J Glob Optim (2007) 37:405–436
DOI 10.1007/s10898-006-9056-6

ORIGINAL PAPER

On initial populations of a genetic algorithm for continuous optimization problems

Heikki Maaranen · Kaisa Miettinen · Antti Penttinen

Received: 7 June 2006 / Accepted: 10 June 2006 /
Published online: 20 July 2006

Abstract Genetic algorithms are commonly used metaheuristics for global optimization, but there has been very little research done on the generation of their initial population. In this paper, we look for an answer to the question whether the initial population plays a role in the performance of genetic algorithms and if so, how it should be generated. We show with a simple example that initial populations may have an effect on the best objective function value found for several generations. Traditionally, initial populations are generated using pseudo random numbers, but there are many alternative ways. We study the properties of different point generators using four main criteria: the uniform coverage and the genetic diversity of the points as well as the speed and the usability of the generator. We use the point generators to generate initial populations for a genetic algorithm and study what effects the uniform coverage and the genetic diversity have on the convergence and on the final objective function values. For our tests, we have selected one pseudo and one quasi random sequence generator and two spatial point processes: simple sequential inhibition process and nonaligned systematic sampling. In numerical experiments, we solve a set of 52 continuous test functions from 16 different function families, and analyze and discuss the results.

Keywords Global optimization · Continuous variables · Evolutionary algorithms · Initial population · Random number generation

H. Maaranen
Patria Aviation Oy, Lentokonetehtaantie 3, FI-35600 Halli, Finland

K. Miettinen (✉)
Helsinki School of Economics, P.O. Box 1210, FI-00101 Helsinki, Finland
e-mail: kaisa.miettinen@hse.fi

A. Penttinen
Department of Mathematics and Statistics,
P.O. Box 35 (MaD), FI-40014 University of Jyväskylä, Finland

1 Introduction

When solving real optimization problems numerically, the solution process typically involves the phases of modeling, simulation and optimization. A simulated model of a real life problem is often complex, and the objective function to be minimized may be nonconvex and have several local minima. Then, global optimization methods are needed to prevent the stagnation to a local minimum. Therefore, in the recent years, there has been a great deal of interest in developing methods for solving global optimization problems (see, e.g., [12, 18, 32] and references therein). Here, we consider global continuous optimization problems.

Genetic algorithms [15, 16, 25] are metaheuristics used for solving problems with both discrete and continuous variables. The population is the main element of genetic algorithms, and the genetic operations like crossover and mutation are just instruments for manipulating the population so that it evolves towards the final population including a "close to optimal" solution. The requirements set on the population also change during the execution of the algorithm.

In the recent years, genetic operators have been developed intensively (see, for example, [25, 26] and references therein). In addition, most of the theoretical studies involve the tuning or controlling of the parameters of genetic operators and their role is often considered significant performance-wise (see, for example, [11]). However, the role of the initial population, which is the topic of this paper, is widely ignored. Often, the whole area of research is set aside by a statement "generate an initial population," without implying how it should be done. We show with a simple example that initial populations may have effects, on the best objective function value found, and these effects may last for several generations. Then, we continue to study whether the traditional way of generating initial populations is recommendable or whether there are other point sets that give faster convergence. Our motivation is to encourage discussion on whether one should pay more attention to the generation of initial populations.

We concentrate on the case where there is no a priori information about the location of the global minima. Then, initial populations of genetic algorithms are traditionally generated "randomly." In practice, genuine random (truly independent) points cannot be generated numerically, and instead, pseudo random points (see, for example, [14]) are used, which imitate genuine random points. However, beside [23, 24], there is, up to our knowledge, practically no research done on whether the initial population should be random.

In [23, 24], quasi random sequences are used to generate initial populations for genetic algorithms. Quasi random points do not imitate random points but are designed to maximally avoid each other [33]. Quasi random sequences are used in numerical integration [5, 35, 38], computer simulations [1, 22] and quasi random searches [21, 29, 37] with good success. For example, in [2, 3], modified controlled random searches and topographical multilevel single linkage are proposed, respectively, which use quasi random sequences when generating an initial set of solutions. The comparison shows that the proposed methods perform significantly better than the other algorithms in the comparison [2, 3]. However, in [2, 3], the influence of the use of quasi random sequences alone cannot be estimated since also other modifications are made simultaneously to the algorithms.

In this paper, we single out the effect of the different initial population for genetic algorithms by keeping the rest of algorithm identical. We study the influence of the initial population more generally than in [23, 24] and discuss the properties of different

types of point generators and the effects that different initial populations have on the convergence and the final objective function values. We also collect information and references about different ways of generating initial populations for those who are interested in alternative ways. For the convenience of the reader, we briefly summarize ways of generating points not widely used in the field of optimization. In the numerical tests, we use a well-established pseudo random number generator, a so-called Niederreiter quasi random sequence generator, which has performed well in our earlier tests [23], simple sequential inhibition (SSI) process [9] and nonaligned systematic sampling, which originates from sampling design [34]. Collectively, we call different ways to generate initial populations *point generators*. The SSI process and the nonaligned systematic sampling are commonly used in statistics where point generators are called *spatial point processes* [9]. Spatial point processes are used, for example, in the statistical analysis of biological phenomena when simulating a distribution of a population of plants or animals (see, e.g., [9, 19] and references therein).

In numerical tests, we fix the genetic algorithm and its parameter values and change only initial populations. We use a test suite of 52 test functions from 16 different function families and test whether the differences in best final objective function values found are statistically significant between the different variants of genetic algorithms. We also study the convergence during the first generations by stopping the algorithm prematurely after 10 and 20 generations.

The rest of the paper is organized as follows. In Section 2, we show that the initial population may have an effect on the convergence of a genetic algorithm and give some further motivation for this work. We also shortly present the genetic algorithm used. In Section 3, we give an overview to different point generators that can be used when generating initial populations, and in Section 4, we discuss and evaluate their speed and usability as well as the coverage and the genetic diversity of the points generated. The numerical results of applying the different variants of genetic algorithms to our test suite are presented and analyzed in Section 5. In Section 6, we discuss the results, and finally, in Section 7, we conclude the paper and present some directions for future research.

2 Preliminaries

Population-based genetic algorithms (see, e.g., [15] and references therein) are designed for solving problems that may have several local minima. They are very general problem solvers, which means that they can be used for solving a wide range of problems. On the other hand, they do not exploit problem-specific information, which makes them less efficient. Hence, genetic algorithms ought to be used when problem-specific methods are not available or if a wide range of problems need to be solved with a single algorithm. We consider *global optimization problems* of the following form

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{x}) \\ &\text{subject to} \quad x_i^l \le x_i \le x_i^u, \quad i = 1, \ldots, n, \end{aligned}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is the objective function, x_i are the decision variables and $\mathbf{x}^l$, $\mathbf{x}^u \in \mathbb{R}^n$ are the vectors containing the lower and upper bounds for the decision variables, respectively.

A simple genetic algorithm includes three basic genetic operations: selection, crossover and mutation. In selection, some solutions from the population are selected as *parents*, in crossover the parents are crossbred to produce *offspring* and in mutation the offspring may be altered according to mutation rules. In genetic algorithms, solutions $\mathbf{x}$ are called *individuals* and iterations of an algorithm are called *generations*. Many genetic algorithms also employ elitism, which means that a number of the best individuals are copied to the next population. We use a real-coded genetic algorithm that employs tournament selection, heuristic crossover, Michalewicz's nonuniform mutation and elitism. For further details of the genetic operations, see [25].

The algorithm used has the following parameters. *Population size* is the number of individuals in a population and *elitism size* is the number of fittest individuals that are copied directly to the next generation. The *fitness* is evaluated using the objective function (fitness function) value $f(\mathbf{x})$. *Tournament size* is the number of individuals randomly picked from the whole population for the tournament selection. *Crossover rate* and *mutation rate* are probabilities on which the parents are crossbred and offspring mutated, respectively. *Max generations*, *steps*, and *tolerance* are parameters for the stopping criteria. The algorithm is stopped if a maximum number of generations (*max generations*) is reached or if there is no change (within the *tolerance*) in the best objective function value during the last *steps* generations.

Genetic algorithms are Markov chains (see, e.g., [31, 46] and references therein). Such chains are expected to converge to an equilibrium distribution independent of the initial state. However, in many applications the state space of the chain (possible point configurations in the feasible region) is extremely large and convergence can be very slow. This poses the question whether the number of the generations used is sufficient in order to achieve the equilibrium. The initial configuration is one factor in the speed of convergence. This is our motivation for studying empirically the role of initial populations.

The difference in the early generations becomes important in solving many real-life problems, where the evaluation of the objective function may require time-consuming simulations and the optimization algorithm may have to be stopped prematurely. We also expect that the influence of the initial population may carry further, in terms of generations, when solving of the problem requires a large number of function evaluations.

Figures 1 and 2 illustrate the convergence of a genetic algorithm for the 10-dimensional Griewangk function and 10-dimensional Katsuura function:

$$\text{Griewangk function: } f(\mathbf{x}) = \sum_{i=1}^{10}(x_i^2/4000) - \prod_{i=1}^{10}(cos(x_i)/\sqrt{i}) \quad -10 \le x_i \le 100,$$

$$\text{Katsuura function: } \quad f(\mathbf{x}) = \prod_{i=1}^{10}\left(1 + i\sum_{k=1}^{30}\frac{|\ 2^k x_i - \lfloor 2^k x_i \rfloor\ |}{2^k}\right), \quad -0.1 < x_i < 1.$$

Each curve illustrates the average convergence for 100 separate runs of a genetic algorithm. The three different curves in Figs. 1 and 2 correspond to runs with different initial populations generated by using a pseudo random number generator. Here *pseudo* stands for the case, where the initial pseudo random population is spread out over the whole feasible region. Furthermore, *clustered 1* and *clustered 2* stand for cases where the initial population is restricted to a subspace of the feasible region. The

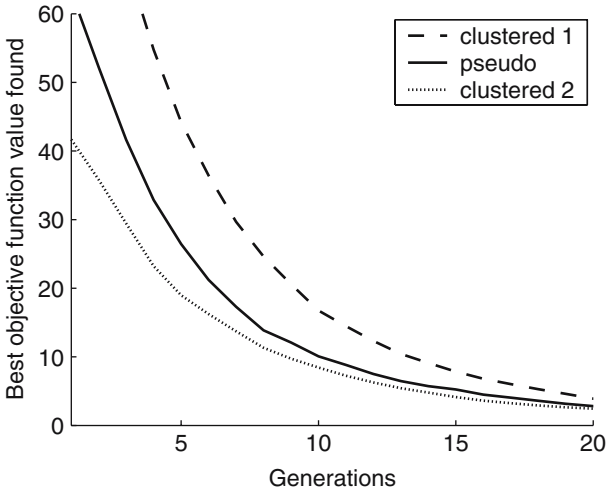

Fig. 1 The convergence of a genetic algorithm for 10D Griewangk function with different initial populations

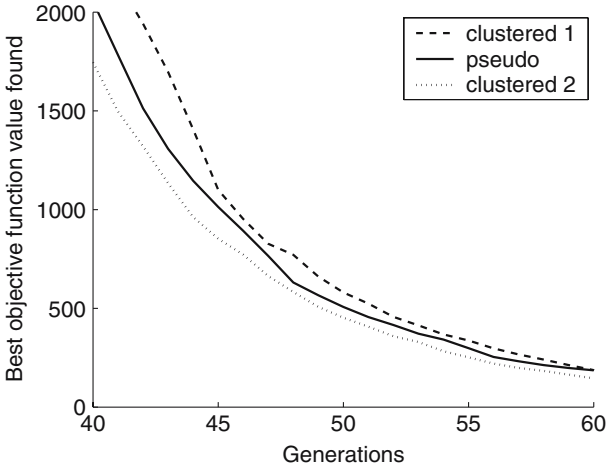

Fig. 2 The convergence of a genetic algorithm for 10D Katsuura function with different initial populations

subspace is defined by restricting each variable x_i to the upper and lower 80% of their total range for *clustered 1* and *clustered 2*, respectively. For the Griewangk function the curves merge after 30 generations. The Katsuura function requires more generations and, therefore, also the curves merge later than for the Griewangk function. (Note the different scales in Figs. 1 and 2.) For both Griewangk and Katsuura functions, the differences in the best objective function values found is quite significant when the number of generations is small. This indicates that the initial population has an effect on the convergence of a genetic algorithm.

The clustered populations used in the simple example above were only theoretical. In practice, such initial populations would hardly be used, unless there were some

a priori knowledge about the location of the global minima. The rest of the paper concentrates on more realistic alternative ways for generating initial populations.

When there is no a priori information available on the location and the number of local optima, then the initial population of a genetic algorithm should be able to reach as large part of the feasible region as possible by means of crossover. We call this property *genetic diversity*, and it is related to the independence of points. Another desirable property for an initial population is a good uniform coverage. By a *good uniform coverage* we mean that the points are well spread out to cover the whole feasible region. Points have a good uniform coverage if they do not form clusters or leave relatively large areas of the feasible region unexplored. A good uniform coverage is desired, because then information is obtained throughout the whole feasible region. This helps to prevent premature convergence.

Genetic diversity and a good uniform coverage are in practice conflicting objectives, because an optimal uniform coverage is often achieved using systematic structures, which limit the set of possible offspring. For example, it is known that for a closely related problem of sphere packing a triangular grid provides the optimal packing in a 2-dimensional case (see, e.g., [44]). Hence, if the pattern was repeated, the distance between points would reach its maximum, when using triangular grid. However, the set of possible offspring is limited as illustrated in Fig. 3. This happens also if the sample points form clusters. Therefore, seemingly random (independent) point sets with no clustering are preferred, since they provide information over the whole feasible region and a large part of the feasible region can be reached by the means of crossover. Figure 3 illustrates a rectangular grid, triangular grid, clustered points, and seemingly random points with no clustering. In the figure, the points marked with

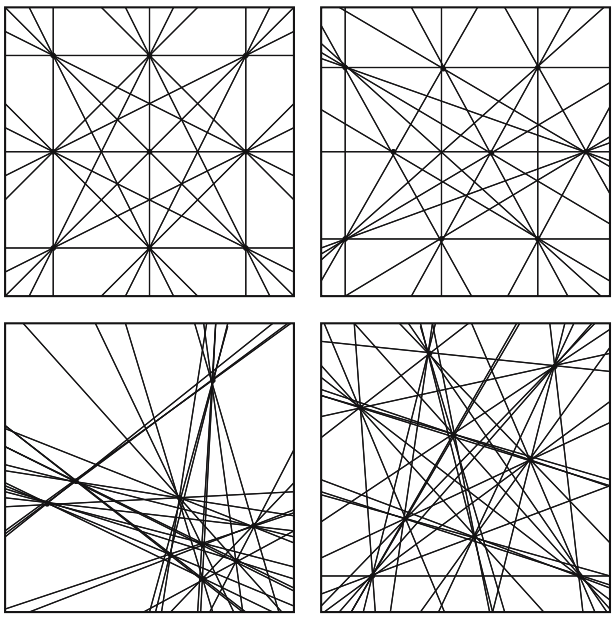

Fig. 3 Coverage of the connecting lines with different patterns

small black dots are parent solutions and the lines running through them illustrate all the possible locations of offsprings.

3 Generating initial population

Many sub-areas of genetic algorithms have been studied elaborately, but the selection of an initial population has been widely ignored. As mentioned earlier, the traditional way to generate the initial population is to use pseudo random numbers, and more recently also quasi random sequences have been applied in [23, 24]. Pseudo random numbers imitate genuine random numbers and quasi random sequences are designed to produce points that maximally avoid each other.

Pseudo random initial populations can be generated in numerous ways. The main classes are congruential and recursive generators. Common congruential generators include linear, quadratic, inversive, additive and parallel linear congruential generators [14, 45]. Recursive generators include multiplicative recursive, lagged Fibonacci, multiply-with-carry-generator, add-with-carry and substract-with-borrow generators [14]. There are also pseudo random vector generators, which produce sequences of vectors instead of scalars. Examples of those are feedback shift register generator [14] and SQRT generator [43].

Common quasi random sequence generators include Van der Corput, Hammersley, Halton, Faure, Sobol' and Niederreiter generators [4, 14, 30]. For the convenience of the reader some examples of both pseudo random number generators and quasi random sequence generators are included in the Appendix along with further references.

Pseudo random numbers and quasi random sequences are quite well-known for researchers in optimization, but there are also other types of point generators. Spatial point processes are commonly used in statistics but they are less well-known in optimization. Therefore, we now shortly describe the spatial point processes used in this paper.

3.1 Spatial point processes

Spatial point processes [9] can be considered to be transformations of pseudo random point initialization which lead to a good uniform coverage and simultaneously avoid periodicity in the configuration generated.

Some spatial point processes include a parameter, by which the proportion of the two conflicting properties, genetic diversity and a good uniform coverage, can be controlled. Hence, the same process can be used to generate genetically diverse points or points with a good uniform coverage or something in between.

There are several types of spatial point processes. For our purposes, it is adequate to differentiate between clustering and inhibition processes. *Clustering processes* generate points that simulate populations, where the individuals tend to form clusters. However, we are more interested in the situation where each individual is as far apart as possible from the adjacent individuals resulting in an evenly distributed population without clusters. Therefore, we concentrate on *inhibition processes*, where an individual (a point) is either explicitly prohibited to be located closer than some predefined minimum distance $\Delta > 0$ to the other individuals, or individuals are kept apart by some implicit means. We describe here two inhibition

processes: a simple sequential inhibition process and a nonaligned systematic sampling.

Simple sequential inhibition process: In the simple sequential inhibition (SSI) process [9], a new individual is accepted to enter the population only if its distance to all the existing individuals in the population is at least Δ.

In the following pseudo code for the SSI process the parameter *population size* is the number of points to be generated, *futile* is the number of rejected trial points during the current iteration, *Max_futile* is the maximum number of rejected trial points in one iteration before terminating the process, and n is the dimension of the space where the points are generated.

0. Initialize parameters *population size*, *Max_futile* and dimension n and set *futile* $= 0$ and $k = 0$.
1. Do until $k ==$ *population size* or *futile*==*Max_futile*
1.1 Generate a trial point.
(Use a pseudo or a quasi random number generator to generate a trial point to the n-dimensional unit hyper cube.)
1.2 Check whether the trial point is accepted.
If the distance to the existing individuals is larger than Δ, accept the point into the population, and set $k = k + 1$ and *futile*$= 0$.
Else, set *futile* $=$ *futile*$+1$.
End do

The distance between points can be computed using different metrics. If the minimum distance Δ is defined to be too large, then the maximum number of rejected trial points in one iteration is reached and the process is terminated prematurely. In that case, we supplement the sample with pseudo random points to match the *population size*.

Nonaligned systematic sampling: In the nonaligned systematic sampling, which originates from sampling design [34], the unit hyper cube is divided into b^n elementary intervals (see Appendix) with equal side lengths (i.e., an equally spaced grid). Then one sample point is selected from each elementary interval according to prescribed rules. Nonaligned systematic sampling uses $b \cdot n$ pseudo random numbers to define the location of the sample points. In two dimensions, the sample points are generated using the formula

$$\mathbf{x} = ([(j-1) + r_{i,1}]\Delta, [(i-1) + r_{j,2}]\Delta),$$

where $i, j = 1, \ldots, b$, $\Delta = 1/b$, and $\mathbf{r}$ is a $b \times n$ array of pseudo random numbers, see Fig. 4. In the following pseudo code for an n-dimensional case of the nonaligned systematic sampling, $\mathbf{v}$ is a vector of auxiliary integer variables. It is used to determine the correct elementary interval and the correct pseudo random number when calculating the nonaligned systematic sample points $\mathbf{x}$.

0. Initialize $v_j = 0$ for $j = 0, \ldots, n-1$, and the $b \times n$ pseudo random number array $\mathbf{r}$.
1. Do $i = 0, \ldots, b^n - 1$.
1.1 Compute sample point $\mathbf{x}^i$
Do $j = 0, \ldots, n-1$ // Compute the j^{th} component of $\mathbf{x}^i$
$S = \left(\sum_{k=1}^{n} v_k\right) - v_j$
$l = (S) \bmod b$
$x_j^i = (r_{l,j} + v_j)\Delta$
End do

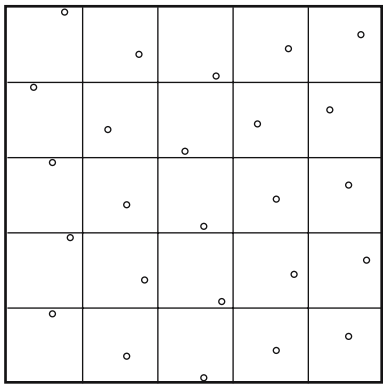

Fig. 4 Nonaligned systematic sampling points in two dimensions

1.2 Update v
Do $j = 0, \ldots, n-1$
If $((i) mod\ b^j == b^j - 1)$ then $v_j = (v_j + 1) \bmod b$
End do
End do

Note that the number of nonaligned systematic sampling points cannot be chosen freely, but is determined by the grid size and the dimension. In practice, when a certain number of sample points is required, we supplement the nonaligned systematic sample with pseudo random points to match the required number of points.

4 Properties of point generators

In this section, we consider the distribution and the genetic diversity of the points generated by pseudo random number generators, quasi random sequence generators and spatial point processes. We also consider the speed and the usability of some specific generators. We are interested in point sets with a relatively small number of points since the number of points generated equals the population size and ranges typically from tens to a few hundreds.

In the illustrations and numerical examples of point generation, we have selected good representatives for the different types of point generators. The pseudo random number generator is a multiplicative linear congruential generator from the well-established numerical library of the Numerical Algorithms Group Ltd (NAG) and the quasi random sequence generator is the Niederreiter generator [30] that has proved successful in our earlier tests [23, 24]. The representatives of spatial point processes are the SSI process and the nonaligned systematic sampling. In the SSI process, the trial points are generated using the Niederreiter generator and the distances are measured using L_2-metric. The nonaligned systematic sampling is as defined in Section 3.

The properties of the point generators are described in the following four subsections and the results are summarized and some conclusions are drawn in the fifth subsection.

4.1 Distribution

Intuitively thinking, it is beneficial if no large areas are left unexplored when sampling individuals for an initial population of a genetic algorithm. The point generators presented in Section 3 produce point clouds with a degree of uniform coverage. However, there are differences in how well the points of these sequences are spread out. In optimization and in numerical integration, the goodness of the uniform coverage of a point set is commonly measured by discrepancy or dispersion [30]. Here, we give the definition for the discrepancy.

Discrepancy Let $I^n \subset \mathbf{R}^n$ be an n-dimensional unit hyper cube, let P be a set consisting of points $\mathbf{x}^1, ..., \mathbf{x}^N \in I^n$, 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 points $\mathbf{x}^k$, $1 \leq k \leq N$, for which $\mathbf{x}^k \in B$. Then, *discrepancy* D_N 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.

Discrepancy is large, when there exists clusters or large unexplored areas. Hence, we are interested in point sets that have low values for discrepancy. A mathematical relationship between discrepancy and dispersion is given in [30], and it shows that every low-discrepancy sequence is also a low-dispersion sequence, but not vice versa.

Quasi random sequences are also called low-discrepancy sequences. Their discrepancy value for a large sample size N is of order of magnitude $C(\log N)^n N^{-1}$, where C is a generator-specific coefficient depending only on the dimension n [27]. This is also a minimum possible discrepancy size for large N (see, e.g., [27]). The bounds for discrepancy, however, are relevant only for a very large number of sample points as used in numerical integration. When generating only initial populations, we are more interested in the distribution of a small number of points. Some quasi random sequences have the property that the first b^m successive points divide evenly on the corresponding elementary intervals (see Appendix). This indicates that quasi random sequences may have good discrepancy values also for small sample sizes. Figure 5 illustrates four two-dimensional initial populations with 1,024 individuals generated using the multiplicative linear congruential (pseudo) generator, the Niederreiter generator, the SSI process and the nonaligned systematic sampling, respectively.

In Fig. 5, we see that pseudo random points form clusters and leave some areas relatively unexplored, whereas the other samples cover the feasible region quite well. However, quasi random sequences may sometimes form point patterns, whose distribution properties depend on the number of points generated. Then, at a certain number of sample points the pattern may be unfavorable for optimization purposes. An example of this behavior is illustrated in Fig. 6, where again 1,024 points were generated with a slightly different configuration. For more information on the distribution of different quasi random sequences, we refer to [27].

Next, we consider the distribution of small sample sizes. In optimization with genetic algorithms we used a population of 201 individuals. Therefore, we next empirically examine the distribution properties of sets with 201 points generated in four different ways. Henceforth, in tables and figures, we will use abbreviations *Pseudo*, *Nieder*, *SSI* and *Nonalig* for the representatives selected from different types of point generators (see the beginning of this section).

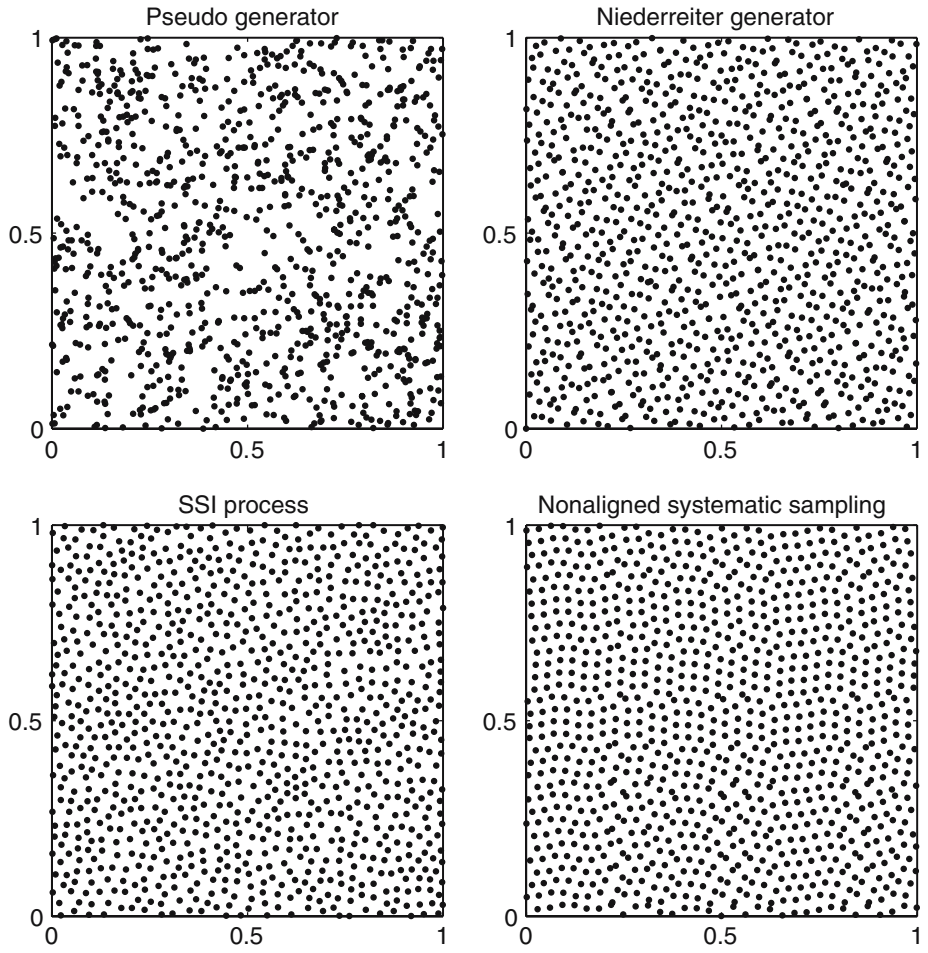

Fig. 5 Point sets of 1,024 points with different generators

We consider 2 and 10-dimensional cases to show how the distribution properties may change with the dimension. First, we use an empirical empty space statistic to examine how well the sample points cover the feasible region [34]. *Empty space statistic* function *ess* is defined as

$$ess(r) = 1 - \Pr(B(\mathbf{x}, r) \text{ is empty}),$$

where $\mathbf{x}$ is a randomly chosen point, $B(\mathbf{x}, r)$ is an $\mathbf{x}$-centered ball with radius r and Pr denotes the probability. Hence, assume there is a point set S and further assume there is a ball B with radius r and whose center point is randomly chosen from the feasible region (in our case, the unit hyper cube). Then, for each radius r, the empty space statistic function tells what the probability is that the ball B is empty, that is, the intersection of the ball B and the point set S is empty.

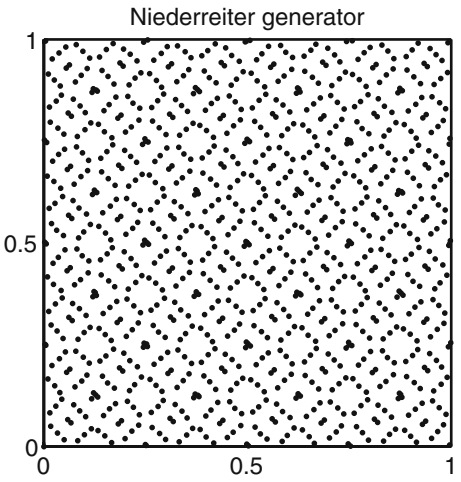

Fig. 6 Another example of 1,024 points generated with the Niederreiter generator

To experimentally define the empty space statistic functions for each point set containing 201 points, we generated 10,000 auxiliary pseudo random points[1] on a unit hyper cube and for each of the 10,000 auxiliary points calculated the maximal radius r so that $B(\mathbf{x}, r)$ is empty, that is, does not contain any of the 201 points under consideration. We did this in both 2- and 10-dimensional cases for the four different point sets.

The empirical empty space statistic functions are illustrated in Fig. 7. For us, the important property of the empty space statistic function is the steepness of the curve (the steeper the better). If the curve is steep, it indicates that if a random ball with radius r is selected from the unit hyper cube, then whether the ball is empty depends mainly on the radius—not the location of the ball. If, on the other hand, the curve is gentle, the emptiness of the ball depends strongly on the location, hence the point set is not evenly distributed.

The empirical empty space statistic functions show that in two dimensions, the non-aligned systematic sampling and the simple sequential inhibition (SSI) process provide the best coverages closely followed by the Niederreiter quasi random sequence generator. The pseudo random initial population has clearly the worst coverage, which confirms the findings earlier shown in Fig. 5. However, in ten dimensions the differences diminish, because the populations become sparse. The only notable difference is that for the population generated with the SSI process the curve of the empirical empty space statistic function is now below the other curves with the small values of r but rises more steeply in the end. This indicates better coverage for the SSI process.

4.2 Genetic diversity

The population of a genetic algorithm evolves largely by crossovers and mutations. In our implementation, the most dominant genetic operator is crossover, since it usually changes the solutions most. We define genetic diversity as a property by which the

[1] Here, we naturally used a different pseudo random number generator than when generating the initial population. To make sure that the use of pseudo random numbers did not bias the results we also computed the empty space statistic functions using a grid of 10,000 points, which led to similar results.

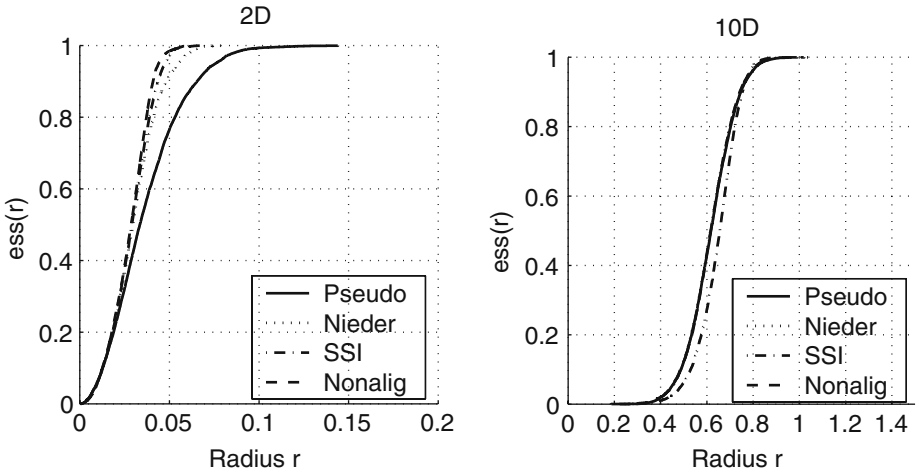

Fig. 7 Empirical empty space statistic functions

genetic algorithm is able to reach as large a part of the feasible region as possible by means of crossover. Genetically more diverse initial populations are preferable.

Next, we concentrate on the independence of points, which is one main criterion for genetic diversity. As mentioned earlier, truly independent points cannot be generated algorithmically. The pseudo random numbers try to imitate independent numbers, yet they are deterministically generated by an algorithm. For example, it has been shown that the pseudo random points generated by a linear congruential generator lie on a simple lattice (see, e.g., [14] and references therein). It is also known that bad choices for the coefficients of a linear congruential generator may result in a very bad lattice structure (see, e.g., [14]). The choices for the coefficients play a crucial role for all the pseudo random number generators.

Also quasi random sequences are generated algorithmically. However, the idea of quasi random points is not to imitate random points but to try to cover the whole search space yet maintaining a certain degree of randomness. These initial settings speak for a better genetic diversity for pseudo random initial populations. In empirical tests, when comparing pseudo and quasi initial populations, we can study plots of 2-dimensional populations or plots of n-dimensional populations that are projected to two dimensions. In Figs. 5 and 6 we can notice that the Niederreiter generator produces points that have a more distinguishable pattern whereas pseudo random points have a pattern that seems more random.

We studied the independence of point sets of 201 2-dimensional points using Ripley's K-function and L-function [34]. The K-function is defined as follows: $\lambda K(r)$ is the expected number of further points within a ball with radius r and with a center at a randomly chosen point (λ is the expected number of points in a unit hyper cube). L-function is obtained from the K-function through transformation

$$L(r) = (K(r)/\Omega_n)^{\frac{1}{n}},$$

where Ω_n is the volume of the unit ball in $\mathbb{R}^n$. Figure 8 illustrates the L-functions for the pseudo random points, the Niederreiter quasi random points, points from the SSI process and the nonaligned systematic sampling points. For independent points

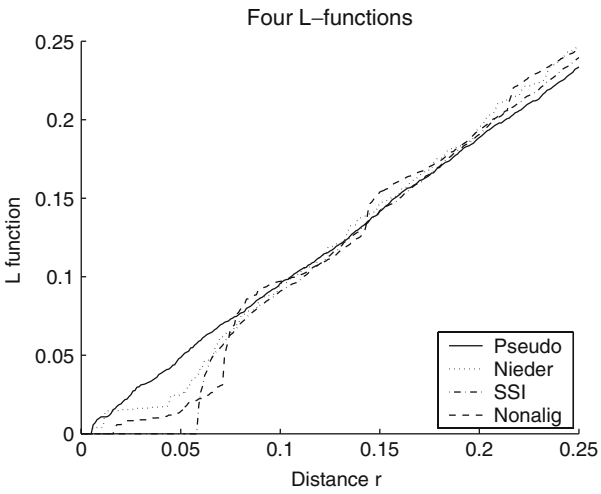

Fig. 8 L-functions for different initial populations

the L-function is a straight line $L(r)=r$. We can see that pseudo random points imitate well independent points, whereas Niederreiter quasi random points and the two spatial point processes lack the points falling close to each other. Based on these observations, we conclude that the pseudo random generator produces genetically more diverse points than the other point generators.

The points generated by the SSI process are essentially pseudo random points. However, the trial points falling too close to each other are not accepted to the population and this affects the genetic diversity. The genetic diversity versus a good uniform coverage can be controlled using the minimum distance parameter Δ. In what follows, we use the term probabilistic maximal minimum distance (PMMD) and we should find the value of PMMD so that a desired number of points can fit in the feasible region with a given probability. In the 2-dimensional example in Fig. 8, we can see that, except for small distances, the points of the SSI process fall on a straight line.

The nonaligned systematic sampling uses only $n \cdot b$ pseudo random points, where n is the dimension and $1/b$ is the side length of the elementary interval (the distance between two adjacent grid points). Therefore, there is not much genetic diversity in nonaligned systematic sample points. In Fig. 8, we see this as an oscillating behavior of the L-function. Furthermore, the number of nonaligned systematic sample points increases quickly with the dimension if the grid size is kept constant. For example, in 10 or 20-dimensional cases, even if we use only two grid points along each dimension, we get $2^{10} = 1,024$ and $2^{20} = 1,048,576$ sample points, respectively. When using a population size of 201 individuals, we get in practice only one genuine systematic sample point in 8 or more dimensions (the rest are supplemented pseudo random points). This means that, in large dimensions, the nonaligned systematic sampling reduces to pseudo random sampling.

4.3 Speed

For some applications the speed of the generator may be of importance. The CPU times for the tested generators and the different numbers of points generated in dimensions 2, 5, 10, 20, 50 are shown in Table 1. The test runs were performed on

Table 1 CPU times in seconds

Dim	Points	Pseudo	Nieder	SSI	Nonalig
2	201	0.00	0.00	0.06	0.00
2	10^4	0.01	0.00	17.60	0.02
2	10^5	0.06	0.02	1774.0	0.23
5	201	0.00	0.00	0.2	0.00
5	10^4	0.02	0.00	17.84	0.06
5	10^5	0.14	0.03	1760.0	0.67
10	201	0.00	0.00	0.23	0.00
10	10^4	0.03	0.01	17.71	0.04
10	10^5	0.28	0.05	1773.0	0.98
20	201	0.00	0.01	0.73	0.00
20	10^4	0.06	0.01	10.78	0.05
20	10^5	0.55	0.09	21.33	0.46
50	201	0.01	0.00	1.54	0.01
50	10^4	0.14	0.02	44.50	0.12
50	10^5	1.38	0.2	45.68	1.2

an HP9000/J5600 computer, and the CPU times were obtained using NAG routine X05BAF, which returns the CPU time with the accuracy of 0.01 s. For this reason, differences between some generators become evident only when using a large number of points. The column *Dim* in Table 1 indicates the dimensions of the points generated and the column *Points* lists the number of points generated. The abbreviations for the generators are the same as earlier, and the CPU times are given in seconds.

In Table 1, we see that the SSI process is by far the slowest resulting from the implementation which emphasizes the good coverage at the expense of speed. Our implementation allows a maximum of 2,000 futile trial points at each iteration before terminating the process. The speed (and the distribution of points) for the SSI process depends strongly on the probabilistic maximal minimum distance PMMD. If PMMD is selected maximally large (i.e., PMMD is as large as possible yet so that the algorithm does not have to supplement the population with pseudo random points, see Section 3.1), then a good coverage of the points is emphasized and the number of futile tries increases slowing down the process. Our implementation of the SSI process automatically estimates the maximally large PMMD. The values in Table 1 indicate that the estimate for PMMD is probably not good (distribution-wise) for dimensions 20 and 50.

Table 1 shows that the implementation of the Niederreiter quasi random number generator is the fastest followed closely by the pseudo random number generator and the nonaligned systematic sampling process. In optimization, where only a relatively small number of points is generated, there is no practical difference between the speed of these three generators. For pseudo and quasi random number generators the speed seems to be directly proportional to the number of points generated. We can see that, compared to other generators, the speed of the nonaligned systematic sampling grows for larger dimensions. This can be explained by different ratios of genuine nonaligned sampling points. As mentioned earlier, the population of nonaligned systematic sampling points is supplemented by pseudo random points to match the required population size. Hence, for example, in 5 dimensions there are 78% and 100% of genuine nonaligned systematic sampling points in sets of 10,000 and 100,000

sample points, respectively, whereas in 10 dimensions the respective values are only 10% and 59%. In 20 dimensions, all except one sample point are already random even if the sample size was as large as one million ($2^{20} > 10^6$).

4.4 Usability

In this section, we discuss the usability of different number generators. Some features affecting usability have already been mentioned earlier and are summarized here.

Pseudo random number generators are by far the most commonly used ones in optimization. The greatest advantage of pseudo random number generators concerning usability is that there exist implementations that are well-established, well-tested and easily available. However, a good random-like behavior is not a matter-of-course, but requires a careful choice of parameters, that is, coefficients and seeds (or initial sequences). As earlier noted for linear congruential generators, wrong choices for parameter values may cause the points in a sequence to be badly distributed. This is true for other generators as well. Many of the pseudo random number generators are easy to implement, but the analysis of the distribution is difficult, especially for an arbitrary seed (or arbitrary initial sequences). According to [14], only well-tested pseudo random generators should be used. These generators often use fixed coefficients and sometimes limit the choice of seeds (or initial sequences). This is one way to secure good distribution properties, assuming those values are tested and approved.

Contrary to pseudo random number generators, quasi random number generators are neither so well-established, well-tested nor easily available. On the other hand, quasi random number generators having solid theoretical properties do not need so much numerical testing. The use of available quasi generators is easy and the quasi random points are guaranteed to have some advantageous properties as described in the Appendix. Quasi generators provide the same sequences on different runs of the generator and, hence, they do not require a seed or an initial sequence from the user. Since quasi generators take into consideration the location of the previous points, they also expect that the sequence is started from the beginning. These two properties make quasi generators easier to use, because the user only has to give the number of points to be generated and the dimension as parameters. At the same time, these properties may become a disadvantage if the user for some reason wants to obtain different sequences in the same dimension. On this occasion, the easiest way out might be to generate points in higher dimensions and then project them to the desired dimension, but this may have an effect on the distribution properties.

The SSI process is not commonly available for an n-dimensional case, but the implementation is straightforward given that the user provides the PMMD value. However, it gets more complicated if the user provides only the number of points to be generated and the optimal PMMD (with respect to the coverage) must be estimated automatically. This issue is discussed in more detail in Section 6. An interesting property of the SSI process is that the proportion of the two conflicting objectives, that is, the genetic diversity and the good uniform coverage, can both be controlled using the probabilistic maximal minimum distance parameter PMMD. Note that, if desired, the SSI process provides different points on every run just like pseudo random number generators.

The nonaligned systematic sampling processes are not commonly available in n-dimensions, but they are easy to implement and to use. However, when

generating a relatively small number of points, it is practical to use the systematic sampling process only in small dimensions. Otherwise, the proportion of genuine nonaligned systematic sample points is small. This is a strong limitation in the area of optimization.

4.5 Summary of features

An ideal generator for our purposes should generate well-distributed, genetically diverse points in n-dimensions, and it should be relatively fast and easy to use. In Table 2, we summarize the evaluation made in this section. The evaluation of different point generators has been marked using plus signs (+): the more plus signs the better the score. In case of a notable difference in how well a generator works for problems with small and large dimensions (here small denotes less than, say, five), the occasions are scored separately (small/large). For example, the nonaligned systematic sampling scores +++/+ in coverage, which means that it works well in small dimensions and poorly in large dimensions.

Note that only coverage and genetic diversity may have a direct influence on objective function values in optimization since they are the properties of the points whereas speed and usability are properties of the generators. Considering just the properties of the points we notice in Table 2 that pseudo random points have good genetic diversity, but the worst coverage, and the SSI process produces points with good coverage, but only average genetic diversity. The properties of the Niederreiter quasi random points settle somewhere between pseudo and the SSI process. In the further analysis, we pay less attention to the nonaligned systematic sampling since it is not applicable for problems with several variables. To find out the effects of the coverage and genetic diversity, we concentrate on the pseudo random number generator and the SSI process.

5 Experimentations

We test the influence of initial populations computationally by using different point generators and a large number of difficult test problems from the literature. The influence of the different initial populations is analyzed after 10 and 20 generations and after the execution of the whole algorithm. The genetic algorithm used is presented in Section 2.

5.1 Test settings

The test runs were performed on an HP9000/J5600 computer. We solved a test suite of 52 problems using genetic algorithms described in Section 2 with the parameter values

Table 2 Summary of generator properties

Properties (small/large)	Pseudo	Nieder	SSI	Nonalig
Coverage	+	++	+++	+++/+
Genetic diversity	+++	++	++	+
Speed	+++	+++	+	++/+++
Usability	+++	+++	++	+++/+

Table 3 Parameter values for genetic algorithms

Parameter	Value
Population size	201
Elitism size	21
Tournament size	3
Crossover rate	0.8
Mutation rate	0.1
Max generations	10, 20 and 2000
Steps	100
Tolerance	10^{-7}

given in Table 3. The only difference between the variants of a genetic algorithm used was the way the initial population was generated. Each problem was solved 10 times with each algorithm, when the algorithms were let run until the stopping criteria were satisfied, and 100 times when the algorithms were stopped prematurely after 10 and 20 iterations. In each run, we recorded the best objective function value in the last population and study them in what follows. The names of the function families, number of variables, box constraints used and references are collected in Table 4. If only one interval is given for the box constraints in Table 4, then each variable was restricted to the same interval. The test problems are the same as used in [24]. They are divided into two problem sets according to the number of variables so that problems with 10 or less variables are here called small and others large.

Originally, initial populations were generated in altogether 21 different ways including 13 variants of the SSI process, 5 quasi random number generators, a nonaligned systematic sampling and a pseudo random number generator. The variants of the SSI process differed in the random number generator used when generating the trial points and in the metric used when computing the probabilistic maximal minimum distance PMMD. The quasi random number generators tested were the Niederreiter,

Table 4 Summary of the test problems

#	Name	Dimensions	Box constraints	Ref.
1–3	Michalewicz	2, 5, 10	$[0, \pi]$	[13]
4–7	Rastrigin	6, 10, 20, 30	$[-600, 400]$	[13]
8–11	Schwefel	6, 10, 20, 50	$[-500, 500]$	[13]
12	Branin rcos	2	$[-5, 10]\times[0, 15]$	[13]
13–17	Griewangk	2, 6, 10, 20, 50	$[-700, 500]$	[13]
18–22	Ackley's Path	2, 6, 10, 20, 30	$[-30.768, 38.768]$	[13]
23	Easom	2	$[-100, 100]$	[13]
24	Levy	4	$[-10, 10]$	[40]
25–27	Levy	5, 6, 7	$[-5, 5]$	[40]
28	P8	3	$[-10, 10]$	[7]
29	P16	5	$[-5, 5]$	[7]
30	Hansen	2	$[-10, 10]$	[40]
31–34	Corona	4, 6, 10, 20	$[-900, 1100]$	[10]
35–38	Katsuura	4, 6, 10, 20	$[-1, 1]$	[10]
39–42	Langerman	5, 5, 10, 10	$[0, 10]$	[41, 13]
43–46	Funtion 10	3, 4, 10, 20	$[-20, 20]$	[42]
47–49	Shekel	4, 4, 4	$[0, 10]$	[40]
50–52	Epistatic Michalewicz	2, 5, 10	$[0, \pi]$	[41]

Faure and Halton generator and two variants of the Sobol' generator. The numerical results of the genetic algorithm with quasi random initial populations are reported in [23], where the Niederreiter generator performed the best. Here, we report the results of the genetic algorithm using initial populations generated by the good representatives from different types of point generators (see the beginning of Section 4).

5.2 Numerical results

In this subsection, we present the results of the experiments described above. In what follows, we consider only the best objective function values that the algorithm has found (in the last population) and pay attention to the average and variance of these values in the repeated runs.

Let us first consider those tests where the genetic algorithm was run until a stopping criterion was satisfied. Table 5 shows the average final objective function values $\overline{f}(x)$ and the average standard deviations $\overline{\sigma}(f)$ for small and large problems. As we can see, for small problems, in the average objective function values, there are no differences in three significant digits. However, for large problems the differences in average objective function values are quite large. Based solely on the average objective function values in Table 5 one could draw the conclusion that nonaligned systematic samples and Niederreiter quasi random points are clearly the most suitable for initial populations. This, however, is not completely true as our further analysis reveals, when we consider also the variances.

Before any statistical tests, we make an observation on the results in Table 5. As mentioned in Section 4.2, nonaligned systematic sampling reduces to pseudo random sampling for large problems. However, the average objective function value and the average standard deviation of those algorithms differ quite remarkably for large problems. This indicates that the average values are not good measures in the comparison.

We applied analysis of variance (ANOVA) to the final objective function values to find out whether the differences were statistically significant. The analysis of variance indicated that only in 4 out of 52 test problems there were statistically significant differences (with 95% confidence). We call these four problems *critical problems*, and they are the problems 40, 49, 51 and 52 in Table 4. Surprisingly, all the critical problems were small (problems with 10 or less variables). This can be explained by the small variance for these problems. The fact that there was no statistically significant difference in the objective function values for any of the large problems is explained by their very large variances. The variances were the largest for the 20-dimensional Katsuura problem for which the variances were 8,473, 2,060, 509 and 17,100 for the pseudo, Niederreiter, SSI and nonaligned, respectively. The 20-dimensional Katsuura

Table 5 Average objective function values and standard deviations for small and large problems

	Pseudo	Nieder	SSI	Nonalig
$\overline{f}(x)$ Small	−186.7	−186.7	−186.8	−186.6
$\overline{f}(x)$ Large	−1896.0	−2205.4	−1680.2	−2295.9
$\overline{\sigma}(f)$ Small	0.34	0.28	0.37	0.58
$\overline{\sigma}(f)$ Large	885.9	355.3	162.5	1697.9

problem biased the average values in Table 5 and was the main cause of differences for large problems.

Next, we study the critical problems more closely. Figure 9 shows the average objective function values for the four critical problems. *Ps*, *Ni*, *SSI* and *No* stand for the pseudo, Niederreiter, SSI process and nonaligned systematic sampling, respectively, and the whiskers illustrate the standard deviations. In addition, the *P* values related to the *F* test of ANOVA [28] are given in the parenthesis subsequent to the problem number. The four plots reveal that there is no overall dominance between the genetic algorithm variants, however the variant applying quasi random points performed best in 3 out of 4 cases and also points generated by the SSI process performed better than pseudo random points in 3 out of 4 instances. Nonaligned systematic sampling performed very similarly to pseudo random points in three cases and was better in one.

So far we have studied the differences in objective function values after running the whole algorithm and there has been only minor differences. Next, we consider also the situations, where the algorithm is stopped prematurely after 10 and 20 generations. Earlier, we noticed difficulties in interpreting values that were not scaled: one large value could strongly bias the results. Therefore, we now define a more reliable measure of performance by scaling the objective function values to the range from 0 to 1. Scaled average objective function value $\overline{sf}$ is defined as follows

$$\overline{sf} = \frac{\overline{f} - \overline{f}_{\min}}{\overline{f}_{\max} - \overline{f}_{\min}},$$

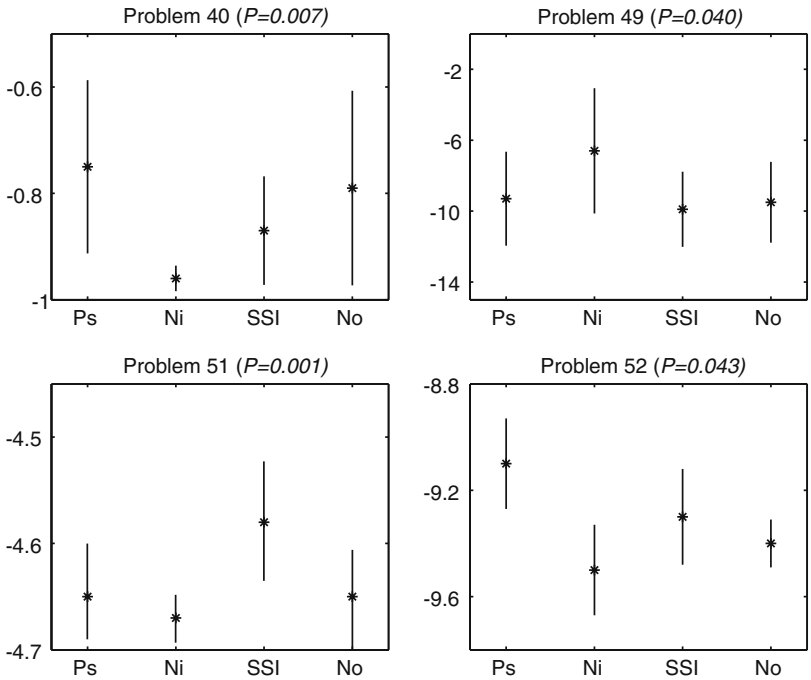

Fig. 9 Average objective function values and standard deviations

where $f_{\min}$ is the best objective function value known (or found after running all the four whole algorithms), and $f_{\max}$ is the worst of the best objective function value found during the repeated runs (either for a fixed number of generations or till a stopping criterion). It is worth noting that $f_{\max}$ changes each time the stopping criteria are changed, but $f_{\min}$ typically remains the same (at least if it is known). Finally, a bar above a function symbol denotes the average value over all the considered test problems (small or large), where the difference between (unscaled values of) $f_{\min}$ and $f_{\max}$ was more than 0.1. Table 6 shows the scaled average objective function values for small and large problems.

We remind again that for large dimensional problems the nonaligned systematic sampling points and pseudo random points are essentially the same, and their results are here reported only for the sake of completeness. We now notice in Table 6 that pseudo and nonaligned systematic sampling receive similar values, which gives indication that the values are more reliable after scaling.

The scaled average objective function value $\overline{sf}$ becomes unstable near the optimum since the divisor $\overline{f}_{\max} - \overline{f}_{\min}$ approaches zero and, therefore, the values for 10 and 20 generations are more reliable than the values after running the whole algorithm. Nevertheless, Table 6 shows an interesting trend: the genetic algorithm variants that converge relatively fast in the first generations are not necessarily the ones obtaining the best final results. This is particularly true for the SSI process, which had the best empty space statistic values in Section 4 and in the numerical results in Table 6 receives worst values during the first generations, but good values at the end. Pseudo random points, on the other hand, had the best genetic diversity, and, in Table 6, they receive good values during the first iterations. The Niederreiter quasi random points had above average genetic diversity and above average coverage and their performance in numerical tests in Table 6 is above average except for one instance. Despite the observations made above, the differences in the results shown in Table 6 are debatable.

We applied ANOVA also for the intermediate results after 10 and 20 generations. Again, we call those problems critical, where there were statistical differences in the objective function values. However, we now exclude the problems, where the differences in objective function values were less than 0.1. There were all together 38 critical problems after 10 generations and 9 after 20 generations. In Table 7, we report the number of critical problems for which each algorithm had the best average intermediate objective function value. The differences are not large, but pseudo random (and nonaligned systematic sampling) initial population seem to provide best intermediate results.

Table 6 Scaled average objective function values after 10 and 20 generations and after running the whole algorithm

	Generations	Pseudo	Nieder	SSI	Nonalig
$\overline{sf}(x)$ Small	10	0.90	0.91	0.95	0.89
$\overline{sf}(x)$ Small	20	0.85	0.88	0.9	0.87
$\overline{sf}(x)$ Small	all	0.52	0.38	0.43	0.57
$\overline{sf}(x)$ Large	10	0.84	0.85	0.99	0.84
$\overline{sf}(x)$ Large	20	0.91	0.90	0.99	0.87
$\overline{sf}(x)$ Large	all	0.45	0.61	0.38	0.42

Table 7 The number of problems with best average objective function values for critical problems

Generations	Pseudo	Nieder	SSI	Nonalig	Total
10	9	9	8	12	38
20	5	1	0	3	9

Before summarizing the results, let us yet consider the magnitudes of the variances in the best objective function values separately from the actual objective function values. In Table 8 are reported the number of test problems for which each algorithm had the smallest variance after 10 and 20 generations and after running the whole algorithms. We have excluded the problems where the differences in variances were less than 0.1. The total number of problems considered is reported in the last column of Table 8. It is noteworthy that the variant of genetic algorithm using pseudo random numbers has most often the smallest variance for the intermediate solutions, which was not expected.

5.3 Summary of numerical results

Here, we summarize the analysis made above. When considering final and intermediate average objective function values, the analysis of variance indicated four critical problems for the final results and many more for the intermediate results where there were significant differences in the best objective function values found. None of the methods clearly dominated the others, but in the scaled objective function values there was a noticeable trend, namely that pseudo random initial populations provided on the average good intermediate results, but not so good final results. The converse was true for the SSI process and quasi random points with better uniform coverage. Earlier we noticed that some clusterizations were harmful and some beneficial. Hence, genetic algorithms using pseudo random numbers and the variants where the initial populations have better uniform coverage performed in a slightly different way. Nevertheless, the differences, in general, were debatable and no strong conclusions could be made.

As far as the magnitudes of variances in the intermediate results are concerned, in most of the cases, the variance was the smallest for the genetic algorithm using pseudo random numbers. There may be both harmful and beneficial clusterizations as seen in Figs. 1 and 2. Harmful clusterizations lead to inferior and beneficial lead to superior objective function values compared to initial populations without clustering. One could have expected this to imply more significant differences in the magnitudes of variances in the early generations but this did not happen. However, again, none of the algorithms clearly dominated the others and the differences were not large.

Finally, there were no significant differences in the magnitudes of variances related to final objective function values when applying clustered or unclustered initial popu-

Table 8 Number of test problems when the variance was smallest

Generations	Pseudo	Nieder	SSI	Nonalig	Total
10	18	11	8	9	46
20	12	11	6	15	44
Final	5	9	3	2	19

lations. This could be assumed keeping in mind the observation that genetic algorithms are Markov chains and, hence, converge independently of the initial population. However, the conclusion is not trivial, since actually the Markov chain property only guarantees convergence when the number of generations approaches infinity. The last row in Table 8 shows that the genetic algorithm applying pseudo random initial population has more often smallest variance than the one applying the SSI process, but less often than the one applying quasi random sequences. Hence, the clusterization of the initial population did not seem to have strong effect on the variance of the final objective function values.

Despite some differences, the results indicate that pseudo random points perform rather well when used in initial populations of genetic algorithms. Moreover, if the minor differences found in this research can be generalized, then they speak for pseudo random points in cases when the algorithm is stopped after relatively small number of generations, which may be the case with some computationally expensive real life problems.

6 Discussion

In this section, we consider four problematic issues that we have touched earlier. First, we point out that dimensionality plays a large role in this research. The initial populations become sparse when the optimization problem has more than few variables. A simple example of this can be seen with unit hyper cubes in three and two dimensions. In three dimensions, eight point can be located in the corners of the cube so that their maximal minimum distance is always one. But locating eight points in the line segment of length one means that the maximal minimum distance between the points is 1/7. The sparsity in higher dimensions means that solutions in the initial population are likely to be relatively far from the global optimum no matter what uniform distribution pattern is used. Moreover, in sparse populations, when the minimum distance between two adjacent points in the population is maximized, that is, when the points in the population try to maximally avoid each other, then the population is likely to be concentrated to the borders of the feasible region, which may not always be desirable. This phenomenon is typical of the SSI process and could also be seen in the example mentioned above.

The second issue is of more technical nature and concerns the SSI process. The SSI process can produce points that are either genetically diverse or have good uniform coverage, depending on the distance parameter PMMD. We optimized the coverage, but did not test values for PMMD that would compromise between the two conflicting objectives. Smaller values for Δ would also make the process considerably faster. For example, in Fig. 10a we have seven point in an interval of length one with $\Delta = 1/7$. We know that one more point can be included but how many random points should be generated before the right location is found?

When optimizing the coverage for the SSI process the automatic determination of a maximal minimum distance PMMD is problematic. We did not find good theoretical estimates for PMMD and the theoretical estimates we tested did not work well

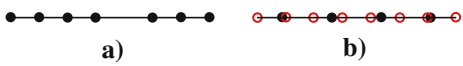

Fig. 10 Difficulties with speed and approximation of Δ

in practice. Hence, we used experimental estimates. In our current implementation, we have numerically solved maximal values of PMMD for some number of points in some dimensions, and we approximate maximal PMMD elsewhere using interpolation. Examples of values used for Δ in our experiments for different numbers of variables (with population size 200) include 0.032 (for $n = 2$), 0.23 (for $n = 5$), 0.49 (for $n = 10$), 0.64 (for $n = 20$), 0.79 (for $n = 50$) and 0.84 (for $n = 100$). Some further research is needed before the SSI process with maximal values for PMMD can be used in practical applications. An example of the difficulty of approximating the value of Δ is given in Fig. 10b. Let us assume that we have fixed $\Delta = 1/7$ and the distance between the end points of the interval and outermost points is $1/8$ and the distance between intermediate points is $2/8$. Here we have only four points (denoted by black dots in the figure) and cannot fit any more points even though eight points could be fit in the interval in the optimal case (denoted by circles in the figure). In other words, it is very hard to determine the maximal minimum distance a priori when points are generated randomly.

The third issue concerns the speed of point generators. We point out that the computational complexity of an algorithm may depend strongly on the implementation, see, e.g., [4], where bit-operations are considered. Thus, the results on the speed of the generators can be considered only suggestive.

The last issue concerns the testing of point generators. There exists a large variety of dynamic statistical tests for the independence of points generated by point generators [14]. These tests are called goodness-of-fit tests. One well-established battery of goodness-to-fit tests is called DIEHARD and the source code is available at [8]. However, the goodness-to-fit tests are designed to test the independence of large numbers of points. For example, DIEHARD requires a binary file of a size 10–11 megabytes to evaluate a generator. Since we are only interested in a few hundred numbers at the beginning of the sequence, we used the Ripley's K-function instead of the goodness-to-fit tests.

7 Conclusions and future research

In this paper, we have tested different initial populations for a real coded genetic algorithm. Traditionally, pseudo random numbers are used for generating initial population, and our motivation has been to study and initiate discussion on whether their use is justified.

We have shown with a simple, academic, example that initial populations may have a significant effect on the best objective function value over several generations. Then we have concentrated on studying different realistic ways to generate initial populations for a case with no a priori information on the location of the global minima. We have briefly summarized the basic properties of a pseudo and a quasi random sequence generator, the SSI process and the nonaligned systematic sampling and applied them. We have discussed their properties including genetic diversity and a good uniform coverage for the points as well as speed and usability for the generators. In the numerical tests with genetic algorithms, we have used a test suite of 52 functions from the literature. We have studied the effects of the different initial populations on the best objective function values after 10 and 20 generations and after running the whole algorithm.

There were differences in the coverage and genetic diversity of the tested point sets. The SSI process has a good uniform coverage, but only average genetic diversity, and for pseudo random points it is vice versa. The Niederreiter quasi random points have both above average coverage and above average genetic diversity.

In the numerical experiments with the genetic algorithms, there was a trend—although weak—showing that the versions of genetic algorithm with good genetic diversity converged fast during the first generations, but did not obtain the best final objective function values on the average. The converse was true for the versions with good uniform coverage. However, the differences were not so large that any strong conclusion could be drawn.

With respect to the speed and usability of the point generators the results show that pseudo and quasi random sequence generators are fast and easy to use. Both the SSI process and the nonaligned systematic sampling require more developing and testing.

We conclude that, based on our research, the traditional way to generate initial populations of genetic algorithms using pseudo random number generator was not worse than the others. It is particularly well suited to cases where the algorithm must be stopped prematurely; which may happen with computationally expensive real life problems. However, there are also good alternative ways such as quasi random sequences and the SSI process, which have advantages in certain cases, especially if the goodness of the final solution is valued higher that the speed of convergence during the first generations.

Our findings show that paying attention to the initial population may have an effect on the success of the genetic algorithm and further research and discussion is encouraged. The topics for future research include, firstly, to study different initial populations for specific types of problems and with different genetic algorithm parameters like population size and maximum number of generations. Secondly, to study more closely the problem of dimensionality discussed in Section 6, and thirdly, to further develop the SSI implementation and to discover good theoretical estimates for the maximal minimum distance PMMD.

Acknowledgements The authors thank doctors Salme Kärkkäinen and Marko M. Mäkelä for useful discussions and professor Michael Mascagni for providing the Niederreiter generator. This research was supported by the Jenny and Antti Wihuri Foundation and the Academy of Finland, grant number 102020.

Appendix A

This appendix is designed to give a short overview to different pseudo random number and quasi random sequence generators. In the following definitions we use $(y)\mathrm{mod}M$ to denote y modulo M.

A.1 Some pseudo random number generators

Traditionally, pseudo random number generators produce sequences of scalars. If vectors are needed, then they are usually formed by taking the first n scalars for the first n-dimensional vector, the next n scalars for the second vector and so forth. According to [30], it would be preferable to generate pseudo random vectors directly. Here, we present traditional pseudo random number generators and one vector generator.

We classify pseudo random number generators to two main categories: congruential and recursive generators. We call a method congruential, if it uses modulo and only the previous iteration value, and we call it recursive, if it uses values from several previous iterations. In addition, we present a feedback shift register generator and a vector generator called SQRT-generator, which do not fall into the two main categories. For another classification, see, e.g., [39]. The classifications are always somewhat vague. Feedback shift register generator also uses values from several previous iterations, but its construction differs significantly from recursive generators, and therefore, we put it in its own class. For more detailed information about pseudo random number generation we refer to [14] and to references therein.

A.1.1 Congruential generators

Next, we present five congruential generators. They are called linear, quadratic, inversive, additive and parallel linear congruential generators. The name congruential comes from the use of modulo. For example, in modulo 5, the numbers 4 and 9 are called congruent. In what follows, M is the modulo and a_j's are prescribed integers. On the ith iteration, the generators first produce an integer $y^i \in [0, M)$ and then a random number $x^i \in [0, 1)$ is obtained by a division $x^i = y^i/M$ unless another formula is given. The *seed* y^0 is a large prescribed integer. An additive congruential generator (of kth order) [45], to be described below, requires $k + 1$ seeds $0 \leq y_j^0 < M, j = 0, \ldots, k$, and the parallel linear generator includes three separate linear generators denoted by y^i, $\hat{y}^i$ and $\hat{\hat{y}}^i$.

$$
\begin{array}{lll}
\text{Linear} & y^i & = (a_1 y^{i-1} + a_2) \bmod M \\
\text{Quadratic} & y^i & = (a_1 (y^{i-1})^2 + a_2 y^{i-1} + a_3) \bmod M \\
\text{Inversive} & y^i & = (a_1 \left(\frac{1}{y^{i-1}}\right) + a_2) \bmod M \\
\text{Additive} & y_0^i & = y_0^{i-1} \\
 & y_m^i & = (y_{m-1}^i + y_m^{i-1}) \bmod M, m = 1, ..., k \\
 & x^i & = \frac{y_k^i}{M} \\
\text{Parallel linear} & y^i & = (a_1 y^{i-1}) \bmod M_1 \\
 & \hat{y}^i & = (a_2 \hat{y}^{i-1}) \bmod M_2 \\
 & \hat{\hat{y}}^i & = (a_3 \hat{\hat{y}}^{i-1}) \bmod M_3 \\
 & x^i & = (\frac{y^i}{M_1} + \frac{\hat{y}^i}{M_2} + \frac{\hat{\hat{y}}^i}{M_3}) \bmod 1.
\end{array}
$$

In all of the above generators $i = 1, 2, \ldots$ Often, a distinction is made between linear congruential generators with a_2=0 and $a_2 \neq 0$. Then a generator with a_2=0 is called a multiplicative linear congruential generator and a generator with $a_2 \neq 0$ a mixed linear congruential generator.

A.1.2 Recursive generators

Generators using two or more previous random numbers when generating a new number in the sequence are here called recursive generators. In what follows, a_j, c^i, s,

and r are prescribed integers $s < r$, $j = 1, \ldots, k$, and c^i is called the *carry* operator on the ith iteration. The determination of c^i is omitted for the multiply-with-carry-generator and also for the add-with-carry and substract-with-borrow generators using more that two previous values. Again, the pseudo random number $x^i \in [0, 1)$ is obtained by a division $x^i = y^i/M$. For more details about the carry operator, see, e.g., [6]. Next, we present five recursive generators.

Multiplicative recursive	$y^i = (a_1 y^{i-1} + \cdots + a_k y^{i-k}) \bmod M$
Lagged Fibonacci	$y^i = (y^{i-r} - y^{i-s}) \bmod M$
Add − with − carry	$y^i = (y^{i-s} + y^{i-r} + c^i) \bmod M$, where $c^i = \begin{cases} 1, & y^{i-s} + y^{i-r} + c^{i-1} \le M \\ 0, & \text{otherwise} \end{cases}$
Substract − with − borrow	$y^i = (y^{i-s} - y^{i-r} - c^i) \bmod M$, where $c^i = \begin{cases} 1, & \text{if } y^{i-s} - y^{i-r} - c^i < 0 \\ 0, & \text{otherwise} \end{cases}$
Multiply − with − carry	$y^i = (a_1 y^{i-s} + a_2 y^{i-r} + c^i) \bmod M$, where c^i is computed using c^{i-1} and previous values of y^i.

Again $i = 1, 2, \ldots$ and y^0 is the seed. The number of previous values of y used in the methods described above may alter. In addition, the lagged Fibonacci generator may generate new values also as a sum or a product of the previous values.

A.1.3 Feedback shift register generator

Feedback shift register generator uses modulo in a different manner compared to the congruential and recursive generators. Modulo M is not a large integer, but very often $M = 2$ in which case the generator produces a sequence $\{\alpha^j\}$ of zeros and ones. The sequence is then cut into subsequences with an appropriate length. These subsequences correspond to y^i's. To generate the sequence $\{\alpha^j\}$, first a primitive polynomial $q(z)$ of order p is selected

$$q(z) = z^p - (a_1 z^{p-1} + \cdots + a_{p-1} z + a_p), \tag{1}$$

where $a_i \in \{0, 1, \ldots, M-1\}$, $i = 1, \ldots, p$. Then using the coefficients a_i, the nth number in the sequence is generated using the formula

$$\alpha^n = (a_p \alpha^{n-p} + a_{p-1} \alpha^{n-p+1} + \cdots + a_1 \alpha^{n-1}) \bmod M. \tag{2}$$

The initial numbers $\alpha^1, \ldots, \alpha^p \in [0, M)$ can be selected arbitrarily.

A.1.4 SQRT sequence

The following definition of the SQRT sequence is taken from [43]. Let p_i now be the *ith* prime number and $z_i = \sqrt{p_i}, i = 1, \ldots, n$. Then the n-dimensional SQRT sequence $\{\mathbf{x}^i\}$ is defined by

$$\mathbf{x}^i = (i\mathbf{z}) = (iz_1, \ldots, iz_n) \bmod 1,$$

where the modulo is taken component-wise. Despite its simplicity, the SQRT sequence performs well in [43], where it is applied to a numerical integration problem and compared with five quasi random sequences.

A.2 Some quasi random sequence generators

To describe the structure and the distribution properties of some quasi random sequences let us define an *elementary interval in base b*

$$E = \prod_{i=1}^{n} \left[\frac{a_i}{b^{d_i}}, \frac{a_i + 1}{b^{d_i}} \right),$$

where a_i, d_i and b are integers $d_i \geq 0$, $0 \leq a_i < b^{d_i}$ for $i = 1, \ldots, n$. Thus, E is a subinterval of an n-dimensional unit cube $I^n \subset \mathbf{R}^n$ and its ith side has a length of $1/b^{d_i}$. Figure 11 illustrates 2-dimensional elementary intervals with base $b = 2$ and $d_1 = d_2 = 2$, and the colored area corresponds to values $a_1 = 1$ and $a_2 = 2$.

Quasi random sequences are called (t, s)-sequences if they satisfy the following distribution property (s is the dimension that we denote by n). For all integers $k \geq 0$, the point set $\{\mathbf{x}^j\}$ of (t, s)-sequence with $kb^m \leq j < (k + 1)b^m$ has exactly b^t points on every elementary interval in base b with volume b^{t-m} (in L_2-metric) [36]. The distribution properties for a (t, s)-sequence are the most preferable, when the distribution parameter $t = 0$, because then the first b^m successive points divide evenly on the corresponding elementary intervals and the same is true for the following b^m successive points and so on.

The class of (t, s)-sequences was introduced by Sobol' in 1966 (see, [36]). He called them LP_τ-sequences ($\tau \equiv t$) and studied them in base 2. Faure generalized the

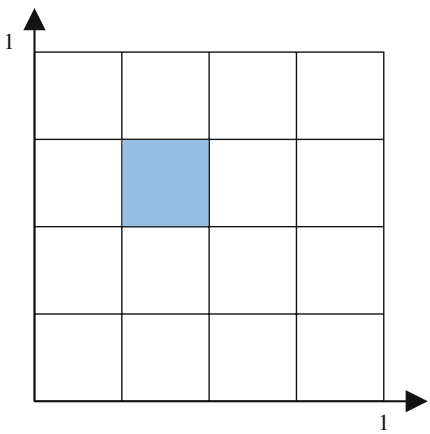

Fig. 11 Two-dimensional elementary intervals

(t,s)-sequences to arbitrary prime base $b \geq 2$ (see, e.g., [36]). Finally, Niederreiter generalized (t,s)-sequence to arbitrary base $b \geq 2$.

Often a (t,s)-sequence is called the Sobol' sequence if $b = 2$ and $t = 0$, and Faure sequence if b is a prime and $t = 0$ [30]. Sequences that are constructed according to the guidelines given in [30] are called Niederreiter sequences. For a general construction of any (t,s)-sequence, see, e.g., [20, 30].

Next, we present some examples how to construct (t,s)-sequences. We omit all implementational details.

A.2.1 Van der Corput sequence

Let $b \geq 2$ be the base and k be an integer. If we write k in base b

$$k = (d_j ... d_1)_b,$$

where $d_i \in \{0, \ldots, b-1\}$, $i = 1, \ldots, j$, and define a *radical inverse* function ϕ_b by

$$\phi_b(k) = \frac{d_1}{b} + \cdots + \frac{d_j}{b^j} = (0.d_1 \ldots d_j)_b, \tag{3}$$

then Van der Corput sequence in base b is the sequence $\{\phi_b(k)\}_{k=0}^{\infty}$. Informally, we may say that in the radical inverse the numbers $d_j, \ldots, d_1$ are reflected using the decimal point as a reflector, for example, if $k = 12 = (1100)_2$, then $\phi_2(12) = 0.0011_2$. Van der Corput sequence is a 1-dimensional (t,s)-sequence with $t = 0$ (see, e.g., [30]).

A.2.2 Hammersley sequence

All but the first component of the n-dimensional Hammersley sequence are van der Corput sequences in an appropriate base. The n-dimensional Hammersley sequence is defined as

$$\mathbf{x}^i = (i/N, \phi_{b_1}(i), \ldots, \phi_{b_{n-1}}(i)), \quad i = 0, 1, \ldots, N-1 \tag{4}$$

where N is the number of points to be generated and the bases $b_1, \ldots, b_n$ are integers greater than one and pairwise prime.

A.2.3 Halton sequence

A Hammersley sequence without the first element is called Halton sequence [17]

$$\mathbf{x}^i = (\phi_{b_1}(i), \ldots, \phi_{b_n}(i)), \quad i = 0, 1, \ldots . \tag{5}$$

The advantage over the Hammersley sequence is that a Halton sequence does not require the knowledge of N, the total length of the sequence.

A.2.4 Sobol' sequence

We construct the Sobol' sequence following [4] and [14]. Let us first consider 1-dimensional Sobol' sequence, for which we need to create a set of *direction numbers* v^i in base 2. To create the direction numbers v^i, we use coefficients of a primitive polynomial q in the field of binary numbers (compare to Eq. 1 in Sect. 7)

$$q(z) = z^p + a_1 z^{p-1} + \cdots + a_{p-1} z + a_p. \tag{6}$$

Then, employing the bitwise binary exclusive-or operator $\oplus$ we derive the direction numbers from the formula

$$v^i = a_1 v^{i-1} \oplus a_2 v^{i-2} \oplus \cdots \oplus a_p v^{i-p} \oplus \frac{v^{i-p}}{2^p}, \quad i > p.$$

The *ith* recurrence in 1-dimensional Sobol' sequence is now formed as

$$x^i = b_1 v^1 \oplus b_2 v^2 \oplus b_3 v^3 \oplus \cdots,$$

where $\cdots b_3 b_2 b_1$ is the binary representation of i (for example, if $i = 1011$, then $b_4 = 1$, $b_3 = 0$, $b_2 = 1$ and $b_1 = 1$). To generate an n-dimensional sequence, it is sufficient to choose n different primitive polynomials and calculate n different sets of direction numbers. For more details, see [4].

A.2.5 Faure sequence

The following example of Faure sequence is mainly from [43]. Let us define a function g

$$g(\phi_b(k)) = \frac{c_0}{b} + \cdots + \frac{c_{j-1}}{b^j},$$

where b is the base, $\phi_b(k) = (0.d_1 \ldots d_j)_b$ is an element of the Van der Corput sequence and

$$c_m = \left(\sum_{n=m}^{j} \frac{h!}{m(h-m)!} d_h \right) \bmod b.$$

Now, an n-dimensional (t,s)-sequence is defined as

$$\mathbf{x}^i = (\phi_b(i), g(\phi_b(i)), \ldots, g^{n-1}(\phi_b(i))),$$

and a Faure sequence is obtained when b is the smallest prime larger or equal to the dimension n [36].

References

1. Acworth, P., Broadie, M., Glasserman, P.: A comparison of some Monte Carlo and quasi Monte Carlo techniques for option pricing. In: Niederreiter, H., Hellekalek, P., Larcher, G., Zinterhof, P. (eds). Monte Carlo and Quasi-Monte Carlo Methods 1996, number 127 in Lecture notes in statistics, pp. 1–18. Springer-Verlag, New York (1998)
2. Ali, M.M., Storey, C.: Modified controlled random search algorithms. Int. J. Comput. Mathemat. **53**, 229–235 (1994)
3. Ali, M.M., Storey, C.: Topographical multilevel single linkage. J. Global Optimizat. **5**(4), 349–358 (1994)
4. Bratley, P., Fox, B.L.: Algorithm 659: implementing Sobol's quasirandom sequence generator. ACM Trans. Mathemat. Software **14**(1), 88–100 (1988)
5. Bratley, P., Fox, B.L., Niederreiter, H.: Implementation and tests of low-discrepancy sequences. ACM Trans. Model. Comput. Simulat. **2**(3), 195–213 (1992)
6. Couture, R., L'Ecuyer, P.: Distribution properties of multiply-with-carry random number generators. Mathemat. Computat. **66**(218), 591–607 (1997)
7. Dekkers, A., Aarts, E.: Global optimization and simulated annealing. Mathemat. Program. **50**(3), 367–393 (1991)
8. DIEHARD: a battery of tests for random number generators developed by George Marsaglia. http://stat.fsu.edu/~geo/diehard.html, September (2004).
9. Diggle, P.J.: Statistical Analysis of Spatial Point Patterns. Academic Press, London (1983)

10. Dykes, S., Rosen, B.: Parallel very fast simulated reannealing by temperature block partitioning. In Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics, vol. 2, pp. 1914–1919 (1994)
11. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evol. Computat. **3**(2), 124–141 (1999)
12. Floudas, C.A., Pardalos, P.M. (eds.): Recent Advances in Global Optimization. Princeton University Press, Princeton, NJ (1992)
13. Genetic and evolutionary algorithm toolbox for use with Matlab. http://www.geatbx.com/, June (2004)
14. Gentle, J.E.: Random Number Generation and Monte Carlo Methods. Springer-Verlag, New York (1998)
15. Glover, F., Kochenberger, G.A. (eds.): Handbook of Metaheuristics. Kluwer Academic Publishers, Boston (2003)
16. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, New York (1989)
17. Halton, J.H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimentional integrals. Numerische Mathematik, **2**, 84–90 (1960)
18. Horst, R., Pardalos, P.M. (eds.): Handbook of Global Optimization. Kluwer Academic Publishers, Dordrecht (1995)
19. Kingsley, M.C.S., Kanneworff, P., Carlsson, D.M.: Buffered random sampling: a sequential inhibited spatial point process applied to sampling in a trawl survey for northern shrimp pandalus borealis in west Greenland waters. ICES J. Mar. Sci. **61**(1), 12–24 (2004)
20. Larcher, G.: Digital point sets: analysis and applications. In: Hellekalek, P., Larcher, G. (eds.) Random and Quasi Random Point Sets, number 138 in Lecture Notes in Statistics, pp. 167–222. Springer-Verlag, New York (1998)
21. Lei, G.: Adaptive random search in quasi-Monte Carlo methods for global optimization. Comput. Mathemat. Appl. **43**(6–7), 747–754 (2002)
22. Lemieux, C., L'Ecuyer, P.: On selection criteria for lattice rules and other quasi-Monte Carlo point sets. Mathemat. Comput. Simulat. **55**(1–3), 139–148 (2001)
23. Maaranen, H.: On Global Optimization with Aspects to Method Comparison and Hybridization. Licentiate thesis, Department of Mathematical Information Technology, University of Jyväskylä (2002)
24. Maaranen, H., Miettinen, K., Mäkelä, M.M.: Quasi-random initial population for genetic algorithms. Comput. Mathemat. Appl. **47**(12), 1885–1895 (2004)
25. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin (1994)
26. Michalewicz, Z., Logan, T.D., Swaminathan, S.: Evolutionary operators for continuous convex parameter spaces. In: Sebald, A.V., Fogel, L.J. (eds.) Proceedings of the 3rd Annual Conference on Evolutionary Programming, pp. 84–97. World Scientific Publishing, River Edge, NJ (1994)
27. Morokoff, W.J., Caflisch, R.E.: Quasi-random sequences and their discrepancies. SIAM J. Sci. Comput. **15**(6), 1251–1279 (1994)
28. Neter, J., Wasserman, W., Whitmore, G.A.: Applied Statistics. Allyn and Bacon, Inc., Boston, third edition (1988)
29. Niederreiter, H.: Quasi-Monte Carlo methods for global optimization. In: Grossmann, W., Pflug, G., Vincze, I., Wertz, W. (eds.) Proceedings of the 4th Pannonian Symposium on Mathematical Statistics, pp. 251–267 (1983)
30. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. SIAM, Philadelphia (1992)
31. Nix, A.E., Vose, M.D.: Modeling genetic algorithms with Markov chains. Ann. Mathemat. Arti. Intell. **5**(1), 79–88 (1992)
32. Pardalos, P.M., Romeijn, H.E. (eds.): Handbook of Global Optimization, vol. 2. Kluwer Academic Publishers, Boston (2002)
33. Press, W.H., Teukolsky, S.A.: Quasi- (that is, sub-) random numbers. Comput. Phys. **3**(6), 76–79 (1989)
34. Ripley, B.D.: Spatial Statistics. John Wiley & Sons, New York (1981)
35. Snyder, W.C.: Accuracy estimation for quasi-Monte Carlo simulations. Mathemat Comput. Simulat. **54**(1–3), 131–143 (2000)
36. Sobol', I.M.: On quasi-Monte Carlo integrations. Mathemat. Comput. Simulat. **47**(2–5), 103–112 (1998)
37. Sobol', I.M., Bakin, S.G.: On the crude multidimensional search. J. Computat. Appl. Mathemat. **56**(3), 283–293 (1994)

38. Sobol, I.M., Tutunnikov, A.V.: A varience reducing multiplier for Monte Carlo integrations. Mathemat. Comput. Model. **23**(8–9), 87–96 (1996)
39. Tang, H.-C.: Using an adaptive genetic algorithm with reversals to find good second-order multiple recursive random number generators. Mathemat. Methods Operat. Res. **57**(1), 41–48 (2003)
40. Test problems for global optimization. http://www.imm.dtu.dk/~km/GlobOpt/testex/, June (2004)
41. Test problems in R^2. http://iridia.ulb.ac.be/~aroli/ICEO/Functions/Functions.html, June (2004)
42. Trafalis, B., Kasap, S.: A novel metaheuristics approach for continuous global optimization. J. Global Optimizat. **23**(2), 171–190 (2002)
43. Tuffin, B.: On the use of low discrepancy sequences in Monte Carlo methods. Monte Carlo Methods Appl. **2**(4), 295–320 (1996)
44. Weisstein, E.W.: "circle packing". From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/CirclePacking.html, September (2004)
45. Wikramaratna, R.S.: ACORN—a new method for generating sequences of uniformly distributed pseudo-random numbers. J. Computat. Phys. **83**(1), 16–31 (1989)
46. Wright, A.H., Zhao, Y.: Markov chain models of genetic algorithms. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 734–741. Morgan Kauffman, SanMateo, California (1999)