Universiteit Leiden
The Netherlands

Opleiding Informatica

Effect of initial sampling
on short-term behaviour of Differential Evolution

Petter Reijalt

Supervisors:
Dr. A.V. Kononova & D. Vermetten

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

14/06/2023

Abstract

This thesis aims to study the effects of various sampling methods on the first 10 generations of the Differential Evolution algorithm. Existing research on the effects of sampling methods on other evolutionary algorithms shows significant improvements when low-discrepancy sequences are used over a default uniform sampling distribution. Our study used a modular approach to isolate the sampling method used, then statistical analysis was conducted on the best solutions found in each generation. We found no significant improvements after 10 generations of the algorithm over a default uniform sampling distribution.

Contents

1 Introduction

Optimisation is the mathematical study of maximising or minimizing a certain function given certain constraints. Black box optimisation could be used to find this maximum. In this case it is assumed that no information is known a priori about the characteristics of the function. A black box gives a certain output depending on the input, but the inner workings are unknown. A computer simulation is a well-known example of a black box [1].

Traditional optimisation algorithms have been used to solve optimisation problems with limited success. Successful however in exploring the neighbourhood of a certain solution, they are not well suited for searching for global optima. Often they would get stuck in local optima [2]. Heuristics have been used to solve optimisation problems within a certain time span. They do not guarantee an optimal solution, solutions are so-called near-optimal. Metaheuristic algorithms expand upon these heuristic algorithms by learning from previous iterations and applying that in the next. Unlike heuristic algorithms, which come down to a trial-and-error method. Metaheuristic algorithms not only use stochastic elements but also deterministic elements [3].

Evolutionary algorithms form a subset of the metaheuristic algorithms, they do not require any upfront knowledge of the problem. Evolutionary algorithms are a family of population-based optimization algorithms inspired by Darwinian evolution. By utilizing the concepts of mutation, crossover and selection an evolutionary pressure is created which leads to improvements in fitness over time. It tries to iteratively improve the fitness of the population as a whole. [4].

One of these evolutionary optimisation algorithms is Differential Evolution. The algorithm starts with an initial population of candidate solutions. In each generation, new offspring are created by means of mutation and crossover. Using selection the new generation is created by choosing the solution with the highest fitness. The mutation step relies on the difference between individuals, i.e. differences in distance and direction [5].

In order to create an initial population, an initialisation method is used. Generally, a uniform probability distribution is used to sample an initial population. Achieved by using pseudo-random number generators. A uniform probability distribution, however, does not cover the domain equally [6]. This can lead to clusters of nearby sampling points. Quasi-random sequences can be used for a better spread of the initial sampling points. Designed to put as much distance between individual points[7]. The effects of initial sampling are often still seen after several iterations for metaheuristic optimisation algorithms and can often influence the final outcome [7]. The effects of initial sampling become even more evident when used with bigger, multimodal problems [8].

1.1 Research question

Although there is evidence to show that quasi-random sequences could improve the performance of other evolutionary algorithms such as genetic algorithms [7]. Such research has not been done for Differential Evolution. This thesis aims to investigate the effects of quasi-random sequences,

probability distributions and other initialisation schemes, such as Latin Hypercube Sampling, on the early dynamics of the Differential Evolution algorithm. This leads to the following research question:

RQ: For a particular example of a heuristic population-based optimisation algorithm called Differential Evolution, what are the effect of different methods of initial sampling on the early dynamics within the population?

1.2 Related work

Li et al. researched the effect of initialisation schemes on five different optimisation algorithms, including Differential Evolution. The only initialisation schemes used, however, were Latin Hypercube Sampling and different probability distributions. It concluded that Differential Evolution is not significantly impacted by the use of different initialisation schemes and heavily relies on the number of iterations used and less on the population size [9]. More initialisation schemes have been researched for other evolutionary algorithms, such as genetic algorithms. Maaranen et al. researched different quasi-random algorithms on genetic algorithms. It concluded that initialisation schemes are of great importance and quasi-random sequences can successfully be used together with problems with higher dimensionalities [7]. This has also been researched for other population-based metaheuristic algorithms by Agushaka & Ezugwu as part of a meta-study. It looked into different sampling methods with regards to the bat algorithm, Grey Wolf Optimizer and the butterfly optimisation algorithm. It researched Monte Carlo Methods, Quasirandom Methods and Probability Distributions among others. It concluded that some algorithms were sensitive to the initialisation scheme as others were not [10].

1.3 Thesis overview

The rest of the paper is structured as follows: Section 2 includes an explanation of the Differential Evolution algorithm; Section 3 outlines the sampling methods used; Section 4 describes the methodology used; Section 5 describes the experiments and their outcome; Section 6 concludes and the appendix can be found in section 8.

This thesis was written for The Leiden Institute of Advanced Computer Science (LIACS) at Leiden University in the spring of 2023 as a bachelor thesis, it was supervised by Dr. A.V. Kononova & D.L. Vermetten.

2 Differential Evolution

Differential evolution is a so-called evolutionary algorithm that tries to solve optimisation problems, which are continuous in nature. The algorithm starts off with an initial population of NP vectors,

which covers the search space, following a certain sampling method. The algorithm runs for a predetermined amount of generations and then halts. In every generation every vector has to serve once as a target vector, resulting in NP competitions. There are many different versions of Differential Evolution. The performance of the Differential Evolution on a certain problem heavily relies on the mutation and crossover scheme used and the control parameters [11]. Beneath the standard, most basic, Differential Evolution variant DE/rand/1/bin will be outlined [12].

Mutation
For every target, a mutant vector is generated. According to the following formula:

$$v_{i,G+1} = x_{r_1,G} + F * (x_{r_2,G} - x_{r_3,G})$$

where x represent different pre-existing vectors and $F > 0$. The scale factor F determines the size of difference vector $x_{r_2,G} - x_{r_3,G}$. r_1, r_2 and r_3 represent the index of a vector from a generation G. Also r_1, r_2 and r_3 should be different then i. Figure 1 shows an example of the possible construction of a mutant vector [12].

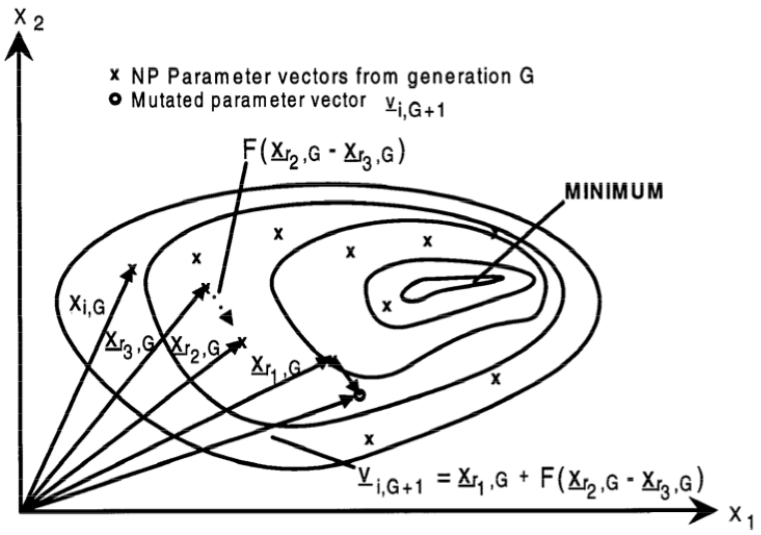


Figure 1: The construction of a mutant vector [12]

Crossover
Then crossover takes place, in order to increase diversity, in which this mutated vector takes on a certain amount of parameters from another vector, the target vector:

$$v_{i,G+1} = (v_{1i,G+1}, v_{2i,G+1}, ..., v_{Di,G+1})$$

is created by:

$$v_{ij,G+1} = \begin{cases} v_{ji,G+1} & \text{if } \mathcal{U}(0,1) \leq \text{CR or j} = \text{rnbr(i)} \\ v_{ji,G} & \text{else} \end{cases}$$
$$j = 1, 2, ..., D.$$

In this certain example the latter part of the formula: *rnbr(i)* is random chosen index $\in 1, 2, ..., D$. It ensures that at least one parameter from the mutant vector is used in the trial vector. CR, the crossover rate determines the likelihood of the trial vector taking on a parameter from the mutant vector. *Randb* is a random number generator, producing a number between 0 and 1. [12]. This process is illustrated in Figure 2.

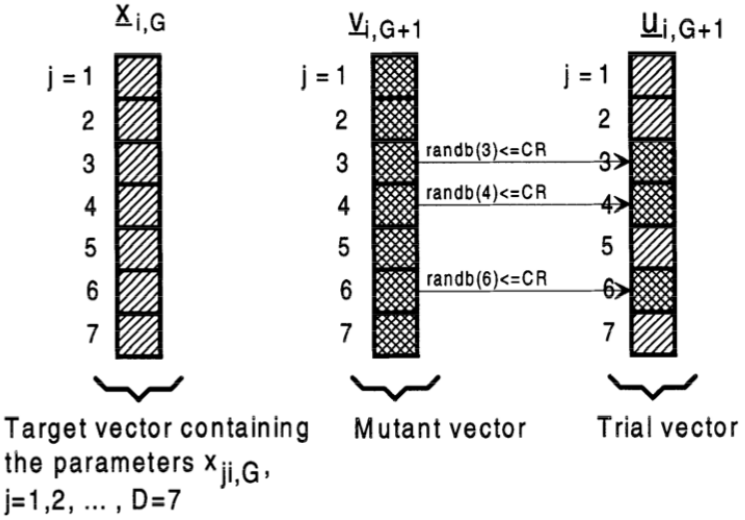


Figure 2: The construction of a trial vector [12]

Selection
This trial vector and target vector are then compared against each other. The best vector will be used in the new generation, the other one will be discarded [12].

DE/x/y/z
There are a lot of different methods that can be used concerning the mutation strategy and crossover method. In order to differentiate between the different versions of Differential Evolution the notation *DE/x/y/z* is used. In which x determines which vector will be mutated, y determines the number of difference vectors used on the vector determined by x, z determines the scheme used in the selection step [12].

In this notation, the initialisation scheme in question is not included. The aim of this paper is to research the effects of initial sampling on the population, with regards to early generations of the algorithm. In order to isolate the initial sampling from the other components a modular approach is needed.

3 Sampling methods

The following section outlines the main categories of sampling methods used in population-based metaheuristics.

Probability distributions
Figures 3 & 4 show examples of a uniform distribution and a Gaussian distribution in a 2D space (for $n = 100$), where each parameter is normalised to a real number between 0 and 1.

1. Uniform distribution

Uniform probability distributions are most normally used in regard to population-based algorithms. Pseudo-random number generators are then used to emulate random numbers [6]. The discrepancy in the sequence created by the uniform distribution will not optimally benefit the convergence of algorithms, as concluded by Niederreiter and Harald [13]. The probability density function can be described as follows where a and b describe the minimum and maximum values respectively:

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & otherwise \end{cases}$$

The average value is equal to $\frac{(a+b)}{2}$ and the variance can be described as $\frac{(b-a)^2}{2}$ [14].

2. Gaussian distribution

The density of the sampling points of the Gaussian distribution follows a bell shape. Meaning that sampling points are more likely to be in the centre and the chance of them being near the edges becomes increasingly smaller. It is possible, though unlikely, that the sampling point lies out of bounds.

The probability density function can be formulated as follows:

$$p(x) = \frac{1}{\sqrt{s\pi}\sigma} \exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$$

where μ represents the mean, and σ represents the standard deviation [14].

Latin Hypercube Sampling
Latin Hypercube Sampling is a stratified Monte Carlo sampling method. The search spaced is divided into equal-sized *stratas* for every dimension. For each *strata* one sampling point is chosen. Using a pseudorandom number generator a value within the bounds of the *strata* is generated [15].

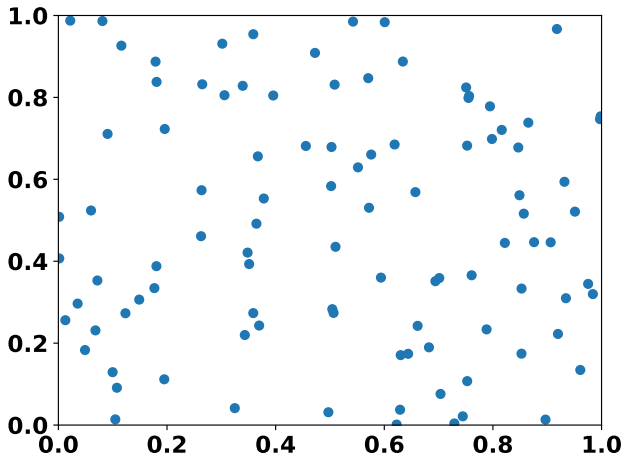


Figure 3: Uniform sampling

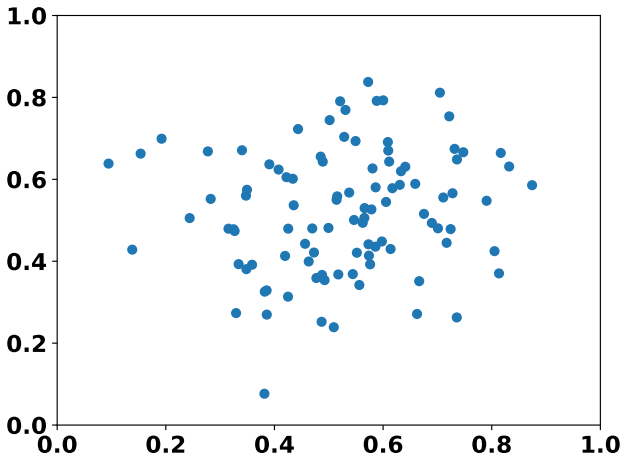


Figure 4: Gaussian sampling

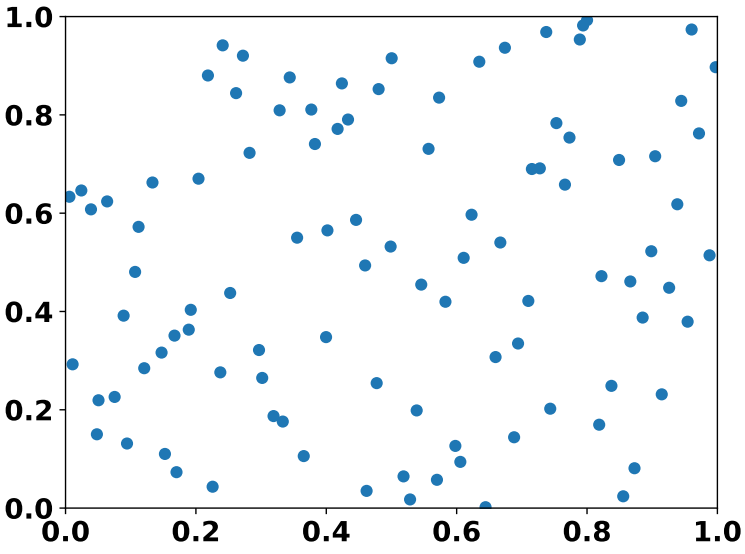


Figure 5: Latin Hypercube Sampling algorithm

In Figure 5 a possible sampling is shown for $n = 100$.

Quasi-random numbers

Quasi-random numbers are sequences that can more uniformly cover a search space. Well-known sequences include Faure, Sobol and Halton [10]. In this paper, we include the latter two. In Figures 7 and 8, the Sobol and Halton sequences are shown for n=1000 and d=2.

1. Halton sequence

When one wants to distribute sampling points as uniformly as possible in a 1-dimensional space in $[0, 1]$, one might think of the following sequence:

$$0, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}...$$

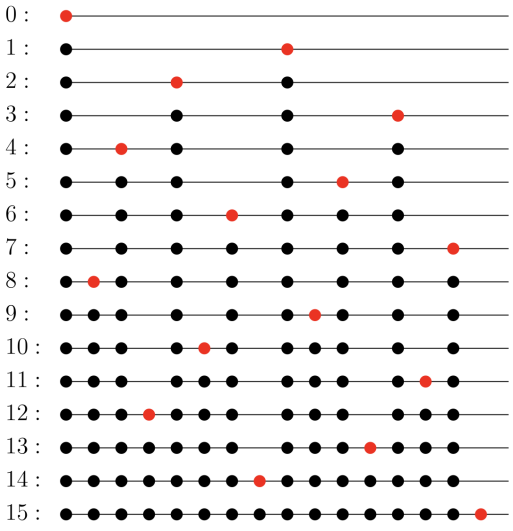


Figure 6: Visualisation of the binary van der Corput sequence [16]

This is the so-called van der Corput sequence in base 2, which van der Corput made into a general construction principle [16]. This sequence is achieved by reversing the digits for some non-negative integer in some base [17]. Every nonnegative integer n can be represented using the binary expansion:

$$n = n_0 + n_1 2 + n_2 2^2 + n_3 2^3 + ...$$

which is a finite sequence. The nth element of the van der Corput sequence is given by:

$$y_n = \frac{n_0}{2} + \frac{n_1}{2^2} + \frac{n_2}{2^3}...$$

in base 2. The Halton sequence uses a multi-dimensional form of the van der Corput sequence using multiple bases without duplicates. [16]. Generally, you choose s bases, where s is the amount of dimensions so that they represent the first s prime numbers [17]. After that, you concatenate the sequences to compute every sampling point for a s-dimensional Halton sequence such as:

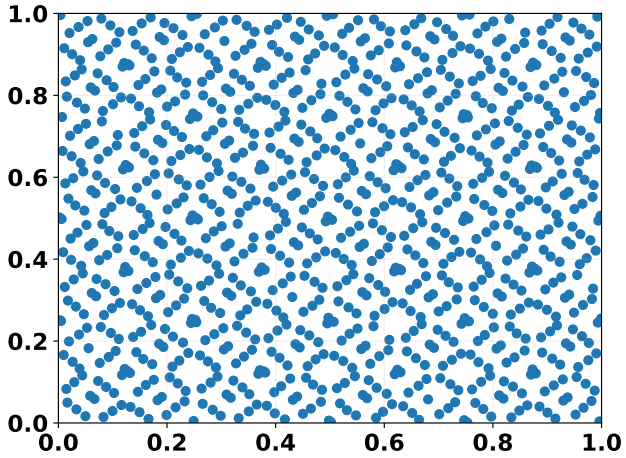


Figure 7: Sobol sampling ($n = 1000$)

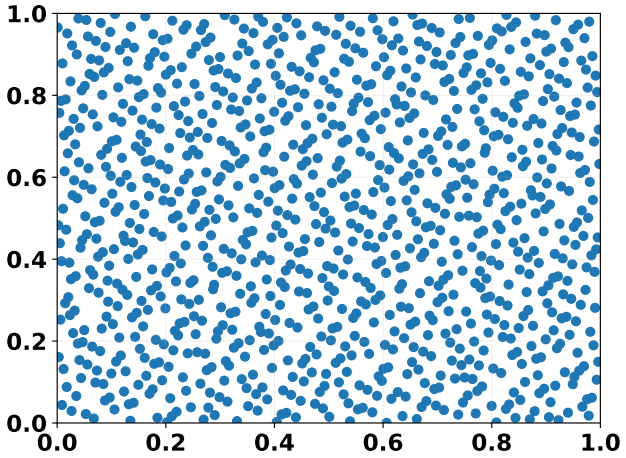


Figure 8: Halton sampling ($n = 1000$)

$$y_n = (Y_{b_1}(n), Y_{b_2}(n), ..., Y_{b_s}(n))$$

where $b_1, b_2, ..., b_s$ represent the prime numbers used as bases [16].

2. Sobol sequence

Here we will outline the general steps that need be taken in order to generate a Sobol sequence. Theoretical details can be found in [16]. Suppose we want to generate sampling points for a 1-dimensional space. This would result in a sequence: $x^1, x^2, ...$ Therefore we would need to create direction numbers $(v_1, v_2, ...)$, which are binary fractions. These can be represented as:

$$v_i = \frac{m_i}{2^i}$$

where m_i is an odd integer. These direction numbers can be obtained by choosing an arbitrary polynomial with coefficients that can be either 0 or 1. For example:

$$P \equiv x^d + a_1 x^{d-1} + ... + a_{d-1} x + 1$$

The coefficients can then be used in order to define a recurrence relation for m_i, in order to get v_i, resulting in:

$$m_i = 2_{a_1} m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus ... \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus m_{i-d}$$

The final sequence can then be generated using:

$$x^n = b_1 v_1 \oplus b_2 v_2 + ...$$

where $...b_3, b_2, b_1$ represent the binary representation of n [18].

Grid sampling

Grid sampling is a sampling method in which all sample points are equidistant from each other forming a grid. Shown in Figure 9

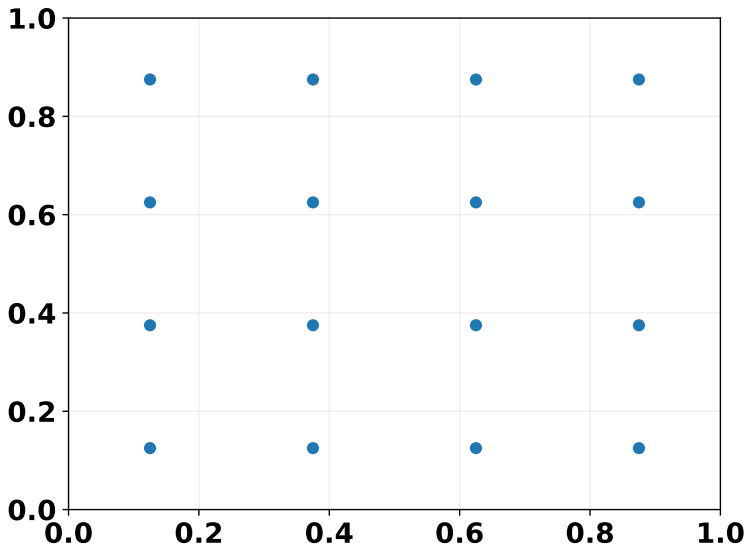


Figure 9: Visualisation of grid sampling for n=16

4 Methodology

4.1 Modular DE

The Differential Evolution algorithm was implemented using Modular DE. This modular framework allows for the investigation of the interaction of different components. This allows for a study of the effects of initial sampling when all else is held constant [19].

4.1.1 Sampling methods

The framework consists of the base samplers: Uniform, Gaussian, Halton and Sobol. It also allows for the use of oppositional initialisation [19]. We expand upon this by adding another quasi-random sequence: Hammersley and added Latin Hypercube sampling and Grid sampling.

Modular DE	**Added**
Gaussian distribution (gaussian)	Grid sampling (grid)
Uniform distribution (uniform)	Latin Hypercube sampling (latin)
Sobol sequence (sobol)	Hammersley sequence (hammersley)
Halton sequence (halton)	

Table 1: Implemented sampling methods

4.1.2 DE version

The version of Differential Evolution that is used during this research is equal to $DE/current-to-pbest/1$:

$$v_i = x_i + F(x_{pbest} - x_i) + F(x_{r_1} - x_{r_2})$$

where x_{r_1} and x_{r_2} represent 2 different vectors which are not x_{pbest} or x_i. *pbest* represents a random vector chosen out of the p best vectors. In this case, p is equal to the 20% best vectors. Other specifications can be found in Table 2.

Operation	**Module Name**	**Choice**
Initialisation	Base sampler	*variable*
Initialisation	Oppositional initialisation	false
Mutation	Base vector	target
Mutation	Reference vector	pbest
Mutation	Number of differences	1
Mutation	Use weighted F	false
Mutation	Use archive	false
Crossover	Crossover method	bin
Bound correction	Bound correction	saturate
Adaptation	F adaptation method	none
Adaptation	CR adaptation method	none
Adaptation	Population size reduction	false
Adaptation	Use JSO caps for F and CR	false
Parameter	Population size	10 * D
Parameter	Scale factor	0.5
Parameter	Crossover rate	0.5

Table 2: Settings used in the Modular DE framework

4.2 IOHprofiler

For this research, we used the benchmarking platform IOHProfiler. This could be used for comparing different iterative optimisation heuristics [20]. In this case, we benchmarked different versions of the Differential Evolution algorithm that were created with the Modular DE framework.

4.2.1 IOHexperimenter

IOHexperimenter is the experimental part of IOHprofiler. It allows for the systematic logging of the behaviour of optimisation algorithms whilst solving an optimisation problem [21]. The 2 problem suites used in this research are as follows:

4.2.1.1 BBOB

The BBOB test suite contains 24 single-objective, noiseless functions. All functions can be modelled into multiple dimensions. The 24 functions try to mimic situations that one would encounter in a real-life setting. The optimum lies somewhere in the domain: $[-5, 5]^D$ [22].

4.2.1.2 SBOX

The SBOX test suite aims to negate some of the downsides that come with using the BBOB test suite. For one it more uniformly distributes the optima across the domain. Some problems of the BBOB test suite do not cover the entire search domain. Secondly, it more strictly enforces box constraints, returning an invalid value when accessing a solution outside the search domain [23].

4.2.2 IOHanalyzer

The data generated by IOHexperimenter was then used in IOHanalyzer. IOHanalyzer visualizes and analyzes the data from optimisation heuristics. Offering both an analysis of the runtimes of fixed targets and the performance of fixed budgets. The latter is interesting for our research because we only investigate the early stages of the algorithm. The tools allow for the visualisation of different sampling methods with regard to the average performance over multiple repetitions and instances. It can also tell us, whether differences between sampling methods are significantly different [24].

4.2.2.1 R Programming Interface

The R programming interface, which is a part of IOHanalyzer, allowed us to extract the data and it to be used in R. Which allows for a more thorough analysis of the data [24].

4.2.3 Test specifications

Every sampling method was run 50 times (10 repetitions $\times$ 5 instances) in each dimension for every function. The dimensions used were $d \in \{2, 5, 20\}$. This was done for all the functions, 1 to 24, in both the BBOB test suite and the SBOX test suite.

4.3 Code

The code used for this project can be found here or go to the next URL: `https://github.com/Petter6/ModDE`

4.4 Data

All data can be found here or go to the next URL: `https://zenodo.org/record/8110708`

5 Results

In order to determine the differences between the sampling methods in the early stages, we looked at the performance of the sampling methods within the first 10 generations. We compared the

best solutions the Differential Evolution algorithm found given a certain sampling method, given a certain number of iterations. To subject the Differential Evolution to a great variety of problems we used different dimensions 2, 5, 20 and the BBOB test suite, which is the industry standard for testing optimisation algorithms [25]. To see if there was any difference at all, we looked at the average best solutions found across all functions of all sampling methods directly after initialisation. In Figure 10, every sampling method is compared against every other sampling method. The colour red means that across the 24 functions, the sampling method on the y-axis had a better average solution for most of the functions, compared to the sampling method on the x-axis. Blue means a worse average for most of the functions. This showed us that Gaussian sampling across all dimensions on average performs the worst directly after the initialisation and is followed by Grid sampling. In order to figure out, if this is the case for all dimensions used we computed graphs based on individual dimensions, this is shown in Figure 11. It shows us that Gaussian Sampling performs badly for 2-dimensional problems and 5-dimensional problems. Grid sampling performs worse than most other sampling methods for 5-dimensional and 20-dimensional problems. This could be the case because, in higher dimensions, it becomes increasingly harder for Grid sampling to fill the search space. For $d = 20$ equal to a population size of 200 in our case, a grid needs to be constructed containing $2^{20} = 1048576$ points (1^{20} does not contain enough points for our purposes). Of which only 200 will be used, making it susceptible to clustering, leaving a lot of points in the grid unused.

To see if Gaussian sampling and Grid sampling also perform worse in the short term, we looked at the performance after 10 iterations of the algorithm across all dimensions, which can be seen in Figure 12. This showed us that after 10 iterations the Grid sampling still has the worst average best solution for most functions when compared 1 to 1 with other sampling methods. This could be attributed to the fact that for Grid sampling it is harder to explore new areas of the search space. Because all points are equidistant, the difference between 2 vectors could lead to a sampling point that is on the grid and could even be already found. The Gaussian sampling actually slightly outperformed the other sampling algorithms. This is due to the fact that the Gaussian sampling severely outperforms the other algorithms in 20-dimensional problems, and to a lesser extent the 5-dimensional problems (Figure 13).

To know whether or not these differences were significant on a function-to-function basis, we compared the distribution of the best solutions of every sampling algorithm to the distribution of best solutions found by Uniform sampling. We computed the p-values for every function for every algorithm for every generation at a significance level of 0.01. The p-values were computed using the Kolmogorov-Smirnov test, which is used to test the goodness of fit for 2 distributions and is believed to be more powerful than the chi-square test [26]. Because the likelihood of the occurrence of a false positive increases as one makes more statistical inferences, we used a multiple comparison procedure. In order to correct for the occurrence of these false positives, the p-values were corrected using the Benjamini-Hochberg Procedure [27]. In Figure 14, a subset of problems is shown for 20-dimensional problems in which the best solution distribution of Grid sampling and Gaussian sampling significantly differ from the Uniform distribution. Particularly for these functions, the Gaussian sampling method performs better than Uniform sampling and Grid sampling performs worse. This holds true for each of the 10 iterations, see Figure 15, 16 and 18. When comparing all performance graphs and probability distributions this leads to the conclusion that Gaussian

sampling significantly outperforms Uniform sampling for 20-dimensional problems in 17 of the 24 problems. Uniform sampling significantly outperformed Grid sampling for 20 of the 24 problems.

This bias towards Gaussian sampling could be attributed to the fact that the domain in which the optimum can be found is not the same as the search space for most functions in the BBOB test suite. Although the search space is equal to $[-5, 5]^d$, the optimum can actually only be found in $[-4, 4]^d$ for most functions. Meaning that for increasingly higher-dimensional problems, the part of the search space that can actually contain the optimum becomes increasingly smaller relative to the search space. This helps algorithms that focus on the centre, e.g. algorithms that use Gaussian sampling as their initialisation method. To negate these effects we used the SBOX test suite. The SBOX test suite uses the same functions as the BBOB test suite, but the optima follow a uniform distribution across the domain $[-5, 5]^d$, for most functions [23].

We then conducted the same experiments as we did using the BBOB test suite. The sampling methods using probability distributions performed worse than the other sampling methods at initialisation. When compared 1 to 1 with other sampling methods, the average best solution was worse for more functions. The other sampling methods showed no big differences from one another. Only slightly outperforming the other algorithm by having a better average solution in a few functions (Figure 19). From 5-dimensional problems onwards the Grid sampling methods start to perform worse than the other methods. Similar to the BBOB test suite the Gaussian function also performs better than other sampling methods directly after initialisation when 20-dimensional problems are used (Figure 20).

Where the SBOX test suite results differ from the BBOB is after 10 iterations. Where the BBOB test suite showed that Gaussian sampling outperformed most sampling methods when d=20, this is no longer the case using the SBOX test suite (Figure 21). We then computed the same p-values as we did with the BBOB test suite. When comparing the p-values with the average best-found f(x)-values, we found that Gaussian sampling is only significantly better in 6 of the 24 problems as opposed to the 17 found by the BBOB test suite. Grid is still clearly the worst, significantly worse in 17 of the 24 problems when comparing the p-values with the average best-found f(x)-values. A subset of the functions found, where Grid performs significantly worse and Gaussian significantly better can be seen in (Figure 22). We did the same comparison for the lower dimensionalities, 2 and 5. The Halton and Hammersley sampling methods are significantly better than other methods directly after initialisation. This is also true to a lesser extent for the Sobol sampling method, for 2-dimensional problems. However, this lead is quickly lost after 1 or 2 generations. For 2-dimensional problems, the Hammersley sequence is significantly better than Uniform sampling in 16 of the 24 problems at initialisation. Examples of this can be seen in (Figure 23). For 20-dimensional problems, it is the other way around at initialisation, the Halton and Hammersley sampling methods perform significantly worse than others, but this deficit is also quickly made up for (Figure 24). The worse performance for higher-dimensional problems could be due to the higher correlation between two points when higher dimensionalities are used [28].

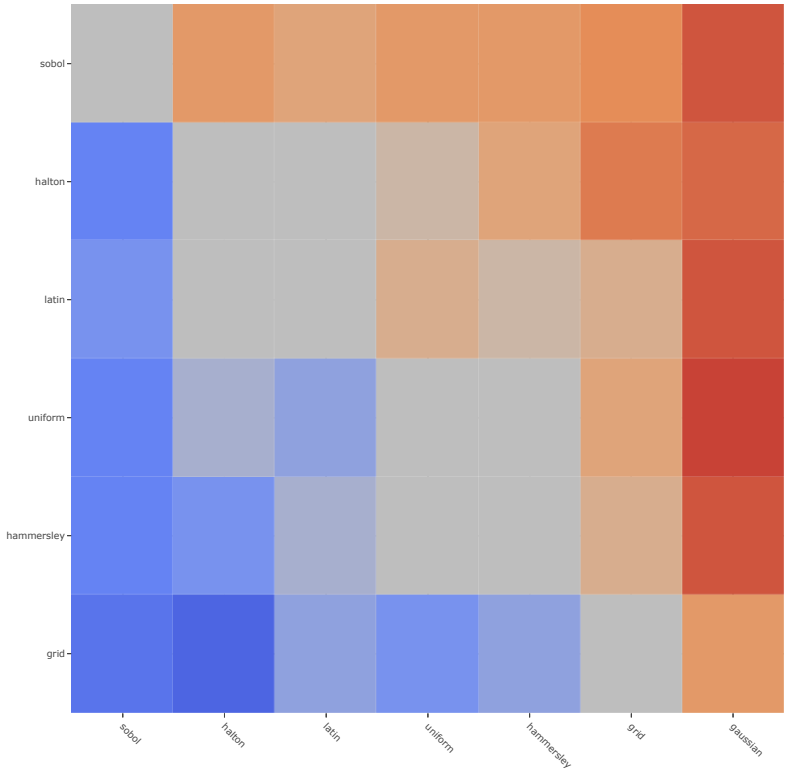
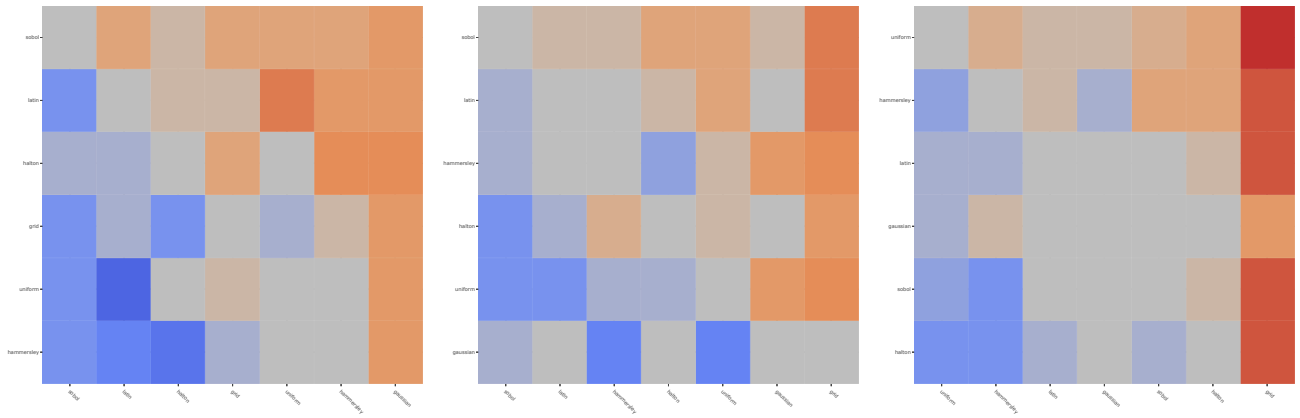


Figure 10: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs ($[1, 24] \times \{2, 5, 20\}$). Results after initialisation for the BBOB test suite.



(a) $[1, 24] \times \{2\}$ (b) $[1, 24] \times \{5\}$ (c) $[1, 24] \times \{20\}$

Figure 11: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs. Results after initialisation for the BBOB test suite.

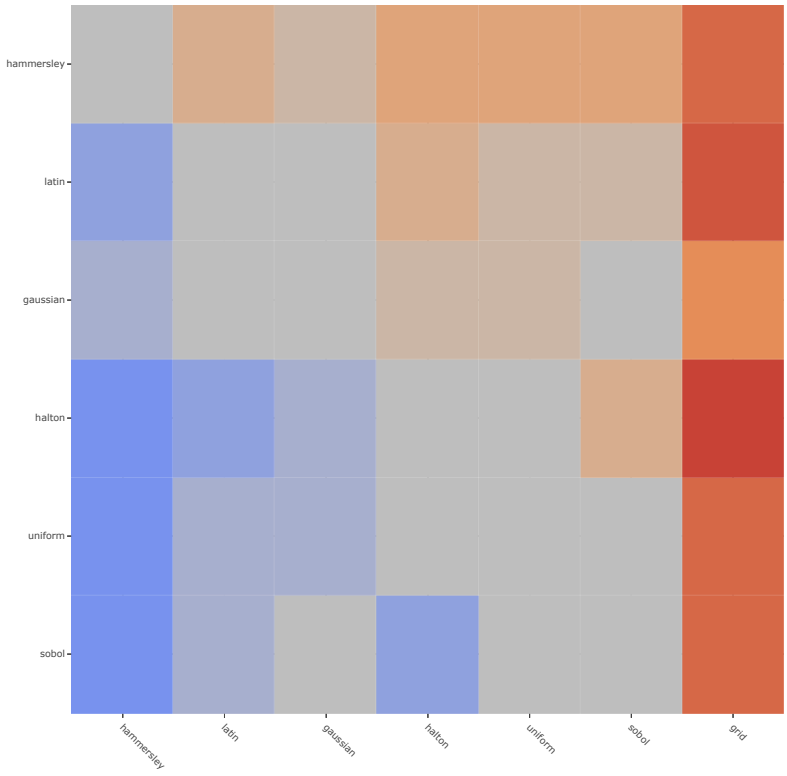
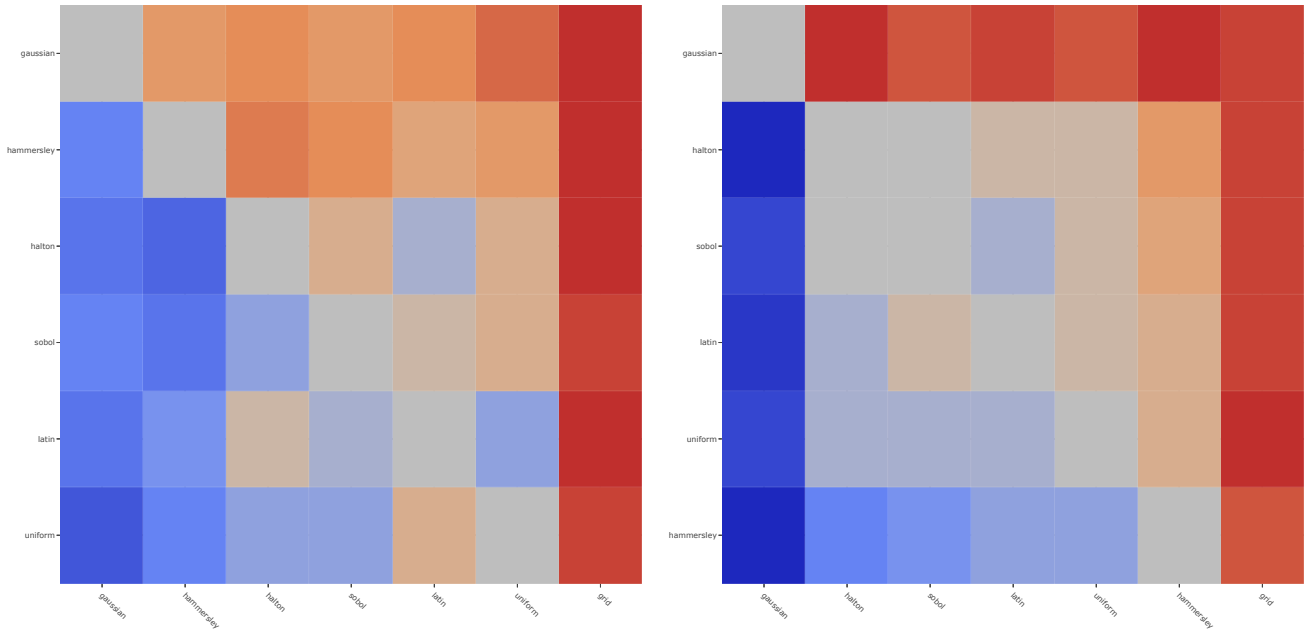
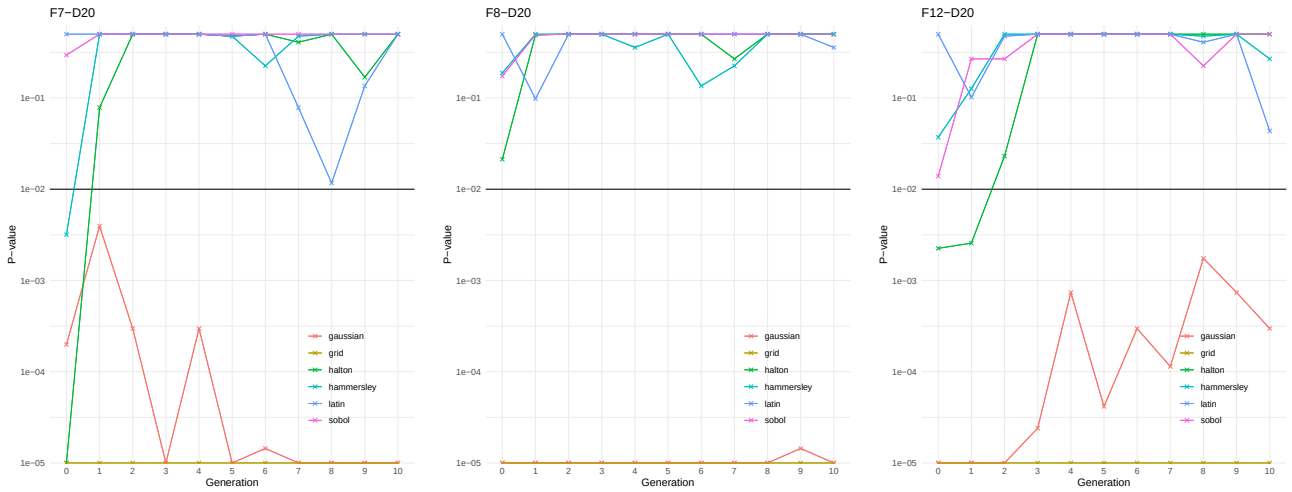


Figure 12: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs ($[1, 24] \times \{2, 5, 20\}$). Results after 10 iterations for the BBOB test suite.



(a) $[1, 24] \times \{5\}$ (b) $[1, 24] \times \{20\}$

Figure 13: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs. Results after 10 iterations for the BBOB test suite.



(a) Function 7, 20-dimensional (b) Function 8, 20-dimensional (c) Function 12, 20-dimensional

Figure 14: Goodness of fit test between the distribution of best solutions of every Sampling method versus the distribution of best solutions of the Uniform sampling method. P values computed with Kolmogorov-Smirnov test, for every generation, for every function, for every dimension, at a 0.01 significance level.

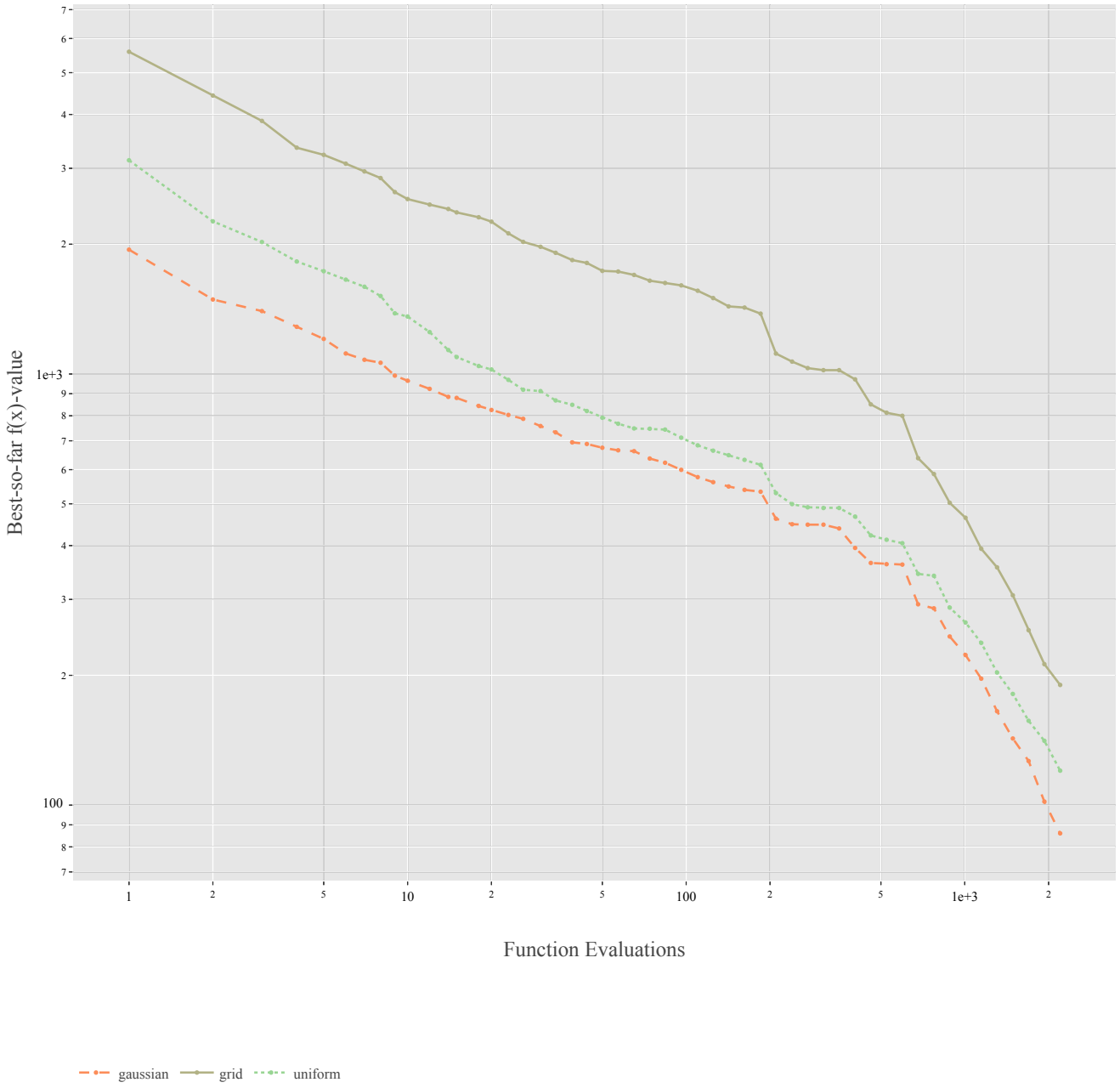


Figure 15: Average of the best-found f(x)-values over all iterations for every function evaluation. For the sampling methods: Grid, Gaussian, Uniform. Shown for Function 7 as a 20-dimensional problem from the BBOB test suite.

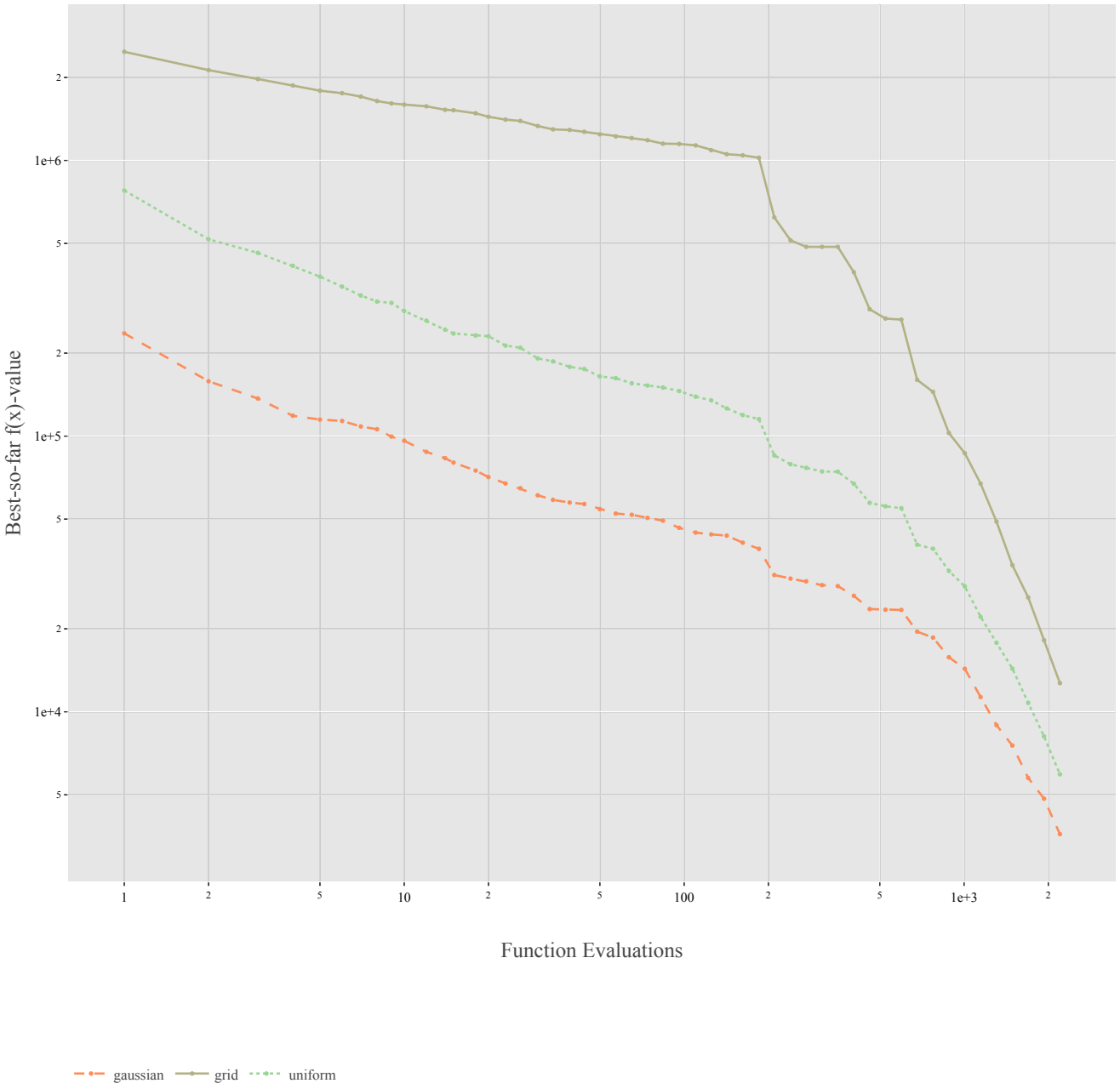


Figure 16: Average of the best-found f(x)-values over all iterations for every function evaluation. For the sampling methods: Grid, Gaussian, Uniform. Shown for Function 8 as a 20-dimensional problem from the BBOB test suite.

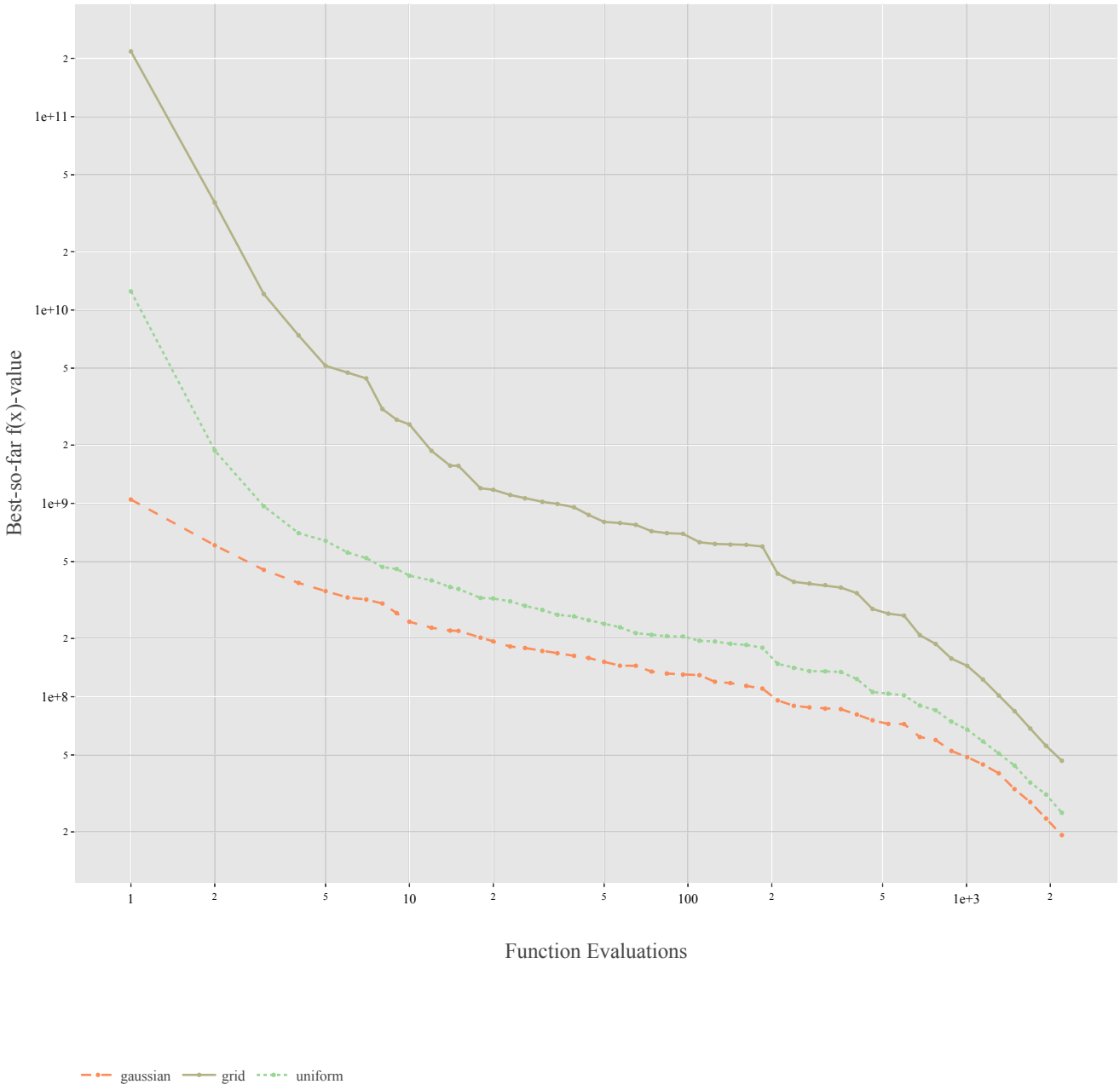


Figure 17: Function 12, 20-dimensional

Figure 18: Average of the best-found f(x)-values over all iterations for every function evaluation. For the sampling methods: Grid, Gaussian, Uniform. Shown for Function 12 as a 20-dimensional problem from the BBOB test suite.

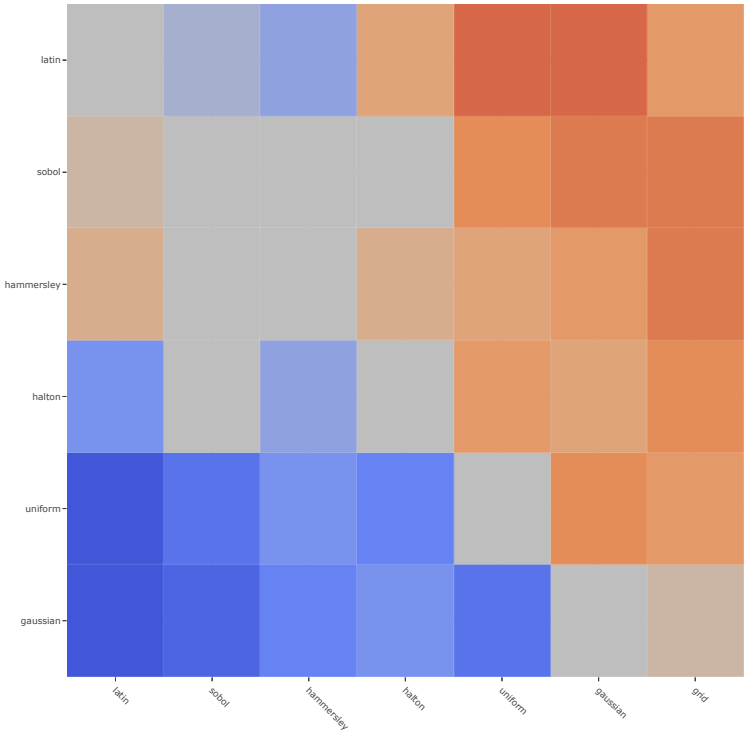
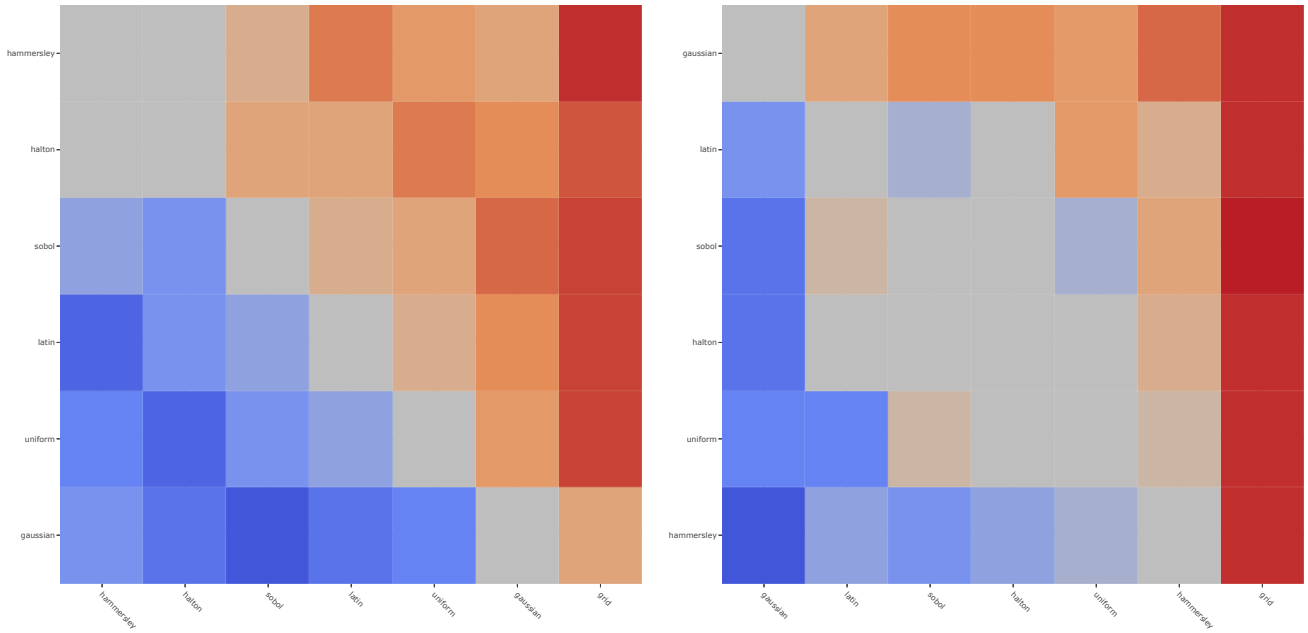


Figure 19: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs ($[1, 24] \times \{2, 5, 20\}$). Results after initialisation for the SBOX test suite.



(a) $[1, 24] \times \{5\}$ (b) $[1, 24] \times \{5\}$

Figure 20: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs. Results after initialisation for the SBOX test suite.

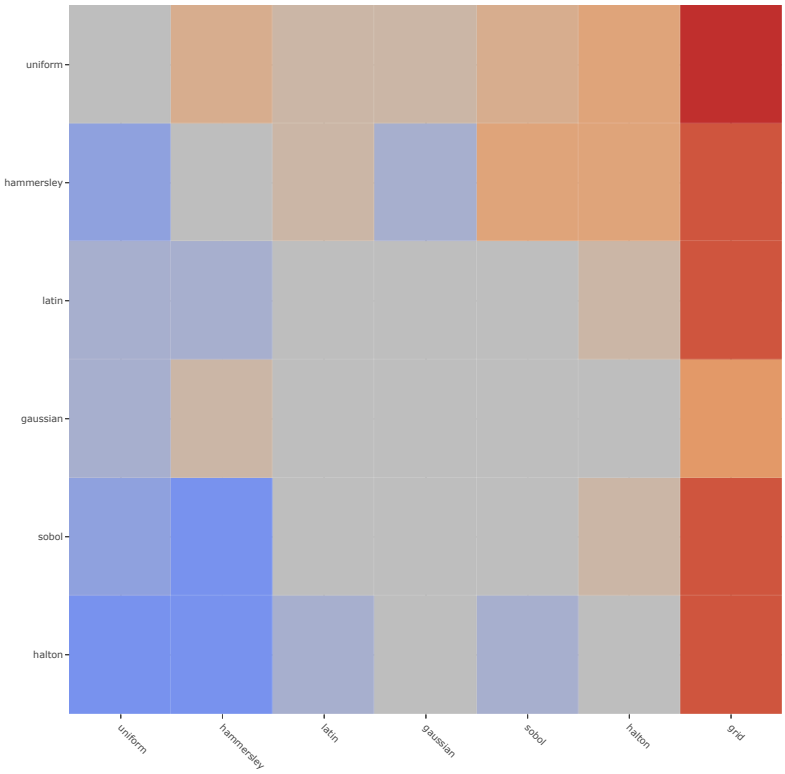
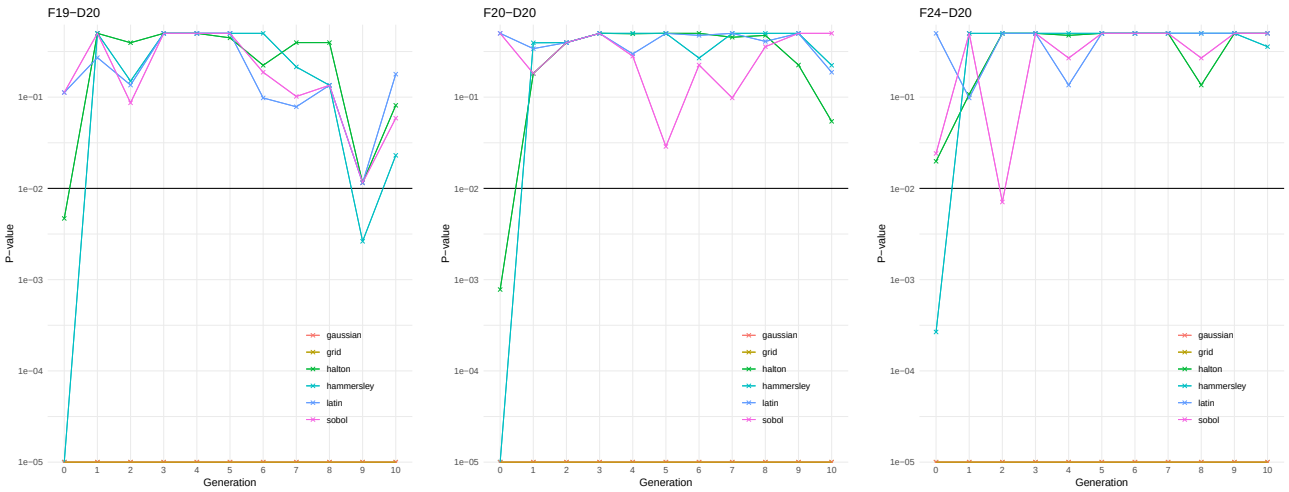
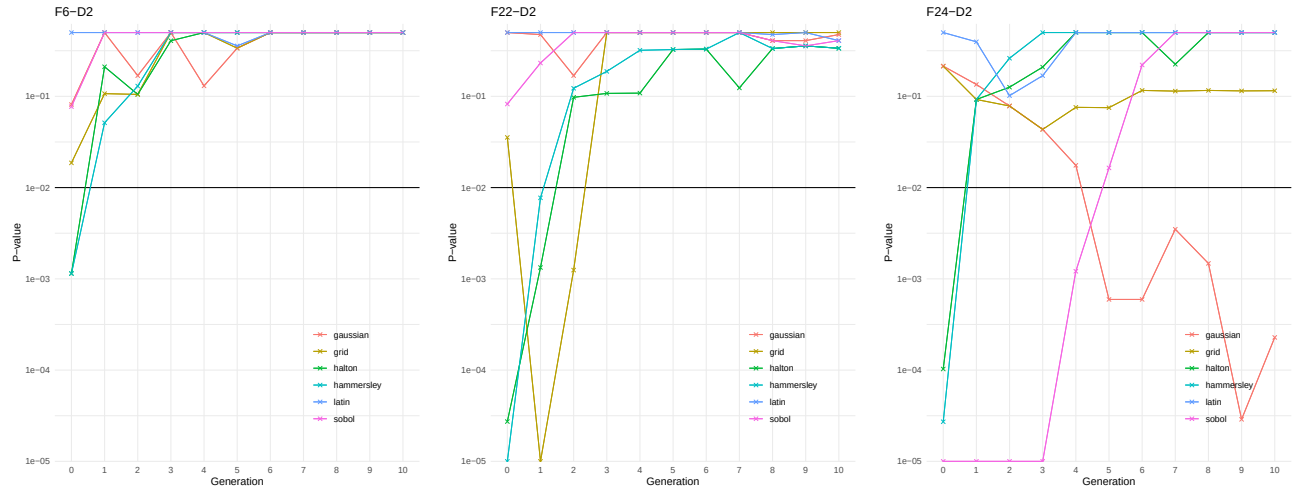


Figure 21: Algorithms compared 1 to 1, expressed as a fraction of times the average mean is better for all selected function-dimension-pairs ($[1, 24] \times \{20\}$). Results after 10 iterations for the SBOX test suite.



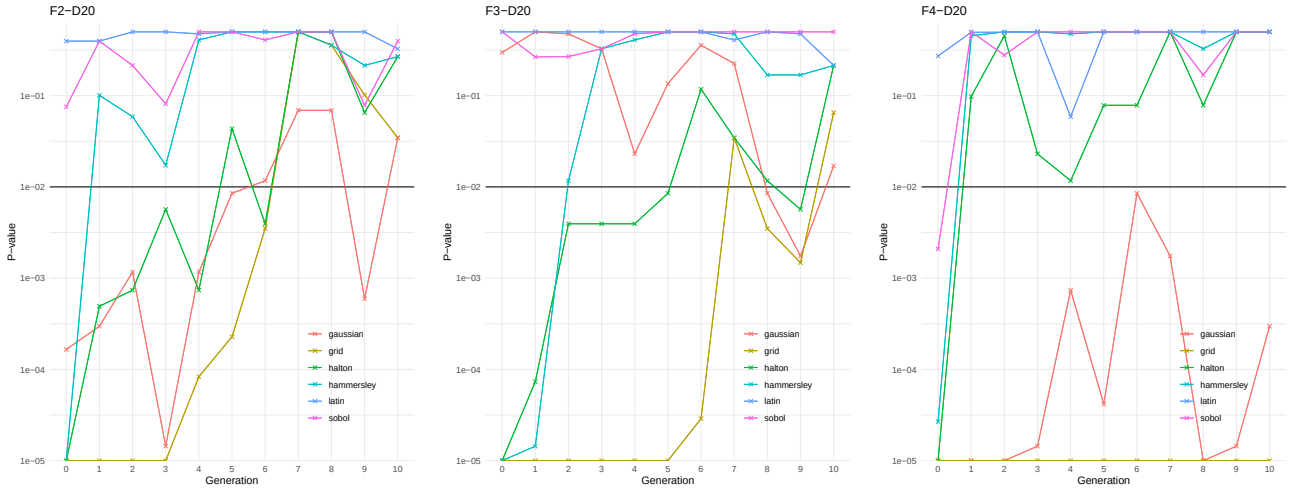
(a) Function 19, $d = 20$ (b) Function 20, $d = 20$ (c) Function 24, $d = 20$

Figure 22: Goodness of fit test between the distribution of best solutions of every Sampling method versus the distribution of best solutions of the Uniform sampling method. P values computed with Kolmogorov-Smirnov test, for every generation, for every function, for every dimension, at a 0.01 significance level.



(a) Function 6, $d = 2$ (b) Function 22, $d = 2$ (c) Function 24, $d = 2$

Figure 23: Goodness of fit test between the distribution of best solutions of every Sampling method versus the distribution of best solutions of the Uniform sampling method. P values computed with Kolmogorov-Smirnov test, for every generation, at a 0.01 significance level.



(a) Function 2, $d = 20$ (b) Function 3, $d = 20$ (c) Function 4, $d = 20$

Figure 24: Goodness of fit test between the distribution of best solutions of every Sampling method versus the distribution of best solutions of the Uniform sampling method. P values computed with Kolmogorov-Smirnov test, for every generation, at a 0.01 significance level.

6 Conclusion and Future work

The SBOX test suite showed us, that initial sampling matters, not every sampling method that tries to spread out points will work for Differential Evolution. This can be seen in the performance of the Grid sampling algorithm. A sampling algorithm that does not work well with the mechanics of Differential Evolution. Because of the use of difference vectors, a set-up that starts off with points equidistant from each other in a grid will have a hard time exploring new areas. The effects of this initial bad sampling are still seen after 10 iterations of the algorithm.

Gaussian was shown by the BBOB test suite to be significantly better than the other sampling methods for 5-dimensional and 20-dimensional problems. The BBOB test suite heavily favours the Gaussian sampling method. This could be attributed to the fact that the difference between the domain in which the optimum can be found and the domain of the search space increases as you increase the dimensionality. The optimums will be relatively closer to the centre as the dimensionality is increased, favouring the Gaussian algorithm, but not necessarily realistic pertaining to real-life settings [23]. This makes the BBOB test suite unsuited for evaluating sampling methods that use a centre-based approach. The SBOX suite showed that the Gaussian sampling method was as good as the other sampling methods.

The Halton and Hammersley sequences were significantly better than Uniform sampling for dimensionalities 2 and 5. They outperformed other sampling methods in the initialisation step, but this lead quickly diminished in later phases. For dimensionality 20 the Halton and Hammersley sampling methods were the worst after the initialisation step. This could be attributed to the high correlation between a set of points between 2 dimensions when a higher dimensionality is used [28]. This deficit was on the other hand quickly made up for.

The Sobol sampling method was shown to be slightly better than Uniform sampling for 2-dimensional problems at initialisation. There were, however, no significant differences beyond the initialisation step found. For 5-dimensional and 20-dimensional problems no clear differences could be seen between Sobol sampling and Uniform sampling. The same holds true for Latin Hypercube sampling. At only 2 iterations across all used functions and dimensions, there was a significant difference found between the distribution of solutions found by Latin Hypercube sampling and Uniform sampling. Latin Hypercube did not show any big differences between dimensionalities and was overall very similar to Uniform sampling.

Future work could research other metrics, other than performance metrics. For example, the variance of initial sampling points or the covariance between dimensions could be researched, and the results can then be linked back to the performance metrics. Are these differences in variance still visible over time and does this impact the performance?

The mutation strategy in this research makes use of 1 difference vector, which is randomly chosen among the current 5 best solutions found, *pbest*. Our version of *pbest* used the 20 per cent best solutions as a possible difference vector. Our research however only focused on the best solutions found. Part of future research could be the investigation of the effect of the other solutions that can be possible difference vectors when using *pbest*. Are these solutions close together, are they well

spread out? What is the quality of those solutions and how do they impact the final result? Other mutation strategies could also be researched. Mutation strategies that do not use difference vectors, or use difference vectors that are randomly chosen or give a certain weight to these difference vectors. In what way does this influence the impact the sampling method has on the outcome? The population size could also be researched. When using a lower population size the initial sampling becomes more and more important. One could look at using a fixed population size. In this research, a population size of $10 * D$ was used, but a fixed population size could yield different results.

Other sampling methods should also be researched. More quasi-random sequences should be researched such as the Faure sequence. As the Faure sequence does not suffer from the same correlation between a set of points, that the Halton sequence suffers from in higher dimensions. To improve on the Halton sequence in higher dimensions, one could also look at the Randomized Halton Sequence [29]. More research should be done on the Hammersley sequence, which often significantly outperformed other sampling methods upon initialisation but could not capitalize on this lead. Why is this the case and could the algorithm be modified to mitigate these effects? To back up these claims more data is needed to provide for higher certainty and make more conclusions about small nuances. This would require more iterations of the algorithm for more instances and more repetitions.

7 References

[1] C. Audet and W. Hare, *Introduction: Tools and Challenges in Derivative-Free and Blackbox Optimization*, pp. 3–14. Cham: Springer International Publishing, 2017.

[2] S. Voß, S. Martello, I. H. Osman, and C. Roucairol, "Meta-heuristics: Advances and trends in local search paradigms for optimization," 2012.

[3] X.-S. Yang, S. F. Chien, and T. O. Ting, "Chapter 1 - bio-inspired computation and optimization: An overview," in *Bio-Inspired Computation in Telecommunications* (X.-S. Yang, S. F. Chien, and T. O. Ting, eds.), pp. 1–21, Boston: Morgan Kaufmann, 2015.

[4] D. Câmara, "1 - evolution and evolutionary algorithms," in *Bio-inspired Networking* (D. Câmara, ed.), pp. 1–30, Elsevier, 2015.

[5] L. Gui, X. Xia, F. Yu, H. Wu, R. Wu, B. Wei, Y. Zhang, X. Li, and G. He, "A multi-role based differential evolution," *Swarm and Evolutionary Computation*, vol. 50, p. 100508, 2019.

[6] M. Pant, R. Thangaraj, C. Grosan, and A. Abraham, "Improved particle swarm optimization with low-discrepancy sequences," pp. 3011–3018, 2008.

[7] H. Maaranen, K. Miettinen, and M. M. Mäkelä, "Quasi-random initial population for genetic algorithms," *Computers & Mathematics with Applications*, vol. 47, no. 12, pp. 1885–1895, 2004.

[8] S. Elsayed, R. Sarker, and C. A. C. Coello, "Sequence-based deterministic initialization for evolutionary algorithms," *IEEE transactions on cybernetics*, vol. 47, no. 9, pp. 2911–2923, 2016.

[9] Q. Li, S.-Y. Liu, and X.-S. Yang, "Influence of initialization on the performance of metaheuristic optimizers," *Applied Soft Computing*, vol. 91, p. 106193, 2020.

[10] J. O. Agushaka and A. E. Ezugwu, "Initialisation approaches for population-based metaheuristic algorithms: a comprehensive review," *Applied Sciences*, vol. 12, no. 2, p. 896, 2022.

[11] M. Georgioudakis and V. Plevris, "A comparative study of differential evolution variants in constrained structural optimization," *Frontiers in Built Environment*, vol. 6, 2020.

[12] R. Storn and K. V. Price, "Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[13] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.

[14] M. R. Hassanzadeh and F. Keynia, "An overview of the concepts, classifications, and methods of population initialization in metaheuristic algorithms," *Journal of Advances in Computer Engineering and Technology*, vol. 7, no. 1, pp. 35–54, 2021.

[15] M. Ali, R. C. Deo, N. J. Downs, and T. Maraseni, "Chapter 3 - monthly rainfall forecasting with markov chain monte carlo simulations integrated with statistical bivariate copulas," in *Handbook of Probabilistic Models* (P. Samui, D. Tien Bui, S. Chakraborty, and R. C. Deo, eds.), pp. 89–105, Butterworth-Heinemann, 2020.

[16] H. Faure, P. Kritzer, and F. Pillichshammer, "From van der corput to modern constructions of sequences for quasi-monte carlo rules," *Indagationes Mathematicae*, vol. 26, no. 5, pp. 760–822, 2015.

[17] J. E. Gentle, *Random number generation and Monte Carlo methods*, vol. 381. Springer, 2003.

[18] P. Bratley and B. L. Fox, "Algorithm 659: Implementing sobol's quasirandom sequence generator," *ACM Transactions on Mathematical Software (TOMS)*, vol. 14, no. 1, pp. 88–100, 1988.

[19] D. Vermetten, F. Caraffini, A. V. Kononova, and T. Bäck, "Modular differential evolution," *CoRR*, vol. abs/2304.09524, 2023.

[20] C. Doerr, H. Wang, F. Ye, S. Van Rijn, and T. Bäck, "Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics," *arXiv preprint arXiv:1810.05281*, 2018.

[21] J. de Nobel, F. Ye, D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "Iohexperimenter: Benchmarking platform for iterative optimization heuristics," *arXiv preprint arXiv:2111.04077*, 2021.

[22] N. Hansen, S. Finck, R. Ros, and A. Auger, *Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions*. PhD thesis, INRIA, 2009.

[23] D. Vermetten, M. López-Ibáñez, O. Mersmann, R. Allmendinger, and A. V. Kononova, "Analysis of modular cma-es on strict box-constrained problems in the sbox-cost benchmarking suite," *arXiv preprint arXiv:2305.15102*, 2023.

[24] H. Wang, D. Vermetten, F. Ye, C. Doerr, and T. Bäck, "Iohanalyzer: Detailed performance analyses for iterative optimization heuristics," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 2, no. 1, pp. 1–29, 2022.

[25] F. X. Long, D. Vermetten, B. van Stein, and A. V. Kononova, "Bbob instance analysis: Landscape properties and algorithm performance across problem instances," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pp. 380–395, Springer, 2023.

[26] H. Leon Harter, H. J. Khamis, and R. E. Lamb, "Modified kolmogorov-smirnov tests of goodness of fit," *Communications in Statistics-Simulation and Computation*, vol. 13, no. 3, pp. 293–323, 1984.

[27] C. Lewis, "Multiple comparisons," in *International Encyclopedia of Education (Third Edition)* (P. Peterson, E. Baker, and B. McGaw, eds.), pp. 312–318, Oxford: Elsevier, third edition ed., 2010.

[28] C. Schlier, "On scrambled halton sequences," *Applied Numerical Mathematics*, vol. 58, no. 10, pp. 1467–1478, 2008.

[29] X. Wang and F. Hickernell, "Randomized halton sequences," *Mathematical and Computer Modelling*, vol. 32, no. 7, pp. 887–899, 2000.

8 Appendix

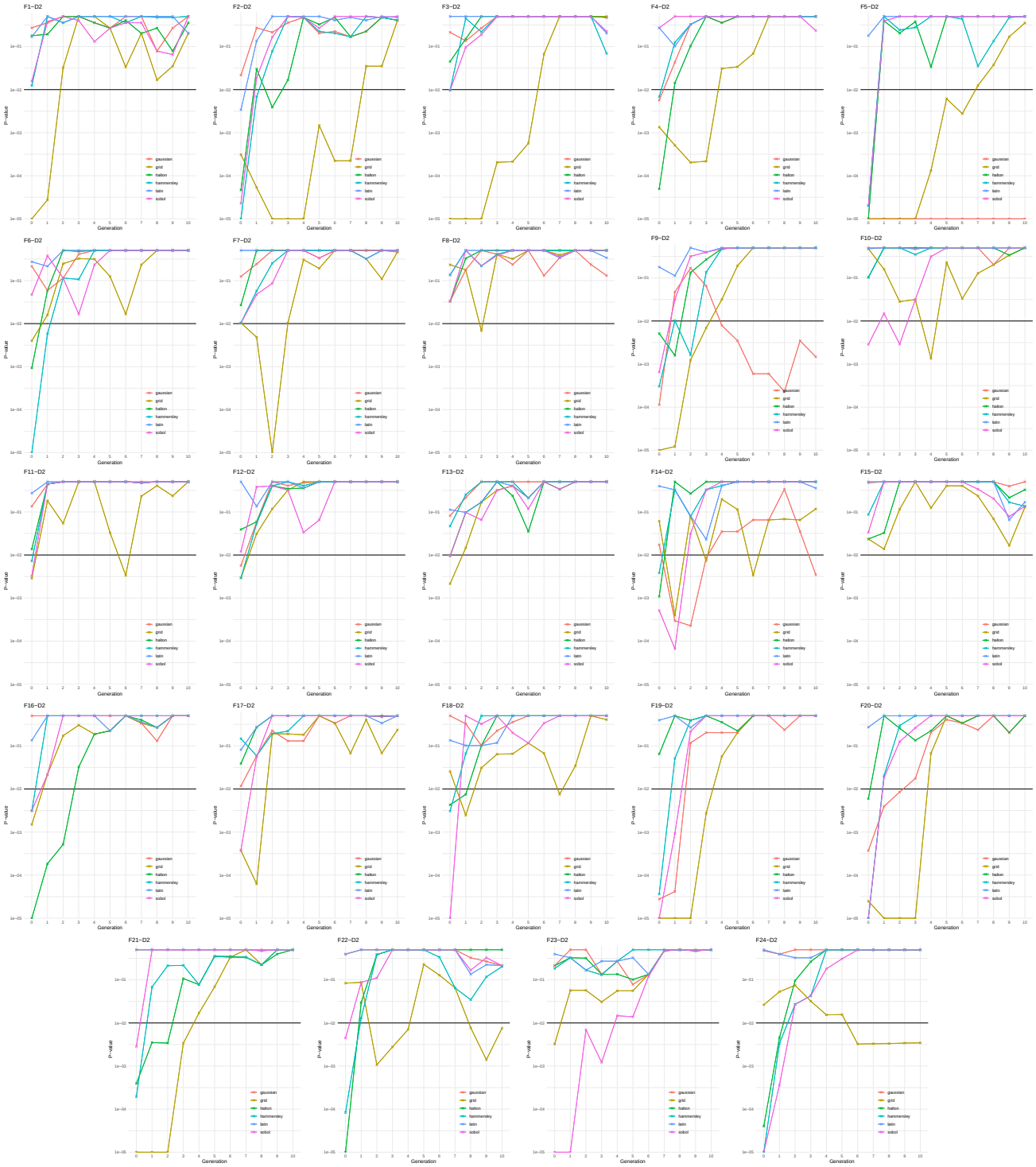


Figure 25: BBOB - 2D

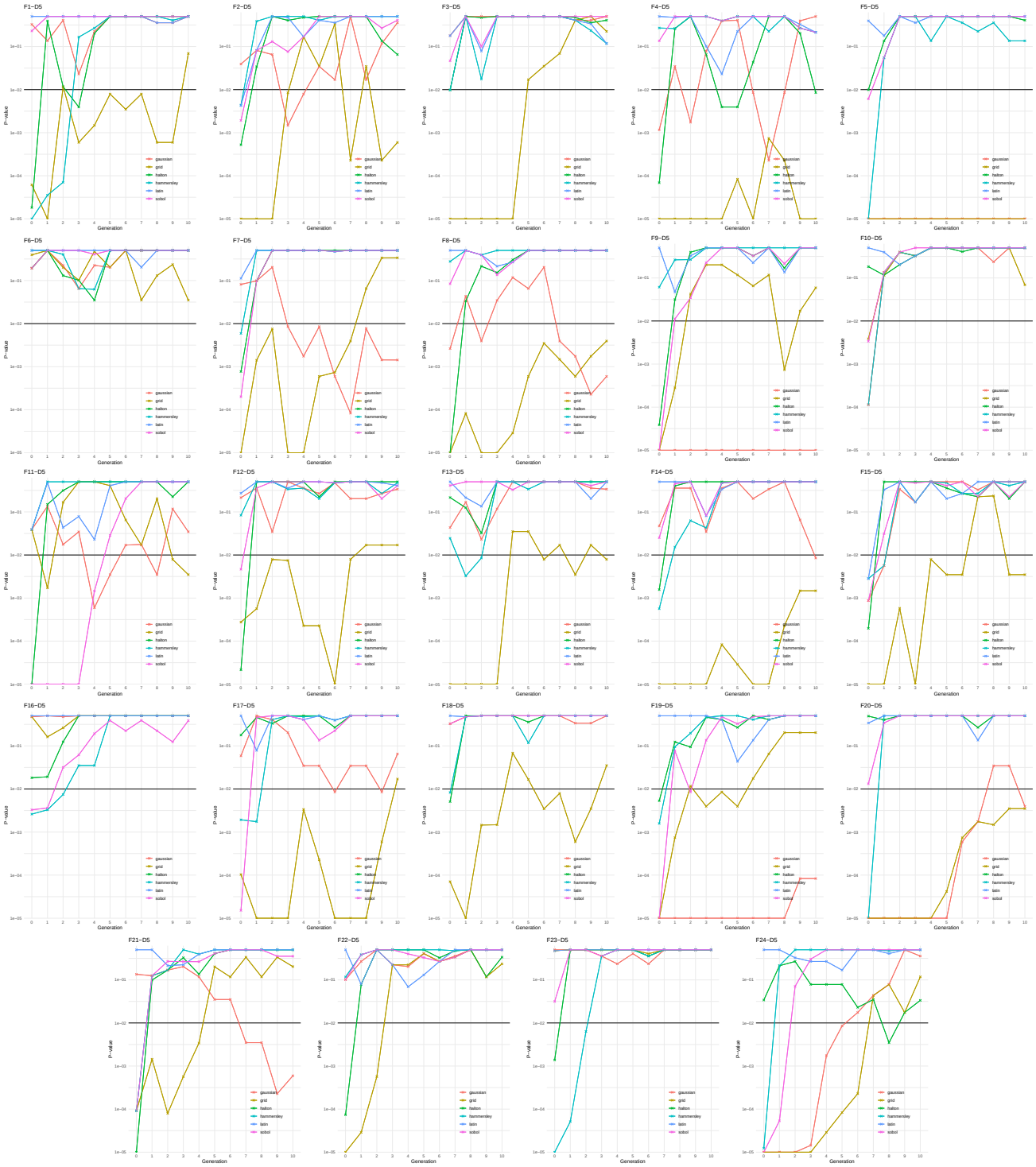


Figure 26: BBOB - 5D

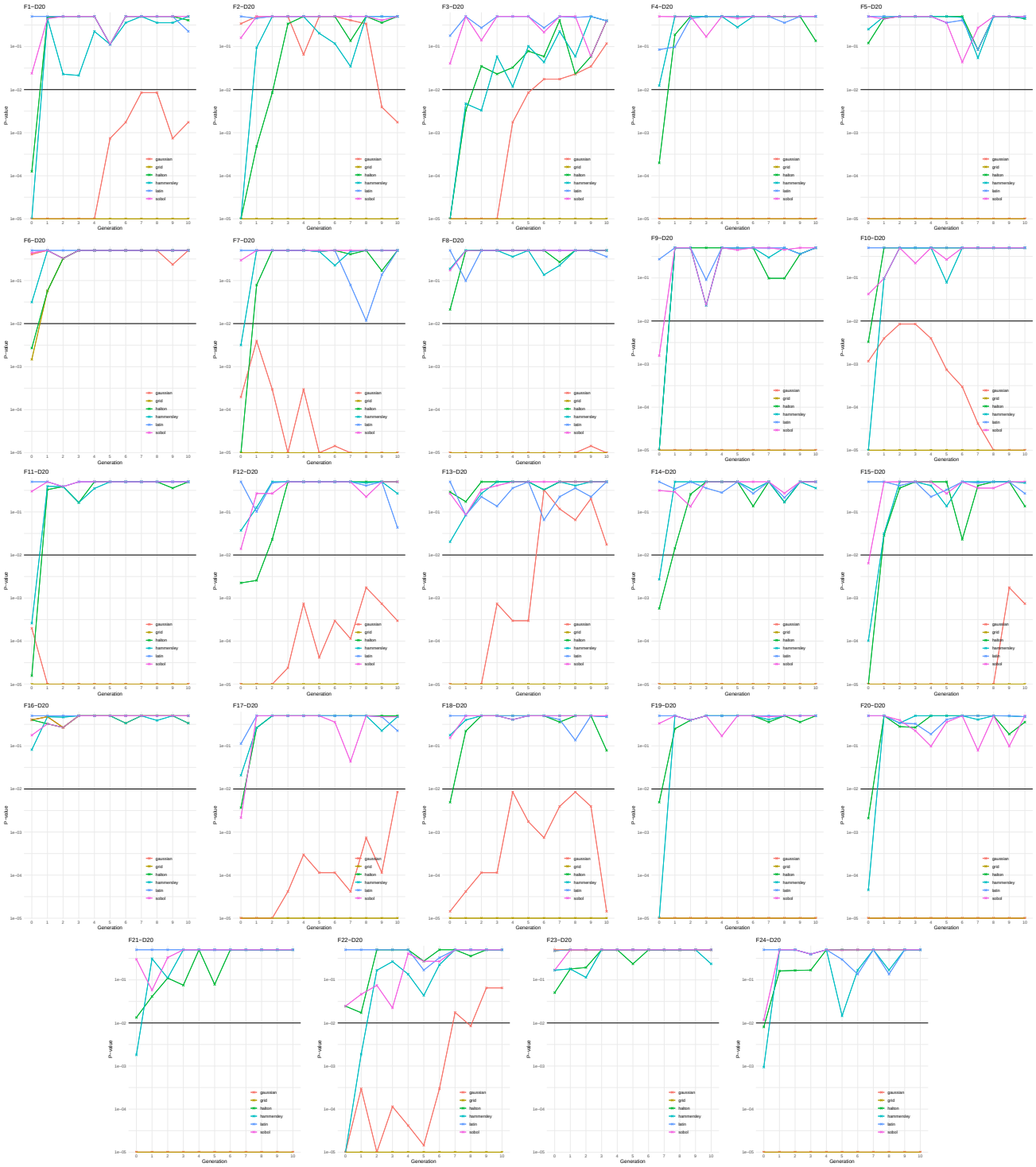


Figure 27: BBOB - 20D

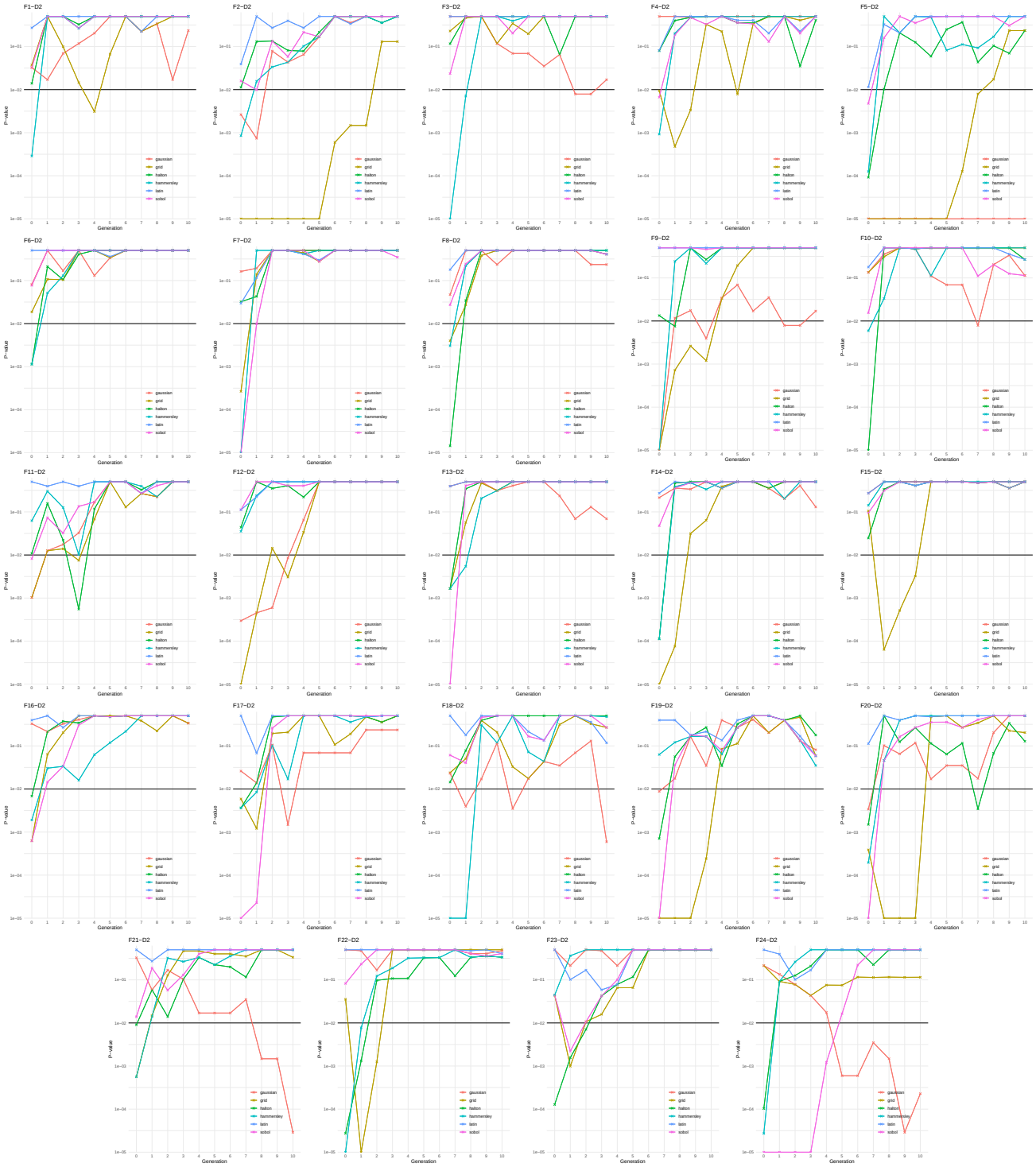


Figure 28: SBOX - 2D

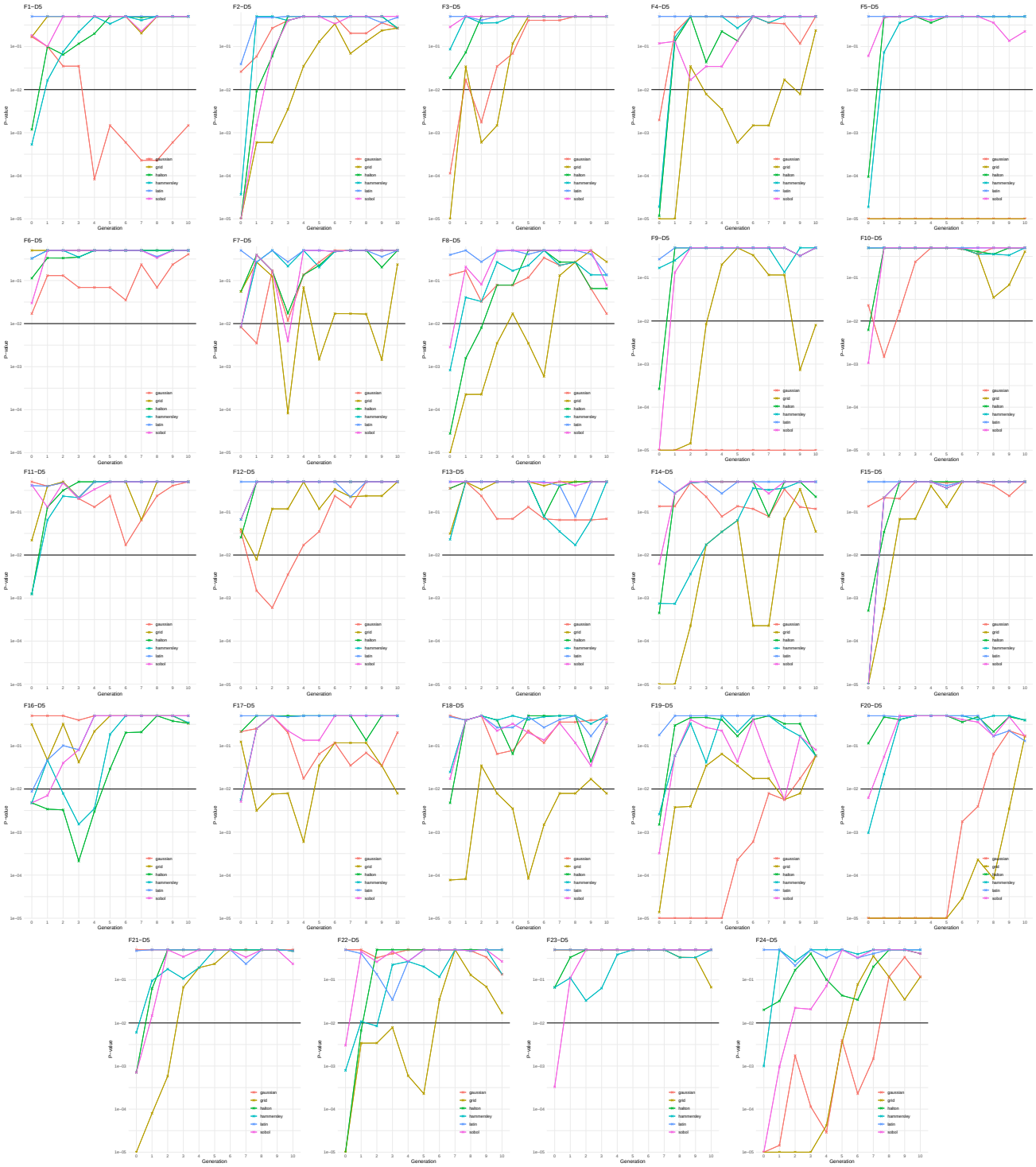


Figure 29: SBOX - 5D

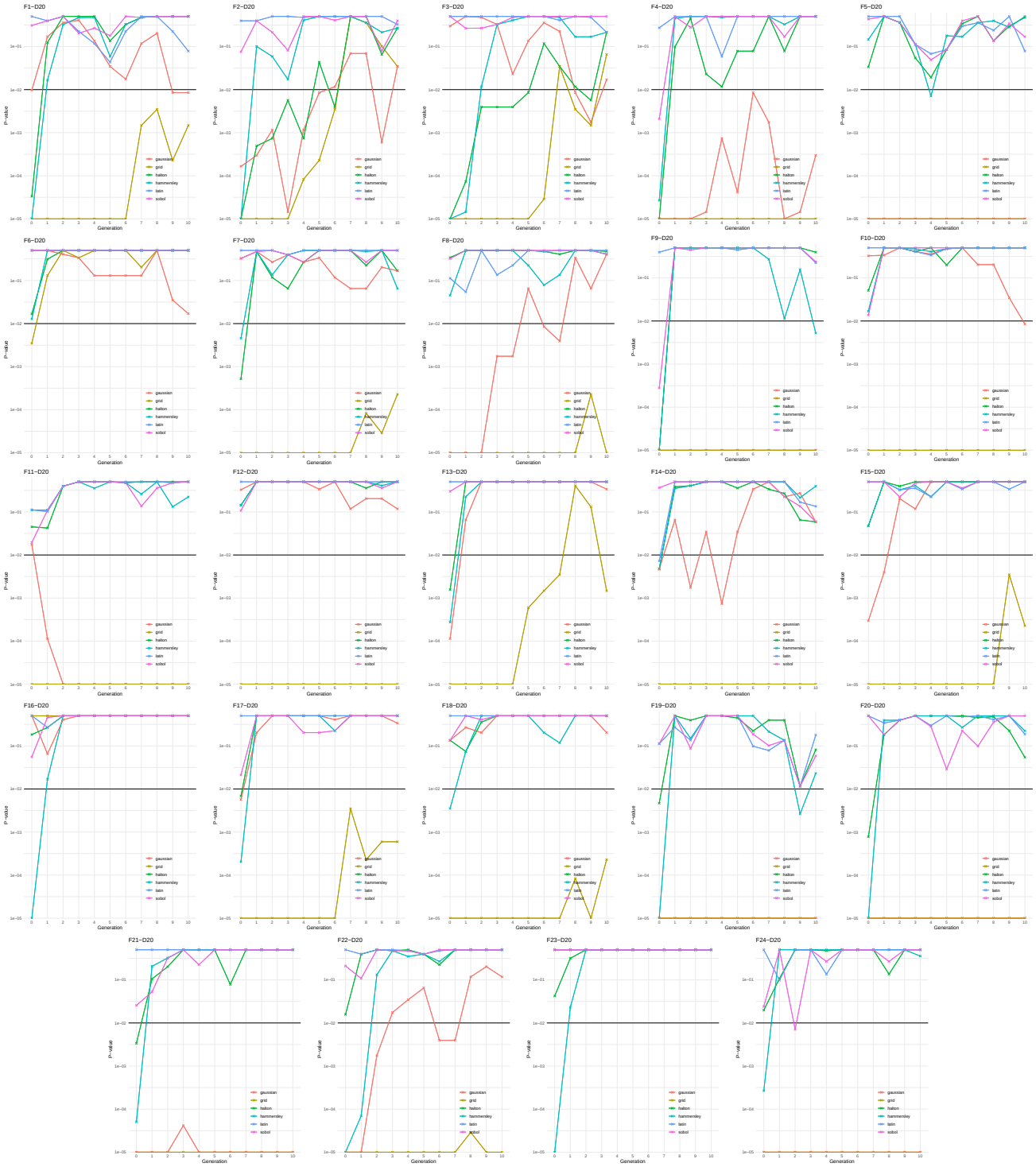


Figure 30: SBOX - 20D