TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Bachelor's Thesis in Informatics

Uniform Sampling Algorithms for Spectrahedra

Vladimir Popa

TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Bachelor's Thesis in Informatics

Uniform Sampling Algorithms for Spectrahedra

Uniforme Sampling Algorithmen für Spektrahedren

Author:	Vladimir Popa
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Adrian Kulmburg, M.Sc.
Submission Date:	15.02.2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, 15.02.2024 Vladimir Popa

Acknowledgements

I express my sincere gratitude to my advisor, Adrian Kulmburg, M.Sc., for providing invaluable supervision and direction during the development of this thesis. I am grateful for his expertise and for having had the opportunity to learn from him.

I extend my appreciation to my family, as I am thankful for their love and support. Their belief in and encouragement of my abilities have been essential for me during, not just the writing of this thesis, but also my studies.

Abstract

Set-based reachability analysis is used extensively in fields such as robotics in the context of, *e.g.*, motion planning and the formal verification of cyber-physical systems. In order to properly model such systems, various set representations have been used throughout the literature. For certain computational tasks, one needs to be able to sample random points within such set representations. This thesis will introduce two approaches, the Ball Walk Algorithm and the Billiard Walk Algorithm, which are both so-called Markov Chain Monte Carlo methods that allow one to generate random samples within a convex set. In addition, the distribution of these samples can be shown to converge to the uniform distribution. The set representation we will focus our efforts on are spectrahedral shadows, which can be seen as the solution sets of positive semi-definite constraints. As a possible application, this thesis will also present two probabilistic solutions to the containment problem for spectrahedral shadows, using uniform random sampling algorithms.

Contents

1 Introduction

1.1 Motivation

The significance of cyber-physical systems is rising in the modern world of today. Autonomous vehicles and robots, health monitoring systems and smart grids are just some examples of such systems that we may encounter on a daily basis in the future. The industrial sector is shaped by the usage of highly automated production lines. However, in order to ensure safe operation of those sometimes delicate or even dangerous systems, there is a need for methods that can verify the correct execution of cyber-physical systems. This is the main goal of reachability analysis [1], which basically boils down to the prediction of certain trajectories of a system. One particular approach is set-based reachability analysis, where the aim is not to predict a single trajectory, but rather the behaviour of an entire region or system, modeled using certain set representations.

An appropriate set representation would be spectrahedra or, more generally, spectrahedral shadows [2], which can execute most basic set operations in polynomial time. Spectrahedra, also known as *linear matrix inequality representable sets*, are the solution sets of positive semi-definite constraints. Spectrahedral shadows are the images of spectrahedra under affine or linear transformations. Many convex sets such as polytopes, zonotopes, ellipsoids and capsules can be represented as spectrahedral shadows.

Moreover, set-based reachability requires one to uniformly sample points [3][4]. This is especially useful, when simulating a trajectory of a single, random point that starts within a spectrahedral shadow. Therefore, being able to generate random points on spectrahedral shadows is important, in order to verify the correctness of reachability algorithms. Another use case presented in this thesis is the approximation of containment, which is very useful, *e.g.*, when steering a system into a target region. The two uniform random sampling methods presented, the Ball Walk and the Billiard Walk, are both Markov Chain Monte Carlo sampling algorithms [3][4]. This indicates that they eventually converge to a uniform distribution.

1.2 Contributions

This thesis provides two different approaches to uniformly sample points on spectrahedral shadows. The following algorithms have been implemented in the MATLAB toolbox CORA [5].

Sampling Algorithms

The **Ball Walk** randomly samples a point inside the ball surrounding the last sample point $\vec{p}_{i-1}$. If the newly sampled $\vec{p}_i$ is contained within the spectrahedral shadow, we add it to the list of points. This is done, until a point that is contained inside the spectrahedral shadow is found. We then proceed to the next point, starting, again, from the ball around $\vec{p}_i$.

The **Billiard Walk** randomly samples a direction $\vec{d}$ and a random segment length l, based on some parameter τ, which represents the expectation value of l. Starting from the point $\vec{p}_{i-1}$, we can trace a line segment in direction $\vec{d}$, until the target length l is reached at point $\vec{p}_i$. Should the line segment reach the boundary of the spectrahedral shadow, the computation of normal vectors at the boundary points is required for determining the direction after the reflection. Both the intersection points and the normal vectors are computed using semi-definite programs.

In addition to the uniform random sampling algorithms, two methods solving the containment problem for spectrahedral shadows will also be presented. Their relevance for this thesis lies in their usage of uniform random sampling for their approach.

Containment Algorithms

The **Inner Sampling Containment Algorithm** checks whether some spectrahedral shadow is inside another one, by uniformly sampling points in the spectrahedral shadow thought to be on the interior and then creating boundary points from these points. Containment can then be disproven if a single point of the inner spectrahedral shadow lies outside the outer one. If no such points can be found, and enough points are sampled, one can conclude with a certain probability that containment holds.

The **Outer Sampling Containment Algorithm** solves the containment problem by uniformly sampling points in the spectrahedral shadow thought to be on the exterior and, similarly to the previous approach, creating boundary points from these points. We can then compute the normal vectors at these boundary points and compare the results of the support function of these normal vectors to the result of the support function of the centre of the inner spectrahedral shadow.

1.3 Related Work

The CORA toolbox already contains an implementation of different uniform random sampling algorithms for other convex sets, such as polytopes and zonotopes, but they cannot be directly applied to spectrahedral shadows.

The R package volesti [6] also contains implementations of uniform sampling algorithms for polytopes, including the two methods presented in this thesis. Similarly to the existing CORA implementations, these methods cannot be applied directly to spectrahedral shadows.

Kulmburg and Althoff have proven that the containment problem for zonotopes is NP-hard [7], meaning that the containment problem for spectrahedral shadows is also NP-hard. Additionally, they have also presented a method for a probabilistic approach for the containment of zonotopes [8]. Methods used in this thesis are of a similar nature to the ones presented in their paper.

2 Preliminaries

2.1 Basic Notation

A letter with an arrow, *e.g.*, $\vec{v}$ denotes a vector in $\mathbb{R}^n$. The vectors $\vec{e_i}$, for $i = 1, ..., n$, will indicate the **canonical basis vectors** of $\mathbb{R}^n$. For a vector $\vec{v} \in \mathbb{R}$, we will denote v_i, for $i = 1, ..., n$, the coordinates of $\vec{v}$ with respect to the canonical basis. Matrices in $\mathbb{R}^{n \times m}$ will be represented by an uppercase letter, which is underlined, *e.g.*, $\underline{A}$. For a matrix $\underline{A} \in \mathbb{R}^{n \times m}$, we will denote $A_{i,j}$, for $i = 1, ..., n$ and $j = 1, ..., m$, the (i, j)-th coordinate of $\underline{A}$. For a vector $\vec{v} \in \mathbb{R}^n$ and $p \in [1, \infty]$, $\|v\|_p := \sqrt[p]{|v_1|^p + ... + |v_n|^p}$ is the p**-norm** of $\vec{v}$ (the ∞-norm is defined as $\|\vec{v}\|_\infty := \max_i |v_i|$). For a matrix $\underline{A} \in \mathbb{R}^{n \times n}$ the **trace** is defined as $tr(\underline{A}) = \sum_{i=1}^{n} A_{i,i}$. We define $A + B = \{a + b \mid a \in A, b \in B\}$ as the Minkowski sum of the sets A and B. The boundary of a set S is represented by ∂S.

Notation in Algorithms

We will now introduce certain functions that we shall use throughout the thesis, which have been implemented in CORA, previous to the writing of this thesis.

The expression `rand(n)` generates a random number in the interval $(0, n)$, according to the uniform distribution, whereas `randn(n,m)` generates an $n \times m$ matrix with random values according to the normal distribution. The method `interval(S)` computes the interval box over-approximation of the set S, *i.e.*, the smallest interval box that contains S. The method `center(S)` computes the centre of the set S. The procedure `feasible(S)` returns a feasible point $\vec{x}$ such that $\vec{x} \in S$. The methods `SpS.G` and `dim(SpS)` return the generator $\underline{G}$ and the dimension of the spectrahedral shadow SpS respectively, whereas the method `getCoeffMat(SpS)` returns the coefficient matrices $\underline{A}_0, \underline{A}_i$ of SpS, for $i = 1, ..., m$.

2.2 Spectrahedra and Spectrahedral Shadows

Before formally introducing spectrahedra and spectrahedral shadows, let us touch on some concepts and notation regarding positive semi-definite matrices.

We write $\underline{A} \succeq \underline{B}$ when $\underline{A} - \underline{B}$ is positive semi-definite. In particular, $\underline{A} \succeq \underline{0}$ denotes that $\underline{A}$ is positive semi-definite. Let $\underline{A}_0, ..., \underline{A}_m \in \mathbb{R}^{n \times n}$ be symmetric matrices. The linear map $\mathcal{A} : \mathbb{R}^m \to \mathbb{R}^{n \times n}$, $\vec{x} \mapsto \underline{A}_0 + \sum_{i=1}^{m} \underline{A}_i x_i$ is a **linear matrix polynomial**.

Definition 2.2.1 (Spectrahedral Shadow)**.** Let $\mathcal{A}(\vec{x})$ be a linear matrix polynomial. The set $S_{\mathcal{A}}$ is then called the **spectrahedron** defined by $\mathcal{A}$, defined as $S_{\mathcal{A}} = \{\vec{x} \in \mathbb{R}^m \mid \mathcal{A}(\vec{x}) \geq 0\}$. Given an affine map $\mathcal{P} : \mathbb{R}^{m+r} \to \mathbb{R}^m$, the set $\mathcal{P}(S_{\mathcal{A}}) = \{\mathcal{P}(\vec{x}) \mid \vec{x} \in S_{\mathcal{A}}\}$ is called a **spectrahedral shadow** [2]. This can be more precisely written as follows:

$$SpS = \left\{ \underline{G}\vec{\beta} + \vec{c} \mid \underline{A}_0 + \sum_{i=1}^{m} \beta_i \underline{A}_i \geq 0 \right\},$$

where $\underline{G} \in \mathbb{R}^{n \times m}$ is called the *generator matrix* and $\vec{c} \in \mathbb{R}^n$ is called the *centre vector* of the spectrahedral shadow SpS of dimension n.

Along spectrahedral shadows, we will also need the definition of polytopes:

Definition 2.2.2 (Polytope)**.** An n-dimensional (convex) **polytope** $P \subset \mathbb{R}^n$ can be defined as the convex hull of a finite set of points $\{\vec{v}_1, \vec{v}_2, ..., \vec{v}_N\}$ in $\mathbb{R}^n$. This is also called the *vertex representation* of P.

Additionally, for the purpose of evaluating the efficiency of our sampling methods, we will introduce capsules as well:

Definition 2.2.3 (Capsule)**.** For $\vec{c}, \vec{g} \in \mathbb{R}^n$ and some $r > 0$, let $L = \{\vec{c} + a\vec{g} \mid a \in [-1, 1]\}$ and $S = \{\vec{x} \mid \|\vec{x}\|_2 \leq r\}$. An n-dimensional **capsule** C is defined as:

$$C := L + S.$$

The vector $\vec{c}$ is known as the centre of the capsule C, $\vec{g}$ is called the generator of C, and r is its radius.

2.3 Uniform Distribution and Sampling

The main goal of this thesis is to find methods to generate random, uniformly distributed points on various sets. We characterise the uniform distribution as follows:

Definition 2.3.1 (Uniform Distribution)**.** Let S be a compact set in $\mathbb{R}^n$ and let $\vec{x} \in S$ be a random variable. Then $\vec{x}$ is said to be ϑ-uniformly distributed on S for some $\vartheta \geq 0$, if and only if $\forall A \subseteq S$:

$$\left| \mathbb{P}(\vec{x} \in A) - \frac{vol(A)}{vol(S)} \right| \leq \vartheta.$$

When referring to the uniform distribution, we mean the ϑ-uniform distribution with $\vartheta = 0$.

2.4 Duality

Duality will also be used in proofs, as such, we will need to define it here. The concept of duality is important in the context of computing normal vectors for boundary points of convex sets, but can also be used to reformulate certain optimisation problems into a more convenient form.

Definition 2.4.1 (Dual Set)**.** For a set $S \subseteq \mathbb{R}^n$ containing the origin, the **dual set** (also referred to as *polar set* in some literature) is defined as:

$$S^* := \left\{ \vec{y} \in \mathbb{R}^n \mid \sup_{\vec{x} \in S} \vec{y}^\top \vec{x} \leq 1 \right\}.$$

While the definition of the dual set can seem to be quite abstract at first, the dual set of a spectrahedral shadow has a relatively simple form. In order to deduce it, we first need the following theorem:

Theorem 2.4.1. *Let $\underline{C}$, $\underline{A}_i \in \mathbb{R}^{n \times n}$ be symmetric matrices, $\vec{b} \in \mathbb{R}^m$ be a vector. Consider the following primal minimisation problem with feasible constraints:*

$$P = \min_{\substack{tr(\underline{A}_i \underline{X}) = \vec{b} \\ \underline{X} \geq 0}} \quad tr(\underline{C}^\top \underline{X})$$

The dual maximisation problem is defined as:

$$D = \max_{\sum_i y_i \underline{A}_i \leq \underline{C}} \quad \vec{y}^\top \vec{b}$$

Then there holds $P = D$.

The prof of this theorem can be found in [9].

Theorem 2.4.2. *Let SpS be a spectrahedral shadow with generator matrix $\underline{G} \in \mathbb{R}^{n \times m}$, symmetric coefficient matrices $\underline{A}_0, ..., \underline{A}_m \in R^{k \times k}$, and centre vector $\vec{c} = \vec{0}$. Furthermore, assume that SpS contains the origin. Then the dual of SpS is given as:*

$$SpS^* := \left\{ \vec{y} \in \mathbb{R}^n \mid \exists \underline{X} \geq \underline{0} \quad s.t. \quad tr(\underline{A}_0 \vec{x}) \leq 1, \quad tr(\underline{A}_i \underline{X}) = \vec{e}_i^\top \underline{G}^\top \vec{y} \right\}.$$

Proof. Using Theorem 2.4.2, we make the following transformations:

$$\begin{aligned}
\max_{\vec{x} \in SpS} \vec{y}^\top \vec{x} &= \max_{\substack{\vec{x} = \underline{G}\vec{\beta} \\ \underline{A}_0 + \sum_i \beta_i \underline{A}_i \geq \underline{0}}} \quad \vec{y}^\top \vec{x} \\
&= \max_{\underline{A}_0 + \sum_i \beta_i \underline{A}_i \geq \underline{0}} \vec{y}^\top \underline{G} \vec{\beta} \\
&= \max_{-\sum_i \beta_i \underline{A}_i \leq \underline{A}_0} \vec{y}^\top \underline{G} \vec{\beta} \\
&= \min_{\substack{-tr(\underline{A}_i \underline{X}) = (\underline{G}^\top \vec{y})_i \\ \underline{X} \geq \underline{0}}} \quad tr(\underline{A}_0 \underline{X}) \\
&= \min_{\substack{-tr(\underline{A}_i \underline{X}) = \vec{e}_i^\top \underline{G}^\top \vec{y} \\ \underline{X} \geq \underline{0}}} \quad tr(\underline{A}_0 \underline{X})
\end{aligned}$$

This proves that the dual of SpS is:

$$SpS^* := \left\{ \vec{y} \in \mathbb{R}^n \mid \exists \underline{X} \geq \underline{0} \quad \text{s.t.} \quad tr(\underline{A}_0 \vec{x}) \leq 1, \quad tr(\underline{A}_i \underline{X}) = \vec{e}_i^{\top} \underline{G}^{\top} \vec{y} \right\}.$$

Q.E.D.

2.5 Support Function

In this thesis, we will mostly work with convex set representations. One common way to characterise convex sets is the support function, which will prove useful later on:

Definition 2.5.1 (Support Function)**.** The **support function** $h_S : \mathbb{R}^n \to \mathbb{R}$ of a non-empty closed convex set S in $\mathbb{R}^n$ defines the distance of supporting hyperplanes of S to the origin. It can be described with the formula:

$$h_S(\vec{x}) = \sup\{\vec{x}^{\top}\vec{s} \mid \vec{s} \in S\}.$$

2.6 Geometric Random Walks

Suppose we have a convex set S. A geometric random walk begins at some point $\vec{p} \in S$ and iteratively moves to a neighbouring point according to some distribution. By choosing an appropriate distribution for the one-step iteration, one ends up with a sequence of points inside the set S, that may converge towards a certain distribution, *e.g.*, the uniform distribution.

This thesis will look at two geometric random walk algorithms and their different heuristics: the Ball Walk Algorithm and the Billiard Walk Algorithm. In fact, it is proven that for an infinite amount of iterations, the Ball Walk Algorithm, in particular, converges to the uniform distribution. Moreover, it has been proven in [4] that, for a finite but large enough sample size, the resulting points are ϑ-uniformly distributed, with ϑ depending on the number of points sampled. It has also been proven that the Billiard Walk is asymptotically uniform [10].

3 Algorithms/Approach

In this chapter, the focus will lie on two sampling algorithms that were implemented in CORA for spectrahedral shadows. Proof of the uniformity of the approaches is also discussed for each individual algorithm. Section 3.2 describes the Ball Walk Algorithm and the choice of parameters for the ball generation. Section 3.3 presents an overview of the billiard walk algorithm, as well as an explanation of the computation of normal vectors, used for the reflection of lines inside the spectrahedral shadows. Finally, the accuracy and the efficiency of the two approaches are described in section 3.4.

3.1 Naive Approach

Prior to the presentation of our methods, we are going to describe an approach, that may seem easier to grasp, but that demonstrates why generating uniformly distributed points on a spectrahedral shadow is not trivial. One straightforward approach to generate uniformly distributed points on a given compact and convex set would be to over-approximate it with a surrounding interval box and uniformly sample random points on that interval, then check for each point whether it is contained in the set in question. For spectrahedral shadows, the algorithm would take the following form:

1. Approximate the SpS with the interval box I.
2. Uniformly sample points $\vec{p}_i \in I$.
3. For each point $\vec{p}_i$, check if it is in SpS. If it is, add it to a solution set of points.
4. The algorithm terminates once the desired number of points is in our solution set.

The approach is intuitive and actually gives an independent uniformly distributed sample. We will now explain why this algorithm does not work very well in general. We imagine a spectrahedral shadow. Using the naive approach on this set would generate a lot of points outside of the spectrahedral shadow itself, but inside its interval box, because the ratio between the volume of the spectrahedral shadow and the volume of the interval box is very small.

This issue becomes even worse in higher dimensions by looking at the volume of the *n-dimensional ball*, which can be represented by a spectrahedral shadow. In Figure 3.1 we can see that the volume of the unit n-ball first increases up to the dimension 7, but then quickly decreases and converges to 0. Since the interval box of the n-ball is the

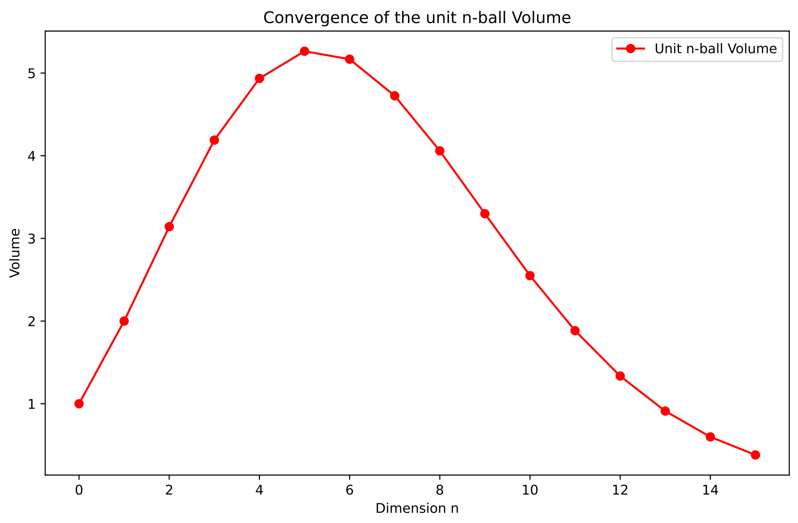


Figure 3.1: Volume of the unit n-ball across dimensions

n-cube (with volume 2^n), this means that the volume ratio between a spectrahedral shadow and its interval over-approximation can become arbitrarily small.

The major issue is thus the runtime, as the algorithm would converge very slowly. This happens, because the volume that can be sampled becomes disproportionately large compared to the target volume, meaning generated points would, more often than not, be thrown out. The advantage of this approach is the near-perfect uniformity, but the major disadvantage is the runtime, which may be at least exponential in n.

Therefore, there is the need for algorithms that can sample a convex set in polynomial time, albeit not perfectly uniformly as for the naive approach. These methods use geometric random walks, which will generate a set of ϑ-uniformly distributed points.

3.2 Ball Walk

3.2.1 Outline

The first sampling algorithm presented in this thesis is the Ball Walk Sampling Algorithm. Intuitively, the approach can be described as follows: We start with a spectrahedral shadow SpS and the number of points we would like to uniformly generate N. We will then proceed as follows:

1. Start from a randomly chosen starting point $\vec{p}_0$.
2. Generate a ball B_r around $\vec{p}_0$ with radius r.

3. Generate a random point $\vec{y}$ randomly inside B_r.
4. Check to see if $\vec{y}$ is in SpS. Should this be the case, $\vec{y}$ is added to our list of points and we set $\vec{p}_0 := \vec{y}$. If this is not the case, we go to step 3.
5. We decrease N and, if $N > 0$, we go to step 1. Otherwise, the algorithm is finished.

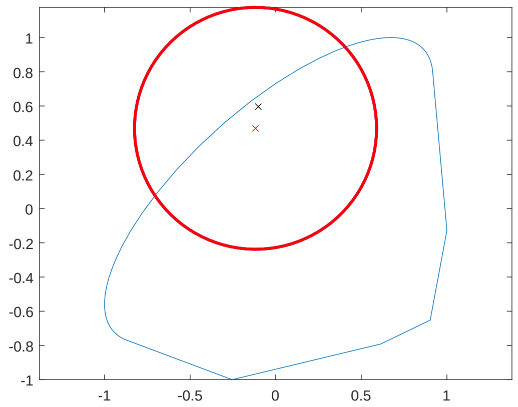
3.2.2 Algorithm

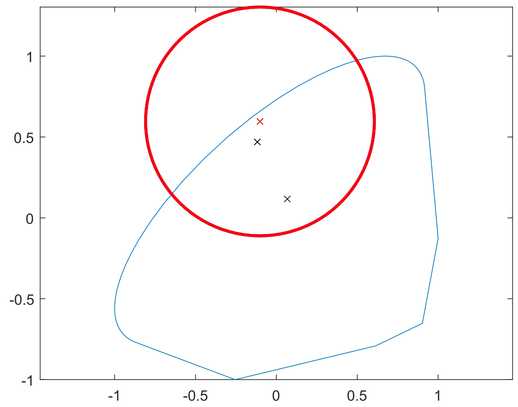
The pseudocode of the algorithm can be seen in Figure 3.2. A visual of the algorithm can be seen in Algorithm 1. A proof for the convergence to the uniform distribution can be found in [4, Theorem 3.7].

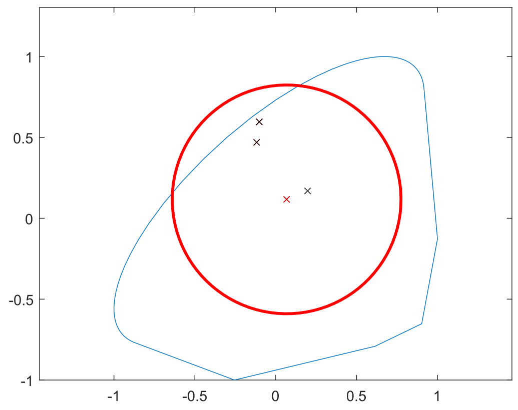
Algorithm 1 Ball Walk Algorithm

Input: a spectrahedral shadow SpS of dimension n with m generators, the number of points N to be sampled in SpS
Output: the set of sampled points

```
1: $\vec{p}_i \leftarrow \vec{0}, \quad i = 1, ..., N$
2: $I \leftarrow$ interval(SpS)
3: $\vec{c} \leftarrow$ center(I)
4: $SpS_0 \leftarrow SpS - \vec{c}$
5: $\vec{l} = (\sup I - \inf I)/2$
6: $SpS_{trans} \leftarrow diag(\vec{l}^{-1}) \cdot SpS_0$
7: $r \leftarrow 1/\sqrt{\texttt{dim(SpS)}}$
8: $\vec{p}_0 \leftarrow$ feasible(SpS)
9: $i \leftarrow N$
10: while i>0 do
11:     $\vec{\gamma} \leftarrow$ randn(dim(SpS),1)
12:     $\vec{\gamma} \leftarrow \vec{\gamma}/\|\vec{\gamma}\|_2$
13:     $d \leftarrow$ rand(1)
14:     $\vec{x} \leftarrow d^{1/\texttt{dim(SpS)}} \cdot \vec{\gamma}$
15:     $\vec{y} \leftarrow \vec{p}_0 + r \cdot \vec{x}$
16:     if $\vec{y} \in SpS_{trans}$ then
17:         $\vec{p}_{N-i+1} \leftarrow \vec{y}$
18:         $i \leftarrow i - 1$
19:         $\vec{p}_0 \leftarrow \vec{y}$
20:     end if
21: end while
22: $\vec{p}_i \leftarrow diag(\vec{l}) \cdot \vec{p}_i, \quad i = 1, ..., N$
23: $\vec{p}_i \leftarrow \vec{p}_i + \vec{c}, \quad i = 1, ..., N$
24: return $\vec{p}_i, \quad i = 1, ..., N$
```


(a)


(b)


(c)

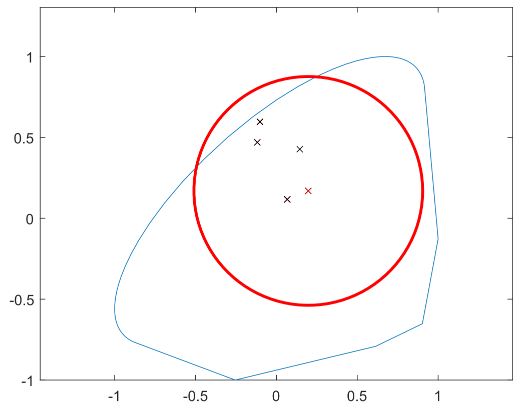

(d)

Figure 3.2: An example of the point generation inside the transformed spectrahedral shadow. In each iteration, a ball with radius r is drawn around the current point (in red), and a new point is generated within that ball. If the point is inside the spectrahedral shadow, we define it as the new current point, and add it to the list of sampled points.

Preparations

We first generate the interval box that surrounds our spectrahedral shadow, which we can accomplish with the method `interval(SpS)`. Since an interval box can be represented as $[\vec{a}, \vec{b}] := \{\vec{x} \mid a_i \leq x_i \leq b_i, \forall i\}$, to compute the side lengths $\vec{l}$ of a box it suffices to set $\vec{l} = \frac{1}{2}(\vec{b} - \vec{a})$. By subtracting the centre of the interval from SpS, we centre it around the origin. We then multiply this newly centred SpS with $diag(\vec{l}^{-1})$, the diagonal matrix of inverse of the length of the box surrounding SpS. This entire heuristic ensures better working conditions for the algorithm, since it has been shown in [4] that the Ball Walk performs particularly well if the set is shaped similarly to the unit hyperball. A typical pre-processing procedure for the Ball Walk is called "rounding", and consists in transforming the set beforehand in such a way, that it resembles a unit ball. Unfortunately, computing the ball over-approximation of a spectrahedral shadow is NP-hard, as can be seen in [11]. In table 1 it is specified that computing the radius with respect to the euclidean norm of a H-polytope is NP-hard. Spectrahedral shadows can represent H-polytopes, so it follows that the ball approximation of a spectrahedral shadow is also NP-hard. However, since a unit hyperbox is relatively similar to a hyperball, approximating spectrahedral shadows with an interval box achieves similar results. We can now generate a random starting point. In order to generate such a point, one possibility is to compute the support function of SpS in a random direction, and check which point in SpS is the maximiser of the corresponding optimisation problem. This generates a random point on the boundary of SpS, which is sufficient for our purpose.

3.3 Billiard Walk

3.3.1 Outline

The second sampling algorithm that will be introduced is the Billiard Walk Sampling Algorithm. It can be described as follows: We, once again, start with SpS and N (as described in section 3.2).

1. Start from a randomly chosen point $\vec{p}_0$.
2. Sample a random direction and a random trajectory length.
3. Follow the direction until the length is reached or the boundary of SpS is hit.
4. If the boundary of SpS is hit, reflect the trajectory. Update the direction accordingly.
5. When the length is reached at point $\vec{p}_i$, add it to the list of points and set $\vec{p}_0 := \vec{p}_i$.
6. Go to step 2. (Do this N times).

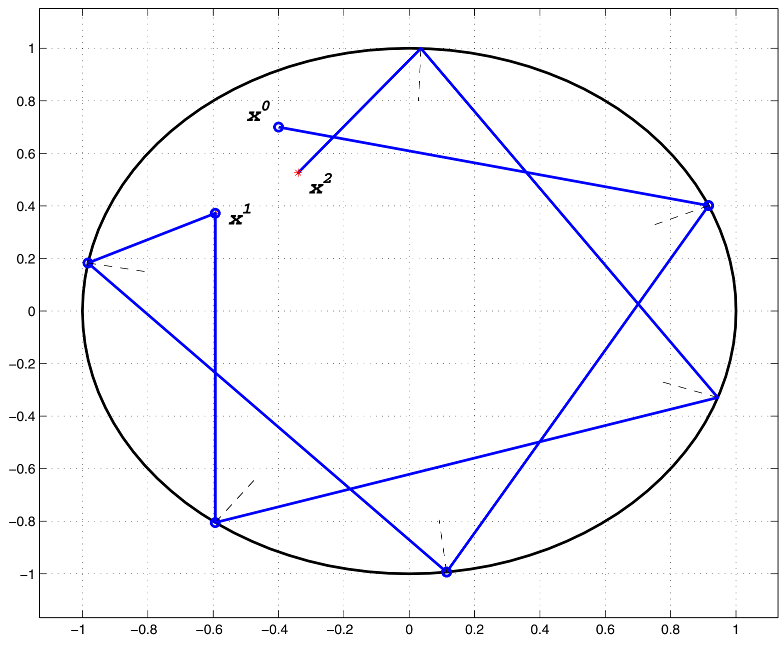


Figure 3.3: Possible trajectory for a point generated by the Billiard Walk (taken from [10])

3.3.2 Algorithm

The pseudocode of the Billiard Walk algorithm can be seen in Algorithm 2. Figure 3.3 illustrates an example of a possible trajectory. A proof for asymptotic uniformity can be found in [10, Theorem 1] .

Preparations

In order to generate the points, the spectrahedral shadow needs to first be centred at the origin, hence we can subtract the centre $\vec{c}$ from every point of SpS. This is because for the next methods we will need to compute the dual of the spectrahedral shadow, which is only well-defined if it contains the origin.

Similarly to the Ball Walk Algorithm, the starting point $\vec{p}_0$ needs to be chosen at random, but not (necessarily) uniformly random. This is done similarly as for the Ball Walk, *i.e.*, by computing the support function of SpS in a random direction, and checking which point solves the optimisation problem, thus giving a feasible, random point on the boundary.

For each point $\vec{p}_i$, we set the segment starting point $\vec{q}_0 := \vec{p}_0$. The length l of the segment corresponding to $\vec{p}_i$ is set to $-\tau \ln(\texttt{rand(1)})$. Two additional parameters for the Billiard Walk Algorithm that are needed are τ and R. The quantity τ was proposed in [10] to be defined as at least the magnitude of the diameter of SpS. Because calculating the diameter would be computationally expensive, τ can be set as the diameter of the interval box `interval(SpS)` instead. R is descibed in detail in Section 3.3.4.

Reflection Loop

We can imagine a line being drawn from the current point in the direction $\vec{d}$ and length l. In order to check if the segment length in the given direction is within the bounds of our spectrahedral shadow, a calculation of the maximum coefficient of $\vec{d}$, such that $\vec{q_0} + \lambda_{max} \cdot \vec{d} \in SpS$ is needed, further denoted λ_{max}. This can be expressed as an optimisation problem:

$$\begin{aligned} \underset{\lambda,\, \vec{x}}{\arg\min} \quad & -\lambda \\ \text{s.t.} \quad & \vec{q_0} + \lambda \cdot \vec{d} = \underline{G}\vec{x} \\ & \underline{A}_0 + \sum_{i=1}^{m} \underline{A}_i x_i \geq 0 \end{aligned} \tag{3.1}$$

The optimisation problem is expressed as an equivalent minimisation problem, as the solver we use is only able to solve such problems. Once computed, we can compare the obtained λ_{max} with l. Should the length be greater or equal the distance from the point to the boundary of our spectrahedral shadow, we can compute the coordinates of our point $\vec{p_i} = \vec{q_0} + l \cdot \vec{d}$ and add it to our list of points and break the while-loop. We can then proceed to the next iteration, starting from $\vec{p_0} := \vec{p_i}$.

Should $l > \lambda$, we need to perform a reflection on the boundary of SpS. We can determine the boundary point $\vec{y} = \vec{q_0} + \lambda_{max} \cdot \vec{d} \in \partial SpS$ by solving Equation (3.1). From this point we can reflect the line back within the spectrahedral shadow. To this end, the normal vector needs to be computed, which is described further in section 3.3.3. Let $\vec{s}$ be the corresponding unit outer normal of SpS at the boundary point $\vec{y}$. We need to update the starting point, direction and remaining length of the segment. To this end, we can set $\vec{q_0} := \vec{y}$, $\vec{d} := \vec{d} - 2(\vec{d}^\top \vec{s}) \cdot \vec{s}$ and $l := l - \lambda_{max}$, as proposed in [10]. The while-loop can now continue until the target length is reached.

3.3.3 Normal Vector Computation

In order to calculate the direction of the reflected line that would pass the boundary of our spectrahedral shadow, we need to compute the normal vector at the boundary point from which the reflection takes place. Computing this normal vector for a general set can be done using the dual:

Theorem 3.3.1. *A normal vector $\vec{\eta}$ at a boundary point $\vec{y}$ of a convex and compact set S can be computed as:*

$$\vec{\eta} = \underset{\vec{x} \in S^*}{\arg\max} \quad \vec{y}^\top \vec{x},$$

where S^* is the dual set of S.

Proof. We only provide a sketch of the proof, since a formal proof, which would involve handling subgradients of certain functions, is beyond the scope of this thesis. Any convex, compact set S can be approximated by a sequence of polytopes P_N that cooverge to S (with respect to the Hausdorff distance). The formula for computing the normal vector at a boundary point $\vec{y} \in \partial P$ is [12]:

$$\vec{\eta} = \underset{\vec{z} \in P^*}{\arg\max} \quad \vec{y}^\top \vec{z}.$$

Let y_N be boundary points on P_N, such that $y_N \to y$, and let η_N be the corresponding normal vectors. Then it is easy to see that $\eta_N \to \eta$ as defined in (3.2). Thus the normal vector definition holds for any boundary point $\vec{y}$. Q.E.D.

Now that we know how to compute the normal vector at the boundary points of any convex set, one can apply this for spectrahedral shadows:

$$\vec{\eta} = \underset{\vec{z} \in SpS^*}{\arg\max} \quad \vec{y}^\top \vec{z},$$

where $\vec{y} \in \partial SpS$.

Theorem 3.3.2. *Let SpS be a spectrahedral shadow containing the origin, with generator $\underline{G} \in \mathbb{R}^{n\times m}$, centre vector $\vec{c} = \vec{0}$, and coefficient matrices $\underline{A}_0, \underline{A}_1, ..., \underline{A}_m \in \mathbb{R}^{k\times k}$. Furthermore, let $\vec{y} \in \partial SpS$ be a boundary point of SpS. Then, an outer normal vector of SpS at the point $\vec{y}$ can be computed as follows:*

$$\begin{aligned} \underset{\vec{z}}{\arg\min} \quad & \vec{y}^\top \vec{z} \\ s.t. \quad & tr(\underline{A}_0 \underline{Z}) \leq 1 \\ & tr(\underline{A}_i \underline{Z}) = \vec{e}_i^\top \underline{G}^\top \vec{z} \quad \forall i = 1, ..., m \\ & \underline{Z} \succeq 0 \end{aligned}$$

Proof. By Theorem 3.3.1 we know that a normal vector $\vec{\eta}$ can be computed as $\vec{\eta} = \arg\max_{\vec{x} \in S^*} \vec{y}^\top \vec{x}$. The constraint $\vec{z} \in SpS^*$ can be evaluated using Theorem 2.4.2, which results in:

$$\begin{aligned} \underset{\vec{z}}{\arg\max} \quad & \vec{y}^\top \vec{z} \\ \text{s.t.} \quad & tr(\underline{A}_0 \underline{Z}) \leq 1 \\ & -tr(\underline{A}_i \underline{Z}) = \vec{e}_i^\top G^\top \vec{z} \quad \forall i = 1, ..., m \\ & \underline{Z} \succeq 0 \end{aligned}$$

As our solver is only able to solve minimisation problems, we will need to transform this accordingly:

$$\begin{aligned}
\arg\min_{\vec{z}} \quad & -\vec{y}^\top \vec{z} \\
\text{s.t.} \quad & tr(\underline{A}_0 \underline{Z}) \leq 1 \\
& -tr(\underline{A}_i \underline{Z}) = \vec{e}_i^\top \underline{G}^\top \vec{z} \quad \forall i = 1, ..., m \\
& \underline{Z} \geq 0
\end{aligned}$$

By substituting $\vec{z} := -\vec{z}$, we obtain the following:

$$\begin{aligned}
\arg\min_{\vec{z}} \quad & \vec{y}^\top \vec{z} \\
\text{s.t.} \quad & tr(\underline{A}_0 \underline{Z}) \leq 1 \\
& tr(\underline{A}_i \underline{Z}) = \vec{e}_i^\top \underline{G}^\top \vec{z} \quad \forall i = 1, ..., m \\
& \underline{Z} \geq 0.
\end{aligned}$$

Q.E.D.

3.3.4 Reflection Issues

A problem that might occur when running the algorithm, is that the trajectory of the Billiard Walk might get stuck in a corner, which would result in a very large number of reflections which are relatively costly to compute. To this end, we can imagine the following example: A point has been generated in the neighbourhood around the corner of this spectrahedral shadow. As the direction is randomised and the trajectory of the point could lead in that same neighbourhood, points might just be contained in that region. To combat this, we will introduce the reflection number R which is set to $R_0 := 10 \cdot \texttt{dim(SpS)}$, as proposed in [10]. If a trajectory produces more than R_0 reflections, we abandon that trajectory and start anew from the point $\vec{p}_0$ of the current iteration, by choosing a new random direction and trajectory length.

3.4 Complexity

In the algorithms above, we only use basic numerical operations on vectors and matrices, as well as solving positive semi-definite programs, which have polynomial runtime. Consequently, each iteration of the algorithms has polynomial runtime. However, in principle, the algorithms might never terminate, and thus would have an infinite runtime. This is because it is theoretically possible, *e.g.*, for the Ball Walk, that the sampled points are always outside the spectrahedral shadow, which means that no further points can be added to the list. A similar problem can occur for the Billiard Walk. Fortunately, the likelihood of this happening is very low, so that both algorithms have on average polynomial runtime. One important question is therefore how many samples need to be generated such that points produced by either algorithm are uniformly distributed. This is called the mixing time of the algorithm, and an upper bound for the Ball Walk algorithm has been presented in [4]. For the Billiard Walk, no such bounds are known apart from the fact that the distribution is asymptotically uniform. However, in practice, the performance of the Billiard Walk is largely superior to that of the Ball Walk, and also compared to other geometric random walks [6].

Algorithm 2 Billiard Walk Algorithm

Input: a spectrahedral shadow SpS of dimension n with m generators, number of points N to be sampled in SpS
Output: the set of sampled points

```
1: $\vec{p}_i \leftarrow \vec{0}, \quad i = 1, ..., N$
2: $\vec{c} \leftarrow$center(SpS)
3: $\underline{G} \leftarrow$ SpS.G
4: $\vec{p}_0 \leftarrow$ feasible(SpS)
5: $R_0 \leftarrow 10\cdot$dim(SpS)
6: $\underline{A}_0, \underline{A}_i$ =getCoeffMatrices(SpS), $\quad i = 1, ..., n$
7: for i=1:N do
8:     $\vec{q}_0 \leftarrow \vec{p}_0$
9:     $I \leftarrow$ interval(SpS)
10:    $\tau \leftarrow \max(\sup I - \inf I)$
11:    $l \leftarrow -\tau \ln($rand(1)$)$
12:    $R \leftarrow R_0$
13:    $\vec{d} \leftarrow$ randn(dim(SpS), 1)
14:    $\vec{d} \leftarrow \vec{d} / \|\vec{d}\|_2$
15:    while True do
16:        $\lambda \leftarrow \arg\min_{\lambda}\{-\lambda : \vec{p}_i + \lambda \cdot \vec{d} = \underline{G} \cdot x\}$
17:        $\vec{y} \leftarrow \vec{q}_0 + \lambda \cdot \vec{d}$
18:        if $l \leq \lambda$ then
19:            $\vec{p}_0 \leftarrow \vec{q}_0 + l \cdot \vec{d}$
20:            $\vec{p}_i \leftarrow \vec{p}_0$
21:            break
22:        else
23:            $\vec{\eta} \leftarrow \arg\min_{\vec{x}}\{\vec{x}^\top \vec{y} : tr(\underline{A}_0 \underline{Z} \leq 1), \quad tr(\underline{A}_j \underline{Z}) = \vec{e}_j^\top \underline{G}^\top \vec{x}, \quad \underline{Z} \geq 0, \quad j = 1, ..., m\}$
24:            $\vec{s} = \vec{\eta} / \|\vec{\eta}\|_2$
25:            $\vec{d} \leftarrow \vec{d} - 2 \cdot (\vec{d}^\top \vec{s}) \cdot \vec{s}$
26:            $l \leftarrow l - \lambda$
27:            $\vec{q}_0 \leftarrow \vec{y}$
28:        end if
29:        if $R \leq 0$ then
30:            $R \leftarrow R_0, \quad \vec{q}_0 \leftarrow \vec{p}_0$
31:            $l \leftarrow -\tau \ln($rand(1)$)$
32:            $\vec{d} \leftarrow$ randn(dim(SpS), 1)
33:            $\vec{d} \leftarrow \vec{d} / \|\vec{d}\|$
34:            continue
35:        else
36:            $R \leftarrow R - 1$
37:        end if
38:    end while
39: end for
40: $\vec{p}_i \leftarrow \vec{p}_i + \vec{c}, \quad i = 1, ..., N$
41: return $\vec{p}_i, \quad i = 1, ..., N$
```

4 Containment

The containment of spectrahedra and spectrahedral shadows is a classical decision problem in convex optimisation theory. It asks whether a spectrahedral shadow is contained inside another spectrahedral shadow. This problem is NP-hard in general, as can be seen in [7], where it is proven that checking for two zonotopes whether one is inside the other is NP-hard. Since zonotopes can be expressed as spectrahedral shadows, this proves that containment for spectrahedral shadows is NP-hard to check.

4.1 Sampling the Inner Spectrahedral Shadow

Parameters

The main parameters used for both algorithms are SpS_1, SpS_2 and N. Both algorithms try to answer whether $SpS_1 \subseteq SpS_2$. The parameter N represents the number of points sampled for the probabilistic solution of the problem. The way the points are used is detailed in the following sections.

4.1.1 Outline

The first approach solves the containment problem, by randomly sampling points from SpS_1 (believed to be the inner spectrahedral shadow) by using one of the random sampling algorithms, described in section 3 (we use the Billiard Walk in the pseudocode below). We can think of the containment verification in the following way:

1. Generate N random uniformly distributed points inside SpS_1 by using the Billiard Walk Algorithm.

2. Draw a line in a random direction from every point $\vec{p}_i \in SpS_1$ until the boundary is reached.

3. Check if the resulting boundary point $\vec{y} \in \partial SpS_1$ is also contained in SpS_2. Stop if it is not and return False. Else, go to step 3.

4. Return True, since every boundary point that was created, has fulfilled the above condition.

This algorithm can only disprove containment, *i.e.*, if we find some $\vec{y} \notin SpS_2$ we know for certain that $SpS_1 \nsubseteq SpS_2$. It can not confirm containment, as this would require sampling a potentially infinite amount of points. However, if the amount N of sampled points is hight enough, and no point $\vec{y}$ has been found that does not lie in SpS_2, one can conclude that containment is likely.

4.1.2 Algorithm

Algorithm 3 Inner Sampling Containment Algorithm

Input: spectrahedral shadows SpS_1, $SpS_2 \subseteq \mathbb{R}^n$ with m generators and number of sampled points N
Output: Return False if $SpS_1 \not\subseteq SpS_2$, True otherwise

```
1: $\underline{G} \leftarrow$ SpS$_1$.G
2: $\vec{p}_i \leftarrow$BilliardWalk(SpS$_1$, N),   $i = 1, ..., N$
3: $\underline{A}_0, \underline{A}_i \leftarrow$ getCoeffMatrices(SpS$_1$),   $i = 1, ..., m$
4: for i=1:N do
5:     $\vec{d} \leftarrow$ randn(1)
6:     $\vec{d} \leftarrow \vec{d} / \|\vec{d}\|_2$
7:     $\lambda \leftarrow \arg\min\{-\lambda : \vec{p}_i + \lambda \cdot \vec{d} = \underline{G} \cdot x\}$
8:     $\vec{y} \leftarrow \vec{p}_i + \lambda \cdot \vec{d}$
9:     if $\vec{y} \notin SpS_2$ then
10:        return False
11:    end if
12: end for
13: return True
```

4.2 Sampling the Outer Spectrahedral Shadow

4.2.1 Outline

The second algorithm approaches the containment problem in a different manner. This time, instead of sampling points from the inner spectrahedral shadow, we will sample points from the outer one. We then generate boundary points of SpS_2 in a similar fashion as we did in the previous algorithm for SpS_1 and use the support function to provide a measure of how far SpS_1 and SpS_2 extend along the direction of each individual normal vector. If we find a direction in which SpS_1 extends further then SpS_2, then we know for certain that $SpS_1 \not\subseteq SpS_2$. The verification of containment can be formulated as follows:

1. Subtract the centre of SpS_2 from both spectrahedral shadows (so as to ensure that the origin is contained in SpS_2, which is necessary to compute the normal vectors.
2. Generate N random uniformly distributed points inside SpS_2 by using the Billiard Walk Algorithm.

3. Draw a line in a random direction from every point $\vec{p}_i \in SpS_2$ until the boundary is reached.
4. Compute the normal vector $\vec{\eta}$ of the resulting boundary point $\vec{y} \in \partial SpS_2$ (detailed in section Section 3.3.3).
5. Define $a := h_{SpS_1}(\frac{\vec{\eta}}{\|\vec{\eta}\|_2})$ and $b := h_{SpS_2}(\frac{\vec{\eta}}{\|\vec{\eta}\|_2})$. h_A is defined in Section 2.5.
6. Should $a > b$, then the algorithm terminates and returns False. Else, go to step 4.
7. If every $a \leq b$ after the loop is finished, the algorithm terminates and returns True.

4.2.2 Algorithm

Algorithm 4 Outer Sampling Containment Algorithm

Input: spectrahedral shadows SpS_1, $SpS_2 \subseteq \mathbb{R}^n$ with m generators and number of sampled points N
Output: Return False if $SpS_1 \not\subseteq SpS_2$, True otherwise

```
1: $\vec{c} \leftarrow$ center(SpS$_2$)
2: $SpS_1 \leftarrow SpS_1 - \vec{c}$
3: $SpS_2 \leftarrow SpS_2 - \vec{c}$
4: $\vec{p}_i \leftarrow$BilliardWalk(SpS$_2$, N),   $i = 1, ..., N$
5: $\underline{A}_0, \underline{A}_i \leftarrow$ getCoeffMatrices(SpS$_2$),   $i = 1, ..., m$
6: for i=1:N do
7:     $\vec{d} \leftarrow randn(1)$
8:     $\vec{d} \leftarrow \vec{d}/\|\vec{d}\|_2$
9:     $\lambda \leftarrow \arg\min\{-\lambda : \vec{p}_i + \lambda \cdot \vec{d} = \underline{G} \cdot x\}$
10:    $\vec{y} \leftarrow \vec{p}_i + \lambda \cdot \vec{d}$
11:    $\vec{\eta} \leftarrow \arg\min_{\vec{x}}\{\vec{x}^\top \vec{y} : tr(\underline{A}_0 \underline{Z} \leq 1),\quad tr(\underline{A}_i \underline{Z}) = \vec{e}_j^\top \underline{G}^\top \vec{x},\quad j = 1, ..., m\}$
12:    $a \leftarrow h_{SpS_1}(\vec{\eta}/\|\vec{\eta}\|_2)$
13:    $b \leftarrow h_{SpS_2}(\vec{\eta}/\|\vec{\eta}\|_2)$
14:    if $a > b$ then
15:        return False
16:    end if
17: end for
18: return True
```

5 Evaluation

In this section we are going to evaluate the performance of the presented algorithms with respect to uniformity and performance. We are also going to focus on comparisons with similar methods for other set representations. Section 5.1 and Section 5.2 will focus on the uniform sampling algorithms, whereas Section 5.3 will be dedicated to evaluating the containment algorithms.

5.1 Uniformity Evaluation

This section will focus on comparing the presented algorithms in regards to the distribution of the generated points, namely the degree to which they are distributed uniformly.

5.1.1 Choice of Set Representation

The main scope of this thesis is analysing methods to generate random, uniformly distributed points. This means that we want to assess the uniformity of the points generated by the two algorithms presented in Chapter 3. In general, one can evaluate how uniformly the points are distributed in a set, by using Definition 2.3.1.

This is more difficult for spectrahedral shadows in general, as there is no way to compute their volume yet. As such, we will focus on a different type of set, whose volume can be computed and that can be represented as a spectrahedral shadow. We can then generalise the results for spectrahedral shadows as a whole.

For simplicity, the following measurements have been taken in dimension $n = 2$, but the results can be generalised to higher dimensions as well.

5.1.2 Preparations for evaluating Capsules

We first need to generate a capsule C around the origin and compute its volume. In order to apply Definition 2.3.1, we need to find a subset within C. We can do this by computing the interval box I of C and scaling it down, by some factor δ that ensures that $\delta I \subseteq C$. We are now able to compute the ratio of the volumes $\frac{vol(\delta I)}{vol(C)}$. Points in C generated by an algorithm are then approximately uniform, if the ratio of the points contained in I, compared to the total number of sampled points, approaches $\frac{vol(\delta I)}{vol(C)}$. In order to use the two geometric random walk methods that have been implemented, we first need to transform C into a spectrahedral shadow, which was readily implemented in CORA prior to the writing of this thesis.

5.1.3 Plot Comparisons

When looking at the plots for points sampled using the two methods, there seems to be no clear issue regarding the uniformity of either the Ball Walk, shown in Figure 5.1, or the Billiard Walk, shown in Figure 5.2.

5.1.4 Nummerical Comparison

In order to compare the uniformity of the two methods, we generate points using the Ball Walk Algorithm, described in Section 3.2, and then count the number of points contained in I. We can do this for different N, in order to analyse the convergence.

The same can be done using the Billiard Walk Algorithm, described in section Section 3.3, also for different N. We can observe, that, in practice, the Billiard Walk converges to the uniform distribution much faster than the Ball Walk, which would also indicate why it is preferable over the Ball Walk, when intending to achieve a better approximation of the uniform distribution.

This development is surprising to a certain extent, knowing that there exists an upper bound for the number of points sampled, such that the Ball Walk converges to the uniform distribution. No such proof exists for the Billiard Walk. The only known result about the convergence of the Billiard Walk is that it will eventually converge to the uniform distribution, but no bounds on the mixing time are known.

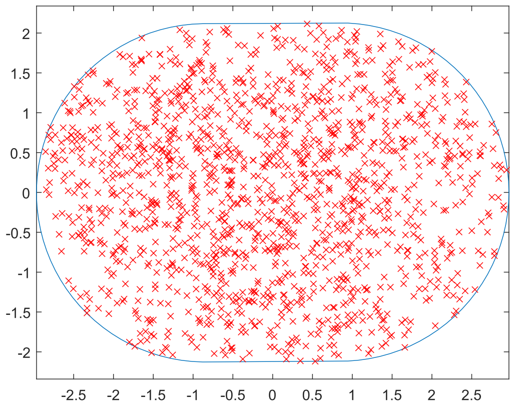
5.2 Performance Evaluation

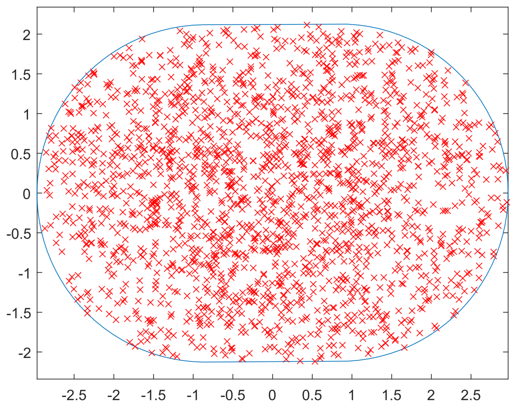
Now that we compared how the algorithms fare with respect to uniformity, it might be useful to evaluate how fast the algorithms run in practice.

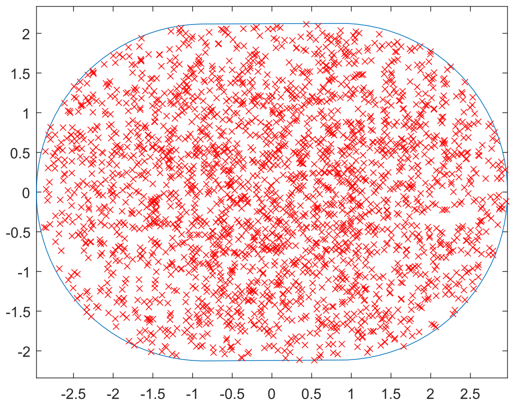
5.2.1 Runtime Comparison

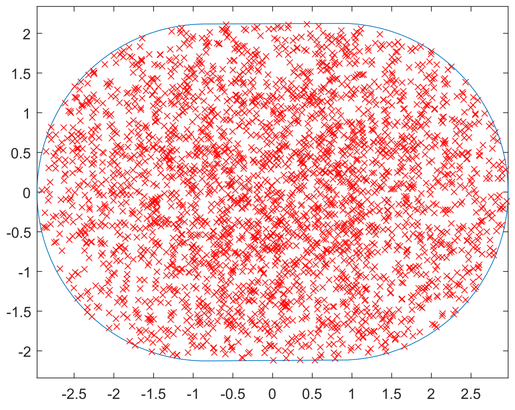
When considering the runtime as a metric for comparing the two methods, we can observe that the Ball Walk outperforms the Billiard Walk by a significant amount, as N increases. This can likely be explained by the procedures used by each algorithm, when sampling points.

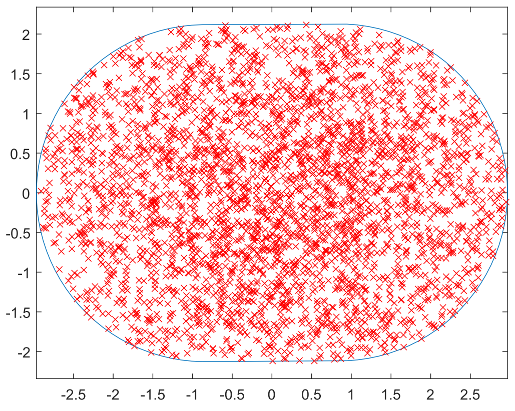
The Ball Walk uses a rounded spectrahedral shadow, in order to work faster (detailed in


(a) $N = 1500$


(b) $N = 2000$


(c) $N = 2500$


(d) $N = 3000$


(e) $N = 3500$

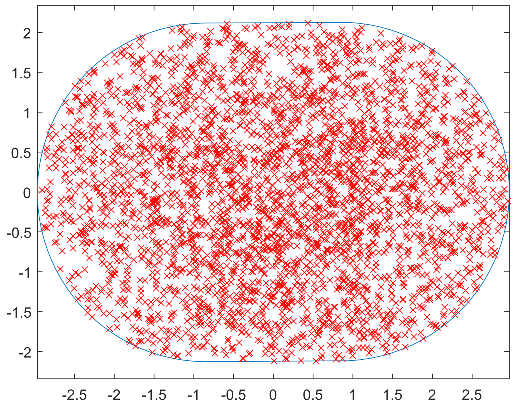
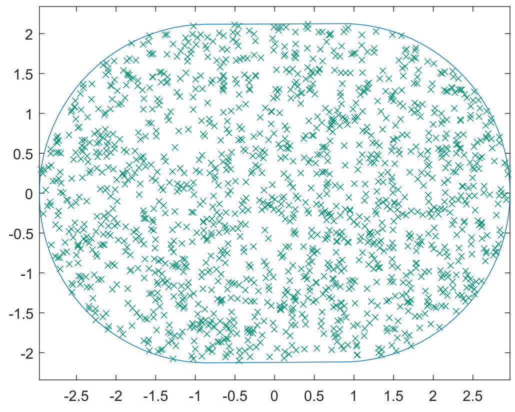
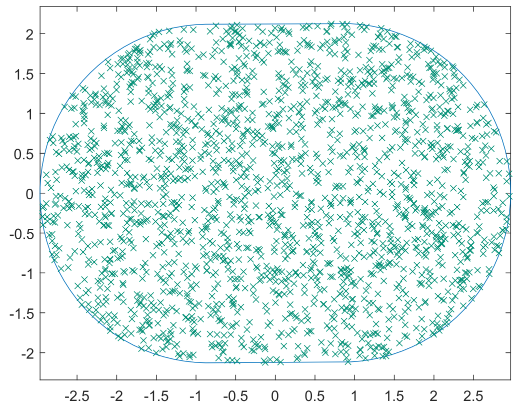

(f) $N = 4000$

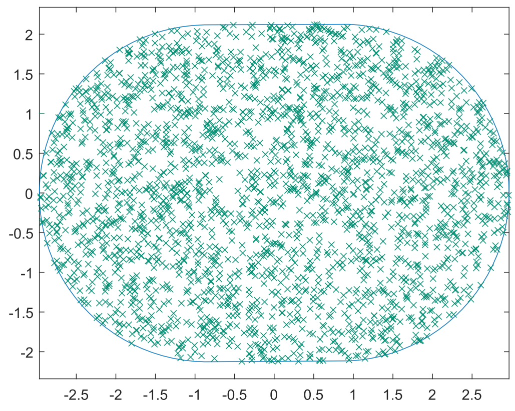
Figure 5.1: Ball Walk Scatter Plot for different sample sizes N



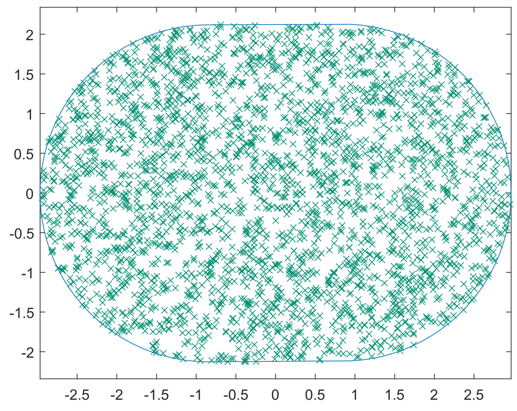
(a) $N = 1500$



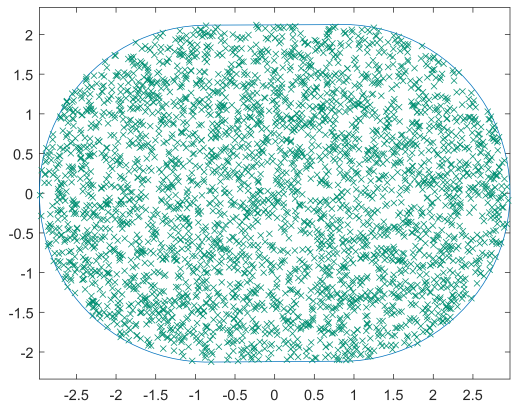
(b) $N = 2000$



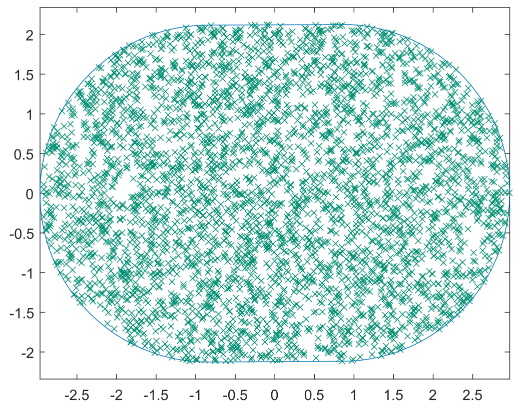
(c) $N = 2500$



(d) $N = 3000$



(e) $N = 3500$



(f) $N = 4000$

Figure 5.2: Billiard Walk Scatter Plot for different sample sizes N

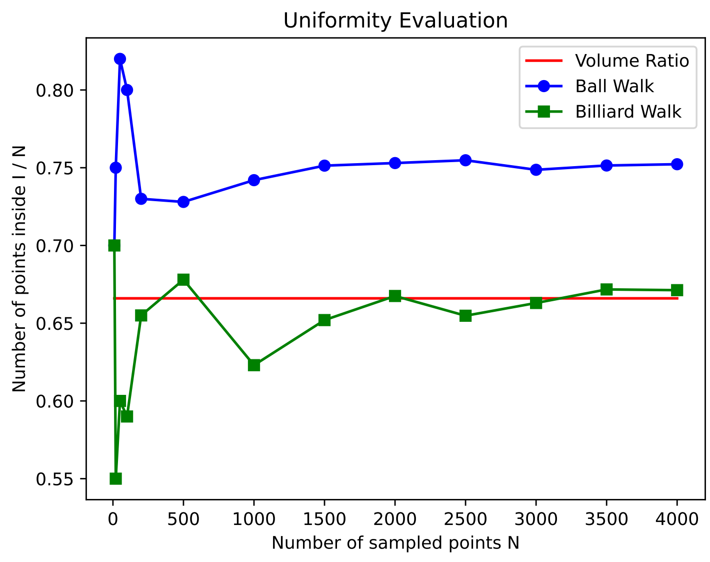


Figure 5.3: Uniformity Comparison: The reference volume ratio for our evaluation is $\frac{vol(\delta I)}{vol(C)} = 0.6659$. We can see that the ratio between the number of points inside I, sampled using the Billiard Walk, and N approaches the reference volume ratio, once $N > 1000$, whereas the same cannot be said about the Ball Walk. There the ratio does seem to be consistently off by about 9%.

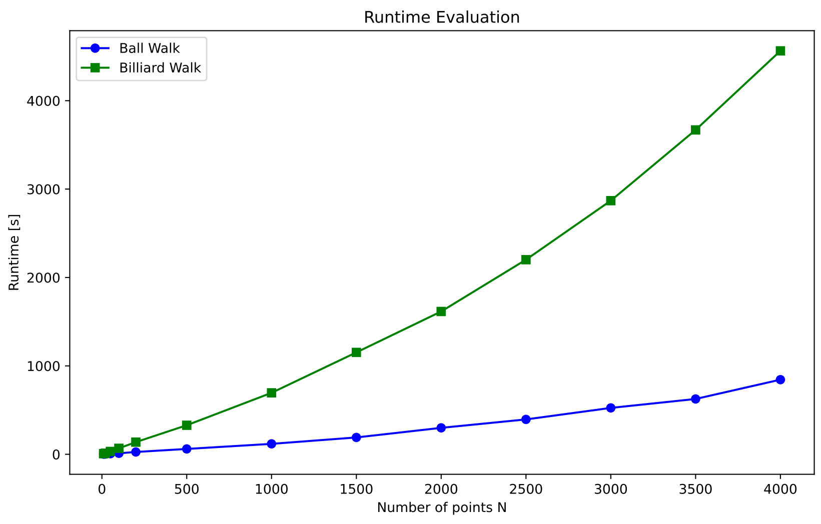


Figure 5.4: Runtime Comparison: Here we can see that the runtime of the Billiard Walk grows significantly faster than the runtime of the Ball Walk. The difference is apparent once $N > 1000$.

Section 3.2.2) and makes use of semi-definite programming only when checking point containment. This is in contrast to the Billiard Walk Algorithm, which uses semi-definite programming when computing the maximum distance a segment can extend to, in addition to the normal vector, should this segment extend over the boundary of the spectrahedral shadow. This, along with the possible reflection issues (described in Section 3.3.4), shows why the difference in computational steps and, hence, speed, is apparent.

5.2.2 Comparison with Polytopes

One other interesting aspect is how well the sampling algorithms for spectrahedral shadows perform, when compared to the same algorithms for other set representations. For instance, spectrahedral shadows are able to represent polytopes, and both random walks were already implemented in CORA for polytopes prior to the writing of this thesis, which begs the question whether the implementation for spectrahedral shadows perfoms similarly to that for polytopes.

The implementation for polytopes has an inherent advantage: While for spectrahedral shadows, in each iteration we need to solve a positive semi-definite program, polytopes only require solving linear programs, which are significantly faster. This is confirmed in Table 5.1.

N	Ball Walk P	Billiard Walk P	Ball Walk SpS	Billiard Walk SpS
100	0.1791	0.1207	14.7981	108.1077
1000	0.4246	0.1179	275.2458	4071.6

Table 5.1: Runtime Comparison [s] of Random Walks for Polyopes P and Spectrahedral Shadows SpS

5.3 Evaluation of the Containment Algorithms

The two presented containment algorithms work very differently and it would be interesting to see how their behaviour differs, with respect to runtime. Since no other method for solving the containment problem for spectrahedral shadows has been developed and there is no naive way of solving it, we can only compare the two presented approaches to each other.

Runtime Evaluation

In order to analyse the runtime of the two containment algorithms, we randomly generate a spectrahedral shadow and centre it. We then copy that same spectrahedral shadow and scale it down by a factor δ, in order to make sure that it is contained in the original set. Multiple N can then be used for these tests. Both methods disprove containment by finding an unfit point or a normal vector respectively. This means that the maximum number of iterations, and hence the maximum runtime, is attained when the spectrahedral shadow is contained, which would give us a good metric for comparisons. The sampling algorithm used for both algorithms will be the Billiard Walk, as it performed better in the Uniformity evaluation, presented in Section 5.1.

We can observe in Figure 5.5, Figure 5.6 and Figure 5.7 that the Inner Sampling Containment Algorithm performs better than the Outer Sampling Containment Algorithm. The difference becomes more evident when increasing the dimension of the spectrahedral shadows. Both algorithms make use of semi-definite programming, for point containment and the computation of normal vectors, respectively, but it is also used for computing the support function in the Outer Sampling Containment Algorithm, which explains why the performance varies.

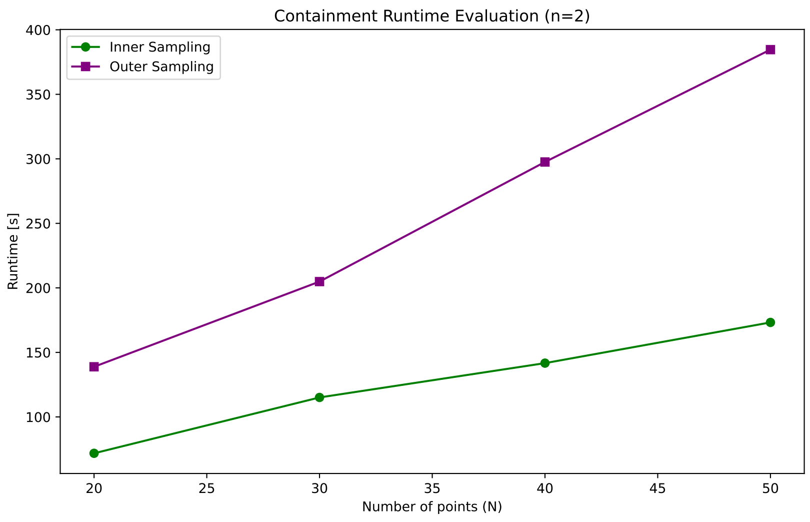


Figure 5.5: Containment Runtime Evaluation for Dimension $n = 2$: The difference in runtime is relatively small at first, but becomes more significant once N grows. For this and subsequent evaluations $N \in \{20, 30, 40, 50\}$

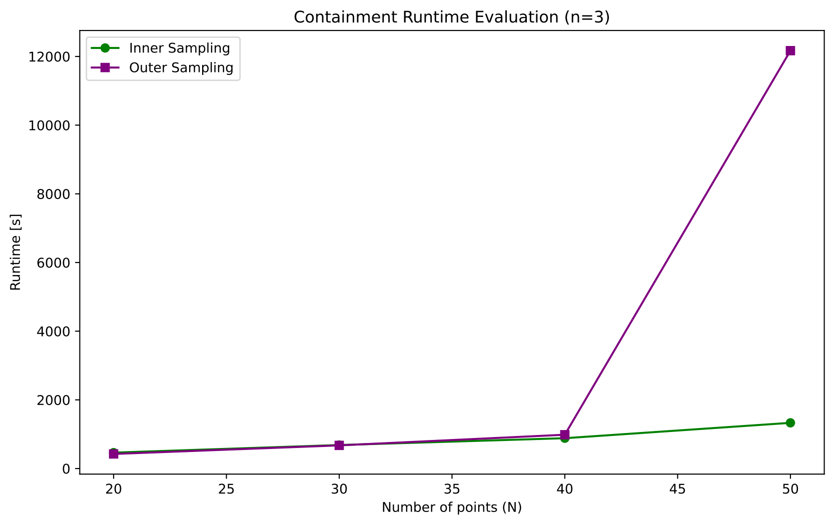


Figure 5.6: Containment Runtime Evaluation for Dimension $n = 3$: The difference in runtime is, for the most part, similar to the evaluation for $n = 2$, but becomes significantly larger ones $N = 50$.

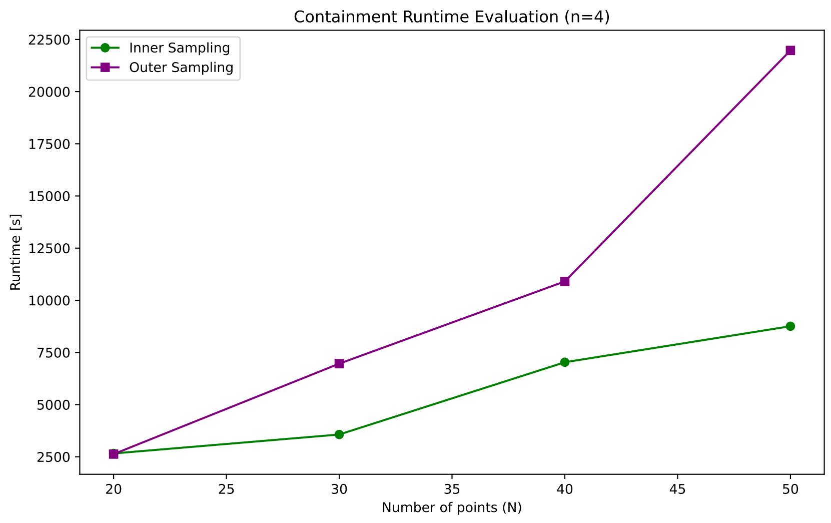


Figure 5.7: Containment Runtime Evaluation for Dimension $n = 4$: The difference in runtime becomes more apparent in this dimension, even for $N = 30$.

6 Conclusion

The main goal of this thesis was to find efficient and reliable methods for uniform sampling on spectrahedral shadows, as naively sampling inside the interval box of spectrahedral shadows does not scale well for higher dimensions. To this end, two sampling algorithms were presented, the Ball Walk and the Billiard Walk. Both methods run on average in polynomial time with respect to the representation size of the spectrahedral shadow.

The secondary goal of this thesis was to find a solution to the containment problem for spectrahedral shadows. We therefore considered two different approaches, which both used uniform random sampling for solving the containment problem probabilistically, by either sampling the inner spectrahedral shadow or sampling the outer spectrahedral shadow.

Uniformity

Asymptotic uniformity has been proven for both methods. Testing the correctness was done using capsules, a special case of spectrahedral shadows, for which the volume was easy to compute. Both algorithms are Markov Chain Monte Carlo methods, hence they sample every additional point from the previous one, through a random method. This means that there is a minimum number of points that have to be sampled until a convergence to the uniform distribution is achieved. Although the Ball Walk has been proven to have an upper bound on its convergence, we observed that, in practice, it converges considerably slower than the Billiard Walk, for which no such proven upper bound exists.

Performance

When comparing the two methods to each other, we observed that the Ball Walk had a far better runtime than the Billiard Walk. We concluded that this happens on account of the multiple semi-definite programs having to potentially be solved for the sampling of even one point when using the Billiard Walk, whereas this is not the case for the Ball Walk. Given that the Billiard Walk converged faster, but the Ball Walk had a better runtime, it is up to the user to decide which algorithm is better in which context, as a trade-off has to be considered between the two metrics.

We also analysed how the two methods perform on polytopes. To this end, we used the implementations for spectrahedral shadows on the polytopes represented as such and the direct implementations for polytopes. The main difference between the two sets of implementations is the usage of semi-definite programming for spectrahderal shadows, compared to the usage of linear programming for polytopes. We concluded that, especially in the case of the Billiard Walk, the polytope implementations run considerably faster and it may not be worth to use semi-definite programming, if one is sure that the set in question is a polytope.

Containment

We looked into one particular application of the uniform sampling algorithms presented in this thesis, the containment problem for spectrahedral shadows. The containment of spectrahderal shadows has a wide range of uses and applications in robotics, such as steering a system into a target space, which is why this thesis had a particular focus on this problem.

When analysing the two methods, we discovered that the algorithm uniformly sampling the inner spectrahedral shadow yielded faster results, as it does not necessitate the same amount of computations as the second method.

Future Work

Applications of Uniform Sampling and Containment

Uniform sampling for spectrahedra and the containment problem for spectrahedral shadows are worth further research, as they provide useful ways of analysing a very general set representation for cyber-physical systems.

Uniform sampling is a useful tool in set-based reachability analysis. Future development in the field could employ the methods presented in this thesis, *i.e.*, using uniform sampling for estimating the set of reachable states of a system and analysing its behaviour over time. The application in safety verification might also be of interest, where one needs to determine whether a system may reach forbidden states. The algorithms presented in this thesis could be used to uniformly sample points inside the state space, in order to determine unsafe states and analysing the likelihood of entering them.

The containment problem for spectrahedral shadows may be relevant for its uses in a variety of fields. For control applications, it could be used when a system needs to be steered toward a state, for which containment needs to checked first. In addition, containment plays a major role in constraint satisfaction problems in control theory. Set containment techniques can be used in order to apply constraints on the state space and the input variables, thus ensuring the feasibility of the resulting optimisation problem.

Further Analysis of the Ball Walk Algorithm

When analysing and implementing the Ball Walk, the concept of "rounding" was mentioned, as a way for the Ball Walk to converge faster. In the future, one could study how rounding effectively affects the convergence of the Ball Walk, since, the numerical testing presented in Section 5.1.4 shows that, in practice, the Ball Walk converges rather slowly to the uniform distribution. It might be useful to study how changing or modifying the rounding procedure influences the behaviour of the Ball Walk algorithm.

Proof of the Probability for Containment

While this thesis did provide two methods for deciding containment for spectrahedral shadows, both methods take a probabilistic approach for the problem. It might be interesting to establish and prove the probability that the decision of containment is indeed accurate, similarly to the proof done for zonotope containment in [8]. This would provide a closer validation of the presented algorithms and indicate a certain degree of reliability when confirming the containment of two spectrahedral shadows.

List of Figures

List of Tables

Bibliography

[1] M. Althoff. “Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars.” PhD thesis. Technische Universität München, 2010.

[2] T. Netzer. “Spectrahedra and Their Shadows.” PhD thesis. Universiät Leipzig, 2011.

[3] S. Vempala. “Geometric Random Walks: A Survey.” In: *Combinatorial and Computational Geometry MSRI Publications Volume* 52 (Jan. 2005).

[4] L. László and M. Simonovits. “Random Walks in a Convex Body and an Improved Volume Algorithm.” In: *Random Struct. Algorithms* (Jan. 1993), pp. 359–412.

[5] M. Althoff. “An Introduction to CORA 2015.” In: *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*. EasyChair, 2015, pp. 120–151.

[6] A. Chalkis and V. Fisikopoulos. “volesti: Volume Approximation and Sampling for Convex Polytopes in R.” In: *The R Journal* (2021), p. 561.

[7] A. Kulmburg and M. Althoff. “On the co-NP-Completeness of the Zonotope Containment Problem.” In: *European Journal of Control* (June 2021).

[8] A. Kulburg, I. Brkan, and M. Althoff. “Search-based and Stochastic Solutions to the Zonotope and Ellipsotope Containment Problems.” In: (to appear).

[9] V. Balakrishnan and L. Vandenberghe. “Semidefinite programming duality and linear time-invariant systems.” In: *Automatic Control, IEEE Transactions on* (Feb. 2003), pp. 30 –41.

[10] B. Polyak and E. Gryazina. “Billiard walk - a new sampling algorithm for control and optimization.” In: *IFAC Proceedings Volumes* (2014). 19th IFAC World Congress, pp. 6123–6128.

[11] P. Gritzmann and V. Klee. “Computational complexity of inner and outerj-radii of polytopes in finite-dimensional normed spaces.” In: *Mathematical Programming* (Jan. 1993), pp. 163–213.

[12] S. Prenitzer. *Uniform Trajectory Planning for Cyber-Physical Systems*. Bachelor’s Thesis. 2023.