Computing star discrepancies with numerical black-box optimization algorithms

Clement, F.; Vermetten, D.L.; Nobel J.P. de; Jesus, A.D.; Paquete, L.; Doerr, C.

Computing Star Discrepancies with Numerical Black-Box Optimization Algorithms

François Clément
Sorbonne Université, CNRS, LIP6
Paris, France

Diederick Vermetten
Leiden Institute for Advanced
Computer Science
Leiden, The Netherlands

Jacob de Nobel
Leiden Institute for Advanced
Computer Science
Leiden, The Netherlands

Alexandre D. Jesus
University of Coimbra, CISUC, DEI
Coimbra, Portugal

Luís Paquete
University of Coimbra, CISUC, DEI
Coimbra, Portugal

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

The L_∞ star discrepancy is a measure for the regularity of a finite set of points taken from $[0, 1)^d$. Low discrepancy point sets are highly relevant for Quasi-Monte Carlo methods in numerical integration and several other applications. Unfortunately, computing the L_∞ star discrepancy of a given point set is known to be a hard problem, with the best exact algorithms falling short for even moderate dimensions around 8. However, despite the difficulty of finding the global maximum that defines the L_∞ star discrepancy of the set, local evaluations at selected points are inexpensive. This makes the problem tractable by black-box optimization approaches.

In this work we compare 8 popular numerical black-box optimization algorithms on the L_∞ star discrepancy computation problem, using a wide set of instances in dimensions 2 to 15. We show that all used optimizers perform very badly on a large majority of the instances and that in many cases random search outperforms even the more sophisticated solvers. We suspect that state-of-the-art numerical black-box optimization techniques fail to capture the global structure of the problem, an important shortcoming that may guide their future development.

We also provide a parallel implementation of the best-known algorithm to compute the discrepancy.

CCS CONCEPTS

• **Computing methodologies** → **Randomized search**; **Parallel algorithms**.

KEYWORDS

Star discrepancy, Black-box optimization, Parallel computing, Evolutionary computation, Uniform distributions

1 INTRODUCTION

Discrepancy measures are designed to quantify how regularly a point set is distributed in a given space. Among the many discrepancy measures, the most common one is the L_∞ star discrepancy. The L_∞ star discrepancy of a finite point set $P \subseteq [0, 1)^d$ measures the worst absolute difference between the Lebesgue measure of a d-dimensional box anchored in $(0, \ldots, 0)$ and the proportion of points that fall inside this box.

The L_∞ star discrepancy is especially important because of the Koksma-Hlawka inequality in numerical integration [21, 24]. This inequality states that the error of approximating the integral $\int_{x \in [0,1]^d} f(x)dx$ of a function f by the average function value $\frac{1}{|P|} \sum_{p \in P} f(p)$ of f evaluated in the points $p \in P$ is upper-bounded by a term depending only on f and another term depending on the L_∞ star discrepancy of P. While we cannot control the integrand f, the Koksma-Hlawka inequality suggests that evaluating f in point sets P of small discrepancy is preferable. Quasi-Monte Carlo integration is undoubtedly the main motivation to study the design of point sets with low L_∞ star discrepancy values [8]. However, such point sets are also used in numerous other applications, including the design of experiments [36], computer vision [31], and financial mathematics [15]. In evolutionary computation, low-discrepancy point sets are considered in the context of one-shot optimization [2, 3], digital art [27], and TSP solvers [28].

The study of low-discrepancy sequences by mathematicians has focused on trying to obtain the optimal asymptotic order of the discrepancy of a sequence of points; see [30] or the more recent [8]. All currently known low-discrepancy sequences have a L_∞ star discrepancy of order $O(\log^d(n)/n)$, with n being the number of points [30]. When d increases, an exponential number of points is required for this bound to be smaller than 1, giving no information on the quality of the sets if n is small. Taking so many points is impossible for typical applications and thus requires the construction of low-discrepancy point *sets* of a fixed size. Put differently, in contrast to the asymptotic behavior studied by the classical low-discrepancy constructions, in Computer Science and similar applications we often require sets of low L_∞ star discrepancy value for concrete set size n. Recent attempts to construct such point sets, presented for example in [11] and [5], require a great number of discrepancy

computations, as it is often the only criterion available to differentiate candidate point sets. That is, we are not aware of (possibly low-fidelity) surrogates of the star discrepancy value. Obtaining efficient methods of computing L_∞ star discrepancies would therefore be an invaluable asset for the construction of low-discrepancy point sets.

Unfortunately, the problem is known to be hard to tackle exactly. More precisely, the decision version of the L_∞ star discrepancy computation was shown to be NP-hard [17] and even W[1]-hard with respect to the dimension [16]. The best exact algorithm to this day runs in $O(n^{1+d/2})$ [9]. This algorithm performs reliably well for up to a few hundred points in dimension 8, or dozens in dimension 10 (see Section 3.1). To calculate the L_∞ star discrepancy in higher dimensions, only a heuristic is available (based on Threshold Accepting [18]), reliable up to around dimension 20 (see Section 3.3). One of the main issues of the heuristic is that it only provides a lower bound: throughout the heuristic, we are only evaluating local discrepancies (the maximum of which defines the star discrepancy value of the set). Therefore, there are no guarantees on the quality of the discrepancy values returned by the heuristic. A comprehensive survey of the different attempts to compute the L_∞ star discrepancy can be found in Chapter 10 of [10].

Many applications would also require discrepancy calculations in much higher dimensions, in some cases $d \geq 100$, far out of reach for the moment. Despite the complexity of finding the global maximum, local evaluations are very cheap. Black-box optimization approaches can be used to tackle this problem, even making a very large number of local evaluations is feasible as the Threshold Accepting heuristic has shown.

Our Contributions. We show in this work that popular numerical black-box optimizers are unsuccessful in computing the L_∞ star discrepancy, even for instances that can be solved within a second with naïve methods. We apply 8 optimizers on three different point set types, for varying dimensions and set sizes, and show that their performance is globally bad. The relative performance rapidly decreases with increasing dimension, whereas the size of the point sets only seems to have a minor impact on the overall (bad) quality of the solvers, which might be surprising given that the differences in the local star discrepancy values at the discontinuities become smaller with increasing set size (inverse linear relationship). The sampler type has relatively little importance, if any, on the performance of the optimizers. We observe that uniform i.i.d. random search seems to be the best performing optimizer over the vast majority of the instances, yet still fails to come close to the exact value, as highlighted by a relative performance comparison with the known exact values.

We complement our empirical comparison of "off-the-shelf" numerical black-box optimization algorithms with a refinement of the problem-specific exact algorithm. More precisely, to stretch the settings for which we can obtain the exact L_∞ star discrepancy values of the problem instances, we provide a new parallel implementation of the Dobkin, Eppstein, Mitchell algorithm [9], the best known to this day to compute the star discrepancy. We show that our parallel implementation has a speed-up factor of up to 17 on 32 threads. This implementation is of independent interest to the discrepancy and numerical integration communities.

Outline. In Section 2, we introduce the discrepancy computation problem and give some basic properties. The best discrepancy algorithms are described in Section 3, along with a performance analysis of our parallel implementation. Section 4 describes the instances and optimizers chosen, while Section 5 describes the results obtained.

Availability of code. Our implementations can be found in a Zenodo repository [6], along with past implementations by the authors of [18]. The repository also contains all code and data used for the analysis of the black-box optimization algorithms, as well as the code used to process this and create the figures included in this paper.

2 L_∞ STAR DISCREPANCY

We provide in this section a formal definition of the L_∞ star discrepancy and discuss some basic properties.

2.1 Definition and properties

Definition 2.1. Let $P \subseteq [0, 1)^d$ be a finite set of points. For $q \in [0, 1]^d$, the local L_∞ star discrepancy of P in q is defined as the absolute difference between the (standard Lebesgue) volume of the box $[0, q)$, $\lambda(q)$, and the fraction of points of P that fall inside this box,

$$d_\infty^*(P, q) := \left| \frac{|P \cap [0, q)|}{|P|} - \lambda(q) \right|.$$

The L_∞ star discrepancy of P measures the worst local discrepancy, i.e.,

$$d_\infty^*(P) := \sup_{q \in [0,1]^d} d_\infty^*(P, q).$$

As mentioned in the introduction, computing the L_∞ star discrepancy of a point set is a difficult task that likely cannot be done in polynomial time in d and $|P|$. Evaluating the local L_∞ star discrepancy, however, is rather simple and can be done naively in $O(d|P|)$. Indeed, it only requires checking if $|P|$ given points are in a specific box in $[0, 1)^d$.

Properties of the local L_∞ star discrepancy problem. Figure 1 illustrates the local L_∞ star discrepancy values for an instance in $d = 2$. We can already observe that the problem of maximizing the local L_∞ star discrepancy bears two important challenges: (1) it is a *multimodal* problem, i.e., there can be several local optima in which the solvers can get trapped (this problem becomes worse with increasing dimension); (2) there are *sharp discontinuities* in the local discrepancy values. Slightly increasing one coordinate can result in a point falling inside the considered box, causing a $1/|P|$ difference in the local star discrepancy value. Figure 2 shows that the problem structure also depends strongly on the point set considered.

2.2 Discrete Embedding

The star discrepancy can also be expressed as a discrete problem, as shown by Niederreiter in [29]. Since every closed box in $[0, 1]^d$ can be obtained as the limit of a sequence of open boxes, the definition of star discrepancy can be extended to include closed boxes. We see that for a fixed $q \in [0, 1]^d$, the local discrepancy can be expressed as the maximum of a local discrepancy term for the open box $[0, q)$

and a local discrepancy term for the closed box $[0, q]$, rather than with an absolute value.

More formally, we define $D(q, P)$ to be the number of points of $P := (x^{(i)})_{i \in \{1,...,n\}}$ that fall inside the open anchored box $[0, q)$, $\overline{D}(q, P)$ the number of points of P that fall inside the closed anchored box $[0, q]$ and the two following functions

$$\delta(q, P) := \lambda(q) - \frac{1}{n}D(q, P) \qquad \text{and} \qquad \overline{\delta}(q, P) := \frac{1}{n}\overline{D}(q, P) - \lambda(q),$$

where here and in the following $n = |P|$.

The local discrepancy in a point $q \in [0, 1]^d$ is given by the maximum of $\delta(q, P)$ and $\overline{\delta}(q, P)$. Given this formulation, we can now show that only positions on specific grids can reach the maximum value for the star discrepancy. For all $j \in \{1, \ldots, d\}$, we define

$$\Gamma_j(P) := \{x_j^{(i)} : i \in 1, \ldots, n\} \qquad \text{and} \qquad \overline{\Gamma}_j(P) := \Gamma_j(P) \cup \{1\},$$

and

$$\Gamma(P) := \Gamma_1(P) \times \ldots \times \Gamma_d(P) \qquad \text{and} \qquad \overline{\Gamma}(P) := \overline{\Gamma}_1(P) \times \ldots \times \overline{\Gamma}_d(P).$$

For any $q \in [0, 1]^d \setminus \overline{\Gamma}(P)$, it is always possible to either increase or decrease slightly one of the coordinates of q without changing the number of points inside the (open or closed) box. If the local discrepancy was given by $\delta(q, P)$ (respectively $\overline{\delta}(q, P)$), increasing (respectively decreasing) the volume will make the local discrepancy worse. Since only points in $\overline{\Gamma}(P)$ can reach the maximum value for the star discrepancy, we obtain with the open/closed box distinction

$$d_\infty^*(P) := \max \left\{ \max_{q \in \overline{\Gamma}(P)} \delta(q, P), \max_{q \in \Gamma(P)} \overline{\delta}(q, P) \right\}. \qquad (1)$$

From this grid definition, the L_∞ star discrepancy can be computed by calculating local discrepancies for each of the $O(n^d)$ points in the grid. This can be further refined by noticing that only boxes whose outer edges have at least one point on each of them (or coordinate 1 for open boxes) can reach the worst discrepancy value, bringing the total number of local discrepancies to evaluate down to $O(n^d/(d!))$. These boxes are called *critical boxes*. To our knowledge, this method can be used only for relatively small sets: tests in [42] indicate that 236 points in dimension 5 or 92 in dimension 6 seem to be the limit.

Convention. Since in this work we only consider the L_∞ star discrepancy, the L_∞ star discrepancy may be referred to as simply discrepancy or star discrepancy from hereon.

3 PROBLEM-SPECIFIC ALGORITHMS

We describe in this section the baselines against which we will compare the numerical black-box optimization approaches. The currently best-known *exact* algorithm for computing the L_∞ star discrepancy of a given point set is presented in Section 3.1. In Section 3.2 we present our parallelization of this approach. Finally, in Section 3.3 we briefly discuss a *problem-specific local search algorithm* that we use to evaluate instances for which we cannot compute the exact value of the L_∞ star discrepancy.

3.1 The DEM Algorithm

The best exact algorithm to compute the star discrepancy was introduced by Dobkin, Eppstein and Mitchell in [9]. It involves

building a decomposition of $[0, 1]^d$ in $O(n^{d/2})$ disjoint boxes B_i, such that we can find the maximum local discrepancy for a top-right corner in B_i in linear time. This leads to a total complexity of $O(n^{1+d/2})$. As mentioned in the introduction, the algorithm can be used up to dimension 8 for a few hundred points or dimension 10 for a few dozens. While the initial algorithm in [9] was based on a point-box dualization, our description in this subsection will follow a more direct approach, as in Chapter 10 of [10] and in the original implementation by the authors of [18]. Some border cases will be ignored for the clarity of the proof, as they do not change the general idea of the implementation or the stated complexity.

Firstly, a point x is said to be *internal* in dimension j for a box $[a, b]$ if $a_j < x_j < b_j$. Starting from $[0, 1]^d$, we build dimension-by-dimension boxes $[a, b]$ such that the two following properties are verified: (1) Any $x \in [0, b)$ is internal in at most one dimension; (2) For each box built up to dimension j in $\{1, \ldots, d\}$, there are at most $O(\sqrt{n})$ points internal in dimension j - for a fully-built box $[a, b]$ this holds for all dimensions.

The decomposition is built recursively in the following manner. We consider $[0, 1]^d$ and start the decomposition in the first dimension. We find the smallest coordinate $c_{1,1}$ such that $[0, c_{1,1}] \times [0, 1]^{d-1}$ contains $\sqrt{n}$ points, then $c_{1,2}$ such that $[c_{1,1}, c_{1,2}] \times [0, 1]^{d-1}$ contains $\sqrt{n}$ points and continue until we obtain a set $(c_{1,i})_{i \in \{0,...,\lceil\sqrt{n}\rceil\}}$ where $c_{1,0} = 0$ and the last non-zero $c_{1,j} = 1$. For each of these boxes $[c_{1,i}, c_{1,i+1}] \times [0, 1]^{d-1}$, we track which points are internal in dimension 1 (at most $\sqrt{n}$) and which points are inside the box $[0, c_{1,i+1}] \times [0, 1]^{d-1}$.

Given a box $B_j = [c_{1,i_1}, c_{1,i_1+1}] \times \ldots \times [c_{j,i_j}, c_{j,i_j+1}] \times [0, 1]^{d-j}$, we recursively perform a similar decomposition in dimension $j + 1$. Let n_j be the number of points inside $\overline{B_j} := [0, c_{1,i_1+1}] \times \ldots \times [0, c_{j,i_j+1}] \times [0, 1]^{d-j}$. The new $c_{j+1,i}$ need to verify the two following properties. Firstly, for any point x in $\overline{B_j}$ that is internal in one of the j first dimensions, there needs to be some $c_{j+1,i} = x_{j+1}$. Secondly, for any $i \in \{0, \ldots, \lceil\sqrt{n}\rceil - 1\}$, there are at most $\sqrt{n}$ points y in $\overline{B_j}$ such that $c_{j+1,i} < y_{j+1} < c_{j+1,i+1}$. The first property guarantees that a point will never be internal in multiple dimensions, and the second that not too many points are in the boxes obtained from $\overline{B_j}$.

After d steps, we obtain boxes $[a, b]$ verifying the two desired properties: a point is internal in at most one dimension and there are at most $O(\sqrt{n})$ points internal in each dimension. We can now find the worst local discrepancy for a box whose top-right corner is in $[a, b]$ with the following dynamic programming approach. Let $m(h, j)$ (respectively $r(h, j)$) be the maximum (respectively minimum) value of $\prod_{i=1}^{j} y_i$ such that the box $[0, y_1] \times \ldots \times [0, y_j] \times [0, b_{j+1}] \times \ldots \times [0, b_d]$ contains exactly h points. By ordering the points in $[0, b]$ according to their first dimension, we can easily obtain $m(\cdot, 1)$ and $r(\cdot, 1)$. Since points can only be internal in a single dimension, taking an internal point in dimension 1 guarantees it will be contained in the box regardless of the other choices. From the $m(\cdot, j)$ and $r(\cdot, j)$, we can therefore calculate $m(\cdot, j + 1)$ and $r(\cdot, j + 1)$. Calculating each $m(\cdot, j + 1)$ takes $O(\sqrt{n})$ time as there are $O(\sqrt{n})$ internal points and therefore potential different choices for the coordinate y_{j+1}. In total, there are at most $d\sqrt{n}$ internal points, $m(\cdot, j + 1)$ needs to be computed for $O(d\sqrt{n})$ different values. The dynamic programming takes $O(dn)$ time (the
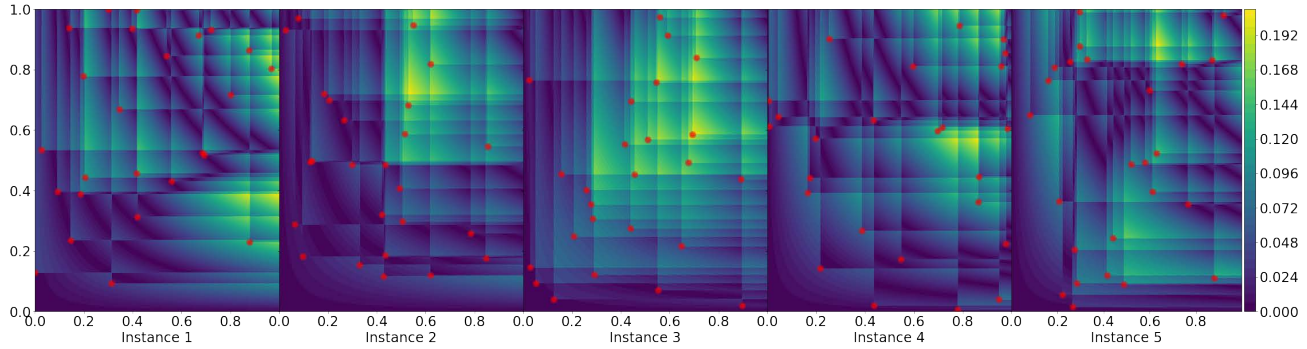
Figure 1: Discrepancy values of the first 5 instances of the 2-dimensional version of F31: Uniform sampler with 20 points. The red points indicate the originally sampled points.
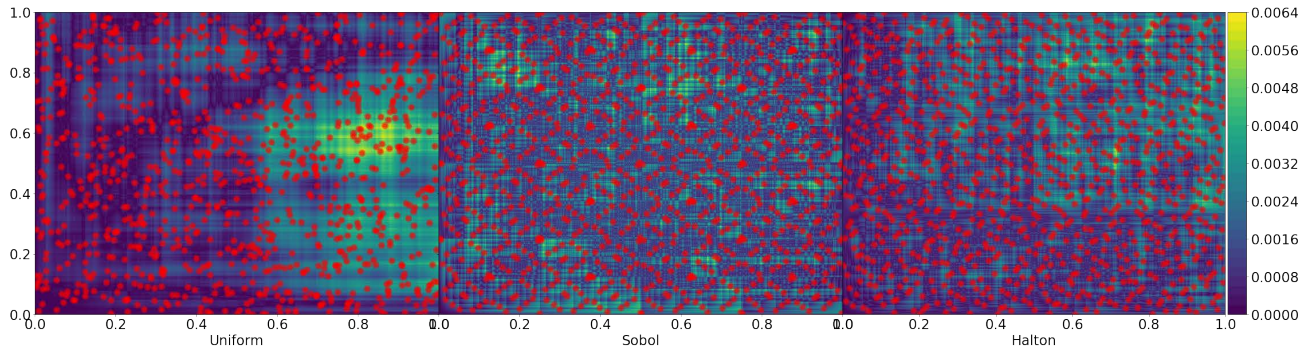

Figure 2: Comparison of the 3 samplers for 1000 points in 2D, and the corresponding discrepancy landscapes (instance 1 for F39, F49 and F59 respectively).

d factor is usually ignored). All that remains to be done is to find $\max_{1 \le h \le n}(m(h, d) - h/n, h/n - r(h, d))$, in other words which (number of points inside a box, box volume) combination gives the worst discrepancy. This will directly give the worst discrepancy value for a box whose top-right corner is in $[a, b)$.

3.2 Parallelizing DEM

To parallelize the DEM algorithm we start by noting that after computing the boxes for a given dimension we can continue the recursive decomposition independently for each box. This naturally gives way to a parallel-task construct where each task that can be scheduled to an available CPU corresponds to the decomposition in dimension $j + 1$ of a given box B_j.

At this point, it is worth noting that another option initially considered was to parallelize the dynamic programming algorithm that is used to compute the worst local discrepancy for a box after d steps. However, this idea was discarded since the dynamic programming algorithm was very fast in practice and preliminary attempts did not yield significant speedups.

To implement the parallel-tasks construct we considered OpenMP Tasks [1] and we adapted the implementation of the DEM algorithm by the authors of [18]. In particular, one relevant change

was that the sequential implementation reused the set of points between recursive calls, with a possible resorting of the points in the current dimension after each recursive call. However, in the parallel variant, we cannot efficiently share the set of points since sorting operations on the set of points in a thread could potentially affect tasks running in other threads. Instead, in the parallel DEM we opted to copy the set of points before entering a recursive call. Although this requires an extra copy before each recursive call, we observed that in some cases copying the vector was faster than resorting after each recursive call, suggesting that the non-parallel DEM implementation could be slightly improved by taking this into account.

Finally, in order to avoid thread starvation with a recursive parallel-tasks construct, it is often useful to consider a cut-off value related to the problem size to switch to a non-parallel variant. In this case, we define the cut-off with respect to the complexity of the DEM algorithm, i.e., $O(n^{1+d/2})$. In particular, when decomposing a box B_j with n_j points in dimension j, we switch to the non-parallel DEM algorithm if $n_j^{1+(d-j)/2} < 1e8$. This cut-off value was defined empirically by experimenting with different values up to $1e15$.

Table 1 gives the CPU time in seconds for the parallelized DEM on several instances generated with the GNU Scientific Library

(using the indicated sequence) and considering different numbers of threads. For a single thread, the best time between the non-parallel and parallel implementations is shown. Experiments were run on a machine with two Intel(R) Xeon(R) Silver 4210R CPUs clocked at 2.40GHz, comprising a total of 20 cores and 40 threads. We can see that for 32 threads we have speedups that range from a factor of 7.4 to a factor of 17.4. These results show that parallelization is generally successful and can give a significant boost. However, we believe that there is still room for improvement by further tuning the cut-off between the parallel and non-parallel variants, and by further analyzing when it is best to copy or resort the set of points.

3.3 The TA Algorithm

Several applications of point sets with low star discrepancy value concern settings that are not efficiently tractable by the exact algorithms described in the previous two subsections. For these applications, we therefore need to resort to heuristic approaches to evaluate the discrepancy of a given point set. To date, the best-known heuristic is a Threshold Accepting (TA) algorithm proposed in [18]. This approach exploits the grid structure introduced in Section 2.2 and operates on the search space $[1..n + 1]^d$, with each point encoding one of the grid points in $\overline{\Gamma}(P)$.

The TA algorithm from [18] builds on an earlier approach by Winker and Fang suggested in [42] and extends it by various problem-specific components. Threshold Accepting [13] is similar to Simulated Annealing [23] but replaces its probabilistic selection criterion with a deterministic one. That is, at each step, the current incumbent solution is compared to a randomly sampled neighboring solution. The neighbor is selected as the new center if its quality is not much worse than that of the previous incumbent. More precisely, neighbor y replaces incumbent x if $f(y) - f(x) \geq \tau(t)$, where $\tau(t) \leq 0$ is the threshold chosen at iteration t. The sequence $(\tau(t))_t$ is monotonically increasing so that the further advanced the optimization process is, the harder the selective pressure.

The TA algorithm from [18] uses a dynamic choice of the neighborhood structure, increasing the number of coordinates $\{j \mid x_j \neq y_j\}$ that may change in each iteration while at the same time decreasing the absolute difference $|x_j - y_j|$. The more important problem-specific component, however, is a "snapping" routine, which rounds a selected grid point to a so-called critical box; see [18] for details.

Using the DEM solver from Section 3.1 as a baseline, it was shown in [18] that the TA algorithm successfully found the optimum on all instances for which DEM could provide exact values. Based on the results presented in [18], the TA algorithm seems reliable for point sets up to dimensions 12 to 20 for a few hundred points.

4 NUMERICAL BLACK-BOX OPTIMIZATION APPROACHES

The L_∞ star discrepancy problem was included as a black-box benchmark problem in IOHexperimenter (version 0.3.7) [7], using three different point set generators:

(1) uniform random sampling,
(2) Halton [19],
(3) Sobol' [38].

For the uniform sampler, a standard Mersenne Twister 19937 pseudo-random number generator was used, provided by the C++ STL. The Halton sequence was generated using a classic Sieve of Eratosthenes prime number generating algorithm, and the Sobol' sequence was generated with a third-party library based on the FORTRAN implementation of Fox [14].

In addition to the generator, the number of points to be sampled can be selected, which along with the dimensionality, controls the complexity of the problem. We define a default suite, which includes for every generator a fixed number of samples $S \in \{10, 25, 50, 100, 150, 200, 250, 500, 750, 1000\}$. This includes a total of 30 benchmark problems, where for each problem the dimension and instance can be varied arbitrarily. Instances are controlled by a unique instance identifier, which is a positive integer that determines the random seed used to generate the point set. The problems can be accessed through both the Python and C++ interfaces of IOHexperimenter, with problem ids $\{30, \ldots, 39\}$, $\{40, \ldots, 49\}$, and $\{50, \ldots, 59\}$ for the problems generated with the uniform, Sobol', and Halton generators, respectively.

We test a total number of eight algorithms, all taken from the Nevergrad platform [35]:

(1) Diagonal Covariance Matrix Adaptation Evolution Strategy (dCMA-ES) [20]
(2) NGOpt14, Nevergrad's algorithm selection wizard [26]
(3) Estimation of Multivariate Normal Algorithm (EMNA) [25]
(4) Differential Evolution [40]
(5) Constrained Optimization BY Linear Approximation (Cobyla) [32]
(6) Random Search
(7) Particle Swarm Optimization (PSO) [22]
(8) Simultaneous Perturbation Stochastic Approximation algorithm (SPSA) [39]

The algorithms are chosen "as they are" from Nevergrad. That is, we did not perform any hyper-parameter tuning nor did we change any of their components. Each algorithm is given a total budget of $2\,500 \cdot d$ local discrepancy evaluations, where d is the dimensionality of the problem. For each instance, 10 independent runs of the algorithm are performed. This is repeated for 10 instances, resulting in a total of 100 runs per function, for each of the 30 functions, with dimensionality $d \in \{2, 3, 4, 6, 8, 10, 15\}$.

We run all our experiments in the IOHprofiler environment [12]. This allows us to track not only the final performance, but also the trajectory of the algorithms in objective space. It furthermore allows a straightforward visualization and analysis of the data using the IOHanalyzer module [41].

For all instances in dimensions 2, 3 and 4, and for all the instances in dimensions 6, 8, 10, and 15 with n not larger than 750, 200, 50 and 10 points respectively, we computed the exact discrepancy values using the parallel DEM algorithm proposed in Section 3.2. For all other instances, we computed a lower bound for the star discrepancy value using the TA algorithm described in Section 3.3.

5 RESULTS

We can compare the performance of the used optimization algorithms by considering the expected running time (ERT) to reach

			CPU time (speedup) per number of threads					
N. Dim.	N. Points	Sequence	1	2	4	8	16	32
2	**50000**	halton	2.54 (1.0)	1.56 (1.6)	0.88 (2.9)	0.53 (4.8)	0.38 (6.7)	0.32 (7.9)
		niederreiter	2.50 (1.0)	1.56 (1.6)	0.87 (2.9)	0.54 (4.6)	0.34 (7.4)	0.31 (8.1)
		reversehalton	2.37 (1.0)	1.59 (1.5)	0.86 (2.8)	0.53 (4.5)	0.40 (5.9)	0.32 (7.4)
		sobol	2.39 (1.0)	1.58 (1.5)	0.86 (2.8)	0.53 (4.5)	0.41 (5.8)	0.31 (7.7)
3	**10000**	halton	15.70 (1.0)	8.19 (1.9)	4.31 (3.6)	2.37 (6.6)	1.52 (10.3)	1.13 (13.9)
		niederreiter	15.77 (1.0)	8.09 (1.9)	4.36 (3.6)	2.39 (6.6)	1.65 (9.6)	1.16 (13.6)
		reversehalton	15.57 (1.0)	7.99 (1.9)	4.37 (3.6)	2.37 (6.6)	1.54 (10.1)	1.17 (13.3)
		sobol	15.74 (1.0)	8.19 (1.9)	4.38 (3.6)	2.39 (6.6)	1.44 (10.9)	1.14 (13.8)
4	**3000**	halton	49.66 (1.0)	26.40 (1.9)	13.97 (3.6)	7.38 (6.7)	3.99 (12.4)	2.89 (17.2)
		niederreiter	50.31 (1.0)	26.82 (1.9)	14.33 (3.5)	7.42 (6.8)	3.96 (12.7)	2.89 (17.4)
		reversehalton	50.42 (1.0)	26.53 (1.9)	14.11 (3.6)	7.44 (6.8)	4.05 (12.4)	2.96 (17.0)
		sobol	50.41 (1.0)	27.20 (1.9)	14.31 (3.5)	7.32 (6.9)	4.06 (12.4)	2.92 (17.3)
5	**1000**	halton	58.87 (1.0)	32.54 (1.8)	16.70 (3.5)	9.18 (6.4)	5.10 (11.5)	3.83 (15.4)
		niederreiter	62.32 (1.0)	33.39 (1.9)	17.78 (3.5)	9.67 (6.4)	5.34 (11.7)	4.08 (15.3)
		reversehalton	62.08 (1.0)	32.77 (1.9)	17.54 (3.5)	9.64 (6.4)	6.88 (9.0)	3.90 (15.9)
		sobol	63.80 (1.0)	33.89 (1.9)	17.73 (3.6)	9.77 (6.5)	5.34 (11.9)	4.04 (15.8)
6	**600**	halton	175.79 (1.0)	103.74 (1.7)	53.04 (3.3)	28.82 (6.1)	17.12 (10.3)	12.57 (14.0)
		niederreiter	196.24 (1.0)	109.22 (1.8)	57.99 (3.4)	30.44 (6.4)	17.94 (10.9)	12.71 (15.4)
		reversehalton	187.99 (1.0)	114.35 (1.6)	58.75 (3.2)	31.56 (6.0)	19.12 (9.8)	13.49 (13.9)
		sobol	200.76 (1.0)	112.53 (1.8)	59.18 (3.4)	30.91 (6.5)	18.34 (10.9)	13.32 (15.1)
7	**300**	halton	160.25 (1.0)	94.43 (1.7)	48.22 (3.3)	27.10 (5.9)	15.64 (10.2)	11.70 (13.7)
		niederreiter	167.32 (1.0)	96.85 (1.7)	51.36 (3.3)	28.40 (5.9)	15.80 (10.6)	12.66 (13.2)
		reversehalton	149.43 (1.0)	88.79 (1.7)	50.57 (3.0)	29.72 (5.0)	16.63 (9.0)	12.49 (12.0)
		sobol	169.70 (1.0)	102.12 (1.7)	52.72 (3.2)	29.68 (5.7)	16.20 (10.5)	12.41 (13.7)

Table 1: CPU time in seconds and corresponding speedup in parenthesis for the parallelized DEM algorithm
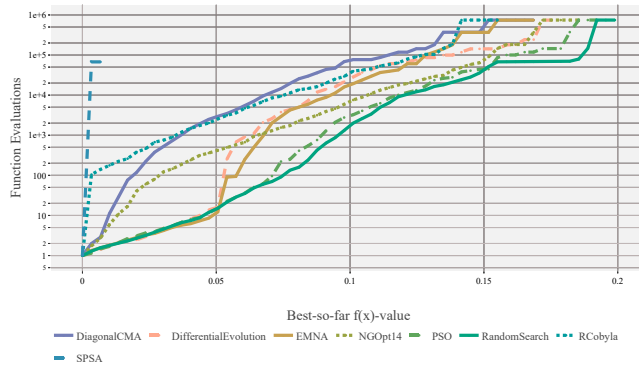

Figure 3: Expected running time (ERT) of the 8 optimization algorithms on the discrepancy calculation for the uniform sampler with 100 samples (F33) in 3 dimensions. ERT is calculated based on 10 runs on 10 instances with a budget of 7 500. Figure generated using IOHanalyzer [41].

increasing discrepancy values. This analysis shows the convergence behavior on the selected function. In Figure 3, we show the ERT on the 3-dimensional version of F33: the uniform sampler

with $n = 100$. From this figure, we can clearly see that the algorithms struggle to optimize this function. Particularly noticeable is the SPSA algorithm, which seems to fail to find even slightly improved discrepancy values. On the other side, we notice that Random Search is surprisingly outperforming all other optimizers. This seems to indicate that even for this relatively simple setting of $n = 100$ and $d = 3$, the high level of multi-modality combined with discontinuities in the landscape cause problems for all of the considered optimization algorithms.

To check whether this is a consistent problem, or something specific to the settings chosen in Figure 3, we can look in more detail at the final solutions found by each optimizer across a wider set of scenarios. In order to create a fair comparison, we can move from the original discrepancy values to a relative measure, based on the bounds found by the TA and DEM algorithms. Specifically, we consider the following measure:

$$R(x) = \frac{OPT(x) - f(x)}{OPT(x)},$$

where $f(x)$ is the final value found after the optimization run, and $OPT(x)$ is the bound calculated by the parallel DEM or TA, depending on the instance size.

Using this relative measure, we can compare the final solutions found by each optimization algorithm across a set of different n
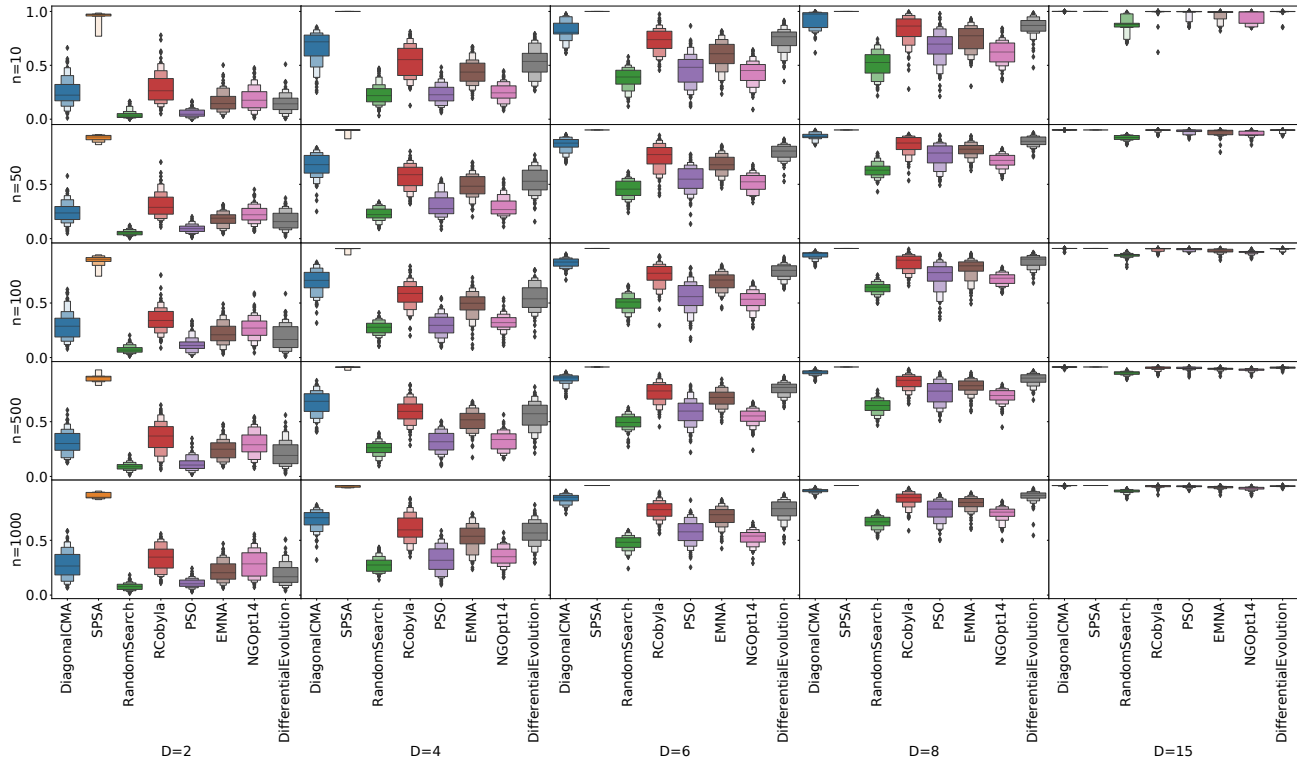
Figure 4: Relative final discrepancy value found by each of the used optimizers. Values of 0 correspond to finding the optimal solution, while 1 corresponds to the worst achievable value (0 discrepancy). Box-plots are aggregations of 10 runs on 10 instances, all for the uniform sampler.

and d. This is visualized in Figure 4. In this figure, we see that the observations made based on ERT from Figure 3 seem to hold across scenarios: the SPSA algorithm is clearly performing poorly, while Random Search seems to be competitive with, if not superior to, all other algorithms for every scenario. In addition to the ranking between algorithms, we also note a clear increase in problem difficulty as the dimensionality increases. Conversely, the number of samples seems to have a rather limited impact on the relative difficulty. This suggests that the structure of the point set has little influence on the performance of the optimizers.

As a final comparison, we can consider the differences in the difficulty of the optimization problem when different samplers are used. As we observed in Figure 2, the landscape is clearly impacted by the choice of the sampler. To see whether this also impacts the performance of the optimization algorithms, we consider the relative final discrepancy found on the grids with $n = 500$, for a few selected dimensions. The resulting distributions are visualized in Figure 5. From this figure, we can see that, while the choice of sampler often has a low impact on the performance of most algorithms, some tendencies can still be observed. In most cases, the Halton sampler seems to be the most challenging, especially in lower dimensions. However, the ordering is not fully consistent between algorithms or even between dimensions.

6 CONCLUSIONS

We have studied the efficiency of numerical black-box optimization approaches for maximizing the local L_∞ star discrepancy values for a given point set P. The results are underwhelming; the obtained results cannot be used as reliable estimates for the overall L_∞ star discrepancy of P.

The results indicate that off-the-shelf black-box optimization approaches have difficulties coping with the multi-modal nature of the problem and/or with the discontinuities in the genotype-phenotype mapping. We believe that this combination makes the problem an interesting use case for comparing diversity mechanisms with or without restarts. In particular, we expect that approaches such as *quality-diversity* (originally introduced in [33, 34], but see [4] for a more recent survey) or *niching* [37] could improve the quality of the search algorithms.

The focus of our work has been on the numerical black-box solvers. However, an interesting aspect of the star discrepancy computation problem is that it can also be studied as a discrete problem, using the grid structure described in Section 2. This grid structure is exploited by the TA algorithm from [18] (see Section 3.3). We suspect that merging some of the problem-specific components of this algorithm (e.g., the *snapping* rounding routines) into evolutionary algorithms operating on discrete search spaces of the type $[1..n]^d$ could be worthwhile.
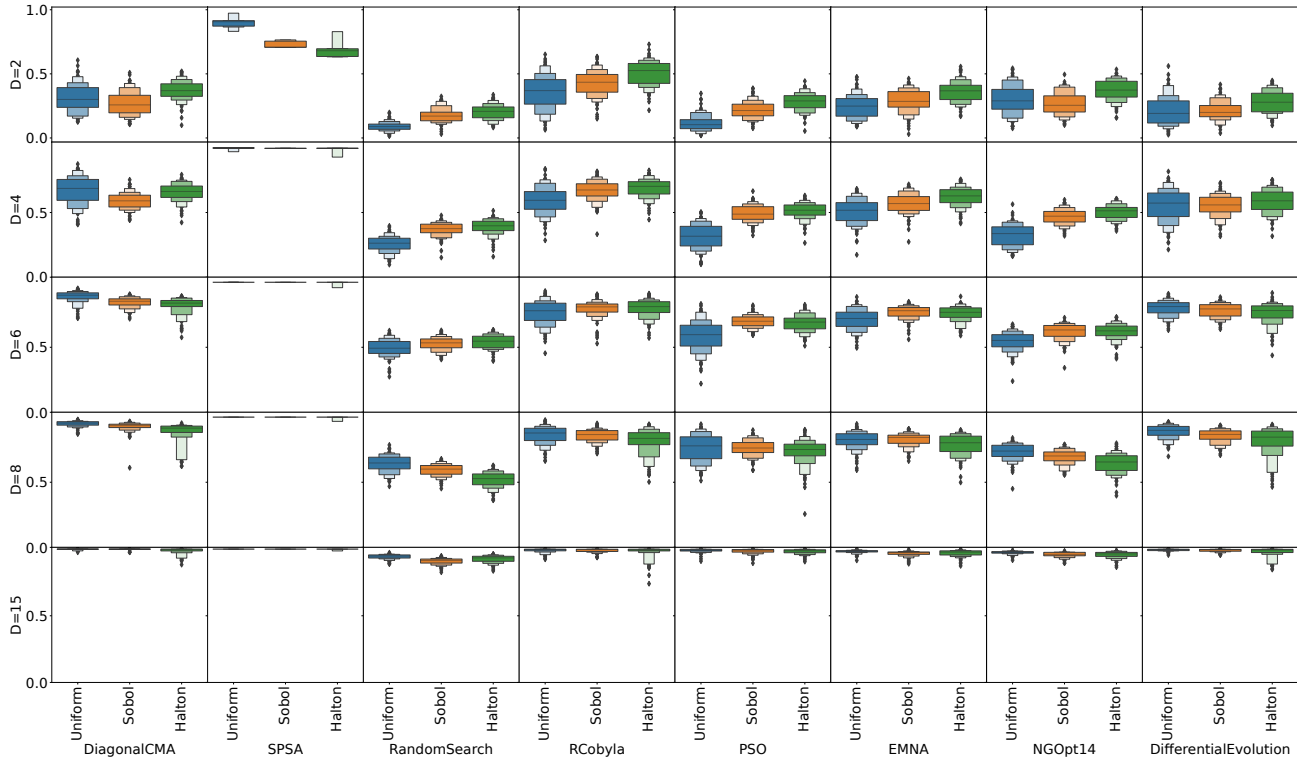
Figure 5: Relative final discrepancy value found by each of the used optimizers, compared between different samplers. Values of 0 correspond to finding the optimal solution, while 1 corresponds to the worst achievable value (0 discrepancy). Boxplots are aggregations of 10 runs on 10 instances, all for $n = 500$.

As we have motivated in the introduction, computing the L_∞ star discrepancy of a given point set is crucial for the design of low-discrepancy point sets, which have numerous applications in a broad range of industrial and academic problems, including evolutionary computation [2, 27]. Apart from providing a challenging configurable problem suite for benchmarking black-box optimization approaches, the design of problem-specific evolutionary strategies or similar approaches would be highly desirable. Both the discrepancy community, via a better understanding of the structure of the discrepancy function, and any application relying on numerical integration or Quasi-Monte Carlo integration would benefit greatly from effective solvers for this problem.

ACKNOWLEDGMENTS

REFERENCES

[1] E. Ayguade, N. Copty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang. 2009. The Design of OpenMP Tasks. *IEEE Transactions on Parallel and Distributed Systems* 20, 3 (March 2009), 404–418. https://doi.org/10.1109/TPDS.2008.105

[2] J. Bossek, C. Doerr, P. Kerschke, A. Neumann, and F. Neumann. 2020. Evolving Sampling Strategies for One-Shot Optimization Tasks. In *Proc. of Parallel Problem Solving from Nature (PPSN) (LNCS, Vol. 12269)*. Springer, 111–124. https://doi.org/10.1007/978-3-030-58112-1_8

[3] O. Bousquet, S. Gelly, K. Kurach, O. Teytaud, and D. Vincent. 2017. Critical Hyper-Parameters: No Random, No Cry. *CoRR* abs/1706.03200 (2017). arXiv:1706.03200 http://arxiv.org/abs/1706.03200

[4] K. I. Chatzilygeroudis, A. Cully, V. Vassiliades, and J.-B. Mouret. 2020. Quality-Diversity Optimization: a Novel Branch of Stochastic Optimization. *CoRR* abs/2012.04322 (2020). arXiv:2012.04322 https://arxiv.org/abs/2012.04322

[5] F. Clément, C. Doerr, and L. Paquete. 2022. Star Discrepancy Subset Selection: Problem Formulation and Efficient Approaches for Low Dimensions. *Journal of Complexity* 70 (2022), 101645. https://doi.org/10.1016/j.jco.2022.101645

[6] F. Clément, D. Vermetten, J. de Nobel, A. D. Jesus, L. Paquete, and C. Doerr. 2023. Reproducibility files and additional figures. https://doi.org/10.5281/zenodo.7630260.

[7] J. de Nobel, F. Ye, D. Vermetten, H. Wang, C. Doerr, and T. Bäck. 2021. IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics. *CoRR* abs/2111.04077 (2021). arXiv:2111.04077 https://arxiv.org/abs/2111.04077

[8] J. Dick and F. Pillichshammer. 2010. *Digital Nets and Sequences*. Cambridge University Press.

[9] D.P. Dobkin, D. Eppstein, and D.P. Mitchell. 1996. Computing the Discrepancy with Applications to Supersampling Patterns. *ACM Trans. Graph.* 15 (1996), 354–376.

[10] C. Doerr, M. Gnewuch, and M. Wahlström. 2014. Calculation of Discrepancy Measures and Applications. In *A Panorama of Discrepancy Theory*, W. Chen, A. Srivastav, and G. Travaglini (Eds.). Springer, 621–678. https://doi.org/10.1007/978-3-319-04696-9_10

[11] C. Doerr and F.-M. De Rainville. 2013. Constructing Low Star Discrepancy Point Sets with Genetic Algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 789–796. https://doi.org/10.1145/2463372.2463469

[12] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck. 2018. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *CoRR* abs/1810.05281 (2018). http://arxiv.org/abs/1810.05281 Available at http://arxiv.org/abs/1810.05281. A more up-to-date documentation of IOHprofiler is available at https://iohprofiler.github.io/.

[13] G. Dueck and T. Scheuer. 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.* 90 (1990), 161–175.

[14] B. Fox. 1986. Algorithm 647:Implementation and Relative Efficiency of Quasirandom Sequence Generators. *ACM Trans. Math. Software* 12, 4 (1986), 362–376.

[15] S. Galanti and A. Jung. 1997. Low-discrepancy sequences: Monte-Carlo simulation of option prices. *J. Deriv* (1997), 63–83.

[16] P. Giannopoulos, C. Knauer, M. Wahlström, and D. Werner. 2012. Hardness of discrepancy computation and ε-net verification in high dimension. *J. Complex.* 28 (2012), 162–176.

[17] M. Gnewuch, A. Srivastav, and C. Winzen. 2009. Finding Optimal volume subintervals with k points and calculating the star discrepancy are NP-hard problems. *J. Complex.* 25 (2009), 115–127.

[18] M. Gnewuch, M. Wahlström, and C. Winzen. 2012. A New Randomized Algorithm to Approximate the Star Discrepancy Based on Threshold Accepting. *SIAM J. Numerical Analysis* 50 (2012), 781–807. https://doi.org/10.1137/110833865

[19] J. H. Halton. 1960. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.* 2 (1960), 84–90.

[20] N. Hansen and A. Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

[21] E. Hlawka. 1961. Funktionen von beschrankter Variation in der Theorie der Gleichverteilung. *Ann. Mat. Pum Appl.* 54 (1961), 325–333.

[22] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proc. of ICNN'95 - International Conference on Neural Networks*, Vol. 4. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

[23] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220 (1983), 671–680.

[24] J. F. Koksma. 1942. Een algemeene stelling inuit de theorie der gelijkmatige verdeeling modulo 1. *Mathematica (Zutphen)* 11 (1942), 7–11.

[25] P. Larrañaga and J.A. Lozano. 2001. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Vol. 2. Springer Science & Business Media.

[26] L. Meunier, H. Rakotoarison, P. K. Wong, B. Roziere, J. Rapin, O. Teytaud, A. Moreau, and C. Doerr. 2021. Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking. *IEEE Transactions on Evolutionary Computation* 26, 3 (2021), 490–500.

[27] A. Neumann, W. Gao, C. Doerr, F. Neumann, and M. Wagner. 2018. Discrepancy-based evolutionary diversity optimization. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 991–998. https://doi.org/10.1145/3205455.3205532

[28] A. Neumann, W. Gao, M. Wagner, and F. Neumann. 2019. Evolutionary diversity optimization using multi-objective indicators. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 837–845. https://doi.org/10.1145/3321707.3321796

[29] H. Niederreiter. 1972. Discrepancy and Convex Programming. *Ann. Mat. Pura Appl.* 93 (1972), 89–97.

[30] H. Niederreiter. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611970081

[31] L. Paulin, N. Bonneel, D. Coeurjoly, J.-C. Iehl, A. Keller, and V. Ostromoukhov. 2022. MatBuilder: Mastering Sampling Uniformity over projections. *ACM Transactions on Graphics (proceedings of SIGGRAPH)* (2022).

[32] M. J. D. Powell. 1994. A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In *Advances in Optimization and Numerical Analysis*, S. Gomez and J.-P. Hennart (Eds.). Springer Netherlands, Dordrecht, 51–67. https://doi.org/10.1007/978-94-015-8330-5_4

[33] J.K. Pugh, L.B. Soros, and K.O. Stanley. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers Robotics AI* 3 (2016), 40. https://doi.org/10.3389/frobt.2016.00040

[34] J.K. Pugh, L.B. Soros, P.A. Szerlip, and K.O. Stanley. 2015. Confronting the Challenge of Quality Diversity. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 967–974. https://doi.org/10.1145/2739480.2754664

[35] J. Rapin and O. Teytaud. 2018. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad.

[36] T.J. Santner, B.J. Williams, and W.I. Notz. 2003. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics, Springer.

[37] O. Shir. 2012. Niching in Evolutionary Algorithms. In *Handbook of Natural Computing*, Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok (Eds.). Springer, 1035–1069. https://doi.org/10.1007/978-3-540-92910-9_32

[38] I.M. Sobol. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *U. S. S. R. Comput. Math. and Math. Phys.* 7, 4 (Jan. 1967), 86–112.

[39] J.C. Spall. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Automat. Control* 37, 3 (1992), 332–341.

[40] R. Storn and K. Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359. https://doi.org/10.1023/A:1008202821328

[41] H. Wang, D. Vermetten, F. Ye, C. Doerr, and T. Bäck. 2022. IOHanalyzer: Detailed Performance Analyses for Iterative Optimization Heuristics. *ACM Trans. Evol. Learn. Optim.* 2, Article 3 (2022). https://doi.org/10.1145/3510426

[42] P. Winker and K.T. Fang. 1997. Applications of Threshold-Accepting to the evaluation of the discrepancy of a set of points. *SIAM J. Numerical Analysis* 34 (1997), 2028–2042.