to some extent, on the preceding population and, eventually, on the initial population.

The original genetic algorithm used in this paper is described in [22]. We apply tournament selection, heuristic crossover [2], and Michalewicz's nonuniform mutation [23]. The parameters used in these operations are given in Table 1. We use two sets of parameter values for problem sets P1 and P2, respectively. The problem sets P1 and P2 are described in more detail in the next section.

The parameter *population size* is the number of individuals in a population, *tournament size* defines the number of randomly selected individuals out of which the one with the best objective function value is chosen as a parent, and *elitism size* is the number of the best individuals directly copied to the next generation. The parameter *crossover rate* regulates the probability on which the selected parents are crossbred and *mutation rate* the probability on which an individual is mutated.

The parameters *max generations*, *steps*, and *tolerance* are attached to the two-fold stopping criterion: the algorithm is stopped after the maximum number of generations (*max generations*) is reached or if there has been no change larger than *tolerance* in the best objective function value for a predefined number (*steps*) of generations. We set the maximum number of generations to be quite large, and hence, the latter part of the two-fold stopping criterion becomes often decisive in the tests to be reported.

Several different parameter values were tested and the values given in Table 1 were chosen since they provided relatively good results for a large number of test problems. Three different versions of genetic algorithms were implemented. The first version, serving as a benchmark, is the original genetic algorithm using pseudorandom numbers in the initial population. The two other

Table 1. Optimization parameters for the genetic algorithm.

Description	P1	P2
Population size	201	501
Elitism size	21	31
Tournament size	3	3
Crossover rate	0.8	0.8
Mutation rate	0.1	0.1
Max generations	2000	10000
Steps	100	500
Tolerance	10^{-7}	10^{-7}

versions use Niederreiter [6,15] and Sobol' [5,16] generators when selecting the initial population. Respectively, another implementation of Sobol' generator was also tested, but it was omitted since it did not perform better than the first Sobol' implementation. The difference between the two Sobol' generators is that the first implementation uses gray coding and the omitted one does not. In addition, the two generators have differences in numerical computations. For further information about implementing Sobol' generators, see, for example, [5].

4. NUMERICAL RESULTS

A test suite of 52 global optimization problems was selected for testing the quasi-genetic algorithms. The test problems were taken from the literature with a view to having a broad selection of difficult problems with respect to the number of variables and the number of minima. Since some problems turned out to be computationally more difficult than the others, the problems were classified into problem sets P1 and P2 and different parameter values were used (given in Table 1). The classification was made by first solving all the problems with the parameter values used for the problem set P1 and then moving to set P2, those problems for which the objective function value was not "close to optimal". Finally, parameter values for the set P2 were readjusted.

All the problems were solved a hundred times with each implementation and the average values are reported. The names of the test problem families (if existing) along with the problem dimensions, box constraints, and the sources of reference are given in Table 2. The problems in P2 are marked with an asterisk (*) in the problem dimension column of Table 2.

For some function families, there exist additional parameters that change the nature of the problem, for example, number of local minima. Therefore, it is possible that there are several instances with the same dimension for one problem in the problem dimension column of Table 2.[1]

The box constraints are given as a cross product of intervals defined by the lower and the upper bound of each variable. In cases where the lower and the upper bounds are the same for all the

Table 2. Test problems.

#	Name	Dimensions	Box Constraints	Ref.
1–3	Michalewicz	2, 5, 10	$[0, \pi]$	[26]
4–7	Rastrigin	6, 10*, 20*, 30*	$[-600, 400]$	[26]
8–11	Schwefel	6, 10, 20, 50*	$[-500, 500]$	[26]
12	Branin rcos	2	$[-5, 10] \times [0, 15]$	[26]
13–17	Griewangk	2, 6, 10, 20, 50*	$[-700, 500]$	[26]
18–22	Ackley's Path	2, 6* , 10*, 20*, 30*	$[-30.768, 38.768]$	[26]
23	Easom	2	$[-100, 100]$	[26]
24	Levy	4	$[-10, 10]$	[25]
25–27	Levy	5, 6, 7	$[-5, 5]$	[25]
28	P8	3	$[-10, 10]$	[27]
29	P16	5	$[-5, 5]$	[27]
30	Hansen	2	$[-10, 10]$	[25]
31–34	Corona	4, 6, 10, 20*	$[-900, 1100]$	[28]
35–38	Katsuura	4, 6, 10, 20*	$[-1, 1]$	[28]
39–42	Langerman	5*, 5*, 10*, 10*	$[0, 10]$	[24,26]
43–46	Function 10	3, 4, 10, 20	$[-20, 20]$	[29]
47–49	Shekel	4*, 4*, 4*	$[0, 10]$	[25]
50–52	Epistatic Michalewicz	2, 5*, 10*	$[0, \pi]$	[24]

---

[1] For the Langerman family of functions, this parameter is $m = 15, 30, 15, 30$, see [24], with the dimension $n = 5, 5, 10, 10$, respectively, and for the Shekel family of functions $m = 5, 7, 10$, see [25] with dimension $n = 4$, for each.

variables, only one interval is given. Note, that even though the test functions have standard definitions in the literature, the size of their feasible regions may vary notably depending on the source of reference.

The Niederreiter sequence starts with a zero vector, which in the implementation of the genetic algorithm projects to the center of the box defined by the lower and the upper bounds of each variable. Since, in many of the test problems, the optimal point was located in the center of the box, it caused the implementation using the Niederreiter sequence to find the optimal point immediately. To obtain more comparable and informative results, we shifted the respective boxes so that the optimal point was no longer in the center of the box. This was done for the following function families: Rastrigin, Griewangk, Ackley's Path, and Corona. The boxes given in Table 2 are, therefore, not always exactly the same as in the respective sources of reference.

As already mentioned, we solved the test problems a hundred times with all three variants of the genetic algorithms and used the results obtained with the original genetic algorithm as benchmark results. For about half of the test problems, there was an observable difference in the results of the three implementations. Figure 3 illustrates the problems in P1 where the differences in the objective function values, when compared to the original genetic algorithm, were noteworthy. The difference is reported as a decrease in the objective function value in relative scaling. The relative decrease was considered insignificant if it was less than 0.2%. Figure 4 illustrates the respective relative decreases in the problem set P2. In Figures 3 and 4, *Nieder* and *Sobol'* denote the Niederreiter and the Sobol' generators, respectively.

We can see that, on the average, the use of quasi-random sequences improved the objective function value over the original genetic algorithm. The average improvement is shown as the bar denoted as *Avg* in the histograms in Figures 3 and 4.

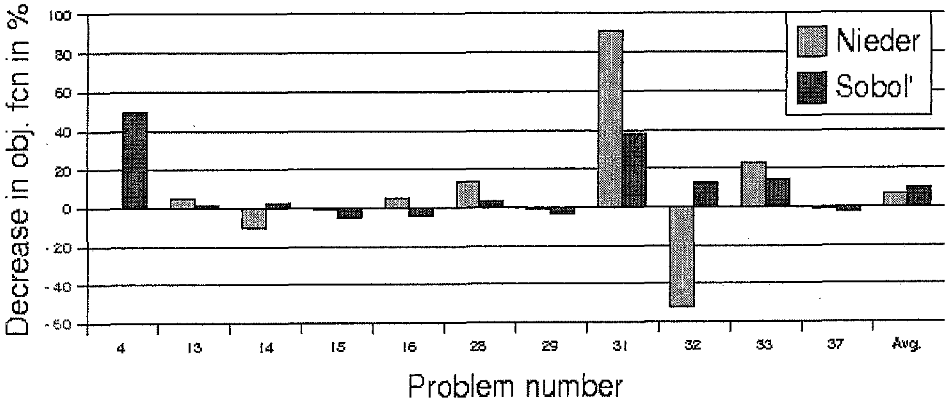

Figure 3. Decrease in objective function values for problem set P1.

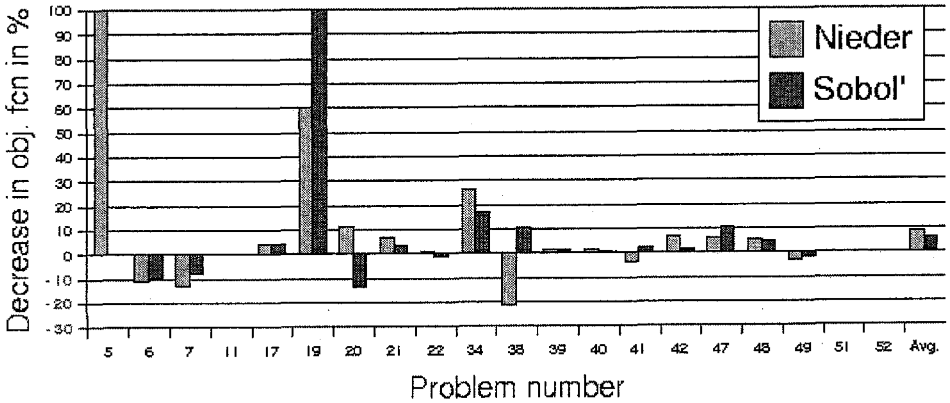

Figure 4. Decrease in objective function values for problem set P2.

Table 3 summarizes the number of problems where the objective function value decreased when compared to the respective benchmark value. The number of problems whose objective function value increased is given in the parenthesis. For example, the Niederreiter implementation for problem set P2 improved the objective function values for 12 problems, while for five problems, it received worse objective function values than the original genetic algorithm. Table 3 shows that the improvement ratio for both Niederreiter and Sobol' implementations was better for problems in the set P2. Overall, the objective function values improved approximately twice as often as they deteriorated.

Table 3. Number of the improved (and the deteriorated) objective function values.

	Nieder	Sobol'
P1	5 (4)	7 (4)
P2	12 (5)	11 (5)
Total	17 (9)	18 (9)

We also studied the effects that different initial populations had on the number of generations used. The overall results are reported in Table 4. We expected to see a small increase in the number of generations as a trade-off for the improved objective function values. The results of Sobol' implementation corresponded to our expectations, but, surprisingly, for the problem set P2, the Niederreiter implementation used fewer generations than the original genetic algorithm.

Table 4. Increase in the number of generations used.

	Nieder	Sobol'
P1	0.48%	0.13%
P2	−1.33%	0.12%
Total	−0.93%	0.12%

The results indicate that different initial populations really have an effect on the results of the genetic algorithm. For our set of test problems, the use of quasi-random sequences improved the average objective function value. Moreover, the expected trade-off between improved objective function value and the number of generations used was smaller than expected. For the Niederreiter implementation, the number of generations used even decreased on the average. Often, when using genetic algorithms, the final objective function value is emphasized while the number of generations used is considered a secondary criterion of method evaluation (see, for example, [30]). Due to this, we consider both quasi-genetic algorithms preferable when compared to the original genetic algorithm.

5. DISCUSSIONS

The generation of quasi-random sequences is numerically more difficult than the generation of pseudorandom numbers, and quasi-random sequence generators are not included in standard mathematical software libraries. These facts make them less convenient to use when compared to pseudorandom number generators. The extra effort required for testing or even implementing a new random number generator is often too much when the random numbers are not the principal subject of the study. Hence, there is a need for standardized and easily accessible quasi-random sequence generators.

A commonly mentioned disadvantage of quasi-random sequence generators is that their good distribution properties degrade when the dimension of the problem increases [19]. According to [15], the quasi-random sequences cease to have a theoretical advantage over the pseudorandom numbers when the problem dimension is more than 12. In numerical integration, this dimensional upper bound is theoretically correct for integrands with all the variables equally important [8].

The practical upper bound for the maximal number of variables is sometimes approximated to be 40, see [31]. These results for numerical integration are, to some extend, valid for optimization as well, because all the low discrepancy sequences are also low dispersion sequences [6].

The distribution properties of quasi-random sequences discussed in [15,19] are relevant in optimization. Nevertheless, in our numerical tests, the deterioration of the performance for larger optimization problems was not observable. As a matter of fact, the quasi-genetic algorithms performed even better for problem set P2 that included large-dimensional problems. These results indicate that the use of quasi-random sequences for large-dimensional problems deserves further research in optimization. One possible reason causing the difference between the theoretical and the practical results may be that the error bounds for quasi-random sequences presented in Section 2 are loose [15]. Another possible reason is the fact that the error bounds for pseudorandom numbers are derived under the unrealistic assumption related to the independence of points.

Although discrepancy and dispersion values can be derived for quasi-random sequences, there exist no standard practical tests to measure their quality. In [19], Sobol', Halton, and Faure quasi-random sequence generators are considered, and the authors state that "all of the sequences have potential problems as dimension increases". Hence, the generators should be further developed to guarantee good distributions.

6. CONCLUSIONS

We studied whether different initial populations of a genetic algorithm have an effect on the final objective function value and the total number of generations used. Our hypothesis was that the distribution of the initial population is meaningful. The numerical tests indicate that our assumption was correct. Additionally, we wanted to find out whether there exists a way to generate the initial population so that the final objective function value would improve in average. In quasi-genetic algorithms, we applied quasi-random sequences when initializing the population. The points in a quasi-random sequence are designed to maximally avoid each other and, hence, have a good uniform distribution. The quasi-genetic algorithms improved the final objective function value, as expected.

Quasi-random sequences have been successfully applied in numerical integration and random search optimization methods. The idea of good initial population has also been used in genetic programming. However, the use of quasi-random sequences has not been earlier extended to genetic algorithms in global optimization.

In this paper, we have discussed the benefits of the sequences with good uniform distributions, and have applied Niederreiter and Sobol' quasi-random sequences when generating points for the initial population of a genetic algorithm. We have studied the effects by comparing the results of the quasi-genetic algorithms to those of the original genetic algorithm using pseudorandom numbers.

A test suite of 52 computationally difficult problems were solved a hundred times with each of the three implementations. The results indicate that the distribution of the initial population has an effect on both the final objective function value and the number of generations. In about one-half of the problems, the difference in the final objective function value was noteworthy. In two-thirds of these cases, quasi-genetic algorithms improved the final objective function values.

The trade-off between the final objective function value and the number of generations used was more favorable for quasi-genetic algorithms than expected. In fact, the Niederreiter implementation used on the average fewer generations than the original genetic algorithm. Also, for the Sobol' implementation, the increase was only moderate. Since the major emphasis is often on objective function values, we consider the results of both the quasi-genetic algorithms favorable when compared to the results of the original genetic algorithm.

Our tests show that quasi-random sequences can be successfully applied to problems with relatively large dimensions. Nevertheless, there are some results in the literature that give a reason to question the quality of the distribution also for quasi-random sequences. Therefore, the topic of future research is to validate the results obtained using statistical methods in generating sequences for which a good uniform distribution is guaranteed.

REFERENCES

1. S. Voß, S. Martello, I.H. Osman and C. Roucairol, Editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic, Boston, MA, (1999).
2. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, (1994).
3. A. Törn and A. Žilinskas, *Global Optimization*, Springer-Verlag, (1989).
4. L. Kuipers and H. Niederreiter, *Uniform Distribution of Sequences*, John Wiley & Sons, New York, (1974).
5. P. Bratley and B.L. Fox, Algorithm 659: Implementing Sobol's quasi-random sequence generator, *ACM Transactions on Mathematical Software* **14** (1), 88–100, (1988).
6. H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM, Philadelphia, PA, (1992).
7. W.H. Press and S.A. Teukolsky, Quasi- (that is, sub-) random numbers, *Computers in Physics* **3** (6), 76–79, (1989).
8. I.M. Sobol', On quasi-Monte Carlo integrations, *Mathematics and Computers in Simulation* **47** (2–5), 103–112, (1998).
9. B. Tuffin, On the use of low discrepancy sequences in Monte Carlo methods, Technical Report 1060, I.R.I.S.A., Rennes, France (1996).
10. I.M. Sobol' and S.G. Bakin, On the crude multidimensional search, *Journal of Computational and Applied Mathematics* **56** (3), 283–293, (1994).
11. G. Lei, Adaptive random search in quasi-Monte Carlo methods for global optimization, *Computers Math. Applic.* **43** (6/7), 747–754, (2002).
12. W. Böhm and A. Geyer-Schults, Exact uniform initialization for genetic algorithms, In *Foundations of Genetic Algorithms 4*, (Edited by R.K. Belew and M.D. Vose), pp. 379–408, Morgan Kaufmann, (1997).
13. J.E. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer-Verlag, New York, (1998).
14. R.S. Wikramaratna, ACORN—A new method for generating sequences of uniformly distributed pseudo-random numbers, *Journal of Computational Physics* **83** (1), 16–31, (1989).
15. P. Bratley, B.L. Fox and H. Niederreiter, Implementation and tests of low-discrepancy sequences, *ACM Transactions on Modeling and Computer Simulation* **2** (3), 195–213, (1992).
16. Sobol' generator implementation, `http://www.taygeta.com/random.html`, June 2004.
17. H. Maaranen, On global optimization with aspects to method comparison and hybridization, Licentiate Thesis, Department of Mathematical Information Technology, University of Jyväskylä (2002).
18. H. Niederreiter and C. Xing, The algebraic-geometry approach to low-discrepancy sequences, In *Monte Carlo and Quasi-Monte Carlo Methods, 1996, Lecture Notes in Statistics, Number 127*, (Edited by H. Niederreiter, P. Hellekalek, G. Larcher and P. Zinterhof), pp. 139–160, Springer-Verlag, New York, (1998).
19. W.J. Morokoff and R.E. Caflisch, Quasi-random sequences and their discrepancies, *SIAM Journal on Scientific Computing* **15** (6), 1251–1279, (1994).
20. X. Wang, A new measure of irregularity of distribution and quasi-Monte Carlo methods for global optimization, *Computers Math. Applic.* **43** (6/7), 657–669, (2002).
21. Mathworks `http://www.mathworks.com`, June 2004.
22. K. Miettinen, M.M. Mäkelä and J. Toivanen, Numerical comparison of some penalty-based constraint handling techniques in genetic algorithms, *Journal of Global Optimization* **27** (4), 427–446, (2003).
23. K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, (2001).
24. Test problems in R^2, `http://iridia.ulb.ac.be/~aroli/ICEO/Functions/Functions.html`, June 2004.
25. Test problems for global optimization, `http://www.imm.dtu.dk/~km/GlobOpt/testex/`, June 2004.
26. Genetic and evolutionary algorithm toolbox for use with MatLab, `http://www.geatbx.com/`, June 2004.
27. A. Decker and E. Aarts, Global optimization and simulated annealing, *Mathematical Programming* **50**, 367–393, (1991).
28. S. Dykes and B. Rosen, Parallel very fast simulated reannealing by temperature block partitioning, In *Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics, Volume 2*, pp. 1914–1919, (1994).
29. B. Trafalis and S. Kasap, A novel metaheuristics approach for continuous global optimization, *Journal of Global Optimization* **23** (2), 171–190, (2002).
30. J. Andre, P. Siarry and T. Dognon, An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization, *Advances in Engineering Software* **32** (1), 49–60, (2001).
31. M. Mascagni and A. Karaivanova, What are quasirandom numbers and are they good for anything besides integration?, In *Proceedings of Advances in Reactor Physics and Mathematics and Computation into the Next Millenium (PHYSOR2000)*, (2000).