Faculty of Sciences and Technology

Department of Informatics Engineering

Algorithms for the Star Discrepancy Subset Selection Problem

Gonçalo Nuno Corte-Real Martins

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Luís Paquete and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering..

September 2019

1 2 9 0

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

Discrepancies quantify differences between two measurements of point sets. The star discrepancy is a particular type of discrepancy with application in areas such as statistics and pseudo-random number generation. It can be defined as the *supremum* of the absolute value of the local discrepancy for all points in the unit hypercube. The star discrepancy subset selection consists of finding the subset of k out of n points that minimizes the star discrepancy. The goal of this work is to develop branch-and-bound algorithms that are able to solve this problem for an arbitrary set of points, number of dimensions and value of k. We have developed two bounding functions that can be used to solve this problem for any number of dimensions, as well as a set of improvements to the base branch-and-bound algorithm. Our approach showed a clear increase in performance, on multiple different scenarios, when compared to a simple search algorithm that evaluates all possible subsets. This increase was most significant for point sets in a two-dimensional space.

Keywords

Star discrepancy problem, subset selection, branch and bound, combinatorial optimization

This page is intentionally left blank.

Resumo

As discrepâncias quantificam diferenças entre duas medições de conjuntos de pontos. A discrepância estrela é um tipo particular de discrepância com aplicações em áreas como a estatística e a geração de números pseudo-aleatórios. Pode ser definida como o *supremo* do valor absoluto da discrepância local para todos os pontos no hipercubo unitário. A selecção de subconjuntos baseada na discrepância estrela consiste em encontrar o subconjunto de k pontos de um conjunto de n pontos, $n \geq k$, que minimiza a discrepância estrela. O objectivo deste trabalho é desenvolver algoritmos de branch and bound que sejam capazes de resolver este problema para um conjunto arbitrário de pontos, número de dimensões e valor de k. Foram desenvolvidas duas funções de bound que podem ser usadas para resolver este problema para qualquer número de dimensões, bem como um conjunto de melhorias para o algoritmo base de branch and bound. A nossa abordagem revelou uma melhoria evidente de desempenho, em multiplos cenários diferentes, quando comparada com um algoritmo simples de pesquisa que avalia todos os possíveis subconjuntos. Esta melhoria foi mais significativa para conjuntos de pontos num espaço com duas dimensões.

Palavras-Chave

Discrepância estrela, selecção de subconjuntos, branch and bound, optimização combinatorial

This page is intentionally left blank.

Acknowledgements

I would like to start by thanking my family, in particular my mother and my brother, for all the support and motivation they gave me to finish this thesis and for always being there for me.

I would also like to thank Professor Luís Paquete for all the advice and help throughout this year of work, and Professor Carola Doerr for all the feedback and ideas shared.

Finally, I would like to thank all my friends who somehow contributed towards this work.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

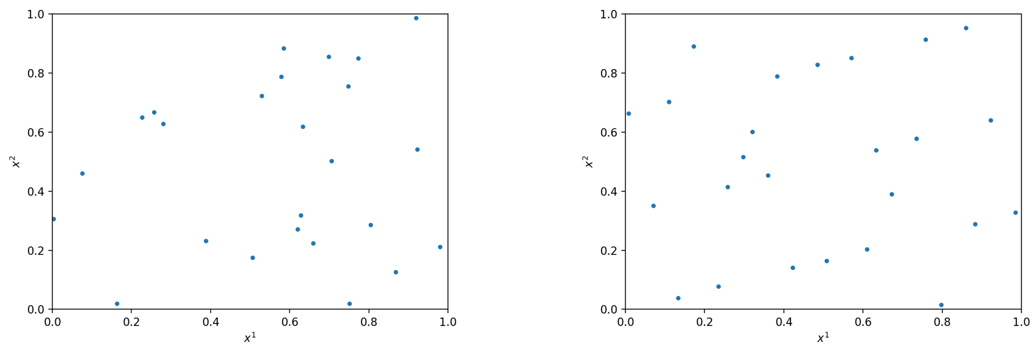
List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

The calculation of discrepancies is an important field of study in many areas, such as statistics and pseudo-random number generation [4, 7]. Discrepancies aim at measuring how much a given point set deviates from a perfectly distributed one, which can be used for assessing its quality. The star discrepancy is one of the many ways of calculating discrepancies, and it is probably the most extensively studied [4]. It can be defined as the *supremum* of the absolute value of the local discrepancy for all points in the unit hypercube. Its value ranges from 0 to 1 and the lower it is, the more diverse the point set is.



(a) Random point set (b) Sobol point set

Figure 1.1: Examples of point sets with different discrepancies

Figure 1.1 shows an example of the correlation between the coverage of the unit hypercube and the value of the star discrepancy. Both points sets are composed of 25 points, but the ones of the left were generated randomly using an uniform distribution and the ones of the right were generated using the Sobol sequence which is known to have a low discrepancy. Even though the random points were generated using an uniform distribution, it is clear that the points are not evenly distributed, having a discrepancy value of 0.226. On the other hand, the same number of points from the Sobol sequence cover the hypercube more uniformly and so they have a discrepancy of 0.121.

The calculation of the star discrepancy is a highly complex problem [6, 7]. Multiple algorithms exist and in the case of the L_∞-norm the most efficient one that works for any number of dimensions was proposed by Dobkin et al. [3]. However, because of the complexity of the star discrepancy, approaches that provide an estimation of its value have also been proposed [8, 13, 15, 16, 17].

While random point sets have a discrepancy of order $1/\sqrt{n}$, there are ways to construct point sets with a discrepancy of order $\log^{d-1}/n$, in a d-dimensional space [10]. These are called low-discrepancy point sets, to which the Halton [9] and Sobol [14] sequences are examples. Halton sequences have been generalized to avoid regularization issues in the two-dimensional projections. Efficiently finding good generators for these generalized Halton sequences is another complex problem that has been addressed heuristically [5, 12]. These low-discrepancy point sets are used in, for example, multivariate integration and computer graphics.

In this thesis we study the *star discrepancy subset selection problem*. The star discrepancy subset selection problem consists of selecting k points from a set X of n points, $n \geq k$, that minimizes the star discrepancy. This arises, for example, in the selection of solutions in the context of population based heuristics, such as evolutionary algorithms, in order to maintain diversity within the population [10, 11]. Depending on the original point set, it might be the case that the subset selected has a low discrepancy value, and so it also presents an alternative way of generating low-discrepancy point sets. Unfortunately, to the best of our knowledge, this problem has not been addressed in a systematic fashion yet, leading Neumann et al. [10] to apply ad-hoc methods in which points are removed by a simple greedy selection.

The aim of this thesis is to develop algorithms for the star discrepancy subset selection problem. In this report, we present two lower bounds for the estimation of the star discrepancy, one for a two-dimensional space and another that works in any d-dimensional space, as well as a branch-and-bound algorithm that uses this bounding functions to solve the star discrepancy subset selection problem. Additionally, we also present improvements done to our algorithm (using a sort, an upper bound initialization and the heuristic evaluation of solutions before the exact calculation of their discrepancy) and an experimental analysis using a variety of different scenarios.

The structure of this document is the following. First, in Chapter 2, we present the notation used and some concepts required to a fully understanding of this work, such as the star discrepancy, branch and bound, star discrepancy subset selection problem, and other related concepts. In Chapter 3, we review the state of the art regarding the exact calculation of the star discrepancy, with an analysis of the most efficient algorithm proposed so far, and the approximation of the star discrepancy, as well as the generation of low-discrepancy point sets. In Chapter 4, we explain the proposed branch-and-bound algorithm in a detailed way, as well as both of the bounding functions. In Chapter 5, we describe an experimental analysis that allows to characterize the performance of the branch and bound and its relation with instance features for different numbers of dimensions, present the results and compare the performance of our algorithm with a simple search algorithm. Finally, in Chapter 6, we summarize all the work done and discuss its contributions, as well as present some possible future lines of work regarding this problem.

Chapter 2

Definitions and Notation

In this chapter, the definitions and notation used throughout this document are presented. These definitions include the star discrepancy, the branch and bound, the star discrepancy subset selection problem, and all the other concepts necessary to understand them. The notation used in this thesis for the star discrepancy is similar to that of Doerr et al. [4], which discusses the existing contributions in the field of calculation of discrepancies.

2.1 Star Discrepancy

The star discrepancy is one of many different ways of calculating discrepancies of point sets. The discrepancy is related to the concept of distance, which is associated to a norm that defines how distances are calculated. Many L_p-norms, $1 \leq p < \infty$ exist, but in most cases only the L_2-norm and the L_∞-norm are considered. L_2-discrepancies can be calculated in $O(dn^2)$ time [4], where n is the number of points and d is the dimension, and not much is known about other L_p-discrepancies since not much work has been done. For this thesis only the L_∞-norm is of interest. Figure 2.1 presents an example of different norms.

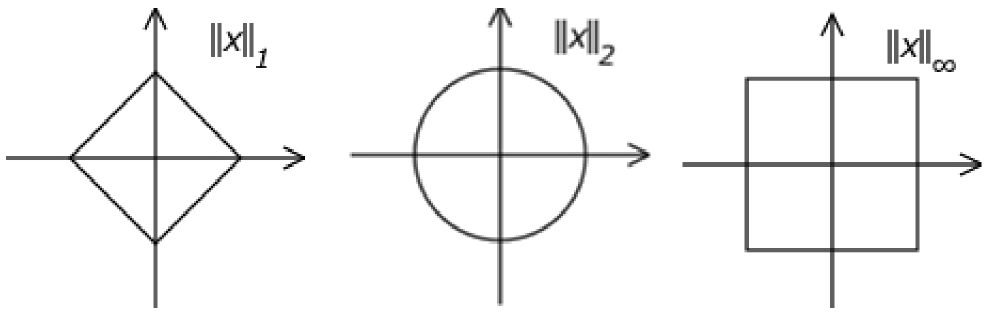


Figure 2.1: Example of different norms [1]

Before defining the star discrepancy, some necessary concepts regarding discrepancies need to be defined. Let $\mathcal{B}$ be a σ-algrebra on a set X and let μ and ν be two different measures for X. Then the local discrepancy Δ measures the difference between the two measures. To calculate the local discrepancy, the following general formula is used.

$$\Delta(B, \mu, \nu) = \mu(B) - \nu(B) \tag{2.1}$$

where $\Delta(B, \mu, \nu)$ is the local discrepancy of measures μ and ν, $B \in \mathcal{B}$. The L_∞-discrepancy of a set, which is defined as the *supremum* of the absolute value of the local discrepancy of all test sets inside that set, can then be defined as

$$disc_{\infty}(\mathcal{B}, \mu, \nu) = \sup_{B \in \mathcal{B}} |\Delta(B, \mu, \nu)|.$$

To understand the star discrepancy it is also necessary to mention the Lebesgue measure λ_d, or the hypervolume indicator. Let us assume without loss of generality that $y \in R^d_{\geq 0}$. Then, the hypervolume is the volume of the box anchored at the origin and delimited by y from above. The value of the hypervolume is equal to the volume of the area dominated by point y in the positive octant. Figure 2.2 illustrates the hypervolume in a two-dimensional space marked in gray.

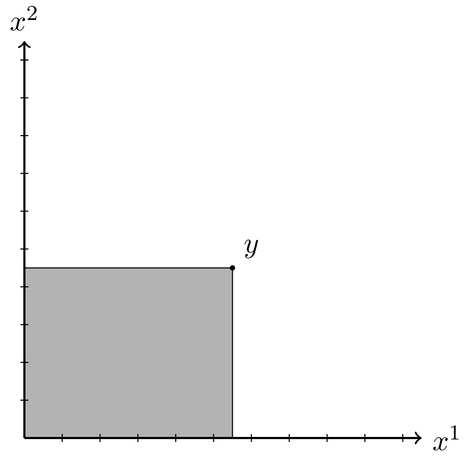


Figure 2.2: Example of the hypervolume of a point y

We can now define how to calculate the star discrepancy of a point set $X = \{x_1, ..., x_n\}$ in $[0, 1)^d$. Let Σ be the σ-algebra of Borel sets of $M = [0, 1]^d, d \in [1, \infty)$ and $\mathcal{C}_d$ is the class of boxes anchored in zero such that $[0, y) = [0, y^1) \times ... \times [0, y^d), y \in [0, 1]^d$, that is, the class of all boxes anchored in zero and limited by all the points in the unit hypercube from above. Also, let μ be the d-dimensional hypervolume λ^d on $[0, 1]^d$ and v_X is the counting measure that counts the number of points from X inside a given Borel set C. We can now obtain the L_{∞}-star discrepancy of X [4] as follows:

$$d^*_{\infty}(X) = disc_{\infty}(\mathcal{C}_d, \lambda_d, v_X) = \sup_{C \in \mathcal{C}_d} |\Delta(C, \lambda_d, v_X)|,$$

where the formula of $v_X(C)$ gives the percentage of points of X inside the subset C, which is computed as follows:

$$v_X(C) = \frac{1}{n} \sum_{i=1}^{n} 1_C(x_i), \text{ for all C} \in \Sigma$$

where $1_C(x_i)$ is the indicator function that returns 1 if $x_i \in C$ (if x_i is contained in the box $[0, y)$ of C) and 0 otherwise.

The star discrepancy is the *supremum* of the absolute value of the local discrepancy for all points inside the unit hypercube, with the local discrepancy being calculated using the hypervolume and the percentage of points inside the box anchored at zero and delimited by each point.

Even though these formulas appear to be quite complex, they can be simplified [4]. Let $y = (y^1, ..., y^d)$ be an arbitrary point in the unit hypercube $[0, 1]^d$ and $X = \{x_1, ..., x_n\}$ be a point set such that $x_i \in [0, 1)^d, i \in \{1, ..., n\}$. Let us also consider the formulas

$$V_y = \prod_{i=1}^{d} y^i \qquad A(y,X) = \sum_{i=1}^{n} 1_{[0,y)}(x_i) \qquad \overline{A}(y,X) = \sum_{i=1}^{n} 1_{[0,y]}(x_i),$$

where V_y is the hypervolume of the box $[0,y)$ and $A(y,X)$ and $\overline{A}(y,X)$ are the number of points inside the boxes $[0,y)$ and $[0,y]$ respectively. Using the formulas presented above, we can calculate $\delta(y,X)$ and $\overline{\delta}(y,X)$ which represent the differences between the hypervolume of the point y and the percentage of points inside the boxes $[0,y)$ and $[0,y]$, respectively, and are computed as follows:

$$\delta(y,X) = V_y - \frac{1}{n}A(y,X) \qquad \overline{\delta}(y,X) = \frac{1}{n}\overline{A}(y,X) - V_y$$

These correspond to the local discrepancy Δ function, introduced by Eq. (2.1), which calculates the difference between the Lebesgue measure and the counting function $v_X(C)$. Using the example of Figure 2.3, we can illustrate how these functions are calculated. The value of $A(y,X)$ is 2, as there are only 2 points in the box $[0,y)$, while the value of $\overline{A}(y,X)$ is 3, as there is one point in the border of the box, and the value of V_y is equal to the area marked in grey.

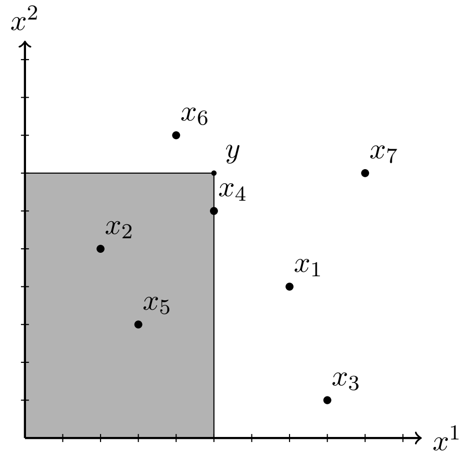


Figure 2.3: Example of the calculation of δ and $\overline{\delta}$ functions

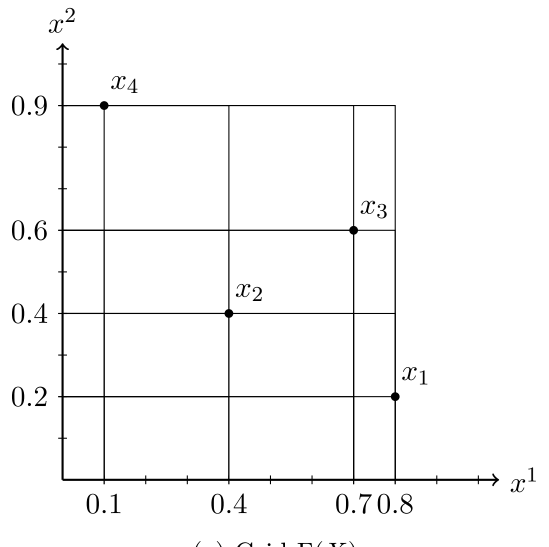
Finally, let us define the grids $\Gamma(X)$ and $\overline{\Gamma}(X)$ induced by X for $j \in \{1,...,d\}$ as follows

$$\Gamma(X) = \Gamma_1(X) \times \Gamma_2(X) \times ... \times \Gamma_d(X) \qquad \overline{\Gamma}(X) = \overline{\Gamma}_1(X) \times \overline{\Gamma}_2(X) \times ... \times \overline{\Gamma}_d(X)$$

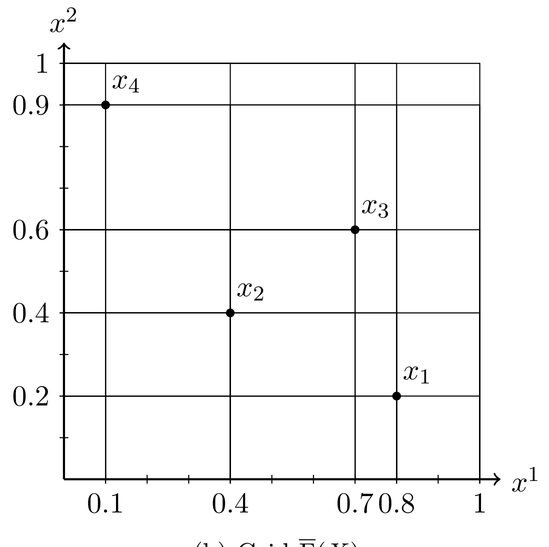
where

$$\Gamma_j(X) = \{x_i^j | i \in \{1,2,...,n\}\} \qquad \overline{\Gamma}_j(X) = \Gamma_j(X) \cup \{1\}$$

Figure 2.4 gives an example of grids generated by a point set X. Given the points in the grids $\Gamma(X)$ and $\overline{\Gamma}(X)$, it has been proven that no other point in the unit hypercube $[0,1]^d$



(a) Grid $\Gamma(X)$



(b) Grid $\overline{\Gamma}(X)$

Figure 2.4: Example of grids $\Gamma(X)$ and $\overline{\Gamma}(X)$

needs to be considered in order to calculate the star discrepancy [4]. For this reason, the star discrepancy of X can then be calculated as

$$d^*_\infty(X) = \max\{ \max_{y \in \overline{\Gamma}(X')} \delta(y, X), \max_{y \in \Gamma(X')} \overline{\delta}(y, X)\} \tag{2.2}$$

It has also been proven that calculating the star discrepancy of a point set is a NP-Hard [7] and even W[1]-Hard [6] problem. Because of this, there is no algorithm to calculate the star discrepancy for an arbitrary number of dimensions in a polynomial amount of time with respect to n. The best algorithm developed is that of Dobkin et al. [3], which has a running time of $O(n^{d/2+1})$. For a better understanding of the calculation of the star discrepancy, an example is given in appendix A.

2.2 Branch and Bound

The following description of the branch-and-bound approach is based on the work of Clausen [2]. The branch and bound is a solution approach that is commonly used to solve NP-Hard combinatorial optimization problems. It works by decomposing the original problem into smaller and smaller sub-problems and using bounds on the optimal solutions to these subproblems in order to prune the search and avoid visiting uninteresting subproblems. It starts with the entire search space as the root node of the search tree. Each time a node is expanded, new subproblems, descendants of that node, are generated by adding constraints. The way that the next node to expand is selected varies for each specific case. Every node has a bound value associated to it, which estimates the value of the best solution to that subproblem. Whenever a node has a bound value that is higher than the best value found so far, in the case of minimization problems, it is discarded without being expanded. The search ends when there are no nodes left to expand, and the solution is equal to the best value found in the search process.

This approach is defined by three components, and each of them influence its performance. The most important one is the bounding function. This function is very problem dependent and presents an estimation of the best value that can be obtained, given the current subproblem. In the case of minimization, this value must always be lower or equal to the true best value, in order to prevent possible optimal solutions to be discarded. There are two main ways of defining bounding functions. The first one is by relaxation, where some of the constraints of the problem are discarded, which implies that the number of feasible solutions increases. When a solution is found, if it satisfies all the original constraints, then it is a solution to the initial problem and a candidate to optimal solution. The other way consists of maintaining the set of feasible solutions but creating a bounding function that is easier to calculate compared to the objective function. It is also possible to combine these two approaches, in which case it is called Lagrangean relaxation. More accurate bounding functions allow a larger prune of the search tree. However, this can be too computationally expensive, and thus a trade-off between the accuracy and the time spent to calculate the bounding function must be made.

The selection of the next subproblem decides which node from the search tree to expand given its current state. The three main types of selection are best-first search, which chooses the node with the lowest bound value, breadth-first search, which considers all nodes at the same level before starting to expand the next level, and depth-first search, which chooses the node at the highest level of the search tree as the next one.

Finally, the last component is the branching rule, which tells us how new subproblems are created. The branching rule subdivides the current division of the search space, and creates new nodes in the search tree. Any current subproblem may generate a range of new subproblems from two up to any number.

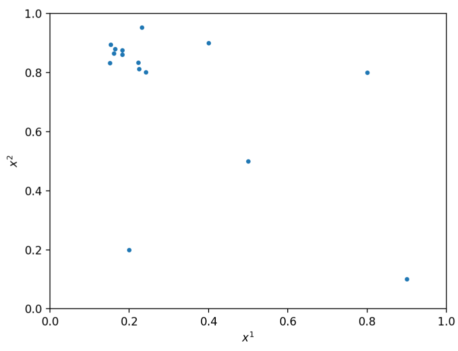
2.3 Star Discrepancy Subset Selection Problem

The star discrepancy subset selection problem can be defined in the following way: given a point set $X = \{x_1, ..., x_n\}$ in a d-dimensional space, where $x_i \in [0,1)^d$ and an integer $k \leq n$, the goal is to find a subset $X^{'} \subseteq X$ of size $|X'| = k$ such that the value of the star discrepancy of $X^{'}$, $d^*_\infty(X^{'})$, is minimized. More formally, the problem consists of:

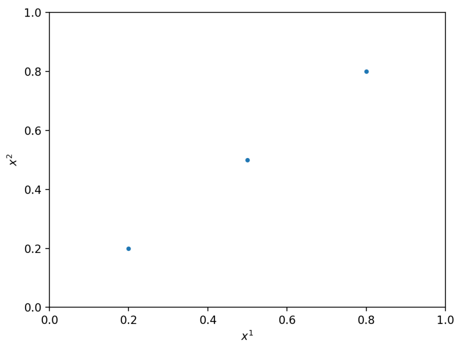
$$\min_{\substack{X' \subset X \\ |X'|=k}} \max\{ \max_{y \in \overline{\Gamma}(X')} \delta(y, X), \max_{y \in \Gamma(X')} \overline{\delta}(y, X)\}.$$

We believe that this problem is NP-Hard for an arbitrary number of dimensions because the calculation of the star discrepancy has been proven to be NP-Hard. However, no proof of this statement has been made so far. This problem can be solved by enumerating all the possible solutions of subsets of size k and then finding which one has the lowest star discrepancy value. This can be done by using a search algorithm, such as a depth-first search, to find the valid subsets of X. However, due to the complexity of the problem and the size of the search tree, a simple algorithm such as the one mentioned above will require a large amount of time as the value of n increases. A more practically efficient approach such as a branch-and-bound algorithm that does not have to consider all the nodes in the search tree is necessary.

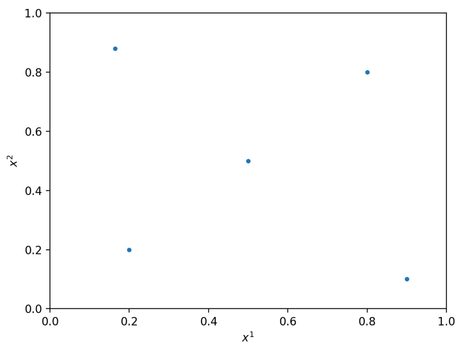
It is also important to note that selecting more points from the original point set does not necessarily translate into a lower star discrepancy value. Let us show that this is indeed true with a simple example. Figure 2.5a presents a point set X with $n = 15$, which clearly contains a cluster of points. If we select $k = 3$ points from X the result is the subset presented in Figure 2.5b which has a discrepancy value of 0.42. It is possible to achieve a lower value, by selecting $k = 5$ points. The result is shown in Figure 2.5c and has a star discrepancy value of 0.32. However, if we increase the number of points to be selected even



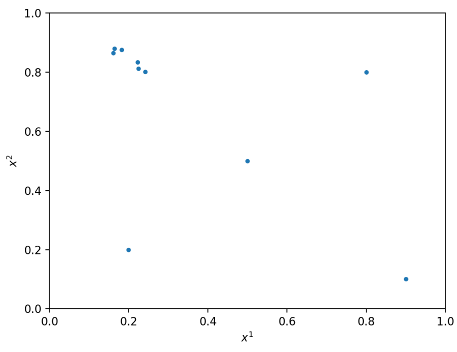
(a) Complete point set ($d^*_\infty(X) = 0.60$)



(b) Best subset for $k = 3$ ($d^*_\infty(X^{'}) = 0.42$)



(c) Best subset for $k = 5$ ($d^*_\infty(X^{'}) = 0.32$)



(d) Best subset for $k = 10$ ($d^*_\infty(X^{'}) = 0.52$)

Figure 2.5: Result of the star discrepancy subset selection problem for different values of k

more, this value increases. For $k = 10$, the discrepancy value is 0.52. From Figure 2.5d we can see that this happens because some points from the cluster were selected. It is then clear that the star discrepancy is not monotonic, that is, the direction of the value change depends on the characteristics of the point set.

This problem has applications in many different areas. As previously mentioned, it can be used in the selection of solutions in, for example, evolutionary algorithms in order to maximize diversity, as Neumann et al. [10] did. Even though they were not using the star discrepancy subset selection problem, they minimized the discrepancy of the population in their algorithm at each iteration.

The star discrepancy subset selection problem can also be used for the generation of low-discrepancy point sets from an original point set, in a different way than those methods that already exist to generate such point sets, such as the low-discrepancy sequences, which are deterministic but always have a low discrepancy. The minimum star discrepancy value that can be achieved by solving this problem depends on the original point set. It might be the case that due to poor coverage of the unit hypercube by the original point set, the subset found will still have a bad discrepancy value.

Chapter 3

State of the Art

This chapter contains an overview of the state of the art regarding the calculation of the star discrepancy, based on the work by Doerr et al. [4], which summarizes all the existent contributions to date, and the generation of low-discrepancy sequences. To the best of our knowledge there is no previous work done regarding the star discrepancy subset selection problem.

Even though it is possible to calculate the star discrepancy by performing and enumeration of all grid points, it requires a large amount of time as it is expected that most point sets generate grids of size n^d. This fact makes enumeration algorithms impractical for high values of n and/or d. There are multiple algorithms and formulas from past contributions but the most efficient one is that of Dobkin et al. [3], which has a running time of $O(n^{d/2+1})$ and calculates the exact value of the star discrepancy for any number of dimensions.

The algorithm of Dobkin et al. starts by decomposing the space. Each point x of the point set is turned into an orthant and each box $[0, y)$ into a point y such that x is in a box only if y is in its orthant. The computation of the star discrepancy can then be made by finding, for $i = 1, ..., n$, the points $y \in \overline{\Gamma}(X)$ with the highest and lowest volume V_y contained in i orthants.

The unit hypercube is divided into regions of the type $[a^1, b^1] \times ... \times [a^i, b^i] \times [0, 1]^{d-i}$ (a region at level i), and we have that $a^j = 0$ and $b^j = 1$ for $j > i$ if $i < d$. A point X is internal in dimension j, $j \in 1, \ldots, i$ to a box $[a, b]$ if it is contained in $[0, b)$ and $a^j < x^j < b^j$. Two invariants are needed:

1. All of the points in a box $[0, b)$ have at most one coordinate $1 \leq j \leq i$ such that they are internal in dimension j
2. There are $O(\sqrt{n})$ points internal in dimension $1 \leq j \leq i$ for every region at level $i > 0$

Let X_b be the set of points in $[0, b)$ and X_I be the set of points internal in some dimension $j \leq i$ for a region begin subdivided at level i. At the beginning of the algorithm the current region is $[0, 1]^d$ and all the points are in X_I.

Regions at level $i < d$ are divided into $O(\sqrt{n})$ regions at level $i + 1$, which guarantees that the total number of regions is $O(n^{d/2})$. A region $[a, b]$ at level i is divided in dimension $i + 1$ into segments of the type $[z_j, z_{j+1}]$, for $j \in \{1, ...l\}$, where l is the number of the subdivisions. By doing this, regions of the type $[a_{(j)}, b_{(j)}] = [a^1, b^1] \times [a^i, b^i] \times [z^j, z^{j+1}] \times [0, 1]^{d-i-1}$ at level $i + 1$ are generated for each $j = 1, ..., l - 1$, and each of these regions is then recursively processed and X_I and X_b are updated. The values of z_j are chosen in order for these conditions to hold:

1. For $x \in X_I$, $x^{i+1} \in \{z_1, \dots, z_l\}$
2. For $j \in \{1, ..., l-1\}$, $|\{x \in X_b : z_j < x^{i+1} < z_{j+1}\}| = O(\sqrt{n})$

Since this partition of the unit hypercube generates exactly $O(\sqrt{n})$ divisions per level, then the total number of divisions generated is $O(\sqrt{n}^d) = O(n^{d/2})$.

With the space fully partitioned, the star discrepancy can then be calculated efficiently using dynamic programming, ensuring that the complexity of the algorithm is $O(n^{d/2+1})$

Some research has also been done in the approximation of the value of the star discrepancy. Even though the focus of this thesis is solving the star discrepancy subset selection problem with the exact value of the star discrepancy these algorithms may still be of use to our approach, as will be explained in Chapter 4.

Multiple approaches exist, such as the one of Thiémard [15, 16], but since the algorithm of Gnewuch et al. provides the best results to date, it will be explained in more detail. The algorithm of Gnewuch et al. [8] is based on threshold accepting, and it is an extension of the algorithm of Winker et al. [18]. The algorithm of Winker et al. starts with a point $y \in \overline{\Gamma}(X)$ selected uniformly at random. At each iteration, up to a limit I, a new point z in the neighborhood of y is selected by changing some of the coordinates of y, chosen uniformly at random. The number of coordinates to be adjusted is not fixed, as it may be up to mc, and these coordinates are changed up to k steps in the grid $\overline{\Gamma}(X)$, chosen independently and uniformly at random. The variables mc and k are inputs of the algorithm. The point z replaces y if $d^*_\infty(z) - d^*_\infty(y) \geq T$, where $T \leq 0$. T is the threshold which is computed initially and remains constant for $\sqrt{I}$ iterations. After this, it keeps increasing so that $0 \geq T^{'} > T$ so that the algorithm converges as it ends.

Gnewuch et al. improved this algorithm, as it did not obtain very good results for $d \geq 10$. Their changes included a different neighborhood structure, as well as varying its size, and splitting the optimization of $d^*_\infty(\cdot, X)$ into $\delta(\cdot, X)$ and $\overline{\delta}(\cdot, X)$ separately. The tests performed to this algorithm showed that it outperformed other existent algorithms, and frequently found the exact star discrepancy value in case where this could be checked.

Other approaches based on integer linear programming [17] and genetic algorithms [13] are not feasible to be used in our approach as these types of algorithms are known to require a high computational time to finish their execution and thus will not be reviewed.

Regarding the generation of low-discrepancy point sets, there are some well-know sequences capable of generating points with this characteristic. Two example of these are the Sobol and the Halton sequences.
In the case of the Halton sequence, its i-th element is given by the following formula:

$$x_i = (\phi_{p_1}(i), \dots, \phi_{p_d}(i))$$

where p_j is the j-th prime number. The function $\phi_p(i)$ can be calculated by:

$$\phi_p(i) = \sum_{l=1}^{k} d_l p^{-l}$$

where p is a prime base and $i \in N$. It is also necessary to calculate the digital expansion of i in base p such that $i = d_k d_{k-1} ... d_2 d_1$, that is, $i = \sum_{l=1}^{k} d_l p^{l-1}$.

There are also some different types of algorithms that are capable of creating such point sets. An example of it is the work of De Rainville et al. [12] that proposes the use of evolutionary algorithms.

Evolutionary algorithms have a biological inspiration. They evolve a set of solutions, called the population, over a set of generations in order to obtain the best possible result. At each generation some of the individuals in the population are selected in order to generate new individuals, that join the population, using a number of recombination methods.

In the work of De Rainville et al. the outcome of their algorithm is actually a generalized Halton sequence, which is obtained from the Halton sequence using a permutation vector to shuffle its digits. The proposed algorithm evolves vectors of permutations. Vectors are iteratively constructed as optimal permutations are found, according to a fitness function. The recombination operators used were crossover, that recombines two permutations by swapping pairs of values, and mutation, that shuffles the values of the permutation. The algorithm was tested for multiple dimensions and the solutions obtained were then evaluated using the L_2-star discrepancy and the Hickernell's modified L_2-discrepancy, which is a type of discrepancy based on orthogonal projections. The results showed that this algorithm performed similar to the existent approaches.

The work of De Rainville et al. was later extended by Doerr et al. [5] to include the star discrepancy. In this case however, the authors generate low star discrepancy point sets instead of sequences and so all the permutations are optimized at the same time. Because of the complexity of calculating the star discrepancy, and unlike what was done in the previously mentioned work, the proposed algorithm uses an exact calculation of the star discrepancy using the algorithm of Dobkin et al. for $d \leq 9$. For an higher number of dimensions, due to the high running time of the algorithm of Dobkin et al., the algorithm of Gnewuch et al. is used, and the optimization is performed using values of lower bounds for the star discrepancy of the point sets. The discrepancy of the generated sequences was then compared to the discrepancy of previous work, using the exact calculation of the discrepancy in cases where this value was known for the point set to compare to or an approximation of the star discrepancy of both point sets otherwise. The generated point sets achieved a better discrepancy value in the majority of the cases.

Chapter 4

A Branch-and-bound Algorithm

This chapter gives a detailed explanation of the branch-and-bound algorithm proposed for the star discrepancy subset selection problem. Moreover, it describes further improvements as well as notions of bounding functions that were explored.

4.1 Branch-and-bound Approach

In order to solve the star discrepancy subset selection problem, we propose a branch-and-bound algorithm. We chose this approach as it is commonly used to solve combinatorial optimization problems and it guarantees that the solution found is optimal since all the possible solutions are implicitly considered in the search process.

Algorithm 1 Branch and Bound

```
  Initialization:
  $A = \{\}$; $R = \{\}$; $N = (x_1, ..., x_n)$;
  $best = \infty$;
  $gridA[i^1, i^2, \dots, i^d]$ is the maximum possible number of points in [0, y) for all $i^1, i^2, \dots, i^d = 1, \dots, n$, and for all points $y$ in grid $\overline{\Gamma}$
  $grid\overline{A}[i^1, i^2, \dots, i^d] = 0$ for all $i^1, i^2, \dots, i^d = 1, \dots, n$
**Function** $branch\_and\_bound(A, R, N)$
 1: **if** $|A| = k$ **then**
 2:     $best = minimum(best, star\_discrepancy(A))$
 3:     **return**
 4: **if** $N = \emptyset$ *or* $|A| + |N| < k$ **then**
 5:     **return**
 6: **if** $lower\_bound(A, N) > best$ **then**
 7:     **return**
 8: $point\ P = first\_point(N);$
 9: $update(grid\overline{A}, P)$
10: $branch\_and\_bound(A \cup \{P\}, R, N \setminus P)$
11: $restore(grid\overline{A}, P)$
12: $update(gridA, P)$
13: $branch\_and\_bound(A, R \cup \{P\}, N \setminus P)$
14: $restore(gridA, P)$
15: **return**
```

Algorithm 1 presents the pseudocode of our branch-and-bound algorithm, where A is the set of chosen points, R is the set of rejected points and N is the sequence of points that have not been considered yet. Both A and R are empty in the beginning, and N contains all the points. The variable *best* corresponds to the best discrepancy value found so far, and is initialized to ∞ because this is a minimization problem. Both d-dimensional matrices $gridA$ and $grid\overline{A}$ contain information about the grid points from $\overline{\Gamma}(X)$ and $\Gamma(X)$, respectively, that will be used to calculate the lower bounds, and their initialization will be explained in the next section.

The branching part of the algorithm work as follows: every point in the point set X is either accepted into the solution or rejected (Algorithm 1, lines 10 and 13 respectively), which means that every node in the search tree generates two new nodes, corresponding to the acceptance (x_i joins set A) and rejection (x_i joins set R) of the point considered. Whenever this happens, it is necessary to update the values of both grids used to calculate the lower bounds (Algorithm 1, lines 9 and 12) and then restore their former values (Algorithm 1, lines 11 and 14). This process will be further explained in the next section. Figure 4.1 shows the structure of the search tree generated by this branching strategy.

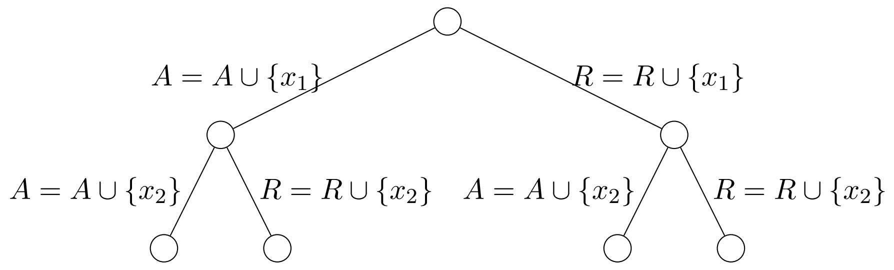


Figure 4.1: Example of a branch-and-bound search tree for this problem

However, in order to prevent the generation of infeasible solutions, such as those with less or more than k points, there must be some stopping conditions. In this case, the most basic one is to stop whenever the size of the subset is equal to k, preventing sizes larger than k. When this happens, the current solution must be valid and so its star discrepancy is calculated and the best value found is updated (Algorithm 1, lines 1, 2 and 3). The other two conditions are to stop: 1) when the last point as been considered; 2) when the number of points left to consider plus the current size of the subset is less than k, as it is not possible to achieve a valid solution at that point (Algorithm 1, lines 4 and 5). The final and more complex part of the algorithm is the bounding function. It estimates the final value of the solutions achievable at any point in the search tree, which leads to a possible pruning that occurs every time the value of the bounding function is higher than the best value found so far (Algorithm 1, lines 6 and 7). The bounding functions are explained in the following section.

4.2 Lower Bounds

While the proposed algorithm can be applied to the generic d-dimensional star discrepancy subset selection problem, it is necessary to choose the bounding function according to the number of dimensions of the point set, in order to minimize the computational time required by the algorithm to obtain the result. Since the star discrepancy subset selection problem is a minimization problem, valid bounds for pruning at a given node in the search tree must be equal or lower than the best value that is possible to achieve from that partial solution.

We have developed two bounding functions for this problem. One of them is a tighter bound for the case $d = 2$, and the other is a more generic bound that works for any value of d. Both of the bounding functions will be further explained, as well as their validity, in the following sections.

4.2.1 A Lower Bound for the Case $d = 2$

Our bounding function for $d = 2$ is composed of two parts. The first part of the lower bound, LB_1, is an under-estimation of the δ function in Eq. (2.2). The second part, LB_2, is a calculation of the $\overline{\delta}$ function in Eq. (2.2). The value of the bound is the maximum of LB_1 and LB_2. Even though this bounding function is valid and can be used for an arbitrary number of dimensions d, its calculation is computational heavy, and thus it only makes sense to use it when $d = 2$.

Calculation of LB_1

We start by noting that the value of function $\delta(y, X^{'})$ for a given point y in the grid, and given a subset $X^{'}$, is composed of two terms:

1. V_y, the volume of the box [0, y).
2. $A(y, X^{'})$, the number of points in the box [0, y).

Clearly, as points are added to set $X^{'}$, only $A(y, X^{'})$ changes as V_y remains constant throughout the optimization process. Moreover, the value of $A(y, X^{'})$ decreases as input points in the box [0, y) are not chosen by the algorithm (leave from N and enter into set R).

At each point y in the grid, we compute an overestimation $B(y, X^{'})$ on the number of points in the box $[0, y)$. The value of $B(y, X^{'})$ consists of counting the number of points in [0, y), truncated by k, that either belong to $X^{'}$ (in set A in Algorithm 1) or were not yet considered by the algorithm (in the sequence N in Algorithm 1).

Let $\gamma(y, X^{'}) = V_y - B(y, X^{'})$. Once $\gamma(y, X^{'})$ has been computed for all points y in the grid $\overline{\Gamma}(X)$, we can start the calculation of the lower bound LB_1. In order to obtain its value, we first need to calculate two other values. The first one is obtained through the points in set A. For all of these points, the maximum of $\gamma(y, X^{'})$ between the point itself and all the grid points generated through a combination of its coordinates with the point $\mathbf{1}$ or the coordinates of other points already in A, is calculated, and the maximum of all of those values is stored. Second, for the points in set N, the same is done but considering only the points themselves and the grid points generated through a combination with the point $\mathbf{1}$. However, in this case, the value that is stored is the minimum of the maximum values calculated. The value of $LB1$ is the maximum of both of these values.

Calculation of LB_2

Similar to the value of function $\delta(y, X^{'})$, the value of $\overline{\delta}(y, X^{'})$ is composed of:

1. V_y, the volume of the box [0, y].
2. $\overline{A}(y, X^{'})$, the number of points in the box [0, y].

Likewise in the previous case, only $\overline{A}(y, X^{'})$ changes (increases) as points are chosen by the algorithm. For each point y in the grid, the number of points from $X^{'}$ in the box $[0, y]$, $\overline{B}(y, X^{'})$, is maintained. It starts at zero and is updated as points are chosen by the algorithm.

Once $\overline{\gamma}(y, X^{'}) = \overline{B}(y, X^{'}) - V_y$, has been calculated for each point in the grid $\Gamma(X)$, the value of LB_2 can be obtained. The calculation of LB_2 is similar to that of LB_1, except that the grid points generated through the combination of the coordinates of points of X with the point $\mathbf{1}$ are never considered, since these points do not exist in the grid $\Gamma(X)$.

4.2.2 A Generic Lower Bound for Any Dimension

The generic bound is a modified version of the bound for $d = 2$ that can be used for an arbitrary number of dimensions d. Since the number of grid points generated from a point set X in $[0, 1]^d$, where $|X| = n$ and $d \in [2, \infty)$, is n^d, it is infeasible to use an extension of the bound for the case $d = 2$ described in the previous section for an higher number of dimensions. As the number of grid points grows exponentially in relation to the number of dimensions d, the computational time required to update all the grid values each time a point is selected or not by the algorithm is too high.

In order to minimize this effect, this bounding function does not consider the grid points generated by combining the coordinates of points from the point set. As in the previous case, this bounding function is composed of two parts, LB_1, which is an under-estimation of the δ function in Eq. (2.2) and LB_2, which is a calculation of the $\overline{\delta}$ function in Eq. (2.2). The difference between these bounding functions lies in which grid points are used in the calculation of these components.

In the case of LB_1, the points from the grid $\overline{\Gamma}(X)$ that are used in the calculation are the points from X and the ones that are generated by a combination of the coordinates of points from X with the point $\mathbf{1}$, as these grid points always exist independently of the current state of the partial solution. In the case of LB_2, and because this component is calculated using the points from grid $\Gamma(X)$, only the points from X are considered, as the points that are generated by a combination of the coordinates of points from X with the point $\mathbf{1}$ do not exist in this grid.

4.2.3 Validity of the Bounds

In order to be valid, the value of the bounding functions to a given partial solution x must be lower or equal to the optimum value that can be obtained by a solution that contains x. To prove the validity of the bound for $d = 2$ let us start by considering the first component, LB_1. For a given partial solution x with value v, the value of LB_1 is less or equal to the lowest value of any solution that contains x. This is equivalent to show that $\gamma(y, X^{'}) \leq \delta(y, X^{'})$ for all points y in the grid $\overline{\Gamma}(X)$.

Proposition 1: $\gamma(y, X^{'}) \leq \delta(y, X^{'})$ for all points y in the grid $\overline{\Gamma}(X)$

Proof: The value of $B(y, X^{'})$ is initialized as the maximum number of points in the box [0, y), for all grid points y, and thus it is an overestimation of the value of $A(y, X^{'})$. Because the value of $B(y, X^{'})$ only changes (decreases) when points are not chosen by the algorithm, then $B(y, X^{'}) \geq A(y, X^{'})$ at all times during the search process. Since the value of V_y remains constant, then it is clear that $\gamma(y, X^{'}) \leq \delta(y, X^{'})$ because $B(y, X^{'}) \geq A(y, X^{'})$.

The process of proving the validity of LB_2 is similar to that of LB_1. In order for this component to be valid, we must ensure that $\overline{\gamma}(y, X^{'}) \leq \overline{\delta}(y, X^{'})$ for all points in the grid $\Gamma(X)$.

Proposition 2: $\overline{\gamma}(y, X^{'}) \leq \overline{\delta}(y, X^{'})$ for all points in the grid $\Gamma(X)$

Proof: The value of $\overline{B}(y, X^{'})$ corresponds to the calculation of, $\overline{A}(y, X^{'})$, the number of points in the box [0, y], for all grid points y. Its value is initialized at 0 and only changes (increases) as points are chosen by the algorithm to be a part of the current solution. Because of this, it is clear that $\overline{B}(y, X^{'}) \geq \overline{A}(y, X^{'})$ throughout the search process. Then, as the value of V_y never changes, we can prove that $\overline{\gamma}(y, X^{'}) \leq \overline{\delta}(y, X^{'})$.

In the case of the generic bound, the principles behind the validity of the bound components for the grid points are the same as for the bound for $d = 2$. The difference is that, in this case, not all grid points are considered. This, however, does not make the bound invalid. By considering only some of the points we lose information about the current state of the grid and therefore the lower bound value is more relaxed, and so it is still valid.

For both of the bounding functions proposed, using the maximum value of both components does not make the bound invalid either. Both of the components are valid and can be used alone, because of Eq. (2.2). However, it is preferable to test the maximum of the two, as this makes the bounds tighter.

4.3 Other Improvements

In order to further improve the performance of the algorithm, aside from the bounding functions, multiple techniques such as sorting the points and the initialization of a upper bound were used, which are described in the following sections.

4.3.1 Upper Bound Initialization

When using a branch-and-bound algorithm it may be the case that the algorithm takes a large amount of time to reach a good upper bound that allows to prune the search tree. In order to prevent this, we evaluate multiple solutions before starting the branch-and-bound algorithm in order to start with a good upper bound value.

Each of these solutions is generated by selecting an initial point, and then selecting others iteratively until the solution has k points. At each iteration, the selected point is the one that maximizes the minimum distance to those points already in the solution. We generate n solutions, ensuring that every point in the point set will be in at least one of the evaluated solutions. By selecting the points that maximize the minimum distances we ensure the diversity of the generated solution. Since a low discrepancy value is correlated with a better coverage of the unit hypercube, we expect that the value of the star discrepancy of generated solutions is in fact a good upper bound for pruning at the start of the algorithm.

4.3.2 Sorting the Points

Since the calculation of the star discrepancy is based on the number of points in the boxes $[0, y)$ and $[0, y]$, where y is an arbitrary grid point, the order in which points are processed by the algorithm affects the tightness of the bound values, and so sorting the points could potentially lower the computational time of the algorithm.

We consider sorting the point set with respect to the layers of maxima. Note that point p dominates q if $p^i \geq q^i$ for all $i \in \{1, ..., d\}$. The problem of the layers of maxima consists on dividing the point set in layers, such that for any layer its points are not dominated by the points of the lower layers. We say that the ith layer is composed of the points that are not dominated by any other point, except by some in upper layers. Figure 4.2 shows an example of the layers obtained by solving the layers of maxima problem on the points shown.

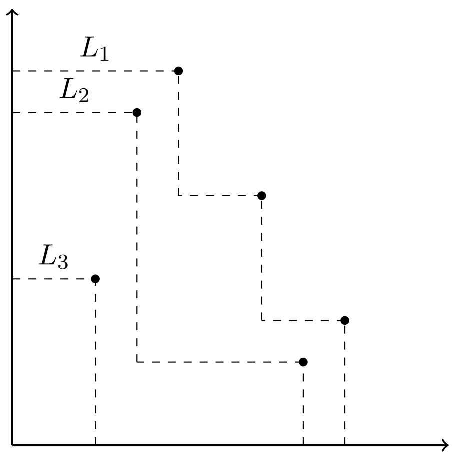


Figure 4.2: Example of the layers of maxima problem

By solving the layers of maxima problem on the point set we can then sort the points based on the layers obtained. The order of the points in N is then given by increasing order of the layers. However, the ordering of the points from each layer is kept equal to their ordering on the original point set, as the order of the points from the same layer does not interfere with their bound value, because these points do not dominate nor are dominated by each other and thus their bound values are as tight as possible.

Since the points are now sorted based on the layers, when the algorithm is considering adding a point x_i to the partial solution, all the points that it dominates have already been considered, and so its bound value is tighter than when considering the points in an arbitrary order, leading to a strong pruning of the search tree and a potential reduction of the computational time of the branch-and-bound algorithm.

4.3.3 Using Heuristics Before Evaluating Nodes

Since the complexity of the algorithm proposed by Dobkin et al. [3] to calculate the star discrepancy is $O(n^{1+d/2})$ it is expected that this calculation degrades the performance of the branch-and-bound algorithm considerably, as the number of dimensions and points increase.

One possible way to reduce the impact of this issue is to lower the number of calculations of the star discrepancy. In the branch-and-bound algorithm, whenever a terminal node is reached the solution is evaluated and the upper bound for the discrepancy, the best value found so far, is updated. Since reaching a new terminal node does not necessarily imply an improvement of the upper bound, it may not be needed to evaluate all of them.

In order to assess which nodes are worth evaluating, we propose the use of an heuristic approach. While the proposed lower bounds could be used on a solution to obtain a lower bound of the value of its star discrepancy, the bound for $d = 2$ is too slow as it considers many of the grid points, and the generic bound is not tight enough to be usable in this way, as it discards many grid points to improve its computational time. Instead, we can use heuristic methods that calculate bounds for the star discrepancy, as the ones described in Chapter 3. While these algorithms cannot be used as a bounding functions, because they calculate bounds for complete point sets, rather than partial solutions, they can be used in this case, since when a terminal node is reached, the solution is complete. Even though the algorithm of Gnewuch et al. [8] has a random factor and so the results obtained may be different for the same set of points on different runs, and they may not be close to a good result on every run, it has obtained the best results to date, and so we believe this is the best algorithm to use in this approach.

Whenever a solution is reached, before its exact evaluation is performed, an heuristic evaluation of the solution is performed to obtain a lower bound value of its star discrepancy. If the lower bound obtained is higher than the current upper bound, the best value found so far by the algorithm, then the solution reached cannot be better than the current upper bound and the calculation of its star discrepancy can be skipped, which can lead to a reduction of the computational time of the algorithm specially in a higher number of dimensions.

This page is intentionally left blank.

Chapter 5

Results

This chapter describes an experimental analysis that was conducted to characterize the performance of our branch-and-bound approach.

5.1 Experimental Setup

The tests performed to the branch-and-bound algorithm have the goal of assessing its performance over instance size, number of dimensions and different types of instances. In order to do this, multiple scenarios were defined: 2D, 3D and 4D point sets.

All of these tests were performed with a time limit of 30 minutes. The maximum values of n are 45 in the 2D case and 30 in the 3D case. With this value defined, we tested the algorithms for $k \in \{\lfloor n/4 \rfloor, \lfloor 3n/8 \rfloor, \lfloor n/2 \rfloor, \lfloor 5n/8 \rfloor, \lfloor 3n/4 \rfloor\}$ as we believe that it is more interesting to study the impact on algorithm performance of the ratio between points to choose and total number of points, rather than fixed values of k. For each combination of n and k, multiple test cases were generated and evaluated, in order to reduce variance and to understand if outliers exist. More information about the generation and content of the test cases is described in the next section.

The experimental process is as follows. We start with a simple search algorithm. This algorithm is recursive, and has the same stopping conditions as the branch-and-bound algorithm. However, it has no bounding functions, which means that the entire search tree will be covered and the star discrepancy will be calculated each time a valid solution is found, evaluating all the possible $\binom{n}{k}$ subsets. Because of the complexity of the star discrepancy, in order to reduce time spent calculating it, the algorithm used to calculate its value is the one proposed by Dobkin et al. [3], which is the most efficient algorithm.

Then, we add the bounding function to the simple search algorithm and we are left with our branch-and-bound approach. Finally, we keep adding the improvements proposed in Chapter 4 in order to achieve the best possible results.

Once an algorithm reaches a value of n where it can no longer obtain results for all values of k, no more values of n are tested. All the results for all the tests are recorded, in order to understand how each component of the final algorithm influences its overall performance.

Additionally, two more tests were also run, one with a high number of points, where $n = 500$ and $d = 2$ and another with a high number of dimensions, where $d = 20$ and $n = 25$. While it is expected that our algorithm does not finish its search process within the defined time limit, our goal with this type of test is to characterize the evolution of the upper bound throughout the search progress.

All the code necessary to run these experiments was implemented in C++ and is available at `https://github.com/gncrm/star_discr_subset`. The code used to calculate the star discrepancy was adapted to C++ from an implementation in C by Magnus Wahlström that was provided by Professor Carola Doerr. Table 5.1 contains the specifications of the computer where the tests were run.

Operating System	macOS Sierra Version 10.12.4
Processor	2,9 GHz Intel Core i5
RAM	8 GB 1867 MHz DDR3
Compiler	Apple LLVM version 8.1.0 (clang-802.0.42)
Flags	g++ -o3

Table 5.1: Computer specifications

Even though the best performance metric is the time elapsed by the search algorithm, we believe that further information is needed to understand how the bounds perform, as the computational time does not allow a complete analysis of the pruning process. Because of this, several other performance metrics were recorded. These metrics include the average depth at which prunes to the search tree occurred and which bound component had the maximum value, which helps us understand the performance of the bounding functions even further. The value of the star discrepancy of the best subset found was also recorded, in order to understand how this value is influenced by the values of n and k as well as the type of test case.

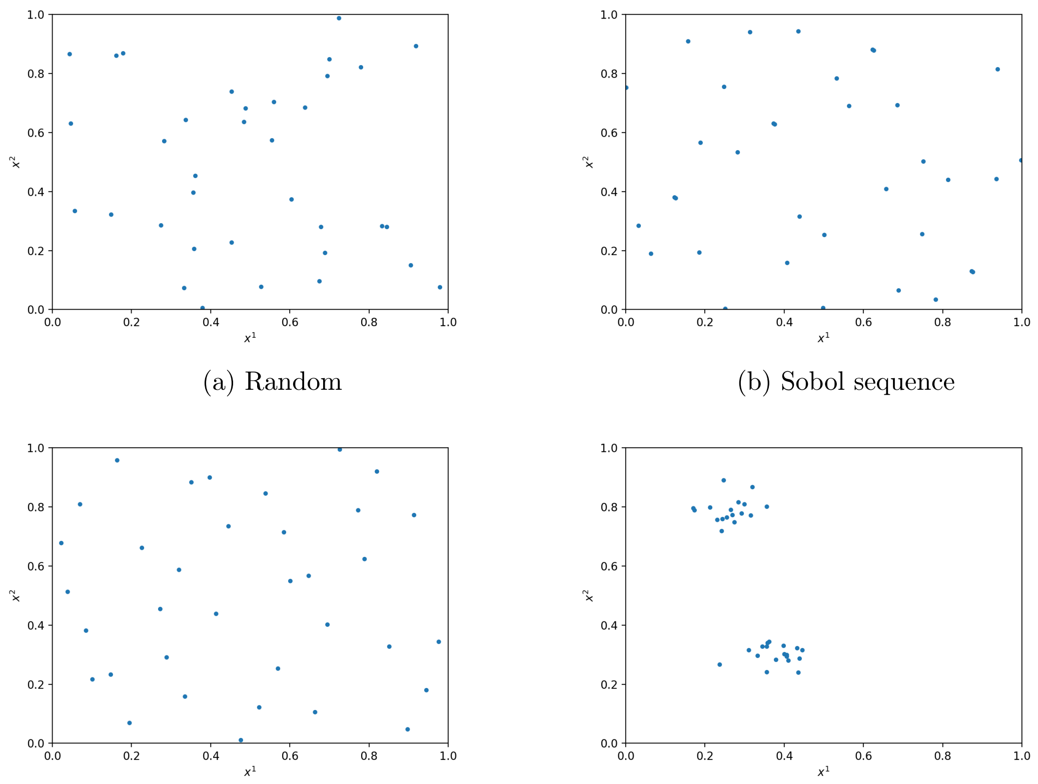
5.2 Test Cases

In order to properly assess the performance of the developed branch and bound algorithm over a multitude of different scenarios, four types of test cases were generated. In our opinion these test cases, described below, provide a good coverage of different types of data that the algorithm can encounter and have a wide range of star discrepancy values, which will allow us to show the correlation between the star discrepancy of the subsets and the star discrepancy of the point set. All of the generated points are in the $[0, 1)^d$ interval.

Random Data The points that compose the random data test cases are generated randomly from a continuous uniform distribution for each dimension. It is expected that these points cover most of the unit hypercube, especially for higher values of n. However, these point set should have a higher discrepancy value than the low discrepancy sequences as the points will not cover the unit hypercube evenly. Because of this, the algorithm should be able to significantly prune the search tree.

Low-discrepancy Data Two types of test cases were constructed using data generated from known low-discrepancy data: the Sobol and the Halton sequences. It is expected that the performance of the algorithm is worse than in the case of the random data, as the low discrepancy makes pruning the search tree harder.

Clustered Data This test case is composed of points belonging to two different clusters, generated using isotropic Gaussian distributions with the centers generated randomly and a standard deviation of 0.05. Since the points are closer together it is expected



(a) Random

(b) Sobol sequence

(c) Halton sequence

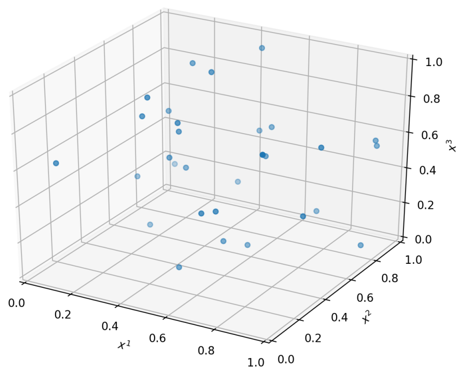
(d) Clustered

Figure 5.1: Examples of 2D test cases for $n = 35$

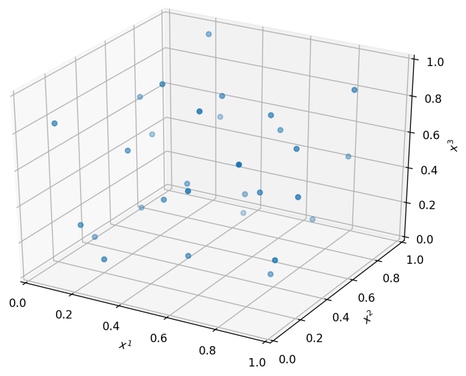
that pruning the search tree will be difficult for the algorithm and that the overall solution will have a high discrepancy value.

5.3 Experimental Results

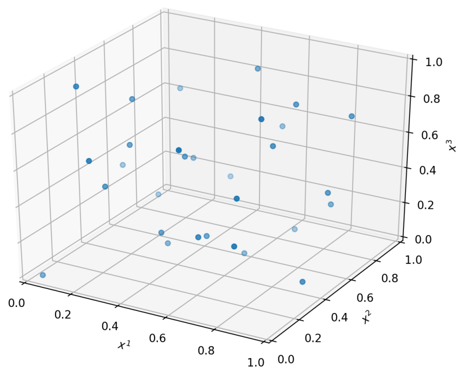
The figures presented in this report show the execution times of the algorithms in logarithmic scale, since the difference between the values of both approaches was too much for a normal scale. Whenever a value is not shown in a figure it means that for that test the result could not be obtained within the time limit of 30 minutes. The performance of the simple search algorithm, in any d-dimensional space, depends only on the values of n and k, as these are the factors that change the size of the search tree. The algorithm of Dobkin et al., which has a running time of $O(n^{d/2+1})$ only depends on n and k as well, and thus the simple search algorithm must have a similar running time for all types of test cases. Because of these, only the values of the tests for the random point sets are presented, for this algorithm. In the legends of the figures shown in this section, SSA refers to the simple search algorithm, BB refers to the branch-and-bound algorithm, BB_S refers to the branch-and-bound algorithm with the sorting of the points, and BB_SI refers to the branch-and-bound algorithm with the sorting of the points and the upper bound initialization. Appendix B contains the results displayed in tables.



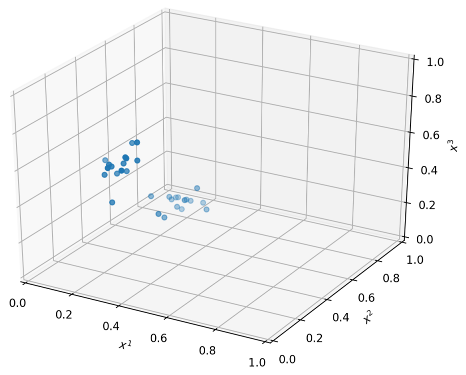
(a) Random



(b) Sobol sequence



(c) Halton sequence



(d) Clustered

Figure 5.2: Examples of 3D test cases for $n = 30$

5.3.1 2D Point Sets

In order to show the difference in performance between the simple search algorithm and the different branch-and-bound algorithms used, Figure 5.3 presents the execution times of the algorithms, for $k = n/2$ and for all instance types.

Figure 5.4 shows the optimal star discrepancy found, for each type of test case. Note that the scales are different based on the values in order to make differences in values more perceptible.

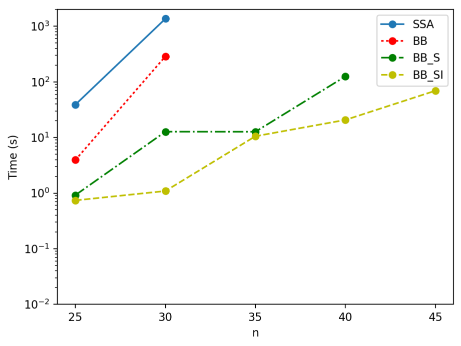
5.3.2 3D Point Sets

Figure 5.5 presents the results regarding the execution time of the algorithms, for the fixed value of $k = n/2$.

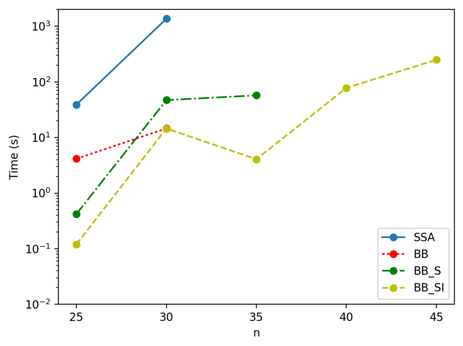
Figure 5.6 shows the star discrepancy of the best subsets. As in the previous cases the scales are different for different instance types.

5.3.3 4D Point Sets

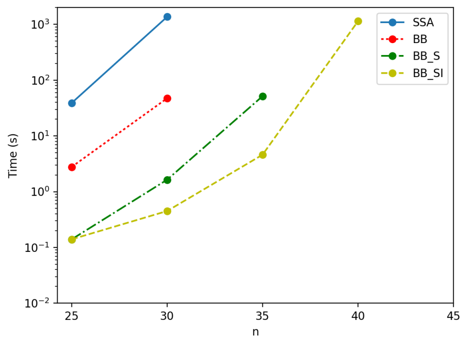
The figures regarding the tests with the 4D point sets present the same information as the figures of other scenarios. Figure 5.7 shows the execution time of the algorithms, and Figure 5.8 shows the optimal star discrepancy found.



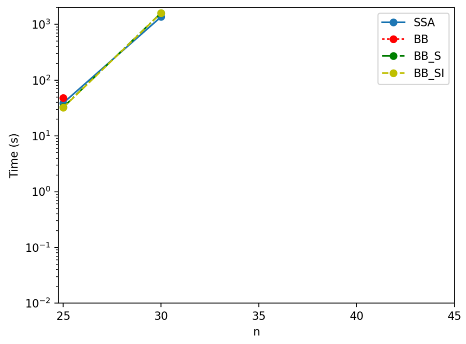
(a) Random



(b) Sobol Sequence



(c) Halton Sequence



(d) Clustered

Figure 5.3: Execution times for the 2D point sets with $k = n/2$

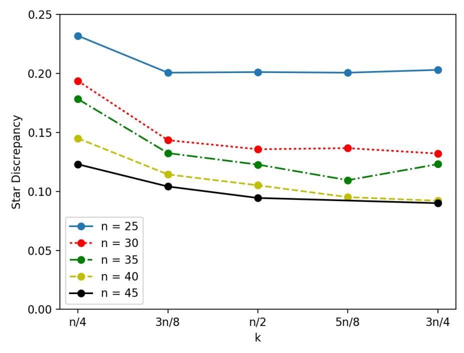
5.3.4 Special Test Cases

The tests using the special test cases where done using the branch-and-bound algorithm with the sorting of the points and the upper bound initialization for the point set where $n = 500$ and $d = 2$, and the branch-and-bound algorithm for the point set where $n = 25$ and $d = 20$. In Figure 5.9 we show the evolution of the upper bound throughout the search process for these tests, with $k = n/2$.

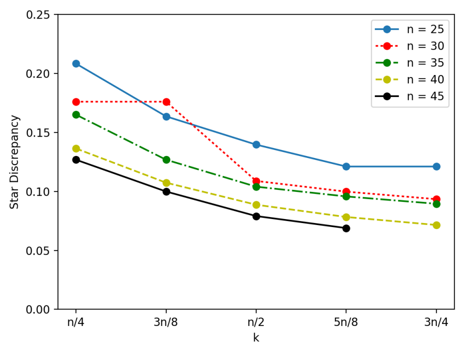
5.4 Discussion

Regarding the 2D point sets, the best results were obtained using the branch-and-bound algorithm with the sort and the upper bound initialization. In fact, both of these improvements provided a huge increase in performance which allowed results to be obtained for values of n up to 45, when the basic branch-and-bound algorithm was only able to solve the problem until $n = 35$. This increase in performance was so high that it also allowed the problem to be solved in less than one or two minutes, for cases in which other approaches tested did not get a result due to not finishing within the time limit of 30 minutes.

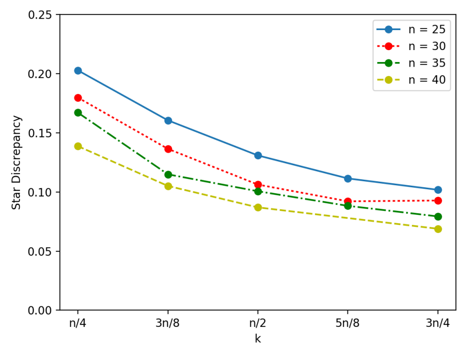
The behaviour of our algorithm was very similar in the case of the 3D and 4D point sets. The best results were achieved by the branch-and-bound algorithm with no improvements. Adding the sort did not lead to an increase in performance. This may be due to the fact that since most of the grid points are discarded, sorting the points does not have much of an effect in the tightness of the bound. The upper bound initialization also failed to fulfill



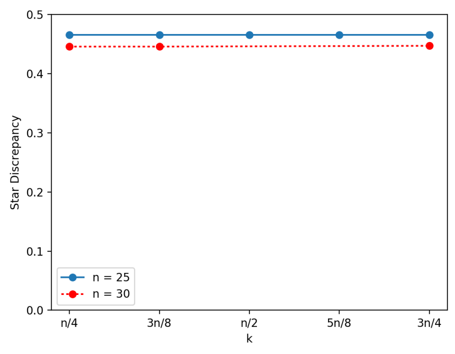
(a) Random



(b) Sobol Sequence



(c) Halton Sequence



(d) Clustered

Figure 5.4: Star discrepancy of the best subset found of 2D point sets

its purpose. The tests showed that unlike what happened when $d = 2$, the upper bound was further away from the optimal value, and so it did not help the pruning of the search tree as much as it did in the 2D case.
In these scenarios, the increase in performance was also much lower than in the 2D case, which was expected due to the higher complexity of the problem and the relaxation of the bounding function. However, the results show that in some cases, the execution time was cut in half, and it was also possible to obtain the results from tests where the simple search algorithm failed to terminate its execution.

The results also show that on all of the scenarios tested, the behavior of the algorithms depending on instance type is similar. Because of the characteristics of the random point sets, it is natural that there are a lot of point combinations that result in bad discrepancy values, which will correspond to easy prunes to the search tree. The low-discrepancy point sets were expected to require more computational time than the random point sets. Our assumption proved to be true for both the Sobol and Halton. While the tests with these point sets were faster or similar for low values of n, when the value of n increased the tests with random point sets were faster. The tests with Halton point sets were also usually slower than those with the Sobol point sets.

Our algorithms showed either a decrease or a minimal improvement in performance on the majority of the experiments using the clustered point sets, for any of the different number of dimensions tested. This is due to the low pruning of the search tree, associated with the computational cost of the bounding functions and the grid updates. This decrease in performance was most significant when $d = 2$ as the bounding function is more demanding in that case.

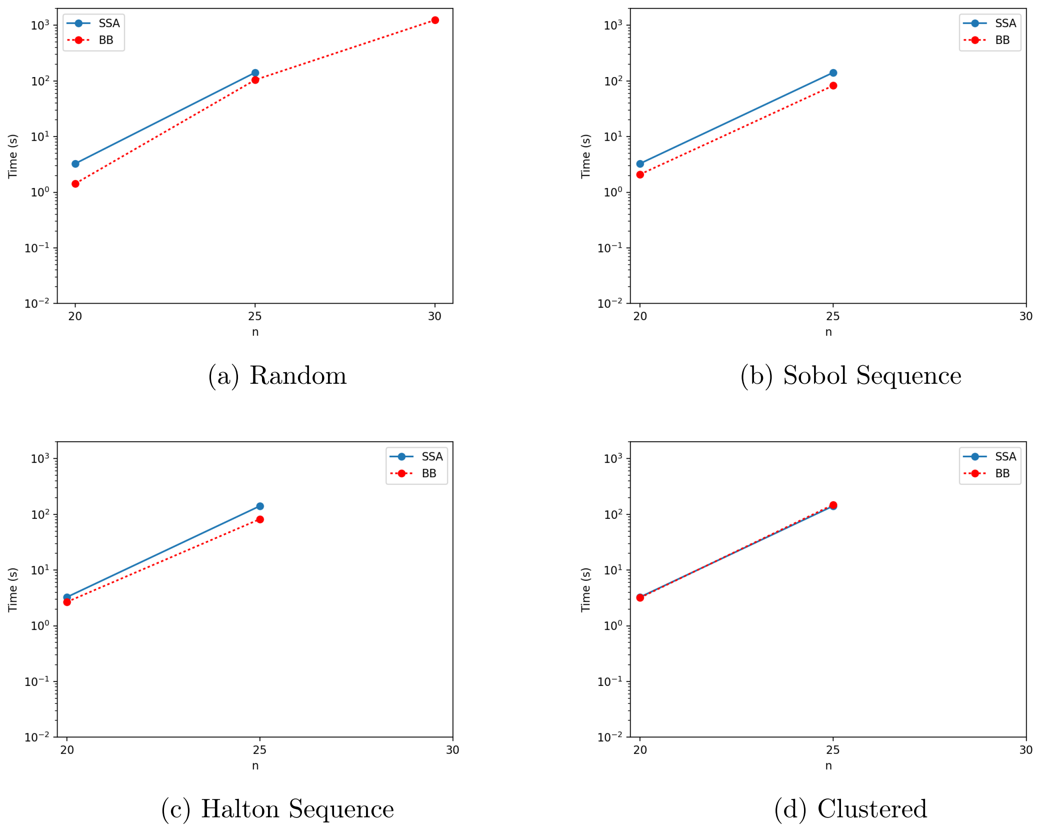
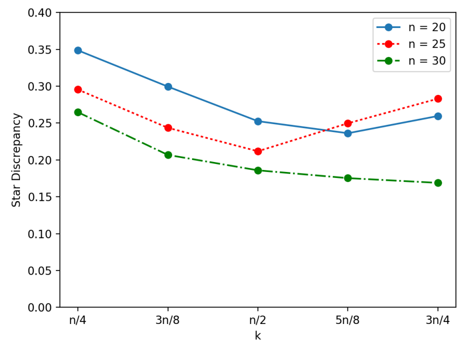


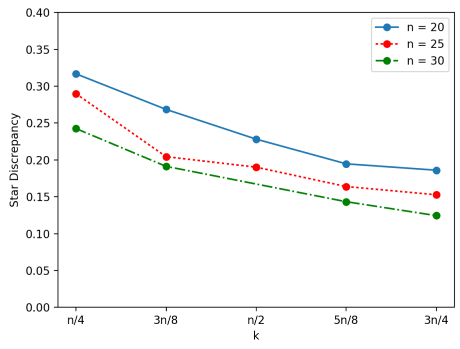
Figure 5.5: Execution times for the 3D point sets with $k = n/2$

In some of the tests, especially for $d = 2$, low values of star discrepancy were achieved. The optimal value of the star discrepancy was higher as the number of dimensions increased, which is somewhat expected, as when the number of dimensions increase more points are required to achieve a low value of discrepancy. Nevertheless, the point sets which exhibited the best values of discrepancy were the Sobol and Halton, with the random point sets becoming close as the value of n increased. This proved to be true for all of the scenarios tested. We can then conclude that using random point sets to generate low-discrepancy point sets is a good method, as our algorithm was faster on these point sets. The clustered point sets always had a high discrepancy value.
Based on the discrepancy values obtained we can also conclude that the star discrepancy of the initial point set defines how low the star discrepancy of the subsets can be, since the low-discrepancy sequences had the best subsets, and the clustered point sets obtained by far the worst results.
From the results obtained, especially from the random test cases, we can also verify what was shown in Chapter 2, since for some values of n selecting more points actually led to an increase in the value of the star discrepancy.

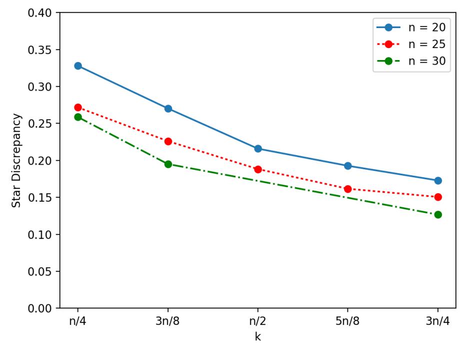
The results obtained also showed that for a given value of n and algorithm tested, the depth at which the prunes to the search tree occurred seemed to remain somewhat constant, with no significant differences, for the various values of k.
Regarding both components of the bounding functions, in the 2D scenario we could not determine which bound component actually contributed the most to the pruning of the search tree. However, in the 3D and 4D scenarios, it was clear that the first component of the bounding function, LB_1, was actually more important than LB_2 in the search process. This may have been due to the fact that LB_1 considers more grid points than LB_2, which



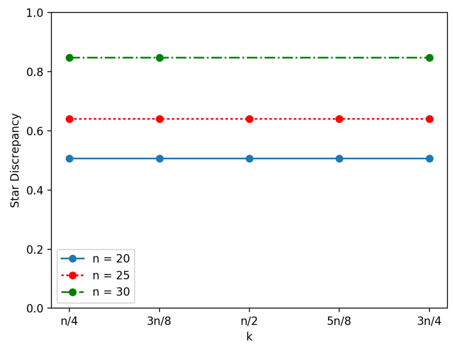
(a) Random



(b) Sobol Sequence



(c) Halton Sequence



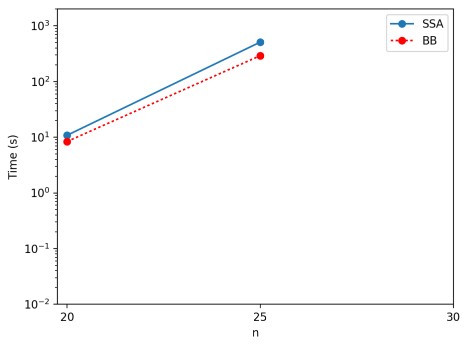
(d) Clustered

Figure 5.6: Star discrepancy of the best subset found of 3D point sets

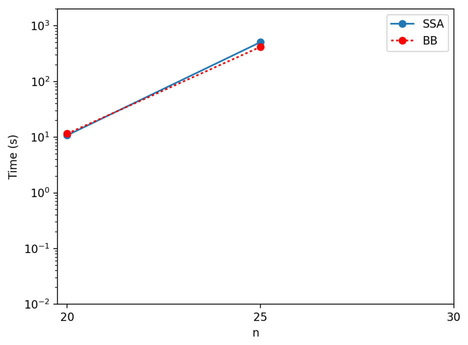
only considers the points from the original point set.

In the case of the special test cases, two tests were performed: one with 500 points generated randomly in a two-dimensional space, and another with 25 points generated in a similar way but with $d = 20$. As expected, these tests did not finish their execution within the defined time limit of 30 minutes, as both of these scenarios are quite complex for the algorithm.

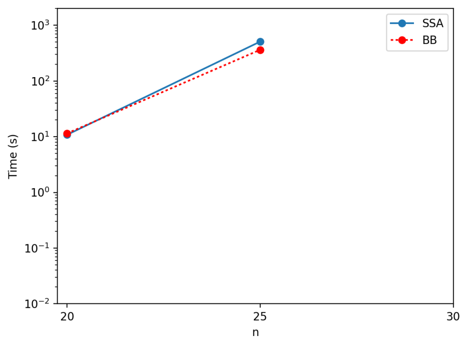
In the high n test case, in fact, the upper bound stayed almost constant for the entire run. This could have happened because the bound for $d = 2$ is computational heavy as the estimation of the grid values needs to be updated at each step of the search, and thus not many solutions were reached, or because the initial value may be close to the optimal solution.
On the other hand, the high d test case, which did not have the upper bound initialization, kept finding new best solutions as the search progressed. However, not many updates to the upper bound happened, which may have been due to the running time of the algorithm of Dobkin et al. on a high-dimensional space and consequently the consideration of a low number of solutions.



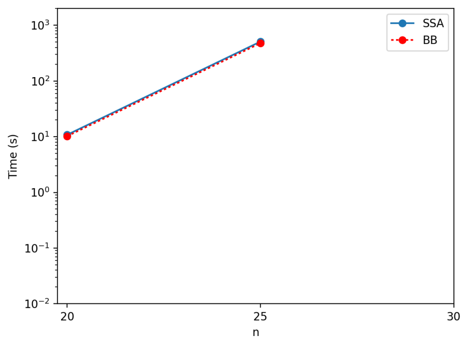
(a) Random



(b) Sobol Sequence

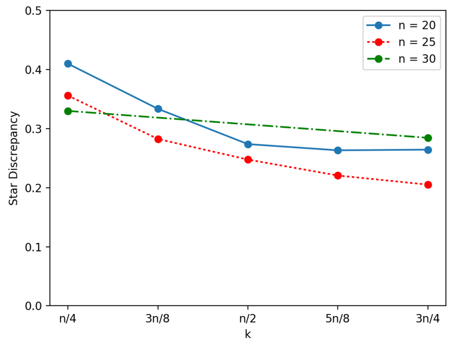


(c) Halton Sequence

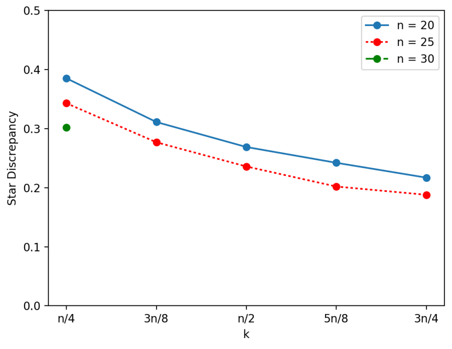


(d) Clustered

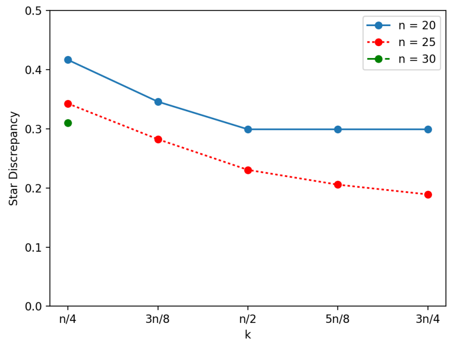
Figure 5.7: Execution times for the 4D point sets with $k = n/2$



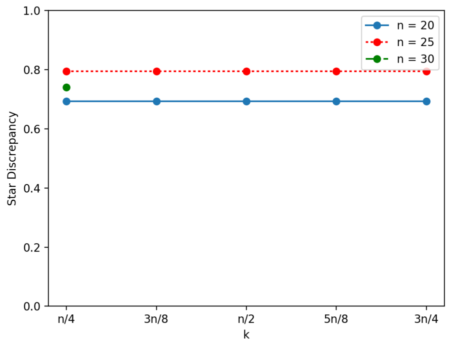
(a) Random



(b) Sobol Sequence

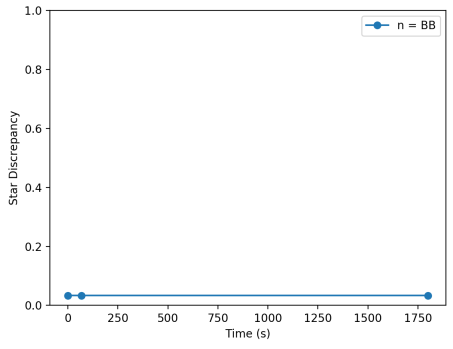


(c) Halton Sequence

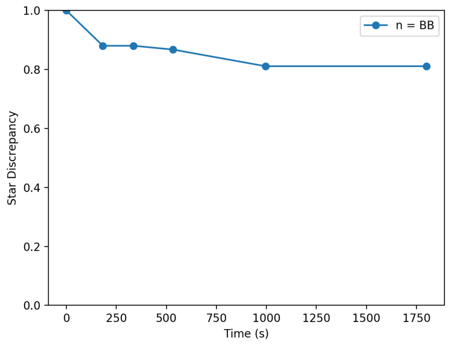


(d) Clustered

Figure 5.8: Star discrepancy of the best subset found of 4D point sets



(a) $n = 500$



(b) $d = 20$

Figure 5.9: Evolution of the upper bound

Chapter 6

Conclusion

In this thesis we propose a branch-and-bound algorithm to solve the star discrepancy subset selection problem for any number of dimensions. Our contributions are two bounding functions, one for the $d = 2$ case and another that works for any number of dimensions, as well as some improvements to the base branch-and-bound algorithm by sorting the points, calculating an upper bound to start the search with and heuristically evaluating the nodes of the search tree prior to calculating their exact discrepancy. Finally, the proposed algorithms were tested using a variety of instance types with different values for the parameters of the problem.

The results showed us that for $d = 2$ our approach presents a very considerable speedup when compared to an evaluation of all the possible solutions for the majority of the instance types. The generic bound for any number of dimensions also performed better than the simple search algorithm, although in this case the speedup factor was much lower, which is expected due to this bounding function being a relaxation of the one for $d = 2$.

Regarding future work, multiple possibilities exist. The first one, more related to the goal of this thesis, would be to find new bounding functions, primarily for $d \geq 3$, that are tighter than our proposed generic bound while also being less computational demanding than our bound for $d = 2$, in order to allow solving the star discrepancy subset selection problem for a higher number of points.
It would also be interesting to prove that the star discrepancy subset selection problem is in fact a NP-Hard problem for an arbitrary number of dimensions. While the calculation of the star discrepancy has been proven to be NP-Hard, which makes us believe that the star discrepancy subset selection problem is as well, this has not been proven yet.
Finally, regarding the calculation of the star discrepancy itself, approximation algorithms could be developed. These algorithms provide an approximation to the real value with a guarantee of how inaccurate they may be, in polynomial time. While they do not calculate the exact value of the discrepancy, they provide a good alternative in cases where an exact calculation would be too computationally demanding.

This page is intentionally left blank.

References

[1] Original image by Esmil available at `https://en.wikipedia.org/wiki/Norm_(mathematics)#/media/File:Vector_norms.svg` with license CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=678101`. The original image was adapted.

[2] Jens Clausen. Branch and bound algorithms - principles and examples, March 1999.

[3] David P. Dobkin, David Eppstein, and Don P. Mitchell. Computing the discrepancy with applications to supersampling patterns. *ACM Trans. Graph.*, 15(4):354–376, 1996.

[4] Carola Doerr, Michael Gnewuch, and Magnus Wahlström. *Calculation of Discrepancy Measures and Applications*, pages 621–678. Springer International Publishing, Cham, 2014.

[5] Carola Doerr and François-Michel De Rainville. Constructing low star discrepancy point sets with genetic algorithms. In Christian Blum and Enrique Alba, editors, *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, pages 789–796. ACM, 2013.

[6] Panos Giannopoulos, Christian Knauer, Magnus Wahlström, and Daniel Werner. Hardness of discrepancy computation and ϵ-net verification in high dimension. *J. Complexity*, 28(2):162–176, 2012.

[7] Michael Gnewuch, Anand Srivastav, and Carola Winzen. Finding optimal volume subintervals with k points and calculating the star discrepancy are NP-hard problems. *Journal of Complexity*, 25(2):115–127, 2009.

[8] Michael Gnewuch, Magnus Wahlström, and Carola Winzen. A new randomized algorithm to approximate the star discrepancy based on threshold accepting. *SIAM J. Numerical Analysis*, 50(2):781–807, 2012.

[9] J. H. Halton. Algorithm 247: Radical-Inverse Quasi-random Point Sequence. *Communications of the ACM*, 7(12):701 – 702, 1964.

[10] Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. Discrepancy-based evolutionary diversity optimization. In Hernán E. Aguirre and Keiki Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 991–998. ACM, 2018.

[11] Aneta Neumann, Wanru Gao, Markus Wagner, and Frank Neumann. Evolutionary diversity optimization using multi-objective indicators. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 837–845, 2019.

[12] François-Michel De Rainville, Christian Gagné, Olivier Teytaud, and Denis Laurendeau. Evolutionary optimization of low-discrepancy sequences. *ACM Trans. Model. Comput. Simul.*, 22(2):9:1–9:25, 2012.

[13] Manan Shah. A genetic algorithm approach to estimate lower bounds of the star discrepancy. *Monte Carlo Meth. and Appl.*, 16(3-4):379–398, 2010.

[14] I. M. Sobol'. On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86 – 112, 1967.

[15] Eric Thiémard. Computing bounds for the star discrepancy. *Computing*, 65(2):169–186, 2000.

[16] Eric Thiémard. An algorithm to compute bounds for the star discrepancy. *J. Complexity*, 17(4):850–880, 2001.

[17] Eric Thiémard. Optimal volume subintervals with k points and star discrepancy via integer programming. *Math. Meth. of OR*, 54(1):21–45, 2001.

[18] Peter Winker and Kai-Tai Fang. Application of threshold-accepting to the evaluation of the discrepancy of a set of points. *SIAM Journal on Numerical Analysis*, 34(5):2028–2042, 1997.

Appendices

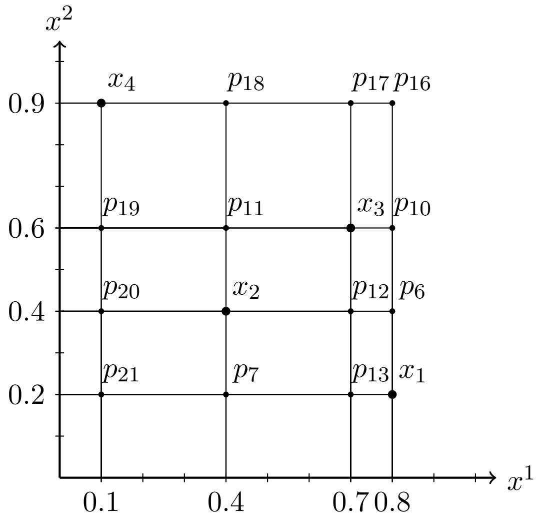
This page is intentionally left blank.

Appendix A

Example Of The Calculation Of The Star Discrepancy

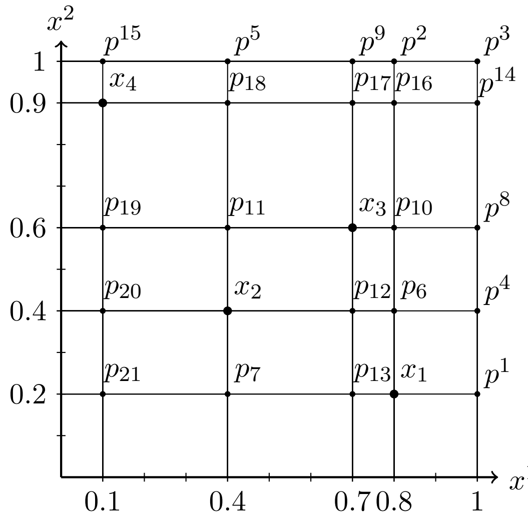
Let $X = \{x_1, x_2, x_3, x_4\}$ be a point set where $x_1 = (0.8, 0.2)$, $x_2 = (0.4, 0.4)$, $x_3 = (0.7, 0.6)$ and $x_4 = (0.1, 0.9)$. In order to calculate its star discrepancy, we need first to consider the grids induced by X, $\Gamma(X)$ and $\overline{\Gamma}(X)$.

$$\Gamma_j(X) = \{x_i^j | i \in \{1, 2, ..., n\}\} \qquad \overline{\Gamma}_j(X) = \Gamma_j(X) \cup \{1\}$$
$$\Gamma(X) = \Gamma_1(X) \times \Gamma_2(X) \times ... \times \Gamma_d(X) \qquad \overline{\Gamma}(X) = \overline{\Gamma}_1(X) \times \overline{\Gamma}_2(X) \times ... \times \overline{\Gamma}_d(X)$$



x^2
x^1
0.9
0.6
0.4
0.2
0.1
0.4
0.7
0.8
x_4 p_{18} p_{17} p_{16}
p_{19} p_{11} x_3 p_{10}
p_{20} x_2 p_{12} p_6
p_{21} p_7 p_{13} x_1

(a) Grid $\Gamma(X)$



x^2
x^1
1
0.9
0.6
0.4
0.2
0.1
0.4
0.7
0.8
1
p^{15} p^5 p^9 p^2 p^3
x_4 p_{18} p_{17} p_{16} p^{14}
p_{19} p_{11} x_3 p_{10} p^8
p_{20} x_2 p_{12} p_6 p^4
p_{21} p_7 p_{13} x_1 p^1

(b) Grid $\overline{\Gamma}(X)$

Figure A.1: Grids induced by X

Then, for each point y in $\Gamma(X)$ and $\overline{\Gamma}(X)$ we must calculate the value of $\overline{\delta}(y, X)$ and $\delta(y, X)$ respectively, using the following formulas.

$$V_y = \prod_{i=1}^{d} y^i \qquad A(y, X) = \sum_{i=1}^{n} 1_{[0,y)}(x_i) \qquad \overline{A}(y, X) = \sum_{i=1}^{n} 1_{[0,y]}(x_i)$$

$$\delta(y, X) = V_y - \frac{1}{n} A(y, X) \qquad \overline{\delta}(y, X) = \frac{1}{n} \overline{A}(y, X) - V_y$$

$$\begin{aligned}
\delta(x_1,X) &= 0.16-0=0.16\\
\delta(p_1,X) &= 0.2-0=0.2\\
\delta(p_2,X) &= 0.8-3/4=0.05\\
\delta(p_3,X) &= 1-1=0\\
\delta(x_2,X) &= 0.16-0=0.16\\
\delta(p_4,X) &= 0.4-1/4=0.15\\
\delta(p_5,X) &= 0.4-1/4=0.15\\
\delta(p_6,X) &= 0.32-0=0.32\\
\delta(p_7,X) &= 0.08-0=0.08\\
\delta(x_3,X) &= 0.42-1/4=0.17\\
\delta(p_8,X) &= 0.6-2/4=0.1\\
\delta(p_9,X) &= 0.7-2/4=0.2\\
\delta(p_{10},X) &= 0.48-1/4=0.23\\
\delta(p_{11},X) &= 0.24-0=0.24\\
\delta(p_{12},X) &= 0.28-0=0.28\\
\delta(p_{13},X) &= 0.14-0=0.14\\
\delta(x_4,X) &= 0.09-0=0.09\\
\delta(p_{14},X) &= 0.9-3/4=0.15\\
\delta(p_{15},X) &= 0.1-0=0.1\\
\delta(p_{16},X) &= 0.72-2/4=0.22\\
\delta(p_{17},X) &= 0.63-1/4=0.38\\
\delta(p_{18},X) &= 0.36-0=0.36\\
\delta(p_{19},X) &= 0.06-0=0.06\\
\delta(p_{20},X) &= 0.04-0=0.04\\
\delta(p_{21},X) &= 0.02-0=0.02
\end{aligned}$$

$$\begin{aligned}
\overline{\delta}(x_1,X) &= 1/4-0.16=0.09\\
\overline{\delta}(x_2,X) &= 1/4=0.16=0.09\\
\overline{\delta}(p_6,X) &= 2/4-0.32=0.18\\
\overline{\delta}(p_7,X) &= 0-0.08=-0.08\\
\overline{\delta}(x_3,X) &= 2/4-0.42=0.08\\
\overline{\delta}(p_{10},X) &= 3/4-0.48=0.27\\
\overline{\delta}(p_{11},X) &= 1/4-0.24=0.01\\
\overline{\delta}(p_{12},X) &= 1/4-0.28=-0.03\\
\overline{\delta}(p_{13},X) &= 0-0.14=-0.14\\
\overline{\delta}(p_{16},X) &= 1-0.72=0.28\\
\overline{\delta}(p_{17},X) &= 3/4-0.63=0.12\\
\overline{\delta}(p_{18},X) &= 2/4-0.36=0.14\\
\overline{\delta}(x_4,X) &= 0-0.09=-0.09\\
\overline{\delta}(p_{19},X) &= 0-0.06=-0.06\\
\overline{\delta}(p_{20},X) &= 0-0.04=-0.04\\
\overline{\delta}(p_{21},X) &= 0-0.02=-0.02
\end{aligned}$$

We can finally calculate the value of the star discrepancy, using the formula

$$d^*_\infty(X) = \max\{\max_{y\in\overline{\Gamma}(X)} \delta(y,X), \max_{y\in\Gamma(X)} \overline{\delta}(y,X)\},$$

which in this case is $d^*_\infty(X) = \max\{0.38, 0.28\} = 0.38$.

Appendix B

Results

This appendix contains the results of the experiments performed with our algorithm, displayed in tables. Whenever a NA is displayed in a table entry, it is because that test was not conducted. If the table entry contains a dash, it means that the test reached the time limit and thus no result was obtained. All the values of time displayed in the tables are in seconds, and SSA refers to the simple search algorithm, BB refers to the branch-and-bound algorithm, BB_S refers to the branch-and-bound algorithm with the sorting of the points, and BB_SI refers to the branch-and-bound algorithm with the sorting of the points and the upper bound initialization.

n	k	SSA	BB	BB_S	BB_SI
25	n/4	0.75	0.11	0.03	0.02
	3n/8	11.28	1.16	0.19	0.15
	n/2	38.95	3.94	0.92	0.74
	5n/8	29.47	6.50	2.52	2.01
	3n/4	5.29	2.48	1.25	0.89
30	n/4	9.41	3.88	0.16	0.08
	3n/8	383.89	73.14	3.09	0.13
	n/2	1361.76	286.62	12.71	1.09
	5n/8	914.59	362.24	20.19	3.29
	3n/4	84.54	104.21	7.16	2.49
35	n/4	109.94	25.67	0.22	0.20
	3n/8	—	1073.89	5.43	2.23
	n/2	—	—	12.67	10.53
	5n/8	—	—	42.05	40.15
	3n/4	1174.57	1038.50	45.94	44.87
40	n/4	NA	NA	0.66	0.42
	3n/8	NA	NA	13.28	0.75
	n/2	NA	NA	125.49	20.68
	5n/8	NA	NA	398.14	130.92
	3n/4	NA	NA	107.29	12.19
45	n/4	NA	NA	88.71	2.25
	3n/8	NA	NA	—	3.20
	n/2	NA	NA	—	69.24
	5n/8	NA	NA	—	—
	3n/4	NA	NA	—	1009.27

Table B.1: Execution times in seconds for the random point sets for $d = 2$

n	k	SSA	BB	BB_S	BB_S
	n/4	0.75	0.16	0.02	0.01
	3n/8	11.28	1.57	0.09	0.06
25	n/2	38.95	4.15	0.42	0.12
	5n/8	29.47	4.73	0.48	0.20
	3n/4	5.29	3.40	0.46	0.33
	n/4	9.41	1.04	0.57	0.55
	3n/8	383.89	8.45	7.78	6.32
30	n/2	1361.76	14.55	46.92	14.50
	5n/8	914.59	11.34	82.23	35.30
	3n/4	84.54	6.40	26.91	4.59
	n/4	109.94	29.87	0.25	0.20
	3n/8	—	1280.72	9.65	1.40
35	n/2	—	—	57.18	4.07
	5n/8	—	—	79.94	4.18
	3n/4	1174.57	811.90	41.50	31.93
	n/4	NA	NA	17.78	4.05
	3n/8	NA	NA	1108.58	148.85
40	n/2	NA	NA	—	77.79
	5n/8	NA	NA	—	96.55
	3n/4	NA	NA	—	35.66
	n/4	NA	NA	NA	3.47
	3n/8	NA	NA	NA	698.54
45	n/2	NA	NA	NA	249.30
	5n/8	NA	NA	NA	56.32
	3n/4	NA	NA	NA	—

Table B.2: Execution times in seconds for the Sobol point sets for $d = 2$

n	k	SSA	BB	BB_S	BB_S
25	n/4	0.75	0.20	0.03	0.03
	3n/8	11.28	1.38	0.10	0.08
	n/2	38.95	2.78	0.14	0.14
	5n/8	29.47	3.77	0.21	0.21
	3n/4	5.29	1.42	0.12	0.12
30	n/4	9.41	2.32	0.07	0.05
	3n/8	383.89	27.09	0.54	0.20
	n/2	1361.76	47.24	1.64	0.45
	5n/8	914.59	65.91	1.06	0.65
	3n/4	84.54	25.67	0.61	0.38
35	n/4	109.94	45.39	0.47	0.14
	3n/8	—	—	16.31	3.53
	n/2	—	—	51.51	4.60
	5n/8	—	—	33.45	1.17
	3n/4	1174.57	1267.00	13.53	2.99
40	n/4	NA	NA	9.74	1.75
	3n/8	NA	NA	1118.39	12.05
	n/2	NA	NA	—	1135.62
	5n/8	NA	NA	—	—
	3n/4	NA	NA	218.13	126.00
45	n/4	NA	NA	NA	NA
	3n/8	NA	NA	NA	NA
	n/2	NA	NA	NA	NA
	5n/8	NA	NA	NA	NA
	3n/4	NA	NA	NA	NA

Table B.3: Execution times in seconds for the Halton point sets for $d = 2$

n	k	SSA	BB	BB_S	BB_S
25	n/4	0.75	0.72	0.23	0.22
	3n/8	11.28	11.41	5.69	5.66
	n/2	38.95	48.22	32.72	32.31
	5n/8	29.47	52.28	42.83	42.73
	3n/4	5.29	15.22	13.57	13.56
30	n/4	9.41	11.38	7.01	6.98
	3n/8	383.89	510.15	304.02	302.79
	n/2	1361.76	—	1598.98	1590.85
	5n/8	914.59	—	—	—
	3n/4	84.54	420.28	311.22	308.52
35	n/4	109.94	NA	NA	NA
	3n/8	—	NA	NA	NA
	n/2	—	NA	NA	NA
	5n/8	—	NA	NA	NA
	3n/4	1174.57	NA	NA	NA
40	n/4	NA	NA	NA	NA
	3n/8	NA	NA	NA	NA
	n/2	NA	NA	NA	NA
	5n/8	NA	NA	NA	NA
	3n/4	NA	NA	NA	NA
45	n/4	NA	NA	NA	NA
	3n/8	NA	NA	NA	NA
	n/2	NA	NA	NA	NA
	5n/8	NA	NA	NA	NA
	3n/4	NA	NA	NA	NA

Table B.4: Execution times in seconds for the clustered point sets for $d = 2$

n	k	Random	Sobol	Halton	Clustered
25	n/4	0.231950	0.208333	0.202801	0.465798
	3n/8	0.200650	0.163628	0.160564	0.465798
	n/2	0.201177	0.139648	0.130884	0.465798
	5n/8	0.200650	0.121094	0.111435	0.465798
	3n/4	0.203028	0.121094	0.101836	0.465798
30	n/4	0.193727	0.176052	0.179808	0.445808
	3n/8	0.143432	0.136124	0.136468	0.445808
	n/2	0.135736	0.108855	0.106241	—
	5n/8	0.136671	0.099792	0.092101	—
	3n/4	0.132058	0.093422	0.092767	0.447115
35	n/4	0.178311	0.165036	0.167067	NA
	3n/8	0.132472	0.126851	0.114912	NA
	n/2	0.122723	0.104006	0.100603	NA
	5n/8	0.109532	0.095728	0.088264	NA
	3n/4	0.123093	0.089520	0.079328	NA
40	n/4	0.145022	0.136384	0.138745	NA
	3n/8	0.114394	0.107405	0.105104	NA
	n/2	0.105161	0.088698	0.086970	NA
	5n/8	0.095161	0.078300	—	NA
	3n/4	0.092041	0.071550	0.068889	NA
45	n/4	0.123000	0.126852	NA	NA
	3n/8	0.104168	0.099911	NA	NA
	n/2	0.094452	0.079102	NA	NA
	5n/8	—	0.068921	NA	NA
	3n/4	0.090038	—	NA	NA

Table B.5: Star discrepancy of the best subsets found for $d = 2$

n	k	SSA	BB
20	n/4	0.15	0.11
	3n/8	1.10	0.64
	n/2	3.27	1.43
	5n/8	3.55	1.63
	3n/4	0.54	0.54
25	n/4	1.88	1.66
	3n/8	35.69	26.31
	n/2	141.23	104.17
	5n/8	109.05	108.78
	3n/4	21.83	25.75
30	n/4	26.50	17.60
	3n/8	1480.35	529.05
	n/2	—	1238.51
	5n/8	—	883.84
	3n/4	442.17	67.36

Table B.6: Execution times in seconds for the random point sets for $d = 3$

n	k	SSA	BB
20	n/4	0.15	0.12
	3n/8	1.10	0.68
	n/2	3.27	2.09
	5n/8	3.55	2.80
	3n/4	0.54	0.44
25	n/4	1.88	1.71
	3n/8	35.69	19.73
	n/2	141.23	82.37
	5n/8	109.05	58.27
	3n/4	21.83	20.16
30	n/4	26.50	18.13
	3n/8	1480.35	637.86
	n/2	—	—
	5n/8	—	1527.30
	3n/4	442.17	205.84

Table B.7: Execution times in seconds for the Sobol point sets for $d = 3$

n	k	SSA	BB
20	n/4	0.15	0.13
	3n/8	1.10	0.81
	n/2	3.27	2.68
	5n/8	3.55	3.37
	3n/4	0.54	0.66
25	n/4	1.88	1.16
	3n/8	35.69	21.44
	n/2	141.23	82.03
	5n/8	109.05	70.82
	3n/4	21.83	22.04
30	n/4	26.50	22.59
	3n/8	1480.35	765.75
	n/2	—	—
	5n/8	—	—
	3n/4	442.17	350.70

Table B.8: Execution times in seconds for the Halton point sets for $d = 3$

n	k	SSA	BB
20	n/4	0.15	0.16
	3n/8	1.10	0.96
	n/2	3.27	3.19
	5n/8	3.55	3.14
	3n/4	0.54	0.49
25	n/4	1.88	2.23
	3n/8	35.69	38.84
	n/2	141.23	148.92
	5n/8	109.05	114.28
	3n/4	21.83	22.23
30	n/4	26.50	29.82
	3n/8	1480.35	1446.32
	n/2	—	—
	5n/8	—	—
	3n/4	442.17	381.57

Table B.9: Execution times in seconds for the clustered point sets for $d = 3$

n	k	Random	Sobol	Halton	Clustered
20	n/4	0.348644	0.316769	0.327955	0.505850
	3n/8	0.299428	0.268518	0.270287	0.505850
	n/2	0.252546	0.228159	0.215885	0.505850
	5n/8	0.236199	0.194691	0.192698	0.505850
	3n/4	0.259548	0.186040	0.172871	0.505850
25	n/4	0.295423	0.289814	0.271709	0.639772
	3n/8	0.243553	0.204414	0.226075	0.639772
	n/2	0.211757	0.190167	0.188256	0.639772
	5n/8	0.249693	0.163904	0.161648	0.639772
	3n/4	0.283027	0.152710	0.150583	0.639772
30	n/4	0.264730	0.242525	0.258618	0.847691
	3n/8	0.206781	0.191274	0.195074	0.847691
	n/2	0.185900	—	—	—
	5n/8	0.175296	0.143305	—	—
	3n/4	0.168899	0.124422	0.126667	0.847691

Table B.10: Star discrepancy of the best subsets found for $d = 3$

n	k	SSA	BB
20	n/4	0.35	0.36
	3n/8	2.88	2.38
	n/2	10.81	8.36
	5n/8	13.54	10.22
	3n/4	2.13	2.09
25	n/4	5.00	4.60
	3n/8	110.89	70.45
	n/2	509.12	290.80
	5n/8	419.35	186.80
	3n/4	92.67	50.48
30	n/4	76.28	62.66
	3n/8	—	—
	n/2	—	—
	5n/8	—	—
	3n/4	—	1623.22

Table B.11: Execution times in seconds for the random point sets for $d = 4$

n	k	SSA	BB
20	n/4	0.35	0.44
	3n/8	2.88	2.75
	n/2	10.81	11.56
	5n/8	13.54	13.56
	3n/4	2.13	2.30
25	n/4	5.00	5.28
	3n/8	110.89	96.33
	n/2	509.12	418.83
	5n/8	419.35	340.30
	3n/4	92.67	93.67
30	n/4	76.28	75.87
	3n/8	—	—
	n/2	—	—
	5n/8	—	—
	3n/4	—	—

Table B.12: Execution times in seconds for the Sobol point sets for $d = 4$

n	k	SSA	BB
20	n/4	0.35	0.42
	3n/8	2.88	3.15
	n/2	10.81	11.46
	5n/8	13.54	14.04
	3n/4	2.13	2.30
25	n/4	5.00	4.85
	3n/8	110.89	94.70
	n/2	509.12	362.90
	5n/8	419.35	341.16
	3n/4	92.67	88.40
30	n/4	76.28	76.29
	3n/8	—	—
	n/2	—	—
	5n/8	—	—
	3n/4	—	—

Table B.13: Execution times in seconds for the Halton point sets for $d = 4$

n	k	SSA	BB
20	n/4	0.35	0.39
	3n/8	2.88	2.85
	n/2	10.81	10.20
	5n/8	13.54	11.32
	3n/4	2.13	1.76
25	n/4	5.00	5.85
	3n/8	110.89	113.05
	n/2	509.12	477.32
	5n/8	419.35	386.93
	3n/4	92.67	82.42
30	n/4	76.28	74.94
	3n/8	—	—
	n/2	—	—
	5n/8	—	—
	3n/4	—	—

Table B.14: Execution times in seconds for the clustered point sets for $d = 4$

n	k	Random	Sobol	Halton	Clustered
20	n/4	0.409910	0.385208	0.416666	0.693693
	3n/8	0.333515	0.311146	0.345899	0.693693
	n/2	0.273753	0.268838	0.299178	0.693693
	5n/8	0.263272	0.242118	0.299178	0.693693
	3n/4	0.264404	0.216967	0.299178	0.693693
25	n/4	0.356021	0.342967	0.342756	0.794170
	3n/8	0.282430	0.276860	0.282417	0.794170
	n/2	0.247570	0.235857	0.230586	0.794170
	5n/8	0.220588	0.201928	0.205578	0.794170
	3n/4	0.205155	0.187934	0.189077	0.794170
30	n/4	0.329978	0.301926	0.309961	0.740540
	3n/8	—	—	—	—
	n/2	—	—	—	—
	5n/8	—	—	—	—
	3n/4	0.284420	—	—	—

Table B.15: Star discrepancy of the best subsets found for $d = 4$