NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES
“THEORETICAL COMPUTER SCIENCE”

MASTER THESIS

Sampling Methods, Spectrahedra and Convex Optimization

Panagiotis G. Repouskos

Supervisors: **Ioannis Emiris,** Professor
Vassilis Zissimopoulos, Professor
Gregory Karagiorgos, Associate Professor

ATHENS

February 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
“ΘΕΩΡΗΤΙΚΗ ΠΛΗΡΟΦΟΡΙΚΗ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μέθοδοι Δειγματοληψίας, Σπεκτράεδρα και Κυρτή Βελτιστοποίηση

Παναγιώτης Γ. Ρεπούσκος

Επιβλέποντες: **Ιωάννης Εμίρης,** Καθηγητής
Ζησιμόπουλος Βασίλειος, Καθηγητής
Γρηγόρης Καραγιώργος, Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

Φεβρουάριος 2023

MASTER THESIS

Sampling Methods, Spectrahedra and Convex Optimization

Panagiotis G. Repouskos
R.N.: cs1180004

SUPERVISORS: **Ioannis Emiris,** Professor
Vassilis Zissimopoulos, Professor
Gregory Karagiorgos, Associate Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μέθοδοι Δειγματοληψίας, Σπεκτράεδρα και Κυρτή Βελτιστοποίηση

Παναγιώτης Γ. Ρεπούσκος
Α.Μ.: cs1180004

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ιωάννης Εμίρης,** Καθηγητής
Ζησιμόπουλος Βασίλειος, Καθηγητής
Γρηγόρης Καραγιώργος, Αναπληρωτής Καθηγητής

ABSTRACT

We present algorithmic, complexity, and implementation results on the problem of sampling points in the interior and the boundary of a spectrahedron, that is the feasible region of a semidefinite program.

Our main tool is random walks. We define and analyze a set of primitive geometric operations that exploits the algebraic properties of spectrahedra and the polynomial eigenvalue problem, and leads to the realization of a broad collection of efficient random walks. We demonstrate random walks that experimentally show faster mixing time than the ones used previously for sampling from spectrahedra in theory or applications, for example Hit and Run. Consecutively, the variety of random walks allows us to sample from general probability distributions, for example the family of log-concave distributions which arise frequently in numerous applications.

We apply our tools to specialize a randomized convex optimization algorithm for spectrahedra, that is to solve semidefinite programs. We provide a C++ open source implementation of several random walks that scale efficiently to a high number of dimensions (in our experiments we tested till 300 dimensions) and of the convex optimization algorithm tailored for spectrahedra.

SUBJECT AREA: Random Walks, Geometry, Convex Optimization, Linear Algebra

KEYWORDS: sampling, convex optimization, spectrahedra, linear matrix inequalities, geometric random walks, semidefinite programming

ΠΕΡΙΛΗΨΗ

Παρουσιάζουμε αποτελέσματα σε αλγορίθμους, πολυπλοκότητα και υλοποίηση σχετικά με το πρόβλημα δειγματοληψίας του εσωτερικού και του συνόρου ενός σπεκτραέδρου.

Το κύριο εργαλείο μας είναι οι τυχαίοι περίπατοι. Ορίζουμε και αναλύουμε ένα σύνολο βασικών γεωμετρικών πράξεων, οι οποίες εκμεταλλεύονται τις αλγεβρικές ιδιότητες των σπεκτραέδρων και το πολυωνυμικό πρόβλημα ιδιοτιμών, και οδηγούν στην πραγματοποίηση μίας ευρείας συλλογής αποδοτικών τυχαίων περιπάτων. Δείχνουμε τυχαίους περιπάτους, οι οποίοι πειραματικά έχουν ταχύτερο χρόνο σύγκλισης από όσους χρησιμοποιούνταν μέχρι τώρα, είτε σε θεωρία είτε σε εφαρμογές. Αυτοί οι τυχαίοι περίπατοι μας επιτρέπουν να κάνουμε δειγματοληψία από μία μεγάλη οικογένεια κατανομών, οι οποίες προκύπτουν σε διάφορες εφαρμογές.

Χρησιμοποιούμε αυτά τα εργαλεία για να ειδικεύσουμε έναν τυχαιοκρατικό αλγόριθμο κυρτής βελτιστοποίησης σε σπεκτράεδρα. Παρέχουμε μία C++ υλοποίηση ανοιχτού κώδικα, διαφόρων τυχαίων περιπάτων (οι οποίοι δουλεύουν και σε περισσότερες από 300 διαστάσεις) και του αλγορίθμου κυρτής βελτιστοποίησης για σπεκτράεδρα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τυχαίοι Περίπατοι, Γεωμετρία, Κυρτή Βελτιστοποίηση, Γραμμική Άλγεβρα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: δειγματοληψία, κυρτή βελτιστοποίηση, σπεκτράεδρα, γεωμετρικοί τυχαίοι περίπατοι

ACKNOWLEDGEMENTS

At this point, I should express my gratitude towards the people that played major roles in this study, Firstly, I would like to thank both Dr. Ioannis Emiris and Dr. Gregory Karagiorgos, whose courses in the University of Athens provided a solid background in the areas of this work and, were always eager to provide advice.

I should especially mention Dr. Elias Tsigaridas, who not only helped in developing many of the theoretical results, but was also willing to provide tutoring in whatever I needed. Also, I would like to thank Dr. Vissarion Fisikopoulos, who provided the foundations for the software we developed with the C++ library volesti and throughout the project assisted in both theory and coding. Last, but by no means least, I would like to thank Apostolos Chalkis, at this time, a PHD student. He played major role in the software development, in the understanding of the underlying theory and in every step of the research. Part of this thesis will also be published in a separate paper [1].

[1] https://hal.inria.fr/hal-02572792

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction

In this thesis, we deal with the problem of sampling points from the interior and boundary of spectrahedra, via the use of geometric random walks. We explore some random walks, which are for general convex bodies, and tailor them for spectrahedra. This requires the development of basic geometric operations on spectrahedra and subsequently, the implementation of a membership or boundary oracle, which will allow the randoms walk to ”navigate” in the spectrahedron.

As to why spectrahedra are of particular interest, we mention that spectrahedra are the most studied convex bodies other than polytopes and, can be regarded as their generalization. Efficient methods for sampling points in spectrahedra are crucial for many applications, such as volume approximation [11], integration [26], semidefinite optimization [26, 19] and applications in robust control analysis [8, 7, 37]. Our focus will be using geometric random walks and a randomized convex optimization algorithm [20] to solve semidefinite programs.

In the remainder of this chapter, we provide some basic background on semidefinite programming, geometric random walks and polynomial eigenvalue problems. In Chapter 2, we develop three basic geometric operations on spectrahedra and demonstrate them in an example in two dimensions. In Chapter 3, we present some well known random walks, tailored for spectrahedra, based on the developed geometric operations of Chapter 2. Finally, in Chapter 4, we use a randomized convex optimization algorithm to solve semidefinite programs.

1.2 Notation

We denote by O, resp. $\mathcal{O}_B$, the arithmetic, resp. bit, complexity and we use $\widetilde{\mathcal{O}}$, respectively $\widetilde{\mathcal{O}}_B$, to ignore (poly-)logarithmic factors. The bitsize of a univariate polynomial $A \in \mathbb{Z}[x]$ is the maximum bitsize of its coefficients. We use bold letters for matrices, $\boldsymbol{A}$, and vectors, $\boldsymbol{v}$; we denote by $A_{i,j}$, resp. v_i, their elements; $\boldsymbol{A}^\top$ is the transpose and $\boldsymbol{A}^*$ the adjoint of $\boldsymbol{A}$. If $\boldsymbol{x} = (x_1, \ldots, x_n)$, then $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + \sum_{i=1}^n x_i \boldsymbol{A}_i$, see (1.1). For two points $\boldsymbol{x}$ and $\boldsymbol{y}$, we denote the line through them by $\ell(\boldsymbol{x}, \boldsymbol{y})$ and their segment as $[\boldsymbol{x}, \boldsymbol{y}]$. For a spectrahedron S, lets its interior be S° and its boundary ∂S. We represent a probability distribution π with a probability density function $\pi(x)$. When π is truncated to S the support of $\pi(x)$ is S. If π is log-concave, then $\pi(\boldsymbol{x}) \propto e^{-\alpha f(\boldsymbol{x})}$, where $f : \mathbb{R}^d \to \mathbb{R}$ a convex function. Finally, let $\mathcal{B}_n$ be the n-dimensional unit ball and denote by $\partial\mathcal{B}_n$ its boundary.

1.3 Background

1.3.1 Spectrahedra

Spectrahedra are probably the most well studied shapes after polyhedra. We can represent polyhedra as the intersection of the positive orthant with an affine subspace. Spectrahedra

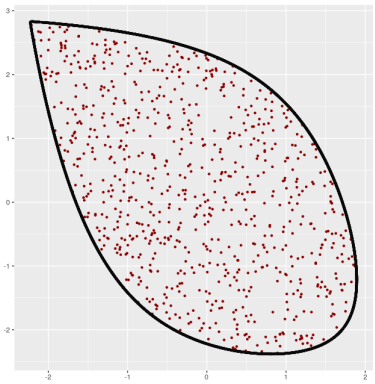

Figure 1: A randomly generated spectrahedron and uniform samples in its interior.

generalize polyhedra, in the sense that they are the intersection of the cone of positive semidefinite matrices with an affine space. Lets define positive definite matrices and spectrahedra.

Definition 1 (Positive definite matrix) *A matrix $\boldsymbol{A} \in \mathcal{S}^m$ is called positive definite, denoted by $\boldsymbol{A} \succeq 0$, if one of the following holds:*

- $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} > 0$ *for all nonzero* $\boldsymbol{x} \in \mathbb{R}^n$
- *all eigenvalues of* $\boldsymbol{A}$ *are positive*

Moreover, if one condition holds, so must the other. If $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0$ and all eigenvalues of $\boldsymbol{A}$ are non negative, then $\boldsymbol{A}$ is called positive semidefinite. We denote the set of positive definite $m \times m$ matrices by $\mathcal{S}^m_{++}$ and the set of positive semidefinite $m \times m$ matrices by $\mathcal{S}^m_{+}$.

Definition 2 (Spectrahedron) *A spectrahedron $S \subset \mathbb{R}^n$ is the feasible set of a linear matrix inequality. That is, if*

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1\boldsymbol{A}_1 + \cdots + x_n\boldsymbol{A}_n, \tag{1.1}$$

$\boldsymbol{A}_i$ are symmetric matrices in $\mathbb{R}^{m\times m}$, then $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{F}(\boldsymbol{x}) \succeq 0\}$.

We assume that S is bounded of dimension n. Spectrahedra are convex sets (Fig. 1) and every polytope is a spectrahedron, but not the opposite. They are the feasible regions of semidefinite programs [33] - as seen from the Definition 4 of semidefinite programs - in the same way that polyhedra are the feasible regions of linear programs.

Efficient methods for sampling points in spectrahedra are crucial for many applications, such as volume approximation [11], integration [26], semidefinite optimization [26, 19] and applications in robust control analysis [8, 7, 37]. To sample in the interior or on the boundary of S, we employ geometric random walks [41].

1.3.2 Geometric Random Walks

We want to address the problem of sampling a convex region. A way to achieve this, is to use geometric random walks. Intuitively, a geometric random walk on a body S, starts

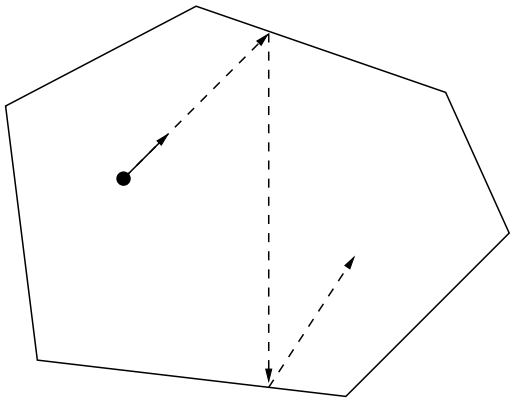

Figure 2: A step of the billiard walk in a polytope.

at some interior point and, at each step moves to a ”neighboring” point, chosen according to some distribution depending only on the current point. For example, in the so-called Billiard walk [17], starting from a point, we randomly and uniformly choose a direction, a distance to travel, and move towards that direction till we have covered the required distance. If we hit the boundary, we continue on a reflected trajectory (Fig 2).

More formally, geometric random walks are Markov chains. To sample under a specific distribution, we set up a random walk in a way, that its steady state is that desired distribution. A random sample is obtained after a sufficient number of steps. The complexity of a random walk depends on its mixing time —the number of steps required to bound the distance between the current and the stationary distribution— and the complexity of the basic geometric operations that we perform at each step of the walk (in the example of the Billiard walk, computing the intersection point of the trajectory with the boundary and the reflection); we also call the latter per-step complexity.

The majority of geometric random walks are defined for general convex bodies and are based on an oracle; usually in literature the membership oracle, though we in this work employ boundary oracles. A boundary oracle, given a convex body $\mathcal{S}$, a point $\boldsymbol{p} \in \mathcal{S}$ and a polynomial curve $\phi(t)$ (we restrict to univariate polynomials, with t serving as a time parameter), it outputs the distance we can travel till we reach the boundary of $\mathcal{S}$, walking on that curve. The complexity of boundary oracles dominates the complexity of a single step of the random walk.

There are also a few, e.g., Vaidya walk [9], sub-linear ball walk [28], specialized for polytopes. Most results on their analysis focus on convergence and mixing time, while the operations they perform at each step are defined abstractly and are enclosed in the corresponding oracle. That is why the complexity bounds involve the number of oracle calls.

To specialize a random walk for a family or representation of convex bodies one has to come up with efficient algorithms for the basic geometric operations to realize the (various) oracles. These operations should exploit the underlying geometric and algebraic properties and are of independent interest. Even more, they dominate the per-step complexity and they are crucial both for the overall complexity to sample a point from the target distribution and for an implementation.

The study of basic geometric operations to sample from non-linear convex objects finds its roots in non-linear computational geometry. During the last two decades, there are

combined efforts [6] to develop efficient algorithms for the basic operations (predicates) that are behind classical geometric algorithms, like convex hull, arrangements, Voronoi diagrams, to go beyond points and lines and handle curved objects.

To our knowledge, only the Hit and Run (HnR) random walk [35] has been studied for spectrahedra [7]. To exploit the various other walks, like Ball walk [41], Billiard walk [17], Hamiltonian Monte Carlo (HMC) [1] we need to provide geometric operations, such as the reflection of a curve at the boundary and computing the intersection point of a curve at the boundary.

We should mention that there is a gap [12, 5] between the theoretical worst case bounds for the mixing times and the practical performance of the random walk algorithms. Thus, it is not accurate to claim (for all the random walks) that the speed of convergence to the target distribution is the same for different families of convex bodies. Furthermore, there are random walks without known theoretical mixing times, such as Coordinate Directions HnR, billiard walk or HMC with reflections. To study them experimentally, the efficient realization of the corresponding oracles is crucial.

1.3.2.1 Previous Work on Geometric Random Walks in Spectrahedra

Sampling convex sets via random walks has attained a lot of interest in the last decades. Most of the works assume either convex sets or polytopes; [28] provides an overview of the state-of-the-art. Random walks on spectrahedra are studied in [31, 14], where it exploits the Hit and Run walk and the computation of the intersection reduces to a generalized eigenvalue problem.

The Billiard walk [17] is a general way of sampling in convex or non-convex shapes from the uniform distribution. A mathematical billiard consists of a domain $\mathcal{D}$ and a point-mass, moving freely in $\mathcal{D}$ [36]. When this point-mass hits the boundary, it performs a specular reflection, albeit without losing velocity. An application of billiards is the study of optical properties of conics [36, Sec. 4].

If the trajectory is not a line, but rather a parametric curve, then the intersection operation reduces to the polynomial eigenvalue problem (PEP). HMC with reflections requires this operation. PEP is a well-studied problem in computational mathematics, e.g., [38], and it appears in many applications. There are important results both for the perturbation analysis of PEP [38, 4, 15], as well as for the condition-based analysis of algorithms for the real and complex versions of PEP, if we assume random inputs [2, 3].

1.3.3 An algorithm for Polynomial Eigenvalue Problems

To estimate the complexity of the geometric operations defined in chapter 2 we need the complexity of PEP. The Polynomial Eigenvalue Problem (PEP) consists in computing $\lambda \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^m$ that satisfy the (matrix) equation

$$P(\lambda)\,\boldsymbol{x} = 0 \Leftrightarrow (\boldsymbol{B}_d\lambda^d + \cdots + \boldsymbol{B}_1\lambda + \boldsymbol{B}_0)\boldsymbol{x} = 0 \ , \tag{1.2}$$

where $\boldsymbol{B}_i \in \mathbb{R}^{m\times m}$. We further assume that $\boldsymbol{B}_d$ and $\boldsymbol{B}_0$ are invertible. In general, there are $\delta = m\,d$ values of λ. We refer the reader to [38, 39] for a thorough exposition of PEP.

One approach for solving PEP is to linearalize the problem and to express λ's as the eigenvalues of a bigger matrix. For this we transform Eq. (1.2) into a linear pencil of

dimension δ. Following closely [4], the *Companion Linearization* consists in solving the generalized eigenvalue problem $\boldsymbol{C}_0 - \lambda \boldsymbol{C}_1$, where

$$\boldsymbol{C}_0 = \begin{bmatrix} \boldsymbol{B}_d & 0 & \cdots & 0 \\ 0 & \boldsymbol{I}_m & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{I}_m \end{bmatrix} \text{ and } \boldsymbol{C}_1 = \begin{bmatrix} \boldsymbol{B}_{d-1} & \boldsymbol{B}_{d-2} & \cdots & \boldsymbol{B}_0 \\ -\boldsymbol{I}_m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & -\boldsymbol{I}_m & 0 \end{bmatrix},$$

and $\boldsymbol{I}_m$ denotes the $m \times m$ identity matrix. The eigenvectors $\boldsymbol{x}$ and $\boldsymbol{z}$ are related $\boldsymbol{z} = [1, \lambda, \ldots, \lambda^{d-2}, \lambda^{d-1}]^\top \otimes \boldsymbol{x}$.

To obtain an exact algorithm for PEP we exploit the assumption that $\boldsymbol{B}_d$ is invertible to transform the problem to the following classical eigenvalue problem $(\lambda \boldsymbol{I}_d - \boldsymbol{C}_2)\boldsymbol{z} = 0$, where

$$\boldsymbol{C}_2 = \begin{bmatrix} \boldsymbol{B}_{d-1}\boldsymbol{B}_d^{-1} & \boldsymbol{B}_{d-2}\boldsymbol{B}_d^{-1} & \cdots & \boldsymbol{B}_0\boldsymbol{B}_d^{-1} \\ -\boldsymbol{I}_m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & -\boldsymbol{I}_m & 0 \end{bmatrix}.$$

The eigenvectors are roots of the characteristic polynomial of $\boldsymbol{C}_2$. Therefore, the problem is to compute the eigenvalues of $\boldsymbol{C} \in \mathbb{R}^{\delta \times \delta}$. From a complexity point of view the best algorithm to compute the eigenvalues relies on computing the roots of the characteristic polynomial [34]. We also follow this approach. However, in practise other methods are more efficient and stable.

Lemma 1.3.1 *Consider a PEP of degree d, involving matrices of dimension $m \times m$, with integer elements of bitsize at most τ, see (1.2). There is a randomized algorithm for computing the eigenvalues and the eigenvectors of PEP up to precision $\epsilon = 2^{-L}$, in $\widetilde{\mathcal{O}}_B(\delta^{\omega+3}L)$, where $\delta = md$ and $L = \Omega(\delta^3\tau)$. The arithmetic complexity is $\widetilde{\mathcal{O}}(\delta^{2.697} + \delta \lg(1/\epsilon))$.*

Proof 1 (Proof of Lemma 1.3.1) *We can compute the characteristic polynomial of an $N \times N$ matrix M in $\widetilde{\mathcal{O}}_B(N^{2.697+1} \lg\|M\|)$ using a randomized algorithm, see [21] and references therein. Here $\|M\|$ denotes the largest entry in absolute value. In our case, the elements of $\boldsymbol{C}_2$ have bitsize $\widetilde{\mathcal{O}}(\delta\tau)$ and its characteristic polynomial is of degree d and coefficient bitsize $\widetilde{\mathcal{O}}_B(\delta^2\tau)$. We compute it in $\widetilde{\mathcal{O}}_B(\delta^{2.697+1}\delta\tau) = \widetilde{\mathcal{O}}_B(\delta^{4.697}\tau)$. We isolate all its real roots in $\widetilde{\mathcal{O}}_B(\delta^5 + \delta^4\tau)$ [29]; they correspond to the real eigenvalues of PEP. We can decrease the width of the isolating interval by a factor of $\epsilon = 2^{-L}$ for* all *the roots in $\widetilde{\mathcal{O}}_B(\delta^3\tau + \delta L)$ [30]. Thus, the overall complexity is $\widetilde{\mathcal{O}}_B(\delta^5 + \delta^{4.697}\tau + \delta L)$.*

It remains to compute the corresponding eigenvectors. For each eigenvalue λ we can compute the corresponding eigenvector $\boldsymbol{z}$ by performing Gaussian elimination and back substitution to the (augmented) matrix $[\lambda \boldsymbol{I}_\delta - \boldsymbol{C}_2 \,|\, \boldsymbol{0}]$. We can do this with $\widetilde{\mathcal{O}}(\delta^\omega)$ arithmetic operations. However, as λ is a root of the characteristic polynomial we have to perform operations with algebraic numbers, which a highly non-trivial task and it is not clear what is the number of bits that we need to compute the elements of $\boldsymbol{z}$ correctly and to recover $\boldsymbol{x}$. For this task we employ the separation bounds for polynomial system adopted to the problem of eigenvector computation [16]. We need, as in the case of eigenvalues, $\widetilde{\mathcal{O}}_B(\delta^4 + \delta^3\tau)$ bits to isolate the coordinates of the eigenvectors. To decrease the width of the corresponding isolating intervals by a factor of $\epsilon = 2^{-L}$, then the number of bits becomes $\widetilde{\mathcal{O}}_B(\delta^4 + \delta^3\tau + L)$. Thus, we compute the eigenvectors in $\widetilde{\mathcal{O}}_B(\delta^\omega(\delta^4 + \delta^3\tau + L)) = \widetilde{\mathcal{O}}_B(\delta^{\omega+4} + \delta^{\omega+3}\tau + \delta^\omega L)$.

For the arithmetic complexity we proceed as follows: We compute the characteristic polynomial in $\widetilde{\mathcal{O}}(\delta^{2.697})$, we approximate its roots up to ϵ in $\widetilde{\mathcal{O}}(\delta \lg(1/\epsilon))$. Finally, we compute the eigenvectors with $\widetilde{\mathcal{O}}(\delta^{\omega})$ arithmetic operations. So the overall cost is $\widetilde{\mathcal{O}}(\delta^{2.697} + \delta \lg(1/\epsilon))$.

We do not claim that the algorithm that we presented for PEP is the best algorithm to use in practice. There are several superior numerical algorithms for this task. However, our approach is convenient to deduce Boolean complexity estimates.

1.3.4 Semidefinite Programming

Semidefinite optimization is a subarea of convex optimization, one of great theoretical and practical interest. We could think of it, informally, as a generalization of linear programming, where the decision variables are symmetric matrices and, the inequalities can be perceived as matrices being positive semidefinite.

Definition 3 (Semidefinite Programming - Primal Form) *A semidefinite program has the following form:*

$$\begin{array}{ll} \textit{Minimize} & \langle \boldsymbol{C}, \boldsymbol{X} \rangle \\ \textit{subject to} & \langle \boldsymbol{A_i}, \boldsymbol{X} \rangle \leq b_i, \; i = 1, \cdots, n \\ & \boldsymbol{X} \succeq 0 \end{array}$$

where $\boldsymbol{C}, \boldsymbol{A_i}, \boldsymbol{X} \in \mathcal{S}^m$ and $\langle \boldsymbol{A}, \boldsymbol{B} \rangle$ denotes the operation $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} B_{i,j} = Tr\left(\boldsymbol{A}^\top \boldsymbol{B}\right)$.

The feasible region of a semidefinite program is a spectrahedron; remember, in linear programming the feasible region was a polyhedron. In the remaining of this work,we will focus on the dual form of semidefinite programs. The reason is, that in its dual representation, the constraint is (without loss of generality) a linear matrix inequality, which is a compact way to describe spectrahedra and will be of use when developing the geometric operations.

Definition 4 (Semidefinite Program - Dual Form) *The dual representation of a semidefinite program is:*

$$\begin{array}{ll} \textit{Minimize} & \boldsymbol{c}^\top \boldsymbol{x} \\ \textit{subject to} & \boldsymbol{A_0} + \sum_{i=1}^{n} x_i \boldsymbol{A_i} \succeq 0 \end{array}$$

where $\boldsymbol{c}, \boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{A_i} \in \mathcal{S}^m$, $i = 0, \cdots, n$. Note that the feasible region of this program, described by $\boldsymbol{A_0} + \sum_{i=1}^{n} x_i \boldsymbol{A_i} \succeq 0$, is a n-dimensional spectrahedron.

2. BASIC GEOMETRIC OPERATIONS

Our toolbox for computations with spectrahedra and implementing random walks, consists of three basic geometric operations: membership, intersection, and reflection. For a spectrahedron S, membership decides if a point is inside S, intersection computes the intersection of an algebraic curved trajectory $\mathcal{C}$ with the ∂S, and reflection computes the reflection of an algebraic curved trajectory when it hits ∂S. We need the last two operations because random walks can move along non-linear trajectories inside convex bodies. For the ones that we consider, the trajectories are parametric polynomial curves, of various degrees. To compute with these curves we need to solve a polynomial eigenvalue problem (PEP).

2.1 membership

The operation membership$(\boldsymbol{F}, \boldsymbol{p})$ decides if a point $\boldsymbol{p}$ lies in the interior of a spectrahedron $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{F}(\boldsymbol{x}) \succeq 0\}$. For this, first, we construct the matrix $\boldsymbol{F}(\boldsymbol{p})$. Next, if the matrix is positive definite, then $\boldsymbol{p} \in S^\circ$, if it is positive semidefinite, then $\boldsymbol{p} \in \partial S$, and otherwise $\boldsymbol{p} \in \mathbb{R}^n \setminus S$. The pseudo-code appears in Alg. 1.

Algorithm 1: membership$(\boldsymbol{F}, \boldsymbol{p})$

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ representing a spectrahedron S and a point $\boldsymbol{p} \in \mathbb{R}^n$.
Output: true if $\boldsymbol{p} \in S$, false otherwise.

1 $\lambda_{min} \leftarrow$ smallest eigenvalue of $\boldsymbol{F}(\boldsymbol{p})$;
2 **if** $\lambda_{min} \geq 0$ **then return** true ;
3 **return** false ;

Lemma 2.1.1 *Alg. 1,* membership$(\boldsymbol{F}, \boldsymbol{p})$*, requires* $\widetilde{\mathcal{O}}(nm^2 + m^{2.697})$ *arithmetic operations. If* $\boldsymbol{F}$ *and* $\boldsymbol{p}$ *have integers elements of bitsize at most* τ*, resp.* σ*, then the bit complexity is* $\widetilde{\mathcal{O}}_B((nm^2 + m^{3.697})(\tau + \sigma))$*.*

Proof 2 (Proof of Lemma 2.1.1) *We construct* $\boldsymbol{F}(\boldsymbol{p})$ *in* $\mathcal{O}(nm^2)$*. Then, with* $\mathcal{O}(m^{2.697})$ *operations we compute its characteristic polynomial [21] and in* $\widetilde{\mathcal{O}}(m)$ *we decide if it has negative roots, for example by solving [29] or using fast sub-resultant algorithms [22, 24]. For the bit complexity bound, the construction costs* $\widetilde{\mathcal{O}}_B(nm^2(\tau + \sigma)$ *and computation of the characteristic polynomial* $\widetilde{\mathcal{O}}_B(m^{2.697+1}(\tau + \sigma))$ *using a randomized algorithm [21]. We test for negative roots, and thus eigenvalues, in* $\widetilde{\mathcal{O}}_B(m^2 n(\tau + \sigma))$ *[24].*

2.2 intersection

Consider a parametric polynomial curve $\mathcal{C}$ such that it has a non-empty intersection with a spectrahedron S. Assume that the value of the parameter $t = 0$ corresponds to a point, $\boldsymbol{p}_0$, that lies in $\mathcal{C} \cap S^\circ$. Further assume that the part of $\mathcal{C}$ that $\boldsymbol{p}_0$ lies on, intersects the boundary of S transversally at two points, say $\boldsymbol{p}_-$ and $\boldsymbol{p}_+$. The operation intersection computes the

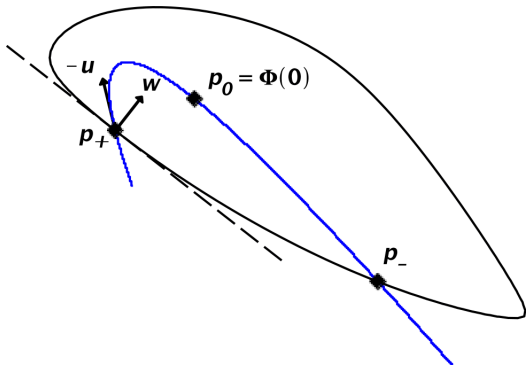

Figure 3: A spectrahedron S described by $\boldsymbol{F}(\boldsymbol{x})$ and a parameterized curve Φ. The point $\boldsymbol{p}_0 = \Phi(0)$ lies in the interior of S, and the points $\boldsymbol{p}_+ = \Phi(t_+)$ and $\boldsymbol{p}_- = \Phi(t_-)$ on the boundary. The vector $\boldsymbol{w}$ is the surface normal of ∂S at $\boldsymbol{p}_+$ and $\boldsymbol{u}$ the direction of Φ at time $t = t_+$.

parameters, t_- and t_+, corresponding to these two points. Fig. 3 illustrates this discussion and the pseudo-code of intersection appears in Alg. 2.

To prove correctness and estimate its complexity we proceed as follows: As before, S is the feasible region of an LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$. Consider the real trace of a polynomial curve $\mathcal{C}$, with parametrization

$$\begin{array}{rccl} \Phi & : & \mathbb{R} & \to & \mathbb{R}^n \\ & & t & \mapsto & \Phi(t) := (p_1(t), \ldots, p_n(t)), \end{array} \tag{2.1}$$

where $p_i(t) = \sum_{j=0}^{d_i} p_{i,j} t^j$ are univariate polynomials in t of degree d_i, for $i \in [m]$. Also let $d = \max_{i\in[m]}\{d_i\}$. If the coefficients of the polynomials are integers, then we further assume that the maximum coefficient's bitsize is bounded by τ.

As t varies over the real line, there may be several disjoint intervals, for which the corresponding part of $\mathcal{C}$ lies in S°. We aim to compute the endpoints, t_- and t_+, of a maximum interval containing $t = 0$. Let $\boldsymbol{p}_0 = \Phi(0)$; by assumption it holds $\boldsymbol{F}(\Phi(0)) = \boldsymbol{F}(\boldsymbol{p}_0) \succ 0$.

The input of intersection (Alg. 2) is $\boldsymbol{F}$, the LMI representation of S, and $\Phi(t)$, the polynomial parametrization of $\mathcal{C}$. Its crux is a routine, PEP, that solves a polynomial eigenvalue problem. The following lemma exploits this relation.

Algorithm 2: intersection$(\boldsymbol{F}, \Phi(t))$

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S and a parametrization $\Phi(t)$ of a polynomial curve $\mathcal{C}$

Require: $\Phi(0) \in S^\circ$

Output : t_-, t_+ s.t. $\Phi(t_-), \Phi(t_+) \in \partial S$

4 $T := \{t_1 \le t_2 \le \cdots \le t_\ell\} \leftarrow$ PEP$(\boldsymbol{F}(\Phi(t)))$;

5 $t_- \leftarrow \max\{t \in T \,|\, t < 0\}$; *// max neg polynomial eigenvalue*

6 $t_+ \leftarrow \min\{t \in T \,|\, t > 0\}$; *// min pos polynomial eigenvalue*

7 **return** t_-, t_+;

Lemma 2.2.1 (PEP and $S \cap \mathcal{C}$) *Consider the spectrahedron $S = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, \boldsymbol{F}(\boldsymbol{x}) \succeq 0\}$. Let $\Phi : \mathbb{R} \to \mathbb{R}^n$ be a parametrization of a polynomial curve $\mathcal{C}$, such that $\Phi(0) \in S^\circ$. Let $[t_-, t_+]$ be the maximum interval containing 0, such that the corresponding part of $\mathcal{C}$ lies in S. Then, t_-, resp. t_+, is the maximum negative, resp. minimum positive, polynomial eigenvalue of $\boldsymbol{F}(\Phi(t))\boldsymbol{x} = 0$, where $\boldsymbol{F}(\Phi(t)) = \boldsymbol{B}_0 + t\boldsymbol{B}_1 + \cdots + t^d \boldsymbol{B}_d$.*

Proof 3 (Proof of Lemma 2.2.1) *The composition of $\boldsymbol{F}(\boldsymbol{x})$ and $\Phi(t)$ gives*

$$\boldsymbol{F}(\Phi(t)) = \boldsymbol{A}_0 + p_1(t)\boldsymbol{A}_1 + \cdots + p_n(t)\boldsymbol{A}_n. \tag{2.2}$$

We rewrite (2.2) by grouping the coefficients for each t^k, $i \in [d]$, then

$$\boldsymbol{F}(\Phi(t)) = \boldsymbol{B}_0 + t\boldsymbol{B}_1 + \cdots + t^d\boldsymbol{B}_d, \tag{2.3}$$

where $\boldsymbol{B}_k = \sum_{j=0}^{n} p_{j,k}\,\boldsymbol{A}_j$, for $0 \leq k \leq d$. We use the convention that $p_{j,k} = 0$, when $k > d_j$.

For $t = 0$, it holds, by assumption, that $\boldsymbol{F}(\Phi(0)) = \boldsymbol{B}_0 \succ 0$; that is the point $\Phi(0)$ is in the interior S. Actually, for any $\boldsymbol{x} \in S^\circ$ it holds $\boldsymbol{F}(\boldsymbol{x}) \succ 0$. On the other hand, if $\boldsymbol{x} \in \partial S$, then $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$. Our goal is to compute the maximal interval $[t_-, t_+]$ that contains 0 and for every t in it, we have $\boldsymbol{F}(\Phi(t)) \succeq 0$.

Starting from the point $\Phi(0)$, by varying t, we move at the trajectory that $\mathcal{C}$ defines (in both directions) until we hit the boundary of S. When we hit ∂S, the matrix $\boldsymbol{F}(\Phi(t))$ is not strictly definite anymore. Thus, its determinant vanishes.

Consider the function $\theta : \mathbb{R} \to \mathbb{R}$, where $\theta(t) = \det \boldsymbol{F}(\Phi(t))$ is a univariate polynomial in t. If a point $\Phi(t)$ is on the boundary of the spectrahedron, then $\theta(t) = 0$. We opt to compute t_- and t_+, such that $t_- \leq 0 \leq t_+$ and $\theta(t_-) = \theta(t_+) = 0$. At $t = 0$, $\theta(0) > 0$ and the graph of θ is above the t-axis. So $\mathcal{C}$ intersects the boundary when the graph of θ touches the t-axis for the first time at $t_1 \leq 0 \leq t_2$. It follows that $t_- = t_1$ and $t_+ = t_2$ are the maximum negative and minimum positive roots of θ, or equivalently the corresponding polynomial eigenvalues of $\boldsymbol{F}(\Phi(t))$.

2.2.1 Complexity of intersection

We have to construct PEP and solve it. If $\Phi(t)$ has degree d, then $\boldsymbol{F}(\Phi(t)) = \boldsymbol{B}_0 + t\boldsymbol{B}_1 + \cdots + t^d\boldsymbol{B}_d$. This construction costs $\mathcal{O}(dnm^2)$ operations. The solving phase, from Lemma 1.3.1 requires $\widetilde{\mathcal{O}}((md)^{2.697} + md \lg L)$ arithmetic operations and dominates the complexity bound of the operation.

2.3 reflection

The reflection operation (Alg. 3) takes as input an LMI $\boldsymbol{F}$ representation of a spectrahedron S and a polynomial curve $\mathcal{C}$, given by a parametrization Φ. Assume that $t = 0$ corresponds to a point $\Phi(0) \in S^\circ \cap \mathcal{C}$. Starting from $t = 0$, we increase t along the positive real semi-axis. As t changes, we move along the curve $\mathcal{C}$ through $\Phi(t)$, until we hit the boundary of S at the point $\boldsymbol{p}_+ := \Phi(t_+) \in \partial S$, for some $t_+ > 0$. Then, a specular reflection occurs at this point with direction $\boldsymbol{s}_+$; this is the reflected direction. We output t_+ and $\boldsymbol{s}_+$. Fig. 3 depicts the procedure.

The boundary of S, ∂S, with respect to the Euclidean topology, is a subset of the real algebraic set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \det(\boldsymbol{F}(\boldsymbol{x})) = 0\}$. The latter is a real hypersurface defined by one (determinantal) equation. For any $\boldsymbol{x} \in \partial S$ we have $\operatorname{rank}(\boldsymbol{F}(\boldsymbol{x})) \leq m - 1$. We assume that $\boldsymbol{p}_+ = \Phi(t_+)$ is such that $\operatorname{rank}(\boldsymbol{F}(\boldsymbol{p}_+)) = m - 1$. The normal direction at a point $\boldsymbol{p} \in \partial S$, is the gradient of $\det \boldsymbol{F}(\boldsymbol{p})$.

We compute the reflected direction using the following formula

$$\boldsymbol{s}_+ = \boldsymbol{u} - \tfrac{2}{|\boldsymbol{w}|^2}\langle \boldsymbol{u}, \boldsymbol{w}\rangle \, \boldsymbol{w}, \tag{2.4}$$

where $\boldsymbol{w}$ is the normalized gradient vector at the point $\Phi(t_+)$ and $\boldsymbol{u} = \frac{d\Phi}{dt}(t_+)$ is the direction of the trajectory at this point. We illustrate the various vectors in Fig. 3.

Algorithm 3: reflection $(\boldsymbol{F}, \Phi(t))$

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S and a parametrization $\Phi(t)$ of a polynomial curve $\mathcal{C}$.

Require: (i) $\Phi(0) \in S^\circ$
(ii) $\mathcal{C}$ intersects ∂S transversally at a smooth point.

Output : t_+ such that $\Phi(t_+) \in \partial S$ and the direction of the reflection, $\boldsymbol{s}_+$, at this point.

8 $t_-, t_+ \leftarrow$ intersection $(\boldsymbol{F}, \Phi(t))$;
9 $\boldsymbol{w} \leftarrow \nabla \det \boldsymbol{F}(\Phi(t_+))$;
10 $\boldsymbol{w} \leftarrow \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$; *// Normalize* $\boldsymbol{w}$
11 $\boldsymbol{s}_+ \leftarrow \frac{d\Phi}{dt}(t_+) - 2\,\langle \nabla \frac{d\Phi}{dt}(t_+), \boldsymbol{w}\rangle \, \boldsymbol{w}$;
12 **return** t_+, $\boldsymbol{s}_+$;

Lemma 2.3.1 (Gradient of $\det \boldsymbol{F}(\boldsymbol{x})$) *Assume that $\boldsymbol{x} \in \partial S$ and the rank of the $m \times m$ matrix $\boldsymbol{F}(\boldsymbol{x})$ is $m-1$. Then*

$$\nabla \det(\boldsymbol{F}(\boldsymbol{x})) = c \cdot (\boldsymbol{v}^\top \boldsymbol{A}_1 \boldsymbol{v}, \cdots, \boldsymbol{v}^\top \boldsymbol{A}_n \boldsymbol{v}), \tag{2.5}$$

where $c = \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{|\boldsymbol{v}|^2}$, $\mu(\boldsymbol{F}(\boldsymbol{x}))$ is the product of the nonzero eigenvalues of $\boldsymbol{F}(\boldsymbol{x})$, and $\boldsymbol{v}$ is a non-trivial vector in the kernel of $\boldsymbol{F}(\boldsymbol{x})$. If $\operatorname{rank}(\boldsymbol{F}(\boldsymbol{x})) \leq m-2$, then the gradient is the zero.

To prove lemma 2.3.1 we will need the following lemmas.

Lemma 2.3.2 (Partial Derivative of Determinant) *Let $\boldsymbol{A}$ be a symmetric $m \times m$ matrix. Then*

$$\frac{\partial \det \boldsymbol{A}}{\partial A_{ij}} = c_{ij}$$

where c_{ij} the cofactor of A_{ij}.

Proof 4 *From Laplace expansion:*

$$\det \boldsymbol{A} = \sum_{j=1}^{m} A_{ij} c_{ij}$$

Notice that $c_{1j}, \cdots, c_{mj}$ are independent of A_{ij}, so we have

$$\frac{\partial \det \boldsymbol{A}}{\partial A_{ij}} = c_{ij}$$

Lemma 2.3.3 *Let* $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1\boldsymbol{A}_1 + \cdots + x_n\boldsymbol{A}_n$. *Then*

$$\frac{\partial \det \boldsymbol{F}(\boldsymbol{x})}{\partial x_k} = Tr\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k)\right)$$

Proof 5 *The function* $\det \boldsymbol{F}(\boldsymbol{x})$ *is the composition of* $\det \boldsymbol{A}$ *and* $\boldsymbol{A} = \boldsymbol{F}(\boldsymbol{x})$, *so from Lemma 2.3.2 and the chain rule:*

$$\frac{\partial \det \boldsymbol{F}(\boldsymbol{x})}{\partial x_k} = \sum_{i=1}^{m}\sum_{j=1}^{m} \frac{\partial \det \boldsymbol{F}}{\partial \boldsymbol{F}_{ij}} \cdot \frac{\partial \boldsymbol{F}_{ij}}{\partial x_k} = \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} A_{ij}^k = Tr\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right)$$

where A_{ij}^k *the* ij*-th element of matrix* $\boldsymbol{A}_k$

Lemma 2.3.4 (Adjoint Matrix of $\boldsymbol{A}$**)** *Let* $\boldsymbol{A}$ *be a* $m \times m$ *matrix of rank* $r(\boldsymbol{A}) = m-1$. *Then*

$$\boldsymbol{A}^* = \mu(\boldsymbol{A}) \frac{\boldsymbol{v}\boldsymbol{u}^\top}{\boldsymbol{u}^\top \boldsymbol{v}}$$

where $\mu(\boldsymbol{A})$ *is the product of the* $m-1$ *non-zero eigenvalues of* $\boldsymbol{A}$, *and* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *satisfy* $\boldsymbol{A}\boldsymbol{v} = \boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{0}$ *(see chapter 3.2 in [27]).*

Proof 6 (Proof of Lemma 2.3.1) *From Lemma 2.3.3:*

$$\frac{\partial \det \boldsymbol{F}(\boldsymbol{x})}{\partial x_k} = Tr\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right) \tag{2.6}$$

If $\operatorname{rank}(\boldsymbol{F}(\boldsymbol{x})) \leq -2$, *then* $\boldsymbol{F}(\boldsymbol{x})^*$ *is the zero matrix. Supposing* $\operatorname{rank}(\boldsymbol{F}(\boldsymbol{x})) = m-1$, *from Lemma 2.3.4:*

$$Tr\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right) = Tr\left(\mu(\boldsymbol{F}(\boldsymbol{x})) \frac{\boldsymbol{v}\boldsymbol{u}^\top}{\boldsymbol{u}^\top \boldsymbol{v}} \boldsymbol{A}_k\right) =$$
$$\frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{\boldsymbol{u}^\top \boldsymbol{v}} \cdot Tr\left(\boldsymbol{v}\boldsymbol{u}^\top \boldsymbol{A}_k\right) = \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{\boldsymbol{u}^\top \boldsymbol{v}} \cdot \boldsymbol{u}^\top \boldsymbol{A}_k \boldsymbol{v}$$

but since $\boldsymbol{F}(\boldsymbol{x})$ *is symmetric, we can choose* $\boldsymbol{v} = \boldsymbol{u}$, *so:*

$$\frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{\boldsymbol{u}^\top \boldsymbol{v}} \cdot \boldsymbol{u}^\top \boldsymbol{A}_k \boldsymbol{v} = \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{|\boldsymbol{v}|^2} \cdot \boldsymbol{v}^\top \boldsymbol{A}_k \boldsymbol{v}$$

reflection exploits Lemma 2.3.1. Nevertheless, it is not necessary to perform all computations indicated by the lemma. Since, we will normalize the resulting vector and we do not need its actual direction (internal or external), we can omit the computation of c. Moreover, the nonzero vector $\boldsymbol{v}$ s.t. $\boldsymbol{F}(\boldsymbol{p})\boldsymbol{v} = 0$, corresponds to the eigenvector w.r.t. the eigenvalue t_+ from the PEP (Lem. 2.2.1). This holds because $\boldsymbol{p} = \Phi(t_+) \in \partial S$ and thus $\det \boldsymbol{F}(\Phi(t_+)) = 0$.

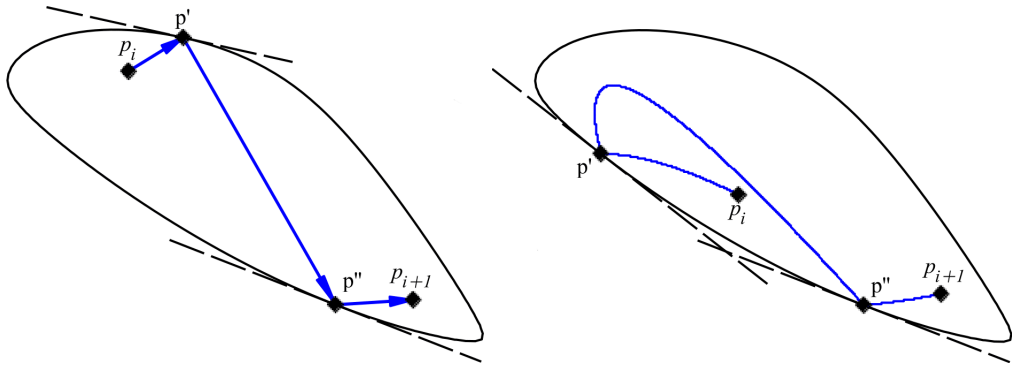

Figure 4: The i-th step of the W-Billard [5] (left) and of the W-HMC-r [6] (right) random walks.

We compute the eigenvalues of PEP up to some precision. Since matrix-vector multiplication is backward stable, a small perturbation on $\boldsymbol{v}$ does not affect the computation of each coordinate of $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ [40, p. 104]. We quantify the accuracy of the computed $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ using floating point arithmetic as follows:

Lemma 2.3.5 *The relative error of each coordinate of the gradient given in Lemma 2.3.1 when we compute it using floating point arithmetic with machine epsilon ϵ_M is $\mathcal{O}(\frac{\epsilon_M}{\sigma_{\max}(A_i)})$, for $i \in [n]$, where $\sigma_{\max}$ is the largest singular value of A_i.*

Proof 7 (Proof of Lemma 2.3.5) *Let $A \in \mathbb{R}^{m\times m}$ be a symmetric matrix and consider the map $f : \boldsymbol{v} \mapsto \boldsymbol{v}^T A \boldsymbol{v}$. The relative condition number of f as defined in [40, p. 90] is*

$$k(\boldsymbol{v}) = \frac{||J(\boldsymbol{v})||}{||f(\boldsymbol{v})||/||\boldsymbol{v}||} = 2\frac{||A\boldsymbol{v}||}{\boldsymbol{v}^T A_i \boldsymbol{v}} = 2\frac{\sigma_{\max}(A)}{\sigma^2_{\max}(A)} = \frac{2}{\sigma_{\max}(A)}$$

where $J(\cdot)$ is the Jacobian of f. According to Theorem 15.1 in [40, p. 111], since matrix-vector multiplication is backward stable, the relative error of each coordinate in the gradient computation of Lemma 2.3.1 is $\mathcal{O}(\frac{\epsilon_M}{\sigma_{\max}(A_i)})$, $i = 1, \ldots, n$.

2.3.1 Complexity of reflection

As mentioned, for $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ we just need to compute $(\boldsymbol{v}^\top \boldsymbol{A}_1 \boldsymbol{v}, \cdots, \boldsymbol{v}^\top \boldsymbol{A}_n \boldsymbol{v})$. If we have already computed $\boldsymbol{v}$, then we need $O(nm^2)$ operations. Computing the derivative of $\Phi(t)$ is straightforward, since Φ is a univariate polynomial. Taking into account the complexity of intersection, the total complexity for reflection is $\widetilde{\mathcal{O}}((md)^{2.697} + md \lg L + dnm^2 + nm^2) = \widetilde{\mathcal{O}}((md)^{2.697} + md \lg L + dnm^2)$.

2.4 An example in 2D

Consider a spectrahedron S in the plane (Fig. 4), given by an LMI $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1 \boldsymbol{A}_1 + x_2 \boldsymbol{A}_2$. The spectrahedron was randomly generated as in [14]. Due to space considerations, the entries of the matrices are rounded to the first decimal.

$$\boldsymbol{A_0} = \begin{bmatrix} 16.7 & 3.7 & 12.3 & 8.7 & 5.1 & 10.4 \\ 3.7 & 9.4 & 2.3 & 4 & -2.3 & -1 \\ 12.3 & 2.3 & 26.8 & 18.7 & 7.1 & 16.7 \\ 8.7 & 4 & 18.7 & 20 & 3.7 & 12.3 \\ 5.1 & -2.3 & 7.1 & 3.7 & 6.1 & 5.4 \\ 10.4 & -1 & 16.7 & 12.3 & 5.4 & 18.7 \end{bmatrix} \tag{2.7}$$

$$\boldsymbol{A_1} = \begin{bmatrix} 0.5 & -0.4 & 2.7 & 0 & 0 & \\ -0.4 & 1.4 & -0.2 & 0 & 0 & 0 \\ 2.7 & -0.2 & 1.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & -0.4 & 2.7 \\ 0 & 0 & 0 & -0.4 & 1.4 & -0.2 \\ 0 & 0 & 0 & 2.7 & -0.2 & 1.7 \end{bmatrix} \tag{2.8}$$

$$\boldsymbol{A_2} = \begin{bmatrix} 2.6 & -0.1 & 3 & 0 & 0 & 0 \\ -0.1 & 1 & -0.1 & 0 & 0 & 0 \\ 3 & -0.1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.6 & -0.1 & 3 \\ 0 & 0 & 0 & -0.1 & 1 & -0.1 \\ 0 & 0 & 0 & 3 & -0.1 & -1 \end{bmatrix} \tag{2.9}$$

Starting from point $\boldsymbol{p}_0 = (-1, 1)^\top$, we walk along the line L with parametrization: $\Phi(t) = \boldsymbol{p}_0 + t\boldsymbol{u}$, where $\boldsymbol{u} = (1.3, 0.8)^\top$. Then, intersection finds the intersection of S with L, by solving the degree one PEP, $(\boldsymbol{B}_0 + t\boldsymbol{B}_1)\boldsymbol{x} = 0$, where $\boldsymbol{B}_0 = \boldsymbol{F}(\boldsymbol{p}_0)$ and $\boldsymbol{B}_1 = u_1\boldsymbol{A}_1 + u_2\boldsymbol{A}_2$. Acquiring $t_- = -0.8$ and $t_+ = 0.5$, we get the intersection point $\boldsymbol{p_1}$, which corresponds to $\boldsymbol{p}_0 + t_+\boldsymbol{u} = (-0.3, 1.4)^\top$.

To compute the direction of the trajectory, immediately after we reflect on the boundary of S at $\boldsymbol{p_1}$, reflection computes

$$\boldsymbol{w} = \frac{\nabla \det \boldsymbol{F}(\Phi(t_+))}{|\nabla \det \boldsymbol{F}(\Phi(t_+))|} = (\boldsymbol{v}^\top \boldsymbol{A}_1 \boldsymbol{v}, \boldsymbol{v}^\top \boldsymbol{A}_2 \boldsymbol{v})^\top = (-0.2, -1)^\top, \tag{2.10}$$

where $\boldsymbol{v}$ is the eigenvector of $(\boldsymbol{B}_0 + t\boldsymbol{B}_1)\boldsymbol{x} = 0$, with eigenvalue t_+. The reflected direction is $\boldsymbol{u'} = \boldsymbol{u} - 2\langle \boldsymbol{u}, \boldsymbol{w} \rangle \boldsymbol{w} = (0.8, -1.3)^\top$.

3. SOME RANDOM WALKS

Using the basic geometric operations of Sec. 2, we implement and analyze three random walks for spectrahedra: Hit and Run (W-HnR) , Billiard Walk (W-Billard), and Hamiltonian Monte Carlo with reflections (W-HMC-r). In Table 1, we present the per-step arithmetic complexity for each random walk.

3.1 Hit and Run

W-HnR (Alg. 4) is a random walk that samples from any probability distribution π, truncated to a convex body K; in our case a spectrahedron S. However, for its mixing time, there exist bounds only when π is log-concave distribution (e.g. the uniform distribution), which is $\widetilde{\mathcal{O}}(n^3)$. At the i-th step, W-HnR chooses uniformly at random a (direction of a) line ℓ, passing from its current position $\boldsymbol{p}_i$. Let $\boldsymbol{p}_1, \boldsymbol{p}_2$ be the intersection points of ℓ with S. Let π_ℓ be the restriction of π on the segment $[\boldsymbol{p_2}, \boldsymbol{p_2}]$. Then, we choose $\boldsymbol{p}_{i+1}$ from $[\boldsymbol{p_1}, \boldsymbol{p_2}]$ w.r.t. the distribution π_ℓ.

Algorithm 4: Hit-and-Run_Walk (W-HnR)

Input : LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S & a point $\boldsymbol{p}_i$.
Require: $\boldsymbol{p}_i \in S$
Output : The point $\boldsymbol{p}_{i+1}$ of the $(i+1)$-th step of the walk.

$\boldsymbol{v} \leftarrow_R \mathcal{U}(\partial\mathcal{B}_n)$; *// choose direction*
$\Phi(t) := \boldsymbol{p}_i + t\boldsymbol{v}$; *// define trajectory*
$t_-, t_+ \leftarrow$ intersection $(\boldsymbol{F}, \Phi(t))$;
$\boldsymbol{p}_{i+1} \leftarrow_R [\boldsymbol{p}_i + t_-\boldsymbol{v},\ \boldsymbol{p}_i + t_+\boldsymbol{v}]$ w.r.t. π_ℓ;
return $\boldsymbol{p}_{i+1}$;

The per-step complexity of W-HnR is dominated by the intersection, which requires $\mathcal{O}(nm^2)$ operations for the construction of the PEP and $\widetilde{\mathcal{O}}(m^{2.697} + m \lg 1/\epsilon)$ for solving it, where we want to approximate the intersection point up to a factor or 2^{-L}.

There is a variation of W-HnR, the *coordinate directions Hit and Run* (W-CHnR) [35], in which the direction vector is chosen randomly and uniformly from the vector basis $\{\boldsymbol{e}_i,\ i \in [n]\}$. In W-CHnR, for every step aside the first, the construction of the PEP takes $\mathcal{O}(m^2)$ operations, and the complexity does not depend on the dimension n. The reason is, that to build the PEP $\boldsymbol{F}(\boldsymbol{p}_i) + t\boldsymbol{e}_j) = \boldsymbol{F}(\boldsymbol{p}_i) + t\boldsymbol{A}_j$, the value of $\boldsymbol{F}(\boldsymbol{p}_i)$ can be obtained via $\boldsymbol{F}(\boldsymbol{p}_i) = \boldsymbol{F}(\boldsymbol{p}_{i-1}) + \hat{t}\boldsymbol{A}_k$, assuming in the previous step $\boldsymbol{e}_k$ was chosen as direction. There is not a theoretical mixing time for W-CHnR.

3.2 Billiard walk

W-Billard [32], Alg. 5, is used to sample a convex body K under the uniform distribution; no theoretical results for its mixing time exist. At i-th step, being on position $\boldsymbol{p}_i$, it chooses uniformly a direction vector $\boldsymbol{v}$ and a number L, where $L = -\tau \ln \eta$, $\eta \sim U(0,1)$. Then, it moves at the direction of $\boldsymbol{v}$ for distance L. If during the movement, it hits the boundary without having covered the required distance L, then it continues on a reflected trajectory.

Table 1: The per-step complexity of the random walks in Sec. 3.

	per-step Complexity
W-HnR	$\mathcal{O}(m^{2.697} + m \lg L + nm^2)$
W-CHnR	$\mathcal{O}(m^{2.697} + m \lg L + m^2)$
W-Billard	$\widetilde{\mathcal{O}}(\rho(m^{2.697} + m \lg L + nm^2))$
W-HMC-r	$\widetilde{\mathcal{O}}(\rho((dm)^{2.697} + md \lg L + dnm^2))$

If the number of reflections exceeds a bound ρ, it stays at $\boldsymbol{p}_i$. In [32] they experimentally conclude that W-Billard mixes faster when $\tau \approx diam(K)$.

Algorithm 5: Billiard_Walk (W-Billard)

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S, a point $\boldsymbol{p_i}$, the diameter τ of S and a bound ρ on the number of reflections.

Require: $\boldsymbol{p_i} \in S$

Output : The point $\boldsymbol{p_{i+1}}$ of the $(i+1)$-th step of the walk.

$L \leftarrow -\tau \ln \eta$; $\eta \leftarrow_R \mathcal{U}((0,1))$; *// choose length*

$\boldsymbol{v} \leftarrow_R \mathcal{U}(\partial\mathcal{B}_n)$; *// choose direction*

$\boldsymbol{p} \leftarrow \boldsymbol{p}_i$;

do

  $\Phi(t) := \boldsymbol{p} + t\boldsymbol{v}$; *// define trajectory*

  $t_+, \boldsymbol{s}_+ \leftarrow$ reflection $(\boldsymbol{F}, \Phi(t))$;

  $\hat{t} \leftarrow \min\{t_+, L\}$; $\boldsymbol{p} \leftarrow \Phi(\hat{t})$; $\boldsymbol{v} \leftarrow \boldsymbol{s}_+$; $L \leftarrow L - \hat{t}$;

while $L > 0$;

if $\#\{$*reflections*$\} > \rho$ **then return** $\boldsymbol{p_{i+1}} = \boldsymbol{p_i}$;

else return $\boldsymbol{p_{i+1}} = \boldsymbol{p}$;

The per-step complexity of W-Billard is dominated by the reflection, which requires $\widetilde{\mathcal{O}}(m^{2.697} + m \lg L + nm^2)$ arithmetic operations, when we want to approximate the intersection point up to a factor of 2^{-L}. Since we allow at most ρ reflections per step, the total complexity become $\widetilde{\mathcal{O}}(\rho(m^{2.697} + m \lg L + nm^2))$.

3.3 Hamiltonian Monte Carlo with Reflections

Hamiltonian Monte Carlo (HMC) samples from any probability distribution π. Our focus lies again on the log-concave distributions ($\pi(\boldsymbol{x}) \propto e^{-\alpha f(\boldsymbol{x})}$). In this case, if we assume that f is a strongly convex function, then the mixing time of HMC is $O(k^{1.5} \log(n/\epsilon))$, where κ is the condition number of $\nabla^2 f$ [23]. If we truncate π in a convex body, then we can use boundary reflections (W-HMC-r) to ensure that the random walk converges to the target distribution [10]; however, in this case the mixing time is unclear.

The Hamiltonian dynamics behind HMC operate on a n-dimensional position vector $\boldsymbol{p}$ and a n-dimensional momenta $\boldsymbol{v}$, so the full state space has $2n$ dimensions. The system is described by a function of $\boldsymbol{p}$ and $\boldsymbol{v}$ known as the Hamiltonian, $H(\boldsymbol{p}, \boldsymbol{v}) = U(\boldsymbol{p}) + K(\boldsymbol{v}) = f(\boldsymbol{p}) + \frac{1}{2}|\boldsymbol{v}|^2$. To sample from π one has to solve the following system of Ordinary Differential Equations (ODE):

$$\begin{array}{l} \frac{d\boldsymbol{p}}{dt} = \frac{\partial H(\boldsymbol{p},\boldsymbol{v})}{\partial \boldsymbol{v}} \\ \frac{d\boldsymbol{v}}{dt} = -\frac{\partial H(\boldsymbol{p},\boldsymbol{v})}{\partial \boldsymbol{p}} \end{array} \quad \left\{ \begin{array}{l} \frac{d\boldsymbol{v}(t)}{dt} = -\alpha \nabla f(\boldsymbol{p}) \\ \frac{d\boldsymbol{p}(t)}{dt} = \boldsymbol{v}(t) \end{array} \right. \tag{3.1}$$

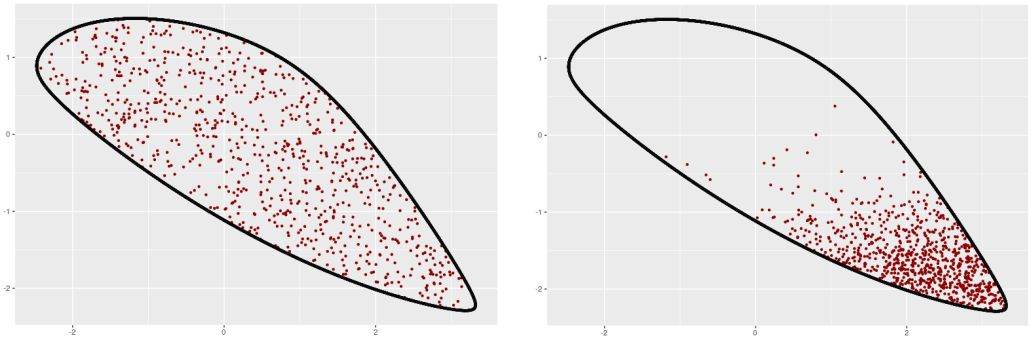

Figure 5: Samples from the uniform distribution with W-Billard (left) and from the Boltzmann distribution $\pi(x) \propto e^{-cx/T}$, where $T = 1,\ c = [-0.09, 1]^T$, with W-HMC-r (right). The volume of this spectrahedron is 10.23.

If $\pi(\boldsymbol{x})$ is a log-concave density, then we can approximate the solution of the ODE with a low degree polynomial trajectory [23], using the collocation method. A degree $d = O(1/\log(\epsilon))$ suffices to obtain a polynomial trajectory with error $O(\epsilon)$, for a fixed time interval, while we perform just $\widetilde{O}(1)$ evaluations of $\nabla f(\boldsymbol{x})$.

HMC at the i-th step uniformly samples a step ℓ from a proper interval to move on the trajectory implied by ODE (3.1), choses $\boldsymbol{v}$ randomly from $\mathcal{N}(0, I)$, and updates $\boldsymbol{p}$ using the ODE in (3.1), for $t \in [0, \ell]$. When π is truncated in a convex body, then W-HMC-r fixes an upper bound ρ on the number of reflections and reflects a polynomial trajectory as we describe in Sec. 2.3.

Algorithm 6: HMC_w_reflection (W-HMC-r)

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ representing a spectrahedron S, a point $\boldsymbol{p_i}$, the diameter τ of S and a bound ρ to the number of reflections.
Require: $\boldsymbol{p_i} \in S$
Output : The point $\boldsymbol{p_{i+1}}$ of the $(i+1)$-th step of the walk.

$\ell \leftarrow \tau\eta;\ \eta \leftarrow_R \mathcal{U}((0,1))$; *// choose length*
$\boldsymbol{v} \leftarrow_R \mathcal{N}(0, I_d)$; *// choose direction*
do
  Compute trajectory $\Phi(t)$ from ODE (3.1);
  $t_+, \boldsymbol{s}_+ \leftarrow$ reflection $(\boldsymbol{F}, \Phi(t))$;
  $\hat{t} \leftarrow \min\{t_+, \ell\}$; $\boldsymbol{p} \leftarrow \Phi(\hat{t})$; $\boldsymbol{v} \leftarrow \boldsymbol{s}$; $\ell \leftarrow \ell - \hat{t}$;
while $L > 0$;
if $\#\ \{$*reflections*$\} > \rho$ **then return** $\boldsymbol{p_{i+1}} = \boldsymbol{p_i}$;
return $\boldsymbol{p_{i+1}} = \boldsymbol{p}$;

Each step of W-HMC-r, when $\pi(x)$ is a log-concave density truncated by S, costs $\widetilde{\mathcal{O}}(\rho((dm)^{2.697} + md \lg L + dnm^2))$, if we approximate the intersection points up to a factor 2^{-L}, where d is the degree of the polynomial that approximates the solution of the ODE (3.1).

4. SAMPLING AND OPTIMIZATION

The use of geometric random walks opens new doors in convex optimization. In this chapter we present a randomized convex optimization algorithm [20], which makes use of random walks. This algorithm is agnostic to the type of the geometric bodies, the only restriction being that they are convex. We will use the tools we developed in chapter 2 to implement this algorithm for spectrahedra, that is to solve semidefinite programs.

4.1 Simulated Annealing Algorithm

The simulated annealing algorithm [20] employs a random walk to sample a convex body under the Boltzmann distribution, to solve convex optimization problems. We will use it to solve semidefinite programs, that is, we will restrict it for convex bodies which are spectrahedra. In particular, we will solve problems of the form

$$\min \langle \boldsymbol{c}, \boldsymbol{x} \rangle, \text{ subject to } \boldsymbol{x} \in S. \tag{4.1}$$

The strategy to approximate the optimal solution $\boldsymbol{x}^*$ of Eq. (4.1), is based on sampling from the Boltzmann distribution, i.e., $\pi(\boldsymbol{x}) \propto e^{-\boldsymbol{c}\boldsymbol{x}/T}$, truncated to S. The scalar T, is called *temperature*. As the temperature T diminishes, the mass of π tends to concentrate around its mode, which is $\boldsymbol{x}^*$. Thus, one could obtain a uniform point using the algorithm in [25], and then use it as a starting point to sample from $\pi_0 \propto e^{-cx/T_0}$, where $T_0 = R$ and $S \subseteq R\mathcal{B}_n$. Then, the cooling schedule $T_{i+1} = T_i(1 - 1/\sqrt{n})$ guarantees that a sample from π_i yields a good starting point for π_{i+1}. After $\widetilde{\mathcal{O}}(\sqrt{n})$ steps the temperature will be low enough, to sample a point within distance ϵ from $\boldsymbol{x}^*$ with high probability. Theorem 4.1.1 states exactly that.

Algorithm 7: Simulated Annealing

Input : a convex body S, number of dimensions n, an initial point $\boldsymbol{p}_0$, direction of minimization $\boldsymbol{c}$ with $|\boldsymbol{c}| = 1$, diameter of body R, number of phases I

Require: $\boldsymbol{p}_0 \in S^\circ$ uniformly sampled

Output : A point $\boldsymbol{p}_I$ s.t. $\mathbb{E}[\boldsymbol{c} \cdot \boldsymbol{p}_I] \leq nT_I + \min_S \boldsymbol{c} \cdot \boldsymbol{x}$

$T_0 \leftarrow R$;
for $i = 1, 2, \cdots, I$ **do**
  $T_i \leftarrow T_{i-1} \cdot \left(1 - \frac{1}{\sqrt{n}}\right)$;
  $\boldsymbol{p}_i \leftarrow_R S$, under distribution $e^{-\boldsymbol{c}\cdot\boldsymbol{x}/T_i}$
return $\boldsymbol{p}_I$;

Theorem 4.1.1 (Theorem 2.1 from [20]) *With probability* $1 - \delta$ *and* $I = (\sqrt{n}\log(Rn/\epsilon\delta))$*, Algorithm 7 outputs* $\boldsymbol{p}_I$ *s.t.*

$$\boldsymbol{c} \cdot \boldsymbol{p}_I \leq \min_{\boldsymbol{x} \in S} \boldsymbol{c} \cdot \boldsymbol{x} + \epsilon \tag{4.2}$$

Table 2: The average #iteration / runtime / failures over 10 generated S-n-m, to achieve error $\epsilon \leq 0.05$. The walk length is one for W-HMC-r and $W_1 = 4\sqrt{n}$ and $W_2 = 4n$ for W-HnR. With "failures" we count the number of times the method fails to converge. Also m is the dimension of the matrix in LMI and n is the dimension that S-n-m lies.

S-n-m	W-HMC-r	W-HnR W_1	W-HnR W_2
S-30-30	20.1 / 2.9/ 0	184.3 / 3.4 / 1	52.1 / 5.2 / 0
S-40-40	24.6 / 7.9 / 0	223.3 / 9.9 / 2	61.9 / 17.1 / 0
S-50-50	29.2 / 12.7 / 0	251.2 / 22.3 / 3	72.3 / 44.6 / 0
S-60-60	32.8 / 24.32 / 0	272.7 / 41.1 / 3	81.5 / 98.9 / 0

4.2 Heuristics

4.2.1 Choosing a Random Walk

At this point, we note that the algorithm in the original paper [20] is a bit different; the author uses W-HnR combined with a heuristic for "choosing a better direction". This heuristic takes into account the geometry of the body when choosing a random direction and requires at each step of the algorithm the computation of a covariance matrix. In algorithm 7, the random walk is not specified and the covariance matrix computation step is omitted (this step does not make sense for all random walks and, is too expensive).

Indeed, in our implementation we use W-HMC-r instead of W-HnR. To make a case for our choice, we present the following experiment. In Table 2 we follow the cooling schedule described, after setting $T_0 \approx R$ and sampling the first uniform point with W-Billard. We give the optimal solution as input and we stop dropping T when an error $\epsilon \leq 0.05$ is achieved. Even in the case when the walk length is set equal to one, W-HMC-r still converges to to the optimal solution. To the best of our knowledge, this is the first time that a randomized algorithm, which is based on random walks, is functional even when the walk length is set to one. On the other hand, we set the walk length of W-HnR $O(\sqrt{n})$ or $O(n)$ in our experiments. Notice that for the smaller walk length, its runtime decreases, but the method becomes unstable, as it sometimes fails to converge. For both cases its runtime in is worse than W-HMC-r.

4.2.2 Estimating Diameter and Temperature schedule

Algorithm 7, as well as W-HMC-r, require the diameter R of the convex body (in our case spectrahedron) as input; in the simulated annealing to determine the starting temperature and in W-HMC-r to determine the length of the trajectory. However, computing the diameter of a body is NP-Hard, so we can only approximate it.

Of course, we need a fast method, since this is only a preparatory step, and we cannot allow it to make a notable difference in the overall performance. In our implementation, we use W-HnR (with coordinate directions) to sample some points (a function of $\sqrt{n}$, where n is the number of dimensions) under the uniform distribution and, take as R, the maximum distance between a pair of those points. We do not claim that this method offers a good approximation of the actual diameter - indeed, even by sampling more points we get better approximations - but for the purpose intended, it works.

Concerning the actual temperature schedule, in algorithm 7, at each phase the temperature

drops with the following schedule:

$$T_i \leftarrow T_{i-1} \cdot \left(1 - \frac{1}{n^k}\right) \tag{4.3}$$

where $k = \frac{1}{2}$. In our tests, it appears that this is a conservative schedule and we can decrease k, thus decreasing the factor $\left(1 - \frac{1}{n^k}\right)$). More on section 4.4.2.

4.3 Implementation and Benchmarks

In this chapter we describe the implementation of the simulated annealing algorithm (Algorithm 7) as elaborated in Chapter 4, the geometric operations of Chapter 2 and some random walks of Chapter 3. Specifically, we focus on the realization of the linear algebra operations used by the above algorithms.

The implementation is based on and extends the functionalities of the open source, C++ GeomScale library[1]. Much of the programming and research took place amidst the Google Summer of Code 2019 program[2], though since then significant improvements were made.

4.3.1 HMC and Boltzmann Distribution

To sample from Boltzmann distributions with W-HMC-r, at each step, starting from $\boldsymbol{p_i}$ and with momenta $\boldsymbol{v_i}$, the ODE of Eq. (3.1) becomes

$$\frac{d^2}{dt^2}\boldsymbol{p}(t) = -\frac{\boldsymbol{c}}{T},\ \frac{d}{dt}\boldsymbol{p}(0) = \boldsymbol{v_i},\ \boldsymbol{p}(0) = \boldsymbol{p_i}. \tag{4.4}$$

Its solution is the polynomial $\boldsymbol{p}(t) = -\frac{\boldsymbol{c}}{2T}t^2 + \boldsymbol{v_i}t + \boldsymbol{p_i}$, which is a parametric representation of a polynomial curve, see Eq. (2.1).

Hence, the intersection for the boundary oracle requires solving the following QEP:

$$(\boldsymbol{A}_2 t^2 + \boldsymbol{A}_1 t + \boldsymbol{A}_0)\boldsymbol{x} = 0 \tag{4.5}$$

where $\boldsymbol{A}_0 = \boldsymbol{F}(\boldsymbol{p})$, $\boldsymbol{A}_1 = \sum_{i=1}^{n} v_i \boldsymbol{A}_i$ and $\boldsymbol{A}_2 = -\sum_{i=1}^{n} \frac{c_i}{2T}\boldsymbol{A}_i$.

4.3.2 Linear Algebra

4.3.2.1 Basic Linear Algebra Operations

The difficulty in implementing the random walks of Chapter 3, laid on implementing the geometric operations of Chapter 2.

[1] `https://github.com/GeomScale/volume_approximation`

[2] `https://summerofcode.withgoogle.com/archive/2019/projects/5081309804756992/`

For basic linear algebra operations, such as arithmetic operations with matrices, we use the Eigen[3] library. It offers its own classes for matrices and vectors, various basic algebra operations like matrix addition and multiplication, as well as more complex ones, such as finding Eigenvalues.

One of its most attractive features is its overloaded operators and the way it evaluates expressions. For example, the standard $\star$ operator, is overloaded for matrix by scalar, matrix by matrix and matrix by vector multiplication.

More importantly, the operators such as $\star,\ +$, are responsible only for creating operation expressions, and not for any actual computations. The actual computations take place in the assignment operator $=$, so the library can optimize complex expressions, usually avoiding many loops which would take place if computations were made separately by each operator. Think of the expression $\boldsymbol{A} = 5 * \boldsymbol{B} + 6 * \boldsymbol{C}$, where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices. If the computations took place at operators $+, *$ we would need three loops, while by computing them at operator $=$, we need only one loop. In addition, by careful use of the library, we can avoid creating temporary variables and unnecessary memory allocations.

To see the difference in efficiency between using the Eigen library against the C++ STL, you can review this pull request[4], in which we modify the random walks to use Eigen (implemented for polytopes).

4.3.2.2 Polynomial Eigenvalue Problems

For the intersection operation (Algorithm 2) we need to solve a PEP of first degree for W-HnR, that is a generalized eigenvalue problem (GEP) and, a PEP of second degree for W-HMC-r, that is a quadratic eigenvalue problem (QEP).

Though Eigen provides functionality for solving generalized eigenvalue problems, it is too expensive. It deals only with complex numbers while we have real and, it finds all the eigenvalues, while we need only two (see section 2.2). To solve a GEP, we employ ARPACK++[5], which deals with the two issues above. ARPACK++ is compatible with Eigen, in the sense that we can pass the Eigen matrices to ARPACK by pointer, without copying them.

Finally, to solve the QEP 4.5, where $\boldsymbol{A}_0$ is Hermitian and positive definite, $\boldsymbol{A}_1, \boldsymbol{A}_2$ Hermitian, we transform it to a GEP, using the following linearization[18]:

$$t \left[\begin{array}{c|c} \boldsymbol{A}_2 & \boldsymbol{0} \\ \hline \boldsymbol{0} & -\boldsymbol{A}_0 \end{array}\right] + \left[\begin{array}{c|c} \boldsymbol{A}_1 & \boldsymbol{A}_0 \\ \hline \boldsymbol{A}_0 & \boldsymbol{0} \end{array}\right] \tag{4.6}$$

Note that the resulting matrices are symmetric, but not positive definite.

Numerical Stability

During experiments we faced numerical stability issues when computing the polynomial eigenvalues. More specific, sometimes we got wrong eigenvalue approximations and got outside of the spectrahedron.

[3] http://eigen.tuxfamily.org/index.php?title=Main_Page
[4] https://github.com/GeomScale/volume_approximation/pull/29
[5] https://www.caam.rice.edu/software/ARPACK/

The fix was, to make the following modifications:

- in the random walk, if the boundary oracle stated we could travel d units of distance till the boundary, we traveled $0.995 * d$, and
- in Algorithm 7, after each iteration, we check if the point returned by the random walk is inside the spectrahedron. If not, we repeat the same repetition. The overhead from the check is not noticeable in the overall execution time.

4.4 Benchmarks

4.4.1 Generating Random Dense Problems

For the semidefinite programs, we generated random spectrahedra with dense matrices, as described in [13]. In specific, the matrices $\boldsymbol{A_i}$ of $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1\boldsymbol{A}_1 + \cdots + x_n\boldsymbol{A}_n$ were taken as such:

- $\boldsymbol{A_0} = \boldsymbol{R_m} \cdot \boldsymbol{R_m^\top} + \boldsymbol{I}$
- for $i = 1, \cdots, n$
  - $\boldsymbol{M} = \boldsymbol{R}_{\frac{m}{2}} + \boldsymbol{R}_{\frac{m}{2}}^\top$
  - $\boldsymbol{A_i} = \begin{pmatrix} -\boldsymbol{M} & 0 \\ 0 & +\boldsymbol{M} \end{pmatrix}$

where $\boldsymbol{R_m}$ is a randomly and uniformly generated matrix of odd numbers, of dimension m. Note that $\boldsymbol{A_0}$ is positive definite, so $\boldsymbol{F}(\boldsymbol{0}) \succ 0$, that is $\boldsymbol{0} \in S^\circ$. Indeed, this is the initial interior point we use.

We should note, that the spectrahedra generated in this way may follow some pattern, for example be well rounded, so additional tests are required. However, randomly generated full dimensional and bounded spectrahedra is by no means trivial.

4.4.2 Against SDPA

In this section we present some benchmarks against the SDPA library [42]. The tests were generated as described in chapter 4.4.1 and consist of dense matrices.

4.4.2.1 Choosing a Temperature Schedule

Remember from section 4.2.2, we have parameterized the temperature schedule of the algorithm. In figure 6 there are time measurements compared to SDPA for problems with $n = m = 10, 20, \cdots, 300$, where n, m are as in the dual semidefinite program in Definition 4. For the randomized approach, the average was taken over three repeats of every experiment. As for the stopping criterion, in each experiment we stopped when the relative error dropped below 1%. We can see that the algorithm behaved the best for $k = 0.25$.

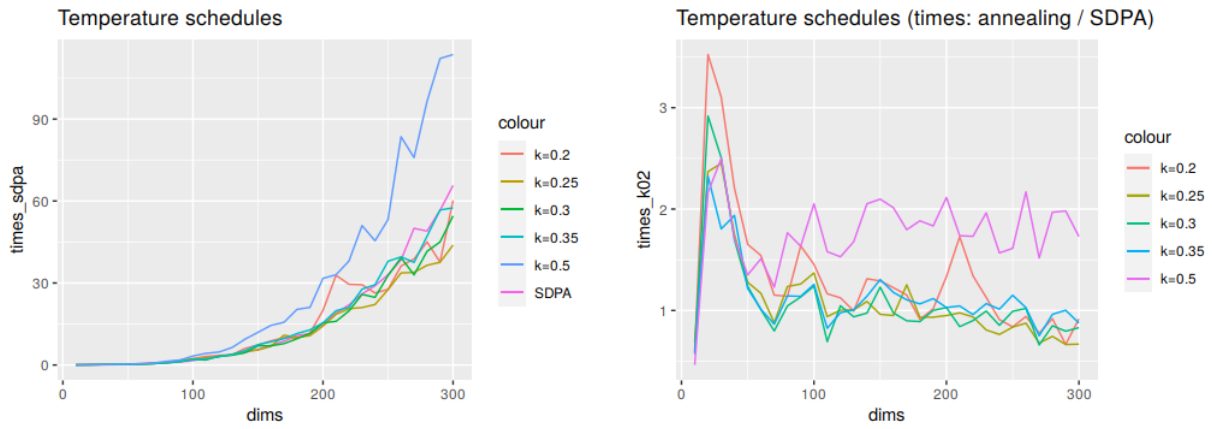

Figure 6: Different temperature schedules (left) for different values of k in Equation 4.3 and the corresponding time ratios SA / SDPA (right).

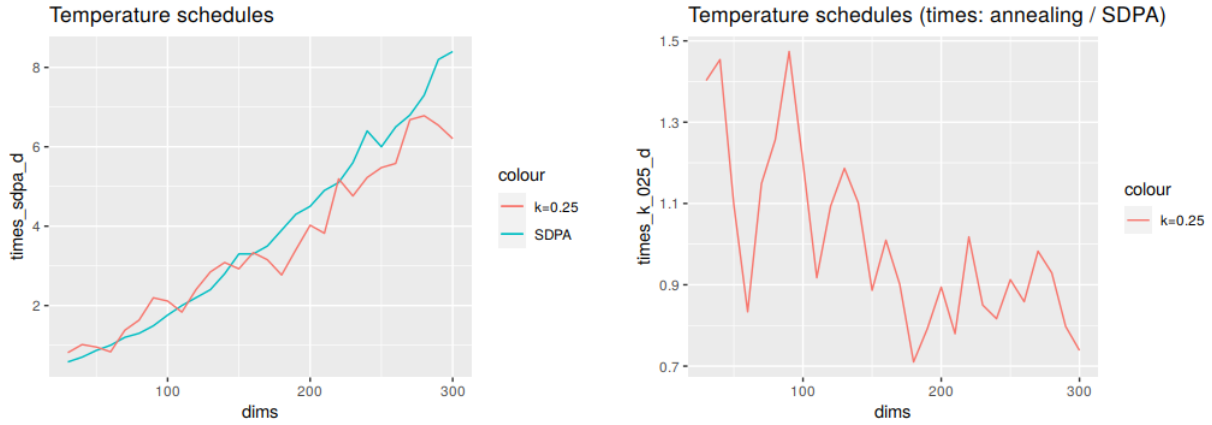

Figure 7: Simulated annealing with $k = 0.25$ in Equation 4.3 (left) and the time ratio SA / SDPA (right), with $m = 100$ and varying d.

4.4.2.2 Scaling of the Algorithm

We have generated tests with fixed $m = 100$ and $n = 10, 20, \cdots, 300$ (Figure 7) and, tests with fixed $n = 100$ and $m = 10, 20, \cdots, 300$ (Figure 8).We run the SA algorithm till the value of the objective function had a less than 10^{-2} relative error compared to the optimal solution. The measurements are the average of three executions. We should mention that SDPA failed to solve some of our tests, while our solver worked; these tests were omitted, since we need to compare their execution times.

The following results are for $k = 0.25$ (see section 4.4.2.1). From Figures 8, 7, we can see that SA scales better with respect to n than SDPA, while it is not far with respect to m. We can conclude that our solver is comparable with SDPA, which is very impressive. SDPA is a well developed package, with highly optimized linear algebra operations and many heuristics, while our solver uses tools not tailored to our needs, especially when solving a QEP, the core operation and bottleneck of the algorithm.

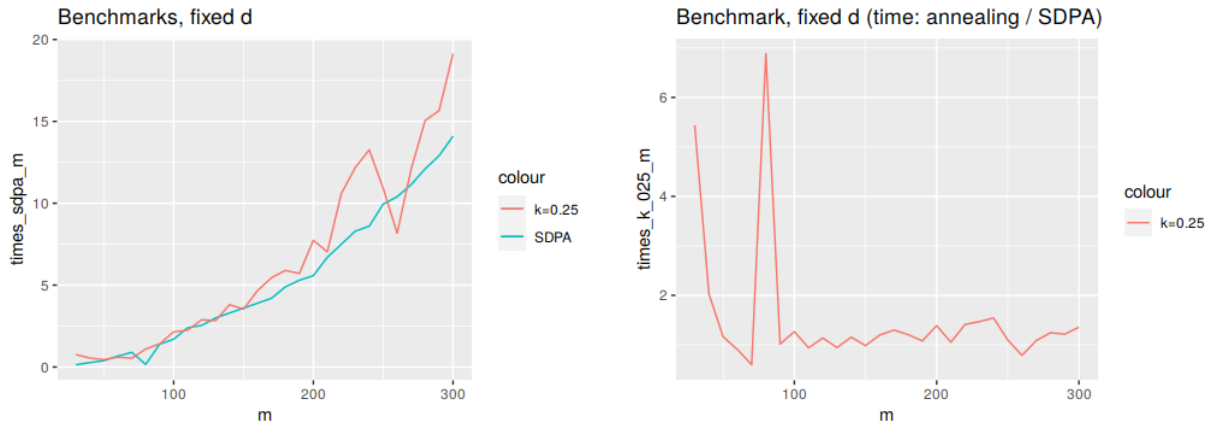

Figure 8: Simulated annealing with $k = 0.25$ in Equation 4.3 (left) and the time ratio SA / SDPA (right), with $d = 100$ and varying m.

ACRONYMS

PEP	Polynomial Eigenvalue Problem
GEP	Generalized Eigenvalue Problem
QEP	Quadratic Eigenvalue Problem
W-Billard	Billiard Random Walk
W-HnR	Random Directions Hit and Run Random Walk
W-CHnR	Coordinate Directions Hit and Run Random Walk
W-HMC-r	Hamiltonian Monte Carlo Random Walk with Reflections
LMI	Linear Matrix Inequality
SDP	Semidefinite Programming
SA	Simulated Annealing

REFERENCES

[1] Hadi Mohasel Afshar and Justin Domke. Reflection, Refraction, and Hamiltonian Monte Carlo. In *Proc. 28th NIPS*, pages 3007–3015, Cambridge, MA, USA, 2015. MIT Press.

[2] Diego Armentano and Carlos Beltrán. The polynomial eigenvalue problem is well conditioned for random inputs. *SIMAX*, 40(1):175–193, 2019.

[3] Carlos Beltrán and Khazhgali Kozhasov. The real polynomial eigenvalue problem is well conditioned on the average. *FoCM*, pages 1–19, 2019.

[4] Michael Berhanu. *The polynomial eigenvalue problem*. PhD thesis, University of Manchester, 2005.

[5] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo, 2017.

[6] Jean-Daniel Boissonnat and Monique Teillaud. *Effective computational geometry for curves and surfaces*. Springer, 2006.

[7] Giuseppe Calafiore. Random walks for probabilistic robustness. In *Proc. CDC*, volume 5, pages 5316–5321. IEEE, 2004.

[8] Giuseppe Calafiore and MC Campi. Robust convex programs: Randomized solutions and applications in control. In *Proc. CDC*, volume 3, pages 2423–2428. IEEE, 2003.

[9] Y. Chen, R. Dwivedi, M. J. Wainwright, and B. Yu. Vaidya walk: A sampling algorithm based on the volumetric barrier. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1220–1227, Oct 2017.

[10] Augustin Chevallier, Sylvain Pion, and Frédéric Cazals. Hamiltonian Monte Carlo with boundary reflections, and application to polytope volume calculations. Research Report RR-9222, INRIA Sophia Antipolis, France, November 2018.

[11] B. Cousins and S. Vempala. Bypassing KLS: Gaussian cooling and an $O^*(n^3)$ volume algorithm. In *Proc. ACM STOC*, pages 539–548, 2015.

[12] B. Cousins and S. Vempala. A practical volume algorithm. *Mathematical Programming Computation*, 8, June 2016.

[13] F. Dabbene, P. S. Shcherbakov, and B. T. Polyak. A randomized cutting plane method with probabilistic geometric convergence. *SIAM J. on Optimization*, 20(6):3185–3207, October 2010.

[14] Fabrizio Dabbene, Pavel Shcherbakov, and Boris T. Polyak. A randomized cutting plane method with probabilistic geometric convergence. *SIOPT*, 20:3185–3207, 2010.

[15] Jean-Pierre Dedieu and Françoise Tisseur. Perturbation theory for homogeneous polynomial eigenvalue problems. *Linear algebra and its applications*, 358(1-3):71–94, 2003.

[16] Ioannis Emiris, Bernard Mourrain, and Elias Tsigaridas. Separation bounds for polynomial systems. *Journal of Symbolic Computation*, 2019.

[17] Elena Gryazina and Boris Polyak. Random sampling: Billiard walk algorithm. *European Journal of Operational Research*, 238, 11 2012.

[18] Nicholas J. Higham, D. Steven Mackey, Niloufer Mackey, and Françoise Tisseur. Symmetric Linearizations for Matrix Polynomials. *SIAM Journal on Matrix Analysis and Applications*, 29(1):143–159, January 2007.

[19] Adam Tauman Kalai and Santosh Vempala. Simulated annealing for convex optimization. *Math. Oper. Res.*, 31(2):253–266, February 2006.

[20] Adam Tauman Kalai and Santosh Vempala. Simulated Annealing for Convex Optimization. *Mathematics of Operations Research*, 31(2):253–266, May 2006.

[21] Erich Kaltofen and Gilles Villard. On the complexity of computing determinants. *Computational complexity*, 13(3-4):91–130, 2005.

[22] Grégoire Lecerf. On the complexity of the Lickteig–Roy subresultant algorithm. *Journal of Symbolic Computation*, 92:243–268, May 2019.

[23] Yin Tat Lee, Zhao Song, and Santosh S. Vempala. Algorithmic theory of odes and sampling from well-conditioned logconcave densities, 2018.

[24] Thomas Lickteig and Marie-Françoise Roy. Sylvester–habicht sequences and fast cauchy index computation. *Journal of Symbolic Computation*, 31(3):315–341, March 2001.

[25] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. In *In Proc. FOCS*, volume 2003, pages 650–659, 2003.

[26] L. Lovasz and S. Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *Proc. FOCS*, pages 57–68, 2006.

[27] J.R. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics (Revised Edition)*. John Wiley & Sons Ltd, 1999.

[28] O. Mangoubi and N. K. Vishnoi. Faster polytope rounding, sampling, and volume computation via a sub-linear ball walk. In *Proc. FOCS*, pages 1338–1357, 2019.
[29] Victor Y Pan. Univariate polynomials: nearly optimal algorithms for numerical factorization and root-finding. *J. of Symbolic Computation*, 33(5):701–733, 2002.
[30] Victor Y Pan and Elias P Tsigaridas. Nearly optimal refinement of real roots of a univariate polynomial. *Journal of Symbolic Computation*, 74:181–204, 2016.
[31] Boris Polyak and Pavel Shcherbakov. The d-decomposition technique for linear matrix inequalities. *Automation and Remote Control*, 67:1847–1861, 11 2006.
[32] B.T. Polyak and E.N. Gryazina. Billiard walk - a new sampling algorithm for control and optimization. *IFAC Proceedings Volumes*, 47(3):6123 – 6128, 2014.
[33] M. Ramana and A. Goldman. Some geometric results in semidefinite programming. *Journal of Global Optimization*, 7, 02 1999.
[34] Arnold Schönhage. The fundamental theorem of algebra in terms of computational complexity. *Manuscript. Univ. of Tübingen, Germany*, 1982.
[35] Robert L. Smith. Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.
[36] Serge Tabachnikov. *Geometry and billiards*. Student mathematical library. American Mathematical Society, Providence, RI, 2005.
[37] Roberto Tempo, Giuseppe Calafiore, and Fabrizio Dabbene. *Randomized algorithms for analysis and control of uncertain systems: with applications*. Springer Science, 2012.
[38] Françoise Tisseur. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra and its Applications*, 309(1):339–361, 2000.
[39] Françoise Tisseur and Karl Meerbergen. The quadratic eigenvalue problem. *SIAM review*, 43(2):235–286, 2001.
[40] Lloyd N. Trefethen and David Bau. *Numerical linear algebra.* SIAM, 1997.
[41] S. Vempala. Geometric random walks: A survey. *Combinatorial and Computational Geometry MSRI Publications Volume*, 52, 01 2005.
[42] Makoto Yamashita, Katsuki Fujisawa, and Masakazu Kojima. Implementation and evaluation of sdpa 6.0. *Optimization Methods and Software*, 18(4):491–505, 2003.