COLOURFUL FEASIBILITY:

ALGORITHMS, BOUNDS AND IMPLICATION

COLOURFUL FEASIBILITY:

ALGORITHMS, BOUNDS AND IMPLICATIONS

By

Sui Huang, B.Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2007) McMaster University
(Computing and Software) Hamilton, Ontario

TITLE: Colourful Feasibility:
Algorithms, Bounds, and Implications

AUTHOR: Sui Huang, B.Eng. (McMaster University)

SUPERVISORS: Dr. Antoine Deza and Dr. Tamás Terlaky

NUMBER OF PAGES: xx, 128.

Abstract

Given a point $\mathbf{p}$ and $d+1$ sets (i.e., colours) of points in dimension d, the *Colourful Feasibility Problem* is to decide whether there are $d+1$ points of different colours containing $\mathbf{p}$ in their convex hull; and if yes, find such a point set. The monochrome version of this problem, expressing $\mathbf{p}$ as a linear combination of $d+1$ points in a set S, can be solved using traditional linear optimization algorithms. The Colourful Feasibility Problem was presented by Bárány and Onn in 1997, and it is still not known if a polynomial-time algorithm exists. The case where we have d colours in dimension d and no restriction on the size of the sets has been shown to be strongly NP-complete through a reduction of 3-SAT. We define the *core* of a configuration to be the intersection of the convex hulls of each colour. We start from the important subcase that we call *Colourful Core Feasibility Problem* where we have $d+1$ points of each colour, and $\mathbf{p}$ in the core. By Bárány's 1982 Colourful Carathéodory Theorem, a solution is guaranteed to exist, and the problem is to exhibit one. This problem is described by Bárány and Onn as "an outstanding problem on the border line between tractable and intractable problems". Besides applications to combinatorics, The Colourful Feasibility Problem models a situation where we want to select a set of points that is both diverse and representative.

While we have not found out whether the Colourful Core Feasibility Problem can be solved in polynomial time, our contributions are on both the theoretical and practical performance of algorithms to solve the Colourful Feasibility Problem. The algorithms proposed by Bárány and Onn are essentially geomet-

ric, and the complexity guarantees depend crucially on having $\mathbf{p}$ inside the core. We consider modifications of these algorithms which update multiple colours at each stage, as well a greedy heuristic where we choose the adjacent simplex of maximum volume in each iteration and a random sampling approach. Our test suite includes unstructured random problems, ill-conditioned problems, problems with a restricted number of solutions and infeasible problems. We conclude that the most robust and nearly fastest algorithm for the Colourful Core Feasibility Problem is the multi-update variant which yields substantial gains over the original ones. Alternative approaches based on nondefinite quadratic optimization problem and positive semidefinite relaxation, and a combinatorial algorithm not depending on having $\mathbf{p}$ in the core are also introduced. Finally, we give the first upper bound for the minimal number of colourful simplices containing a core point and the first improvement of the lower bound since Bárány's result in 1982.

Acknowledgments

The thesis was written under the guidance and with the help of my supervisors, Dr. Antoine Deza and Dr. Tamás Terlaky. They encouraged me to pursue graduate studies, and their valuable advices and extended knowledge were a constant help. I wish to thank the members of the examination committee: Dr. Antoine Deza, Dr. Franya Franek, Dr. Nedialko S. Nedialkov (Chair) and Dr. Tamás Terlaky.

I sincerely thank Dr. Tamon Stephen, Simon Fraser University, who guided me on this research project while he was a postdoctoral fellow at McMaster University, and kept on giving me suggestions and encouragements since then. His topological and geometric ideas were a constant source of inspiration and provided the key ingredients for proving the bounds presented in the thesis.

I wish to acknowledge the help of Hanxing Zhang who implemented and tested several algorithms. Thanks to all the members of the Advanced Optimization Laboratory for their support. Special thanks to Imre Pólik for his generous help on many algorithmic and theoretical questions and, in particular, for his insights of the optimization formulation presented in the thesis.

Finally, I wish to thank my family for their continuous encouragement and support.

Contents

Abstract **iii**

Acknowledgments **v**

List of Figures **xii**

List of Tables **xiii**

List of Definitions **xiv**

List of Symbols **xv**

List of Abbreviations **xix**

1 Preliminaries **1**

1.1 Vectors and Points 1

1.2 Norms and Distances 1

1.3 Balls and Spheres 2

1.4 Projections 2

1.5 Convex Hulls, Affine Hulls, Affine Subspaces and Subspaces . . 2

1.6 Line Segments 3

1.7 Cones . . . 3
1.8 Dimension of a Point Set . . . 4
1.9 Affine Hyperplanes, Hyperplanes and Half Spaces . . . 4
1.10 General Position and Degeneracy . . . 4
1.11 Convex Polytopes, Faces, Facets and Vertice . . . 5
1.12 Simplex and Regular Simplex . . . 5
1.13 Set Partitioning . . . 6
1.14 Optimization . . . 6
1.15 Real Number Arithmetic . . . 7

2 Introduction **9**
2.1 Carathéodory Theorem and the Linear Feasibility Problem . . . 9
2.2 Colourful Feasibility and Colourful Core Feasibility Problems . . . 11

3 Bárány-Onn Algorithms **15**
3.1 First Bárány-Onn algorithm . . . 16
3.2 Second Bárány-Onn algorithm . . . 22
3.3 Oscillations for the Second Bárány-Onn Algorithm . . . 28
3.4 Oscillation Detection . . . 31
3.5 Implementation . . . 32
3.5.1 Input Data . . . 32
3.5.2 Numerical Tolerance . . . 33
3.5.3 Testing whether a Simplex Covers a Point . . . 34
3.5.4 Finding the Minimum Norm Point in a Simplex . . . 35
3.5.5 Selecting a Vertex to be Replaced . . . 35

4 Multi-Update Algorithm for Colourful Core Feasibility Problem **37**

4.1 Multi-Update Algorithm 37

4.2 Implementation 40

4.2.1 Accumulation of Numerical Errors 40

4.2.2 Improving Numerical Sum by Sorting 41

5 Two Alternative Algorithms **45**

5.1 Volume of a Simplex 45

5.2 Max-Volume Algorithm 46

5.3 Random-Picking Algorithm 50

6 Enumeration with geometric Heuristic for General Cases **51**

6.1 Enumeration Framework 51

6.2 Selecting Colourful Points by Stretching 55

6.3 Implementation 59

6.3.1 Colourful Sets Represented by Indexing 59

6.3.2 Enumerating Transverses of a Given Set 60

6.3.3 Enumerating Transverses in $\tau(\mathbf{W} \cup \mathbf{V}) \setminus (\tau(\mathbf{W}) \cup \tau(\mathbf{V}))$ 61

7 Optimization Approach **63**

7.1 Nonconvex Quadratic Optimization Formulation 63

7.2 Positive Semidefinite Relaxation 66

7.2.1 Semidefinite Optimization 66

7.2.2 Relaxation 67

7.3 Toolboxes Used for Implementation 70

8 Random Case generation **75**

8.1 Notations 75

8.1.1 Uniform Random Selection 75

8.1.2 Universal Set of Colour Indices 76

8.1.3 Notation for Colourful Points 76

8.2 Colourful Core Feasibility Problem Generators 76

8.2.1 Unstructured Random Cases 76

8.2.2 Ill-conditioned Cases 77

8.2.3 Restricted Number of Solutions 78

Cases Having $d^{d+1} + 1$ Solutions 79

Cases Having $(d + 1)!$ Solutions 82

Cases Having Few Solutions 83

8.3 General Cases 85

8.3.1 Unstructured Cases 86

8.3.2 Infeasible Cases 86

9 Test Results **89**

9.1 Colourful Core Feasibility Problems 90

9.1.1 Unstructured Cases 91

9.1.2 Ill-conditioned Cases 93

Balanced Tube Cases 93

Unbalanced Tube Cases 95

9.1.3 Cases having Restricted Number of Solutions 97

Cases Having Few Solutions 97

Cases Having $(d + 1)!$ and $d^{d+1} + 1$ Solutions 99

9.2 General Cases 102

9.2.1 Feasible Cases 102
9.2.2 Infeasible Cases 102

10 Colourful Simplicial Depth **107**
10.1 Simplicial Depth and Colourful Simplicial Depth 107
10.2 Observing Simplicial Depth by Cones 108
10.3 Parity Property 111
10.4 Colourful Simplicial Depth Lower Bound 112
10.4.1 Low Depth Configurations 112
10.4.2 Application of Colourful Simplicial Depth 118

11 Conclusions and Future Work **121**
11.1 Conclusion 121
11.2 Future Work 123

Bibliography **125**

List of Figures

3.1 The points $\mathbf{t}_i$, $\mathbf{p}$, $\mathbf{w}$, $\mathbf{v}$ and $\mathbf{0}$ in a 2D subspace 18
3.2 A sample iteration of Solver-Bárány-Onn-1 20
3.3 A sample iteration of Solver-Bárány-Onn-2 24
3.4 The 3D pattern of an oscillating **CCFP** case 30

4.1 A sample iteration for Solver-Multi-Update 44

7.1 Sample *MATLAB* code that uses *YALMIP* to interface *SeDuMi* 71
7.2 The data flow diagram of the Solver-SDP-Relax implementation 73

8.1 3D example of **CCFP** with 10 solutions 85

10.1 Points placement in dimension 3 for constructing $\mathbf{S}^-$ 113

List of Tables

9.1 Test results on unstructured **CCFP** cases. 92
9.2 Test results on balanced tube **CCFP** cases. 94
9.3 Test results on unbalanced tube **CCFP** cases. 96
9.4 Test results on **CCFP** cases with few solutions. 98
9.5 Test results on **CCFP** cases with $(d+1)!$ solutions. 100
9.6 Test results on **CCFP** cases with $d^{d+1}+1$ solutions. 101
9.7 Test results on general feasible **CFP** cases. 104
9.8 Test results on general infeasible **CFP** cases. 105

List of Definitions

1.14.1 Optimization Problem Formulation 6
2.1.1 Linear Feasibility Problem 9
2.2.1 Colourful Feasibility Problem 12
2.2.2 Colourful Core Feasibility Problem 13
7.1.1 Quadratic Formulation of **CFP** 63
7.2.1 Semidefinite Optimization Problem 66
7.2.2 Alternative Optimization Formulation of **CFP** 67
7.2.3 Semidefinite Relaxation of **QP** 67

List of Symbols

$\|\cdot\|_k$: k-norm

$\|\cdot\|$: 2-norm

$\sim$: uniform random selection

$\sum$: summation

$\leftarrow$: assignment operation

$\subset$: proper subset

$\subseteq$: subset

$\setminus$: set minus

$\biguplus$: set partition

$\bigcup$: set union

$\bigcap$: set intersection

$\in$: set belongs

$\notin$: set not belongs

$\emptyset$**:** empty set

$\prod$**:** product

$\bullet$**:** inner product

$|\cdot|$**:** absolute value, or size of set

$\succeq$**:** $A \succeq 0$ means the matrix A is positive semidefinite

$\ll$**:** much less than

$[\mathbf{u}, \mathbf{v}]$**:** the closed line segment between $\mathbf{u}$ and $\mathbf{v}$

$(\mathbf{u}, \mathbf{v})$**:** the open line segment between $\mathbf{u}$ and $\mathbf{v}$

$(\mathbf{u}, \mathbf{v}]$ **and** $[\mathbf{u}, \mathbf{v})$**:** the line segments between $\mathbf{u}$ and $\mathbf{v}$ with one end closed and one end open

ϵ **and** ε**:** small positive numbers usually used as precision

$\tau(\mathbf{V})$**:** the set of all transverses generated by $\mathbf{V}$

$\mu(d)$**:** minimum colourful simplicial depth

α **and** β**:** scalers between 0 and 1 used for scaling

ρ**:** maximum radius of a ball inside the core of a Colourful Core Feasibility Problem

$\mathbf{0}$**:** origin or zero vector

$\mathbb{B}(r, \mathbf{p})$**:** ball of radius r and centered at $\mathbf{p}$

$\mathbb{B}^d$**:** ball of radius 1 and centered at the origin in d-dimensional space

$\mathbb{S}(r, \mathbf{p})$**:** sphere of radius r and centered at $\mathbf{p}$

$\mathbb{S}^d$**:** sphere of radius 1 and centered at the origin in d-dimensional space

$\text{aff}(\cdot)$**:** affine hull

$\underset{x \in D}{\text{argmin}}(\text{f}(x))$**:** returns one x that minimizes $\text{f}(x)$

$\text{cone}(\cdot)$**:** cone

$\text{conv}(\cdot)$**:** convex hull

d**:** dimension of the Euclidean space

$\det(\cdot)$**:** determinant

$\text{depth}_{\mathbf{p}}(S)$**:** simplicial depth of $\mathbf{p}$ in the point set S

$\mathbf{depth}_{\mathbf{p}}(\mathbf{S})$**:** colourful simplicial depth of $\mathbf{p}$ in the colourful point set $\mathbf{S}$

$\dim(\cdot)$**:** dimension

$\text{dist}(\cdot)$**:** distance

$\text{g}(S)$**:** point of maximum simplicial depth in general position given a point set S

H**:** affine hyperplane

H^+ **and** H^-**:** half spaces defined by affine hyperplane H

H_i**:** affine hyperplane indexed by i

I **and** J**:** universal set of indices

I^0**,** I^+**,** I^- **and** $\mathcal{I}$**:** index sets

log**:** logarithm with base 2

min**:** minimization

$\mathbf{t}$**:** vector

$\mathbf{t}^T$**:** transpose of $\mathbf{t}$

$O(\cdot)$**:** big-O notation

$\underset{P}{\text{proj}}(\mathbf{t})$**:** the projection of $\mathbf{t}$ on P

Q**:** square matrix of the objective function of a quadratic optimization problem

Q_{k_1,k_2}**:** the element at the k_1th row and the k_2th column of the matrix Q

$\mathbf{S}$**:** a colourful set representing a **CFP**

S_i**:** the ith colour subset of $\mathbf{S}$

S**:** a set of monocolour points

t_i**:** ith elements of $\mathbf{t}$

T**:** a set of $d+1$ points in $\mathbb{R}^d$ with different colours

$T_{i,\mathbf{s}}$**:** the colourful set of points obtained by replacing the $\mathbf{t}_i$ in T with $\mathbf{s}$

vol$(\cdot)$**:** volume

$z_S(\mathbf{s})$**:** the number of zero-containing $\mathbf{s}$-simplex for given S

List of Abbreviations

CFP: Colourful Feasibility Problem

CCFP: Colourful Core Feasibility Problem

QP: quadratic formulation of the Colourful Feasibility Problem

SDP: semidefinition relaxation of the quadratic formulation of the Colourful Feasibility Problem

Chapter 1

Preliminaries

In this chapter we introduce some fundamental concepts and notations in geometry and optimization used throughout the thesis.

1.1 Vectors and Points

We use the same symbol to denote a point in the d-dimensional Euclidean space and the column vector representing that point, i.e., a point $\mathbf{t}$ in $\mathbb{R}^d$ can also be interpreted as the column vector representing $\mathbf{t}$. We use **bold** font and lower case characters to denote points and vectors. We use $\mathbf{0}$ to denote both the origin of the Euclidean space and the zero-vector. The ith coordinate of the vector $\mathbf{t}$ is denoted by t_i.

1.2 Norms and Distances

For $k \in \mathbb{N}$ and $\mathbf{t} \in \mathbb{R}^d$, the *k-norm* of $\mathbf{t}$ is $\|\mathbf{t}\|_k = \sqrt[k]{|t_1|^k + \cdots + |t_d|^k}$. We often use the Euclidean 2-norm and simply call it *norm*, and denote it by $\|\mathbf{t}\|$. The *distance* between two points $\mathbf{t}_1$ and $\mathbf{t}_2$ is the norm of their difference, namely, $\|\mathbf{t}_1 - \mathbf{t}_2\|$. The distance between two sets S_1 and S_2 of points is the minimum

value of $\|\mathbf{t}_1 - \mathbf{t}_2\|$ such that $\mathbf{t}_1 \in S_1$ and $\mathbf{t}_2 \in S_2$, and we denote it

$$\text{dist}(S_1, S_2) = \text{dist}(\mathbf{t}_1, S_2) = \text{dist}(S_1, \mathbf{t}_2). \tag{1.2.1}$$

1.3 Balls and Spheres

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

1.4 Projections

As in this thesis we only consider projections on polytopes or affine hyperplanes, the projection of a point is unique. More precisely, the *projection* $\underset{P}{\text{proj}}(\mathbf{t})$ of a point $\mathbf{t}$ on a point set P is the point $\mathbf{p} \in P$ that minimizes its distance to $\mathbf{t}$, i.e., $\underset{P}{\text{proj}}(\mathbf{t}) = \underset{\mathbf{p} \in P}{\text{argmin}} \|\mathbf{t} - \mathbf{p}\|$. The projection of a set A onto another set B is $\{\, \underset{B}{\text{proj}}(a) \,:\, a \in A \,\}$. For $\mathbf{t} \in P$, we obviously have $\underset{P}{\text{proj}}(\mathbf{t}) = \mathbf{t}$.

1.5 Convex Hulls, Affine Hulls, Affine Subspaces and Subspaces

For a set $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_k\} \subset I\!R^d$, the *convex hull* $\text{conv}(T) = \text{conv}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ is the set of their linear combinations with non-negative coefficients whose sum is 1; that is,

$$\text{conv}(\mathbf{t}_1, \ldots, \mathbf{t}_k) = \{\, \lambda_1 \mathbf{t}_1 + \cdots + \lambda_k \mathbf{t}_k \,:\, \lambda_1, \ldots, \lambda_k \geq 0,\, \lambda_1 + \cdots + \lambda_k = 1 \,\}.$$

For a set $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_k\}$, the *affine hull* $\text{aff}(T) = \text{aff}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ is the set of their linear combinations with coefficients whose sum is 1; that is,

$$\text{aff}(\mathbf{t}_1, \ldots, \mathbf{t}_k) = \{\, \lambda_1\mathbf{t}_1 + \cdots + \lambda_k\mathbf{t}_k \,:\, \lambda_1 + \cdots + \lambda_k = 1 \,\}.$$

In this thesis we only consider the convex hulls and affine hulls of a finite number of points. A finite set T of points *generates* its convex hull and affine hull, or we can say those points are the *generators* of their convex hull and affine hull.

A set $P \subseteq \mathbb{R}^d$ is an *affine subspace* if there exist a finite set $T \subset \mathbb{R}^d$ such that $P = \text{aff}(T)$. If T is a minimum set generating P and $|T| = k + 1$, for $0 \leq k \leq d$, then P is a k-dimensional affine subspace of $\mathbb{R}^d$. A *subspace* is an affine subspace that contains the origin.

1.6 Line Segments

A *line segment* is a continuous subset of a line. We can use two end points to represent a line segment. An end point of a line segment can be either open or closed. For two points $\mathbf{u}$ and $\mathbf{v}$, the *closed line segment* represented by them is $[\mathbf{u}, \mathbf{v}] = \text{conv}(\mathbf{u}, \mathbf{v})$, where both $\mathbf{u}$ and $\mathbf{v}$ are *closed end points*; the *open line segment* represented by them is $(\mathbf{u}, \mathbf{v}) = \text{conv}(\mathbf{u}, \mathbf{v}) \setminus \{\mathbf{u}, \mathbf{v}\}$, where both $\mathbf{u}$ and $\mathbf{v}$ are *open end points*. A line segment can also be open at one end but closed at the other end: $(\mathbf{u}, \mathbf{v}] = \text{conv}(\mathbf{u}, \mathbf{v}) \setminus \{\mathbf{u}\}$ and $[\mathbf{u}, \mathbf{v}) = \text{conv}(\mathbf{u}, \mathbf{v}) \setminus \{\mathbf{v}\}$.

1.7 Cones

A *cone* K is a set of points satisfying: if $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_n\} \subseteq K$ and $\lambda_1, \ldots, \lambda_n \geq 0$, then $\sum_{i=1}^{n} \lambda_i\mathbf{t}_i \in K$. The *polyhedral cone* generated by T is

$$\text{cone}(T) = \text{cone}(\mathbf{t}_1, \ldots, \mathbf{t}_n) = \{ \sum_{i=1}^{n} \lambda_i \mathbf{t}_i \, : \, \lambda_1, \ldots, \lambda_n \geq 0 \}.$$

In $\mathbb{R}^d$, we call a polyhedral cone generated by d points a *simplicial cone*.

1.8 Dimension of a Point Set

Let P be a set of points. The *dimension* $\dim(P)$ of P is k if:

- P is in a k-dimensional affine subspace; and
- P is not in a $(k-1)$-dimensional affine subspace, or $k = 0$ such that a $(k-1)$-dimensional subspace is undefined.

1.9 Affine Hyperplanes, Hyperplanes and Half Spaces

An *affine hyperplane* in $\mathbb{R}^d$ is a $(d-1)$-dimensional affine subspace of $\mathbb{R}^d$; a *hyperplane* is an affine hyperplane that contains $\mathbf{0}$. For each affine hyperplane H in $\mathbb{R}^d$, we can find $\mathbf{n} \in \mathbb{R}^d$ and $c \in \mathbb{R}$ such that $H = \{\mathbf{x} \, : \, \mathbf{n}^T\mathbf{x} = c\}$; for each hyperplane H we can find $\mathbf{n} \in \mathbb{R}^d$ such that $H = \{\mathbf{x} \, : \, \mathbf{n}^T\mathbf{x} = 0\}$. In both cases, $\mathbf{n}$ is called a *normal vector* of H.

An affine hyperplane $H = \{\mathbf{x} \, : \, \mathbf{n}^T\mathbf{x} = c\}$ in $\mathbb{R}^d$ separates $\mathbb{R}^d$ into two *half spaces* $\{\mathbf{x} \, : \, \mathbf{n}^T\mathbf{x} \geq c\}$ and $\{\mathbf{x} \, : \, \mathbf{n}^T\mathbf{x} \leq c\}$ (or equivalently, $\{\mathbf{x} \, : \, -\mathbf{n}^T\mathbf{x} \geq -c\}$), which are usually denoted by H^+ and H^- respectively. We can say that H is the *affine hyperplane* defining H^+ and H^-.

1.10 General Position and Degeneracy

A finite set $T \subset \mathbb{R}^d$ is in *general position* if for any $k < d$ there is no k-dimensional affine subspace that contains $k+2$ points from T. A finite set T

not in general position is *degenerate*.

1.11 Convex Polytopes, Faces, Facets and Vertice

A *convex polytope* is the convex hull of a finite set of points. In this thesis we omit the word *convex* and simply use the word *polytope*. A polytope is bounded. A polytope $P \subset \mathbb{R}^d$ is *full dimensional* if $\dim(P) = d$.

Let P be a polytope and H^* be a half space (where "*" is either "+" or "-"). If $P \subset H^*$, then we can say H^* is *valid* for P, or the affine hyperplane H defining H^* is *valid* for P. A *face* F of a polytope P is any non-empty set of the form $F = P \cap H \neq \emptyset$, where H is valid for P. A $(d-1)$-dimensional face of a polytope in $\mathbb{R}^d$ is called a *facet*, and a 0-dimensional face is called a *vetex*. If P is full dimensional, then the union of its facets form its boundary. Otherwise P does not have any facet.

A vertex of a polytope is a point. The minimum set of points that generates a polytope P is its set of vertices, so we can use the vertices to represent a polytope. On the other hand, for each polytope P, there exist a set of half spaces such that their intersection is P, so we can also use half spaces to represent a polytope.

1.12 Simplex and Regular Simplex

A *simplex* in $\mathbb{R}^d$ is the convex hull of $d+1$ points in $\mathbb{R}^d$. A full dimensional simplex is a *regular simplex* if and only if all the pairwise distances of its vertices are equal.

1.13 Set Partitioning

If a set $\mathbf{S}$ is the union of the sets $S_1, \ldots, S_k$ and $S_i \cap S_j = \emptyset$ for $1 \leq i \neq j \leq k$, then $S_1, \ldots, S_k$ is a partition of $\mathbf{S}$, and this relationship can be denoted by

$$\mathbf{S} = \biguplus_{i=1}^{k} S_i \tag{1.13.2}$$

using the $\biguplus$ sign.

1.14 Optimization

An *optimization problem* is to find an assignment for a set of variables minimizing a given objective function, while satisfying the given constraints.

Definition 1.14.1 (Optimization Problem Formulation)

$$\begin{array}{rl} \min & f(\mathbf{x}) \\ \text{such that}: & h(\mathbf{x}) \geq \mathbf{0}, \end{array} \tag{1.14.3}$$

where $\mathbf{x} = [x_1\, x_2\, \ldots\, x_n]^T$ *is the vector of variables,* $f : I\!R^n \rightarrow I\!R$ *is the objective function, and* $h : I\!R^n \rightarrow I\!R^m$ *gives the constraints. Further,* n *and* m *are assumed to be the number of scalar variables and the number of constraints, respectively.*

□

If there exist any $\mathbf{x}$ satisfying $h(\mathbf{x}) \geq \mathbf{0}$, then the optimization problem is *feasible*, otherwise the optimization problem is *infeasible*. The set $\{\, \mathbf{x} \,:\, h(\mathbf{x}) \geq \mathbf{0} \,\}$ is called the *feasible region*. An alternative formulation for the optimization problem is:

$$\min_{\mathbf{x}} \{\, f(\mathbf{x}) \,:\, h(\mathbf{x}) \geq \mathbf{0} \,\}. \tag{1.14.4}$$

In this thesis we consider *Linear Optimization Problem* which corresponds to linear objective functions and constraints, as well as *Quadratic Optimization Problem* and *Semidefinite Optimization Problem*, which involves quadratic functions and positive semidefinite matrices, see Chapter 7, respectively.

If D is the feasible region of an optimization problem, then $\underset{\mathbf{x}\in D}{\operatorname{argmin}}(f(\mathbf{x}))$ denotes the value of $\mathbf{x}$ that minimizes $f(\mathbf{x})$, namely, the optimal $\mathbf{x}$. In case of non-unique optimal $\mathbf{x}$, $\underset{\mathbf{x}\in D}{\operatorname{argmin}}(f(\mathbf{x}))$ denotes an arbitrary optimal value. The statement

$$\mathbf{x}^* \leftarrow \underset{\mathbf{x}\in D}{\operatorname{argmin}}(f(\mathbf{x})) \tag{1.14.5}$$

denotes the operation assigning an arbitrary optimal $\mathbf{x}$ value to $\mathbf{x}^*$.

1.15 Real Number Arithmetic

Bárány and Onn [7] analyzed algorithms for **CCFP** with both real arithmetic computation and rational data computation. However, we assume that real numbers are stored and computed exactly in the algorithm description and analysis presented in this thesis. Namely, we assume *real number arithmetic* instead of handling rational numbers. For example, from Section 3.1 to Section 3.3 we do not discuss rational number or floating number handling, except clarifying the relationship with the paper of Bárány and Onn [7]. We discuss the efficiency of algorithms in term of *arithmetic operations*. Each *arithmetic operation* is one elementary operation on a scalar or a pair of scalars, such as a negation, addition, substraction, multiplication, division, comparison, or square root operation.

On the other hand, as we use floating arithmetic to implement the algorithms, we discuss the handling of floating point numbers in the implementation

sections such as in Section 3.5.

Chapter 2

Introduction

This chapter introduces the *Colourful Feasibility Problem* (**CFP**) and the *Colourful Carathéodory Theorem*. These notions were proposed as generalizations of the Linear Feasibility Problem and the Carathéodory Theorem by Bárány and Onn [7] in 1997 and Bárány [3] in 1982, respectively.

2.1 Carathéodory Theorem and the Linear Feasibility Problem

Theorem 2.1.1 (Carathéodory Theorem) *Given a finite set of points $S \subset \mathbb{R}^d$ and a point $\mathbf{p} \in \text{conv}(S)$, then $\mathbf{p} \in \text{conv}(T)$ for some subset $T \subseteq S$ and $|T| \leq d+1$.*

Definition 2.1.1 (Linear Feasibility Problem) *Given a finite set of points $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\} \subset \mathbb{R}^d$ and a point $\mathbf{p}$, decide whether there is a subset $T \subseteq S$ of size at most $d+1$, such that $\mathbf{p} \in \text{conv}(T)$, and find T if it exists.*

While the Linear Feasibility Problem is called "Linear Programming Problem" in [7], we use a different name to avoid conflict with the definition of Linear

Programming in the optimization field, see [24]. We introduce and use some optimization models, mainly in Chapter 7. The Carathéodory Theorem states that, if $\mathbf{p} \in \text{conv}(S)$, then the corresponding Linear Feasibility Problem is *feasible*, i.e., has a solution T. The Linear Feasibility Problem can be formulated as a special case of the *Linear Optimization Problem*:

$$\min_{\mathbf{x}} \{ \mathbf{c}^T \mathbf{x} \mid A\mathbf{x} = \mathbf{b}; \mathbf{x} \geq \mathbf{0} \} \tag{2.1.1}$$

where $\mathbf{x}$ is the vector variable, $\mathbf{c}$ is the vector representing the linear objective function, and A is a matrix representing the linear equality constraints. To solve a Linear Feasibility Problem, we can formulate a Linear Optimization Problem with:

$$A = \begin{bmatrix} \mathbf{s}_1 & \dots & \mathbf{s}_n \\ 1 & \dots & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{2.1.2}$$

and $\mathbf{c}$ set to any proper length vector. Pivoting algorithms, see [30], find a solution $\mathbf{x}$ with at most $d + 1$ positive elements. With $\mathbf{c}$ not perpendicular to any edge of the feasible region, interior point methods also find a feasible solution $\mathbf{x}$. The positive elements of $\mathbf{x}$ correspond to the selected points from S.

Linear Optimization is a well studied area with a wide range of applications, and goes back at least till Danzig's paper in 1948, see [9]. The Linear Feasibility Problem is a special case of the Linear Optimization Problem, see [8] and [12] for results dedicated to this specific instance.

2.2 Colourful Feasibility and Colourful Core Feasibility Problems

For the Linear Feasibility Problem, the elements of S have no attribute to distinguish one from the other besides the coordinates.

With, for each point except $\mathbf{p}$, an additional attribute, called the *colour* and indexed by $1 \dots k$, the points can be partitioned into $k+1$ sets $S_1, \dots, S_k$. The *Colourful Feasibility Problem* (**CFP**) was introduced by Bárány and Onn [7] in 1997 in the following form: Given k sets $S_1, \dots, S_k$ of points in $\mathbb{R}^d$ and a point $\mathbf{p} \in \mathbb{R}^d$, decide if there is a set $T = \{\mathbf{t}_1, \dots, \mathbf{t}_k\}$ such that $\mathbf{p} \in \text{conv}(T)$, and $\mathbf{t}_i \in S_i$ for all i; and, if such a T exists, find one.

The **CFP** is clearly a generalization of the Linear Feasibility Problem as a Linear Feasibility Problem in $\mathbb{R}^d$ can be formulated as a **CFP** by setting $k = d+1$ and $S_1 = S_2 = \cdots = S_k$.

We define the *core* of a **CFP** as $\bigcap_{i=1}^{k} \text{conv}(S_i)$, and a **CFP** is a *Colourful Core Feasibility Problem* (**CCFP**) if $\mathbf{p}$ is in the core. Bárány and Onn [7] proved that a **CCFP** is NP-Complete for $k = d$. For **CCFP** with $k = d+1$, Bárány and Onn [7] commented that it is "an outstanding problem on the border line between tractable and intractable computational problems" as they proposed efficient algorithms (reviewed in Chapter 3) contrasting the complexity result for the case of $k = d$. Note that if **CCFP** with $k = d+1$ is polynomially solvable, then **CCFP** with any $k > d+1$ is polynomially solvable. Assume that **CCFP** with $k = d+1$ is polynomially solvable and we are given a **CCFP** problem with $k = d+2$. We can obtain an artificial **CCFP** problem with $k = d+1$ by removing a colour from the original problem, find a solution for artificial problem in polynomial time, then obtain a solution for the original

problem by adding an arbitrary point from the removed colour to the found solution. Since Bárány and Onn [3] proved that a **CCFP** with $k = d + 1$ is always feasible, the above method will not yield an infeasible artificial problem.

Without loss of generality, we make the following assumptions for **CFP**:

- $\mathbf{p} = \mathbf{0}$ (up to translation);
- $\mathbf{p} \notin S_i$ for all i (otherwise we have a trivial solution);
- $\forall_{i \neq j}(S_i \cap S_j = \emptyset)$ (up to removing duplicates);
- $\forall_i \forall_{\mathbf{s} \in S_i}(\|\mathbf{s}\| = 1)$ (with $\mathbf{p} = \mathbf{0}$, the membership of $\mathbf{p}$ to a convex hull is independent from scaling with positive coefficients).

In addition, we assume that the **CFP** satisfies:

- $k = d + 1$;
- $\forall_{1 \leq i \leq k}(|S_i| = d + 1)$.

Note that for $k = d+1$, a solution T represents the simplex $\text{conv}(T)$. Assuming that $|S_i| = d + 1$, we investigate the effect of the dimension d on the running time of the algorithms. Bárány [3] proved that a **CCFP** is always feasible if it satisfies the above two conditions. As Linear Feasibility Problem solvers could be used to reduce the size of $|S_i|$ to at most $d + 1$, we can assume without loss of generality that $|S_i| = d+1$ for a **CCFP** with $d+1$ colours. We also assume that $|S_i| = d + 1$ for general **CFP** in order to be able to benchmark different algorithms.

Definition 2.2.1 (Colourful Feasibility Problem) *Given* d+1 *sets* $S_1, \ldots, S_{d+1} \subset \mathbb{S}^d$ *such that* $|S_i| = d + 1$ *for all* i*, decide if there is a set* $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\}$ *such that* $\mathbf{0} \in \text{conv}(T)$ *and* $\mathbf{t}_i \in S_i$ *for each* i*.*

Definition 2.2.2 (Colourful Core Feasibility Problem) *Colourful Feasibility Problem with* $\mathbf{0} \in \bigcap_{i=1}^{d+1} \mathrm{conv}(S_i)$.

Let $\mathbf{S}$ denote $\biguplus_{i=1}^{d+1} S_i$ and call it a *colourful configuration*. It is still not known whether a polynomial-time algorithm to solve **CCFP** exists and, while not providing an answer to this question, we:

- discuss algorithms to solve **CFP** in Chapter 3 to Chapter 7;
- describe the random case generators used to test the algorithms in Chapter 8;
- provide the test results in Chapter 9;
- show a parity property and a lower bound for the number of solution of **CCFP** in Chapter 10.

Chapter 3

Bárány-Onn Algorithms

This chapter reviews two geometric algorithms for the *Colourful Core Feasibility Problem* (**CCFP**) proposed by Bárány and Onn [7], our implementation techniques and the corresponding complexity results. The convergency and efficiency proofs (Propositions 3.1.1, 3.1.2 and 3.1.3) originate from Bárány and Onn [7], and the following parts are our contributions:

- the geometric and algorithmic features, see Propositions 3.2.4 and 3.2.5,
- the complexity result for the general position, see Proposition 3.2.6,
- the discovery of an oscillation phenomenon, see Section 3.3.

Chapter 4 introduces a variant of the algorithms reviewed in this chapter which achieves a robust time performance as shown in Chapter 9.

3.1 First Bárány-Onn algorithm

Algorithm 1: Solver-Bárány-Onn-1

Input: **S**
Output: T
1 **begin**
2 initialize $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in S_i$ for $i = 1, ..., d+1$
3 **while** $\mathbf{0} \notin \text{conv}(T)$ **do**
4 $\mathbf{p} \leftarrow \underset{\mathbf{t} \in \text{conv}(T)}{\text{argmin}} (\|\mathbf{t}\|)$
5 Find an i such that $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$
6 $\mathbf{t}_i \leftarrow \underset{\mathbf{t} \in S_i}{\text{argmin}} \left(\mathbf{t}^T \mathbf{p}\right)$
7 **end**

The following proposition warranties that line 5 can always be executed, i.e., the algorithm keeps on replacing one point from T until $\mathbf{0} \in \text{conv}(T)$.

Proposition 3.1.1 *For* $\mathbf{0} \notin \text{conv}(T)$ *and* $\mathbf{p}$ *on the boundary of* $\text{conv}(T)$*, Solver-Bárány-Onn-1 can always find an index* $i \in \{1, \ldots, d+1\}$ *such that* $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$*. If* $\text{conv}(T)$ *is degenerate, all the points are considered to be on the boundary.*

Proof: For $\mathbf{0} \notin \text{conv}(T)$, $\mathbf{p}$ is the point of minimum norm on the boundary of $\text{conv}(T)$. We have two subcases depending on $\text{conv}(T)$ being full dimensional or degenerate. If $\text{conv}(T)$ is full dimensional, the system of equations

$$\begin{bmatrix} \mathbf{t}_1 & ... & \mathbf{t}_{d+1} \\ 1 & ... & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{3.1.1}$$

has a unique solution $\mathbf{x}$ with at least one zero coordinate x_i since $\mathbf{p}$ is on the boundary of $\text{conv}(T)$. Then an index i corresponding to a zero element of $\mathbf{x}$ satisfies $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$. If $\text{conv}(T)$ is degenerate, $T \cup \{\mathbf{p}\}$ can be embedded

into a lower dimensional space, and a Linear Feasibility Problem solver can find the proper i, see Section 2.1. □

Note that the algorithm convergence rate depends on the maximum radius $0 \leq \rho \leq 1$ of a ball inside the core.

Proposition 3.1.2 *Consider one iteration of Solver-Bárány-Onn-1, and let*

- $\overline{T}$ *and* $\underline{T}$ *be the values of* T *before and after the iteration, and*
- $\mathbf{p} = \underset{\mathbf{t} \in \mathrm{conv}(\overline{T})}{\mathrm{argmin}} (\|\mathbf{t}\|)$,

then a point $\mathbf{v} \in \mathrm{conv}(\underline{T})$ *can be found, such that:*

$$\|\mathbf{v}\|^2 \leq (1-\rho^2)\|\mathbf{p}\|^2 \quad \textit{if} \quad \rho > 0, \tag{3.1.2}$$

$$\frac{1}{\|\mathbf{v}\|^2} \geq 1 + \frac{1}{\|\mathbf{p}\|^2} \quad \textit{if} \quad \rho = 0. \tag{3.1.3}$$

Proof: As the case $d = 1$ is trivial, we assume that $d \geq 2$. Let $\mathbf{t}_i$ be the point found at line 6 of the iteration, and $\mathbf{v} = \underset{[\mathbf{p},\mathbf{t}_i]}{\mathrm{proj}}(\mathbf{0})$, then $\mathbf{v}$ satisfies the proposition. Since $\mathbf{0} \in \mathrm{conv}(S_i)$ and $\mathbf{t}_i$ is obtained from S_i by minimizing the dot product with $\mathbf{p}$, we have $\mathbf{t}_i^T\mathbf{p} \leq 0$ which implies that the angle between $\mathbf{t}_i$ and $\mathbf{p}$ is at least 90 degrees. Therefore $\mathbf{v}$ lies in the line segment $[\mathbf{p}, \mathbf{t}_i]$, and therefore $\mathbf{v} \in \mathrm{conv}(\underline{T})$ since both $\mathbf{p}$ and $\mathbf{t}_i$ are in $\mathrm{conv}(\underline{T})$. Let $\mathbf{w}$ be the point in the opposite direction of $\mathbf{p}$ such that $\mathrm{conv}(\mathbf{p}, \mathbf{w}, \mathbf{t}_i)$ is a right triangle. We have $\|\mathbf{w}\| \geq \rho$ and the points $\mathbf{t}_i$, $\mathbf{p}$, $\mathbf{w}$, $\mathbf{v}$ and $\mathbf{0}$ are in the same 2-dimensional plane, see Figure 3.1.

The two triangles $\mathrm{conv}(\mathbf{p}, \mathbf{v}, \mathbf{0})$ and $\mathrm{conv}(\mathbf{p}, \mathbf{w}, \mathbf{t}_i)$ have identical angle sizes and proportional edge lengths. Therefore we have:

$$\frac{\|\mathbf{v}\|}{\|\mathbf{p}\|} = \frac{\sqrt{\|\mathbf{t}_i\|^2 - \|\mathbf{w}\|^2}}{\sqrt{(\|\mathbf{p}\| + \|\mathbf{w}\|)^2 + (\|\mathbf{t}_i\|^2 - \|\mathbf{w}\|^2)}}. \tag{3.1.4}$$

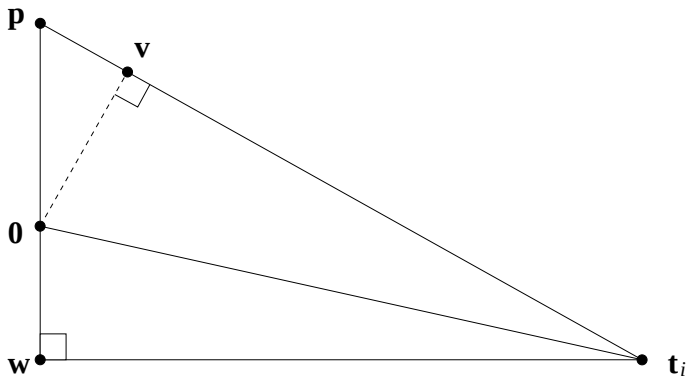


Figure 3.1: The points $\mathbf{t}_i$, $\mathbf{p}$, $\mathbf{w}$, $\mathbf{v}$ and $\mathbf{0}$ in a 2D subspace

that is, since $\|\mathbf{t}_i\| = 1$,

$$\frac{\|\mathbf{v}\|^2}{\|\mathbf{p}\|^2} = \frac{1 - \|\mathbf{w}\|^2}{\|\mathbf{p}\|^2 + 2\|\mathbf{p}\|\|\mathbf{w}\| + 1}. \tag{3.1.5}$$

that is, since $\|\mathbf{w}\| \geq \rho$,

$$\frac{\|\mathbf{v}\|^2}{\|\mathbf{p}\|^2} \leq \frac{1 - \rho^2}{\|\mathbf{p}\|^2 + 2\rho\|\mathbf{p}\| + 1}. \tag{3.1.6}$$

For $\rho > 0$, it yields inequality (3.1.2) and, for $\rho = 0$, inequality (3.1.3). □

Under the conditions of Proposition 3.1.2, Bárány and Onn [7] proved that:

$$\|\mathbf{v}\|^2 \leq (1 - \frac{\rho^2}{4})\|\mathbf{p}\|^2 \quad \text{if} \quad \rho > 0, \tag{3.1.7}$$

$$\frac{1}{\|\mathbf{v}\|^2} \geq \frac{1}{4} + \frac{1}{\|\mathbf{p}\|^2} \quad \text{if} \quad \rho = 0. \tag{3.1.8}$$

The differences with the inequalities (3.1.2) and (3.1.3) result from Bárány and Onn's assumption that $1 \leq \|\mathbf{t}_i\| \leq 2$ (for compatibility with rational number computation) instead of $\|\mathbf{t}_i\| = 1$. Despite the differences, both [7] and Proposition 3.1.2 imply the same upper bound on the number of iterations to get conv(T) ϵ-close to $\mathbf{0}$.

Proposition 3.1.3 *Solver-Bárány-Onn-1 needs at most* $O\left(\frac{1}{\rho^2}\log\frac{1}{\epsilon}\right)$ *(if* $\rho > 0$*) or* $O\left(\frac{1}{\epsilon^2}\right)$ *(if* $\rho = 0$*) iterations to get* $\mathrm{conv}(T)$ *ϵ-close to* $\mathbf{0}$.

Proof: We prove this proposition assuming inequalities (3.1.2) and (3.1.3). The method to prove this proposition assuming inequalities (3.1.7) and (3.1.8) is similar and is left to the readers. Let $\mathbf{p}_0$ be the point of minimum norm in $\mathrm{conv}(T)$ right before the first iteration and $\mathbf{p}_i$ be point of minimum norm in $\mathrm{conv}(T)$ right after the ith iteration. From inequalities (3.1.2) and (3.1.3) we obviously have

$$\|\mathbf{p}_i\|^2 \leq (1-\rho^2)\|\mathbf{p}_{i-1}\|^2 \quad \text{if} \quad \rho > 0, \tag{3.1.9}$$

$$\frac{1}{\|\mathbf{p}_i\|^2} \geq 1 + \frac{1}{\|\mathbf{p}_{i-1}\|^2} \quad \text{if} \quad \rho = 0. \tag{3.1.10}$$

The case $\rho = 1$ is trivial. If $0 < \rho < 1$, as obviously $\|\mathbf{p}_0\| \leq 1$, from inequality (3.1.9) we obtain

$$\|\mathbf{p}_i\|^2 \leq (1-\rho^2)^i\|\mathbf{p}_0\|^2 \leq (1-\rho^2)^i. \tag{3.1.11}$$

To make $\|\mathbf{p}_i\| \leq \epsilon$, we only need

$$i \geq \log_{1-\rho^2}\left(\epsilon^2\right) = \frac{-2\log\frac{1}{\epsilon}}{\log\left(1-\rho^2\right)}. \tag{3.1.12}$$

Then we can expand $\log\left(1-\rho^2\right)$ and obtain

$$i \geq \frac{-2\log\frac{1}{\epsilon}}{-\sum_{k=1}^{\infty}\left(\frac{\rho^{2k}}{k}\right)} \leq \frac{2\log\frac{1}{\epsilon}}{\rho^2} = O\left(\frac{1}{\rho^2}\log\frac{1}{\epsilon}\right). \tag{3.1.13}$$

If $\rho = 0$, by summing up inequality (3.1.10) from $i = 1$ to k, we obtain

$$\frac{1}{\|\mathbf{p}_k\|^2} \geq k + \frac{1}{\|\mathbf{p}_0\|^2} \geq k. \tag{3.1.14}$$

If $k = \frac{1}{\epsilon^2}$ then from inequality (3.1.14) we have $\|\mathbf{p}_k\| \leq \epsilon$, so the proposition is proved. □

Figure 3.2 illustrates Solver-Bárány-Onn-1 convergence.

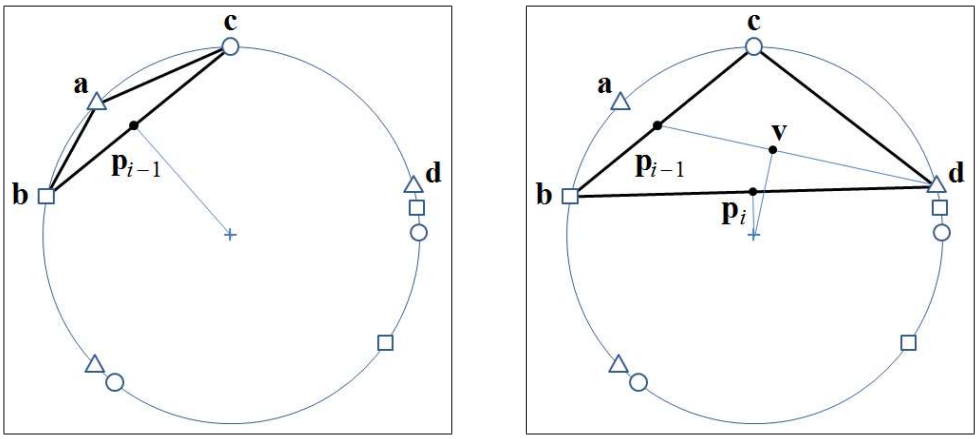


Figure 3.2: A sample iteration of Solver-Bárány-Onn-1

The shapes ◯, □ and △ stand for different colours. (a) and (b) show two states of the algorithm in the same iteration. The triangles connecting three colourful points represent the selection of T in those states. $\mathbf{p}_i$ denotes the point of minimum norm at the end of the ith iteration. One can reasonability assume $i = 1$ as no other initial T that leads to the state in (a) using Solver-Bárány-Onn-1 was found.

(a) at the beginning of the ith iteration, $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\mathbf{a}$ is to be replaced because $\mathbf{p}_{i-1} \in \text{conv}(\mathbf{b}, \mathbf{c})$;

(b) $\mathbf{a}$ has been replaced and $T = \{\mathbf{b}, \mathbf{c}, \mathbf{d}\}$; $\mathbf{v}$ is the projection of $\mathbf{0}$ on the line segment $[\mathbf{p}_{i-1}, \mathbf{d}]$; $\|\mathbf{v}\| < \|\mathbf{p}_{i-1}\|$ because of triangularity (see the proof of Proposition 3.1.2 for details) and $\|\mathbf{p}_i\| \leq \|\mathbf{v}\|$ because $\mathbf{v} \in \text{conv}(\mathbf{b}, \mathbf{c}, \mathbf{d})$, so $\|\mathbf{p}_i\| < \|\mathbf{p}_{i-1}\|$ guarantees the convergence.

For the rate of convergence, see Propositions 3.1.2 and 3.1.3.

Each iteration of Solver-Bárány-Onn-1 takes a number of arithmetic operations polynomial to d and $\log \frac{1}{\varepsilon}$, where ε is the precision of $\mathbf{p}$. We can finish all steps of an iteration in $O(d^3)$ arithmetic operations except for line 4, which computes $\mathbf{p}$, the point of minimum norm in a simplex. Therefore we can con-

sider this step as the time complexity bottleneck. The task of line 4 can be formulated as a continuous Convex Quadratic Optimization Problem (see Subsection 3.5.4), which is in turn solvable to ε-precision in a number of arithmetic operations polynomial to d and $\log \frac{1}{\varepsilon}$, see, for example, Anstreicher et al. [2] for an interior point algorithm that takes $O\left(n^{3.5} \log \frac{1}{\varepsilon}\right)$ arithmetic operations, where n is the number of variables of the Quadratic Optimization Problem. The Quadratic Optimization Problem to find $\mathbf{p}$ (formulation (3.5.21)) has $n = d+1$, and therefore each iteration takes $O\left((d+1)^{3.5} \log \frac{1}{\varepsilon}\right) = O\left(d^{3.5} \log \frac{1}{\varepsilon}\right)$ arithmetic operations.

By Proposition 3.1.3 and the above complexity result for each iteration, Solver-Bárány-Onn-1 needs at most $O\left(\frac{d^{3.5}}{\rho^2} \log \frac{1}{\epsilon} \log \frac{1}{\varepsilon}\right)$ (if $\rho > 0$) or $O\left(\frac{d^{3.5}}{\epsilon^2} \log \frac{1}{\varepsilon}\right)$ (if $\rho = 0$) arithmetic operations to get $\mathrm{conv}(T)$ ϵ-close to $\mathbf{0}$.

Note that the question whether the Colourful Core Feasibility Problem can be solved in a number of arithmetic operations polynomial to the problem size is still open, primarily because this bound depends on the value of ρ, which is a geometric property of the input coordinates. In this thesis we consider problem size as the number of real number necessary to represent the problem, which is $O(d^3)$. In Turing machine model handling rational numbers, the problem size can be considered as the number of bits encoding the problem, which is $O(d^3 N)$, where N is the number of bits to represent a rational number. We discuss the complexity only on real arithmetic model as introduced in Section 1.15. In case $\rho > 0$, the complexity is polynomial to the logarithmic of $\frac{1}{\epsilon}$ and $\frac{1}{\varepsilon}$, which is not considered a large factor because of the logarithmic. However, $\frac{1}{\rho^2}$ can be very large as ρ can get arbitrarily close to 0. In case $\rho > 0$, the complexity depends polynomially on $\frac{1}{\epsilon^2}$, which can also be very large.

Although the complexity upper bound is polynomial to the large factors such as $\frac{1}{\rho^2}$ and $\frac{1}{\epsilon}$, we have not found any input to make Solver-Bárány-Onn-1 achieve that upper bound. We notice that Solver-Bárány-Onn-1 does not visit the same T twice as after each iteration $\text{conv}(T)$ is warranted to be closer to $\mathbf{0}$. An alternative algorithm, Solver-Bárány-Onn-2, which does not hold this property but has lower complexity bound on each iteration is introduced in the next section.

3.2 Second Bárány-Onn algorithm

Solver-Bárány-Onn-1 computes the point of minimum norm in $\text{conv}(T)$ for each iteration, and this step is time-expensive. Bárány and Onn [7] proposed Solver-Bárány-Onn-2 to reduce the cost per iteration.

Algorithm 2: Solver-Bárány-Onn-2

Input: $\mathbf{S}$
Output: T
begin
  initialize $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in S_i$ for $i = 1, ..., d+1$
  $\mathbf{p} \leftarrow \mathbf{t}_1$
  while $\mathbf{0} \notin \text{conv}(T)$ **do**
    Find an i such that $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$
    $\mathbf{t}_i \leftarrow \underset{\mathbf{t} \in S_i}{\text{argmin}}(\mathbf{t}^T \mathbf{p})$
    $\mathbf{p} \leftarrow \underset{[\mathbf{p}, \mathbf{t}_i]}{\text{proj}}(\mathbf{0})$
    $\alpha \leftarrow \min\{\, \beta \,:\, \beta\mathbf{p} \in \text{conv}(T)\}$
    $\mathbf{p} \leftarrow \alpha\mathbf{p}$
end

Solver-Bárány-Onn-2 is similar to Solver-Bárány-Onn-1 except that $\mathbf{p}$ is

updated according to its previous value. The following three propositions are the analogue of Propositions 3.1.1, 3.1.2 and 3.1.3.

Proposition 3.2.1 *For* $\mathbf{0} \notin \text{conv}(T)$ *and* $\mathbf{p}$ *on the boundary of* $\text{conv}(T)$*, Solver-Bárány-Onn-2 can always find an index* $i \in \{1, \ldots, d+1\}$ *such that* $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$*. If* $\text{conv}(T)$ *is degenerate, all the points are considered to be on the boundary.*

Proof: Identical to Proposition 3.1.1. □

Proposition 3.2.2 *Consider one (except the last) iteration of Solver-Bárány-Onn-2, and let* $\overline{\mathbf{p}}$ *and* $\underline{\mathbf{p}}$ *be the values of* $\mathbf{p}$ *before and after the iteration. We have:*

$$\|\underline{\mathbf{p}}\|^2 \leq (1-\rho^2)\|\overline{\mathbf{p}}\|^2 \quad \text{if} \quad \rho > 0, \tag{3.2.15}$$

$$\frac{1}{\|\underline{\mathbf{p}}\|^2} \geq 1 + \frac{1}{\|\overline{\mathbf{p}}\|^2} \quad \text{if} \quad \rho = 0. \tag{3.2.16}$$

Proof: Similar to the proof of Proposition 3.1.2. □

Proposition 3.2.3 *Solver-Bárány-Onn-2 needs at most* $O\left(\frac{1}{\rho^2}\log\frac{1}{\epsilon}\right)$ *(if* $\rho > 0$*) or* $O\left(\frac{1}{\epsilon^2}\right)$ *(if* $\rho = 0$*) iterations to get* $\text{conv}(T)$ *ϵ-close to* $\mathbf{0}$*.*

Proof: Similar to the proof of Proposition 3.1.3. □

Figure 3.3 illustrates Solver-Bárány-Onn-2 convergence.

Besides line 8, each step of a Solver-Bárány-Onn-2 iteration can clearly be performed in $O(d^3)$ arithmetic operations. Line 8 computes a scalar α pushing $\mathbf{p}$ to the boundary of $\text{conv}(T)$ if $\mathbf{0} \notin \text{conv}(T)$, and to $\mathbf{0}$ otherwise. Bárány and Onn [7] used the following method taking $O(d^4)$ arithmetic operations:

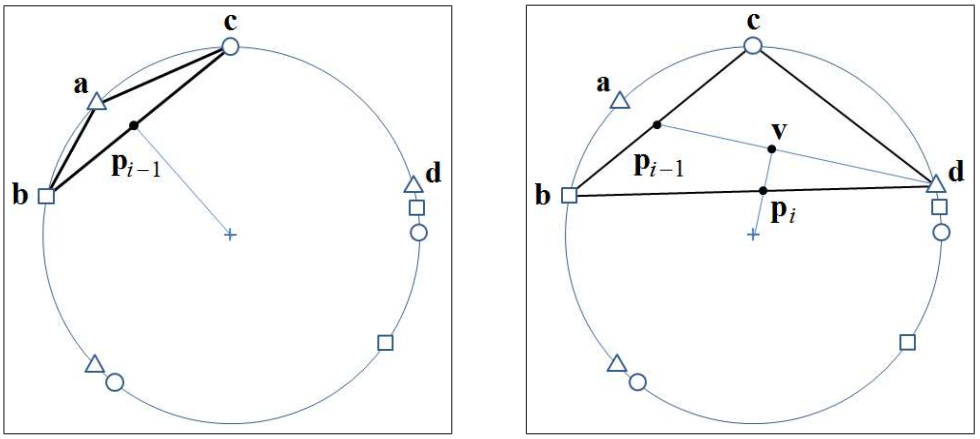


Figure 3.3: A sample iteration of Solver-Bárány-Onn-2

The shapes ◯, □ and △ stand for different colours. (a) and (b) show two states of the algorithm in the same iteration. The triangles connecting three colourful points represent the selection of T in those states. $\mathbf{p}_i$ denotes the point of minimum norm at the end of the ith iteration. One can reasonability assume $i = 1$ for this figure as no other initial T that leads to the state in (a) using Solver-Bárány-Onn-2 was found.

(a) at the beginning of the ith iteration, $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\mathbf{a}$ is to be replaced because $\mathbf{p}_{i-1} \in \text{conv}(\mathbf{b}, \mathbf{c})$;

(b) $\mathbf{a}$ has been replaced and $T = \{\mathbf{b}, \mathbf{c}, \mathbf{d}\}$; $\mathbf{v}$ is the projection of $\mathbf{0}$ on the line segment $[\mathbf{p}_{i-1}, \mathbf{d}]$; $\|\mathbf{v}\| < \|\mathbf{p}_{i-1}\|$ because of triangularity and $\|\mathbf{p}_i\| \leq \|\mathbf{v}\|$ because $\mathbf{p}_i$ is obtained by scaling $\mathbf{v}$ down to the boundary of $\text{conv}(\mathbf{b}, \mathbf{c}, \mathbf{d})$, so $\|\mathbf{p}_i\| < \|\mathbf{p}_{i-1}\|$ guarantees the convergence.

For the rate of convergence, see Propositions 3.2.2 and 3.2.3.

Intersect the line segment $[\mathbf{0}, \mathbf{p}]$ with each facet of $\text{conv}(T)$ and find a facet containing $\alpha\mathbf{p}$ with $0 \leq \alpha \leq 1$.

For T in general position, we found a technique to compute α in $O(d^3)$ arithmetic operations. We first need the following Propositions 3.2.4 and 3.2.5.

Proposition 3.2.4 *With the following assumptions:*

1. $d \geq 1$;
2. $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ *is in general position;*

3. $\mathbf{p} \in \text{conv}(T)$ *and* $\mathbf{0} \notin \text{conv}(T)$*;*
4. $\alpha = \min\{\beta \in (0,1] \,:\, \beta\mathbf{p} \in \text{conv}(T)\}$*;*
5. *for* $1 \le i \le d+1$, $\mathbf{y}_i \in I\!R^d$ *and* $b_i \in I\!R$ *satisfies:*
   (a) $\mathbf{y}_i^T \mathbf{t}_i = 1 - b_i$,
   (b) $\mathbf{y_i}^T \mathbf{t}_j = -b_i$ *if* $1 \le j \ne i \le d+1$*;*
6. *for* $1 \le i \le d+1$*:*
   (a) $H_i = \{\mathbf{x} \,:\, \mathbf{y}_i^T \mathbf{x} = -b_i\}$,
   (b) $H_i^+ = \{\mathbf{x} \,:\, \mathbf{y}_i^T \mathbf{x} \ge -b_i\}$,
   (c) $H_i^- = \{\mathbf{x} \,:\, \mathbf{y}_i^T \mathbf{x} \le -b_i\}$*;*
7. $I^0 = \{i \,:\, b_i = 0\}$, $I^+ = \{i \,:\, b_i > 0\}$ *and* $I^- = \{i \,:\, b_i < 0\}$*;*
8. $\mathcal{I} = \{i \,:\, \alpha\mathbf{p} \in H_i\}$ *and* $I = \{1, \ldots, d+1\} = I^0 \cup I^+ \cup I^-$.

we have the following properties:

1. $\text{aff}(T \setminus \{\mathbf{t}_i\}) = H_i$ *for all* $1 \le i \le d+1$*;*
2. $\text{conv}(T) = \bigcap_{i=1}^{d+1} H_i^+ = \bigcap_{i=1}^{d+1} \{\mathbf{x} \,:\, \mathbf{y}_i^T \mathbf{x} \ge -b_i\}$*;*
3. $\mathbf{y}_i^T \mathbf{p} > 0$ *for* $i \in I^-$*;*
4. $\mathcal{I} \ne \emptyset$*;*
5. $I^- \ne \emptyset$*;*
6. $\mathcal{I} \cap I^- \ne \emptyset$.

Proof: Property *1* holds as $\mathbf{t}_j \in H_i$ for all $1 \le j \ne i \le d+1$ by Assumptions *5(b)* and *6(a)*. Property *1* states that the simplex $\text{conv}(T)$ is bounded by the H_i's. For each $1 \le i \le d+1$, either $\text{conv}(T) \in H_i^+$ or $\text{conv}(T) \in H_i^-$. Assumption *5(a)* means that $\mathbf{t}_i$ is in H_i^+ but not in H_i^-, yielding $\text{conv}(T) \in H_i^+$

and Property *2*. Property *2* and Assumption *3* imply Property *3*. If $\mathcal{I} = \emptyset$, there is an $0 < \varepsilon < \min(\frac{\alpha}{\|\mathbf{p}\|}, \alpha)$ such that the ball $\mathbb{B}(\varepsilon, \alpha\mathbf{p})$ is inside $\text{conv}(T)$. We have $(\alpha - \varepsilon) \in (0, 1]$ and $(\alpha - \varepsilon)\mathbf{p} \in \mathbb{B}(\varepsilon, \alpha\mathbf{p}) \subset \text{conv}(T)$, which contradict Assumption *4*. Hence, Property *4* holds. If $I^- = \emptyset$, we have $b_i \geq 0$ for $1 \leq i \leq d+1$ and, by Property *2*, we have $\mathbf{0} \in \text{conv}(T)$ which contradict Assumption *3*. Hence, Property *5* holds.

There exist $0 < \varepsilon < \min(\frac{\alpha}{\|\mathbf{p}\|}, \alpha)$ such that $\mathbb{B}(\varepsilon, \alpha\mathbf{p}) \in \bigcap_{i \in I \setminus \mathcal{I}} H_i^+$ and then $(\alpha - \varepsilon)\mathbf{p} \in \bigcap_{i \in I \setminus \mathcal{I}} H_i^+$. For $i \in \mathcal{I} \cap I^0$, the fact that both $\mathbf{0}$ and $\alpha\mathbf{p}$ are in H_i implies $(\alpha - \varepsilon)\mathbf{p} \in H_i^+$. For $i \in \mathcal{I} \cap I^+$, $(\alpha - \varepsilon)\mathbf{y}_i^T\mathbf{p} = -(\alpha - \varepsilon)b_i > -b_i$ implies $(\alpha - \varepsilon)\mathbf{p} \in H_i^+$. Hence, $(\alpha - \varepsilon)\mathbf{p} \in \bigcap_{i \in I}^{i \notin \mathcal{I} \cap I^0} H_i^+$

If we assume $\mathcal{I} \cap I^- = \emptyset$, then $(\alpha - \varepsilon)\mathbf{p} \in \bigcap_{i \in I^-} H_i^+$ because of the definition of I^- in Assumption *7*. This in turn implies $(\alpha - \varepsilon)\mathbf{p} \in \text{conv}(T)$ contradicting the definition of α. Hence, Property *6* is true. □

Proposition 3.2.5 *With the following assumptions:*

1. $d \geq 1$;
2. *the set* $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ *is in general position;*
3. $\begin{bmatrix} \mathbf{y}_1^T & b_1 \\ ... & ... \\ \mathbf{y}_{d+1}^T & b_{d+1} \end{bmatrix} = \begin{bmatrix} \mathbf{t}_1 & ... & \mathbf{t}_{d+1} \\ 1 & ... & 1 \end{bmatrix}^{-1}$, *where* $\mathbf{y}_i \in \mathbb{R}^d$ *and* $b_i \in \mathbb{R}$.

for $1 \leq i \leq d+1$, $\mathbf{y}_i$ *and* b_i *satisfy the following properties:*

1. $\mathbf{y}_i^T\mathbf{t}_i = 1 - b_i$;
2. $\mathbf{y_i}^T\mathbf{t}_j = -b_i$ *if* $1 \leq j \neq i \leq d+1$.

Proof: According to Assumption *3*, we have:

$$\begin{bmatrix} \mathbf{y}_1^T & b_1 \\ \dots & \dots \\ \mathbf{y}_{d+1}^T & b_{d+1} \end{bmatrix} \begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{3.2.17}$$

By inspecting the row-column multiplication that yield the 0-1 elements on right hand side, one can easily check that the properties are true. □

Following is a method to find the minimum scalar factor that shrinks $\mathbf{p}$ to the boundary of $\text{conv}(T)$, given that $\mathbf{0} \notin \text{conv}(T)$, $\mathbf{p} \in \text{conv}(T)$ and $\dim(T) = d + 1$.

Algorithm 3: Shrink-Simplex-Point

Input: $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$, $\mathbf{p}$
Output: α
1 **begin**
2 $\quad \begin{bmatrix} \mathbf{y}_1^T & b_1 \\ \dots & \dots \\ \mathbf{y}_{d+1}^T & b_{d+1} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix}^{-1}$
3 $\quad \alpha \leftarrow 0$
4 $\quad$ **for** *i such that* $b_i < 0$ **do**
5 $\quad\quad$ obtain β_i by solving the equation $\mathbf{y}_i^T(\beta_i \mathbf{p}) = -b_i$
6 $\quad\quad$ **if** $\beta_i \geq \alpha$ **then** $\alpha \leftarrow \beta_i$
7 **end**

Proposition 3.2.6 *If* $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ *is in general position,* $\mathbf{p} \in \text{conv}(T)$ *and* $\mathbf{0} \notin \text{conv}(T)$*, the algorithm Shrink-Simplex-Point gives* $\alpha = \{\beta : \beta\mathbf{p} \in \text{conv}(T)\}$ *in* $O(d^3)$ *arithmetic operations.*

Proof: By Proposition 3.2.5, the $\mathbf{y}_i$'s and b_i's computed in Line 2, together with T and $\mathbf{p}$ fit the assumptions of Proposition 3.2.4. Hence, the minimum β

satisfying $\beta\mathbf{p} \in \text{conv}(T)$ is on one of the affine hyperplanes $\{\mathbf{x} : \mathbf{y}_i^T\mathbf{x} = -b_i\}$ such that $b_i < 0$, by Property *6* of Proposition 3.2.4.

By Property *5* of Proposition 3.2.4, the algorithm gets into the for-loop of lines 4-6 and find some β_i's grater than 0, and the only maximum β_i found satisfies $\beta\mathbf{p} \in \text{conv}(T)$. Hence, the algorithm obtains

$$\alpha = \min\{\beta : \beta\mathbf{p} \in \text{conv}(T)\}. \tag{3.2.18}$$

The complexity of Shrink-Simplex-Point is dominated by line 2, which inverts a $(d+1)\times(d+1)$ matrix to find the defining affine hyperplanes of $\text{conv}(T)$. It can be done in $O(d^3)$ arithmetic operations. Therefore the algorithm takes $O(d^3)$ arithmetic operations.

T must be in general position for the algorithm Shrink-Simplex-Point, otherwise $\begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix}$ is not invertible. □

3.3 Oscillations for the Second Bárány-Onn Algorithm

Solver-Bárány-Onn-1 does not visit the same colourful simplex represented by T twice. This nice property is guaranteed by its definition of $\mathbf{p}$. However, Solver-Bárány-Onn-2 does not guarantee the same property for its $\mathbf{p}$. During the numerical experiments we found **CCFP** cases that let Solver-Bárány-Onn-2 visit the same sequence of T for many times with $\|\mathbf{p}\|$ decreasing slowly, especially when ρ, the maximum radius of a ball inscribed in the core, is small. We call this phenomenon *oscillation*. When ρ is small, the large upper bound of number of iterations indicated by Proposition 3.2.3 allows Solver-Bárány-Onn-2 to oscillate for many iterations.

Different **CCFP** cases can oscillate in different ways. For example, the length of the sequence of oscillating T that might be different. Following is the description of a three-dimensional construction - which can be generalized to higher dimension - that causes Solver-Bárány-Onn-2 to oscillate between two T. We use $\mathbf{s}_j^i$ to denote the jth point of the ith colour.

1. select a small enough $\epsilon > 0$;
2. $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_3$ are on an affine hyperplane that is ϵ-close to $\mathbf{0}$; in other words, $\mathbf{0}$ is ϵ-"below" $\text{aff}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$;
3. the area of $\text{conv}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ is large enough, and $\underset{\text{aff}(\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}_3)}{\text{proj}}\ \mathbf{0}$ is close to the center of $\text{conv}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$;
4. $\mathbf{s}_1^4$ and $\mathbf{s}_2^4$ are separated by $\text{aff}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ from $\mathbf{0}$, and they are ϵ^2-close to $\text{aff}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ and almost at opposite directions of each other; in other words, $\mathbf{s}_1^4$ and $\mathbf{s}_1^4$ are ϵ^2 "above" $\text{aff}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ and $(\mathbf{s}_1^4)^T\mathbf{s}_2^4$ is close to -1;
5. $\mathbf{s}_3^4$ is a normal vector of $\text{aff}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ and "below" $\mathbf{0}$;
6. $\mathbf{t}_4$ equals to $\mathbf{s}_1^4$;
7. $\mathbf{p} \in \text{conv}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$, $\|\mathbf{p}\| \gg \epsilon$, also the angle between $\mathbf{p}$ and $\mathbf{t}_2$ is significantly greater than $\frac{\pi}{2}$ and less than π.

Once the execution of Solver-Bárány-Onn-2 reach the above state, T oscillates between $\{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4 = \mathbf{s}_1^4\}$ and $\{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4 = \mathbf{s}_2^4\}$ until $\|\mathbf{p}\|$ is as small as a value comparable to ϵ. The smaller ϵ is, the longer the oscillation.

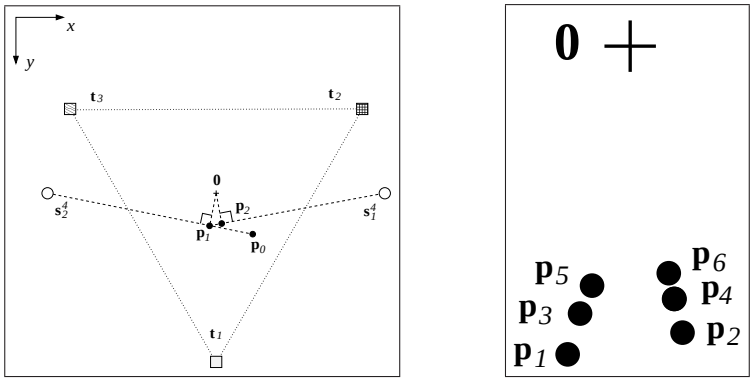
Example 3.3.1 (A 3D oscillation CCFP pattern)

Construct the **CCFP** *such that it reaches the following state during the execution of Solver-Bárány-Onn-2:*

- $\mathbf{t}_1 = \begin{bmatrix} 0 \\ \sqrt{1-\epsilon^2} \\ \epsilon \end{bmatrix}$, $\mathbf{t}_2 = \begin{bmatrix} \sqrt{1-\epsilon^2}\cos\frac{\pi}{6} \\ -\sqrt{1-\epsilon^2}\sin\frac{\pi}{6} \\ \epsilon \end{bmatrix}$, $\mathbf{t}_3 = \begin{bmatrix} -\sqrt{1-\epsilon^2}\cos\frac{\pi}{6} \\ -\sqrt{1-\epsilon^2}\sin\frac{\pi}{6} \\ \epsilon \end{bmatrix}$;

- $\mathbf{s}_1^4 = \begin{bmatrix} \sqrt{1-(\epsilon+\epsilon^2)^2} \\ 0 \\ \epsilon+\epsilon^2 \end{bmatrix}$, $\mathbf{s}_2^4 = \begin{bmatrix} -\sqrt{1-(\epsilon+\epsilon^2)^2} \\ 0 \\ \epsilon+\epsilon^2 \end{bmatrix}$,
  $\mathbf{s}_3^4 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$, $\mathbf{s}_4^4 = \begin{bmatrix} 0 \\ -\sqrt{1-(\epsilon+\epsilon^2)^2} \\ \epsilon+\epsilon^2 \end{bmatrix}$;
- $\mathbf{p} = \mathbf{p}_0 = \begin{bmatrix} \alpha\sqrt{1-\epsilon^2}\cos\frac{\pi}{4} \\ \alpha\sqrt{1-\epsilon^2}\sin\frac{\pi}{4} \\ \epsilon \end{bmatrix}$ *where* $\epsilon \ll \alpha < 1$, $\mathbf{t}_4 = \mathbf{s}_1^4$.

Once the execution of Solver-Bárány-Onn-2 reaches the above state, see Figure 3.4, T oscillates between $\mathbf{t}_4 = \mathbf{s}_1^4$ *and* $\mathbf{t}_4 = \mathbf{s}_2^4$ *without replacing* $\mathbf{t}_1$, $\mathbf{t}_2$ *and* $\mathbf{t}_3$. *It then sets* $\mathbf{t}_4 = \mathbf{s}_3^4$ *and quits the oscillation only when* $\|\mathbf{p}\|$ *is small enough. If we start with* $\mathbf{t}_4 = \mathbf{p} = \mathbf{s}_4^4$ *and let* $\mathbf{t}_1$ *to* $\mathbf{t}_3$ *be the same as the above state, then the above state is reached at the next iteration.*



(a) (b)

Figure 3.4: The 3D pattern of an oscillating **CCFP** case
(a) is the orthographic view from $[0 \quad 0 \quad 1]^T$. $\mathbf{p} = \mathbf{p}_0$. $\mathbf{p}_i$ is the value of $\mathbf{p}$ after another i iterations. Because $\epsilon^2 \ll \epsilon$, we can approximate $\mathbf{p}_{i+1}$ as the projection of $\mathbf{0}$ on $[\mathbf{s}_2^4, \mathbf{p}_i]$ if i is even, and on $[\mathbf{s}_1^4, \mathbf{p}_i]$ if i is odd. $\mathbf{s}_3^4$ and $\mathbf{s}_4^4$ are not in this figure. (b) is a magnification of the central part of (a) showing the slow convergence of $\mathbf{p}$.

□

3.4 Oscillation Detection

Solver-Bárány-Onn-2 can oscillate, while Solver-Bárány-Onn-1 will not, see Section 3.1. However, an iteration of Solver-Bárány-Onn-2 is faster than an iteration of Solver-Bárány-Onn-1 (at least in the currently found implementation technique) so Solver-Bárány-Onn-2 iterations are better in this sense. A way to improve the time performance is to use a hybrid algorithm: perform Solver-Bárány-Onn-2 steps in each iteration with some detection of oscillation, and use a computationally heavier Solver-Bárány-Onn-1 step to break possible oscillation. We implemented a version of the hybrid algorithm which yields improvement in practical performance. The test results are provided in [11]. However, we do not discuss this algorithm further in this thesis because

1. Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2 algorithms already introduced the geometric idea of the convergence rate of the steps;

2. we have not investigated an efficient and reliable ways of detecting oscillation, except using a fixed length, say 10, buffer to keep a certain length of history of T, and compare the value of T with those in buffer at each iteration;

3. according to the current stage of algorithm development for **CCFP**, we want to achieve theoretical improvements (such as getting algorithms with lower big-O complexity), but the hybrid algorithm does not seems to yield one;

4. the geometric structure is exploited in Chapter 4 and an algorithm is designed on top of Solver-Bárány-Onn-1, such that both the number of

iterations and the running time are significantly improved, and without oscillation;

As the practical performance becomes important, the oscillation detection can become important. In particular two problems need to considered:

- how to detect oscillation efficiently and reliability;
- how to distinguish long oscillations from short oscillations (i.e., oscillations less than 10 rounds), so the algorithm can ignore short oscillations.

3.5 Implementation

This section describes some implementation details of algorithms Solver-Bárány-Onn-1 and Solver-Báráiny-Onn-2, including the data structures and the subroutines used to achieve the best-effort efficiency. Since the algorithms are implemented in *MATLAB* language, knowledge of *MATLAB* is assumed fo this section. The source code of the algorithms in this thesis is available at [14].

3.5.1 Input Data

We use a pair of matrix variables named `Pts` and `ColorPartition` together to represent a configuration $\mathbf{S}$. Each column of `Pts` holds the coordinates of a point from $\mathbf{S}$, and the columns arrange in the way that the points with smaller colour indices have smaller column indices than those with larger colour indices. `ColorPartition` is a vector matrix of length $d + 1$, and its ith element tells the number of points in S_i.

Example 3.5.1 (Input Data Representation in 2D)

The data

- $S_1 = \left\{ \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -0.5 \\ -0.4 \end{bmatrix} \right\}$, *and*
- $S_2 = \left\{ \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} -0.3 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 1 \end{bmatrix} \right\}$, *and*
- $S_3 = \left\{ \begin{bmatrix} 5 \\ 0.1 \end{bmatrix}, \begin{bmatrix} -6 \\ 0.2 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} \right\}$

can be represented in MATLAB by running the following code:

```
Pts = [0.5  1  -0.5  -1  -0.3  0.3    5   -6    1; ...
       0.5  0  -0.4   0     1    1  0.1  0.2  0.5 ];
ColorPartition = [3 3 3];
```

□

3.5.2 Numerical Tolerance

Our implementations of algorithms use double precision floating point numbers to approximately represent numbers in $\mathbb{R}$, and the arithmetic operations introduce round off errors, which need some way tolerate. We define a variable named `TOLERANCE` holding a small positive value in our program, then any number with magnitude less than can be considered zero. Typically we set `TOLERANCE` to 10^{-8} or 10^{-12}.

An example of using `TOLERANCE` at each iteration is when selecting a vertex from T to be replaced. Mathematically, if $\lambda_i = 0$ then $\mathbf{t}_i$ can be replaced (see Subsection 3.5.5). However, because of the numerical error, the computed

floating point representation of λ_i will usually not exactly be 0 even if, mathematically, it should be. Therefore, our implementation treats λ_i as 0 if the magnitude of its computed floating representation is less than `TOLERANCE`.

3.5.3 Testing whether a Simplex Covers a Point

In our algorithms we need to check whether $\mathbf{0} \in \text{conv}(T)$ in each iteration. Our implementation works as follows:

First invoke `linsolve` function in *MATLAB* to solve the linear system

$$\begin{bmatrix} \mathbf{t}_1 & ... & \mathbf{t}_{d+1} \\ 1 & ... & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{3.5.19}$$

and `linsolve` returns a solution $\mathbf{x}$ and detect whether the linear system is well-conditioned at the same time. If the linear system is well-conditioned then we use the returned $\mathbf{x}$ to decide whether $\mathbf{0} \in \text{conv}(T)$. If the linear system is ill-conditioned then use the `linprog` function in *MATLAB Optimization Toolbox* to solve the following Linear Optimization Problem with variable $\mathbf{x}$:

$$\begin{array}{lll} \min & & \mathbf{0}^T\mathbf{x} \\ \text{s.t.} & \begin{bmatrix} \mathbf{t}_1 & ... & \mathbf{t}_{d+1} \\ 1 & ... & 1 \end{bmatrix} \mathbf{x} & = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \\ & \mathbf{x} & \geq \mathbf{0}. \end{array} \tag{3.5.20}$$

If $\mathbf{0} \in \text{conv}(T)$, then the above problem is feasible and `linprog` returns a solution, otherwise `linprog` will indicate infeasibility, which means $\mathbf{0} \notin \text{conv}(T)$.

When T is in general position, it takes $O(d^3)$ arithmetic operations to check whether $\mathbf{0} \in \text{conv}(T)$; when T is degenerate, `linprog` is invoked to find a feasible solution with acceptable performance.

3.5.4 Finding the Minimum Norm Point in a Simplex

In the implementation of Solver-Bárány-1, we use the `quadprog` function in *MATLAB Optimization Toolbox* to solve the following Convex Quadratic Optimization Problem with variables λ_1, ..., λ_{d+1}:

$$\begin{aligned} \min \quad & \sum_{1\le i,j\le d+1} \lambda_i \lambda_j \mathbf{t}_i^T \mathbf{t}_j \\ \text{s.t.} \quad & \sum_{1\le i\le d+1} \lambda_i = 1 \\ & \lambda_i \ge \mathbf{0} \quad \text{for } 1 \le i \le d+1. \end{aligned} \tag{3.5.21}$$

Let λ_1^*, ..., λ_{d+1}^* be the optimal solution, then the point in conv(T) with minimum norm is,

$$\lambda_1^* \mathbf{t}_1 + \lambda_2^* \mathbf{t}_2 + ... + \lambda_{d+1}^* \mathbf{t}_{d+1}. \tag{3.5.22}$$

3.5.5 Selecting a Vertex to be Replaced

The algorithms introduced in this chapter need to select a colour index i such that $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$ for a point $\mathbf{p}$ on the boundary of conv(T). Then the point $\mathbf{t}_i \in T$ is replaced.

For Solver-Bárány-Onn-1, $\mathbf{p}$ is the point of minimum norm in conv(T). When we obtain an optimal solution λ_1^*, ..., λ_{d+1}^* of formulation (3.5.21) to find $\mathbf{p}$, any i such that $\lambda_i^* = 0$ can be selected. If conv(T) is degenerate, we may have $\lambda_i^* > 0$ for all i's. In that case we use the `linprog` function to test the colour indices, with a method similar to Subsection 3.5.4, and find an i such that $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$.

In the implementation of Solver-Bárány-Onn-2, i is also selected when computing $\mathbf{p}$ if T is in general solution: while Shrink-Simplex-Point is used to compute α, the index i such that $\alpha = \beta_i$ can be selected. If T is degenerate,

the colour indices are tested in the same way as Solver-Bárány-Onn-1.

Chapter 4

Multi-Update Algorithm for Colourful Core Feasibility Problem

In the algorithms Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2 (from Chapter 3) solving the *Colourful Core Feasibility Problem* (**CCFP**), the point $\mathbf{p}$ is updated until $\mathbf{0} \in \text{conv}(T)$. The fact that $\|\mathbf{p}\|$ is reduced after each update proves that the algorithms convergence and we can consider $\mathbf{p}$ as the *convergence indicator* of Solver-Bárány-Onn-1 and Solver-Bárány-2. To improve the time performance of an algorithm, one can consider whether the convergence indicator can be reduced faster. In this chapter we introduce such an algorithm, which is based on Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2, but attempts to reduce $\mathbf{p}$ more frequently.

4.1 Multi-Update Algorithm

Each iteration of Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2 selects a colour i such that $\mathbf{t}_i$ can be replaced, and we can consider $\mathbf{t}_i$ to be a *replaceable point*

of the iteration. These two algorithms replace one replaceable point at each iteration, and with each replacement $\|\mathbf{p}\|$ is reduced by a bounded factor, see Propositions 3.1.2 and 3.2.2.

When $\mathbf{p}$ is in a k-dimensional face of $\text{conv}(T)$, the number of replaceable points in this iteration is $d-k$, therefore, for some iteration, we may find multiple replaceable points corresponding to a $k < d-1$ dimensional face. To speed up the convergence of algorithm, we studied whether there is a method to replace all replaceable points at each iteration, while having each replacement reducing $\|\mathbf{p}\|$ by a bounded factor. The answer is yes, and such an algorithm follows:

Algorithm 4: Solver-Multi-Update

Input: $\mathbf{S}$
Output: T
begin
  initialize $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in S_i$ for $i = 1, ..., d+1$
  $\mathbf{p} \leftarrow \mathbf{t}_1$
  while $\mathbf{0} \notin \text{conv}(T)$ **do**
    if $\dim(T) = d$ **then**
      $I_{\text{generate}} \leftarrow \{\, i \,:\, \mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})\}$
    else
      let $I_{\text{generate}} = \{i\}$ such that $\mathbf{p} \in \text{conv}(T \setminus \{\mathbf{t}_i\})$
    for $i \in I_{\text{generate}}$ **do**
      $\mathbf{t}_i \leftarrow \underset{\mathbf{t} \in S_i}{\text{argmin}}(\mathbf{p}^T\mathbf{t})$
      $\mathbf{p} \leftarrow \underset{[\mathbf{p},\mathbf{t}_i]}{\text{proj}}(\mathbf{0})$
    $\mathbf{p} \leftarrow \underset{\mathbf{t} \in \text{conv}(T)}{\text{argmin}}(\|\mathbf{t}\|)$
end

Solver-Multi-Update diffs from Solver-Bárány-Onn-1 and Solver-Bárány-

Onn-2 as it might replace multiple points in T between two consecutive computations of minimum norm point in $\text{conv}(T)$.

Similar to the algorithms from Chapter 3, we have the following propositions to guarantee that a colourful simplex ϵ-close to $\mathbf{0}$ can be obtained in at most $O\left(\frac{1}{\rho^2}\log\frac{1}{\epsilon}\right)$ (if $\rho > 0$) or $O\left(\frac{1}{\epsilon^2}\right)$ (if $\rho = 0$) inner-iterations, where ρ is the maximum radius of a ball inside the core.

Proposition 4.1.1 *If* $\mathbf{0} \notin \text{conv}(T)$ *and* $\mathbf{p}$ *is on the boundary of* $\text{conv}(T)$ *(in case* $\text{conv}(T)$ *is degenerate, all points on it are considered to be on the boundary), then* $I_{\text{generate}} \neq \emptyset$ *and there exist a method to find it.*

Proof: The proof of $I_{\text{generate}} \neq \emptyset$ is identical to the proof of Proposition 3.1.1. The proof of existing method to find I_{generate} is also the same as for Proposition 3.1.1, except putting all possible i into I_{generate} if T is in general position. □

Proposition 4.1.2 *For an arbitrary, except the last, inner-iteration of Solver-Multi-Update, let* $\overline{\mathbf{p}}$ *and* $\underline{\mathbf{p}}$ *be the values of* $\mathbf{p}$ *at the beginning and the end, respectively. We have:*

$$\|\underline{\mathbf{p}}\|^2 \leq (1-\rho^2)\|\overline{\mathbf{p}}\|^2 \quad \text{if} \quad \rho > 0, \tag{4.1.1}$$

$$\frac{1}{\|\underline{\mathbf{p}}\|^2} \geq 1 + \frac{1}{\|\overline{\mathbf{p}}\|^2} \quad \text{if} \quad \rho = 0. \tag{4.1.2}$$

Proof: Similar to the proof of Proposition 3.1.2. □

Proposition 4.1.3 *Solver-Multi-Update needs at most* $O\left(\frac{1}{\rho^2}\log\frac{1}{\epsilon}\right)$ *(if* $\rho > 0$*) or* $O\left(\frac{1}{\epsilon^2}\right)$ *(if* $\rho = 0$*) inner-iterations to get* $\text{conv}(T)$ *ϵ-close to* $\mathbf{0}$*.*

Proof: Similar to the proof of Proposition 3.1.3. □

Proposition 4.1.2 states that each replacement of a point in T reduces $\|\mathbf{p}\|$, and the reduction rate has the same lower bound as an iteration of Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2. It is remarkable that Solver-Multi-Update does not have two outer-iterations ending up with the same T, as for to Solver-Bárány-Onn-1.

Figure 4.1 illustrates Solver-Multi-Update convergence.

4.2 Implementation

As the Solver-Multi-Update is based on the algorithms from Bárány and Onn [7], its implementation is also similar. All the implementation methods from Section 3.5 also apply to the implementation of Solver-Multi-Update. In addition, a sorting method is used to reduce the numerical error.

4.2.1 Accumulation of Numerical Errors

We use double precision floating point arithmetic to represent real numbers in the implementation of the algorithms. Almost all the arithmetic operations, such as addition and multiplication, bring in either truncation or round-off errors.

The coordinates of the colourful points are the inputs to the algorithm implementations and we can consider them to be exact and have no numerical error. Any data computed directly or indirectly from the coordinates by arithmetic operations can have error. Assume x is the piece of data that we start with, and we want to apply the real arithmetic functions f_1, f_2, ..., f_n on it to obtain $\mathrm{f}_\mathrm{n}(\mathrm{f}_{\mathrm{n}-1}(...(\mathrm{f}_1(x))...))$. Assume $\hat{\mathrm{f}}_\mathrm{i}$ is the actual implementation of f_i for

each i, and ϵ_i is the corresponding upper bound of relative error. Therefore, as a worst case, the computed result can become,

$$\hat{\mathrm{f}}_{\mathrm{n}}(\hat{\mathrm{f}}_{\mathrm{n-1}}(...(\hat{\mathrm{f}}_1(x))...)) = \mathrm{f}_{\mathrm{n}}(\mathrm{f}_{\mathrm{n-1}}(...(\mathrm{f}_1(x))...)) \cdot \prod_{i=1}^{n}(1+\epsilon_i). \tag{4.2.3}$$

The numerical error may accumulate as the error caused by $\hat{\mathrm{f}}_1$, $\hat{\mathrm{f}}_2$, ..., $\hat{\mathrm{f}}_{\mathrm{i}}$ may propagate to $\hat{\mathrm{f}}_{\mathrm{i+1}}$.

In Solver-Multi-Update, the update of point $\mathbf{p}$ in each inner-iteration is based on its previous value, so the variable representing it in our implementation has accumulating numerical error during the executions. The implementations of Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2 also accumulate numerical error on $\mathbf{p}$, but the effect is not as significant as Solver-Multi-Update.

4.2.2 Improving Numerical Sum by Sorting

Let a and b be the floating representation of two real numbers, and a is much larger than b in magnitude, let's say $|a| > 10^6 |b|$. Let c be the result of applying floating point addition on a and b. Then the bnumerical error is caused by discarding the end significant digits of b during the addition. Without considering whether floating point addition is commutative, from the following example we can notice that summing up a list of floating numbers in different orders makes difference.

Example 4.2.1 (Improving Numerical Sum by Changing Order)
a_0, a_1,...,a_{100} are double precision floating point numbers. a_0 represents 1, and a_1 to a_{100} all represent 10^{-16}. If we sum them up in the order of a_0, a_1,...,a_{100}, then the result has no difference with 1. However, if we sum up a_1 to a_{100}

before adding a_0 to the final sum, we have a result larger than 1*, which is more accurate.*

Following is the test result with MATLAB:

```
>> A = ones(1,101);
>> A(1,2:101) = 10^(-16);
>> disp(sum(A)-1);
     0


>> disp(sum(A(1,2:101))+A(1,1)-1);
  9.9920e-015
```

□

The heuristic we applied to improve the numerical sum for a list L of floating numbers is: partition L into L^+ and L^-, such that L^+ contains all non-negative numbers from L, and L^- contains the rest. Then we sort the lists L^+ and L^-. Next, a^+ and a^- are obtained by summing up the numbers in L^+ and L^- in increasing order of magnitudes, respectively. The last step is to sum up a^+ and a^- to obtain the numerical sum of numbers in L.

There are some drawbacks to this heuristic. First of all, we do not have a warranty telling how good this method is. Especially we do not know whether there this heuristic may yield a worse result for some configuration. Secondly, this heuristic increases the time cost of summing up n numbers from $O(n)$ to $O(n \log n)$ arithmetic operations because of the sorting. Therefore it is used only in the implementation of Solver-Multi-Update so it does not increase the overall big-O complexity, i.e., used only for non-bottleneck steps.

There are sophisticated algorithms to sum up n floating numbers in $O(n)$ time with proven accuracy, see [28] and [29]. Though accurate summation is not the most essential part of our project (we are currently only using the simple heuristic), we should mention these sophisticated algorithms.

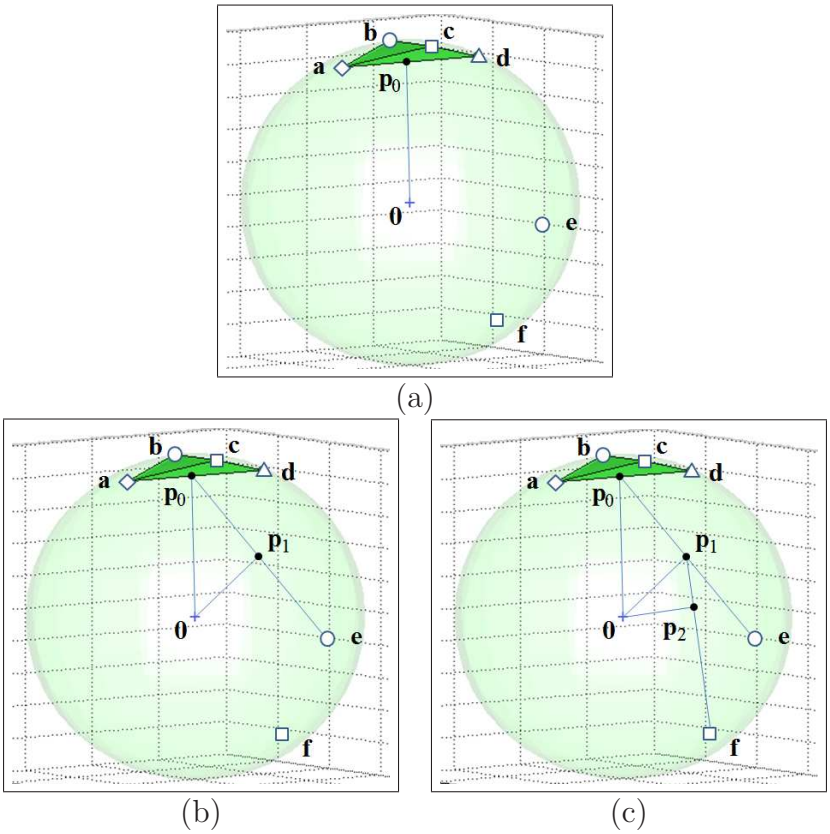


Figure 4.1: A sample iteration for Solver-Multi-Update

The shapes ◯, □, △ and ◊ stand for different colours. (a), (b) and (c) show different states of the algorithm in the same iteration, and the simplex in each of them represents the initial selection of T. Only the points involved in this iteration, i.e., the points in T or to be inserted to T, are displayed. $\mathbf{p}_0$ denotes the convergence indicator $\mathbf{p}$ at the beginning of the iteration. $\mathbf{p}_1$ and $\mathbf{p}_2$ are the values of $\mathbf{p}$ after incrementally replacing points in T. One can reasonability assume this figure illustrates the first iteration as no other initial T that leads to the state (a) using Solver-Multi-Update was found.

(a) the algorithm starts with $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ where $\mathbf{b}$ and $\mathbf{c}$ are to be replaced because $\mathbf{p}_0 \in \text{conv}(\mathbf{a}, \mathbf{d})$;

(b) $\mathbf{b}$ has been replaced with $\mathbf{e}$ and T becomes $\{\mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{e}\}$; $\mathbf{p}_1$ is the projection of $\mathbf{0}$ on the line segment $[\mathbf{p}_0, \mathbf{e}]$;

(c) $\mathbf{c}$ has been replaced with $\mathbf{f}$ and T becomes $\{\mathbf{a}, \mathbf{d}, \mathbf{e}, \mathbf{f}\}$; $\mathbf{p}_2$ is the projection of $\mathbf{0}$ on the line segment $[\mathbf{p}_1, \mathbf{f}]$; because $\|\mathbf{p}_2\| < \|\mathbf{p}_1\| < \|\mathbf{p}_0\|$ (triangular inequalities) and $\mathbf{p}_1, \mathbf{p}_2 \in \text{conv}(\mathbf{a}, \mathbf{d}, \mathbf{e}, \mathbf{f})$, $\text{conv}(\mathbf{a}, \mathbf{d}, \mathbf{e}, \mathbf{f})$ is closer to $\mathbf{0}$ than $\text{conv}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$.

Chapter 5

Two Alternative Algorithms

In this chapter we introduce alternative algorithms for the *Colourful Feasibility Problem* (**CFP**). Solver-Max-Volume using effective but simple geometric heuristic is implemented to compare the performance with other algorithms, and Solver-Random-Pick is primarily used to check the density of feasible solutions, i.e., the number of colourful simplices containing $\mathbf{0}$.

5.1 Volume of a Simplex

The *volume* is the amount of space an object occupies and is applied to 3-dimensional space. The generalization of *volume* to higher dimensions is called *content* or *hyper volume* but we simply use the term *volume*. If $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$, the volume of the d-dimensional simplex $\text{conv}(T)$ can be computed by the classic, see O'Rourke [20], formula:

$$\text{vol}\left(\text{conv}(T)\right) = \frac{\left|\det\left(\left[\begin{array}{ccc} (\mathbf{t}_1 - \mathbf{t}_{d+1}) & ... & (\mathbf{t}_d - \mathbf{t}_{d+1}) \end{array}\right]\right)\right|}{d!}. \tag{5.1.1}$$

5.2 Max-Volume Algorithm

The algorithms in the previous chapters are based on the following general idea: Keep updating the vertices of a colourful simplex until it covers $\mathbf{0}$. They use a point $\mathbf{p}$ inside the colourful simplex to help finding new vertices. How about other rule to find new vertices?

A simple idea to start with: A subset B of $\mathbb{B}^d$ has a larger probability to cover $\mathbf{0}$ if its volume is larger, without considering the other factors such as the shape and continuity of B. Consider two subsets B_1 and B_2 of $\mathbb{B}^d$, and we are told that $\text{vol}(B_1) > \text{vol}(B_2)$ but nothing else, then we are asked to chose the subset that is more likely to cover $\mathbf{0}$. We chose B_1 because its probability to cover $\mathbf{0}$ is $\frac{\text{vol}(B_1)}{\text{vol}(\mathbb{B}^d)}$, which is larger than that of B_2. Based on this simple idea, we made a rule to select a vertex that generates a simplex of largest volume in each iteration, compared to other possible vertices. The algorithm Solver-Max-Volume is using this rule.

For simplices with all the vertices on $\mathbb{S}^d$, we can easily find small volume ones covering $\mathbf{0}$, as well as relatively larger ones that not covering $\mathbf{0}$. This means the rule can be misleading in some cases. However, we implemented Solver-Max-Volume using this rule and tested it in Chapter 9 with different input to evaluate its average performance.

Algorithm 5: Solver-Max-Volume

Input: $\mathbf{S}$
Output: T

```
1  begin
2      initialize T = {t_1, ..., t_{d+1}} such that t_i ∈ S_i for i = 1, ..., d+1
3      while 0 ∉ conv(T) do
4          if dim(T) = d then
5              I_candidate ← { i : 0 is separated from t_i by aff(T \ {t_i})}
6          else
7              Let I_candidate contain a random element in {1, ..., d+1}
8          v* ← 0
9          for i ∈ I_candidate do
               s ← argmin_{t∈S_i} dist(t, aff(T \ {t_i}))
10             T_{i,s} ← (T ∪ {s}) \ {t_i}
11             v ← vol(conv(T_{i,s}))
12             if v ≥ v* then
13                 i* ← i
14                 s* ← s
15                 v* ← v
16         t_{i*} ← s*
17 end
```

Proposition 5.2.1 *When* $\mathbf{0} \notin \text{conv}(T)$ *and* $\dim(T) = d$*, Solver-Max-Volume can always obtain* $I_{\text{candidate}} \neq \emptyset$ *if the input is a* **CCFP**.

Proof: results *2* and *5* of Proposition 3.2.4 shows that $I_{\text{candidate}} \neq \emptyset$, and Proposition 3.2.5 shows that $I_{\text{candidate}}$ can be computed. □

The time cost of each outer-iteration in Solver-Max-Volume is up to $O(d^4)$, because $I_{\text{candidate}}$ can contain no more than $d+1$ colours and computing the volume for each colour (line 12 of the Solver-Max-Volume pseudocode) costs $O(d^3)$ time. One advantage of Solver-Max-Volume: no accumulation of numerical er-

ror (see Subsection 4.2.1) between each outer-iterations as for each iteration it computes the simplex volumes base on point coordinates but not computed results from the previous outer-iterations. Solver-Max-Volume can solve most of the **CCFP** cases during the test. However, it has some drawbacks:

- the idea of using simplex volume is not relevant for degenerate configurations (zero volumes);
- there is no convergence proof even for general position **CCFP**.

The *Symbolic Math Toolbox* under *MATLAB* was used to verify, using exact computation, that Solver-Max-Volume cycles for the input shown in Example 5.2.1 from Deza, et al. [11].

Example 5.2.1 (A 4D cycling example for Solver-Max-Volume)

points in S_1*:*

$$\begin{bmatrix} 7/52 \\ 1/176 \\ 4/29 \\ -\frac{\sqrt{4238906046}}{66352} \end{bmatrix} \begin{bmatrix} 1/89 \\ -8/65 \\ 1/961 \\ \frac{\sqrt{30434652805951}}{5559385} \end{bmatrix} \begin{bmatrix} -1/60 \\ 5/49 \\ -8/191 \\ \frac{\sqrt{11360296502737439}}{107254140} \end{bmatrix} \begin{bmatrix} -1/28 \\ 6/35 \\ 1/40 \\ -\frac{\sqrt{69789743}}{31640} \end{bmatrix}$$
$$\begin{bmatrix} 4/127 \\ 9/118 \\ -1/75 \\ -\frac{7\sqrt{25600756871}}{1123950} \end{bmatrix}$$

points in S_2*:*

$$\begin{bmatrix} 3/85 \\ -1/67 \\ 1/173 \\ \frac{\sqrt{29008089867051134}}{170445655} \end{bmatrix} \begin{bmatrix} -5/71 \\ 1/10 \\ -2/101 \\ -\frac{\sqrt{5063381959}}{71710} \end{bmatrix} \begin{bmatrix} 8/45 \\ -38/155 \\ 1/95 \\ \frac{2\sqrt{159502559}}{26505} \end{bmatrix} \begin{bmatrix} 3/88 \\ -2/131 \\ 3/53 \\ \frac{5\sqrt{14863381455}}{610984} \end{bmatrix}$$
$$\begin{bmatrix} -1/114 \\ -24/185 \\ 7/85 \\ \frac{\sqrt{125498719055}}{358530} \end{bmatrix}$$

points in S_3*:*

$$\begin{bmatrix} -3/77 \\ -3/20 \\ -2/71 \\ -\frac{\sqrt{470161115387}}{694309} \end{bmatrix} \begin{bmatrix} 4/141 \\ -4/63 \\ -3/173 \\ -\frac{8\sqrt{122080034994545}}{88619769} \end{bmatrix} \begin{bmatrix} 3/22 \\ -3/17 \\ -5/79 \\ -\frac{\sqrt{826050579}}{29546} \end{bmatrix} \begin{bmatrix} 16/111 \\ 5/29 \\ -1/210 \\ \frac{-\sqrt{48208184671}}{225330} \end{bmatrix}$$
$$\begin{bmatrix} -3/46 \\ 3/47 \\ 1/33 \\ \frac{\sqrt{5043188147}}{71346} \end{bmatrix}$$

points in S_4*:*

$$\begin{bmatrix} 1/59 \\ 1/29 \\ 3/56 \\ \frac{25\sqrt{14625287}}{95816} \end{bmatrix} \begin{bmatrix} 6/151 \\ -1/122 \\ 1/536 \\ \frac{\sqrt{554855708771634695}}{745501496} \end{bmatrix} \begin{bmatrix} 8/45 \\ -7/32 \\ 8/97 \\ \frac{\sqrt{17827555751}}{139680} \end{bmatrix} \begin{bmatrix} -3/29 \\ 4/43 \\ -1/14 \\ -\frac{\sqrt{297327743}}{17458} \end{bmatrix} \begin{bmatrix} 11/76 \\ -1/8 \\ 9/59 \\ \frac{\sqrt{75612155}}{8968} \end{bmatrix}$$

points in S_5*:*

$$\begin{bmatrix} 1/167 \\ 1/241 \\ 1/53 \\ -\frac{5\sqrt{1201121068645021462891}}{173320847963} \end{bmatrix} \begin{bmatrix} 3/43 \\ -1/244 \\ 2/9 \\ -\frac{\sqrt{8432767415}}{94428} \end{bmatrix} \begin{bmatrix} 11/52 \\ -5/134 \\ 13/142 \\ -\frac{\sqrt{57852799351}}{247364} \end{bmatrix} \begin{bmatrix} -19/65 \\ 2/129 \\ 1/4386 \\ -\frac{\sqrt{74312211919}}{285090} \end{bmatrix}$$
$$\begin{bmatrix} -3/100 \\ 1/62 \\ -4/73 \\ \frac{\sqrt{50998516979}}{226300} \end{bmatrix}$$

The initial simplex is taken to be $(1, 1, 1, 1, 1)$*, i.e., the first point of each colour. The algorithm proceeds to visit simplices* $(1, 1, 4, 1, 1)$*,* $(3, 1, 4, 1, 1)$*,* $(3, 1, 4, 3, 1)$*,* $(3, 1, 1, 3, 1)$ *and* $(1, 1, 1, 3, 1)$ *before returning to the original simplex and repeating. At steps one, three and five, there are two candidate colours for pivoting, the candidates that are not chosen for pivoting are* 1*,* 3 *and* 4 *respectively. For the even numbered steps there is a single candidate colour for pivoting.*

□

5.3 Random-Picking Algorithm

We implemented a simple guess-and-check algorithm where we sample colourful simplices randomly until we find one that covers **0**. This algorithm can be used for general **CFP**.

Algorithm 6: Solver-Random-Pick

Input: **S**
Output: T
begin
  initialize $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$ such that $\mathbf{t}_i \in S_i$ for $i = 1, ..., d+1$
  while $\mathbf{0} \notin \text{conv}(T)$ **do**
    for $i \in \{1, ..., d+1\}$ **do**
      $\mathbf{t}_i \sim S_i$
end

We would not expect Solver-Random-Pick to find the solution efficiently in general cases. However, as discussed in [10], feasible solutions to a given **CFP** (or its special class, **CCFP**) case may not be all that rare, and in some cases can be quite frequent. Since guessing and checking are relatively fast operations, it is worth considering the possibility that this naive algorithm may perform well in special cases or low dimension.

Solver-Random-Pick has another special usage: to detect the average density of feasible solution of a class of cases. We implement random case generators, both for **CCFP** and general **CFP** cases. The output from some of these generators should have restricted numbers of feasible solutions. Solver-Random-Pick can be used to test these generators, see Chapter 8.

Chapter 6

Enumeration with geometric Heuristic for General Cases

The algorithms we discussed in the previous chapters, except Solver-Random-Pick, are all designed for the *Colourful Core Feasibility Problem* (**CCFP**). However, **CCFP** is only a special class of the *Colourful Feasibility Problem* (**CFP**). In this chapter we propose an algorithm to solve the general **CFP**.

6.1 Enumeration Framework

For the **CFP** cases with $d+1$ colours and $d+1$ points in each colour, a simple generic enumeration algorithm is to enumerate and test each $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$, until we find a T such that $\mathbf{0} \in \text{conv}(T)$, or all possible T are tested. Note that $\mathbf{t}_i \in S_i$ for each $i = 1, ..., d+1$.

For an infeasible **CFP** case, all the $(d+1)^{d+1}$ combinations have to be tested. We currently need $O\left((d+1)^{d+1}d^3\right)$ arithmetic operations to do that. For a feasible **CFP** case, the number of colourful simplices covering $\mathbf{0}$ can also be as low as 1 and that many arithmetic operations are still expected. For a **CCFP** case, Bárány and Matoušek [5] and Stephen and Thomas [25]

independently proved that there are at least $O(d^2)$ feasible solutions. In Deza, et al. [11], a kind of **CCFP** cases only having $d^2 + 1$ feasible solutions was proposed, see Chapter 10. Therefore, we expect

$$O\left(\frac{(d+1)^{d+1}}{O(d^2)}d^3\right) = O(d^{d+2}) \tag{6.1.1}$$

arithmetic operations to get a solution if the colourful simplices are tested in arbitrary order. Hence, the performance of enumeration algorithm without well-selected enumeration order are not acceptable.

On the other hand, some geometric heuristic can help. For example, if a simplex has its vertices spread away from each other, then intuitively it has higher chance to cover the $\mathbf{0}$. However, this kind of heuristic does not seem to guarantee convergence.

We combine the enumeration and geometric heuristic together to obtain the following generic algorithm.

For a **CFP** case represented by $\mathbf{S}$, we call each set $T = \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$ with $\mathbf{t}_i \in S_i$ a *transverse*. This definition of transverse is not exactly the same as Bárány and Matoušek [5], but has a similar flavor. Let $\mathbf{V}$ be a set of colourful points, we use $\tau(\mathbf{V})$ to denote the set of transverses generated by $\mathbf{V}$, which means, the set of transverses with all points from $\mathbf{V}$. For example, $\tau(\mathbf{S})$ is a set of $(d+1)^{d+1}$ transverses. We use V_i to notate $\mathbf{V} \cap S_i$. If $V_i = \emptyset$ for any i, then $\tau(\mathbf{V}) = \emptyset$.

Algorithm 7: Solver-Enum

Input: $\mathbf{S}, \Gamma$
Output: T

```
1  begin
2      W ← ∅
3      U ← S
4      while U ≠ ∅ do
5          {V, U} ← Γ(U)
6          for T ∈ τ(V) do
7              if 0 ∈ conv(T) then return
8          for T ∈ τ(W ∪ V) \ (τ(W) ∪ τ(V)) do
9              if 0 ∈ conv(T) then return
10         W ← W ∪ V
11 end
```

We can consider Γ as a subroutine using some algorithm to select points that are "likely" to generate a transverse T such that $\mathbf{0} \in \text{conv}(T)$. Let its input/output interface be

$$\left(\mathbf{V}, \mathbf{U}^{\text{out}}\right) \leftarrow \Gamma(\mathbf{U}). \tag{6.1.2}$$

For the convergence and effectiveness of Solver-Enum, we require Γ to satisfy the following conditions:

- $\mathbf{V} \neq \emptyset$ if $\mathbf{U} \neq \emptyset$;
- $\mathbf{U}^{\text{out}} = \mathbf{U} \setminus \mathbf{V}$;
- always terminates.

At lines 7 and 9, Solver-Enum tests whether $\mathbf{0} \in \text{conv}(T)$. Solver-Enum is an algorithm not missing any transverse T and Proposition 6.1.1 gives a loop-invariant to support this claim.

Proposition 6.1.1 *For the execution of any outer-iteration of Solver-Enum, if a transverse T satisfying $\mathbf{0} \in \text{conv}(T)$ is not found, then right before and right after the execution, it holds that all transverses in $\tau(\mathbf{W})$ are tested.*

Proof: We can be prove this proposition by induction. Right before the first outer-iteration, $\mathbf{W} = \emptyset$ and $\tau(\mathbf{W}) = \emptyset$, so the proposition holds.

For an arbitrary outer-iteration. Let $\overline{\mathbf{W}}$ and $\underline{\mathbf{W}}$ be the values of $\mathbf{W}$ right before and right after execution, respectively. Assume that all transverses in $\tau\left(\overline{\mathbf{W}}\right)$ are tested right before the execution. During the execution all transverses in $\tau(\mathbf{V})$ and $\tau\left(\overline{\mathbf{W}} \cup \mathbf{V}\right) \setminus \left(\tau(\overline{\mathbf{W}}) \cup \tau(\mathbf{V})\right)$ are tested in two different inner-loops, respectively. Merging these three sets together we find that all transverses in $\tau(\underline{\mathbf{W}}) = \tau(\overline{\mathbf{W}} \cup \mathbf{V})$ are tested. □

Since we assume that the output $\mathbf{V}$ of Γ is nonempty if the input $\mathbf{U}$ is not, $\mathbf{W}$ will become the same as $\mathbf{S}$ if there is no T satisfying $\mathbf{0} \in \text{conv}(T)$ is found. Proposition 6.1.1 states that no transverse of $\mathbf{W}$ will be missed. Hence, we can obtain the convergence of Solver-Enum.

Another loop-invariant of the outer-loop of Solver-Enum: $\mathbf{U} \cup \mathbf{W} = \mathbf{S}$ and $\mathbf{U} \cap \mathbf{W} = \emptyset$. Solver-Enum works using $\mathbf{V}$ as an intermediate set and keeps moving points from $\mathbf{U}$ to $\mathbf{W}$, until a valid transverse T is found, or $\mathbf{U} = \emptyset$. Each transverse T is tested at most once.

Proposition 6.1.2 *If a for-loop runs one and only one execution for each element of its given iterator set, then Solver-Enum tests each transverse T at most once.*

Proof: Assume there is a transverse T tested twice in the jth and the kth outer-iterations, respectively. We can set $j \leq k$ without loss of generality. We

will contradict both cases $j = k$ and $j < k$.

For $j = k$, there are two subcases. The first is that two tests happen in the same inner-loop, which is contradicted by the assumption of this proposition. The second is that two tests happen in the two different inner-loops, and this is contradicted by the fact that $\tau(\mathbf{V}) \cap (\tau(\mathbf{W} \cup \mathbf{V}) \setminus (\tau(\mathbf{W}) \cup \tau(\mathbf{V}))) = \emptyset$.

For the $j < k$, let $\underline{\mathbf{W}}_j$ be the value of $\mathbf{W}$ right after the jth outer-iteration, $\overline{\mathbf{W}}_k$ the value of $\mathbf{W}$ right before the kth outer-iteration, and $\mathbf{V}_k$ the value of $\mathbf{V}$ found in the kth outer-iteration. The set of transverses tested in the kth outer-iteration is $\tau\left(\overline{\mathbf{W}}_k \cup \mathbf{V}_k\right) \setminus \tau\left(\overline{\mathbf{W}}_k\right)$ which is impossible to have any overlap with $\tau\left(\underline{\mathbf{W}}_j\right) \subseteq \tau\left(\overline{\mathbf{W}}_k\right)$. □

6.2 Selecting Colourful Points by Stretching

Solver-Enum can be considered as a framework for enumeration algorithm because its subroutine Γ provides flexibility. In this section we propose a routine that can be used as Γ in Solver-Enum.

If a set of colourful points scatter far away from each other, then the total volume of colourful simplices generated by them is expected to be large, and have better chance to cover $\mathbf{0}$ than randomly picked points. Based on this intuitive idea we design the routine Gamma-Stretch to select $\mathbf{V} \subseteq \mathbf{U}$, such that:

- $\tau(\mathbf{V})$ is likely to contain T such that $\mathbf{0} \in \text{conv}(T)$;
- $|\tau(\mathbf{V})|$ is reasonably small such that the complexity of testing all the transverses in $\tau(\mathbf{V})$ is upper bounded by $O(2^d d^3)$ arithmetic operations, where $O(2^d)$ is the number of transverses and $O(d^3)$ is the cost of testing each transverse.

Algorithm 8: Gamma-Stretch

Input: $\mathbf{U}$
Output: $\mathbf{V}, \mathbf{U}^{\text{out}}$

1 **begin**
2 $I_{\text{empty}} \leftarrow \{ i : |U_i| = 0\}$
3 $I_{\text{singular}} \leftarrow \{ i : |U_i| = 1\}$
4 $I_{\text{multiple}} \leftarrow \{ i : |U_i| \geq 2\}$
5 $\mathbf{U}^{\text{out}} \leftarrow \mathbf{U}$; $\mathbf{V} \leftarrow \emptyset$
6 $k \leftarrow 0$; $\mathbf{p} \leftarrow \mathbf{0}$
7 **for** $i \in I_{\text{singular}}$ **do**
8 let $\mathbf{s}$ be the point in U_i^{out}
9 $\mathbf{t}_i \leftarrow \mathbf{s}$
10 move $\mathbf{s}$ from U_i^{out} to V_i
11 $\mathbf{p} \leftarrow \frac{k\mathbf{p}+\mathbf{s}}{k+1}$; $k \leftarrow k+1$
12 **for** $i \in I_{\text{multiple}}$ **do**
13 $\mathbf{s} \leftarrow \underset{\mathbf{t} \in U_i}{\text{argmin}}(\mathbf{t}^T\mathbf{p})$
14 $\mathbf{t}_i \leftarrow \mathbf{s}$
15 move $\mathbf{s}$ from U_i^{out} to V_i
16 $\mathbf{p} \leftarrow \frac{k\mathbf{p}+\mathbf{s}}{k+1}$; $k \leftarrow k+1$
17 **if** $I_{\text{empty}} \neq \emptyset$ **then return**
18 $T \leftarrow \{\mathbf{t}_1, ..., \mathbf{t}_{d+1}\}$
19 **if** $\dim(T) \neq d$ **then return**
20 $I_{\text{seperate}} \leftarrow \{ i \in I_{\text{multiple}} : \mathbf{0}$ and $\mathbf{t}_i$ are separated by $\text{aff}(T \setminus \{\mathbf{t}_i\})\}$
21 **for** $i \in I_{\text{seperate}}$ **do**
22 $\mathfrak{U}_i \leftarrow \{ \mathbf{t} \in U_i^{\text{out}} : \mathbf{t}$ is on the same side of $\text{aff}(T \setminus \{\mathbf{t}_i\})$ as $\mathbf{0}\}$
23 $\mathbf{s} \leftarrow \underset{\mathbf{t} \in \mathfrak{U}_i}{\text{argmin}} (\text{dist}(\text{aff}(T \setminus \{\mathbf{t}_i\}), \mathbf{t}))$
24 **if** $\text{dist}(\text{aff}(T \setminus \{\mathbf{t}_i\}), \mathbf{s}) > \text{dist}(\text{aff}(T \setminus \{\mathbf{t}_i\}), \mathbf{0})$ **then**
25 move $\mathbf{s}$ from U_i^{out} to V_i
26 **end**

At line 2-4, Gamma-Stretch partitions the colour indices into three sets. I_{empty} holds the colours that do not have any point in $\mathbf{U}$ to select; I_{singular} holds the colours that only have one point in $\mathbf{U}$; I_{multiple} holds the colours that have

multiple points in $\mathbf{U}$ to select.

Line 5 initializes the outputs $\mathbf{U}^{\text{out}}$ and $\mathbf{V}$.

Lines 6-16 select one point from each colours in $I_{single} \cup I_{\text{multiple}}$ and move these points into $\mathbf{V}$ one by one. $\mathbf{p}$ is a reference point kept inside the convex hull of the points that have been selected. At line 13, $\mathbf{p}$ is used to help choosing points from the colours in I_{multiple}. The idea behind line 13 is that the point with smaller dot product with $\mathbf{p}$ is expected to stay further away from the points that have already been moved into $\mathbf{V}$. The integer k is a counter used to update $\mathbf{p}$.

In case there is a colour that has no point in $\mathbf{U}$, Gamma-Stretch terminates at line 17. Otherwise, Gamma-Stretch will attempt to select more points from the colours that have extra points.

Line 19 terminates Gamma-Stretch if $\text{conv}(T)$ is degenerate, because in that case I_{seperate} will be empty at line 20 even if the algorithm continues to execute.

Lines 20-25 attempt to select more points. They find all the facets of $\text{conv}(T)$ contained in an affine hyperplane separating $\mathbf{0}$ from $\text{conv}(T)$, and attempt to select one point for each of their corresponding colours. Each index i in I_{seperate} also indices the affine hyper plane $\text{aff}(T \setminus \{\mathbf{t}_i\})$. Any point $\mathbf{s}$ which is on the same side of $\text{aff}(T \setminus \{\mathbf{t}_i\})$ as $\mathbf{0}$ can satisfy the condition that the volume of $\text{conv}(T) \cap \text{conv}((T \setminus \{\mathbf{t}_i\}) \cup \{\mathbf{s}\})$ is 0. Selecting such an $\mathbf{s}$ is the heuristic to enlarge the total volume covered by the colourful simplices generated by $\mathbf{V}$.

It may be interesting to see for a d-dimensional case how many points, at most, could Gamma-Stretch select. Following is a tight bound obtained by normalizing all points in $\mathbf{S}$ to $\mathbb{S}^d$.

Proposition 6.2.1 *When T is in general position, Gamma-Stretch satisfies* $|I_{\text{seperate}}| \leq d-1$.

Proof: We can denote $H_i = \text{aff}(T \setminus \{\mathbf{t}_i\})$ for convenience. To prove $|I_{\text{seperate}}| \leq d-1$, it is sufficient to prove there are at least two i such that $\mathbf{t}_i$ and $\mathbf{0}$ are at the same side of H_i.

Now let's find the first i. Let

$$\begin{bmatrix} \mathbf{y}_1^T & b_1 \\ \dots & \dots \\ \mathbf{y}_{d+1}^T & b_{d+1} \end{bmatrix} = \begin{bmatrix} \mathbf{t}_1 & \dots & \mathbf{t}_{d+1} \\ 1 & \dots & 1 \end{bmatrix}^{-1}, \tag{6.2.3}$$

where each $\mathbf{y}_i$ is a d-dimensional vector. As for Proposition 3.2.5, for $1 \leq i \leq d+1$ we have:

1. $\mathbf{y}_i^T \mathbf{t}_i = 1 - b_i$;
2. $\mathbf{y}_i^T \mathbf{t}_j = -b_i$ for $1 \leq j \neq i \leq d+1$.

Then according to result *2* of Proposition 3.2.4, $\mathbf{y}_i$ is the normal vector of H_i and $\mathbf{y}_i^T \mathbf{x} = -b_i$ is the equation of H_i. When $b_i \geq 0$, both $\mathbf{0}$ and $\mathbf{t}_i$ are at the same side of H_i, because both $1 - b_i$ and 0 are greater than or equal to $-b_i$. An index $1 \leq i' \leq d+1$ satisfying $b_{i'} \geq 0$ can always be found because $\sum_{i=1}^{d+1} b_i = 1$.

Assuming that $H_{i'}$ does not separate $\mathbf{0}$ and $\mathbf{t}_{i'}$, we can always find an index $i^* \neq i'$ such that H_{i^*} does not separate $\mathbf{0}$ and $\mathbf{t}_{i^*}$. Let

- $H_i^+ = \{\mathbf{x} : \mathbf{y}_i^T \mathbf{x} \geq -b_i\}$, and $H_i^- = \{\mathbf{x} : \mathbf{y}_i^T \mathbf{x} \leq -b_i\}$;
- $\mathcal{C}_{i'}^+ = \bigcap_{1 \leq i \leq d+1}^{i \neq i'} H_i^+$, and $\mathcal{C}_{i'}^- = \bigcap_{1 \leq i \leq d+1}^{i \neq i'} H_i^-$.

Then $\text{conv}(T) = \bigcap_{i=1}^{d+1} H_i^+$. $\mathcal{C}_{i'}^+$ and $\mathcal{C}_{i'}^-$ are the translated polyhedral cones such that:

- $\text{conv}(T) \subset \mathcal{C}_{i'}^+$;

- $\mathcal{C}_{i'}^{+} \cap \mathcal{C}_{i'}^{-} = \{\mathbf{t}_{i'}\}$;
- $\mathbf{0} \notin \mathcal{C}_{i'}^{-}$.

Because $\mathbf{0} \notin \mathcal{C}_{i'}^{-}$, the projection of $\mathbf{0}$ on $\mathcal{C}_{i'}^{+}$ is not on H_{i^*} for some $1 \leq i^* \neq i' \leq d+1$; that is, H_{i^*} does not separate $\mathbf{0}$ from $\text{conv}(T)$. □

6.3 Implementation

Zhang [33] implemented most of the algorithm Solver-Enum. This section describes some of the proposed techniques.

6.3.1 Colourful Sets Represented by Indexing

A real number is represented by a fixed size floating point arithmetic in our implementation, so the size of memory needed to store the input $\mathbf{S}$ is $O(d^3)$. To speed up the set manipulations (such as Γ in Section 6.1), and reduce the memory overhead of using subroutines that input or output colourful sets of points, our implementation shares the original coordinate data representing $\mathbf{S}$ in the scope of Solve-Enum, and uses an indexing method to represent colourful sets of points. The shared coordinate data is stored in a *MATLAB* matrix variable `Pts`, which is introduced in Subsection 3.5.1. A colourful set of points in $\mathbb{R}^d$ is represented by an array of length $d+1$. Each element of the array contains a list of column indices of `Pts`.

Example 6.3.1 (Representing a Colourful Set of Points)
Following is MATLAB struct array `U` *representing a colourful set of points. The represented colourful set has two points from the first colour, and the coordinates*

of these two points can be found in the first and the third columns of `Pts`*. The interpretations for the second and the third colours are similar.*

```
U(1).list=[1 3]
U(2).list=[4]
U(3).list=[8 9]
```

□

6.3.2 Enumerating Transverses of a Given Set

For a colourful set $\mathbf{V}$ of points in d-dimensional space, we use a sequence of $d+1$ integers to enumerated all the transverses. The enumeration starts with $(1, ..., 1)$ taking the first point of each V_i, and ends with $(|V_1|, ..., |V_{d+1}|)$ taking the last point of each V_i.

Example 6.3.2 (Enumerate the Transverses)

The colourful set of points in Example 6.3.1 is enumerated in the following way.

order	*enumerated sequence*	*represented column numbers*		
		1*st colour*	2*nd colour*	3*rd colour*
1	$(1,1,1)$	1	4	8
2	$(2,1,1)$	3	4	8
3	$(1,1,2)$	1	4	9
4	$(2,1,2)$	3	4	9

□

Our implementation takes $O(d)$ arithmetic operations to obtain the next enumerated sequence of integer from the previous one, while testing the colourful simplex corresponding to an enumerated sequence takes $O(d^3)$ arithmetic operations. Therefore the enumeration is not part of the time bottleneck.

6.3.3 Enumerating Transverses in $\tau(\mathbf{W} \cup \mathbf{V}) \backslash (\tau(\mathbf{W}) \cup \tau(\mathbf{V}))$

In Solver-Enum, we need a method to do this task: given two disjoint colourful sets of points $\mathbf{W}$ and $\mathbf{V}$, enumerate all the transverses in

$$\tau(\mathbf{W} \cup \mathbf{V}) \setminus (\tau(\mathbf{W}) \cup \tau(\mathbf{V})) . \tag{6.3.4}$$

Our method to do this is:

1. enumerate all the colourful subsets of $\mathbf{W} \cup \mathbf{V}$ in the format of $\mathbf{M} = \bigcup_{i=1}^{d+1} M_i$, where either $M_i = W_i$ or $M_i = V_i$ for each i, excluding the subsets $\mathbf{W}$ and $\mathbf{V}$ themselves;
2. for each enumerated $\mathbf{M}$, enumerate the transverses in $\tau(\mathbf{M})$.

We use a binary sequence to enumerate all the $\mathbf{M}$. We enumerate from $(1, 0, ..., 0)$ to $(0, 1, ..., 1)$. Each sequence corresponds to a $\mathbf{M}$, and the ith sequence element indicates whether $M_i = W_i$ or $M_i = V_i$. Our implementation takes $O(d)$ arithmetic operation to obtain the next binary sequence from the previous one, and this computation is not part of the time bottleneck.

Chapter 7

Optimization Approach

The *Colourful Feasibility Problem* (**CFP**) can be formulated as a *Nonconvex Quadratic Optimization Problem*. We believe the formulations and algorithms based on the optimization approach could be promising.

Combinatorial-geometric algorithms and optimization approaches are actually blended. For example, the semidefinite relaxation presented Section 7.2 needs to be tested by infeasible cases, and the generator of infeasible cases (Gen-Infeasible from Subsection 8.3.2) can use a combinatorial-geometric algorithm (Solver-Enum from Chapter 6) as a subroutine.

7.1 Nonconvex Quadratic Optimization Formulation

We can use the following formulation suggested by Polik [21] to solve a **CFP**.

Definition 7.1.1 (Quadratic Formulation of CFP)

$$\begin{array}{rl} \min & \mathbf{x}^T Q \mathbf{x} \\ \text{such that}: \quad P\mathbf{x} & = \mathbf{0} \\ \sum_{i=1}^{|\mathbf{S}|} x_i & = 1 \\ \mathbf{x} & \geq \mathbf{0}, \end{array} \tag{7.1.1}$$

where $\mathbf{x} = [x_1\, x_2\, \ldots\, x_{|\mathbf{S}|}]^T$ *is the vector of variables.* $Q \in I\!R^{|\mathbf{S}|\times|\mathbf{S}|}$ *is a symmetric matrix. Let* Q_{k_1,k_2} *denote the element at the* k_1*th row and the* k_2*th column of* Q *and* $Q_{k_1,k_2} = 1$ *if the* k_1*th and the* k_2*th points are in the same colour group, and* $Q_{k_1,k_2} = 0$ *otherwise. The matrix* $P \in I\!R^{d\times|\mathbf{S}|}$ *is formed by the coordinates of the point from* $\mathbf{S}$*.*

□

The formulation of Definition 7.1.1 is denoted by **QP**. For each **CFP** case represented by a colourful set of points $\mathbf{S}$ (it can also denote its corresponding **CFP** and the interpretation depends on the context), there exist a case $\mathcal{Q}$ of **QP** corresponding to it, such that we can solve $\mathbf{S}$ by solving $\mathcal{Q}$.

QP is a global optimization problem with nonconvex objective function over a polytope defined by linear equalities and inequalities. If we solve a case $\mathcal{Q}$ of **QP**, then we solve the corresponding case $\mathbf{S}$ of **CFP** in the following way:

1. in case $\mathcal{Q}$ is feasible, assume $\mathbf{x}_0$ is the global optimal solution found,
   (a) if $\mathbf{x}_0^T Q \mathbf{x}_0 = 0$, then the nonzero elements of $\mathbf{x}_0$ indicates the selected points from $\mathbf{S}$;
   (b) if $\mathbf{x}_0^T Q \mathbf{x}_0 > 0$, then $\mathbf{S}$ is infeasible;
2. in case $\mathcal{Q}$ is infeasible, then $\mathbf{S}$ is infeasible.

However, there is no polynomial time algorithm under deterministic Turing Machine model (therefore nor in the real arithmetic model we are using) to

solve general Nonconvex Quadratic Optimization Problem unless P=NP, see Pardalos and Vavasis [23]. We do not know whether **QP** is a special class of Nonconvex Quadratic Optimization Problem that has a polynomial time algorithm to solve.

Proposition 7.1.1 *Let $\mathcal{Q}$ be a case of* **QP**. *If $\mathbf{x}_0$ is feasible for $\mathcal{Q}$ with objective value $\mathbf{x}_0^T Q \mathbf{x}_0 = 0$, then $\mathbf{x}_0$ is a global optimal solution of $\mathcal{Q}$.*

Proof: Because all elements of the variable $\mathbf{x}$ and the matrix Q are nonnegative, $\mathbf{x}^T Q \mathbf{x}$ cannot have negative value. Therefore the proposition is true. □

According to Proposition 7.1.1, the criteria $\mathbf{x}^T Q \mathbf{x} = 0$ can be used as the certificate of a global optimal solution, which means that any $\mathbf{x}$ satisfying this criteria is a global optimal solution. A heuristic method to solve **QP**: Keep finding the local optimal solutions until getting a solution with zero objective value. Some Interior Point Method can find a local optimal solution in ε-precision in a number of arithmetic operations polynomial to d and $\frac{1}{\varepsilon}$, see Ye [32]. Invoking such an Interior Point Method with different initial solutions may lead to different local optimal solutions. Hence, we can use Interior Point Method to implement such a heuristic method to solve **QP** and hope that it performs well. However, if the case $\mathcal{Q}$ is feasible but the corresponding **CFP** case is not, the heuristic does not have a certificate to decide whether a local optimal solution is globally optimal since the global optimal objective value is greater than zero. This heuristic method is not implemented in this thesis.

7.2 Positive Semidefinite Relaxation

In this section we relax **QP** to obtain a *semidefinite optimization problem*, which is primarily used to detect infeasible **CFP** cases. Both Chapter 6 and Section 7.1 introduce an algorithm that has the worst performance (long execution time and non-terminating execution, respectively) when the input **CFP** case is infeasible, so a polynomial time method to detect some infeasible input seems worthwhile, and the semidefinite relaxation provides one.

7.2.1 Semidefinite Optimization

A square matrix A is a *positive semidefinite matrix* if $\mathbf{v}^T A\mathbf{v} \geq 0$ for all vectors $\mathbf{v}$ of proper dimension. We use $A \succeq 0$ to express that A is a positive semidefinite matrix.

A *Semidefinite Optimization Problem* is an optimization problem that has a symmetric positive semidefinite matrix as its variable.

Definition 7.2.1 (Semidefinite Optimization Problem)

$$\min_X \{\, C \bullet X \;:\; X \succeq 0 \;\; \textit{and } A_i \bullet X = b_i \;\textit{for } i = 1, ..., m \,\}, \tag{7.2.2}$$

where the matrix $X \in \mathbb{R}^{n\times n}$ is symmetric and contains the variables. The square matrices C, A_1, ..., $A_m \in \mathbb{R}^{n\times n}$ and the scalars $b_1, \ldots, b_m \in \mathbb{R}$ are given. Further, the operator $\bullet$ denotes the inner product, i.e., the sum of all the elements of the elementwise product of the operands.

□

Semidefinite Optimization Problems are solvable in polynomial number of arithmetic operations, see for example Potra and Sheng [22].

7.2.2 Relaxation

Before obtaining the relaxation, we get the following formulation equivalent to **QP** as a transition step.

Definition 7.2.2 (Alternative Optimization Formulation of CFP)

$$\begin{array}{rrl} \min & Q \bullet X & \\ \text{s.t.} & P\mathbf{x} & = \mathbf{0} \\ & \sum_{i=1}^{|\mathbf{S}|} x_i & = 1 \\ & \mathbf{x} & \geq \mathbf{0} \\ & X & = \mathbf{x}\mathbf{x}^T, \end{array} \tag{7.2.3}$$

where $\mathbf{x} = [x_1\, x_2\, \ldots\, x_{|\mathbf{S}|}]^T$ *and the symmetric matrix* $X \in I\!R^{|\mathbf{S}|\times|\mathbf{S}|}$ *contain the variables, the notations* Q *and* P *are the same as those in* **QP**.

□

The following formulation is obtained by removing $X = \mathbf{x}\mathbf{x}^T$ from formulation (7.2.3) and adding some extra constraints.

Definition 7.2.3 (Semidefinite Relaxation of QP)

$$\begin{array}{rrll} \min & Q \bullet X & & \\ \text{s.t.} & P\mathbf{x} & = \mathbf{0} & \\ & \sum_{i=1}^{|\mathbf{S}|} x_i & = 1 & \\ & \mathbf{x} & \geq \mathbf{0} & \\ & \begin{bmatrix} 1 & \mathbf{x}^T \\ \mathbf{x} & X \end{bmatrix} & \succeq 0 & \\ & \sum_{1\leq k_1,k_2\leq|\mathbf{S}|} X_{k_1,k_2} & = 1 & \\ & X_{k_1,k_2} & = X_{k_2,k_1} & \text{for} \quad 1 \leq k_1 < k_2 \leq |\mathbf{S}| \\ & X_{k_1,k_2} & \geq 0 & \text{for} \quad 1 \leq k_1 < k_2 \leq |\mathbf{S}|\,. \end{array} \tag{7.2.4}$$

where $\mathbf{x} = [x_1\, x_2\, \ldots\, x_{|\mathbf{S}|}]^T$ *and the symmetric matrix* $X \in I\!R^{|\mathbf{S}|\times|\mathbf{S}|}$ *contain the variables, the notations* Q *and* P *are the same way as in* **QP**.

□

The formulation of Definition 7.2.3 is denoted by **SDP**. **SDP** is a Semidefinite Optimization Problem. Converting **SDP** to the standard format in Subsection 7.2.1 is not discussed in this thesis. Using the data Q and P from a **QP** case $\mathcal{Q}$, we can obtain the corresponding **SDP** case.

Proposition 7.2.1 *The optimization problem* **SDP** *is a relaxation of the optimization problem* **QP**.

Proof: Since formulation (7.2.3) is equivalent to **QP**, we only need to prove that all pairs $(\mathbf{x}, X)$ feasible for formulation (7.2.3) are also feasible for **SDP**. Specifically, we only need to prove that all pairs $(\mathbf{x}, X)$ satisfying the constraints of formulation (7.2.3) also satisfy the two conditions:

1. $\begin{bmatrix} 1 & \mathbf{x}^T \\ \mathbf{x} & X \end{bmatrix} \succeq 0$;
2. $\sum_{1\le k_1,k_2\le|\mathbf{S}|} X_{k_1,k_2} = 1$.

as the other constraints of **SDP** are obviously satisfied by $(\mathbf{x}, X)$.

Condition 1 can be proved by the definition of positive semidefinite matrix. For arbitrary $(1 + |\mathbf{S}|)$-dimensional vector $\begin{bmatrix} \alpha \\ \mathbf{v} \end{bmatrix}$, where α is a scaler and

$\mathbf{v}$ is a $|\mathbf{S}|$-dimensional vector, we have:

$$\begin{aligned} & \begin{bmatrix} \alpha & \mathbf{v}^T \end{bmatrix} \begin{bmatrix} 1 & \mathbf{x}^T \\ \mathbf{x} & X \end{bmatrix} \begin{bmatrix} \alpha \\ \mathbf{v} \end{bmatrix} \\ = \quad & \alpha^2 + 2\mathbf{v}^T\mathbf{x} + \mathbf{v}^T X\mathbf{v} \\ = \quad & \alpha^2 + 2\mathbf{v}^T\mathbf{x} + \mathbf{v}^T\mathbf{x}\mathbf{x}^T\mathbf{v} \\ = \quad & \left(\alpha + \mathbf{v}^T\mathbf{x}\right)^2 \\ \geq \quad & 0. \end{aligned}$$

For condition 2, because $\sum_{i=1}^{|\mathbf{S}|} x_i = 1$ and $X = \mathbf{x}\mathbf{x}^T$, we have,

$$\sum_{1 \leq k_1, k_2 \leq |\mathbf{S}|} X_{k_1,k_2} = \sum_{k=1}^{|\mathbf{S}|} X_{k,k} + \sum_{1 \leq k_1 < k_2 \leq |\mathbf{S}|} 2X_{k_1,k_2} = \left(\sum_{i=1}^{|\mathbf{S}|} x_i\right)^2 = 1.$$

□

We return to **CFP** by solving its corresponding **SDP**. Let $\mathbf{S}$ be a **CFP** case and $\mathcal{D}$ be the corresponding **SPD** case, then we have:

1. in case $\mathcal{D}$ is infeasible, $\mathbf{S}$ is infeasible;
2. in case $\mathcal{D}$ is feasible, assume $(\underline{\mathbf{x}}, \underline{X})$ is an optimal solution found,
   (a) if $Q \bullet \underline{X} > 0$, then $\mathbf{S}$ is infeasible;
   (b) if $Q \bullet \underline{X} = 0$ but $\underline{\mathbf{x}}^T Q \underline{\mathbf{x}} > 0$, then $(\underline{\mathbf{x}}, \underline{X})$ does not tell whether $\mathbf{S}$ is feasible;
   (c) if $\underline{\mathbf{x}}^T Q \underline{\mathbf{x}} = 0$, then the nonzero elements of $\underline{\mathbf{x}}$ indicates the selected points for a feasible solution of $\mathbf{S}$.

We obtain the following algorithm that can test the feasibility of **CFP** cases.

Algorithm 9: Solver-SDP-Relax

Input: **S**
Output: feasible, T

```
1  begin
2      let D be the corresponding SDP case of S)
3      let Q be the matrix for the objective function of D
4      solve D for the optimal solution (x, X)
5      if D is infeasible then
6          feasible ← false
7          T ← undefined
8      else if Q • X > 0 then
9          feasible ← false
10         T ← undefined
11     else if x^T Q x > 0 then
12         feasible ← undefined
13         T ← undefined
14     else
15         feasible ← true
16         pick the points from S indicated by nonzero elements of x and
           put them into the set T
17     return
18 end
```

7.3 Toolboxes Used for Implementation

The *MATLAB* implementation of Solver-SDP-Relax uses *SeDuMi 1.1R3* [27] library interfaced by the *YALMIP R20070302* [16] Toolbox to solve the **SDP** problems.

A typical section of *MATLAB* code in the implementation of Solver-SDP-Relax utilizing *SeDuMi 1.1R3* [27] and *YALMIP R20070302* [16] to formulate and solve the positive Semidefinite Optimization Problem is at the following.

```
133   % Formulate and solve the SDP.
134 - Q=[];
135 - for i=1:length(ColorPartition)
136 -     Q=blkdiag(Q,~eye(ColorPartition(i)));
137 - end
138 - xvar = sdpvar(NumPts,1);
139 - bigXvar = sdpvar(NumPts, NumPts, 'symmetric');
140 - F1 = set(Pts*xvar==b, 'x elements are cofficients to get b');
141 - F2 = set(sum(xvar)==1, 'x elements sum up to 1');
142 - F3 = set(xvar>=0, 'elements in x are non-negative');
143 - F4 = set([1 xvar'; xvar bigXvar]>=0, '[1 x; x bigX] is PSD');
144 - F5 = set(sum(sum(bigXvar))==1, 'bigX elements sum up to 1');
145 - F6 = set(bigXvar(:)>=0, 'bigX elements are non-negatives');
146 - F = F1+F2+F3+F4+F5+F6;
147 - solveroption = sdpsettings('solver','sedumi', 'verbose',0, ...
148                              'sedumi.stepdif',2, 'sedumi.sdp',1, ...
149                              'sedumi.cg.qprec', 1);
150 - diagnostics = solvesdp(F, sum(sum(Q.*bigXvar)), solveroption);
151 - x = double(xvar);
152 - bigX = double(bigXvar);
153   % initialized: diagnostics x
```

Figure 7.1: Sample *MATLAB* code that uses *YALMIP* to interface *SeDuMi*

Figure 7.1 is the screenshot of the code in the Solver-SDP-Relax implementation to formulate and solve the **SDP**. We assume that the variables `Pts`, `ColorPartition` and `NumPts` are properly defined before running this section of code. `Pts` and `ColorPartition` are as described in Subsection 3.5.1, while `NumPts` is a scalar variable telling the total number of points in **S**. Lines 134-137 construct the variable `Q` to represent the matrix Q in Definition 7.2.3. Lines 138 and 139 use the function `sdpvar` in *YALMIP* to define two objects `xvar` and `bigXvar` representing `x` and X in Definition 7.2.3. Lines 140-145 use the function `set` in *YALMIP* to define the objects `F1` to `F6` representing the constraints of Definition 7.2.3. Line 146 obtains the object `F` by combining `F1` to `F6`. Now

we have the three variables `xvar`, `bigXvar` and `F` together representing an instance of the formulation (7.2.4). Lines 147-149 uses the function `sdpsettings` from $YALMIP$ to define the object `solveroption` representing the options to *SeDuMi*, which is the actual solver of the formulated Semidefinite Optimization Problem. Line 150 uses the function `solvesdp` from *YALMIP* to solve the formulated instance of Semidefinite Optimization Problem. `xvar` and `bigXvar` are atomically assigned values representing the optimal solution for the formulated instance of Semidefinite Optimization Problem. Lines 151-153 extract the matrix variables `x` and `bigX` from `xvar` and `bigXvar` using the overloaded *MATLAB* function `double`.

From the code in Figure 7.1, we illustrate the data flow in Figure 7.2.

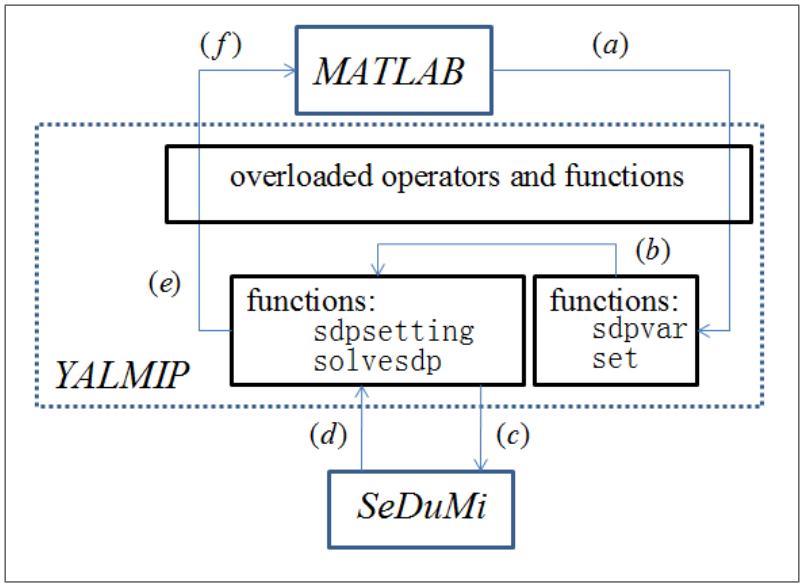


Figure 7.2: The data flow diagram of the Solver-SDP-Relax implementation
The implementation of Solver-SDP-Relax consists of three major components: *MATLAB*, *YALMIP* toolbox, and *SeDuMi*. *YALMIP* includes more utility functions than what we show in this figure, but we only illustrate what we used. The overloaded *MATLAB* operators and functions form the first layer of *YALMIP* communicating with *MATLAB*, they include `==`, `>=`, `+`, `*`, `sum` and `double`.

(*a*) the *MATLAB* variables `Pts`, `NumPts` and `Q` that together sufficiently defining the instance of Semidefinite Optimization Problem;

(*b*) the objects `xvar`, `bigXvar` and `F` used by *YALMIP* that defining the Semidefinite Optimization Problem;

(*c*) the input data to *SeDuMi* defining the Semidefinite Optimization Problem;

(*d*) the output data of *SeDumi* representing the result of solving the Semidefinite Optimization Problem;

(*e*) the objects `xvar`, `bigXvar` and `diagnostics` used by *YALMIP* to represent the result of solving the Semidefinite Optimization Problem;

(*f*) the variables `x` and `bigX` directly used by *MATLAB* to represent the result of solving the Semidefinite Optimization Problem.

Chapter 8

Random Case generation

To achieve a better understanding of the various algorithms performance, we produced a test suite of challenging *Colourful Feasibility Problem*s (**CFP** cases). This chapter introduce the algorithms, called *generators*, used to generate different random **CFP** cases.

8.1 Notations

This section introduces some notations used in this chapter.

8.1.1 Uniform Random Selection

We use the operator "$\sim$" to notate the process of selecting an element from a set randomly and uniformly. The set can be either finite or infinite, bounded or unbounded. In the chapter we will frequently use random selection within the generators.

Example 8.1.1 (Uniform Random Selection)
$i \sim \{1, 2, 3, 4\}$ *denotes the process of randomly selecting the value of variable* i *among* 1*,* 2*,* 3 *and* 4*. Each value has the chance of* 25% *to be selected.*

□

8.1.2 Universal Set of Colour Indices

In the random case generators we frequently use the random process of selecting one number from the set $\{1, ..., d+1\}$, where d is the number of dimensions of the space. To shorten the notations, we use the symbols I and J instead of $\{1, ..., d+1\}$. In this chapter, $I = J = \{1, ..., d+1\}$.

8.1.3 Notation for Colourful Points

We use **S** to denote a **CFP** case generated by the algorithms, and have the following settings:

1. $\mathbf{S} = \biguplus_{i=1}^{d+1} S_i$, where each S_i contains the points of an individual colour;
2. $S_i = \{\mathbf{s}_1^i, ..., \mathbf{s}_{d+1}^i\}$ for $1 \leq i \leq d+1$.

8.2 Colourful Core Feasibility Problem Generators

Several algorithms are specifically designed, in Chapters 3 to 5, for **CCFP**, which is a special class of **CFP**. To test these algorithms, we need to randomly generate **CCFP** cases. Therefore we designed several **CCFP** generators for unstructured random cases, ill-conditioned cases, and cases with restricted number of solutions.

8.2.1 Unstructured Random Cases

The first class of problems we consider are the unstructured random **CCFP** cases. We take $d+1$ points on $\mathbb{S}^d$ for each of the $d+1$ colours. The only

restriction required is that $\mathbf{0}$ is in the core. This is achieved by taking one of the points to a random convex combination of the antipodes of the other d points. We call this generator Gen-Core-Random. For generating points uniformly distributed on $\mathbb{S}^d$, we refer to Muller [18].

Algorithm 10: Gen-Core-Random

Input: d
Output: $\mathbf{S}$
1 **begin**
2 **for** $i \leftarrow 1$ **to** $d+1$ **do**
3 $j^* \sim J$
4 **for** $j \in J \setminus \{j^*\}$ **do** $\mathbf{s}^i_j \sim \mathbb{S}^d$
 $\mathbf{s}^i_{j^*} \sim \text{conv}(-\mathbf{s}^i_1, ..., -\mathbf{s}^i_{j^*-1}, -\mathbf{s}^i_{j^*+1}, ..., -\mathbf{s}^i_{d+1})$
 $\mathbf{s}^i_{j^*} \leftarrow \dfrac{\mathbf{s}^i_{j^*}}{\|\mathbf{s}^i_{j^*}\|}$
5
6 **end**

Statistically, this generator is always generating cases that all points are in general position. This is true for other generators introduced in this chapter, even though some generators such as Gen-Core-Tube generates ill-conditioned cases which can be considered almost degenerate.

8.2.2 Ill-conditioned Cases

Next, we consider ill-conditioned problems. We place d points of each colour on the spherical cap around the point $[0\,0\,...\,0\,1]^T$ or $[0\,0\,...\,0\quad -1]^T$ (points are concentrated towards the cap centers) and the final point of the colour in the opposite spherical cap, as a positive combination of the antipodes of the first d points. Since the points all lie in the tube around the final coordinate axis, we call it *tube generator* (Gen-Core-Tube) and it has two parameters. The first

parameter is the maximum angle $0 \leq \overline{\theta} \leq \frac{\pi}{2}$ between a chosen vector and the final coordinate axis. The second is β_{balanced}: a true/false parameter deciding the distribution of points. Gen-Core-Tube will randomly place either 1 or d points of each colour on a side of the tube when $\beta_{\text{balanced}} = \text{true}$, and force d points of each colour on one side when $\beta_{\text{balanced}} = \text{false}$.

Algorithm 11: Gen-Core-Tube

Input: d, $\overline{\theta}$, β_{balanced}
Output: $\mathbf{S}$

begin
  for $i \leftarrow 1$ **to** $d+1$ **do**
    $j^* \sim J$
    if $\beta_{\text{balanced}} = \text{true}$ **then** $u \sim \{-1, 1\}$ **else** $u \leftarrow 1$
    for $j \leftarrow J \setminus \{j^*\}$ **do**
      $\mathbf{s} \sim \mathbb{S}^{d-1}$
      $\theta \sim [0, \overline{\theta}]$
      $\mathbf{s}^i_j \leftarrow \begin{bmatrix} \sin(\theta)\mathbf{s} \\ \cos(\theta)u \end{bmatrix}$
    $\mathbf{s}^i_{j^*} \sim \text{conv}(-\mathbf{s}^i_1, ..., -\mathbf{s}^i_{j^*-1}, -\mathbf{s}^i_{j^*+1}, ..., -\mathbf{s}^i_{d+1})$
    $\mathbf{s}^i_{j^*} \leftarrow \frac{\mathbf{s}^i_{j^*}}{\|\mathbf{s}^i_{j^*}\|}$
end

8.2.3 Restricted Number of Solutions

We consider problems where we control the number of colourful simplices containing $\mathbf{0}$. It turns out that the number of simplices containing $\mathbf{0}$ in dimension d can be as low as quadratic in d, but not lower, see [5, 10, 25], or as high as $d^{d+1}+1$, see [10], which is more than one third of the total number of colourful simplices.

We might expect that the difficulty of a **CFP** increases as the number of solutions, i.e. colourful simplices containing $\mathbf{0}$, decreases, so we wrote three problem generators based on the constructions in [10]. The first, Gen-Core-High-Dense generates perturbed versions of the configuration with many solutions. These problems have $d^{d+1}+1$ of the $(d+1)^{d+1}$ simplices containing $\mathbf{0}$, many more than uniform random configurations, and we expect them to be quite easy to solve. The second, Gen-Core-Mid-Dense, generates configurations where the points of each colour are close to the vertices of a regular simplex on $\mathbb{S}^d$. There are $(d+1)!$ solutions corresponding to picking a different colour from each regular simplex vertex. Finally, we have Gen-Core-Low-Dense generating perturbed versions of the configuration from [10] which has only d^2+1 solutions. The generators Gen-Core-High-Dense, Gen-Core-Mid-Dense and Gen-Core-Low-Dense randomly permute the order the points appear within each colour.

Cases Having $d^{d+1}+1$ Solutions

The algorithm Gen-Core-High-Dense uses the vertices of a regular simplex as reference. For each colour, it generates d points clustered around a reference vertex and generates the last point as a random linear combination of the antipodes of the others. This algorithm takes a parameter r to control the maximum allowed distance between a generated colourful point and its reference vertex.

Algorithm 12: Gen-Core-High-Dense

Input: d, r
Output: $\mathbf{S}$
begin
  generate $\mathbf{p}_1, ..., \mathbf{p}_{d+1} \in I\!R^d$ such that $\text{conv}(\mathbf{p}_1, ..., \mathbf{p}_{d+1})$ is a regular simplex
  for $i \in I$ **do**
    $j^* \sim J$
    for $j \in J \setminus \{j^*\}$ **do**
      $\mathbf{t} \sim \mathbb{S}(r, \mathbf{p}_i)$
      $\mathbf{s}_j^i \leftarrow \dfrac{\mathbf{t}}{\|\mathbf{t}\|}$
    $\mathbf{s}_{j^*}^i \sim \text{conv}(-\mathbf{s}_1^i, ..., -\mathbf{s}_{j^*-1}^i, -\mathbf{s}_{j^*+1}^i, ..., -\mathbf{s}_{d+1}^i)$
    $\mathbf{s}_{j^*}^i \leftarrow \dfrac{\mathbf{s}_{j^*}^i}{\|\mathbf{s}_{j^*}^i\|}$
end

Some definitions and facts about regular simplex are utilized by Gen-Core-High-Dense. For a d-dimensional regular simplex:

1. the *center* is the elementwise average of its vertices;
2. the *center-vertex distance* and the *center-facet distance* are the distance from a vertex and a facet to its center, respectively;
3. the ratio of center-vertex distance to center-facet distance is d.

The following proposition supports the claim that if the input r to Gen-Core-High-Dense is not greater than $1/d$, then there will be $d^{d+1} + 1$ feasible solutions for each generated **CFP** case.

Proposition 8.2.1 *If the points* $\mathbf{p}_1, ..., \mathbf{p}_{d+1} \in \mathbb{S}^d$ *generate a regular simplex, and* $\mathbf{s}_i \in \mathbb{B}(1/d, \mathbf{p}_i)$ *for* $1 \leq i \leq d+1$, *then* $\mathbf{0} \in \text{conv}(\mathbf{s}_1, ..., \mathbf{s}_{d+1})$.

Proof: This proposition can be proved by induction. It is trivial for $d = 1$, and we will prove it for $d = k \geq 2$ assuming it is true for $d = k - 1$. Since all $\mathbf{p}_i$

are on the unit sphere $\mathbb{S}^d$ centered at the origin, we have $\mathbf{0} \in \text{conv}(\mathbf{p}_1, ..., \mathbf{p}_{d+1})$. Without loss of generality, we can assume $\mathbf{p}_{d+1} = [0 \, ... \, 0 \, 1]^T$, and $\mathbf{p}_1$ to $\mathbf{p}_d$ all have their last coordinates equal to $-1/d$, which implies $\mathbf{s}_1$ to $\mathbf{s}_d$ all have their last coordinates non-positive. We use the following setting:

1. $\mathbf{q}_i \in \mathbb{B}(1/d, \mathbf{p}_i)$ for $1 \leq i \leq d+1$;
2. $H_i = \text{aff}(\mathbf{q}_1, ..., \mathbf{q}_{i-1}, \mathbf{q}_{i+1}, ..., \mathbf{q}_{d+1})$ for $1 \leq i \leq d+1$;
3. $H = \{ \mathbf{x} : x_{d+1} = 0 \}$;
4. the projections of $\mathbf{p}_1$, ..., $\mathbf{p}_d$, $\mathbf{q}_1$, ..., $\mathbf{q}_d$, $\mathbb{B}(1/d, \mathbf{p}_1)$, ..., $\mathbb{B}(1/d, \mathbf{p}_d)$ onto H are $\mathbf{p}'_1$, ..., $\mathbf{p}'_d$, $\mathbf{q}'_1$, ..., $\mathbf{q}'_d$, $\mathbb{B}'_1$, ..., $\mathbb{B}'_d$, respectively.

H is a $(k-1)$-dimensional subspace; $\text{conv}(\mathbf{p}'_1, ..., \mathbf{p}'_d)$ is a $(k-1)$-dimensional regular simplex (embedded in H) with center-vertex distance equals to $\frac{\sqrt{k^2-1}}{k}$; $\mathbb{B}'_i$ is a $(k-1)$-dimensional ball around $\mathbf{p}'_i$ with radius $1/k$ for each i between 1 and k. If we scale up the center-vertex distance and the radius proportionally such that the center-vertex distance becomes 1, then the radius will become

$$\frac{1}{\sqrt{k^2-1}} = \frac{1}{\sqrt{(k+1)(k-1)}} < \frac{1}{k-1}. \tag{8.2.1}$$

Hence $\mathbf{0} \in \text{conv}(\mathbf{q}'_1, ..., \mathbf{q}'_d)$ according to our assumption on $(d-1)$-dimensional space. Let $\lambda_1, \ldots, \lambda_d$ be the convex combination coefficients such that $\mathbf{0} = \sum_{i=1}^{d} \lambda_i \mathbf{q}'_i$, then $\mathbf{q} = \sum_{i=1}^{d} \lambda_i \mathbf{q}_i$ is the only point in H_{d+1} with the first $d-1$ coordinates equal to 0. Since $\mathbf{q}_1$ to $\mathbf{q}_d$ all have their last coordinates non-positive, $\mathbf{q}$ has its last coordinate non-positive. Then let $\mathbf{q}_i$ move inside $\mathbb{B}(1/d, \mathbf{p}_i)$ for $1 \leq i \leq d$; $\mathbf{0}$ will never go across H_{d+1}. By symmetry, we have same property for H_1 to H_d. If we initialize $\mathbf{q}_i$ as $\mathbf{p}_i$ for $1 \leq i \leq d+1$ and move each $\mathbf{q}_i$ to the position of $\mathbf{s}_i$, then during this process $\mathbf{0} \in \text{conv}(\mathbf{q}_1, ..., \mathbf{q}_{d+1})$ holds because $\mathbf{0}$

does not go across any facet of $\text{conv}(\mathbf{q}_1, ..., \mathbf{q}_{d+1})$. $\mathbf{0} \in \text{conv}(\mathbf{s}_1, ..., \mathbf{s}_{d+1})$ follows and the proposition is true for $d = k$. □

By Proposition 8.2.1, as long as the input r is between 0 and $1/d$, all the d^{d+1} colourful simplices generated by clustered points covers $\mathbf{0}$. In addition, the colourful points not clustered together generate another simplex covering the $\mathbf{0}$. Hence, the number of feasible solutions is $d^{d+1} + 1$ for the **CFP** cases randomly generated by Gen-Core-High-Dense.

Cases Having $(d+1)!$ Solutions

Gen-Mid-Dense generates $d + 1$ points of different colours clustered near each vertex of a regular simplex. Similar to Gen-High-Dense, it also takes a parameter r to control how close should the points clustered together. By Proposition 8.2.1 we can easily notice that it generates **CCFP** cases with $(d + 1)!$ feasible solutions if $r \leq 1/d$.

Algorithm 13: Gen-Core-Mid-Dense

Input: d, r
Output: $\mathbf{S}$
1 **begin**
2 generate $\mathbf{p}_1, ..., \mathbf{p}_{d+1} \in I\!R^d$ such that $\text{conv}(\mathbf{p}_1, ..., \mathbf{p}_{d+1})$ is a regular simplex
3 **for** $i \in I$ **do**
4 **for** $j \in J$ **do**
5 $\mathbf{s}_j^i \sim \mathbb{S}(r, \mathbf{p}_j)$
6 $\mathbf{s}_j^i \leftarrow \frac{\mathbf{s}_j^i}{\|\mathbf{s}_j^i\|}$
7 **end**

Cases Having Few Solutions

Algorithm 14: Gen-Core-Low-Dense

Input: d, θ_{equ}, θ_{pole}
Output: **S**

begin
  generate $\mathbf{p}_1, ..., \mathbf{p}_d \in \mathbb{S}^{d-1}$ such that $\text{conv}(\mathbf{p}_1, ..., \mathbf{p}_d)$ is a regular simplex
  $i^* \sim I$
  $k \leftarrow 1$
  for $i \in I \setminus \{i^*\}$ in random order **do**
    $\{ j^D, j^U, j^* \} \sim J$
    $\mathbf{s}^i_{j^D} \leftarrow \begin{bmatrix} \cos(2\theta_{\text{equ}})\mathbf{p}_k \\ -\sin(2\theta_{\text{equ}}) \end{bmatrix}$; $\mathbf{s}^i_{j^U} \leftarrow \begin{bmatrix} -\cos(\theta_{\text{equ}})\mathbf{p}_k^T \\ \sin(\theta_{\text{equ}}) \end{bmatrix}$
    for $j \in J \setminus \{j^D, j^U, j^*\}$ **do** $\mathbf{t}_j \sim \mathbb{S}^{d-1}$
    $\mathbf{t}_{j^*} \sim \text{conv}(\{ -\mathbf{t}_j : j \neq j^D, j^U, j^* \})$
    $\mathbf{t}_{j^*} \leftarrow \dfrac{\mathbf{t}_{j^*}}{\|\mathbf{t}_{j^*}\|}$
    for $j \in J \setminus \{j^D, j^U\}$ **do** $\mathbf{s}^i_j \leftarrow \begin{bmatrix} \sin(\theta_{\text{pole}})\mathbf{t}_j \\ \cos(\theta_{\text{pole}}) \end{bmatrix}$
    $k \leftarrow k + 1$
  $\mathbf{t} \leftarrow \sum_{k=1}^{d-1} \begin{bmatrix} -\cos(2\theta_{\text{equ}})\mathbf{p}_k \\ \sin(2\theta_{\text{equ}}) \end{bmatrix}$; $\theta_{\text{safe}} \leftarrow \sin^{-1}\left(\dfrac{|t_d|}{\|\mathbf{t}\|}\right)$
  $j^* \sim J$
  for $j \in J \setminus \{j^*\}$ **do**
    $\mathbf{s}^{i^*}_j \sim \mathbb{S}^d \cap \{ \mathbf{x} : x_d = \sin(\theta_{\text{safe}}) \}$
    $\alpha \sim [0, 1]$
    if $\alpha > 1/d$ **then** $\lambda_j \sim [0, 1]$ **else** $\lambda_j \sim [0, d]$
  if $\lambda_j = 0$ for all $j \in J \setminus \{j^*\}$ **then** set one of the λ_i to 1
  $\mathbf{s}^{i^*}_{j^*} \leftarrow \sum_{j \in J}^{j \neq j^*} -\lambda_j \mathbf{s}^{i^*}_j$; $\mathbf{s}^{i^*}_{j^*} \leftarrow \dfrac{\mathbf{s}^{i^*}_{j^*}}{\|\mathbf{s}^{i^*}_{j^*}\|}$
end

Consider $\mathbb{S}^d$ as a high dimensional globe, such that $[0 \, ... \, 0 \, 1]^T$ is the *north pole*, and $\{\mathbf{x} : x_d = 0\} \cap \mathbb{S}^d$ is the *equator*. Gen-Core-Low-Dense generates the

points in d of the colours such that each colour has two points near equator (called *equator points*) and the rest around the north pole (called *pole points*), then generates the points of the last colour in areas giveng the **CCFP** case a low density of feasible solutions. The input data d must be at least 3 because of the special structures of generated cases; θ_{equ} and θ_{pole} should be very small angles (say, $0.05\pi/d$). θ_{equ} is the angle that equator points deviate from the equator, and θ_{pole} is the angle that pole points deviate from the north pole.

Lines 4-12 generate the points in d colours. Following properties for each i of these d colours make sure $\mathbf{0}$ is in the convex hull of its points:

1. $\mathbf{s}^i_{j_D}$ and $\mathbf{s}^i_{j_U}$ are the equator points;
2. $[0 \dots 0 \quad \frac{\sin(\theta_{\text{equ}})-\sin(2\theta_{\text{equ}})}{2}]^T \in [\mathbf{s}^i_{j_D}, \mathbf{s}^i_{j_U}]$;
3. the others are pole points, and $[0 \dots 0 \quad \cos(\theta_{\text{pole}})]^T$ is in the convex hull of them.

Lines 13-20 generate the points in the last colour. The following properties of this colour make the density of feasible solutions small:

1. d of the points only each generates one colourful simplex covering $\mathbf{0}$ with equator points;
2. the last point, $\mathbf{s}^{i^*}_{j^*}$, has significant chance to be close to the boundary of the cone generated by the antipodes of the other d points (one of these antipodes is expected to have relatively large coefficient by using α), and this prevents $\mathbf{s}^{i^*}_{j^*}$ from generating many colourful simplices covering $\mathbf{0}$.

Figure 8.1 illustrates an example of **CCFP** cases that can be generated by Gen-Core-Low-Dense.

The **CCFP** cases generated by Gen-Core-Low-Dense approximates the case having only $d^2 + 1$ solutions introduced in Chapter 10. The testing results of algorithm Solver-Random-Pick show that the number of solutions for

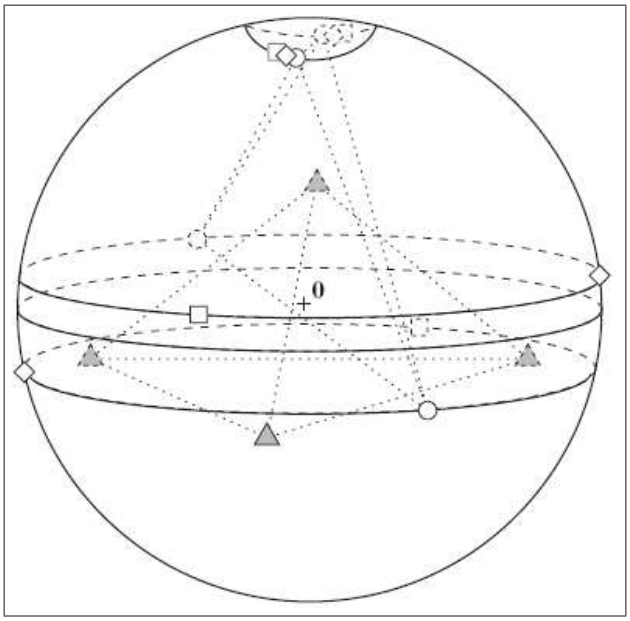


Figure 8.1: 3D example of **CCFP** with 10 solutions
The shapes ◯, □, ◊ and △ stand for different colours. The shaded △ are the antipodes of the colourful points.

the generated cases is low comparing to the cases from other generators (see Chapter 9).

8.3 General Cases

In the last section we introduced several algorithms to randomly generate **CCFP** cases. In this section we will introduce algorithms to generate both feasible and infeasible general random cases. They are used to test Solver-Enum and Solver-SDP-Relax, respectively.

8.3.1 Unstructured Cases

Gen-Random constructs unstructured random cases. β_{feasi} is a true/false parameter to decide whether the generated **CFP** case is forced to be feasible.

Algorithm 15: Gen-Random

Input: d, β_{feasi}
Output: **S**

```
1  begin
2      for i ← 1 to d + 1 do
3          for j ← 1 to d + 1 do
4              s_j^i ~ 𝕊^d
5      if β_feasi = true then
6          i* ~ I
7          k = 1
8          for i ∈ I \ {i*} do
9              j ~ J
10             t_k ← s_j^i
11             k ← k + 1
12         j ~ J
13         s_j^{i*} ~ conv(−t_1, ..., −t_d)
14         s_j^{i*} ← s_j^{i*} / ||s_j^{i*}||
15 end
```

8.3.2 Infeasible Cases

Gen-Infeasible keep constructing unstructured cases using Gen-Random until an infeasible case is obtained. It uses Solver-Enum to examine whether the constructed cases are feasible. The running time of Gen-Infeasible is expected to be long.

Algorithm 16: Gen-Infeasible

Input: d
Output: **S**

```
1 begin
2     β_feasi ← false
3     while true do
4         S ← Gen-Random(d, β_feasi)
5         T ← Solver-Enum(S)
6         if T = undefined then return
7 end
```

Chapter 9

Test Results

In this chapter, we describe the results of computational experiments in which we run the *Colourful Feasibility Problem* (**CFP**) solver algorithms to solve the problems generated by our random case generators. The running results are organized in tables, and the following abbreviations for different solver algorithms are used:

- BO1: Solver-Bárány-Onn-1 (Chapter 3),
- BO2: Solver-Bárány-Onn-2 (Chapter 3),
- MU: Solver-Multi-Update (Chapter 4),
- MV: Solver-Max-Volume (Chapter 5),
- RP: Solver-Random-Pick (Chapter 5),
- ES: Solver-Enum with Gamma-Stretch as its Γ subroutine (Chapter 6),
- SDPR: Solver-SDP-Relax (Chapter 7).

Section 9.1 tests BO1, BO2, MU, MV, RP and ES with different *Colourful Core Feasibility Problem* (**CCFP**) cases; Section 9.2 tests RP, ES and SDPR against general **CFP** cases. Note that we also test MV while a cycling example (see

Section 5.2) has been found, because cycling does not usually happen. All the solver algorithms are tested with the same set of generated cases when they are compared with each other in the same table.

9.1 Colourful Core Feasibility Problems

For each type of problems we have run tests of solver algorithms in dimensions $d = 3 \times 2^n$ for $n = 0, 1, 2, 3$ and so on. Dimension 3 is our starting point since some algorithms are simple and effective in dimension 2 (i.e., BO1 is the same as MU when $d = 2$). We believe this yields a reasonable sample of low, intermediate and high dimensional problems. We tested 1000 cases for the dimensions $d \leq 24$, and 50 cases for each dimension after that. The notation "N/A" in the tables indicate the part of test that could not be done due to long running time or numerical error.

The server used to perform tests in this section has eight 64-bit CPUs (*Dual Core AMD Opteron^{TM} Processor 885*). The clock frequency of each CPU core is 2.6 GHz, and the cache size of each CPU core is 1024 Kb. The server totally has 64 Gb random access memory. Despite the number of CPU cores in the server, we can safely assume that the tests have been run in sequence with a single CPU core, according to our observation on the CPU resource distribution.

The operating system is *OpenSUSE 10.2 Linux*. All the codes are implemented in *MATLAB* language and tested under *MATLAB Version 7.4.0.287 (R2007a)* with *Optimization Toolbox Version 3.1.1*.

9.1.1 Unstructured Cases

Table 9.1 presents the results of testing the algorithms with the **CCFP** cases generated by Gen-Core-Random (Subsection 8.2.1). BO2 takes more iterations than BO1 and MU but much less average time. The effect of oscillation phenomenon on BO2 is not obvious. BO2 is also the most numerically stable one that can solve all the 384-dimensional cases. The time and average number of iterations taken by MV increase rapidly between dimensions 96 and 192. ES has better performance than RP in average, but we can notice that in dimension 12 the maximum iteration number of ES is higher than RP. A possible **CCFP** case to make this happen is one that has almost all feasible colourful simplices collapsed (i.e., being almost degenerate) while having the infeasible colourful simplices relatively well-conditioned (i.e., having larger volumes).

d	Average (maximum) Number of Simplices					
	BO1	BO2	MU	MV	RP	ES
3	1.336 (4)	2.175 (10)	1.15 (3)	1.324 (4)	7.168 (41)	1.359 (6)
6	2.538 (6)	4.944 (12)	1.664 (4)	2.871 (8)	61.39 (430)	2.4 (75)
12	4.878 (10)	11.15 (20)	2.126 (5)	7.065 (17)	3965 (33567)	99.76 (79671)
24	8.879 (17)	24.94 (41)	2.578 (5)	18.907 (42)	N/A	9382 (1588195)
48	15.88 (20)	52.32 (61)	3.14 (5)	57.56 (90)	N/A	N/A
96	29.28 (36)	107.9 (122)	3.7 (5)	193.4 (301)	N/A	N/A
192	53.06 (60)	214.7 (230)	4.4 (6)	741.5 (1237)	N/A	N/A
384	N/A	433.6 (463)	N/A	N/A	N/A	N/A

d	Average (maximum) Running Time (seconds)		
	BO1	BO2	MU
3	5.428×10^{-3} (0.4905)	7.501×10^{-4} (0.01184)	4.401×10^{-3} (0.02021)
6	0.01071 (0.02614)	1.832×10^{-3} (3.911×10^{-3})	8.152×10^{-3} (0.01911)
12	0.03102 (0.06308)	5.789×10^{-3} (9.731×10^{-3})	0.01790 (0.03683)
24	0.1206 (0.2420)	0.02188 (0.03368)	0.05051 (0.08273)
48	0.7884 (1.036)	0.09667 (0.1101)	0.2347 (0.3093)
96	6.548 (8.399)	0.5131 (0.5739)	1.118 (1.508)
192	74.58 (92.48)	3.465 (3.762)	6.261 (8.545)
384	N/A	34.23 (36.38)	N/A

d	Average (maximum) Running Time (seconds)		
	MV	RP	ES
3	5.387×10^{-4} (0.01484)	5.360×10^{-4} (3.937×10^{-3})	1.821×10^{-3} (0.07486)
6	1.016×10^{-3} (1.954×10^{-3})	5.036×10^{-3} (0.03441)	3.327×10^{-3} (0.01373)
12	3.054×10^{-3} (6.167×10^{-3})	0.4898 (4.111)	0.01496 (5.224)
24	0.01520 (0.02955)	N/A	0.8926 (144.9)
48	0.1884 (0.2905)	N/A	N/A
96	3.906 (6.136)	N/A	N/A
192	188.6 (299.6)	N/A	N/A

Table 9.1: Test results on unstructured **CCFP** cases.

9.1.2 Ill-conditioned Cases

In this subsection we present the test results of the cases generated by Gen-Core-Tube (Subsection 8.2.2), which places colourful points only on two opposite spherical caps. The boundary of such a spherical cap is $\frac{\pi}{6}$ from its center (namely, $\overline{\theta} = \frac{\pi}{6}$), because this angle can make obviously different results from Gen-Core-Random without making too serious numerical problem.

Balanced Tube Cases

Table 9.2 presents the test results of the cases generated with $\beta_{\text{balanced}} = \text{true}$. Comparing to the results of Gen-Core-Random (Table 9.1), we notice that the cases from Gen-Core-Tube take more iterations and time to solve.

BO2 still has the best average performance and numerical stability, but we can find that it takes many iterations for some cases. Especially for one of the 3-dimensional cases it takes 788 iterations, while there are only $4^4 = 256$ different colourful simplices. Obviously BO2 oscillates for this case.

Comparing to Gen-Core-Random, the cases generated by Gen-Core-Tube with $\beta_{\text{balanced}} = \text{true}$ have roughly the same density of feasible solutions, because RP takes roughly the same average number of iterations to solve them. On another hand, these cases takes many iterations of ES.

d	Average (maximum) Number of Simplices					
	BO1	BO2	MU	MV	RP	ES
3	1.357 (4)	4.146 (788)	1.395 (4)	1.359 (4)	7.136 (64)	1.786 (17)
6	2.885 (8)	12.12 (267)	2.563 (6)	3.612 (12)	69.71 (552)	14.63 (633)
12	5.828 (11)	29.41 (436)	3.8 (8)	10.36 (28)	4061 (73073)	7792 (1509492)
24	11.14 (19)	60.15 (323)	4.920 (9)	31.34 (72)	N/A	N/A
48	20.96 (30)	121.8 (166)	5.76 (9)	108.8 (198)	N/A	N/A
96	36.56 (45)	234.3 (287)	7.02 (9)	407.6 (689)	N/A	N/A
192	N/A	463.1 (533)	N/A	3150 (7757)	N/A	N/A
384	N/A	918.5 (1028)	N/A	N/A	N/A	N/A

d	Average (maximum) Running Time (seconds)	
	BO1	BO2	MU
3	5.609×10^{-3} (0.415)	1.175×10^{-3} (0.1625)	5.426×10^{-3} (0.02917)
6	0.01298 (0.03686)	3.906×10^{-3} (0.07651)	0.01258 (0.02853)
12	0.03927 (0.08733)	0.01390 (0.1945)	0.03083 (0.06467)
24	0.1588 (0.2970)	0.04914 (0.2445)	0.09026 (0.1485)
48	0.9735 (1.444)	0.2088 (0.2832)	0.3519 (0.4786)
96	8.298 (11.27)	1.038 (1.259)	1.788 (2.145)
192	N/A	6.863 (7.850)	N/A
384	N/A	64.10 (74.08)	N/A

d	Average (maximum) Running Time (seconds)		
	MV	RP	ES
3	5.586×10^{-4} (0.01612)	5.336×10^{-4} (3.97×10^{-3})	2.020×10^{-3} (0.07435)
6	1.160×10^{-3} (2.453×10^{-3})	5.677×10^{-3} (0.04428)	5.079×10^{-3} (0.04455)
12	4.057×10^{-3} (9.166×10^{-3})	0.4979 (8.918)	0.5273 (97.08)
24	0.02401 (0.05503)	N/A	N/A
48	0.3664 (0.6816)	N/A	N/A
96	8.849 (15.08)	N/A	N/A
192	787.1 (2127)	N/A	N/A

Table 9.2: Test results on balanced tube **CCFP** cases.

Unbalanced Tube Cases

Table 9.3 presents the test results of the cases generated with $\beta_{\text{balanced}} = \text{false}$. While changing the parameter β_{balanced} to false, the generated cases takes more iterations in average to solve, especially for RP and ES. Increasing the average number of iterations taken by RP means decreasing the density of feasible solutions.

d	Average (maximum) Number of Simplices					
	BO1	BO2	MU	MV	RP	ES
3	1.462 (4)	3.695 (159)	1.513 (4)	1.482 (5)	8.742 (61)	1.924 (18)
6	3.458 (8)	11.46 (356)	2.682 (7)	4.181 (13)	152.39 (979)	16.22 (719)
12	7.49 (15)	29.10 (261)	4.048 (8)	13.45 (36)	3.695×10^4 (364162)	1.075×10^4 (177808)
24	16.23 (26)	69.98 (264)	5.671 (10)	48.22 (121)	N/A	N/A
48	32.3 (44)	150.2 (246)	6.8 (10)	155.0 (301)	N/A	N/A
96	59.96 (78)	298.9 (379)	8.52 (11)	577.1 (971)	N/A	N/A
192	N/A	625.8 (738)	N/A	1.279×10^4 (84148)	N/A	N/A

d	Average (maximum) Running Time (seconds)		
	BO1	BO2	MU
3	5.944×10^{-3} (0.4078)	1.083×10^{-3} (0.03311)	5.799×10^{-3} (0.01475)
6	0.01545 (0.03803)	0.003716 (0.09816)	0.01304 (0.03269)
12	0.05010 (0.1004)	0.01372 (0.1128)	0.03194 (0.05479)
24	0.2350 (0.4052)	0.05623 (0.2035)	0.1018 (0.1666)
48	1.516 (2.048)	0.2517 (0.4015)	0.4114 (0.5631)
96	13.07 (17.43)	1.278 (1.596)	2.144 (2.464)
192	N/A	9.253 (10.67)	N/A

d	Average (maximum) Running Time (seconds)		
	MV	RP	ES
3	5.683×10^{-4} (0.01122)	6.386×10^{-4} (3.707×10^{-3})	2.021×10^{-3} (0.07287)
6	1.247×10^{-3} (2.559×10^{-3})	0.01257 (0.07957)	5.231×10^{-3} (0.04967)
12	4.999×10^{-3} (0.01248)	4.660 (45.85)	0.7219 (114.6)
24	0.03626 (0.08234)	N/A	N/A
48	0.5097 (0.9342)	N/A	N/A
96	12.34 (21.41)	N/A	N/A
192	3419 (2.415×10^4)	N/A	N/A

Table 9.3: Test results on unbalanced tube **CCFP** cases.

9.1.3 Cases having Restricted Number of Solutions

This section presents the test result of highly structured random cases generated by the algorithms introduced in Subsection 8.2.3.

Cases Having Few Solutions

Table 9.4 presents the test results of the cases generated by Gen-Core-Low-Dense with the parameters $\beta_{\text{pole}} = \beta_{\text{equ}} = 0.05\pi/d$. From the results of RP we can notice that a case generated by Gen-Core-Low-Dense nearly has only $d^2 + 1$ feasible colourful simplices. If a case has exactly $d^2 + 1$ feasible colourful simplices, then in average RP should take as much as $\frac{(d+1)^{d+1}}{d^2+1}$ iterations to solve it. For example, the expected number of iterations should be 25.6 and 2.229×10^4 for the dimensions 3 and 6, respectively. The results for RP in Table 9.4 are only slightly lower. We also notice that BO1 takes long time to solve these cases.

d	Average (maximum) Number of Simplices					
	BO1	BO2	MU	MV	RP	ES
3	2.221 (5)	2.876 (6)	1.565 (3)	2.271 (6)	23.82 (174)	1.850 (14)
6	6.326 (11)	6.812 (13)	2.323 (6)	6.785 (12)	2.118×10^4 (267549)	20.21 (4974)
12	14.84 (24)	14.26 (24)	2.876 (7)	15.93 (23)	N/A	9.137×10^3 (2583443)
24	30.60 (50)	28.53 (46)	3.503 (11)	34.58 (48)	N/A	N/A
48	61.44 (83)	59.9 (87)	4.46 (14)	69.22 (93)	N/A	N/A
96	120.8 (149)	111.0 (183)	6.52 (17)	137.7 (179)	N/A	N/A
192	254.3 (383)	221.6 (345)	8.28 (27)	N/A	N/A	N/A
384	N/A	459.8 (704)	7.32 (38)	N/A	N/A	N/A

d	Average (maximum) Running Time (seconds)		
	BO1	BO2	MU
3	8.283×10^{-3} (0.4007)	9.090×10^{-4} (0.01148)	5.795×10^{-3} (0.01450)
6	0.02788 (0.04862)	2.404×10^{-3} (4.252×10^{-3})	0.01123 (0.02746)
12	0.1004 (0.1670)	0.007225 (0.01170)	0.02261 (0.05037)
24	0.4592 (0.6791)	0.02491 (0.03915)	0.06082 (0.1400)
48	3.057 (3.871)	0.1058 (0.1545)	0.2444 (0.5681)
96	28.49 (32.83)	0.5425 (0.8366)	1.291 (2.734)
192	398.9 (484.6)	3.711 (5.353)	7.633 (19.42)
384	N/A	37.01 (53.80)	42.28 (159.7)

d	Average (maximum) Running Time (seconds)		
	MV	RP	ES
3	6.383×10^{-4} (0.01403)	1.503×10^{-3} (0.01016)	1.978×10^{-3} (0.06770)
6	1.684×10^{-3} (2.591×10^{-3})	1.685 (21.16)	4.747×10^{-3} (0.2895)
12	5.936×10^{-3} (8.363×10^{-3})	N/A	0.6100 (165.6)
24	0.02942 (0.04087)	N/A	N/A
48	0.2896 (0.376)	N/A	N/A
96	4.016 (5.061)	N/A	N/A

Table 9.4: Test results on **CCFP** cases with few solutions.

Cases Having $(d+1)!$ and $d^{d+1}+1$ Solutions

Table 9.5 and Table 9.6 present the test results of the random cases from Gen-Core-Mid-Dense and Gen-Core-High-Dense, respectively. Both generators use the parameter $r = 1/d$. These cases are fairly "easy" for most of the algorithms to solve. We can notice that, for these cases:

- RP confirms the density of feasible solutions;
- ES only takes one iteration to solve;
- BO1 and MV perform exactly the same.

d	Average (maximum) Number of Simplices					
	BO1	BO2	MU	MV	RP	ES
3	1.23 (3)	2.235 (4)	0.899 (2)	1.23 (3)	9.751 (63)	1.003 (2)
6	2.375 (5)	5.262 (6)	0.999 (1)	2.375 (5)	160.3 (1008)	1 (1)
12	4.663 (8)	11.51 (12)	1 (1)	4.663 (8)	4.814×10^4 (358158)	1 (1)
24	9.01 (14)	23.76 (24)	1 (1)	9.01 (14)	N/A	1 (1)
48	18.06 (25)	47.9 (48)	1 (1)	18.6 (25)	N/A	1 (1)
96	36.4 (43)	95.92 (96)	1 (1)	36.4 (43)	N/A	1 (1)
192	71.28 (83)	191.9 (192)	1 (1)	71.28 (83)	N/A	1 (1)
384	140.6 (157)	384 (384)	1 (1)	N/A	N/A	1 (1)

d	Average (maximum) Running Time (seconds)		
	BO1	BO2	MU
3	4.915×10^{-3} (0.4091)	7.613×10^{-4} (0.01189)	3.492×10^{-3} (0.01539)
6	0.01012 (0.02259)	1.936×10^{-3} (3.263×10^{-3})	5.499×10^{-3} (6.603×10^{-3})
12	0.02885 (0.05511)	5.984×10^{-3} (6.997×10^{-3})	0.01114 (0.01182)
24	0.1201 (0.2089)	0.02138 (0.02213)	0.03270 (0.03480)
48	0.7570 (1.146)	0.09240 (0.09444)	0.1289 (0.1300)
96	6.858 (8.580)	0.4881 (0.5017)	0.5902 (0.6007)
192	83.33 (102.7)	3.274 (3.317)	3.131 (3.161)
384	1501 (1715)	30.15 (32.31)	19.96 (20.69)

d	Average (maximum) Running Time (seconds)		
	MV	RP	ES
3	5.286×10^{-4} (0.01471)	6.921×10^{-4} (8.902×10^{-3})	1.698×10^{-3} (0.06983)
6	9.529×10^{-4} (1.409×10^{-3})	0.01297 (0.08080)	2.946×10^{-3} (3.114×10^{-3})
12	2.400×10^{-3} (3.56×10^{-3})	5.887 (43.6993)	6.860×10^{-3} (7.23×10^{-3})
24	9.344×10^{-3} (0.01338)	N/A	0.02012 (0.02059)
48	0.07837 (0.1060)	N/A	0.07306 (0.07401)
96	0.9741 (1.194)	N/A	0.2895 (0.2978)
192	23.13 (26.93)	N/A	1.345 (1.362)
384	N/A	N/A	6.7438 (7.174)

Table 9.5: Test results on **CCFP** cases with $(d+1)!$ solutions.

d	Average (maximum) Number of Simplices					
	BO1	BO2	MU	MV	RP	ES
3	0.947 (2)	1.527 (4)	0.744 (3)	0.947 (2)	2.256 (31)	1.043 (4)
6	1.006 (3)	1.731 (5)	0.667 (1)	1.006 (3)	1.853 (16)	1.001 (2)
12	0.957 (5)	1.614 (11)	0.617 (1)	0.957 (5)	1.623 (14)	1 (1)
24	0.981 (6)	1.65 (11)	0.633 (1)	0.981 (6)	1.739 (13)	1 (1)
48	0.86 (3)	1.48 (7)	0.6 (1)	0.86 (3)	1.3 (5)	1 (1)
96	0.88 (4)	1.58 (9)	0.62 (1)	0.88 (4)	1.9 (17)	1 (1)
192	1.14 (4)	1.8 (5)	0.66 (1)	1.14 (4)	1.7 (7)	1 (1)
384	1 (1)	1.64 (6)	0.64 (1)	1 (1)	1.92 (10)	1 (1)

d	Average (maximum) Running Time (seconds)		
	BO1	BO2	MU
3	3.654×10^{-3} (0.3694)	5.735×10^{-4} (6.774×10^{-3})	2.773×10^{-3} (0.02006)
6	4.140×10^{-3} (0.01239)	8.450×10^{-4} (1.975×10^{-3})	3.688×10^{-3} (5.545×10^{-3})
12	5.852×10^{-3} (0.03046)	1.457×10^{-3} (5.721×10^{-3})	7.339×10^{-3} (0.01234)
24	0.01224 (0.07059)	3.685×10^{-3} (0.01076)	0.02239 (0.03274)
48	0.03761 (0.1009)	0.01589 (0.02347)	0.08758 (0.1284)
96	0.1736 (0.5470)	0.08279 (0.1150)	0.4159 (0.5839)
192	1.144 (2.758)	0.5597 (0.6009)	2.33 (3.020)
384	7.466 (21.19)	4.355 (4.649)	14.71 (18.85)

d	Average (maximum) Running Time (seconds)		
	MV	RP	ES
3	4.560×10^{-4} (0.01057)	2.303×10^{-4} (2.563×10^{-3})	1.625×10^{-3} (0.06566)
6	6.527×10^{-4} (1.084×10^{-3})	2.438×10^{-4} (1.396×10^{-3})	2.841×10^{-3} (3.081×10^{-3})
12	1.232×10^{-3} (2.454×10^{-3})	3.060×10^{-4} (1.858×10^{-3})	6.740×10^{-3} (7.184×10^{-3})
24	3.436×10^{-3} (6.431×10^{-3})	5.194×10^{-4} (2.971×10^{-3})	0.01991 (0.02045)
48	0.01728 (0.02132)	8.563×10^{-4} (2.62×10^{-3})	0.07224 (0.07290)
96	0.09721 (0.1408)	3.197×10^{-3} (0.02115)	0.2831 (0.2894)
192	0.7826 (1.151)	0.01108 (0.03403)	1.284 (1.291)
384	8.524 (16.75)	0.07922 (0.3121)	6.848 (6.946)

Table 9.6: Test results on **CCFP** cases with $d^{d+1}+1$ solutions.

9.2 General Cases

In this section we present test results for both the general feasible and infeasible **CFP** cases. RP and ES are tested with the feasible cases generated by Gen-Random (Subsection 8.3.1); SDPR is tested against the infeasible cases generated by Gen-Infeasible (Subsection 8.3.2).

The hardware and software environments of the tests on the feasible cases are the same as those in Section 9.1.

The computer to run the tests on infeasible cases has a 32-bit CPU (*Intel Core^{TM} 2 CPU T5600*). The CPU has two cores and the frequency of each core is 1.83 GHz. The total cache amount of the CPU is 2 Mb. The computer has 1 Gb random access memory. The operating system is *Windows XP home Edition* with *service pack 2*. The testing program runs on *MATLAB Version 7.4.0.287 (R2007a)* with *SeDuMi 1.1R3* [27] and *YALMIP R20070302* [16]. We can safely assume that the tests have been run in sequence with a single CPU core, according to our observation on the CPU resource distribution.

9.2.1 Feasible Cases

Table 9.7 presents the test results on the general feasible **CFP** cases. 1000 cases are tested for each dimension. In average ES is faster than RP, but in dimension 13 we find a case such that ES is slower than RP.

9.2.2 Infeasible Cases

Table 9.8 presents the results of testing SDPR on the infeasible **CFP** cases. The results show that the semidefinite relaxation **SDP** (Section 7.2) does not give a tight enough lower bound for the optimal objective value of **QP** (Section 7.1)

to detect the infeasibility. Tighter relaxation should be looked for.

d	Average (maximum) Number of Simplices		Average (maximum) Running Time (seconds)	
	RP	ES	RP	ES
3	5.06 (93)	1.98 (53)	3.822 (54.00)$\times 10^{-4}$	1.797 (78.11)$\times 10^{-3}$
5	30.06 (1378)	10.09 (505)	2.284 (100.0)$\times 10^{-3}$	3.379 (37.13)$\times 10^{-3}$
7	140.1 (1903)	45.2 (3420)	1.227 (16.51)$\times 10^{-2}$	7.169 (207.0)$\times 10^{-3}$
9	527.4 (9617)	145.4 (7788)	5.355 (97.23)$\times 10^{-2}$	1.576 (50.34)$\times 10^{-2}$
11	2248 (42156)	437.1 (23506)	2.596 (48.35)$\times 10^{-1}$	3.913 (153.6)$\times 10^{-2}$
13	9063 (324993)	3763 (2411461)	1.180 (41.87)	2.746 (1586) $\times 10^{-1}$
15	3.736$\times 10^4$ (434088)	4426 (368801)	5.381 (62.64)	3.569 (282.8)$\times 10^{-1}$
17	1.582$\times 10^5$ (2755684)	1.826$\times 10^4$ (4110061)	25.15 (439.6)	1.490 (307.1)
19	5.651$\times 10^5$ (5656177)	3.806$\times 10^4$ (6684801)	98.35 (979.3)	3.241 (541.8)
21	2.734$\times 10^6$ (27421450)	9.828$\times 10^4$ (22315968)	448.6 (5192)	8.656 (2024)

Table 9.7: Test results on general feasible **CFP** cases.

d	Number of Infeasible Cases		Average (maximum) Running Time (seconds)
	Detected (SDPR Infeasible)	Undetected	
2	727 (727)	273	0.1207 (0.2350)
3	542 (542)	458	0.2826 (0.8750)
4	372 (372)	628	0.6869 (2.7340)
5	257 (257)	743	1.484 (9.375)

Table 9.8: Test results on general infeasible **CFP** cases.

"Detected" means the number of input cases whose infeasibility is detected by SDPR; "SDPR Infeasible" means the number of input cases infeasible for SDPR; "Undetected" means the number of input cases whose infeasibility is not detected. For each dimension between 2 and 5, 1000 infeasible **CFP** cases are tested.

Chapter 10

Colourful Simplicial Depth

Bárány [3] proved that a *Colourful Core Feasibility Problem* (**CCFP**) always has a solution. We are consider the following question: What i sthe minimum number of solutions for a **CCFP** case? We present some results concerning this question, and show an induced lower bound for the monochrome (non-colourful) simplicial depth. In addition we show a parity property for the number of **CCFP** solutions.

10.1 Simplicial Depth and Colourful Simplicial Depth

In statistics there are several measures of the depth of a point $\mathbf{p}$ in $\mathbb{R}^d$ relative to a fixed set S of sample points. Two surveys on data depth are [1] and [13], see references therein. The depth measure we are interested in is the *simplicial depth* of $\mathbf{p}$, denoted $\text{depth}_{\mathbf{p}}(S)$, which is the number of simplices generated by points in S that contain $\mathbf{p}$. In dimension d if we consider sample points partitioned into at least $(d+1)$ colours, then we define the *colourful simplicial depth* of a point $\mathbf{p}$ relative to this sample to be the number of *colourful simplices*

(i.e., simplices with all vertices in different colours) that contain $\mathbf{p}$, denoted $\mathbf{depth_p}(\mathbf{S})$, where $\mathbf{S}$ is the colourful sample of points.

The number of solutions of a **CFP** case represented by $\mathbf{S}$ (introduced in Chapter 2) is the colourful simplicial depth of $\mathbf{0}$ relative to $\mathbf{S}$, namely, $\mathbf{depth_0}(\mathbf{S})$. We are interested in the the minimum colourful simplicial depth that a **CCFP** of dimension d can reach, so hereafter in this chapter we will assume that $\mathbf{0}$ is in the core of $\mathbf{S}$ and remind that $\mathbf{S}$ satisfies the assumptions in Section 2.2. If $\{\mathbf{0}\} \cup \mathbf{S}$ is not in general position, then we can perturb $\mathbf{S}$ to obtain $\mathbf{S}'$ such that $\mathbf{depth_0}(\mathbf{S}') \leq \mathbf{depth_0}(\mathbf{S})$, so we can assume $\{\mathbf{0}\} \cup \mathbf{S}$ to be in general position in this chapter.

We use the following notation: $\mu(d) = \min_{\mathbf{S}} \mathbf{depth_0}(\mathbf{S})$.

10.2 Observing Simplicial Depth by Cones

We can observe the simplicial depth by cones. The following proposition states how cones are related to convex hulls.

Proposition 10.2.1 *If $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_{d+1}\} \subset \mathbb{R}^d$ and $\{\mathbf{0}\} \cup T$ is in general position, then $\mathbf{0} \in \text{conv}(T)$ if and only if the antipode $-\mathbf{t}_{d+1}$ is in $\text{cone}(\mathbf{t}_1, ..., \mathbf{t}_d)$.*

Proof: If $\mathbf{0} \in \text{conv}(T)$, there exist λ_1, ..., $\lambda_{d+1} \geq 0$ such that $\sum_{i=1}^{d+1} \lambda_i = 1$ and $\sum_{i=1}^{d+1} \lambda_i \mathbf{t}_i = \mathbf{0}$. Because $\{\mathbf{0}\} \cup T$ is in general position, $\lambda_{d+1} > 0$. Then we can let $\sigma_i = \lambda_i / \lambda_{d+1}$ for $1 \leq i \leq d+1$, and $-\mathbf{t}_{d+1} = \sum_{i=1}^{d+1} \sigma_i \mathbf{t}_i$ indicates $-\mathbf{t}_{d+1} \in \text{cone}(\mathbf{t}_1, ..., \mathbf{t}_d)$.

If $-\mathbf{t}_{d+1} \in \text{cone}(\mathbf{t}_1, \ldots, \mathbf{t}_d)$, then we have $\sigma_1, \ldots, \sigma_d \geq 0$ such that $-\mathbf{t}_{d+1} = \sum_{i=1}^{d+1} \sigma_i \mathbf{t}_i$. Let $\lambda_i = \sigma_i / \left(1 + \sum_{j=1}^{d} \sigma_j\right)$ for $1 \leq i \leq d$ and $\lambda_{d+1} = 1 / \left(1 + \sum_{j=1}^{d} \sigma_j\right)$, then $\mathbf{0} \in \text{conv}(T)$. □

Take a point $\mathbf{s}$ from a finite set $S \in I\!R^d$. Call a simplex generated by points in S a $\mathbf{s}$-*simplex* if $\mathbf{s}$ is one of the points used to generate the simplex, and call a simplex *zero-containing* if it contains $\mathbf{0}$ in its interior. Define $\mathrm{z}_S(\mathbf{s})$ to be the number of zero-containing $\mathbf{s}$-simplices for a given S.

Proposition 10.2.1 states that $\mathrm{z}_S(\mathbf{s})$ is the number of simplicial cones generated by $S \setminus \{\mathbf{s}\}$ that contain $-\mathbf{s}$. We find it useful to think about what happens to $\mathrm{z}_S(\mathbf{s})$ if we move $\mathbf{s}$ while fixing the remaining points of S. This is particularly illustrative if we confine $-\mathbf{s}$ to the surface of the unit sphere $\mathbb{S}^d$ centered at $\mathbf{0}$.

Let $U = S \setminus \{\mathbf{s}\}$ with $|U| = u$. Initially $\mathrm{z}_S(\mathbf{s})$ will be the number of simplicial cones generated by sets of d points from U that contain $-\mathbf{s}$. Now consider what happens as $\mathbf{s}$ (and hence $-\mathbf{s}$) move. The value of $\mathrm{z}_S(\mathbf{s})$ will stay fixed until $-\mathbf{s}$ crosses the boundary of any simplicial cone. These boundaries are defined by the hyperplanes generated by $\mathbf{0}$ and subsets of $(d-1)$ points from U. Taking all such subsets from U, we can generate all such boundaries dividing $\mathbb{S}^d$ into open cells that are topologically $(d-1)$-dimensional open sets. We can define the depth of a cell of U to be the number of simplicial cones generated by points from U containing any given point in the interior of the cell.

For a d-dimensional simplicial cone K which is generated by d points (these points can be called *generators*), a *cone facet* is the cone generated by $(d-1)$ of K's generators. The boundary of K is the union of its cone facets.

Consider moving $\mathbf{s}$ along the surface of $\mathbb{S}^d$ to a new point $\mathbf{s}'$. If $-\mathbf{s}$ and $-\mathbf{s}'$ are in the same cell, we will have $\mathrm{z}_S(\mathbf{s}) = \mathrm{z}_S(\mathbf{s}')$. Now suppose $-\mathbf{s}$ is in a cell C adjacent to the cell containing $-\mathbf{s}'$. Then as we move from $-\mathbf{s}$ to $-\mathbf{s}'$

we cross a single cone facet F defined by a set U^0 of $(d-1)$ points from U belonging to F. Let us say that $-\mathbf{s}$ is on the left of H and $-\mathbf{s}'$ is on the right, where H is the hyperplane containing F. For the moment we assume that only $(d-1)$ points of U lie on H because of the general positioning. Let U^+ be the set of k points from U on the left of H, and let U^- be the $u-k-(d-1)$ points from U on the right. Since $-\mathbf{s}$ is in a cell bordered by H, it lies in the cone defined by the points from U^0 and any point $\mathbf{x} \in U^+$. On the other hand, $-\mathbf{s}$ is separated by H from the cones formed by U^0 and any $\mathbf{y} \in U^-$. Hence $-\mathbf{s}$ is contained in exactly k simplicial cones generated by U^0 and a single other point from U. Similarly, $-\mathbf{s}'$ is contained in exactly $u-k-(d-1)$ such cones. Simplicial cones that do not contain U^0 in their generating set will not have F as a cone facet, so they will contain $-\mathbf{s}$ if and only if they contain $-\mathbf{s}'$. Suppose $-\mathbf{s}$ is in l such cones, then $\mathrm{z}_S(\mathbf{s}) = l + k$, while $\mathrm{z}_S(\mathbf{s}') = l + u - k - (d-1)$.

We conclude that given the value of $\mathrm{z}_S(\mathbf{s})$ at some point $\mathbf{s}$, we can in principle compute $\mathrm{z}_S(\mathbf{s}')$ for any other point $\mathbf{s}'$ by tracing a path from $\mathbf{s}$ to $\mathbf{s}'$, and seeing how each hyperplane generated from points in $U = S \setminus \{\mathbf{s}\}$ divides the points of U. To do this formally, we need a topological lemma that says we can always draw a path between two points on $\mathbb{S}^d$ that crosses only cone facets generated by points from U (as opposed to passing through cones generated by fewer than $(d-1)$ points). This reduces to the following fact.

Proposition 10.2.2 *The sphere $\mathbb{S}^d$, a $(d-1)$-dimensional manifold, remains path connected after removing finitely many $(d-3)$-dimensional manifolds.*

Proof: classic algebraic topology arguments, see for example, [19]. □

To apply the observation method using cones to points partitioned into different colours, we define *colourful simplicial cones* and *colourful cone facets* as simplicial cones and cone facets with all generators in different colours, respectively.

10.3 Parity Property

Proposition 10.3.1 *For any colourful configuration* $\mathbf{S}$ *in general position and in odd dimension* d, $\mathbf{depth_0}(\mathbf{S})$ *is even.*

Proof: Suppose we begin with a configuration $\mathbf{S}^0$ (with $(d+1)$ points in each of $(d+1)$ colours) having all points clustered together. Then we can move one point at a time from its initial position in $\mathbf{S}^0$ to its final position in $\mathbf{S}$ generating a sequence of configurations $\mathbf{S}^0$, $\mathbf{S}^1, \ldots, \mathbf{S}^{(d+1)^2} = \mathbf{S}$. Clearly, $\text{depth}_{\mathbf{0}}(\mathbf{S}^0) = 0$. As we move a given point $\mathbf{s}_i$ of colour j from its initial position in $\mathbf{S}^0$ (and $\mathbf{S}_i$) to its final position in $\mathbf{S}$ (and $\mathbf{S}^{i+1}$), we need only to know what happens when the antipode $-\mathbf{s}_i$ crosses colourful cone facets defined by a set of $(d-1)$ points of $(d-1)$ different colours but not of colour j. The hyperplane H containing such a colourful cone facet F will miss only one other colour, denoted j'. There will be k points of colour j' on one side of H, and $(d+1-k)$ on the other side. As $-\mathbf{s}_i$ crosses F the number of colourful simplicial cones containing $-\mathbf{s}_i$ generated by points from F and a point of colour j' changes from k to $(d+1-k)$. As long as $(d+1)$ is even, the parity does not change. □

10.4 Colourful Simplicial Depth Lower Bound

Bárány [3] proved $\mu(d) \geq d+1$ by showing that each point in $\mathbf{S}$ participates in the generation of at least one colourful simplex covering $\mathbf{0}$. In [10] we proved $\mu(d) \geq 2d$ (the proof is not provided in this thesis since tighter bounds have been proved and the techniques are similar). Soon after that Stephen and Thomas [25] and Bárány and Matoušek [5] proved $\mu(d) \geq \lfloor (d+2)^2/4 \rfloor$ and $\mu(d) \geq d(d+1)/5$, respectively. In addition, Bárány and Matoušek [5] proved $\mu(d) \geq 3d$ for $d \geq 3$, which it settles $\mu(3) = 10$ when together with the parity argument (in Section 10.3) and $\mu(d) \leq d^2+1$ (in Subsection 10.4.1).

In Subsection 10.4.2 we introduce an application of $\mu(d)$: to provide an bound on monochrome simplicial depth.

10.4.1 Low Depth Configurations

We now describe how to build a colourful configuration $\mathbf{S}^-$ that contains $\mathbf{0}$ in its core, and only d^2+1 colourful simplices contain $\mathbf{0}$. Our strategy is to fix the first d sets S_1, S_2, ..., S_d and then consider possible placements of the points $\mathbf{s}_1^{d+1}$, $\mathbf{s}_2^{d+1}$, ..., $\mathbf{s}_{d+1}^{d+1}$ to form S_{d+1}. We will place the points from $S^- = S_1 \cup ... \cup S_d$ on the sphere $\mathbb{S}^d$ in such a way that some regions of $\mathbb{S}^d$ are sparsely covered by colourful cones from them. We begin by fixing $\epsilon = 1/100d$. We will place the points from S^- in three locations on $\mathbb{S}^d$. The first is on the *Tropic of Capricorn*, which we define to be the set of points on $\mathbb{S}^d$ whose dth coordinates are -2ϵ. The second is on the *Tropic of Cancer*, whose dth coordinate is ϵ. The two tropics are topologically copies of $\mathbb{S}^{d-1}$, but unlike their namesakes they are not equally spaced from the equator. The final region is the polar region of points in $\mathbb{S}^d$ which are within ϵ of the *North Pole* $\mathbf{p}_{\text{north}} = [0\,0\,...\,0\,1]^T$ (see

Figure 10.1).

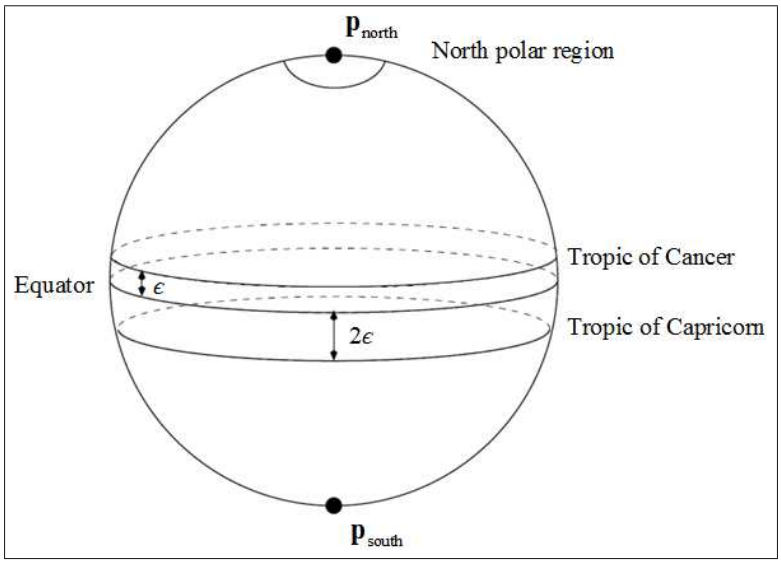


Figure 10.1: Points placement in dimension 3 for constructing $\mathbf{S}^-$

Now let us fix the positions of the points in S_1. Take

$$\mathbf{s}_1^1 = \left[\sqrt{1-4\epsilon^2} \quad 0 \ldots 0 \quad -2\epsilon\right]^T, \tag{10.4.1}$$

$$\mathbf{s}_2^1 = \left[\sqrt{1-\epsilon^2} \quad 0 \ldots 0 \quad \epsilon\right]^T. \tag{10.4.2}$$

Note that the line segment between $\mathbf{s}_1^1$ and $\mathbf{s}_2^1$ passes just below $\mathbf{0}$ in the sense that it contains a point whose first $(d-1)$ coordinates are 0's, and whose dth coordinate is negative (and small). We now place the remaining points $\mathbf{s}_3^1$, ..., $\mathbf{s}_{d+1}^1$ in the polar region in such a way as to ensure that $\mathbf{0}$ in the interior of $\text{conv}(S_1)$. For $d = 2$ we can do this by placing $\mathbf{s}_3^1$ at the North Pole. For $d \geq 3$ we can place the points on the section of the *Arctic Circle* (points with distance ϵ to the North Pole) with their first coordinates 0. Topologically the section of Arctic Circle is a copy of $\mathbb{S}^{d-2}$; we can take $\mathbf{s}_3^1$, ..., $\mathbf{s}_{d+1}^1$ to be the vertices of a

regular simplex inscribed on this sphere.

The points of colours 2 to d are chosen similarly. The first points from each of the d colours are arranged in a regular simplex on Capricorn. The remaining points in the same relative position to the first point, so that each S_i is a rotation of S_1 around the dth coordinate axis. In particular, for each $i = 1, \ldots, d$, the second point of S_i will lie on Cancer and the final $(d-1)$ points will lie in the polar region.

We finish our construction by considering possible placements of the points in S_{d+1}. We want to place them in such a way that their *antipodes* (the $-\mathbf{s}_i^{d+1}$s) are contained in few colourful simplicial cones generated from S^-.

Consider the cell C_{south} defined by colours 1 to d of $\mathbf{S}^-$ on $\mathbb{S}^d$ which contains the *South Pole* $\mathbf{p}_{\text{south}} = [0 \ldots 0 \ -1]^T$. We claim this is exactly the intersection of $\mathbb{S}^d$ with the single colourful simplicial cone K_{Cap} defined by the d colourful points on Capricorn. This follows since any other colourful cone is generated by a set of d coloured points chosen from Capricorn, Cancer and the northern polar region. Fix such a set and let G_{Cap}, G_{Can} and G_{Pole} be the subsets of it from Capricorn, Cancer and the northern polar region, respectively. Let $K_G = \text{cone}(G_{\text{Cap}}, G_{\text{Can}}, G_{\text{Pole}})$. We assume that we have $|G_{\text{Cap}}| < d$, otherwise K_G will be identical to K_{Cap}. We need to show that $\text{int}(K_{\text{Cap}}) \cap \text{int}(K_G) = \emptyset$. To do this, we find a hyperplane separating K_{Cap} and $\text{int}(K_G)$. If $G_{\text{Cap}} = \emptyset$, the hyperplane through the equator will do. Otherwise, take any hyperplane H that goes through $\mathbf{0}$ and all points in G_{Cap}, such that one of the points in G_{Cap} is the steepest direction on H to decrease the last coordinate. Then all points from $G_{\text{Can}} \cup G_{\text{Pole}}$ will be separated from K_{Cap} as long as ϵ is small enough. This completes the proof. We conclude that the cell

C_{south} is covered only by the colourful cone K_{Cap} and closely approximates the spherical cap bounded by Capricorn.

It is a good strategy to place the antipodes $-\mathbf{s}_i^{d+1}$ in C_{south}. If we do this for all points of S_{d+1}^-, however, the resulting configuration will not have $\mathbf{0} \in \text{conv}(S_{d+1}^-)$ (S_{d+1}^- would certainly be contained in an open hemisphere). Therefore we must have at least one antipode, say $-\mathbf{s}_1^{d+1}$, outside C_{south}. Indeed, if we place the remaining $-\mathbf{s}_i^{d+1}$'s inside C_{south}, we would need to have $-\mathbf{s}_1^{d+1}$ in the cone generated by the antipodes of the points on Capricorn. In particular, it is above Cancer.

Let $S_A = \{\mathbf{s}_1^1, \mathbf{s}_1^2 \ldots, \mathbf{s}_1^d\}$ be the set of points from S_1, S_2, ..., S_d on Capricorn. Similarly, let $S_B = \{\mathbf{s}_2^1, \mathbf{s}_2^2 ..., \mathbf{s}_2^d\}$ be the set of points on Cancer. Let us count how many colourful simplicial cones from S^- must contain $-\mathbf{s}_1^{d+1}$ if we place $-\mathbf{s}_1^{d+1}$ in cone$(-\mathbf{s}_1^1, -\mathbf{s}_1^2 ..., -\mathbf{s}_1^d)$. To do this, we start with $-\mathbf{s}_1^{d+1}$ in C_{south} and then move it above Cancer noting which cell boundaries it crosses as suggested in Section 10.2. This structure of the cell boundaries is a topological question, so we find it convenient to remove the $\mathbf{p}_{\text{south}}$ and equate $\mathbb{S}^d$ with $\mathbb{R}^{d-1}$.

With the exception of the single colourful simplicial cone that contains C_{south}, the colourful simplicial cones generated by S^- correspond to colourful simplices in $\mathbb{R}^{d-1}$. The polar points on $\mathbb{S}^d$ will be clustered near the origin in $\mathbb{R}^{d-1}$. Let $A = \{\mathbf{a}_1, ..., \mathbf{a}_d\}$ and $B = \{\mathbf{b}_1, ..., \mathbf{b}_d\}$ be the projections of S_A and S_B in $\mathbb{R}^{d-1}$, respectively. Then conv(A) and conv(B) form nested simplices which contain the projection of the polar region. The boundaries of the colourful simplicial cones on $\mathbb{S}^d$ map to facets of simplices in $\mathbb{R}^{d-1}$; both are defined by sets of $(d-1)$ colourful points. Moving $-\mathbf{s}_1^{d+1}$ from C_{south} to cone$(-\mathbf{s}_1^1, -\mathbf{s}_1^2 ..., -\mathbf{s}_1^d)$ corresponds to moving the projection of it, denoted $\mathbf{c}$,

from outside conv(A) to inside conv(B). More precisely, $\mathbf{c}$ should be moved inside the projection of cone($-\mathbf{s}_1^1$, $-\mathbf{s}_1^2$ ..., $-\mathbf{s}_1^d$), which is in turn inside conv(B). Since this can be done while not changing the depth after $\mathbf{c}$ is moved inside conv(B), we only discuss moving $\mathbf{c}$ inside conv(B).

Let us now see what simplicial facets $\mathbf{c}$ must cross to do this. If we keep $\mathbf{c}$ far away from the $\mathbf{a}_i$'s and $\mathbf{b}_i$'s themselves, we can avoid any facets involving the polar points: These facets involve at most $(d-2)$ generators from A and B, and hence have ends that are at most $(d-3)$-dimensional manifolds in conv(A) $\setminus$ int(conv(B)). The ends can be avoided by Proposition 10.2.2.

There are still $d2^{d-1}$ colourful facets defined by choosing $(d-1)$ colourful points from A and B. We can enumerate them by first choosing an index (colour) to omit and then representing the choices of $\mathbf{a}_i$'s and $\mathbf{b}_i$'s by a 0-1 sequence of length $(d-1)$. Letting 0 represent the choice of an $\mathbf{a}_i$, conv(A) is bounded by the facets defined by an index choice and a sequence of 0's, while conv(B) is bounded by the facets defined by an index choice and a sequence of 1's. In fact there are 2^d colourful simplices generated by A and B, and they are enumerated by 0-1 sequences of length d. Their facets are enumerated by choosing an index to drop from the enumerating sequence.

Now start with $\mathbf{c}$ outside conv(A). To bring $\mathbf{c}$ inside conv(B), we must start by bringing it into conv(A). This involves crossing some boundary face of conv(A), say the one defined by $\mathbf{a}_1, \ldots, \mathbf{a}_{d-1}$. This is enumerated as $(d, 0, 0, ..., 0, 0)$. We can proceed through facets $(d-1, 0, 0, ..., 0, 1)$, $(d-2, 0, 0, ..., 0, 1, 1)$ until finally we cross $(1, 1, 1, ..., 1)$ into a cell of conv(B). This involves d facet crossings, which is minimal since at each crossing we can only add a single 1 to the 0-1 part of the enumerating sequence.

We claim that as $\mathbf{c}$ crosses each facet, it makes a net gain of $(d-1)$ containing simplices. At the first facet, $(d, 0, 0, ..., 0, 0)$, $\mathbf{c}$ leaves the single exterior simplex defined by the points A projected from Capricorn and enters the d simplices defined by $\mathbf{a}_1$, ..., $\mathbf{a}_{d-1}$ and the d points of colour d other than $\mathbf{a}_d$ (we do not count $\mathbf{a}_d$ because conv(A) corresponds to C_{south}). At subsequent facet crossings, the same thing happens for the remaining colours: $\mathbf{c}$ leaves the simplex defined by the crossing facet and $\mathbf{a}_i$. As $\mathbf{c}$ leaves, it enters the simplices defined by this facet and the d remaining points of colour i. Hence the number of simplices containing $\mathbf{c}$ immediately after crossing into conv(B) is exactly $1 + d(d-1)$.

We now return our attention to $\mathbb{S}^d$ and use $C_{\mathbf{c}}$ to denote the cell containing $-\mathbf{s}_1^{d+1}$ whose projection on $I\!R^{d-1}$ lies inside conv(B). From our construction, $C_{\mathbf{c}}$ is a cell above Cancer. We want to claim that in fact it contains some point above the set of antipodes of Capricorn, that is, a point whose antipode is in C_{south}. This is a complicated geometric calculation. However, we observe that nothing in our topological argument above changes if we change the constant 2ϵ in our definition of Capricorn to $c\epsilon$ for any $1 < c \leq 2$. In particular, the cell $C_{\mathbf{c}}$ is not degenerate if we move the antipodes of Capricorn towards Cancer by decreasing c close to 1. Therefore for some $c > 1$ (this condition maintains $\mathbf{0} \in \text{int}(\text{conv}(S_i))$ for $i = 1, ..., d$), $C_{\mathbf{c}}$ includes some point above the antipodes of Capricorn. Any such c and point in $C_{\mathbf{c}}$ would be sufficient for our construction. We have used $c = 2$ for concreteness and take it as an article of faith that this is a small enough for our choice of ϵ.

The construction can now be completed. Take $-\mathbf{s}_2^{d+1}$ to be the midpoint of the shortest spherical segment between Capricorn and $\mathbf{s}_1^{d+1}$ (which lies below

Capricorn). Let $z < -2\epsilon$ be the final coordinate of $-\mathbf{s}_2^{d+1}$ and arrange the remaining points so that $-\mathbf{s}_2^{d+1}$, $-\mathbf{s}_3^{d+1}$, ..., $-\mathbf{s}_{d+1}^{d+1}$ form a $(d-1)$-dimensional regular simplex on $\mathbb{S}^d \cap \{\, \mathbf{x} \in I\!R^d \,:\, x_d = z \,\}$. Then $\mathbf{0}$ is in the convex hull of $\mathbf{S}_{d+1}$. Finally we can calculate $\mathbf{depth_0}(\mathbf{S}^-)$ from the location of the $-\mathbf{s}_i^{d+1}$'s: $\mathbf{0}$ lies in $1 + d(d-1)$ colourful simplices generated with $\mathbf{s}_1^{d+1}$ and one colourful simplex each including $\mathbf{s}_2^{d+1}$, $\mathbf{s}_3^{d+1}$, ..., $\mathbf{s}_{d+1}^{d+1}$. Hence,

$$\mathbf{depth_0}(\mathbf{S}^-) = 1 + d(d-1) + d = d^2 + 1. \tag{10.4.3}$$

This construction shows $\mu(d) \leq d^2 + 1$.

10.4.2 Application of Colourful Simplicial Depth

Even before the notion of simplicial depth was introduced in statistics, the question of computing bounds for the number of simplices generated by a set S of n points and covering a point $\mathbf{p} \in \mathrm{conv}(S)$ was studied in the combinatorics and computational geometry communities. We denote,

$$\mathrm{g}(S) = \max_{\mathbf{p} \text{ in general position with } S} \mathrm{depth}_{\mathbf{p}}(S). \tag{10.4.4}$$

The 2-dimensional question dates back at least to Kárteszi [15] who showed that for n points in the plane, $\mathrm{g}(S)$ is at most $(n^3 - n)/24$ for odd n and at most $(n^3 - 4n)/24$ for even n, and showed that these bounds were attained when S is the set of vertices of a regular n-gon. In the early 1980s, Boros and Füredi [4] showed $\mathrm{g}(S)$ is at least $n^3/27 + O(n^2)$, and gave configurations achieving this bound.

Bárány [3] gave bounds for the monochrome simplicial depth in dimension d as an application of his Colourful Carathéodory Theorem (see Chapter 2). He showed that after colouring the points in S, some point $\mathbf{p}$ must be contained in

many colourful simplices. A key point of Bárány's proof is that a core point $\mathbf{p}$ of a colourful configuration must lie in at least one colourful simplex. Using this fact, for a set S of n points (assume n is sufficiently large) in general position in $\mathbb{R}^d$ Bárány obtains a lower bound of

$$\mathrm{g}(S) \geq \frac{1}{(d+1)^{d+1}}\binom{n}{d} + O(n^d). \tag{10.4.5}$$

This result is asymptotically sharp up to a constant factor as a function of n (for fixed d). However, as Bárány remarks, the constant is probably quite far from the truth. Indeed, he gives a sharp upper bound of

$$\mathrm{g}(S) \leq \frac{1}{2^d(d+1)!}\binom{n}{d} + O(n^d). \tag{10.4.6}$$

We speculate that the true lower bound is not much less than the upper bound. One way to improve the lower bound is to show that a core point $\mathbf{p}$ lies in more colourful simplex. In Bárány's original paper, he notes that $\mathbf{p}$ must in fact lie in at least $(d+1)$ colourful simplices, thereby improving the lower bound to

$$\mathrm{g}(S) \geq \frac{1}{(d+1)^{d}}\binom{n}{d} + O(n^d). \tag{10.4.7}$$

More generally

$$\mathrm{g}(S) \geq \frac{\mu(d)}{(d+1)^{d+1}}\binom{n}{d} + O(n^d). \tag{10.4.8}$$

Because $\mu(d) \leq d^2+1$, this construction cannot give cannot give a stronger bound than

$$\mathrm{g}(S) \geq \frac{d^2+1}{(d+1)^{d+1}}\binom{n}{d} + O(n^d). \tag{10.4.9}$$

Wagner proved exactly this bound in his thesis [31] as a special case of his *First Selection Lemma*. This is, to our knowledge, the first improvement since Bárány's original paper [3]. We find the appearance of the constant d^2+1, which for us arrives from colourful combinatorics, quite remarkable.

Chapter 11

Conclusions and Future Work

11.1 Conclusion

This thesis deals with the *Colourful Feasibility Problem* (**CFP**) and, in particular, with the *Colourful Core Feasibility Problem* (**CCFP**).

We presented the two previously known algorithms for **CCFP**, see [7]: Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2. Both these algorithms are essentially geometric. Their complexity guarantees depend crucially on having $\mathbf{0}$ in the interior of the convex hull of each colour, and therefore the algorithms have the same iteration complexity which depends on the dimension number d of the maximum radius ρ of a largest ball around $\mathbf{0}$ inscribed in the core. Solver-Bárány-Onn-1 visits each colourful simplex at most once but each iteration has a relatively higher cost. Solver-Bárány-Onn-2 may revisit some colourful simplex but each iteration has a relatively lower cost: We give a method to reduce the order of arithmetic operations per iteration from $O(d^4)$ to $O(d^3)$ under the the general position assumption.

We have first implemented the Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2 algorithms and two alternative simple algorithms: Solver-Random-Pick

and Solver-Max-Volume. We developed a set of random case generators to evaluate the performance of these algorithms. We showed that Solver-Bárány-Onn-1 and Solver-Bárány-Onn-2 are usually faster, and exhibited a cycling example for Solver-Max-Volume. On another hand, we showed that oscillations leading to extremely slow convergence could occur for Solver-Bárány-Onn-2, in particular when the **CCFP** is ill-conditioned, i.e., when ρ is small.

While Solver-Bárány-Onn-2 is the most efficient algorithm for **CCFP** in average, in Chapter 4 we designed a Solver-Multi-Update variant which further exploits the the geometric structure by performing multiple updates for each iteration. The main features of this enhanced algorithm are:

- same worst case iteration complexity as the previously known algorithms;
- no oscillation occurs;
- fast and robust performance on the benchmark tests.

In Chapter 6 we considered the Colourful Feasibility Problem (**CFP**) and proposed an algorithm: Solver-Enum. This combinatorial approach checks colourful simplex likely to contain **0** until a solution is found or, for infeasible problems, check each colourful simplex exactly once. The performances for Solver-Enum are typically much better than for Solver-Random-Pick as, in addition of exploiting the combinatorial structure, it incorporated geometric heuristics.

Following the combinatorial and geometric approaches, we investigated the optimization approach in Chapter 7 and formulated **CFP** as a Nondefinite Quadratic Optimization Problem **QP**. We also considered a positive semidefinite relaxation **SDP** of **QP** which can be solved in polynomial time in order to identify infeasible **CFP** cases.

Besides the algorithmic aspects, in Chapter 10 we investigate the lower bound $\mu(d)$ for the number of feasible solutions for d-dimensional **CCFP**. We constructed a **CCFP** case with an unexpected low number of solutions: $d^2 + 1$, and proved that $\mu(d) \geq 2d$ using topological arguments which triggered a new series of results: Bárány and Matoušek [5] and Stephen and Thomas [25] independently proved, using similar topological techniques, quadratic lower bounds for $\mu(d)$. The introduced d^2+1 upper bound for $\mu(d)$ was the first non-trivial one and the $2d$ lower bound was the first improvement since Bárány's result in 1982. The lower bound for $\mu(d)$ yields a lower bound of maximum monochrome simplicial depth of n points.

11.2 Future Work

The proposed algorithms for **CCFP** are quite efficient and therefore the next focus could be on the general **CFP**. It includes refining the **QP** formulation yielding an efficient algorithm and proposing tighter convex relaxation for the **QP**. Another possible approach is to design a branch-and-bound or branch-and-reduce algorithm, which could safely discard the branched nodes that have positive objective function value lower bounds. A good convex relaxation may also improve the quality of branch-and-bound or branch-and-reduce algorithm.

Besides the application to monochrome simplicial depth mentioned above, the questions studied in this thesis are connected to combinatorial questions related to the *Colourful Carathéory Theorem*. For this kind of questions, see Bárány and Onn [6].

The **CFP** models a situation where we want to select a set of points that is both diverse, in the sense that it includes representatives from predetermined

classes (colours), and representative, in the sense that the selected points surround a specified point common to all the classes. This viewpoint suggested by Lu [17], in the context of data-mining, may provide applications to the **CFP**.

Bibliography

[1] G. Aloupis: Geometric Measure of Data Depth. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 72 (2006) 147–158.

[2] K. M. Anstreicher, D. den Hertog, C. Roos and T. Terlaky: A Long-Step Barrier Method for Convex Quadratic Programming. *Algorithmica* 10 (1993) 365–382.

[3] I. Bárány: A Generalization of Carathéodory's Theorem *Discrete Mathematics* 40(2-3) (1982) 141–152.

[4] E. Boros and Z. Füredi: The Number of Triangles Covering the Center of an n-set. *Geometriae Dedicata* 17(1) (1984) 69–77.

[5] I. Bárány and J. Matoušek: Quadratically Many Colourful Simplices. *SIAM Journal on Discrete Mathematics* 21(1) (2007) 191–198.

[6] I. Bárány and S. Onn: Carathéory's Theorem: Colourful and Applicable. *Intuitive Geometry* (Budapest, 1995); *Bolyai Society Mathematical Studies* 6 (1997), 11-21; *János Bolyai Mathematical Society* (1997) 11–21.

[7] I. Bárány and S. Onn: Colourful Linear Programming and its Relatives *Mathematics of Operations Research* 22(3) (1997) 550–567.

[8] Y. Censor and T. Elfving: Block-Iterative Algorithms with Diagonally Scaled Oblique Projections for the Linear Feasibility Problem *SIAM Journal on Matrix Analysis and Applications* 24(1) (2002) 40–58.

[9] G. Dantzig: Programming in a Linear Structure. *Econometrics* 17 (1949).

[10] A. Deza, S. Huang, T. Stephen, and T. Terlaky: Colourful Simplicial Depth. *Discrete and Computational Geometry* 35(4) (2006) 597-604.

[11] A. Deza, S. Huang, T. Stephen, and T. Terlaky: Colourful Feasibility Problem. *AdvOL-Report 2005/20, McMaster University* 2005.

[12] B. Filiz, Z. Csizmadia and T. Illés: Anstreicher-Terlaky type Monotonic Simplex Algorithms for Linear Feasibility Problems. *Optimization Online* 2005.

[13] K. Fukuda and V. Rosta: Data Depth and Maximal Feasible Subsystems. *Graph Theory and Combinatorial Optimization (D. Avis, A. Hertz, and O. Marcotte, eds)* Springer-Verlag (2005) 37–67.

[14] S. Huang and H. Zhang: MATLAB code for Colour Feasibility Problem, available at: http://optlab.mcmaster.ca/∼huangs3/cfp/cfp.htm

[15] V. F. Kárteszi: Extremalaufagaben über endliche Punksysteme. *Publicationes mathematicae Debrecen* 4 (1955) 16–27.

[16] J. Löfberg: YALMIP: A Toolbox for Modeling and Optimization in MATLAB. *Proceedings of the CACSD Conference*, Taipei, Taiwan (2004).

[17] Z. Lu: personal communication.

[18] M. E. Muller: A Note on a Method for Generating Points Uniformly on N-Dimensional Spheres. *Communications of the Association for Computing Machinery* 2 (1959) 19–20.

[19] J. R. Munkres: Elements of Algebraic Topology. *Addison-Wesley* (1984).

[20] J. O'rourke: Computational Geometry in C, 2nd Edition. *Cambridge University Press* (1998).

[21] I. Pólik: personal communication.

[22] F. A. Potra and R. Sheng: A Superlinearly Convergent Primal-Dual Infeasible-Interior-Point algorithm for Semidefinite Programming. *SIAM J. Optim.* 8 (4) (1998) 1007–1028.

[23] P. M. Pardalos and S. A. Vavasis: Quadratic Programming with One Negative Eigenvalue Is NP-Hard. *Journal of Global Optimization* 1 (1991) 15–22.

[24] A. Schrijver: *Theory of Linear and Integer Programming*. John Wiley & Sun (1986).

[25] T. Stephen and H. Thomas: A Quadratic Lower Bound for Colourful Simplicial Depth. *submitted, arXiv:math.CO/0512400.*

[26] T. Stephen: personal communication.

[27] SeDuMi conic optimization solver: http://sedumi.mcmaster.ca

[28] S. M. Rump, T. Ogita and S. Oishi: Accurate Floating-point Summation Part I: Faithful Rounding. Submitted for publication in *SIAM Journal on Scientific Computing*.

[29] S. M. Rump, T. Ogita and S. Oishi: Accurate Floating-point Summation Part II: Sign, κ-fold Faithful and Rounding to Nearest. Submitted for publication in *SIAM Journal on Scientific Computing*.

[30] T. Terlaky and S. Zhang: Pivot Rules for Linear Programming: a Survey on Recent Theoretical Developments. *Annals of Operations Research* 46 (1993) 203–233.

[31] U. Wagner: On k-sets and applications, Ph.D. Thesis, Department of Mathematics, ETH Zürich (2003).

[32] Y. Ye: On Affine Scaling Algorithms for Nonconvex Quadratic Programming. *Mathematical Programming* 56 (1992) 285–300.

[33] H. Zhang: personal communication.