ON ALGORITHMS FOR THE COLOURFUL LINEAR PROGRAMMING FEASIBILITY PROBLEM

ON ALGORITHMS FOR THE COLOURFUL LINEAR PROGRAMMING FEASIBILITY PROBLEM

By

Guohong Rong

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2012) McMaster University
(Computer Science) Hamilton, Ontario

TITLE: On Algorithms for the Colourful Linear Programming Feasibility Problem

AUTHOR: Guohong Rong

SUPERVISOR: Dr. Antoine Deza

NUMBER OF PAGES: xiv, 55.

Abstract

Given colourful sets $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ of points in $\mathbb{R}^d$ and a point $\mathbf{p}$ in $\mathbb{R}^d$, the colourful linear programming problem is to express $\mathbf{p}$ as a convex combination of points $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}$ with $\mathbf{x}_i \in \mathbf{S}_i$ for each i. This problem was presented by Bárány and Onn in 1997 and it is still not known if a polynomial-time algorithm for the problem exists. The monochrome version of this problem, expressing $\mathbf{p}$ as a convex combination of points in a set $\mathbf{S}$, is a traditional linear programming feasibility problem. The colourful Carathéodory Theorem, due to Bárány in 1982, provides a sufficient condition for the existence of a colourful set of points containing $\mathbf{p}$ in its convex hull. Bárány's result was generalized by Holmsen et al. in 2008 and by Arocha et al. in 2009 before being recently further generalized by Meunier and Deza. We study algorithms for colourful linear programming under the conditions of Bárány and their generalizations. In particular, we implement the Meunier-Deza algorithm and enhance previously used random case generators. Computational benchmarking and a performance analysis including a comparison between the two algorithms of Bárány and Onn and the one of Meunier and Deza, and random picking are presented

Acknowledgments

The thesis was written under the guidance and with the help of my supervisor, Dr. Antoine Deza. His valuable advice and extended knowledge helped me to do my best while working on the thesis. My special thanks go to the members of the examination committee: Dr. Franya Franek (Chair), Dr. Sanzheng Qiao and Dr. Antoine Deza.

I would like to appreciate the great aid and support from all the members of the Advanced Optimization Laboratory.

Finally, I am indebted to thank my family and friends for their patience, understanding and continuous support.

Contents

List of Figures

List of Tables

Chapter 1

Preliminaries

In this chapter, we introduce some fundamental concepts and notations in d-dimensional geometry used throughout this thesis.

1.1 Points and Vectors

In this thesis, a *point* in d-dimensional Euclidean space $\mathbb{R}^d$ is denoted by a symbol with **bold** font and lower case character, and it can be represented by a column matrix with d coordinates. The *origin* is denoted by $\mathbf{0}$.

A *point set* is a set which contains a finite or infinite number of points, and is denoted by a symbol with **bold** font and upper case character, such as $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_k\}$ with k points.

A *vector* which originates from $\mathbf{0}$ towards point $\mathbf{t}$ is identified to the point $\mathbf{t}$.

1.2 Norm, Distance and Projection

Given $\mathbf{t} \in \mathbb{R}^d$ and $k \in \mathbb{N}$, the *k-norm* is $\|\mathbf{t}\|_k = \sqrt[k]{|t_1|^k + \ldots + |t_d|^k}$. We often use the Euclidean 2-norm, and simply call it *norm* and write it as $\|\mathbf{t}\|$. The

2-norm can be expressed as $\|\mathbf{t}\| = \sqrt{t_1^2 + \ldots + t_d^2} = \sqrt{\langle \mathbf{t}, \mathbf{t} \rangle}$. The *normalized vector* or *unit vector* of a non-zero vector $\mathbf{t}$ is represented by $\bar{\mathbf{t}} = \dfrac{\mathbf{t}}{\|\mathbf{t}\|}$.

The *distance* between two points $\mathbf{t_1}$ and $\mathbf{t_2}$ in $\mathbb{R}^d$ is the norm of their difference, namely $\operatorname{dist}(\mathbf{t_1}, \mathbf{t_2}) = \|\mathbf{t_1} - \mathbf{t_2}\|$. The distance between two point sets $\mathbf{S_1}$ and $\mathbf{S_2}$ in $\mathbb{R}^d$ is the minimum value of $\|\mathbf{t_1} - \mathbf{t_2}\|$, where $\mathbf{t_1} \in \mathbf{S_1}$ and $\mathbf{t_2} \in \mathbf{S_2}$.

The *projection* $\operatorname*{proj}_{\mathbf{S}}(\mathbf{t})$ of a point $\mathbf{t}$ on a point set $\mathbf{S}$ is the point $\mathbf{p} \in \mathbf{S}$ which minimizes the distance from $\mathbf{t}$ to $\mathbf{S}$, i.e., $\operatorname*{proj}_{\mathbf{S}}(\mathbf{t}) = \operatorname*{argmin}_{\mathbf{p} \in \mathbf{S}} \|\mathbf{t} - \mathbf{p}\|$. The projection of one set $\mathbf{A}$ onto another set $\mathbf{B}$ is $\{\operatorname*{proj}_{\mathbf{B}}(\mathbf{a}) \colon \mathbf{a} \in \mathbf{A}\}$. If $\mathbf{t} \in \mathbf{S}$, we have $\operatorname*{proj}_{\mathbf{S}}(\mathbf{t}) = \mathbf{t}$.

1.3 Ball and Sphere

The d-dimensional *ball* of radius $r \geq 0$ centered at point $\mathbf{p} \in \mathbb{R}^d$ is the set $\left\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} - \mathbf{p}\| \leq r\right\}$, and denoted by $\mathbb{B}(\mathbf{p}, r)$. The d-dimensional *sphere* of radius $r \geq 0$ centered at point $\mathbf{p} \in \mathbb{R}^d$ is the set $\left\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} - \mathbf{p}\| = r\right\}$, and denoted by $\mathbb{S}(\mathbf{p}, r)$. The *unit ball* $\mathbb{B}^{d-1}$ and *unit sphere* $\mathbb{S}^{d-1}$ correspond to $\mathbb{B}^{d-1} = \mathbb{B}(\mathbf{0}, 1)$ and $\mathbb{S}^{d-1} = \mathbb{S}(\mathbf{0}, 1)$ respectively.

1.4 Convex Hull, Affine Hull and Affine Subspace

The *convex hull* of $\mathbf{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_k\}$ in $\mathbb{R}^d$ is the set of all linear combinations with non-negative coefficients whose sum is one, i.e.

$$\operatorname{conv}(\mathbf{S}) = \left\{ \sum_{i=1}^{k} \lambda_i \mathbf{x}_i \colon \mathbf{x}_i \in \mathbf{S}, \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$

The *affine hull* of $\mathbf{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_k\}$ in $\mathbb{R}^d$ is the set of all linear combinations with the sum of coefficients being one, i.e.

$$\operatorname{aff}(\mathbf{S}) = \left\{ \sum_{i=1}^{k} \lambda_i \mathbf{x}_i \colon \mathbf{x}_i \in \mathbf{S}, \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$

The expression $\sum_{i=1}^{k} \lambda_i \mathbf{x}_i$ with $\sum_{i=1}^{k} \lambda_i = 1$ is called *affine combination* of $\mathbf{x}_1, \cdots, \mathbf{x}_k$. If none of $\mathbf{x}_i \in \mathbf{S}$ can be an affine combination of the others in $\mathbf{S}$, then $\mathbf{x}_1, \cdots, \mathbf{x}_k$ are *affinely independent.*

1.5 Affine Hyperplane, Hyperplane and Half Space

An *affine hyperplane* is a $(d-1)$-dimensional affine subspace of $\mathbb{R}^d$. A *hyperplane* is an affine hyperplane that contains $\mathbf{0}$ and it is a generalization of the plane in $\mathbb{R}^d$. Each hyperplane can be expressed as $H = \{\mathbf{x} \colon \mathbf{n}^{\mathrm{T}}\mathbf{x} = c\}$, where $\mathbf{n} \in \mathbb{R}^d$ is called the *normal vector* of H and $c \in \mathbb{R}$.

A hyperplane $H = \{\mathbf{x} \colon \mathbf{n}^{\mathrm{T}}\mathbf{x} = c\}$ in $\mathbb{R}^d$ divides d-dimensional space into two *open half space* $\{\mathbf{x} \colon \mathbf{n}^{\mathrm{T}}\mathbf{x} > c\}$ and $\{\mathbf{x} \colon \mathbf{n}^{\mathrm{T}}\mathbf{x} < c\}$.

1.6 Dimension of Point Set, General Position and Degeneracy

Let $\mathbf{S}$ be a finite point set in $\mathbb{R}^d$, the *dimension* $\dim(\mathbf{S})$ of $\mathbf{S}$ is k if $\mathbf{S}$ is in a k-dimensional affine subspace; and $\mathbf{S}$ is not in a $(k-1)$-dimensional affine subspace.

A finite point set $\mathbf{S} \subset \mathbb{R}^d$ is in *general position* if for any $k < d$ there is no k-dimensional affine subspace which contains $k+2$ points from $\mathbf{S}$. A finite set $\mathbf{S}$ not in general position is *degenerate.*

1.7 Transversal

Given a collection $\boldsymbol{C}$ of sets, a *transversal* is a set containing exactly one element from each member of the collection. *Partial transversal* is a set containing at most one element from each member of the collection.

1.8 Convex Polytope, Face, Facet and Vertex

A *convex polytope* is the convex hull of a finite point set in $\mathbb{R}^d$. Another definition is that convex polytope is the intersection of finitely many closed half spaces in $\mathbb{R}^d$ and is bounded. The two definitions are equivalent. In this thesis we omit the word *convex* and simply use the word *polytope*. A polytope $\mathbf{P} \subset \mathbb{R}^d$ is *full dimensional* if $\dim(\mathbf{P}) = d$.

Let $\mathbf{P}$ be a d-dimensional polytope in $\mathbb{R}^d$, a closed half space is valid if $\mathbf{P}$ belongs to it. A hyperplane associated with a valid half space is called a *valid hyperplane*. A *face* of $\mathbf{P}$ is the intersection of $\mathbf{P}$ with some valid hyperplanes. The 0-face, 1-face, (d-2)-face and (d-1)-face are called *vertex*, *edge*, *ridge* and *facet* respectively. A vertex of a polytope is a point.

Chapter 2

Colourful Carathéodory's Theorems

This chapter introduces the colourful Carathéodory's theorem and its generalizations; and presents the colourful linear programming problem related to the colourful Carathéodory's theorem and its generalizations.

2.1 Carathéodory's Theorem and its Linear Programming Problem

Theorem 2.1.1 (Carathéodory's Theorem). *Given a set of points* $\mathbf{S} \subset \mathbb{R}^d$*, if the origin* $\mathbf{0} \in \operatorname{conv}(\mathbf{S})$*, then there is a subset* $\mathbf{T} \subseteq \mathbf{S}$ *such that* $\mathbf{0} \in \operatorname{conv}(\mathbf{T})$ *with* $|\mathbf{T}| \leq d+1$*.*

The theorem is named for Constantin Carathéodory because he first proved this theorem in 1911. Based on this theorem, the following algorithmic problem is proposed in [3] to suggest finding a subset $\mathbf{T}$ in $\mathbf{S}$ with $|\mathbf{T}| \leq d+1$.

Problem 2.1.1 (Linear Programming Problem). *Given a finite set of points* $\mathbf{S} \subset \mathbb{R}^d$ *and origin* $\mathbf{0}$*, decide whether there is a subset* $\mathbf{T} \subseteq \mathbf{S}$ *of size at most*

$d+1$ *such that* $\mathbf{0} \in \mathrm{conv}(\mathbf{T})$*, and if there is one, find it.*

The Carathéodory's theorem guarantees that if $\mathbf{0} \in \mathrm{conv}(\mathbf{S})$, the corresponding linear programming problem has at least one feasible solution. Since this problem is a special case of the linear optimization problem, we can use pivoting algorithms or interior point algorithms to solve it.

2.2 Colourful Carathéodory's Theorem and its Linear Programming Problem

In 1982, Imre Bárány firstly proposed and proved a colourful generalization to the Carathéodory's Theorem in [1]. Before we state it, we introduce the following terminology: Given $d+1$ sets, or *colours*, $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ of points in $\mathbb{R}^d$, a *colourful* set is a set $\mathbf{T} \subset \cup_i \mathbf{S}_i$ such that $|\mathbf{T} \cap \mathbf{S}_i| \leq 1$ for $i = 1, \ldots, d+1$. A *colourful simplex* is the convex hull of a colourful set $\mathbf{T}$, and a colourful set of d points which misses colour $\mathbf{S}_i$ is called an $\widehat{\imath}$*-transversal.*

Theorem 2.2.1 (Colourful Carathéodory's Theorem)**.** *Let* $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ *be finite sets of points in* $\mathbb{R}^d$ *such that the origin* $\mathbf{0} \in \mathrm{conv}(\mathbf{S}_i)$ *for* $i = 1, \ldots, d+1$*. Then there exists a set* $\mathbf{T} \subset \bigcup_i \mathbf{S}_i$ *such that* $|\mathbf{T} \cap \mathbf{S}_i| = 1$ *for* $i = 1, \ldots, d+1$ *and* $\mathbf{0} \in \mathrm{conv}(\mathbf{T})$*.*

The monochromatic Carathéodory's Theorem can be obtained from the colourful Carathéodory's Theorem by setting $\mathbf{S} = \mathbf{S}_1 = \cdots = \mathbf{S}_{d+1}$.

In 1997, Imre Bárány and Shmuel Onn in [3] proposed the following algorithmic problem suggested by Theorem 2.2.1, which is a generalization of Problem 2.1.1.

Problem 2.2.1 (Colourful Linear Programming Problem)**.** *Given colours* $\mathbf{S}_1, \ldots, \mathbf{S}_k \subset \mathbb{R}^d$ *and origin* $\mathbf{0}$*, decide whether there is a colourful set* $\mathbf{T} \subset \bigcup_i \mathbf{S}_i$ *such that* $|\mathbf{T} \cap \mathbf{S}_i| = 1$ *for* $i = 1, \ldots, k$ *and* $\mathbf{0} \in \operatorname{conv}(\mathbf{T})$*, and if there is one, find it.*

The specialization of this problem to linear programming is obtained by taking $\mathbf{S} = \mathbf{S}_1 = \cdots = \mathbf{S}_{d+1}$.

The colourful linear programming problem is denoted by *CLPP*. Paper [3] shows that the *CLPP* is *NP*-complete. The colourful Carathéodory's Theorem 2.2.1 provides a sufficient condition for the existence of a colourful simplex containing the origin $\mathbf{0}$. The sufficient condition is: $k = d + 1$ and $\mathbf{0} \in \bigcap_{i=1}^{d+1} \operatorname{conv}(\mathbf{S}_i)$. In this thesis, we use *CLPP-TH1* to indicate the *CLPP* which satisfies the sufficient condition of Theorem 2.2.1.

In [3], two algorithms are proposed to find a colourful simplex containing $\mathbf{0}$ for the sets satisfying the condition of Theorem 2.2.1 and we will introduce them in Chapter 3. It is proved that the number of real arithmetic operations taken by these two algorithms is $O(\frac{1}{\rho^2}\log\frac{1}{\epsilon})$. Therefore the *CLPP-TH1* is suggested to be on the border line between tractable and intractable computational problems.

Subsequently, Antoine Deza, Sui Huang, et al. in [4] and [5] provide multi-update modifications to the two Bárány-Onn's algorithms in order to achieve big improvement in practical performance. In addition, they also present three algorithms trying to find solutions for general *CLPP*: random picking, enumeration with geometric heuristic and non-definite quadratic optimization approach. We will introduce random picking algorithm in Chapter 3.

Paper [3] proves that counting the colourful simplices containing $\mathbf{0}$ for the

CLPP is $\#P$-complete. Let $\mu(d)$ denote minimum number of colourful simplices containing $\mathbf{0}$ for sets satisfying the condition of Theorem 2.2.1. Paper [9] shows that $2d \le \mu(d) \le d^2+1$, that $\mu(d)$ is even for odd d, and that $\mu(2) = 5$. [10] provides a lower bound of $\mu(d) \ge \max(3d, \lceil \frac{d(d+1)}{5} \rceil)$ for $d \ge 3$, while [11] independently provides a lower bound of $\mu(d) \ge \lfloor \frac{(d+2)^2}{4} \rfloor$, before [12] shows that $\mu(d) \ge \lceil \frac{(d+2)^2}{2} \rceil$.

2.3 Generalization of Colourful Carathéodory's Theorem

The colourful Carathéodory's Theorem 2.2.1 was generalized by A. Holmsen, J. Pach, et al. in 2008 in [6] and by J. Arocha, I. Bárány, et al. in 2009 in [7] independently.

Theorem 2.3.1 (General Colourful Carathéodory's Theorem)**.** *Let* $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ *be finite sets of points in* $\mathbb{R}^d$ *such that the origin* $\mathbf{0} \in \operatorname{conv}(\mathbf{S}_i \cup \mathbf{S}_j)$ *for* $1 \le i < j \le d+1$*. Then there exists a set* $\mathbf{T} \subset \bigcup_i \mathbf{S}_i$ *such that* $|\mathbf{T} \cap \mathbf{S}_i| = 1$ *for* $i = 1, \ldots, d+1$ *and* $\mathbf{0} \in \operatorname{conv}(\mathbf{T})$*.*

The general colourful Carathéodory's Theorem 2.3.1 provides a more general sufficient condition for the existence of a colourful simplex containing the origin $\mathbf{0}$. The sufficient condition is: $k = d+1$ and $\mathbf{0} \in \bigcap_{1 \le i < j \le d+1} \operatorname{conv}(\mathbf{S}_i \cup \mathbf{S}_j)$. In this thesis, *CLPP-TH2* denotes the *CLPP* which satisfies the sufficient condition of Theorem 2.3.1.

Until now, there is no published paper which proposes an efficient algorithm for the *CLPP-TH2*. Paper [13] shows that the minimum number of solutions for the *CLPP-TH2* is $d+1$.

2.4 Further Generalization of Colourful Carathéodory's Theorem

In 2011, Frédéric Meunier and Antoine Deza in [8] strengthened the colourful Carathéodory's Theorem 2.3.1 by further generalizing the sufficient condition for the existence of a colourful simplex containing the origin $\mathbf{0}$. There are two strengthened colourful Carathéodory's theorems they proposed.

Theorem 2.4.1. *Let $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ be finite sets of points in $\mathbb{R}^d$. Assume that, for each $1 \leq i < j \leq d+1$, there exists $k \notin \{i, j\}$ such that, for all $\mathbf{x}_k \in \mathbf{S}_k$, the convex hull of $\mathbf{S}_i \cup \mathbf{S}_j$ intersects the ray $\overrightarrow{\mathbf{x}_k\mathbf{0}}$ in a point distinct from $\mathbf{x}_k$. Then there exists a set $\mathbf{T} \subset \bigcup_i \mathbf{S}_i$ such that $|\mathbf{T} \cap \mathbf{S}_i| = 1$ for $i = 1, \ldots, d+1$ and $\mathbf{0} \in \mathrm{conv}(\mathbf{T})$.*

Before we state the second theorem, we introduce the following notations: $H^+(\mathbf{T}_i)$ denotes , for any $\widehat{i}$-transversal $\mathbf{T}_i$, the open half space defined by $\mathrm{aff}(\mathbf{T}_i)$ and containing the origin $\mathbf{0}$.

Theorem 2.4.2 (Further General Colourful Carathéodory's Theorem)**.** *Let $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ be finite sets of points in $\mathbb{R}^d$ such that the points in $\cup_i \mathbf{S}_i \cup \{\mathbf{0}\}$ are distinct and in general position. Assume that, for any $i \neq j$, $(\mathbf{S}_i \cup \mathbf{S}_j) \cap H^+(\mathbf{T}_i) \neq \varnothing$ for any $\widehat{i}$-transversal $\mathbf{T}_i$. Then there exists a set $\mathbf{T} \subset \bigcup_i \mathbf{S}_i$ such that $|\mathbf{T} \cap \mathbf{S}_i| = 1$ for $i = 1, \ldots, d+1$ and $\mathbf{0} \in \mathrm{conv}(\mathbf{T})$.*

Theorem 2.4.1 can be derived from the slightly stronger Theorem 2.4.2. In this thesis, we will mainly focus on the discussion of Theorem 2.4.2. The further general colourful Carathéodory's Theorem 2.4.2 provides a even more general sufficient condition for the existence of a colourful simplex containing origin $\mathbf{0}$.

The sufficient condition is: $k = d+1$ and for any $i \neq j$, $(\mathbf{S}_i \cup \mathbf{S}_j) \cap H^+(\mathbf{T}_i) \neq \varnothing$ for any $\widehat{i}$-transversal $\mathbf{T}_i$. *CLPP-TH3* denotes the *CLPP* satisfying the sufficient condition of Theorem 2.4.2.

An algorithm finding a solution for *CLPP-TH3* is proposed in [8] based on following proposition. We will detail this algorithm in the following chapters of this thesis.

Proposition 2.4.3. *Given d+1 sets, or colours,* $\mathbf{S}_1^*, \ldots, \mathbf{S}_{d+1}^*$ *of points in* $\mathbb{R}^d$ *with* $|\mathbf{S}_i^*| = 2$ *for* $i = 1, \ldots, d+1$, *if there is a colourful simplex containing* $\mathbf{0}$, *then there is another colourful simplex containing* $\mathbf{0}$.

Paper [8] shows that the minimum number of colourful simplices containing $\mathbf{0}$ for the *CLPP-TH3* is d+1.

From the generalizations of colourful Carathéodory's theorem, we know that the algorithms designed for solving *CLPP-TH3* are also suitable for finding solutions for *CLPP-TH2* or *CLPP-TH1*.

2.5 Thesis Outline

This thesis mainly focuses on the design and implementation of algorithms for the *CLPP* which satisfies different sufficient conditions. Three kinds of random case generators are constructed to meet the sufficient conditions of *CLPP-TH1*, *CLPP-TH2* and *CLPP-TH3* respectively. Finally the proposed algorithms are tested and benchmarked against different random case generators and some conclusions are given.

Chapter 1 presents some fundamental geometric concepts in $\mathbb{R}^d$ which will be used throughout this thesis.

Chapter 2 introduces the colourful Carathéodory's theorem and its generalizations, and their colourful linear programming problems.

Chapter 3 discusses the algorithms designed for solving the problems of *CLPP-TH1*, *CLPP-TH2* and *CLPP-TH3*.

Chapter 4 provides the details of software implementation for the algorithms proposed in Chapter 3.

Chapter 5 describes the methods of the three kinds of random case generators which satisfy the three sufficient conditions of *CLPP* respectively.

Chapter 6 deals with the benchmark testing for the algorithms proposed in Chapter 3 against the random cases generated by the generators presented in Chapter 5, and some analyses are given.

Chapter 7 contains conclusion remarks and suggestions for future work.

Chapter 3

Algorithms for Colourful Linear Programming Problem

In this chapter, Bárány-Onn algorithms in [3] are presented, Meunier-Deza algorithm in [8] is detailed, and Random-Picking algorithm is introduced.

For simplicity of algorithm design and without loss of generality, we introduce the *colourful configuration* of the *CLPP*:

- All points in $\cup_i \mathbf{S}_i \cup \{\mathbf{0}\}$ are distinct and in general position;
- For every point $\mathbf{s} \in \cup_i \mathbf{S}_i$, $\|\mathbf{s}\| = 1$, i.e. the point $\mathbf{s}$ is *normalized*; if not, scale all the points onto the unit sphere $\mathbb{S}^{d-1}$.

We assume that all the *CLPP*s which satisfy one of the three sufficient conditions described in Chapter 2 will satisfy the colourful configuration.

3.1 Bárány-Onn Algorithms

Paper [3] provides two algorithms for the *CLPP-TH1*, both of which are based on geometric theories and involve certain iterative pivoting steps. We will discuss these two algorithms as follows:

3.1.1 First Bárány-Onn Algorithm

Algorithm 1 : Solver-Bárány-Onn-1

```
   Input: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
   Output: $\mathbf{T} = \{\mathbf{t}_1, \dots, \mathbf{t}_{d+1}\}$, where $\mathbf{t}_i \in \mathbf{S}_i$
1  begin
2      initialize $\mathbf{T} = \{\mathbf{t}_1, \dots, \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in \mathbf{S}_i$ for $i = 1, \dots, d+1$
3      while $\mathbf{0} \notin \text{conv}(\mathbf{T})$ do
4          $\mathbf{x} \leftarrow \underset{\mathbf{t} \in \text{conv}(\mathbf{T})}{\text{argmin}} (\|\mathbf{t}\|)$
5          find an $i$ such that $\mathbf{x} \in \text{conv}(\mathbf{T} \setminus \{\mathbf{t}_i\})$
6          $\mathbf{t}_i \leftarrow \underset{\mathbf{t} \in \mathbf{S}_i}{\text{argmin}}(\langle \mathbf{t}, \mathbf{x} \rangle)$
7  end
```

Note that Algorithm 1 picks an arbitrary colourful set $\mathbf{T}_1 = \{\mathbf{t}_1, \dots, \mathbf{t}_{d+1}\}$ at first. In the kth iteration, either $\mathbf{0} \in \text{conv}(\mathbf{T}_k)$ and the algorithm stops, or the algorithm involves a minimum norm computation over the conv$(\mathbf{T}_k)$ in step 4, which is a fairly heavy task. After the minimum norm point $\mathbf{x}_k$ is acquired, it is possible to find a colour i to be exchanged in step 5: either conv$(\mathbf{T}_k)$ is full dimensional and $\mathbf{x}_k$ lies on it boundary, or conv$(\mathbf{T}_k)$ has affine dimension less than d; in both cases $\mathbf{x}_k$ can be expressed as a convex combination of at most d points from $\mathbf{T}_k$. Step 6 is to find a new point $\mathbf{t}$ in colour i with $\langle \mathbf{x}_k, \mathbf{t} \rangle$ being minimum and form a new colourful set $\mathbf{T}_{k+1}$.

The following proposition establishes the correctness of Solver-Bárány-Onn-1.

Proposition 3.1.1. *Let $\epsilon > 0$ and $0 \leq \rho \leq 1$, and let $\mathbf{S}_1, \dots, \mathbf{S}_{d+1} \subset \mathbb{R}^d$ be normalized sets of points, each satisfying $\mathbb{B}(\mathbf{0}, \rho) \subset \text{conv}(\mathbf{S}_i)$. Then, when*

Solver-Bárány-Onn-1 is applied, the following recursions hold while $\mathbf{x}_k \neq \mathbf{0}$*:*

$$\textit{If } \rho > 0: \ \|\mathbf{x}_{k+1}\|^2 \leq (1-\rho^2)\|\mathbf{x}_k\|^2; \quad \textit{If } \rho = 0: \ \frac{1}{\|\mathbf{x}_{k+1}\|^2} \geq 1 + \frac{1}{\|\mathbf{x}_k\|^2}.$$

Proof. Consider the kth iteration. The point $\mathbf{q} = -\rho\mathbf{x}_k/|\mathbf{x}_k|$ lies in the ball $\mathbb{B}(\mathbf{0},\rho)$ hence in $\mathrm{conv}(\mathbf{S}_i)$, and satisfies $\langle \mathbf{x}_k, \mathbf{q}\rangle = \langle \mathbf{x}_k, -\rho\mathbf{x}_k/|\mathbf{x}_k|\rangle = -\rho|\mathbf{x}_k|$. Therefore, there must be a point in $\mathbf{S}_i$, in particular the new point $\mathbf{t}_i$ chosen by the algorithm, which satisfies $\langle \mathbf{x}_k, \mathbf{t}_i\rangle \leq -\rho|\mathbf{x}_k|$. Let $\mathbf{p}$ be the projection point of $\mathbf{0}$ onto the line segment $[\mathbf{x}_k, \mathbf{t}_i]$, hence

$$\mathbf{p} = \frac{\langle \mathbf{t}_i - \mathbf{x}_k, \mathbf{t}_i\rangle \mathbf{x}_k + \langle \mathbf{x}_k - \mathbf{t}_i, \mathbf{x}_k\rangle \mathbf{t}_i}{\langle \mathbf{t}_i - \mathbf{x}_k, \mathbf{t}_i - \mathbf{x}_k\rangle} \ \text{ and } \ \|\mathbf{p}\|^2 = \frac{\|\mathbf{x}_k\|^2\|\mathbf{t}_i\|^2 - \langle \mathbf{x}_k, \mathbf{t}_i\rangle^2}{\|\mathbf{x}_k\|^2 + \|\mathbf{t}_i\|^2 - 2\langle \mathbf{x}_k, \mathbf{t}_i\rangle}.$$

Since $\langle \mathbf{x}_k, \mathbf{t}_i\rangle \leq -\rho|\mathbf{x}_k|$, we have

$$\|\mathbf{p}\|^2 \leq \frac{(\|\mathbf{t}_i\|^2 - \rho^2)\|\mathbf{x}_k\|^2}{\|\mathbf{t}_i\|^2 + 2\rho\|\mathbf{x}_k\| + \|\mathbf{x}_k\|^2}.$$

Since the input is normalized, i.e., $\|\mathbf{t}_i\| = 1$, we get

$$\text{if } \rho > 0, \ \|\mathbf{p}\|^2 \leq \frac{(\|\mathbf{t}_i\|^2 - \rho^2)\|\mathbf{x}_k\|^2}{\|\mathbf{t}_i\|^2} \leq (1-\rho^2)\|\mathbf{x}_k\|^2;$$

$$\text{if } \rho = 0, \ \frac{1}{\|\mathbf{p}\|^2} \geq \frac{1}{\|\mathbf{t}_i\|^2} + \frac{1}{\|\mathbf{x}_k\|^2} \geq 1 + \frac{1}{\|\mathbf{x}_k\|^2}.$$

From the fact that $\mathbf{p}$ is on the line segment $[\mathbf{x}_k, \mathbf{t}_i]$, we know that $\mathbf{p} \in [\mathbf{x}_k, \mathbf{t}_i] \subset \mathrm{conv}(\mathbf{T}_{k+1})$. Since $\mathbf{x}_{k+1}$ is defined as the point in $\mathrm{conv}(\mathbf{T}_{k+1})$ of minimum norm, we have $\|\mathbf{x}_{k+1}\| \leq \|\mathbf{p}\|$ and the proposition follows. □

3.1.2 Second Bárány-Onn Algorithm

Finding a point $\mathbf{x}$ of minimum norm in $\mathrm{conv}(\mathbf{T})$ in each iteration of Solver-Bárány-Onn-1 is a time-expensive task which involves the minimization of a

Algorithm 2 : Solver-Bárány-Onn-2

```
    Input: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
    Output: $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$, where $\mathbf{t}_i \in \mathbf{S}_i$
1   begin
2       initialize $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in \mathbf{S}_i$ for $i = 1, \ldots, d+1$
3       $\mathbf{x} \leftarrow \mathbf{t}_1$
4       while $\mathbf{0} \notin \text{conv}(\mathbf{T})$ do
5           find an $i$ such that $\mathbf{x} \in \text{conv}(\mathbf{T} \backslash \{\mathbf{t}_i\})$
6           $\mathbf{t}_i \leftarrow \underset{\mathbf{t} \in \mathbf{S}_i}{\text{argmin}}(\langle \mathbf{t}, \mathbf{x} \rangle)$
7           $\mathbf{p} \leftarrow \underset{[\mathbf{x}, \mathbf{t}_i]}{\text{proj}}(\mathbf{0})$
8           if conv($\mathbf{T}$) is full dimensional containing $\mathbf{p}$ in its interior then
9               compute boundary point $\alpha\mathbf{p}$ of conv($\mathbf{T}$) firstly stabbed by ray $\overrightarrow{\mathbf{0p}}$
10              $\mathbf{p} \leftarrow \alpha\mathbf{p}$
11          $\mathbf{x} \leftarrow \mathbf{p}$
12  end
```

quadratic equation. To avoid this, an efficient variant of the algorithm is presented in [3], in which only linear algebraic computations (such as solutions of linear equations) are required.

Note that Algorithm 2 picks an arbitrary colourful set $\mathbf{T}_1 = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ at first and chooses $\mathbf{t}_1 \in \mathbf{T}_1$ to be $\mathbf{x}_1$ such that $\mathbf{x}_1 = \sum_{i=1}^{d+1} \lambda_i \mathbf{t}_i$ with $\lambda = (1, 0, \ldots, 0)$. In the kth iteration, if $\mathbf{0} \notin \text{conv}(\mathbf{T}_k)$, it is possible to find a colour i in step 5 because $\mathbf{x}_k$ can be expressed as a convex combination of at most d points from $\mathbf{T}_k$: either conv($\mathbf{T}_k$) is full dimensional and $\mathbf{x}_k$ lies on its boundary, or conv($\mathbf{T}_k$) has affine dimension less than d. Step 6 is to find a new point $\mathbf{t}$ in colour i with $\langle \mathbf{x}_k, \mathbf{t} \rangle$ being minimum and construct a new colourful set $\mathbf{T}_{k+1}$. In step 7, the projection point $\mathbf{p}$ of $\mathbf{0}$ onto the line segment $[\mathbf{x}_k, \mathbf{t}]$ is computed. If conv($\mathbf{T}_{k+1}$) happens to contain $\mathbf{p}$ on its boundary or is not full dimensional, then $\mathbf{p}$ is expressible as a convex combination of d or fewer points from $\mathbf{T}_{k+1}$, so $\mathbf{p}$ itself can be taken as the next point $\mathbf{x}_{k+1}$. Otherwise, $\mathrm{conv}(\mathbf{T}_{k+1})$ is a d-simplex containing $\mathbf{p}$ in its interior. In this case it is possible to compute the boundary point $\alpha\mathbf{p}$ of $\mathrm{conv}(\mathbf{T}_{k+1})$ where the simplex is firstly stabbed by the ray from $\mathbf{0}$ to $\mathbf{p}$ as in step 9. This boundary point $\alpha\mathbf{p}$ can then be taken as the next point $\mathbf{x}_{k+1}$.

3.2 Meunier-Deza Algorithm

Frédéric Meunier and Antoine Deza in paper [8] presents an algorithm based on Proposition 2.4.3 to find a colourful simplex containing $\mathbf{0}$ for the sets satisfying the sufficient condition of Theorem 2.4.2, i.e., for *CLPP-TH3*. In this section, we will specify this algorithm in detail and make it easy to implement in Algorithm 3.

Algorithm 3 firstly picks an arbitrary colourful set $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ and, without loss of generality, chooses $c = 1$ to be an outside colour. In the kth iteration, if $\mathbf{0} \in \mathrm{conv}(\mathbf{T})$, the algorithm stops. Otherwise the algorithm obtains d points from $\mathbf{T}$ forming a $\widehat{c}$-transversal $\mathbf{P} = \mathbf{T} \backslash \{\mathbf{t}_c\}$ in step 5. A colourful $(d-1)$-simplex $\boldsymbol{\sigma}$ is generated by the $\widehat{c}$-transversal $\mathbf{P}$ in step 6 and a ray $\mathbf{r}$ is selected from the origin $\mathbf{0}$ towards point $\mathbf{x}$ which is in the interior of $\boldsymbol{\sigma}$ in step 7. Thus, the ray $\mathbf{r}$ intersects $(d-1)$-simplex $\boldsymbol{\sigma}$ in its interior. In the following steps, the algorithm selects a distinct point $\mathbf{t}$ from colour c each time, and then the $\widehat{c}$-transversal $\mathbf{P}$ and the point $\mathbf{t}$ form a new colourful set $\mathbf{T}$ which constructs a new colourful d-simplex having $\boldsymbol{\sigma}$ as a facet. If $\mathbf{0} \in \mathrm{conv}(\mathbf{T})$, the algorithm stops. Otherwise, if there is a facet $\boldsymbol{\tau}$ of the colourful d-simplex intersecting ray $\mathbf{r}$ before facet $\boldsymbol{\sigma}$, a closer facet is found in the d-simplex and the outside colour c is set to be the colour without appearing in the facet $\boldsymbol{\tau}$, and then the algorithm continues from step 4. If none of the two cases happens, the algorithm goes to the subroutine Find-Closer-Facet.

Note that in step 13 the algorithm to find a facet $\boldsymbol{\tau} \subset \text{conv}(\mathbf{T})$ intersecting ray $\overrightarrow{\mathbf{0x}}$ before facet $\boldsymbol{\sigma}$ will be discussed in Chapter 4.

Algorithm 3 : Solver-Meunier-Deza

```
    Input: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
    Output: $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$, where $\mathbf{t}_i \in \mathbf{S}_i$
1   begin
2       initialize $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in \mathbf{S}_i$ for $i = 1, \ldots, d+1$
3       $c \leftarrow 1$
4       while $\mathbf{0} \notin \text{conv}(\mathbf{T})$ do
5           $\mathbf{P} \leftarrow \mathbf{T} \backslash \{\mathbf{t}_c\}$
6           $\boldsymbol{\sigma} \leftarrow \text{conv}(\mathbf{P})$
7           $\mathbf{x} \leftarrow$ interior point of $\boldsymbol{\sigma}$
8           $f \leftarrow$ false
9           for point $\mathbf{t} \in \mathbf{S}_c$ do
10              $\mathbf{T} \leftarrow \mathbf{P} \cup \{\mathbf{t}\}$
11              if $\mathbf{0} \in \text{conv}(\mathbf{T})$ do
12                  return
13              else if there is a facet $\boldsymbol{\tau} \subset \text{conv}(\mathbf{T})$ intersecting ray $\overrightarrow{\mathbf{0x}}$ before $\boldsymbol{\sigma}$
14                  $c \leftarrow$ the colour not included in $\boldsymbol{\tau}$
15                  $f \leftarrow$ true
16                  break
17          if $f \neq$ true
18              $\mathbf{T} \leftarrow$ Find-Closer-Facet$(\mathbf{T}, c, \mathbf{x})$
19  end
```

Subroutine 1 : Find-Closer-Facet

```
    Input: $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$, where $\mathbf{t}_i \in \mathbf{S}_i$,
           $c$ = outside colour,
           $\mathbf{x}$ = interior point of conv($\mathbf{T} \backslash \mathbf{t}_c$)
    Output: $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$, where $\mathbf{t}_i \in \mathbf{S}_i$
1   begin
2       $\mathbf{U} \leftarrow \mathbf{T} \backslash \{\mathbf{t}_c\}$
3       for $i \leftarrow 1$ to $d+1$ except $c$ do
4           find a point $\mathbf{t} \in \mathbf{S}_i$ such that $\mathbf{t} \in H^+(\mathbf{U})$
5           $\mathbf{V} \leftarrow \mathbf{t}$
6       $\mathbf{y} \leftarrow$ antipode of $\mathbf{x}$
7       for $i \leftarrow 0$ to $(2^d - 2)$ do
8           convert $i$ to a binary number $b$
9           for $j \leftarrow 1$ to $d$ do
10              if $b_j = 0$ then
11                  $\mathbf{W} \leftarrow \mathbf{v}_j$, where $\mathbf{v}_j \in \mathbf{V}$
12              else
13                  $\mathbf{W} \leftarrow \mathbf{u}_j$, where $\mathbf{u}_j \in \mathbf{U}$
14          $\mathbf{T} \leftarrow \mathbf{W} \cup \{\mathbf{t}_c\}$
15          if $\mathbf{0} \in$ conv($\mathbf{T}$) then
16              return
17          $\mathbf{T} \leftarrow \mathbf{W} \cup \{\mathbf{y}\}$
18          if $\mathbf{0} \in$ conv($\mathbf{T}$) then
19              $\mathbf{T} \leftarrow \mathbf{W} \cup \{\mathbf{t}_c\}$
20              return
21  end
```

In Subroutine 1, the vertices of $(d-1)$-simplex $\boldsymbol{\sigma}$ form an upper $\widehat{c}$-transversal $\mathbf{U}$ in step 2. Let $H^+(\mathbf{U})$ be the open half space delimited by aff$(\mathbf{U})$ and containing $\mathbf{0}$. From each colour except the outside colour c, the subroutine finds a point $\mathbf{t}$ which is in the half space of $H^+(\mathbf{U})$ in step 4. All of these points form a lower $\widehat{c}$-transversal $\mathbf{V}$ in step 5. The $\widehat{c}$-transversal $\mathbf{V}$ is guaranteed to exist because of the sufficient condition of Theorem 2.4.2: for any $i \neq c$, $(\mathbf{S}_i \cup \mathbf{S}_c) \cap H^+(\mathbf{U}) \neq \varnothing$, i.e., for each colour i, there is a point either in $\mathbf{S}_c \cap H^+(\mathbf{U})$ or in $(\mathbf{S}_i \backslash \{\mathbf{u}_i\}) \cap H^+(\mathbf{U})$. From the main routine, we know that the algorithm enters into the subroutine because of $\mathbf{S}_c \cap H^+(\mathbf{U}) = \varnothing$, therefore we have $(\mathbf{S}_i \backslash \{\mathbf{u}_i\}) \cap H^+(\mathbf{U}) \neq \varnothing$ for each colour $i \neq c$. Considering point $\mathbf{y}$ of the antipode of $\mathbf{x}$ in step 6 and point $\mathbf{t}_c$ of colour c, we get $\mathbf{0} \in \text{conv}(\mathbf{U} \cup \{\mathbf{y}\})$ and $\mathbf{0} \notin \text{conv}(\mathbf{U} \cup \{\mathbf{t}_c\})$. Enumerate a colourful $(d-1)$-simplex by taking vertices from the two $\widehat{c}$-transversals $\mathbf{U}$ and $\mathbf{V}$ with at least one vertex in $\mathbf{V}$ and without any two vertices being of same colour. From the Proposition 2.4.3, we know that there exists a new colourful d-simplex containing $\mathbf{0}$, which is formed by a colourful $(d-1)$-simplex enumerated and point $\mathbf{t}_c$ or $\mathbf{y}$. If $\mathbf{t}_c$ is a vertex of the new d-simplex, we do find a colourful simplex containing $\mathbf{0}$; otherwise, if $\mathbf{y}$ is a vertex of the new d-simplex, the facet of the new d-simplex not containing $\mathbf{y}$ is a colourful $(d-1)$-simplex $\boldsymbol{\tau}$ intersecting ray $\mathbf{r}$ before facet $\boldsymbol{\sigma}$ since aff$(\mathbf{U})$ forms the boundary of $H^+(\mathbf{U})$. In either case, the subroutine returns to main routine.

3.3 Random-Picking Algorithm

In [5], a simple guess and check algorithm was proposed where colourful set $\mathbf{T}$ is randomly sampled until one conv$(\mathbf{T})$ is found covering $\mathbf{0}$. This algorithm can be used to solve general *CLPP* in low dimension.

Algorithm 4 : Solver-Random-Pick

```
    Input: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
    Output: $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$, where $\mathbf{t}_i \in \mathbf{S}_i$
1   begin
2       initialize $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in \mathbf{S}_i$ for $i = 1, \ldots, d+1$
3       while $\mathbf{0} \notin \text{conv}(\mathbf{T})$ do
4           for $i \leftarrow 1$ to $d+1$ do
5               $\mathbf{t}_i \sim \mathbf{S}_i$
6   end
```

Note that in Algorithm 4 the symbol "$\sim$" denotes the operation of selecting an element from a point set, or colour $\mathbf{S}_i$, randomly and uniformly.

Algorithm 4 first arbitrarily selects a colourful set $\mathbf{T}$ in step 2 and tests if its convex hull covers origin $\mathbf{0}$. If not, the algorithm randomly selects a point from each colour to form another colourful set $\mathbf{T}$ in step 4-5 and tests if its convex hull covers origin $\mathbf{0}$. This process will continue until a colourful set $\mathbf{T}$ containing $\mathbf{0}$ in its convex hull is found.

Intuitively we would not expect Solver-Random-Pick to find the solution efficiently for general *CLPP* cases. However, as discussed in [9]–[12], solutions to a given colourful linear programming problem may not be all that rare, and in some cases can be quite frequent. Since guessing and checking are relatively fast operations, it is possible that this naive algorithm may perform well in special cases or low dimension.

Chapter 4

Software Implementation for the Algorithms

The algorithms introduced in Chapter 3 are implemented by MATLAB language. This chapter will describe the implementation details for the algorithms including the data structure and some key subroutines.

4.1 Data Structure

4.1.1 Input Data Structure

We use a pair of matrix variables named *Pts* and *ColorPartition* together to represent a colourful configuration $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$.

Pts: an $d-$by$-(d+1)$ matrix variable with each column storing the coordinates of a colourful point from $\mathbf{S}$. Each point is numbered by column index, and all the points of a colour are bunched together and stored in consecutive columns of the matrix. This variable is a mandatory input for all algorithms.

ColorPartition: an $1-$ by $-\ (d+1)$ matrix, and the ith element holds the number of points of colour $\mathbf{S}_i$. This variable is optional. If omitted, the program assumes that every colour has $d+1$ points.

b: a point being covered by a colourful simplex. It is an optional input. If omitted, the program assumes that b is the origin $\mathbf{0}$.

options.initT: an $1-\text{by}-(d+1)$ matrix which contains the point indices of an initial colourful simplex $\mathbf{T}$. It is an optional input. If not given, the program will choose every first point of each colour to form a colourful simplex $\mathbf{T}$.

Example 4.1.1 (3 colours with 3 points of each colour in $\mathbb{R}^2$)**.** The colourful configuration $\mathbf{S}$ is:

$$\mathbf{S}_1 = \left\{ \begin{bmatrix} -0.8428 \\ 0.5383 \end{bmatrix}, \begin{bmatrix} -0.3821 \\ 0.9241 \end{bmatrix}, \begin{bmatrix} -0.5012 \\ 0.8654 \end{bmatrix} \right\}$$
$$\mathbf{S}_2 = \left\{ \begin{bmatrix} 0.4405 \\ -0.8978 \end{bmatrix}, \begin{bmatrix} -0.3401 \\ -0.9404 \end{bmatrix}, \begin{bmatrix} -0.9975 \\ 0.0711 \end{bmatrix} \right\}$$
$$\mathbf{S}_3 = \left\{ \begin{bmatrix} 0.6296 \\ -0.7769 \end{bmatrix}, \begin{bmatrix} 0.9532 \\ 0.3024 \end{bmatrix}, \begin{bmatrix} 0.6623 \\ 0.7492 \end{bmatrix} \right\}.$$

And above points are normalized onto the unit sphere $\mathbb{S}^{d-1}$ which is centered at origin $\mathbf{0}$ as shown in Figure 4.1.

These points are inputted into solvers as:

Pts=[-0.8428 -0.3821 -0.5012 0.4405 -0.3401 -0.9975 0.6296 0.9532 0.6623;
0.5383 0.9241 0.8654 -0.8978 -0.9404 0.0711 -0.7769 0.3024 0.7492].

The solvers set *ColorPartition*=$[3,3,3]$, b=$[0,0,0]^{\mathrm{T}}$, and *options.initT* $=[1,4,7]$ as default values if they are not given in the input.

The initial colourful simplex $\mathbf{T}$ is represented by the dash lines in Figure 4.1.

4.1.2 Internal Data Structure

ColorMap is an $(d+1)$-structure array. Each array's index corresponds to a colour and the elements of an array hold the indices of points of a colour in *Pts*.

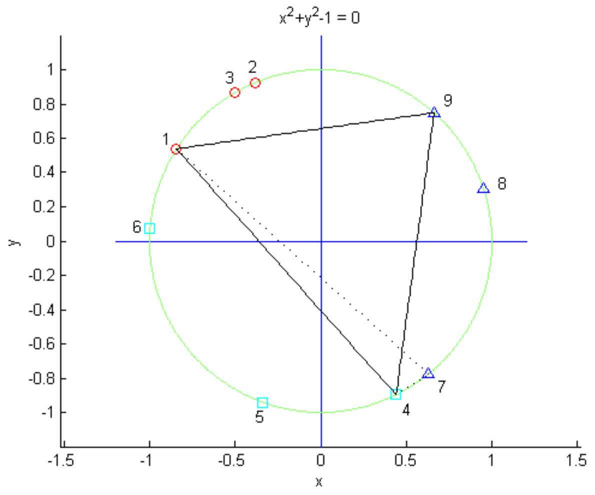

Figure 4.1: 3 colours with 3 points of each colour in $\mathbb{R}^2$.

For instance, in Example 4.1.1, *ColorMap* can be expressed as: $ColorMap(1).list = [1, 2, 3], ColorMap(2).list = [4, 5, 6], ColorMap(3).list = [7, 8, 9]$

4.1.3 Output Data Structure

T: an array which holds the point indices of a colourful simplex covering **0**.

x: a column array which contains the coefficients of the convex combination of all points in *Pts*.

info: a structure which includes the computing status, such as iterations of finding a solution, time being used, and whether a solution is feasible.

In Example 4.1.1, the results are shown as follows:

T=[1, 4, 9],

x=[0.3802, 0, 0, 0.4062, 0, 0, 0, 0, 0.2136]$^{\mathrm{T}}$,

info=[iter: 2, time: 0.0960, feasible: 1]$^{\mathrm{T}}$.

The final solution of a colourful simplex covering the origin **0** is represented by the solid lines in Figure 4.1.

4.2 Normalizing the Input Points

The solvers firstly translate the input point b to the origin $\mathbf{0}$. Then every point in *Pts* will be translated to $\mathbf{0}$ and scaled to the unit sphere $\mathbb{S}(\mathbf{0}, 1)$, i.e., the points are normalized. These operations can be implemented by following formula:

$$Pts(i) = (Pts(i) - b)/\texttt{norm}(Pts(i) - b) \tag{4.2.1}$$

where `norm` is the function in MATLAB to calculate norms of a vector.

4.3 Testing whether a Colourful Simplex Covers the Origin 0

During a solver being executed, it selects a colourful set $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ in each iteration to test whether the origin $\mathbf{0}$ is covered by $\text{conv}(\mathbf{T})$. This testing is implemented by following steps:

Firstly, we use the function `linsolve` in MATLAB to solve following linear system:

$$\begin{bmatrix} \mathbf{t}_1 & \ldots & \mathbf{t}_{d+1} \\ 1 & \ldots & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{4.3.2}$$

where $\mathbf{x} = [x_1, x_2, \cdots, x_{d+1}]^{\mathrm{T}}$ is the variable.

Secondly, determine if the linear system is feasible from the output of `linsolve`. If the system is feasible, we can use $\mathbf{x}$ to determine whether $\mathbf{0} \in \text{conv}(\mathbf{T})$: If all coordinates of $\mathbf{x}$ are not less than 0, we get a colourful set $\mathbf{T}$ containing $\mathbf{0}$ in its convex hull; otherwise, we can conclude that $\mathbf{0} \notin \text{conv}(\mathbf{T})$.

Thirdly, if the linear system is not feasible, i.e., the colourful set $\mathbf{T}$ is not in general position, we use the function `linprog` in MATLAT to solve following linear optimization problem:

$$\begin{aligned} \min: \quad & \mathbf{0}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{4.3.3}$$

where $\mathbf{x} = [x_1, x_2, \cdots, x_{d+1}]^{\mathrm{T}}$ is the variable.

If the linear optimization problem has feasible solution, we obtain a colourful set $\mathbf{T}$ containing $\mathbf{0}$ in its convex hull; otherwise, the linear problem is infeasible and we conclude that $\mathbf{0} \notin \mathrm{conv}(\mathbf{T})$.

In our implementations, we use double precision floating number to represent a number in $\mathbb{R}$ and the coordinates of a point in $\mathbb{R}^d$. Since the arithmetic operations introduce round off errors, a constant variable `TOLERANCE` is defined as numerical error tolerance which is a very small positive value. When a number is less than this value, it can be considered zero. In our program, we set `TOLERANCE` to be 10^{-10}, since from testing we observed that if setting `TOLERANCE` less than 10^{-10} some cases will produce loop conditions for the algorithms.

The maximum iteration, which is the maximum pivoting steps to search for a colourful simplex covering 0, is set to be 1,000,000.

4.4 Computing the Minimum Norm Point in conv(T)

In each iteration of Solver-Bárány-Onn-1, it needs to find the point $\mathbf{x}$ of minimum norm in $\mathrm{conv}(\mathbf{T})$, which involves the minimization of a quadratic equation.

Since the point $\mathbf{x}$ is in $\operatorname{conv}(\mathbf{T})$, it can be expressed as

$$\mathbf{x} = \sum_{i=1}^{d+1} \lambda_i \mathbf{t}_i \tag{4.4.4}$$

with $\sum_{i=1}^{d+1} \lambda_i = 1$ and $\lambda_i \geq 0$ for $1 \leq i \leq d+1$, and $\mathbf{t}_i \in \mathbf{T}$. Meanwhile, the square of the norm of $\mathbf{x}$ can be expressed as

$$\|\mathbf{x}\|^2 = \sum_{i=1}^{d} x_i^2 \tag{4.4.5}$$

and we want $\|\mathbf{x}\|^2$ to be minimum. Substitute Equation 4.4.4 into the right hand of Equation 4.4.5, we get following quadratic optimization problem with $\lambda_1, \ldots, \lambda_{d+1}$ as variables.

$$\begin{aligned} \text{min:} \quad & \sum_{1 \leq i,j \leq d+1} \mathbf{t}_i^{\mathrm{T}} \mathbf{t}_j \lambda_i \lambda_j \\ \text{s.t.} \quad & \sum_{i=1}^{d+1} \lambda_i = 1 \\ & \lambda_i \geq 0 \quad \text{for} \quad 1 \leq i \leq d+1 \end{aligned} \tag{4.4.6}$$

We use the function `quadprog` in MATLAB to solve this quadratic optimization problem and it is a time-expensive task to solve this problem. After the solution $[\lambda_1, \ldots, \lambda_{d+1}]^{\mathrm{T}}$ is found , the point $\mathbf{x}$ of minimum norm can be obtained by $\mathbf{x} = \sum_{i=1}^{d+1} \lambda_i \mathbf{t}_i$ and there is at lease one λ_i $(1 \leq i \leq d+1)$ being 0 because $\mathbf{x}$ is on the boundary of $\operatorname{conv}(\mathbf{T})$.

4.5 Selecting an Interior Point in Facet $\boldsymbol{\sigma}$

In Solver-Meunier-Deza, it needs to select an interior point of a facet $\boldsymbol{\sigma} \subset \operatorname{conv}(\mathbf{T})$. Assuming the facet $\boldsymbol{\sigma}$ has vertices $\mathbf{t}_1, \ldots, \mathbf{t}_{i-1}, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{d+1}$ in $\mathbb{R}^d$

and colour i is not included in the facet, then the interior point can be selected as the center point of facet $\boldsymbol{\sigma}$, which can be calculated by following formula:

$$\mathbf{p} = (\sum_{j=1,\ j\neq i}^{d+1} \mathbf{t}_j)/d. \tag{4.5.7}$$

4.6 Computing Intersection of a Ray r with a Facet of conv(T)

Proposition 4.6.1. *Let set* $\mathbf{T} = \{\mathbf{t}_1, \dots, \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ *in general position and*

$$\begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{y}_1^{\mathrm{T}} & b_1 \\ \dots & \dots \\ \mathbf{y}_{d+1}^{\mathrm{T}} & b_{d+1} \end{bmatrix}, \tag{4.6.8}$$

where $\mathbf{y}_i \in \mathbb{R}^d$ *and* $b_i \in \mathbb{R}$ *for* $1 \leq i \leq d+1$*, then we have*

$$\mathbf{y}_i^{\mathrm{T}}\mathbf{t}_i = 1 - b_i \quad \textit{for} \quad 1 \leq i \leq d+1; \tag{4.6.9}$$

$$\mathbf{y}_i^{\mathrm{T}}\mathbf{t}_j = -b_i \quad \textit{for} \quad 1 \leq j \neq i \leq d+1. \tag{4.6.10}$$

Proof. According to the properties of inverse matrix, we have

$$\begin{bmatrix} \mathbf{y}_1^{\mathrm{T}} & b_1 \\ \dots & \dots \\ \mathbf{y}_{d+1}^{\mathrm{T}} & b_{d+1} \end{bmatrix} \begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}. \tag{4.6.11}$$

Perform the row-column multiplication of the left hand of Equation 4.6.11 and compare each element to the right hand of the equation, we can get the result of the proposition. □

Corollary 4.6.2. *Let colourful set* $\mathbf{T} = \{\mathbf{t}_1, \dots, \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ *in general position and* $\mathbf{T}_i$ *be an* $\widehat{i}$*-transversal of colours* $\mathbf{S}_1, \dots \mathbf{S}_{d+1}$*, then the equation of the* *hyperplane passing through $\widehat{i}$-transversal $\mathbf{T}_i$ is:*

$$\mathbf{y}_i^{\mathrm{T}}\mathbf{x} = -b_i \tag{4.6.12}$$

where $\mathbf{y}_i$ and b_i are the elements of Equation 4.6.8.

Proof. From the Equation 4.6.10, we know that every point of the $\widehat{i}$-transversal $\mathbf{T}_i$ satisfies the Equation 4.6.12. Therefore the conclusion can be drawn. □

Corollary 4.6.3. *Given a colourful set $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ in general position, and a ray $\mathbf{r}$ originating from $\mathbf{0}$ towards point $\mathbf{p}$. If the ray $\mathbf{r}$ intersects a hyperplane passing through a $(d-1)$-simplex $\boldsymbol{\tau}$ formed by an $\widehat{i}$-transversal $\{\mathbf{t}_1, \ldots, \mathbf{t}_{i-1}, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{d+1}\}$ for $1 \leq i \leq d+1$, then the intersection point is $\alpha\mathbf{p}$ and*

$$\alpha = \frac{-b_i}{\mathbf{y}_i^{\mathrm{T}}\mathbf{p}}. \tag{4.6.13}$$

where $\mathbf{y}_i$ and b_i are the elements of Equation 4.6.8.

Proof. From Corollary 4.6.2, we know the equation of the hyperplane, which passes through an $\widehat{i}$-transversal $\{\mathbf{t}_1, \ldots, \mathbf{t}_{i-1}, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{d+1}\}$ for $1 \leq i \leq d+1$, is $\mathbf{y}_i^{\mathrm{T}}\mathbf{x} = -b_i$. The equation of the ray $\mathbf{r}$ is $\mathbf{x} = \alpha\mathbf{p}$. Substituting the ray's equation into hyperplane's equation, we get $\mathbf{y}_i^{\mathrm{T}}\alpha\mathbf{p} = -b_i$. Since the ray intersects the hyperplane, i.e., $\mathbf{y}_i^{\mathrm{T}}\mathbf{p} \neq 0$, we get $\alpha = \frac{-b_i}{\mathbf{y}_i^{\mathrm{T}}\mathbf{p}}$. □

Assuming $\mathbf{p}$ is an interior point of a colourful $(d-1)$-simplex $\boldsymbol{\sigma}$, if the ray $\overrightarrow{\mathbf{0p}}$ intersects the hyperplane passing through an $(d-1)$-simplex $\boldsymbol{\tau}$ at point $\alpha\mathbf{p}$ with $0 < \alpha < 1$, we can say the intersection point $\alpha\mathbf{p}$ is between $\mathbf{0}$ and $\mathbf{p}$. In order to determine whether $\boldsymbol{\tau}$ intersects the ray $\overrightarrow{\mathbf{0p}}$ before $\boldsymbol{\sigma}$, we need test if the point $\alpha\mathbf{p}$ is in $\boldsymbol{\tau}$. The task of testing if $\alpha\mathbf{p}$ is in $\boldsymbol{\tau}$ can be achieved by using the function `linsolve` in MATLAB to solve following linear equation

$$[\mathbf{t}_1\ \dots\ \mathbf{t}_{i-1}\ \mathbf{t}_{i+1}\ \dots\ \mathbf{t}_{d+1}]\mathbf{x} = \alpha\mathbf{p} \tag{4.6.14}$$

where $\mathbf{x} = [x_1, x_2, \cdots, x_d]^{\mathrm{T}}$ is the variable, and $\sum_{k=1}^{d} x_k = 1$ with $x_k \geq 0$ for $1 \leq k \leq d$. If the solution exists, we can determine that the intersection point $\alpha\mathbf{p}$ is in the $(d-1)$-simplex $\boldsymbol{\tau}$.

Algorithm 5 illustrates the method of finding a facet $\boldsymbol{\tau} \subset \mathrm{conv}(\mathbf{T})$ intersecting ray $\overrightarrow{\mathbf{0p}}$ before facet $\boldsymbol{\sigma}$, which is required by Solver-Meunier-Deza.

Algorithm 5 : Find closer facet $\boldsymbol{\tau} \subset \mathrm{conv}(\mathbf{T})$

```
   Input: $\mathbf{T} = \{\mathbf{t}_1, \dots, \mathbf{t}_{d+1}\}$,
          $\mathbf{p}$ = interior point in facet $\boldsymbol{\sigma} \subset \mathrm{conv}(\mathbf{T})$,
          $c$ = colour not included in facet $\boldsymbol{\sigma}$
   Output: closer facet $\boldsymbol{\tau} \subset \mathrm{conv}(\mathbf{T})$
1  begin
2      $\begin{bmatrix} \mathbf{y}_1^{\mathrm{T}} & b_1 \\ \dots & \dots \\ \mathbf{y}_{d+1}^{\mathrm{T}} & b_{d+1} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix}^{-1}$
3      for $i \leftarrow 1$ to $d+1$ except $c$ do
4          $m \leftarrow \mathbf{y}_i^{\mathrm{T}}\mathbf{p}$
5          if $|m| > 0$
6              $\alpha \leftarrow -b_i/m$
7              if $0 < \alpha < 1$
8                  solve linear equation $[\mathbf{t}_1\ \dots\ \mathbf{t}_{i-1}\ \mathbf{t}_{i+1}\ \dots\ \mathbf{t}_{d+1}]\mathbf{x} = \alpha\mathbf{p}$
9                  if $\sum_{k=1}^{d} x_k = 1$ and $x_k \geq 0$
10                     return the facet not including colour $i$
11 end
```

4.7 Finding a Point $\mathbf{v} \in \mathbf{S}_i$ such that $\mathbf{v} \in H^+(\mathbf{U})$

In the subroutine Find-Closer-Facet of Solver-Meunier-Deza, every colour $\mathbf{S}_i$ except the outside colour needs to find a point $\mathbf{v}_i \in \mathbf{S}_i$ such that the point $\mathbf{v}_i$ is in the open half space $H^+(\mathbf{U})$ delimited by aff($\mathbf{U}$) and containing $\mathbf{0}$, where, without loss of generality, $\mathbf{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$ is a $\widehat{d+1}$-transveral. This task can be fulfilled by following method.

By arbitrarily selecting a point $\mathbf{t}$ from the outside colour $\mathbf{S}_{d+1}$, the point set $\{\mathbf{u}_1, \ldots, \mathbf{u}_d, \mathbf{t}\}$ forms a colourful set with $n+1$ points. From the Corollary 4.6.2, we know that the equation of the hyperplane passing through $\widehat{d+1}$-transversal $\mathbf{U}$ is:

$$\mathbf{y}_{d+1}^{\mathrm{T}}\mathbf{x} = -b_{d+1} \tag{4.7.15}$$

where $\mathbf{y}_{d+1}$ and b_{d+1} are the elements of

$$\begin{bmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_d & \mathbf{t} \\ 1 & \ldots & 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{y}_1^{\mathrm{T}} & b_1 \\ \ldots & \ldots \\ \mathbf{y}_d^{\mathrm{T}} & b_d \\ \mathbf{y}_{d+1}^{\mathrm{T}} & b_{d+1} \end{bmatrix}.$$

For a point $\mathbf{v}$ in colour $\mathbf{S}_i$, if $-b_{d+1} \geq 0$ and $\mathbf{y}_{d+1}^{\mathrm{T}}\mathbf{v} < -b_{d+1}$, or $-b_{d+1} < 0$ and $\mathbf{y}_{d+1}^{\mathrm{T}}\mathbf{v} > -b_{d+1}$, we can determine that the point $\mathbf{v} \in H^+(\mathbf{U})$.

4.8 Converting an Integer to a Binary Number

In the subroutine Find-Closer-Facet of Solver-Meunier-Deza, step 8 needs to convert a non-negative integer n to a binary value b. We employ the function `bitget`(n, bit) in MATLAB to finish this job, where n is an integer to be converted and *bit* is the maximum bits to hold a binary value. Now, the maximum value of *bit* is limited to be 52, i.e., the biggest integer which can be converted into a binary value is 2^{52}. This integer is big enough for our software to perform the enumeration operation. However, if the number of colours is greater than 53, the number of bits of the binary value will be more than 52. In this case, the function $\texttt{bitget}(n, bit)$ can only produce the first 52 bits of the binary value; for the rest of bits beyond 52-bit, we can fill them with zero because in the softwere we set the maximum iteration to be 1,000,000 and the software already terminates when the enumeration operation could reach 2^{52}.

4.9 Generating Random Integer and Real Numbers

In the algorithm of Solver-Random-Pick and the random case generators in Chapter 5, random integer and real numbers need to be generated. The following methods introduce how to generate random numbers.

For real numbers, the following formula generates an m-by-n random real number matrix whose elements are in the interval (r_1, r_2):

$$R = r_1\texttt{ones}(m, n) + (r_2 - r_1)\texttt{rand}(m, n), \tag{4.9.16}$$

where $\texttt{rand}(m, n)$ is a MATLAB function which generates an m-by-n random matrix whose elements are uniformly distributed in the interval $(0, 1)$, and (r_1, r_2) is the interval of the real numbers.

For integer numbers, The following formula generates an m-by-n random integer number matrix whose elements are in the interval $[r_1, r_2]$:

$$R = r_1\texttt{ones}(m, n) + \texttt{floor}((r_2 - r_1 + 0.99999)\texttt{rand}(m, n)), \tag{4.9.17}$$

where $\texttt{ones}(m, n)$ is a MATLAB function generating an m-by-n matrix with all elements being one, $\texttt{floor}(A)$ is a MATLAB function which rounds elements of A to the nearest integers less than or equal to A, and $[r_1, r_2]$ is the interval of the integer numbers with r_1 and r_2 being integers.

Chapter 5

Random Case Generators

This Chapter will describe the algorithms and implementation details of the random case generators which satisfy the three kinds of sufficient conditions of colourful linear programming problem: *CLPP-TH1*, *CLPP-TH2* and *CLPP-TH3* respectively. With these random case generators, we can carry on the benchmark testing for the algorithms which are proposed in Chapter 3 and compare their performances.

We assume that, for each case, $d+1$ colours need to be generated and each colour has $d+1$ points which are normalized onto unit sphere $\mathbb{S}^{d-1}$, i.e., $\|\mathbf{s}\| = 1$ for $\mathbf{s} \in \bigcup_{i=1}^{d+1} \mathbf{S}_i$ in $\mathbb{R}^d$. And we express the d+1 points of colour $\mathbf{S}_i$ for $1 \leq i \leq d+1$ in $\mathbb{R}^d$ as $\{\mathbf{s}_1^i, \mathbf{s}_2^i, \ldots, \mathbf{s}_{d+1}^i\}$ and $\mathbf{s}_j^i(k)$ is the kth coordinate of the point $\mathbf{s}_j^i$ for $1 \leq j \leq d+1$ and $1 \leq k \leq d$.

The symbol "$\sim$" is used to denote the operation of randomly and uniformly selecting an element from a set which has finite or infinite, bounded or unbounded, elements. Here we introduce the concept of *random convex combination* of a set $\mathbf{T}$, which means, for every point $\mathbf{t}_i$ of $\mathbf{T}$ for $1 \leq i \leq d+1$ a real number $\lambda_i \in (0,1)$ is randomly generated to be as a coefficient of $\mathbf{t}_i$, and then the convex combination of $\mathbf{T}$ is computed as follows: $\mathbf{t} = \sum_{i=1}^{d+1} \lambda_i \mathbf{t}_i / \sum_{i=1}^{d+1} \lambda_i$. This operation can be expressed as: $\mathbf{t} \sim \text{conv}(\mathbf{T})$. The methods of generating random integer and real numbers have been discussed in Chapter 4.

5.1 Random Case Generator of *CLPP-TH1*

The sufficient condition of *CLPP-TH1* is $\mathbf{0} \in \bigcap_{i=1}^{d+1} \text{conv}(\mathbf{S}_i)$. In order to generate $d+1$ colours with $d+1$ points for each colour to satisfy this sufficient condition, we randomly generate d points for a colour $\mathbf{S}_i$ $(i = 1, \ldots, d+1)$, and then the last point of this colour can be achieved by the random convex combination of the antipodes of the first d points of this colour. The algorithm is shown as follows:

Algorithm 6 : Random-Generator-*CLPP-TH1*

```
  Input: $d$
  Output: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
1 begin
2     for $i \leftarrow 1$ to $d+1$ do
3         $k \sim \{1, \ldots, d+1\}$
4         for $j \in \{1, \ldots, d+1\} \backslash \{k\}$ do
5             $\mathbf{s}_j^i \sim \mathbb{S}^{d-1}$
6         $\mathbf{s}_k^i \sim \text{conv}(\mathbf{s}_1^i, \ldots, \mathbf{s}_{k-1}^i, \mathbf{s}_{k+1}^i, \ldots, \mathbf{s}_{d+1}^i)$
7         $\mathbf{s}_k^i \leftarrow -\mathbf{s}_k^i / \|\mathbf{s}_k^i\|$
8 end
```

In Algorithm 6, step 2 repeatedly generates the points for colour $\mathbf{S}_i$ from 1 to $d+1$. In step 3, an integer number k is randomly selected from $\{1, \ldots, d+1\}$. Step 4-5 randomly generate d points for the ith colour and scale them onto the unit sphere $\mathbb{S}^{d-1}$. To fulfill this, we employ the MATLAB function `rand`$(d, 1)$ to generate a random point in $\mathbb{R}^d$ and normalize it. In step 6-7, the algorithm computes the random convex combination of the antipodes of these d points, then normalizes this value and sets it as the kth point of the colour $\mathbf{S}_i$. The value of the random convex combination guarantees that the colour $\mathbf{S}_i$ contains origin $\mathbf{0}$ in its convex hull, i.e., $\mathbf{0} \in \text{conv}(\mathbf{S}_i)$. Therefore, the cases generated by this algorithm satisfy the sufficient condition of *CLPP-TH1*.

Figure 5.1 illustrates a case in $\mathbb{R}^2$ which is generated by the Random Case Generator of *CLPP-TH1*.

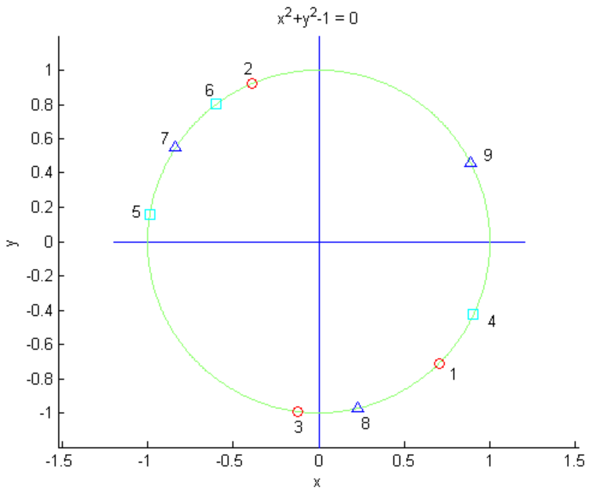

Figure 5.1: A case generated by the Random Case Generator of *CLPP-TH1*.

5.2 Random Case Generator of *CLPP-TH2*

The sufficient condition of *CLPP-TH2* is $\mathbf{0} \in \bigcap_{1\le i<j\le d+1} \text{conv}(\mathbf{S}_i \cup \mathbf{S}_j)$. In order to generate $d+1$ colours with $d+1$ points for each colour to satisfy this sufficient condition, we construct the colours in following sequence: for the first colour $\mathbf{S}_1$, randomly generate $d+1$ points, then generate all the first points for the rest of colours by selecting different d points from the colour $\mathbf{S}_1$ each

time and computing the random convex combination of the antipodes of these d points; for the colour $\mathbf{S}_2$, randomly generate all the points which are not produced by the previous colour, and then generate all the second points for all the next colours by choosing different d points from this colour each time and computing the random convex combination of the antipodes of these d points; and so on. The algorithm is shown as follows:

Algorithm 7 : Random-Generator-*CLPP-TH2*

```
    Input: $d$
    Output: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
1   begin
2       for $i \leftarrow 1$ to $d+1$ do
3           for $j \leftarrow i$ to $d+1$ do
4               $\mathbf{s}_j^i \sim \mathbb{S}^{d-1}$
5           for $k \leftarrow i+1$ to $d+1$ do
6               $\mathbf{s}_i^k \sim \text{conv}(\mathbf{s}_1^i, \dots, \mathbf{s}_{k-1}^i, \mathbf{s}_{k+1}^i, \dots, \mathbf{s}_{d+1}^i)$
7               $\mathbf{s}_i^k \leftarrow -\mathbf{s}_i^k / \|\mathbf{s}_i^k\|$
8   end
```

In Algorithm 7, step 2 repeatedly generates the points for colour $\mathbf{S}_i$ from 1 to $d+1$. Step 3-4 randomly generate the rest of points for the ith colour, which are not produced by the previous colours, and scale them onto the unit sphere $\mathbb{S}^{d-1}$. In step 5-7, for every next colour $\mathbf{S}_k$ for $i < k \leq d+1$, the algorithm gets d points from colour $\mathbf{S}_i$ except the point $\mathbf{s}_k^i$ and computes the random convex combination of the antipodes of these d points, then normalizes this value and sets it as the ith point of the colour $\mathbf{S}_k$. The value of the random convex combination guarantees that the colour $\mathbf{S}_i$ and one of its next colours $\mathbf{S}_k$ ($k > i$) contain origin $\mathbf{0}$ in their convex hull, i.e., $\mathbf{0} \in \text{conv}(\mathbf{S}_i \cup \mathbf{S}_k)$ for $k > i$. Therefore, the cases generated by this algorithm satisfy the sufficient condition of *CLPP-TH2*.

Figure 5.2 illustrates a case in $\mathbb{R}^2$ which is generated by the Random Case Generator of *CLPP-TH2*.

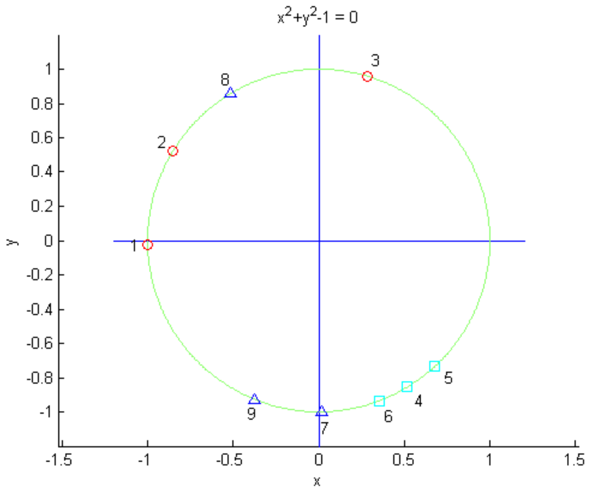

Figure 5.2: A case generated by the Random Case Generator of *CLPP-TH2*.

5.3 Random Case Generator of *CLPP-TH3*

The sufficient condition of *CLPP-TH3* is for any $i \neq j$, $(\mathbf{S}_i \cup \mathbf{S}_j) \cap H^+(\mathbf{T}_i) \neq \varnothing$ for any $\widehat{i}$-transversal $\mathbf{T}_i$. Here we will not introduce the general method to generate cases satisfying this sufficient condition. We will present a special generator which can generate cases satisfying the sub-condition: $\mathbf{S}_i \cap H^+(\mathbf{T}_i) \neq \varnothing$ for any $\widehat{i}$-transversal $\mathbf{T}_i$, but not satisfying the sufficient conditions of *CLPP-TH1* or *CLPP-TH2*. The main idea of the generator is that at first a regular colourful simplex containing origin $\mathbf{0}$ is generated in $\mathbb{R}^d$ as a reference, then, for every colour $\mathbf{S}_i$ $(i = 1, \ldots, d+1)$, $d+1$ points are generated and clustered around the ith vertex of the regular simplex. The algorithm takes a parameter

r to control the maximum allowed perturbation between a generated colourful point and its reference vertex. Note that while this algorithm cannot guarantee all the $(d+1)^{d+1}$ colourful simplices contain $\mathbf{0}$, it can assure that the number of colourful simplices containing $\mathbf{0}$ is very high.

Firstly we introduce the algorithm of generating a regular colourful simplex containing origin $\mathbf{0}$ in $\mathbb{R}^d$. The algorithm obtains an additional vertex for an additional dimension by updating a lower dimensional regular simplex vertex set to a higher dimensional one. It uses the fact that, if the edge length of an d-dimensional regular simplex is L, then its height is $L\sqrt{\frac{d+1}{2d}}$. After all vertices are generated in $\mathbb{R}^d$, they are scaled to the center point of the regular simplex and make the center point as the origin $\mathbf{0}$. The algorithm is shown as follows:

Algorithm 8 : Regular-Simplex-Generator

```
   Input: $d$
   Output: $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{d+1}$
1  begin
2      initialize $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{d+1}]$ to be zeros
3      $\mathbf{P}(1,1) \leftarrow -1$
4      $\mathbf{P}(1,2) \leftarrow 1$
5      $L \leftarrow 2$
6      for $i \leftarrow 2$ to $d$ do
7          $H \leftarrow L\sqrt{\frac{i+1}{2i}}$
8          $\mathbf{P}(i, i+1) \leftarrow H$
9          for $j \leftarrow i+2$ to $d+1$ do
10             $\mathbf{P}(i,j) \leftarrow H/(i+1)$
11     $\mathbf{c} \leftarrow (\sum_{i=1}^{d+1} \mathbf{p}_i)/(d+1)$
12     for $i \leftarrow 1$ to $d+1$ do
13         $\mathbf{p}_i \leftarrow \mathbf{p}_i - \mathbf{c}$
14         $\mathbf{p}_i \leftarrow \mathbf{p}_i/\|\mathbf{p}_i\|$
15 end
```

In Regular-Simplex-Generator, step 2-10 generate a regular simplex from the points $\mathbf{p}_1$ and $\mathbf{p}_2$ with the edge length being 2. After an d-dimensional regular simplex is generated, the center point of the simplex is calculated by step 11. Step 12-14 scale all the vertices of the simplex to the center point and normalize them onto the unit sphere $\mathbb{S}^{d-1}$.

In order to generate the cases satisfying the sub-condition of *CLPP-TH3*: $\mathbf{S}_i \cap H^+(\mathbf{T}_i) \neq \varnothing$ for any $\widehat{\imath}$-transversal $\mathbf{T}_i$, Algorithm 9 is introduced in which all the points of a colour are clustered around a reference vertex of a regular simplex. It is shown as follows:

Algorithm 9 : Random-Generator-*CLPP-TH3*

```
    Input: $d$, $r$
    Output: $\mathbf{S} = \bigcup_{i=1}^{d+1} \mathbf{S}_i$
1   begin
2       generate a regular simplex in $\mathbb{R}^d$ with $\mathbf{p}_1, \ldots, \mathbf{p}_{d+1}$ being vertices
3       for $i \leftarrow 1$ to $d+1$ do
4           for $j \leftarrow 1$ to $d+1$ do
5               $\mathbf{t} \sim \mathbb{R}^d$
6               $\mathbf{t} \leftarrow \mathbf{p}_i + r\mathbf{t}/\|\mathbf{t}\|$
7               $\mathbf{s}_j^i \leftarrow \mathbf{t}/\|\mathbf{t}\|$
8   end
```

In Algorithm 9, step 2 calls the Regular-Simplex-Generator to generate a colourful regular simplex with $\mathbf{p}_1, \ldots, \mathbf{p}_{d+1}$ being vertices. After that, for every point of colour $\mathbf{S}_i$ for $i = 1, \ldots, d+1$, the algorithm randomly selects a point $\mathbf{t}$ in $\mathbb{R}^d$ in step 5. We employ the MATLAB function `rand`$(d, 1)$ to generate a random point $\mathbf{t}$ in $\mathbb{R}^d$. Then the point $\mathbf{t}$ is normalized and multiplied with the perturbation parameter r whose maximum value is $1/d$ as we specified. In step 6-7, the colourful vertex $\mathbf{p}_i$ of the regular simplex is added by the perturbation value $r\mathbf{t}$, and this perturbed point becomes a point of colour $\mathbf{S}_i$ and is normalized onto the unit sphere $\mathbb{S}^{d-1}$. This perturbation process can assure that all the points of a colour are clustered around a reference vertex of a regular simplex within a small perturbation range.

Figure 5.3 illustrates a case in $\mathbb{R}^2$ which is generated by the Random Case Generator of *CLPP-TH3*.

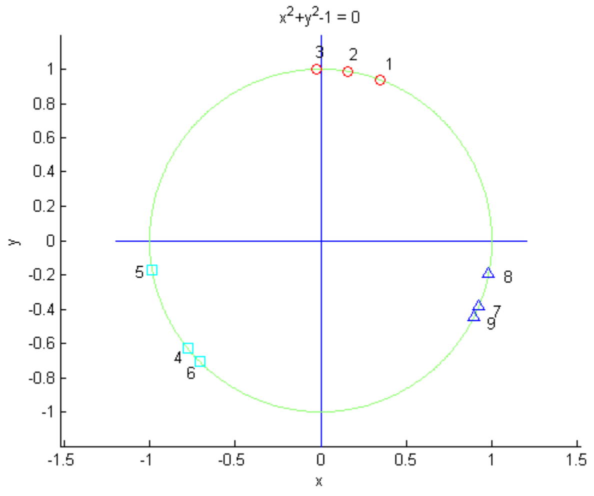

Figure 5.3: A case generated by the Random Case Generator of *CLPP-TH3*.

Chapter 6

Testing and Results

In this chapter, we will carry on the benchmark testing where the four *CLPP* solvers proposed in Chapter 3 will be tested against the random cases generated by the three generators presented in Chapter 5.

We use following abbreviations to denote the four solvers in Chapter 3.

- BO1: Solver-Bárány-Onn-1,
- BO2: Solver-Bárány-Onn-2,
- MD: Solver-Meunier-Deza,
- RP: Solver-Random-Pick.

And we use following notations to represent the three random case generators in Chapter 5:

- TH1: Random-Generator-*CLPP-TH1*,
- TH2: Random-Generator-*CLPP-TH2*,
- TH3: Random-Generator-*CLPP-TH3*.

From the discussions in previous chapters, we know that BO1, BO2 can theoretically solve all the cases generated by TH1 but cannot solve all the cases by TH2 and TH3; whereas MD and RP can solve all the problems generated by TH1, TH2 and TH3. We summarize this condition in Table 6.1.

	BO1	BO2	MD	RP
TH1	✓	✓	✓	✓
TH2	×	×	✓	✓
TH3	×	×	✓	✓

Table 6.1: The problems solved by different solvers.

In this chapter, BO1, BO2, MD and RP against TH1 will be tested in Section 6.1; MD and RP against TH2 will be tested in Section 6.2; and Section 6.3 will test MD and RP against TH3.

Even though there is no warranty that BO1 and BO2 can solve all the cases generated by TH2 and TH3, in practice they might work well on TH2 and TH3. Therefore, in Section 6.2 and Section 6.3, we will also investigate how well BO1 and BO2 work with TH2 and TH3, and test their performance against TH2 and TH3.

In order to compare the solvers' average performance, in each section the solvers are tested with the same set of generated cases. The testing is performed by following process:

1. Select a random case generator from TH1, TH2 and TH3, the dimension d, and number of cases;

2. Randomly generate a case in $\mathbb{R}^d$ by the selected generator and solve it by the solvers respectively;

3. For each solver, if a valid solution is obtained, record to a MAT file the iteration number and the running time used to find the solution;

4. Repeat step 2 until the number of cases is reached;

5. For each solver, read the MAT file to sum the iteration number and the running time of all validly solved cases respectively, and then divide them by the number of the validly solved cases to get the average number of iterations and the average running time.

The dimension d of the testing cases is selected by $d = 3 \times 2^n$ for n=0, 1, 2, 3 and so on. This selection will generate reasonable random cases of low, intermediate and high dimensional colourful linear programming problems. 1,000 cases will be produced for the dimension $d \leq 24$, and 100 cases for each dimension after that. The notation "N/A" in the tables of test results indicates that the testing cannot be done by the specified solver due to the running time being too long. The symbol "*" after a testing value in the testing table indicates that the specified solver cannot find valid solutions for some generated cases; and below the symbol is the ratio of the cases with valid solutions to the total generated cases.

The server used to perform the testing is a machine with Intel Core2 Duo CPU E4500 at 2.20GHz each and random memory is 2GB. The operating system is Windows Vista and the MATLAB version is 7.0.019920 (R14) with Optimization Toolbox. All the algorithms proposed in this thesis are implemented by MATLAB language and executed in MATLAB Development Environment.

6.1 Testing against the Cases of *CLPP-TH1*

In this section, the solvers BO1, BO2, MD, RP are tested against the cases of *CLPP-TH1* with dimension d=3, 6, 12, 24, 48, 96. The test results are presented in Table 6.2 and Table 6.3.

d	Average (Maximum) Number of Iterations			
	BO1	BO2	MD	RP
3	2.324 (5)	3.228 (28)	2.487 (7)	7.921 (60)
6	3.575 (8)	5.919 (14)	4.927 (14)	66.701 (399)
12	5.857 (10)	12.168 (20)	12.299 (41)	3843.67 (25468)
24	9.893 (17)	25.958 (38)	34.694 (120)	N/A
48	17.29 (23)	53.74 (68)	145.9 (464)	N/A
96	29.97 (36)	107.87 (122)	11598.89 (81700)	N/A

Table 6.2: Test results of iteration against the cases of *CLPP-TH1*.

d	Average (Maximum) Running Time (seconds)			
	BO1	BO2	MD	RP
3	0.007428 (3.531)	0.000936 (0.094)	0.000502 (0.046)	0.000675 (0.016)
6	0.009089 (0.031)	0.001868 (0.016)	0.001485 (0.016)	0.009176 (0.063)
12	0.028805 (0.062)	0.005762 (0.016)	0.006661 (0.032)	0.88856 (5.859)
24	0.13266 (0.25)	0.023398 (0.047)	0.041438 (0.141)	N/A
48	0.92339 (1.266)	0.10497 (0.125)	0.62441 (2.015)	N/A
96	11.6281 (14.546)	0.5877 (0.687)	315.4681 (2120.469)	N/A

Table 6.3: Test results of running time against the cases of *CLPP-TH1*.

From the test results, we know that:

- Before $d \leq 48$, there is no significant difference in the number of iterations among BO1, BO2 and MD; MD is faster than BO1 but a little bit slower than BO2.

- After $d > 48$, the number of iterations and running time taken by MD increase rapidly; but the average running time of BO1 and BO2 is still acceptable and BO2 is much quicker than BO1.

- RP is a very efficient solver when the dimension is very low, such as $d \leq 12$, but after that the solver is unacceptable to solve problems.

6.2 Testing against the Cases of ***CLPP-TH2***

In this section, the solvers BO1, BO2, MD and RP are tested against the cases of *CLPP-TH2* with dimension d=3, 6, 12, 24, 48, 60, 96. The test results are presented in Table 6.4 and Table 6.5.

d	Average (Maximum) Number of Iterations			
	BO1	BO2	MD	RP
3	2.6644 (5)* (867/1000)	3.5069 (33)* (795/1000)	3.435 (28)	7.763 (47)
6	5.0841 (8)* (963/1000)	7.2165 (16)* (924/1000)	8.917 (135)	57.181 (486)
12	10.1882 (14)* (999/1000)	14.5568 (24)* (995/1000)	24.792 (57)	3703.55 (23450)
24	20.303 (26)	28.989 (43)	111.002 (291)	N/A
48	40.3 (46)	56.48 (68)	1620.51 (4942)	N/A
60	50.23 (57)	69.82 (82)	12808.9 (117891)	N/A
96	79.59 (87)	112.92 (132)	N/A	N/A

Table 6.4: Test results of iteration against the cases of *CLPP-TH2*.

From the test results, we know that:

- Before $d \leq 48$, the performance of MD is pretty good; but after $d > 48$, the number of iterations and running time increase dramatically and become unacceptable after $d > 60$.

d	Average (Maximum) Running Time (seconds)			
	BO1	BO2	MD	RP
3	0.0040842 (0.016)* (867/1000)	0.00076352 (0.062)* (795/1000)	0.000688 (0.047)	0.000576 (0.047)
6	0.013238 (0.032)* (963/1000)	0.0019903 (0.016)* (924/1000)	0.002033 (0.016)	0.007431 (0.062)
12	0.055647 (0.094)* (999/1000)	0.0068111 (0.016)* (995/1000)	0.013084 (0.047)	0.77833 (4.891)
24	0.323 (0.515)	0.024326 (0.047)	0.15126 (0.422)	N/A
48	2.5234 (2.859)	0.10609 (0.141)	8.0753 (23.813)	N/A
60	5.1099 (5.969)	0.16932 (0.203)	110.7931 (1048.047)	N/A
96	33.0814 (36.453)	0.57654 (0.671)	N/A	N/A

Table 6.5: Test results of running time against the cases of *CLPP-TH2*.

- Before $d < 24$, BO1 and BO2 cannot solve all the cases, but as the dimension increases, they work better than before. And BO1 can solve more cases than BO2.

- After $d \geq 24$, BO1 and BO2 can solve all the cases and their performance is much better than that of MD after $d \geq 48$. Especially in higher dimension, BO2 is very fast. But we still cannot guarantee that BO1 and BO2 can solve any case of *CLPP-TH2* in high dimension.

- RP is a very efficient solver for solving the cases of *CLPP-TH2* when the dimension is very low, such as $d \leq 12$, but after that the solver become unacceptable to solve problems.

6.3 Testing against the Cases of *CLPP-TH3*

In this section, the solvers BO1, BO2, MD and RP are tested against the cases of *CLPP-TH3* with dimension d=3, 6, 12, 24, 48, 96, 192. The test results are presented in Table 6.6 and Table 6.7.

d	Average (Maximum) Number of Iterations			
	BO1	BO2	MD	RP
3	1 (1)	1 (1)	1 (1)	1 (1)
6	1 (1)	1 (1)	1 (1)	1 (1)
12	1 (1)	1 (1)	1 (1)	1 (1)
24	1 (1)	1 (1)	1 (1)	1 (1)
48	1 (1)	1 (1)	1 (1)	1 (1)
96	1 (1)	1 (1)	1 (1)	1 (1)
192	1 (1)	1 (1)	1 (1)	1 (1)

Table 6.6: Test results of iteration against the cases of *CLPP-TH3*.

d	Average (Maximum) Running Time (seconds)			
	BO1	BO2	MD	RP
3	0.000749 (0.004)	0.000295 (0.002)	0.000545 (0.002)	0.000115 (0.002)
6	0.000631 (0.003)	0.000466 (0.003)	0.000578 (0.004)	0.000188 (0.003)
12	0.001069 (0.014)	0.000877 (0.016)	0.001022 (0.01)	0.000253 (0.005)
24	0.002726 (0.029)	0.002692 (0.008)	0.002877 (0.007)	0.000333 (0.006)
48	0.01312 (0.023)	0.01288 (0.017)	0.01363 (0.018)	0.00075 (0.008)
96	0.07049 (0.113)	0.07094 (0.117)	0.07401 (0.096)	0.00184 (0.004)
192	0.46217 (0.567)	0.46094 (0.603)	0.47604 (0.621)	0.0073 (0.01)

Table 6.7: Test results of running time against the cases of *CLPP-TH3*.

From the test results, we know that:

- All the cases can be solved very fast in one iteration even in very high dimension, because the cases are special ones of CLPP-TH3 and the density of colourful simplices containing $\mathbf{0}$ is very high.

- The average running time of RP is much faster than that of other solvers because RP does not need to perform some extra processes in each iteration; meanwhile, the running time of BO1, BO2 and MD has no big difference.

Chapter 7

Conclusions and Future Work

In this thesis four algorithms, BO1, BO2, MD and RP, are presented and implemented to solve the colourful linear programming problems (*CLPP*) satisfying three different kinds of sufficient conditions: *CLPP-TH1*, *CLPP-TH2*, *CLPP-TH3*. We mainly focus on the software design and implementation for the Meunier-Deza algorithm because it is a more general algorithm proposed recently and can solve all the three kinds of *CLPP*. For the sake of comparing the algorithms' performance, three random case generators are designed and implemented to meet the three sufficient conditions respectively. Finally the four proposed algorithms are tested and benchmarked against the three random case generators and testing results are given.

From the testing results, we can conclude that MD has very good performance to solve all the three kinds of CLPP before $d \leq 48$; it is almost as fast as BO2 to solve CLPP-TH1 and very quick to solve CLPP-TH2 and CLPP-TH3. But after $d > 48$ the running time of MD to solve *CLPP* will increase rapidly because it gets into the subroutine Find-Closer-Facet to execute an enumeration operation which is a very time-expensive task.

The algorithms BO1 and BO2 can solve all the cases of *CLPP-TH1*, and BO2 is the most efficient algorithm for this kind of problem while BO1 is not recommended because it involves the time-expensive computation of minimum norm. There is no warranty that BO1 and BO2 can solve all the cases of *CLPP-TH2* and *CLPP-TH3*; but in practice they can solve most of the cases of *CLPP-TH2* when $d < 24$, and after $d \geq 24$ all the cases of CLPP-TH2 can be solved and BO2 has very good performance.

Theoretically algorithm RP can solve all the problems of *CLPP*, and actually it is very efficient when the problems are in low dimension such as $d \leq 12$ or in special condition such as the problems generated by TH3. But generally after $d > 12$ the performance of RP is unacceptable.

Since the Meunier-Deza algorithm is the only one that can solve all the three kinds of *CLPP*, it can be regarded as a reliable and time-effective algorithm to solve *CLPP-TH1*, *CLPP-TH2* and *CLPP-TH3* in intermediate dimension.

For the future work, in order to improve the performance of Solver-Meunier-Deza and make it competent to solve *CLPP* in high dimension, an efficient method needs to be found to replace the enumeration operation in the subroutine Find-Closer-Facet.

Bibliography

[1] Imre Bárány: *A generalization of Carathéodory's theorem*, Discrete Mathematics 40 (1982), 141-152.

[2] Imre Bárány, Shmuel Onn: *Carathéodory's theorem, colourful and applicable*, Intuitive Geometry (Budapest, 1995), Bolyai Society Mathematical Studies 6 (1997), 11-12; János Bolyai Mathematical Society, Budapest, 1997, pp.11-21.

[3] Imre Bárány, Shmuel Onn: *Colourful linear programming and its relatives*, Mathematics of Operations Research 22(3) (1997), 550-567.

[4] Antoine Deza, Sui Huang, Tamon Stephen, Tamás Terlaky: *The colourful feasibility problem*, Discrete Applied Mathematics 156 (2008), 2166-2177.

[5] Sui Huang: *Colourful feasibility: algorithms, bounds and implications*, Master thesis, McMaster University, 2007.

[6] Andreas F. Holmsen, János Pach, Helge Tverberg: *Points surrounding the origin*, Combinatorica 28(6) (2008), 633-644.

[7] Jorge L. Arocha, Imre Bárány, Javier Bracho, Ruy Fabila, Luis Montejano: *Very colourful theorems*, Discrete and Computational Geometry 42 (2009), 142-154.

[8] Frédéric Meunier, Antoine Deza: *A further generalization of the colourful Carathéodory theorem*, (2011).

[9] Antoine Deza, Sui Huang, Tamon Stephen, Tamás Terlaky: *Colourful simplicial depth*, Discrete and Computational Geometry 35 (2006), 597-604.

[10] Imre Bárány, Jiří Matoušek: *Quadratically many colourful simplices*, SIAM Journal on Discrete Mathematics 21(1) (2007), 191-198.

[11] Tamon Stephen, Hugh Thomas: *A quadratic lower bound for colourful simplicial depth*, Journal on Combinatorial Optimization 16(4) (2008), 324-327.

[12] Antoine Deza, Tamon Stephen, and Feng Xie: *More colourful simplices*, Discrete and Computational Geometry 45 (2011), 272-278.

[13] Grant Custard, Antoine Deza, Tamon Stephen, Feng Xie: *Small octahedral systems*, Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG'11), 2011.

[14] Jiří Matoušek: *Lectures on discrete geometry*, Graduate Texts in Mathematics 212, Springer-Verlag, New York, 2002.

[15] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: *Computational geometry: algorithms and applications*, 3rd edition, Springer-Verlag, Berlin, 2008.

[16] Joseph O'Rourke: *Computational geometry in C*, 2nd edition, Cambridge University Press, 1998.

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Introduction to algorithms*, 2nd edition, The MIT Press, 2001.

[18] Katta G. Murty, Vincent F. Yu: *Linear complementarity, linear and non-linear programming*, internet edition.

[19] Mervin E. Muller: *A note on a method for generating points uniformly n-dimensional spheres*, Communications of the Association for Computing Machinery 2 (1959), 19-20.

[20] Tim Davis: *MATLAB primer*, 8th edition, CRC Press, 2011

[21] Sui Huang, Hanxing Zhang: *MATLAB code for colourful feasibility problem*, available at: http://www.cas.mcmaster.ca/˜deza/clp.html.