DISS. ETH NR. 24335

Redundancy in Linear Systems: Combinatorics, Algorithms and Analysis

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

May Krisztina Szedlák
M. Sc. In Mathematics, ETH Zurich

born December 10, 1989
citizen of Worb, BE

accepted on the recommendation of

Prof. Dr. Bernd Gärtner, examiner
Prof. Dr. Komei Fukuda, co-examiner
Dr. Kenneth Clarkson, co-examiner
Prof. Dr. Rico Zenklusen, co-examiner

2017

Contents

Contents **iv**

1 Overview **9**

1.1 Motivation 9

1.2 Problems and Goals 15

1.3 Main Results 16

I Generalizations of Clarkson's Redundancy Removal Algorithm **21**

2 Preliminaries **23**

2.1 Redundancy Removal in Linear Programming 23

2.2 Clarkson's Redundancy Removal Algorithm 25

2.2.1 The Algorithm 25

2.2.2 Preprocessing of the Algorithm 27

3 Combinatorial Redundancy Removal **29**

3.1 Introduction 29

3.2 Basics 33

3.2.1 LP in Dictionary Form 33

3.2.2 Pivot Operations 36

3.3 Combinatorial Redundancy 36
3.4 Certificates 38
3.4.1 A Certificate for Redundancy in the Dictionary Oracle 39
3.4.2 A Certificate for Nonredundancy in the Dictionary Oracle 40
3.4.3 Finite Pivot Algorithms for Certificates 43
3.5 An Output Sensitive Redundancy Detection Algorithm . 44
3.5.1 General Redundancy Detection 45
3.5.2 Strong Redundancy Detection 49
3.6 Number of Dictionaries for all Certificates 51
3.7 Alternative Nonredundancy Certificate 54
3.7.1 Another Certificate 54
3.7.2 Comparison of Certificates 56
3.8 Discussion and Open Questions 57

4 Redundancy Detection in Linear Systems with two Variables per Inequality 59

4.1 Introduction 59
4.2 Definitions and Preliminaries 61
4.3 A Strongly Polynomial Time Redundancy Detection Algorithm for Linear Programs with two Variables per Inequality 62
4.4 Revision of the Hochbaum-Naor Method 65
4.4.1 The Ingredients 65
4.4.2 The Hochbaum-Naor Algorithm for the Feasible Case 74
4.4.3 Discussion of the Hochbaum Naor Algorithm . . 77

4.4.4 Discussion of the Hochbaum-Naor Algorithm in the Infeasible Case 78
4.5 Modification of Hochbaum-Naor Method 79
4.6 The Non-Full-Dimensional Case 81
4.7 Discussion and Open Questions 83

5 Complexity of Polytopes with two Variables per Inequality 85
5.1 Introduction 85
5.2 Definitions and Known Results 86
5.3 Lower Bound on Maximum Complexity of $LI(2)$ 88
5.4 Upper Bound on Maximum Complexity of $LI(2)$ 97
5.5 Discussion and Open Questions 102

II Sampling with Removal in Generalizations of Linear Programming 103

6 Sampling with Removal 105
6.1 Introduction 105
6.2 Basics and Definitions 108
6.2.1 LP-type Problems 109
6.2.2 Violator Spaces 110
6.2.3 Consistent Spaces 115
6.2.4 Sampling with Removal 116
6.3 An Upper Bound for Consistent Spaces 117
6.4 An Upper Bound for Violator Spaces 125
6.4.1 Extreme Constraints after Removal 126

6.4.2 Sampling Lemma after Removal 130
6.4.3 Violators after Removal 132
6.5 A Lower Bound for Consistent Spaces 133
6.6 Lower Bounds for Violator Spaces 136
6.7 Characterization of Violator Spaces with Combinatorial Dimension 1 139
6.8 Discussion and Open Questions 143

Abstract

The problem of detecting and removing redundant constraints is fundamental in optimization. Finding the redundancies helps in understanding the underlying problem and can greatly improve the running time of subsequent computations. Intuitively, we are provided with a problem described by a set of constraints, but only some of them are needed to describe any of its solutions. The constraints that are not needed for this description are redundant.

More specifically, we consider the setting of linear programs (LPs) in d variables, given by a linear function that has to be maximized subject to a system of n inequalities. The region satisfying all the inequalities defines a convex polyhedron in $\mathbb{R}^d$, the feasible region of the LP. An inequality of this system is called *redundant*, if after removing it the LP still has the same feasible region, otherwise it is called nonredundant. The currently fastest method to detect all redundancies is the one by Clarkson: it solves n linear programs, but each of them has at most s inequalities, where s is the number of nonredundant inequalities. Additionally the algorithm executes s ray shootings. This algorithm is very efficient in the case where the number of nonredundant inequalities is small.

Firstly, we modify Clarkson's algorithm in a way that it does not rely on geometrical notions such as ray shooting. We show that knowing only the *signs* of all the dictionaries, — which is combinatorial information — suffices to detect all redundancies. A dictionary can be thought of as a matrix, which gives an encoding of the relative positions of the inequalities to each other. Although our algorithm is slower than Clarkson's redundancy removal algorithm, it is still output sensitive, meaning that the running time depends on the number of nonredundant inequalities. Moreover it uses the minimum information needed to detect all redundancies; for an exact implementation, only signs need to be evaluated correctly. Furthermore our algorithm is naturally extendable to the setting of oriented matroids. In the case where all the inequalities are in general position, our running time essentially matches the time of the Clarkson method.

Secondly we give a strongly polynomial time algorithm (polynomial number of elementary operations in n and d) to detect all redundancies in the special case where the LP has at most two variables per inequality. Although in this case the structure is simpler than in a general LP, the defined polyhedron can still have large complexity. In general there is no known strongly polynomial time algorithm for solving linear programs, finding a point in the feasible region or redundancy detection. However in the special case where the LP has at most two variables per inequality, Hochbaum and Naor showed that there is a strongly polynomial algorithm to find a feasible solution. Their result makes use of some nice properties of systems with two variables per inequality and uses them together with an efficient implementation of the Fourier-Motzkin method. Using the result of Hochbaum and Naor we modify Clarkson's method, such that it detects all redundancies in strongly polynomial time. This algorithm, as the original Clarkson method, relies on geometrical notions such as ray shooting.

Finally, we consider the following alternative definition of redundant inequalities. We minimize the objective function with respect to a subset R of the inequalities of the LP, and obtain a solution that satisfies some of the inequalities (in particular R). An inequality that does not satisfy this solution is called a violator of R. In this setting an inequality is redundant w.r.t. R if it is not a violator of R. From the Sampling Lemma it is known that for a random sample of large enough sublinear size, we only expect a sublinear number of violators, which means that almost all inequalities are redundant w.r.t. R.

In this thesis we consider the following variant of sampling. After choosing a random sample we remove k elements with respect to any deterministic or random rule. Is it then still true that the number of violated inequalities is small? We give two different upper bounds which show that the maximum increase possible is a multiplicative factor of $O(\ln r + k)$ or $O(d^{2k})$. In particular, for most relevant values of $|R|$, d and k this bound is still sublinear.

In fact, we show that those bounds hold for the much more general combinatorial setting of consistent spaces and violator spaces. Consistent spaces (a generalization of violator spaces), is a concept first

introduced in this work. For a big range of values of k, our bound improves the best previous bounds. Those have also only been known for a subfamily of violator spaces and for a specific rule of removing k elements. For both bounds we give matching lower bounds in their respective settings.

Zusammenfassung

Redundante Bedingungen zu finden und zu entfernen, ist eine grundlegende Fragestellung in der Optimierung. Für das Verständnis des zugrundeliegenden Problems und die Verbesserung der Laufzeit von nachfolgenden Berechnungen kann es hilfreich sein, Redundanzen zu finden. Wir betrachten Probleme, die sich durch eine Menge von Bedingungen (constraints) beschreiben lassen, die nicht alle zur Beschreibung der Lösung notwendig sind; solche überflüssigen Bedingungen werden redundant genannt.

Im Folgenden liegt der besondere Fokus auf linearen Programmen (LPs) mit d Variablen. Die linearen Programme sind durch eine zu maximierende lineare Funktion gegeben, welche ein System von n Ungleichungen erfüllen muss. Die Lösungsmenge, also der Bereich der Punkte, die alle Ungleichungen erfüllen, definiert ein konvexes Polyeder in $\mathbb{R}^d$. Eine Ungleichung des linearen Systems wird *redundant* genannt, wenn das LP nach ihrer Entfernung immer noch die gleiche Lösungsmenge hat. Andernfalls ist die Ungleichung nichtredundant. Der Algorithmus von Clarkson ist nach heutigem Forschungsstand die schnellste Methode zur Bestimmung aller Redundanzen: Sie löst n lineare Programme mit jeweils höchstens s Ungleichungen, wobei s die Anzahl nichtredundanter Ungleichungen bezeichnet. Zusätzlich führt der Algorithmus s Ray-Shooting-Operationen aus. Dieser Algorithmus ist sehr effizient, wenn die Anzahl der nichtredundanten Ungleichungen klein ist.

Zuerst modifizieren wir Clarksons Algorithmus so, dass er nicht mehr von geometrischen Prinzipien wie der Ray-Shooting-Operation abhängt. Wir zeigen: Um alle Redundanzen zu entdecken, genügt die rein kombinatorische Information eines Dictionarys, das ist eine Matrix, welche die relativen Positionen der Ungleichungen zueinander kodiert. Unser Algorithmus ist zwar langsamer als der Algorithmus von Clarkson, aber immer noch output-sensitiv, das bedeutet, die Laufzeit hängt von der Anzahl nichtredundanter Ungleichungen ab. Ferner benutzt unser Algorithmus nur die minimal notwendige Information um die Redundanzen zu entdecken: Für eine exakte Implementierung müssen nur

die Vorzeichen richtig evaluiert werden. Zudem lässt sich unser Algorithmus auf orientierte Matroide natürlich erweitern. Wenn alle Ungleichungen in allgemeiner Lage sind, entspricht die Laufzeit im Grunde der Laufzeit von Clarksons Algorithmus.

Als Zweites betrachten wir LPs, in denen jede Ungleichung höchstes zwei Variablen hat. In diesem Fall haben wir einen Algorithmus um alle Redundanzen zu entdecken, mit streng polynomieller Laufzeit (die Anzahl Elementaroperationen ist polynomiell in n und d). Obwohl in diesem Fall die Struktur einfacher ist als bei einem allgemeinen LP, kann das zugrundeliegende Polyeder immer noch eine hohe Komplexität aufweisen. Im Allgemeinen sind für folgende Probleme keine Algorithmen mit streng polynomieller Laufzeit bekannt: Lösen eines linearen Programms, Finden einer zulässigen Lösung und Entdecken der Redundanzen. Jedoch haben Hochbaum und Naor im Fall von zwei Variablen pro Ungleichung gezeigt, dass es einen Algorithmus mit streng polynomieller Laufzeit gibt, welcher eine zulässige Lösung findet. Ihr Resultat benutzt einige schöne Eigenschaften von Systemen mit zwei Variablen pro Ungleichung und verbindet diese mit einer effizienten Implementierung der Fourier-Motzkin-Methode. Wir benutzen dieses Resultat von Hochbaum und Naor, um Clarkons Algorithmus so zu modifizieren, dass er alle Redundanzen in stark polynomieller Laufzeit endeckt. Wie der ursprüngliche Algorithmus von Clarkson, basiert dieser Algorithmus auf geometrischen Prinzipien wie Ray-Shooting.

Zum Schluss betrachten wir die folgende alternative Definition von redundanten Ungleichungen: Wir maximieren die Zielfunktion bedingt auf eine Teilmenge R der Ungleichungen. So erhalten wir eine Lösung die einige Ungleichungen erfüllt (insbesondere R). Eine Ungleichung welche die Lösung nicht erfüllt, wird als Violator von R bezeichnet. In diesem Zusammenhang heisst eine Ungleichung redundant bezüglich R, falls sie kein Violator von R ist. Wählen wir ein genügend grosses zufälliges sublineares Sample, so wissen wir vom Sampling Lemma, dass wir nur sublinear viele Violators erwarten. Das bedeutet, dass fast alle Ungleichungen redundant bezüglich R sind.

In dieser Arbeit betrachten wir die folgende Variante von Sampling.

Wir wählen ein zufälliges Sample und entfernen anschliessend k Elemente bezüglich einer beliebigen deterministischen oder zufälligen Regel. Trifft es zu, dass die Anzahl nicht erfüllter Ungleichungen immer noch klein ist? Wir geben zwei verschiedene obere Schranken an und zeigen, dass die Anzahl höchstens um einen multiplikativen Faktor von $O(\ln r + k)$ oder $O(d^{2k})$ zunimmt. Insbesondere ist diese Schranke für die meisten relevanten Werte von $|R|$, d und k immer noch sublinear.

Genau genommen zeigen wir, dass diese Schranken in einem viel allgemeineren Rahmen gelten (Consistent Spaces und Violator Spaces). Die Consistent Spaces (eine Verallgemeinerung von Violator Spaces) werden erstmals in dieser Arbeit eingeführt. Für einen grossen Bereich von Werten von k verbessern unsere Schranken die besten bisher bekannten Schranken. Diese waren auch nur für eine Unterfamilie von Violator Spaces und einer speziellen Regel zur Entfernung der k Elemente bekannt. Für beide Schranken geben wir passende untere Schranken an.

Acknowledgments

This thesis would not have been written without the support and help of several people throughout the whole period of development. First and foremost, I would like to express my gratitude to my supervisors Prof. Komei Fukuda and Prof. Bernd Gärtner. With their guidance and assistance throughout the years, they challenged and supported me and therefore made this thesis possible.

My appreciation goes to Prof. Emo Welzl, who supervised my Master and Bachelor theses and who welcomed me into his research team.

I would like to thank my coauthors Prof. Komei Fukuda, Prof. Bernd Gärtner, and Dr. Johannes Lengler for their ideas, help and the successful completion of the work. I also express my appreciation to my Master thesis advisor Dr. Anna Gundert, thanks to her I found deep interest in research and published my first paper.

I am thankful to the committee members Dr. Kenneth Clarkson, Prof. Komei Fukuda, Prof. Bernd Gärtner and Prof. Rico Zenklusen. I would also like to address my gratitude to the Swiss National Science Foundation (SNF) for their financial support.

I am grateful to my parents and my brother for their constant support throughout this thesis and my whole life. Finally, I would like to express my deepest gratefulness to Luis Barba for his unconditional emotional, moral, and professional support in all aspects of my life.

Chapter 1

Overview

1.1 Motivation

The detection and removal of redundant constraints is an important tool in optimization. Intuitively, we are given a system of constraints that describe some solutions. The constraints that are not needed to define the solutions are redundant in the system. Models of realistic data often contain redundancies. It that case identifying them helps in understanding the given data. Moreover, removing the redundancies can vastly improve the running time of subsequent operations.

This thesis studies different aspects of the problem of detecting redundancies in linear programs. We consider *linear systems (of inequalities)* of the form $Ax \leq b$, where $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$. We call the rows of $Ax \leq b$ the *constraints* of the linear system. A point $x \in \mathbb{R}^d$ that satisfies the constraints of the system is called a *(feasible) solution*. The set of feasible solutions is called the *feasible region* and forms a convex *polyhedron*, since it is defined as the intersection of halfspaces given by the linear constraints. A bounded polyhedron is called a *polytope*. In a linear system $Ax \leq b$ an inequality is called *redundant*, if after its removal the system still has the same feasible region. If an inequality is not redundant it is called *nonredundant*.

Before discussing the redundancy problem in more detail we give an overview of linear programming. We consider linear programs (LPs) of the form

$$\begin{array}{lll} \text{maximize} & c^T x & \\ \text{subject to} & Ax & \leq \quad b, \end{array} \tag{1.1}$$

where A, b are as above and $c \in \mathbb{R}^d$. We call $c^T x$ the *objective function* of the LP. Let us assume that the LP is *feasible*, i.e., there exists a solution to the linear system. The intersection of d linearly independent hyperplanes determined by the constraints of $Ax \leq b$, is called a *basic solution*. We always assume that a basic solution exists, in particular $d \leq n$. We can identify the basic feasible solutions with the vertices of the polyhedron defined by the LP. A feasible solution x^* that satisfies $c^T x^* \geq c^T x$ for all feasible solutions x, is called an *optimal solution*. An LP is unbounded, if for all $z \in \mathbb{R}$ there exists a feasible solution x_z, such that $c^T x_z \geq z$.

Running time of linear programming solvers. Let $LP(n, d)$ denote the time (number of elementary operations) needed to solve an LP of form (1.1). Concretely, $LP(n, d)$ is the time needed to find either an optimal solution or a certificate for unboundedness. Although practical algorithms to solve LPs exist, no strongly polynomial-time algorithm is known. We call an algorithm *strongly polynomial* if the number of elementary operations is polynomial in the number of inputs (here n and d) but independent of the encoding size (i.e., the bit size) of the input. However there are algorithms, whose running times are polynomial in the encoding size, those are simply called polynomial algorithms.

Many algorithms have been developed to solve LPs. We will review some widely known ones among them and discuss their different approaches.

The main idea of the *simplex algorithm* is as follows: in the first stage one finds a vertex x of the feasible region. In the second stage, if x is not an optimal solution, the algorithm moves to a vertex x' that has better objective value i.e., $c^T x' > c^T x$. This is done with a fixed pivot rule. This step is repeated until the algorithm finds an optimal vertex or a certificate for unboundedness.

The *criss-cross algorithm* [45, 47, 22] is similar to the simplex algorithm. It however moves between basic solutions that are not necessarily feasible and may decrease the objective value in some steps.

Although both algorithms run fast in practice, in general they can have exponential running times [13, 33].

On the other hand the *ellipsoid method* runs in polynomial time in the encoding length of the input, but is not practical [32]. The ellipsoid method finds either a feasible solution of the given LP or a certificate for infeasibility. The main idea is to find a sequence of ellipsoids $E_0, E_1, \ldots, E_k$ of decreasing volume, such that either the center of E_k is a feasible solution of the LP or E_k is a certificate for infeasibility. We will discuss below how an optimization problem can be transformed into a feasibility problem.

Karmarkar [31] introduced the *interior-point method*, considered to be the first practical polynomial time algorithm, which has been modified in many ways since [49]. We start with a linear system with bounded optimal solution and a fixed feasible point. The algorithm moves on a unique path towards a face where the objective value is maximized. Polynomially many steps suffice to find such a face and hence an optimal solution.

A completely different approach of sampling is used in the *LP solver of Clarkson* [11], which is a randomized algorithm. Its expected running time is exponential in d but linear in n, and is hence particularly useful if d is small. It uses the fact that optimizing with respect to a random sample of n^α, $0 < \alpha < 1$ many inequalities of the LP, we only expect $O(n^{1-\alpha} \cdot d)$ many inequalities of the LP to be not satisfied by the solution [28]. Hence using a sublinear sample, we expect only a sublinear number of unsatisfied constraints. As the simplex and criss-cross algorithm, it finds an optimal solution considering the basic solutions of the LP.

We can see that the above methods to solve LPs are fundamentally different. The beauty of the simplex, criss-cross and Clarkson's algorithm is, that they only consider the basic solutions of the linear system. This can easily be implemented correctly with exact rational arithmetic, but none of the running times is polynomial in general.

Equivalence of optimization and feasibility. Let $LP'(n,d)$ denote the time to solve the associated feasibility problem, i.e., the time needed to find a solution to

$$Ax \leq b. \tag{1.2}$$

As the optimal solution is also a feasible solution, $LP'(n,d) \leq LP(n,d)$. This means that the feasibility problem can be reduced to the optimization problem.

We claim that the reduction in the other direction also holds and to prove it, we use the powerful tool of the *dual* LP of (1.1), (see [40, Chapter 7]) given by

$$\begin{array}{lrcl} \text{minimize} & b^T y & & \\ \text{subject to} & A^T y & = & c \\ & y & \geq & 0. \end{array} \tag{1.3}$$

If (1.1) is feasible and bounded, then its optimal value equals the optimal value of (1.3), i.e., the maximum of $c^T x$ equals the minimum of $b^T y$. If (1.1) is unbounded, then its dual is infeasible. It follows that we can optimize (1.1) by finding a solution to the following system of inequalities.

$$\begin{array}{rcl} c^T x & = & b^T y \\ Ax & \leq & b \\ A^T y & = & c \\ y & \geq & 0. \end{array} \tag{1.4}$$

If (1.4) has a solution (x^*, y^*), then x^* is an optimal solution of (1.1). If (1.4) is infeasible, then (1.1) is unbounded. (This is because we assumed that (1.1) is feasible, without the assumption (1.1) could be infeasible as well.) We can hence solve the optimization problem (1.1) in time $LP'(n+d, n+d)$, i.e., $LP(n,d) \leq LP'(n+d, n+d)$. Therefore finding a strongly polynomial-time algorithm to solve the feasibility problem, would also imply a strongly polynomial-time algorithm for the optimization problem.

Equivalence of LP and redundancy detection. We now show how one can test redundancy of a constraint through linear programming and vice versa. Suppose we are given a system of form (1.1) and we want to know whether the k-th constraint, denoted $A_k x \leq b_k$, is redundant. We can answer this question by solving the linear program

$$\begin{array}{lrcll} \text{maximize} & A_k x & & & \\ \text{subject to} & A_i x & \leq & b_i & \forall i \in \{1, \ldots, n\} \setminus \{k\} \\ & A_k x & \leq & b_k + 1. & \end{array} \tag{1.5}$$

We show that $A_k x \leq b_k$ is nonredundant, if and only if the optimal solution of (1.5) is larger than b_k. It follows that a single redundancy can be detected in time $LP(n, d)$. To prove the claim, first note that (1.5) is feasible (since (1.1) is feasible) and the objective value is bounded by $b_k + 1$. Assume that $A_k x \leq b_k$ is redundant. In that case every solution to

$$A_i x \leq b_i \qquad \forall i \in \{1, \ldots, n\} \setminus \{k\},$$

also satisfies $A_k x \leq b_k$, in particular for the optimal solution x^* it holds that $A_k x^* \leq b_k$. For the other direction assume that $A_k x \leq b_k$ is nonredundant, i.e., there exists a solution x' that satisfies all constraints except $A_k x' \leq b_k$. In particular $A_k x' > b_k$ and hence the optimal solution x^* satisfies $A_k x^* > b_k$ as well.

The other direction can be shown as follows. Let x_0 be a new variable. Then it is not hard to show that (1.1) has no feasible solution if and only if $x_0 \leq 0$ is redundant in the linear system

$$\begin{array}{rcl} Ax - bx_0 & \leq & 0 \\ x_0 & \leq & 0. \end{array} \tag{1.6}$$

Let $x_0 \leq 0$ be redundant. This is equivalent to the statement that there is no solution (x, x_0) with $x_0 > 0$ such that $Ax \leq bx_0$. This again is equivalent to the statement that there is no feasible solution of (1.1). Hence deciding whether a constraint is redundant is at least as hard as the feasibility problem of linear programming (for more detail see [18]).

Redundancy detection algorithm. In the last paragraph we saw that we can find a single redundancy in time $LP(n, d)$. It follows that solving $n + d$ LPs of form (1.5) suffices to detect all redundancies. The currently fastest known algorithm to find all redundancies is due to Clarkson and has running time $O((n + d) \cdot LP(s, d))$ [10, 18], where s is the number of nonredundant inequalities. The algorithm still solves n linear programs, but each of them has at most s constraints. Hence in the case where $s \ll n$, this is a major improvement. We will discuss this algorithm in Section 2.2.

Redundancy from a different viewpoint. In this section, for simplicity let us assume that our LP has a bounded optimal solution. In this part we also assume that c is generic, that is, the optimal solution is unique. Clarkson's redundancy removal algorithm is very useful to preprocess a linear system, if for instance one wants to solve an LP for many objective functions c. If one is not interested in this kind of preprocessing, but only solves one linear program, we can observe that most constraints are unnecessary in the sense that removing them still results in the same optimal value. In fact there exists a set of d constraints with index set I, such that

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & A_i x \leq b_i & \forall i \in I \end{array} \tag{1.7}$$

has the same optimal value as (1.1). We call the constraints $i \notin I$ *redundant w.r.t. direction c.*

If one can find an optimal solution, then by basic linear algebra it is not hard to find a set of d constraints whose boundaries intersect in this optimal solution. Hence up to some additional polynomial factor, finding the nonredundant constraints w.r.t. c, takes at least as long as finding an optimal solution to (1.1). Let us go a step further and see what happens if we sample a subset $A' \subset A$, of say $\sqrt{n}$ constraints at random. Assume that $c^T x$ subject to the constraints in A' has an optimal solution. As before a constraint $h \in A \setminus A'$ is redundant w.r.t. c in A' if adding it does not change the optimal solution. We call the constraints whose addition changes the optimal solution the *violators.*

This is a useful notion as using the so-called Sampling Lemma (see Theorem 6.2.8), one can show that there are only $O(\sqrt{n} \cdot d)$ violators in expectation w.r.t. the sample A' [9, 28] . Using the Sampling Lemma, Clarkson showed that a linear program can be solved in expected time linear in n but exponential in d [11]. In general sampling is a powerful tool in linear programming.

Since the above results consider only the combinatorial aspect of linear programming, (we are only interested in the violators but not their exact positions,) it is often easier to consider generalizations of LPs such as LP-type problems [41], or even more general violator spaces [26]. A violator space can be thought of as an LP without objective function: each subset $A' \subset A$, is associated with a set of violators instead of the optimal value. This often simplifies the problem analysis, since the objective function is replaced by the combinatorial notion of violators. Brise and Gärtner showed that Clarkson's algorithm to solve an LP can be generalized (and simplified) for violator spaces [4].

1.2 Problems and Goals

In the first part of this thesis, we study Clarkson's redundancy removal algorithm (see Section 2.2) in more detail. The original version [10, p.696] solves the problem where given a set of n points in $\mathbb{R}^d$, one can find the set of extreme points. We consider the equivalent dual version of the problem, where given a system $Ax \leq b$, we want to detect all redundant constraints [18]. Here we use the notion of duality between points and planes, as discussed in Section 2.2. Clarkson's algorithm depends strongly on geometrical notions such as ray shooting (or scalar products in the original version), that need exact computations in order to guarantee correctness. Moreover, small perturbations of the system may lead to different behavior of the algorithm, even if the underlying polyhedron stays combinatorially the same (the same faces stay adjacent). Our goal is to modify this algorithm, such that it is purely combinatorial. The combinatorial algorithm should only rely on the relative positions between the basic solutions and the constraints

and should only need to compute signs exactly. In particular, it should run equally on two polyhedra that are combinatorially equivalent.

During Clarkson's redundancy removal algorithm we need to solve LPs with running time $LP(s, d)$, for which in general no strongly polynomial time algorithm is known. Thus we would also like to detect some special cases where Clarkson's algorithm runs in strongly polynomial time. We study a family of LPs for which strongly polynomial time algorithms for the feasibility problem exist. Using these tools, we modify Clarkson's algorithm for redundancy detection to run in strongly polynomial time for any instance in this restricted family.

Finally, in the second part of this thesis we turn our attention to a powerful technique used in algorithms to solve LPs, namely random sampling. We investigate this notion in the setting of LPs and their combinatorial generalizations. Moreover, we study the variation of "quality" of a sample after some of its most restrictive elements are removed. In the next section we describe in more details each of the results mentioned above.

1.3 Main Results

This is an accumulative thesis, where the main results are based on the following papers.

"Combinatorial Redundancy Detection" by K. Fukuda, B. Gärtner and M. Szedlák [19] .

"Redundancies in Linear Systems with two Variables per Inequality" by K. Fukuda and M. Szedlák [20].

"Sampling with Removal" by B. Gärtner, J. Lengler and M. Szedlák [25].

Part I: Modifications of Clarkson's Redundancy Removal Algorithm. In this part we study two different modifications of Clarkson's redundancy removal algorithm. As mentioned above, the algo-

rithm relies on ray shooting, which is a purely geometrical notion. Moreover, depending on the input, floating point errors or slow performance can easily occur. The detailed description of this algorithm is given in Section 2.2. On the other hand, simplex-type algorithms rely only on the finitely many dictionaries of the LP. A dictionary can be thought of as matrix with entries that encode the relative position of each basic solution of the LP with respect to the linear constraints that define the LP (refer to Chapter 3 for a formal definition). In fact, knowing only the signs of these entries suffices to run simplex-type algorithms, hence we only need exact computations of signs.

In Chapter 3 we show that knowing all of the finitely many dictionaries of the LP is sufficient for the purpose of redundancy detection. More generally, we show that it is enough to know only the signs of the entries of the dictionary, which makes the algorithm purely combinatorial. Concretely, we show that for any inequality one can find a dictionary, such that its sign pattern is either a redundancy or nonredundancy certificate for this inequality (see Theorem 3.4.3 and Theorem 3.4.5).

Furthermore, we show that considering only the sign patterns of the dictionary, there is an output-sensitive algorithm of running time

$$O(d \cdot (n+d) \cdot s^{d-1} \cdot LP(s,d) + d \cdot s^d \cdot LP(n,d))$$

to detect all redundancies (see Theorem 3.5.2). Recall that s denotes the number of nonredundant constraints. In the case where all constraints satisfy some non-degeneracy assumptions, the running time can be reduced to

$$O(s \cdot LP(n,d) + (n+d) \cdot LP(s,d)),$$

which is essentially the running time of the Clarkson method. Our algorithm extends naturally to a more general setting of arrangements of oriented topological hyperplane arrangements [3, Chapter 10].

Although Clarkson's algorithm is output sensitive, its running time depends on $LP(s,d)$, for which there is no known strongly polynomial-time algorithm. In Chapter 4 we give a strongly polynomial-time algorithm for the special case where every constraint has at most two

variables with nonzero coefficients. We denote this family of linear systems by $LI(2)$. Due to its easy structure, this family has some nice properties that are not known to be generalizable. In particular Aspvall and Shiloach [2] showed that given a variable x_i and a value λ, we can test in time $O(nd)$ whether there is a feasible solution with $x_i = \lambda$.

An elegant algorithm for solving the feasibility problem in $LI(2)$ in $O(d^2 n \log n)$ time was presented by Hochbaum and Naor [29]. Their algorithm uses an efficient version of the Fourier-Motzkin elimination method and the aforementioned result by Aspvall and Shiloach. Although the feasibility problem can be solved in strongly polynomial time, it is not known whether there is a strongly polynomial-time algorithm to optimize in $LI(2)$. As the dual of a LP in $LI(2)$ is in general not in $LI(2)$, we cannot turn an optimization problem into a feasibility problem, using the technique mentioned above.

We present a strongly polynomial time algorithm that solves redundancy detection in time $O(nd^2 s \log s)$ (see Theorem 5.3.5). It uses a modification of Clarkson's algorithm, together with a revised version of Hochbaum and Naor's technique. Finally we show that dimensionality testing can be done with the same running time as solving feasibility. In general we need to solve up to d LPs for the feasibility problem to decide the dimensionality of a polyhedron (see Section 2.2.2).

In Chapter 5, we discuss the complexity of polytopes in $LI(2)$. In general it is known that given n constraints, the dual cyclic polytope maximizes the number of vertices. Using polytopes that were first introduced in [1], we show that polytopes in $LI(2)$ can have high complexity as well: they differ from the dual cyclic polytope by a factor exponential in d but independent of n (see Theorem 5.3.5). Hence if d is constant, there are polytopes in $LI(2)$ that have asymptotically the same complexity as the dual cyclic polytope. We also show, that if $d \geq 4$, no polytope in $LI(2)$ can exactly achieve the complexity of the dual cyclic polytope (Theorem 5.4.2).

Part II: Sampling with Removal in Generalizations of Linear Programming. Random sampling is an often used, important tool

in optimization. The hope is that the optimal solution subject to a small subset of random constraints approximates the global optimal solution well, i.e., in the setting of LPs there are only a few unsatisfied constraints.

For instance, in the context of linear programming the Sampling Lemma says that the optimal solution of a random sample of n^α, $0 < \alpha < 1$ constraints only violates $O(n^{1-\alpha} \cdot d)$ constraints in expectation [11]. We call a constraint *violated* if it is not satisfied. From the Sampling Lemma we can observe that even with a small random sample, we only expect a small number of violators. In this thesis we want to study the robustness of sampling. Consider the following variant of sampling. After sampling a subset of the constraints at random, we remove a fixed number of constraints from the sample following some arbitrary rule. The question is how many constraints can be violated after such a removal. Is the number still small?

This question is natural if we want to approximate the solution of the original problem with the sampled constraints. For example, consider the following geometric problem. Say we are given a set of points in $\mathbb{R}^d$, and we want to find a ball that contains most of the points and whose radius is small. The Sampling Lemma allows us to find such a ball, by finding the smallest ball that contains a set of randomly sampled points. If we remove a fixed number of outliers from this sample, can we still guarantee to have only a few points outside the smallest ball that contains the remaining points?

The problem of sampling with removal was originally motivated by *chance-constrained optimization*. In that setting we have a probability distribution over a finite or infinite set of constraints. Using a finite random sample, we can find a solution that satisfies a random constraint with high probability [5]. In this setting bounds for the tradeoff between the solution quality and violation probability are given [6].

We can summarize our results in the setting of linear programs as follows. Let us assume that our sample is drawn uniformly at random from the family of all sets of constraints of size n^α. We prove upper bounds for the expected number of constraints violated by a random

sample after the removal of k elements by any arbitrary rule. More specifically, we show that expected number of violated constraints after this removal can be bounded by $O(n^{1-\alpha} \cdot (d \ln r + k))$ and $O(n^{1-\alpha} \cdot d^{2k+1})$ (see Theorem 6.3.4 and Theorem 6.4.7). Using some nondegeneracy assumptions, the second bound improves to $O(n^{1-\alpha} \cdot d^{k+1})$ (see Corollary 6.4.8). Comparing against the bounds of the Sampling Lemma, our results imply that the number of violated constraints can increase at most by a factor of either $O(\ln r)$ or $O(d^{2k})$ after the removal. The second result is stronger iff $d^{2k} = o(\ln r)$, hence in particular for constant d and k.

In fact all our results hold for violator spaces (Definition 6.2.4), a combinatorial generalization of linear programs. The first bound mentioned even holds for a completely abstract setting that generalizes violator spaces (and LP), where we assign to each subset R of the constraints an arbitrary set $V(R)$ of constraints disjoint from R; For both results we provide matching lower bounds in their respective settings (Section 6.5 and Section 6.6).

Our bounds improve the previously best bounds which had only been known for a subfamily of violator spaces and a specific rule for removal [23].

Part I

Generalizations of Clarkson's Redundancy Removal Algorithm

Chapter 2

Preliminaries

2.1 Redundancy Removal in Linear Programming

We consider linear systems of inequalities of form $Ax \leq b$, for $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$, $d < n$. A linear program in standard form is given by a linear objective function to maximize (or minimize), subject to a linear system of inequalities, i.e.,

$$\begin{array}{lrcl} \text{maximize} & c^T x & & \\ \text{subject to} & Ax & \leq & b \end{array} \tag{2.1}$$

Let $LP(n, d)$ denote the time needed to solve an LP with n inequalities and d variables, with some linear objective function $c^T x$. There are many ways to solve a linear program, but there is no known strongly polynomial algorithm. Although the simplex algorithm runs fast in practice, in general it can have exponential running time [13, 33]. On the other hand, the ellipsoid method runs in polynomial time on the encoding of the input size, but is not practical [32]. A first practical polynomial time algorithm, the interior-point method, was introduced in [31].

The j-th constraint, denoted $A_j x \leq b_j$, is called *redundant*, if its removal does not change the set of feasible solutions. We also say that the index j is redundant if the corresponding constraint is redundant. By removing $A_j x \leq b_j$ from the system we get a new system denoted $A_{[n]\setminus\{j\}} x \leq b_{[n]\setminus\{j\}}$, where $[n] = \{1, \cdots, n\}$. Solving the following lin-

ear program (LP) we can decide redundancy of $A_j x \leq b_j$.

$$\begin{array}{lrcl} \text{maximize} & A_j x & & \\ \text{subject to} & A_{[n]\setminus\{j\}} x & \leq & b_{[n]\setminus\{j\}} \\ & A_j x & \leq & b_j + 1. \end{array} \tag{2.2}$$

As shown in the overview, a constraint $A_j x \leq b_j$ is redundant if and only if the optimal solution has value at most b_j.

Solving n linear programs of form (2.2), each with running time $LP(n, d)$, is enough for detecting all redundancies. The currently fastest method is due to Clarkson with running time $O(n \cdot LP(s, d))$ [10, p. 696], where we initially assume an interior point is given. Without loss of generality we can assume that such a point is given, since it can be found by solving $O(d)$ linear programs (we will discuss this in Section 2.2.2). Clarkson's method solves n linear programs, but each of them has at most s variables, where s is the number of nonredundant constraints. Hence, if $s \ll n$, this *output-sensitive* algorithm is a major improvement. We will discuss this algorithm in detail in the following section.

As already mentioned, Clarkson's original redundancy detection algorithm finds the convex hull of n points in $\mathbb{R}^d$. We discuss how this is equivalent to the redundancy detection problem. Assume we are given a set P of n points in $\mathbb{R}^d$. Without loss of generality assume that P spans $\mathbb{R}^d$ and that the origin is in the interior of the convex hull. By the unit sphere point-hyperplane duality we map every point $p \in P$ to a hyperplane p^* as follows. The plane p^* is orthogonal to the line from the origin to p and has distance $1/\|p\|$ from the origin. The plane, given by equalities is now changed to halfspaces (inequalities), by orienting the equalities such that each of them contains the origin. We obtain a system of the form $Ax \leq b$, and it can be shown that the points of the convex hull of P correspond to the nonredundant constraint in $Ax \leq b$. One can also easily reverse this construction: given a system $Ax \leq b$ with an interior point, we can use the above duality to construct a convex hull problem.

2.2 Clarkson's Redundancy Removal Algorithm

In this section, we present Clarkson's redundancy removal algorithm, which we will modify in the following two chapters. The following two sections are based on [18].

As before assume we are given a linear system in standard form by

$$Ax \leq b. \tag{2.3}$$

We assume that the solution space of (2.3) is full-dimensional. We can always assume this after appropriate preprocessing, as we will discuss in Section 2.2.2.

A constraint $A_i x \leq b_i$ is called redundant w.r.t. $S \subseteq [n]$ if it is redundant in the system induced by i and S, i.e.,

$$\begin{array}{rcll} A_i x & \leq & b_i & \\ A_j x & \leq & b_j & \forall j \in S \setminus \{i\}. \end{array} \tag{2.4}$$

We can test whether $A_i x \leq b_i$ is redundant w.r.t. S by solving the following linear program denoted $Red(S, i)$.

$$\begin{array}{lrcll} \text{maximize} & A_i x & & & \\ & A_i x & \leq & b_i + 1 & \\ & A_j x & \leq & b_j & \forall j \in S \setminus \{i\}. \end{array} \tag{2.5}$$

Observe that $A_i x \leq b_i$ is nonredundant w.r.t. S if and only if (2.5) has optimal value larger than b_i.

2.2.1 The Algorithm

Theorem 2.2.1. *[10] Let z be an interior point solution of the system (2.3) i.e., z satisfies $Az < b$, where " $<$ " denotes the componentwise strict inequality. Let $z^\epsilon = z + (\epsilon, \epsilon^2, \ldots, \epsilon^d)^T$ for ϵ small enough, i.e., z^ϵ is a generic interior point solution. Then the following algorithm returns the indices of the nonredundant constraints.*

Algorithm Clarkson (A,b,z);
begin
$R := \emptyset, S := \emptyset$;
while $R \cup S \neq [n]$ **do**
pick any $r \in [n] \setminus (R \cup S)$ *and check whether* r *is redundant w.r.t.* S *using* $Red(S, r)$;
if r *is redundant w.r.t.* S **then**
$R = R \cup \{r\}$;
else r *is nonredundant w.r.t.* S
Let x^* *be the solution found to the LP* $Red(S, r)$;
$S = S \cup \{q\}$, *where* $q = RayShoot(A, b, z^\epsilon, x^* - z^\epsilon)$;
endif;
endwhile;
output S;
end.

The function $RayShoot(A, b, z, t)$ *returns the index* q *of the hyperplane* $\{x : A_q x = b_q\}$, *which is hit first by the ray starting at* z *along the direction of* t.

Note that q, the index of the facet hit by RayShoot$(A, b, z^\epsilon, x^* - z^\epsilon)$ is unique, since z^ϵ is generic. Moreover, since the ray shooting from inside the feasible region to the outside, q is nonredundant.

Proof. Let S^* be the indices of the nonredundant constraints and R^* the indices of the redundant constraints. We show by induction that in every step $R \subseteq R^*$ and $S \subseteq S^*$. Moreover we show that in every round one new element is added to R or S. This shows the running time of the algorithm, as in each round we solve an LP with at most s constraints ($Red(S, r)$), hence with running time at most $LP(s, d)$. As in each round we add one element to R or S we solve n linear programs of this form. With every ray shooting we add a new constraint to S, hence there are at most s ray shootings which take time $O(s \cdot n \cdot d)$ in total. The running time follows.

It remains to show the correctness of the algorithm. In the beginning we trivially have $R \subseteq R^*$ and $S \subseteq S^*$. Now suppose in some round of the algorithm it holds that $R \subseteq R^*$, $S \subseteq S^*$ and $R \cup S \neq [n]$. We pick an index $r \in [n] \setminus (R \cup S)$. If r is redundant w.r.t. $S \subseteq S^*$, then it is also redundant w.r.t. S^*. Therefore we add a new constraint to R i.e. $R = R \cup \{r\}$. If r is nonredundant w.r.t. S, we do not know whether r is redundant or nonredundant w.r.t. S^*. Let q be the index of the facet-inducing hyperplane $\{x : A_q x = b_q\}$, which is hit first by the ray starting at z through x^*. Since z is a generic solution of (2.3), q is unique. Moreover because $A_r x^* > b_r$, x^* is not a solution of the linear system (2.3) and therefore q must be nonredundant. It also holds that $q \notin S$, since $A_i x^* \leq b_i$ for all $i \in S$, but $A_q x^* > b_q$. Hence a new nonredundant constraint is added to S. □

2.2.2 Preprocessing of the Algorithm

In this section we discuss how to preprocess appropriately for Clarkson's algorithm, so that we can assume that the feasible region is full-dimensional. We show how to find a relative interior point of the feasible region. The relative interior point is defined as follows. For a system $Ax \leq b$ let $I \subseteq [n]$ be the indices of the inequalities that are forced to equality by any solution. That is, i is in I if for all feasible solutions x' of $Ax \leq b$ it holds that $A_i x' = b_i$. A feasible solution x^* of $Ax \leq b$ is a *relative interior point* if $A_j x < b_j$ for all $j \notin I$.

Given a relative interior point we find all inequalities that are forced to equalities and one can assume full-dimensionality of the system.

Theorem 2.2.2. *Suppose we are given a linear system with n inequalities and d variables of form*

$$Ax \leq b. \tag{2.6}$$

We can detect whether the LP is feasible in time $LP(n, d)$. Suppose that the feasible region of $Ax \leq b$ has dimension $0 < k \leq d$. Then we can find a relative interior point in time

$$O((d - k + 1) \cdot LP(n, d) + (d - k) \cdot LP(d, n)).$$

Proof. Consider the system

$$\begin{array}{lrcl} \text{maximize} & \epsilon & & \\ \text{subject to} & Ax + \mathbf{1} \cdot \epsilon & \leq & b \\ & \epsilon & \leq & 1, \end{array} \tag{2.7}$$

where $\mathbf{1}$ denotes the all-ones vector.

Let (x^*, ϵ^*) be an optimal solution of (2.7). If $\epsilon > 0$, then x^* is an interior solution point of (2.6) and therefore the dimension of the solution space is d. If $\epsilon^* < 0$, then (2.6) is infeasible.

Therefore let us assume that $\epsilon^* = 0$. The dual of (2.7) is given by

$$\begin{array}{lrcl} \text{minimize} & b^T y + z & & \\ \text{subject to} & A^T y & = & 0 \\ & \mathbf{1}^T y + z & = & 1 \\ & y, z & \geq & 0. \end{array} \tag{2.8}$$

Let (y^*, z^*) be an optimal solution of this dual system. By strong duality the optimal solution has also value zero, i.e., $b^T y^* + z^* = 0$. Since additionally $\mathbf{1}^T y^* + z^* = 1$, it follows that y^* can not be totally zero. Let $I = \{i \mid y_i^* > 0\}$, the indices for which y^* is not zero. By complementary slackness (see e.g. [40, Chapter 7.9]) we know that for any feasible solution (x, ϵ) with $\epsilon = 0$ it must hold that $A_I x = b_I$, i.e., all inequalities in I are forced to equality. Using Gauss' elimination algorithm we can find the index set $J \supseteq I$ of all inequalities forced to equality, given that $A_I x = b_I$. We now recurse our procedure on the system

$$\begin{array}{lrcl} \text{maximize} & \epsilon & & \\ \text{subject to} & A_J x & = & b_J \\ & A_{[n]\setminus J} x + \mathbf{1} \cdot \epsilon & \leq & b_{[n]\setminus J} \\ & \epsilon & \leq & 1. \end{array} \tag{2.9}$$

This system (and hence the dual) has nonnegative optimal value. If it is positive, the solution x^* is a relative interior point. Otherwise we repeat the procedure on the new system that has lower dimension. Since the dimension of the linear system decreases by at least one, we need at most $(d - k + 1)$ repetitions of this step. The running time follows. □

Chapter 3

Combinatorial Redundancy Removal

This chapter is based on [19] by K. Fukuda, B. Gärtner and M. Szedlák.

3.1 Introduction

In this chapter, we focus on redundancies in linear systems. We consider systems of the form

$$\begin{array}{rcl} x_B & = & b - Ax_N \\ x_B & \geq & 0 \\ x_N & \geq & 0 \end{array} \tag{3.1}$$

where B and N are disjoint finite sets of variable indices with $|B| = n$, $|N| = d$, $b \in \mathbb{R}^B$ and $A \in \mathbb{R}^{B \times N}$ are given input vector and matrix. We assume that the system (3.1) has a feasible solution. Any consistent system of linear equalities and inequalities can be reduced to this form. In particular any system of the form $Ax \leq b$, as given in the previous chapters, can be reduced to this form by adding $O(n)$ slack variables.

In this setting, a variable x_r is called *redundant* in (3.1) if $x_B = b - Ax_N$ and $x_i \geq 0$ for $i \in B \cup N \setminus \{r\}$ implies $x_r \geq 0$, i.e., if after removing constraint $x_r \geq 0$ from (3.1) the resulting system still has the same feasible region. Testing redundancy of x_r can be done by solving the

linear program (LP)

$$\begin{array}{lrcl} \text{minimize} & x_r & & \\ \text{subject to} & x_B & = & b - Ax_N \\ & x_i & \geq & 0 \quad \forall i \in B \cup N \setminus \{r\} \\ & x_r & \geq & -1. \end{array} \tag{3.2}$$

Namely, a variable x_r is redundant if and only if the LP has an optimal solution and the optimal value is nonnegative.

With abuse of notation let $LP(n, d)$ denote the time needed to solve an LP of form (3.2). Throughout, we are working in the real RAM model of computation, where practical algorithms, but no polynomial bounds on $LP(n, d)$ are known. However, our results translate to the standard Turing machine model, where they would involve bounds of the form $LP(n, d, \ell)$, with ℓ being the bit size of the input. In this case, $LP(n, d, \ell)$ can be polynomially bounded. The notation $LP(n, d)$ abstracts from the concrete representation of the LP, and also from the algorithm being used; as a consequence, we can also apply it in the context of LPs given by the signs of their dictionaries. A dictionary can be thought of as an enriched encoding of the vertices of the polyhedron defined by the LP (see Section 3.2.1).

By solving $n + d$ linear programs, $O((n + d) \cdot LP(n, d))$ time is enough to detect all redundant variables in the real RAM model, but it is natural to ask whether there is a faster method. As already mentioned, the currently fastest practical method is the one by Clarkson, with running time $O((n + d) \cdot LP(s, d) + s \cdot n \cdot d)$ [10]. This case arises quite naturally. For example, when one needs to compute a projection of a polyhedron, one natural method is Fourier-Motzkin elimination. The method is known to generate a large number of redundant constraints in each step (a quadratic increase in the worst case) and it is essential to remove redundant constraints frequently for any practical implementation. (The reader is referred to [40, pp. 155-156] for more details).

Specialized (and output-sensitive) algorithms for the extreme points problem exist [38, 14], but they are essentially following the ideas of

Clarkson's algorithm [10]. For fixed d, Chan uses elaborate data structures from computational geometry to obtain a slight improvement over Clarkson's method [7].

In this chapter, we study the *combinatorial* aspects of redundancy detection in linear systems. The basic questions are: What kind of information about the linear system do we need in order to detect all redundant variables? With this restricted set of information, how fast can we detect all of them? Our motivation is to explore and understand the boundary between geometry and combinatorics with respect to redundancy. For example, Clarkson's original method [10, p. 696] to find the vertices of the convex hull of a set of points uses scalar products, an intrinsically geometric procedure. Similarly, the dual algorithm that we presented in Chapter 2 uses ray shooting, another geometric procedure. In a purely combinatorial setting, neither ray shooting nor scalar products are well-defined notions, so it is natural to ask whether we can do without them.

We will show that our results solely depend on the finite combinatorial information given by the signed dictionaries, i.e., the size is bounded by a function of d and n only. As already mentioned, a dictionary can be thought of as an encoding of the associated arrangements of hyperplanes, the corresponding signed dictionary only contains the signs of the encoding (see Section 3.2). On the other hand Clarkson's algorithm depends on the exact values of the input data A and b.

Our approach is very similar to the combinatorial viewpoint of linear programming pioneered by Matoušek, Sharir and Welzl [34] in form of the concept of *LP-type problems*. The question they ask is: how quickly can we *optimize*, given only combinatorial information? As we consider redundancy detection and removal as important towards efficient optimization, it is very natural to extend the combinatorial viewpoint to also include the question of redundancy. The results that we obtain are first steps and leave ample space for improvement. An immediate theoretical benefit is that we can handle redundancy detection in structures that are more general than systems of linear inequalities; most notably, our results naturally extend to the realm of *oriented matroids* [3, Chapter 10] [21]. An oriented matroid can be thought

of as an arrangement of oriented topological halfspaces, (also called pseudo halfspaces). It can be encoded in the same manner with their associated dictionaries and the following methods extend naturally.

Statement of Results. First of all, we note that for the purpose of redundancy testing, it is sufficient to know all the finitely many dictionaries associated with the system of inequalities (3.2). Moreover, we show that it is sufficient to know only the *signed* dictionaries, i.e., the *signs* of the dictionary entries. Their actual numerical values do not matter.

In Theorem 3.4.3, we give a characterization of such a redundancy certificate. More precisely, we show that, for every redundant variable x_r there exists at least one signed dictionary such that its sign pattern is a redundancy certificate of x_r. Similarly, as shown in Theorem 3.4.5, for every nonredundant variable there exists a nonredundancy certificate. An alternative nonredundancy certificate is given in Theorem 3.7.1. Such a single certificate can be detected in time $LP(n, d)$ (see Section 3.4.3). The number of dictionaries needed to detect *all* redundancies depends on the LP and can vary between constant and linear in $n + d$ (see Section 3.6).

In a second part, we present a Clarkson-type, output-sensitive algorithm that detects all redundancies in running time $O(d \cdot (n + d) \cdot s^{d-1} LP(s, d) + d \cdot s^d \cdot LP(n, d))$ (Theorem 3.5.2). Under some general position assumptions the running time can be improved to $O((n + d) \cdot LP(s, d) + s \cdot LP(n, d))$, which is basically the running time of Clarkson's algorithm. In these bounds, $LP(n, d)$ denotes the time to solve an LP to which we have access only through signed dictionaries. As in the real RAM model, no polynomial bounds are known, but algorithms that are fast in practice exist.

In general our algorithm's running time is worse than Clarkson's, but it only requires the combinatorial information of the system and not its actual numerical values. If the feasible region is not full-dimensional (i.e. not of dimension d), then a redundant constraint may become nonredundant after the removal of some other redundant constraints.

To avoid these dependencies of the redundant constraints we assume full-dimensionality of the feasible region. Because of our purely combinatorial characterizations of redundancy and nonredundancy, our algorithm works in the combinatorial setting of oriented matroids [3], and can be applied to remove redundancies from oriented topological hyperplane arrangements.

3.2 Basics

Before discussing redundancy removal and combinatorial aspects in linear programs, we fix the basic notation on linear programming — such as dictionaries and pivot operation — and review finite pivot algorithms. (For further details and proofs see e.g. [8, Part 1], [16, Chapter 4].)

3.2.1 LP in Dictionary Form

Throughout this section, if not stated otherwise, we always consider *linear programs* (LPs) of the form

$$\begin{array}{lrcl} \text{minimize} & c^T x_N & & \\ \text{subject to} & x_B & = & b - A x_N \\ & x_E & \geq & 0, \end{array} \tag{3.3}$$

where $E := B \cup N$, B and N are disjoint finite sets of variable indices with $|B| = n$, $|N| = d$, $b \in \mathbb{R}^B$ and $A \in \mathbb{R}^{B \times N}$ are given input vector and matrix. An LP of this form is called LP in *dictionary form* and its *size* is $n \times d$. The set B is called a (initial) *basis*, N a (initial) *nonbasis* and $c^T x_N$ the *objective function*.

The *feasible region* of the LP is defined as the set of $x \in \mathbb{R}^E$ that satisfy all constraints, i.e., the set $\{x \in \mathbb{R}^E \mid x_B = b - A x_N, x_E \geq 0\}$. A feasible solution x^* is called *optimal* if for every feasible solution x, $c^T x^* \leq c^T x$. The LP is called *unbounded* if for every $z \in \mathbb{R}$, there exists a feasible solution x_z such that $c^T x_z \leq z$. If there exists no feasible solution, the LP is called *infeasible*.

The *dictionary* $D(B) \in \mathbb{R}^{B\cup\{f\}\times N\cup\{g\}}$ of an LP (3.3) w.r.t. a basis B is defined as

$$D := D(B) = \begin{bmatrix} 0 & c^T \\ b & -A \end{bmatrix},$$

where f is the index of the first row and g is the index of the first column. For each $i \in B \cup \{f\}$ and $j \in N \cup \{g\}$, we denote by d_{ij} its (i,j) entry, by $D_{i\bullet}$ the row indexed by i, and by $D_{\bullet j}$ the column indexed by j.

Hence by setting $x_f := c^T x_N$, we can rewrite (3.3) as

$$\begin{array}{lrcl} \text{minimize} & x_f & & \\ \text{subject to} & x_{B\cup\{f\}} & = & Dx_{N\cup\{g\}} \\ & x_E & \geq & 0, \\ & x_g & = & 1. \end{array} \tag{3.4}$$

Whenever we do not care about the objective function, we may set $c = 0$, and with abuse of notation, set $D = [b, -A]$.

The *basic solution* w.r.t. B is the unique solution $\overline{x}$ to $x_{B\cup\{f\}} = Dx_{N\cup\{g\}}$ such that $\overline{x}_g = 1$, $\overline{x}_N = 0$ and hence $\overline{x}_{B\cup\{f\}} = D_{\bullet g}$.

The *dual LP* of LP (3.4) is defined as

$$\begin{array}{lrcl} \text{minimize} & y_g & & \\ \text{subject to} & y_{N\cup\{g\}} & = & -D^T y_{B\cup\{f\}} \\ & y_E & \geq & 0, \\ & y_f & = & 1. \end{array} \tag{3.5}$$

It is useful to define the following four different types of dictionaries (and bases) as shown in Figure 3.1 below, where "$+$" denotes positivity, "$\oplus$" nonnegativity and similarly "$-$" negativity and "$\ominus$" nonpositivity.

A dictionary D (or the associated basis B) is called *feasible* if $d_{ig} \geq 0$ for all $i \in B$. A dictionary D (or the associated basis B) is called *optimal* if $d_{ig} \geq 0$, $d_{fj} \geq 0$ for all $i \in B, j \in N$. A dictionary D (or the associated basis B) is called *inconsistent* if there exists $r \in B$ such that $d_{rg} < 0$ and $d_{rj} \leq 0$ for all $j \in N$. A dictionary D (or the associated

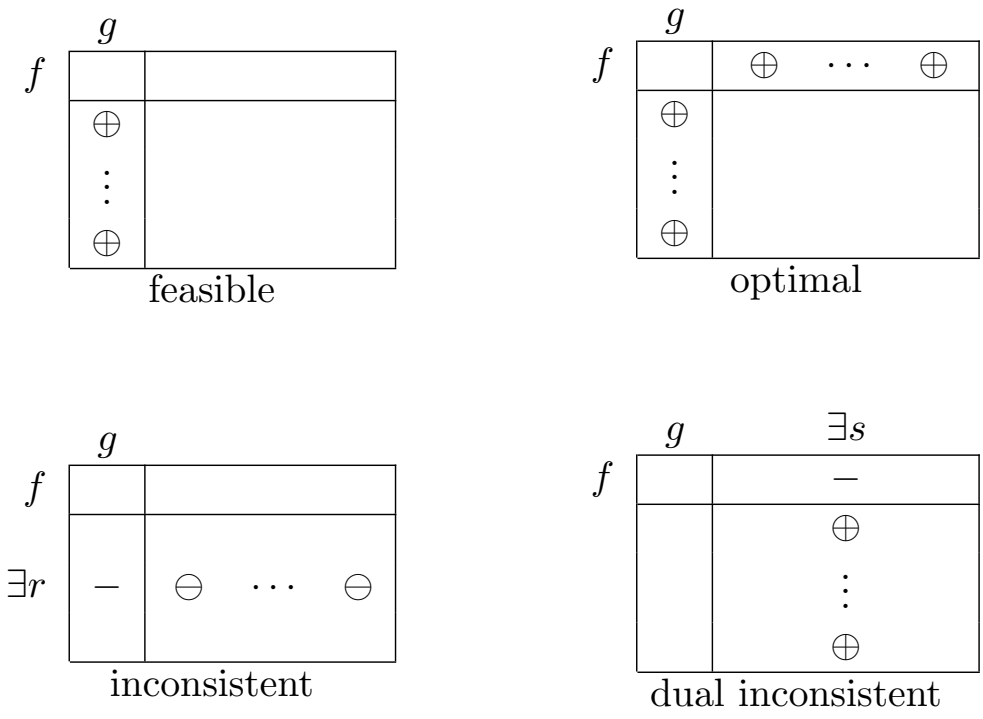

Figure 3.1: Four types of dictionaries

basis B) is called *dual inconsistent* if there exists $s \in N$ such that $d_{fs} < 0$ and $d_{is} \geq 0$ for all $i \in B$.

The following proposition follows from standard calculations.

Proposition 3.2.1. *For any LP in dictionary form the following statements hold.*

1. *If the dictionary is feasible then the associated basic solution is feasible.*

2. *If the dictionary is optimal, then the associated basic solution is optimal.*

3. *If the dictionary is inconsistent, then the LP is infeasible.*

4. *If the dictionary is dual inconsistent, then the dual LP is infeasible. If in addition the LP is feasible, then the LP is unbounded.*

3.2.2 Pivot Operations

We now show how to transform the dictionary of an LP into a modified dictionary using an elementary matrix operation, preserving the equivalence of the associated linear system. This operation is called a *pivot operation.*

Let $p \in B$, $q \in N$ and $d_{pq} \neq 0$. Then one can transform $x_{B\cup\{f\}} = Dx_{N\cup\{g\}}$ to an equivalent system (i.e., with the same feasible region):

$$x_{B'\cup\{f\}} = D'x_{N'\cup\{g\}}, \tag{3.6}$$

where $B' = B \setminus \{p\} \cup \{q\}$ ($N' = N \setminus \{q\} \cup \{p\}$, respectively) is a new (non)basis and

$$d'_{ij} = \begin{cases} \frac{1}{d_{pq}} & \text{if } i = q \text{ and } j = p \\ -\frac{d_{pj}}{d_{pq}} & \text{if } i = q \text{ and } j \neq p \\ \frac{d_{iq}}{d_{pq}} & \text{if } i \neq q \text{ and } j = p \\ d_{ij} - \frac{d_{iq}\cdot d_{pj}}{d_{pq}} & \text{if } i \neq q \text{ and } j \neq p \end{cases} (i \in B' \cup \{f\} \text{ and } j \in N' \cup \{g\}). \tag{3.7}$$

In the transformation one solves equality (3.6) with x_p on the left side, such that afterwards x_q is on the left side. The right side of the equation is then substituted for the x_q's in all other equalities.

We call a dictionary *terminal* if it is optimal, inconsistent or dual inconsistent. There are several finite pivot algorithms such as the simplex and the criss-cross method that transform any dictionary into one of the terminal dictionaries [45, 47, 22],[13, Section 4]. This will be discussed further in Section 3.4.3.

3.3 Combinatorial Redundancy

Consider an LP in dictionary form as given in (3.3). Then $x_r \geq 0$ is *redundant*, if the removal of the constraint does not change the feasible

region, i.e., if

$$\begin{array}{lrcl} \text{minimize} & c^T x_N & & \\ \text{subject to} & x_B & = & b - Ax_N \\ & x_i & \geq & 0, \quad \forall i \in E \setminus \{r\}, \end{array} \tag{3.8}$$

has the same feasible region as (3.3). Then the variable x_r and the index r are called *redundant*.

If the constraint $x_r \geq 0$ is not redundant it is called *nonredundant*, in that case the variable x_r and the index r are called *nonredundant*.

It is not hard to see that solving $n+d$ LPs of the same size as (3.8) suffices to find all redundancies. Hence running time $O((n+d) \cdot LP(n,d))$ suffices to find all redundancies, where $LP(n,d)$ is the time needed to solve an LP of size $n \times d$. Clarkson showed that it is possible to find *all* redundancies in time $O((n+d) \cdot LP(s,d) + s \cdot n \cdot d)$, where s is the number of nonredundant variables [10]. In the case where $s \ll n$, this is a major improvement. To be able to execute Clarkson's algorithm, one needs to assume full-dimensionality and an interior point of the feasible region. In the LP setting this can be done by some preprocessing as shown in Section 2.2.2.

In the following we focus on the combinatorial aspect of redundancy removal. We give a combinatorial way, the *dictionary oracle*, to encode LPs in dictionary form, where we are basically only given the signs of the entries of the dictionaries. In Section 3.4 we will show how the signs suffice to find all redundant and nonredundant constraints of an LP in dictionary form.

Consider an LP of form (3.3). For any given basis B, the *dictionary oracle* returns a matrix

$$D^\sigma = D^\sigma(B) \in \{+,-,0\}^{B \times N \cup \{g\}},$$

with

$$d^\sigma_{ij} = \text{sign}(d_{ij}), \forall i \in B, j \in N \cup \{g\}.$$

Namely, for basis B, the oracle simply returns the matrix containing the signs of $D(B)$, without the entries of the objective row f.

3.4 Certificates

We show that the dictionary oracle is enough to detect all redundancies and nonredundancies of the variables in E. More precisely for every $r \in E$, there exists a basis B such that $D^\sigma(B)$ is either a redundancy or nonredundancy certificate for x_r. We give a full characterization of the certificates in Theorems 3.4.3 and 3.4.5. An alternative characterization of the nonredundancy certificate is given in Theorem 3.7.1. The number of dictionaries needed to have *all* certificates depends on the LP. See Section 3.6 for examples where constantly many suffice and where linearly many are needed.

For convenience throughout we make the following assumptions, which can be satisfied with simple preprocessing.

Assumption 3.4.1. *The feasible region of (3.3) is full-dimensional (and hence nonempty).*

Assumption 3.4.2. *There is no $j \in N$ such that $d_{ij} = 0$ for all $i \in B$.*

Regarding Assumption 3.4.1: In Section 3.4.3 we will see that both the criss-cross and the simplex method can be used on the dictionary oracle for certain objective functions. Testing whether the feasible region is empty can hence be done by solving one linear program in the oracle setting. As mentioned in the introduction the full-dimensionality assumption is made to avoid dependencies between the redundant constraints. This can be achieved by some preprocessing on the LP, including solving a few ($O(d)$) LPs (see Section 2.2.2).

Regarding Assumption 3.4.2: It is easy to see that if there exists a column j such that $d_{ij} = 0$ for all $i \in B$, then x_j is nonredundant. As it can take any value independent of the others, in particular, there are solutions with $x_j < 0$, which implies that x_j is nonredundant. The redundancies and nonredundancies of all other variables are independent of x_j, hence we can mark x_j as nonredundant and simply remove the column.

3.4.1 A Certificate for Redundancy in the Dictionary Oracle

We say a basis B is *r-redundant*, if $r \in B$ and $D^{\sigma}_{r\bullet} \geq 0$, i.e., if $D^{\sigma}(B)$ is of the form of Figure 3.2.

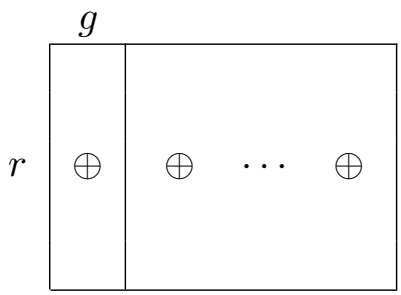

Figure 3.2: r-redundant

Note that an r-redundant basis is not necessarily feasible. Since the r-th row of the dictionary represents $x_r = d_{rg} + \sum_{j \in N} d_{rj} x_j$, $x_r \geq 0$ is satisfied as long as $x_j \geq 0$ for all $j \in N$. Hence $x_r \geq 0$ is redundant for (3.3).

Theorem 3.4.3 (**Redundancy Certificate**). *An inequality $x_r \geq 0$ is redundant for the system (3.3) if and only if there exists an r-redundant basis.*

Proof. We only have to show the "only if" part.

Suppose $x_r \geq 0$ is redundant for the system (3.3). We will show that there exists an r-redundant basis.

Consider the LP minimizing the variable x_r subject to the system (3.3) without the constraint $x_r \geq 0$. Since $x_r \geq 0$ is redundant for the system (3.3), the LP is bounded. By Assumption 3.4.1 and the fact that every finite pivot algorithm terminates in a terminal dictionary the LP has an optimal dictionary.

If the initial basis contains r, then we can consider the row associated with r as the objective row. Apply any finite pivot algorithm to the LP. Otherwise, r is nonbasic. By Assumption 3.4.2, one can pivot on

the r-th column to make r a basic index. This reduces the case to the first case.

Let's consider an optimal basis and optimal dictionary for the LP where x_r is the objective function. Since it is optimal, all entries d_{rj} for $j \in N$ are nonnegative. Furthermore, d_{rg} is nonnegative as otherwise we would have found a solution that satisfies all constraints except $x_r \geq 0$, implying nonredundancy of x_r. □

From the proof of Theorem 3.4.3 the following modification of Theorem 3.4.3 is immediate.

Corollary 3.4.4. *An inequality $x_r \geq 0$ is redundant for the system (3.3) if and only if there exists a* feasible *r-redundant basis.*

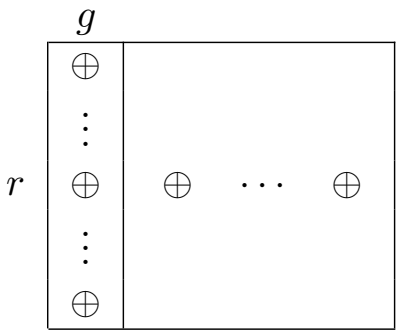

Figure 3.3: feasible r-redundant

3.4.2 A Certificate for Nonredundancy in the Dictionary Oracle

Similarly to the redundancy case, we introduce a certificate for nonredundancy using the dictionary oracle. A basis B is called *r-nonredundant* if B is feasible, $r \in N$ and $d_{tg} = 0$ implies $d_{tr} \leq 0$ for all $t \in B$ i.e. $D^\sigma(B)$ is of the form of Figure 3.4.

Theorem 3.4.5 (**Nonredundancy Certificate**)**.** *An inequality $x_r \geq 0$ is nonredundant for the system (3.3) if and only if there exists an r-nonredundant basis.*

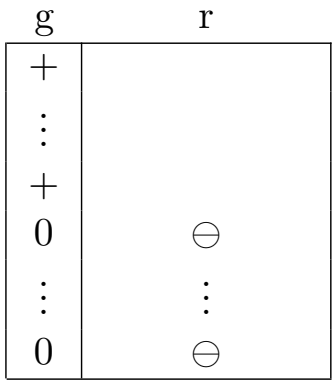

Figure 3.4: r-nonredundant

Before proving the theorem, we observe the following.

1. Unlike in the redundancy certificate an r-nonredundant basis needs to be feasible. To verify the correctness of a nonredundancy certificate we need to check between n and $2n$ entries, which is typically much larger than the $d+1$ entries we need for the redundant case.

2. If the g-column of a feasible basis does not contain any zeros, then all nonbasic variables are nonredundant. In general when $x_r \geq 0$ is nonredundant, not necessarily every feasible basis B with $r \in N$ is r-nonredundant. Consider the system:

$$\begin{aligned} x_3 &= x_1 + x_2 \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

   Then the basis $\{3\}$ is not a certificate of nonredundancy of x_1, as $d^{\sigma}_{31} = +$ in the associated dictionary. On the other hand, the basis $\{2\}$ is 1-nonredundant:

	g	1	2
3	0	+	+

	g	1	3
2	0	−	+

Proof of Theorem 3.4.5. Let (LP) be an LP of form (3.3) and suppose that $x_r \geq 0$ is nonredundant. Then it follows that for ϵ small enough

$x_r \geq -\epsilon$ is nonredundant in

$$\begin{array}{lrcl} \text{minimize} & x_r & & \\ \text{subject to} & x_B & = & b - Ax_N \\ & x_i & \geq & 0, \quad \forall i \in B \cup N \setminus \{r\} \\ & x_r & \geq & -\epsilon. \end{array} \tag{3.9}$$

Note that this LP can easily be transformed to an LP of form (3.3) by the straightforward variable substitution $x'_r = x_r + \epsilon$. We denote this LP obtained after perturbation and substitution by (LP^ϵ).

(LP^ϵ) attains its minimum at $-\epsilon$ and hence there exists an optimal dictionary where r is nonbasic. Let B be such a feasible optimal basis of (LP^ϵ) with $r \in N$. We show that if we choose ϵ small enough, B is r-nonredundant in (LP).

Let $B_1, B_2, \ldots, B_m$ be the set of all bases (feasible and infeasible) of (LP), that have r as a nonbasic variable. Choose $\epsilon > 0$ such that

$$\epsilon < \min \left\{ \frac{d_{tg}}{d_{tr}} \,\middle|\, t \in B_i : d_{tg}, d_{tr} < 0; i = 1, 2, \ldots, m \right\}.$$

If the right hand side (RHS) is undefined, we choose any $\epsilon < \infty$.

Geometrically this means that if for $t \in B_i$ $x_t \geq 0$ is violated in the basic solution w.r.t. B_i in (LP), then it is still violated in the corresponding basic solution (LP^ϵ). Let D and D^ϵ be the dictionaries w.r.t. B in (LP) and (LP^ϵ) respectively.

D and D^ϵ only differ in their entries of column g, where

$$d^\epsilon_{tg} = d_{tg} - \epsilon \cdot d_{tr}, \forall t \in B. \tag{3.10}$$

We need to show that B is r-nonredundant in (LP). To show that B is a feasible basis we need that $d_{tg} \geq 0$ for all $t \in B$. If $d_{tr} \geq 0$, then this is clear. For the case where $d_{tr} < 0$ assume that $d_{tg} < 0$. Then by choice of ϵ, substituting into (3.10) gives $d^\epsilon_{tg} < 0$, which is a contradiction to the nonnegativity of d^ϵ_{tg}. If $d_{tg} = 0$, by (3.10) it follows that $d_{tr} \leq 0$. Therefore B is r-nonredundant.

For the other direction let B be r-nonredundant and D and D^ϵ the corresponding dictionaries in (LP) and (LP^ϵ), respectively. Choose $\epsilon > 0$ such that

$$\epsilon \leq \min \left\{ \frac{d_{tg}}{d_{tr}} \,\middle|\, t \in B : d_{tg}, d_{tr} > 0 \right\}.$$

If the RHS is undefined, we choose any $\epsilon < \infty$.

We claim that for such an ϵ, B is still feasible for (LP^ϵ) and hence $x_r \geq 0$ is nonredundant. Again the two dictionaries only differ in column g, where

$$d^\epsilon_{tg} = d_{tg} - \epsilon \cdot d_{tr}, \forall t \in B.$$

In the case where $d_{tg} = 0$, it follows that $d^\epsilon_{tg} \geq 0$ by r-nonredundancy. If $d_{tg} > 0$, then

$$d^\epsilon_{tg} = d_{tg} - \epsilon \cdot d_{tr} \geq d_{tg} - \min \left\{ \frac{d_{t'g}}{d_{t'r}} \,\middle|\, t' \in B : d_{t'g}, d_{t'r} > 0 \right\} \cdot d_{tr} \geq 0.$$

□

3.4.3 Finite Pivot Algorithms for Certificates

In this section we discuss how to design finite pivot algorithms for the dictionary oracle model. Both the criss-cross method and the simplex method can be used for the dictionary oracle to find redundancy and nonredundancy certificates. A finite pivot algorithm chooses in every step a pivot according to some given rule and terminates in an optimal, inconsistent or dual inconsistent basis in a finite number of steps. Note that both the criss-cross method and the simplex method may not be polynomial in the worst case, but are known to be fast in practice [33, 39]. Furthermore there exists no known polynomial algorithm to solve an LP given by the dictionary oracle. Fukuda conjectured that the randomized criss-cross method is an expected polynomial time algorithm [17].

By the proof of Theorem 3.4.3, in order to find a redundancy certificate in (3.3) it is enough to solve (3.3) with objective function x_r. Similarly

by the proof of Theorem 3.4.5, for a nonredundancy certificate it is enough to solve the ϵ-perturbed version (3.9).

For the criss-cross method, the pivot rule is solely dependent on the signs of the dictionary entries and not its actual values [16, Chapter 4], [22]. Standard calculations show that the signs in the ϵ-perturbed dictionary (for $\epsilon > 0$ small enough) are completely determined by the signs of the original dictionary. We recall that the dictionary oracle does not output the objective row, but since we minimize in direction of x_r the signs of the objective row are completely determined. (If r is basic then the objective row has the same entries as the r-th row and if r is nonbasic then $d_{fr} = +$ and all other entries of the objective row are zero.) Therefore the dictionary oracle is enough to decide on the pivot steps of the criss-cross method.

For the simplex method with the smallest index rule, we are given a feasible basis and the nonbasic variable of the pivot element is chosen by its sign only [8, Part 1 Section 3]. The basic variable of the pivot is chosen as the smallest index such that feasibility is preserved after a pivot step. Using the dictionary oracle one can test the at most n possibilities and choose the appropriate pivot.

3.5 An Output Sensitive Redundancy Detection Algorithm

Throughout this section, we denote by S' the set of nonredundant indices and by R' the set of redundant indices. Denote by $LP(n, d)$ the time needed to solve an LP. By the discussion in Section 3.4.3, for any x_r, $r \in E$, we can find a certificate in time $LP(n, d)$. Theorem 3.5.2 presents a Clarkson type, output sensitive algorithm with running time $O(d \cdot (n + d) \cdot s^{d-1} \cdot LP(s, d) + d \cdot s^d \cdot LP(n, d))$, that for a given LP outputs the set S', where $s = |S'|$. Typically s and d are much smaller than n.

3.5.1 General Redundancy Detection

For $F \subseteq B$, $F \neq \emptyset$, we say that $r \in E$ is *redundant* (*nonredundant*, respectively) w.r.t. $F \cup N$, if it is redundant (nonredundant, respectively) in the system induced by $F \cup N$, i.e. the system obtained by removing all rows with indices in $B \setminus F$.

The following algorithm detects all redundancies of a linear system of form (3.3).

Redundancy Detection Algorithm(D,g,f);
begin
  $R := \emptyset, S := \emptyset$;
  while $R \cup S \neq E$ **do**
    Pick any $r \notin R \cup S$ and test if r is redundant w.r.t. S;
    /*For the case where $S \not\supseteq N$ see Remark 3.5.1*/
    if r is redundant w.r.t. S **then**
      $R = R \cup \{r\}$;
    else /* r is nonredundant w.r.t. S */
      test if r is redundant w.r.t. $E \setminus R$;
      if r is nonredundant w.r.t. $E \setminus R$ **then**
        $S = S \cup \{r\}$;
      else /* r is redundant w.r.t. $E \setminus R$ */
        Find some sets $S^F \subseteq S'$, $S^F \neq \emptyset$ and $R^F \subseteq R'$
        such that $S^F \not\subseteq S$;
        $R = R \cup R^F$, $S = S \cup S^F$;
      endif;
    endif;
  endwhile;
  $S^* := S$;
  output S^*;
end.

Since in every round at least one variable is added to S or R, the algorithm terminates. The correctness of the output can easily be verified: If in the outer loop r is added to R, r is redundant w.r.t. S

and hence redundant w.r.t. $S^* \supseteq S$. If in the inner loop r is added to S, r is nonredundant w.r.t. $E \setminus R$ and hence nonredundant w.r.t. $S^* \subseteq E \setminus R$. It follows that $S^* = S'$.

The main issue is how to find the sets S^F and R^F efficiently in the last step. This will be discussed in (the proof of) Lemma 3.5.3.

Remark 3.5.1. A technical problem is that we cannot test for redundancy in the dictionary oracle when S does not contain a nonbasis. Therefore as long as this is the case, we fix an arbitrary nonbasis N and execute the redundancy detection algorithm on $S \cup N$ instead of S. Since this stronger checking of redundancy does not change correctness or the order of the running time, we will omit this detail in the further discussion.

Theorem 3.5.2. *The redundancy detection algorithm outputs S', the set of nonredundant constraints in time*

$$R(n,d,s) = O\left(\sum_{i=0}^{d-1}((n+d)\cdot s^i \cdot LP(s,d-i) + s^{i+1}\cdot LP(n,d-i))\right)$$

and consequently in time

$$R(n,d,s) = O\left(d\cdot(n+d)\cdot s^{d-1}\cdot LP(s,d) + d\cdot s^d \cdot LP(n,d)\right).$$

The following Lemma implies Theorem 3.5.2.

Lemma 3.5.3. *Let $R(n,d,s)$ be the running time of the redundancy detection algorithm in n basic variables, d nonbasic variables and s the number of nonredundant variables. Then in the last step of the inner loop some sets $S^F \subseteq S'$ and $R^F \subseteq R'$, with $S^F \nsubseteq S$, can be found in time $O(R(n,d-1,s) + LP(n,d))$.*

Proof of Theorem 3.5.2. Termination and correctness of the algorithm are discussed above. The iteration of the outer loop of the algorithm takes time $O(LP(s,d))$ and is executed at most $n+d$ times. By Lemma 3.5.3, the running time of the inner loop is $O(R(n,d-1,s)+LP(n,d))$ and since in each round at least one variable is added to S, it is executed

at most s times. Therefore the total running time is given recursively by

$$R(n,d,s) = O\left((n+d)\cdot LP(s,d) + s\cdot(R(n,d-1,s) + LP(n,d))\right).$$

The claim follows by solving the recursion and noting that $R(n,0,s)$ can be set to $O(n)$. □

It remains to prove Lemma 3.5.3, for which we first prove some basic results below, using the dictionary oracle setting.

Lemma 3.5.4. *Let $D = D(B)$ be a feasible dictionary of (LP), an LP of form (3.3) and assume $F := \{i \in B \mid b_i = 0\} \neq \emptyset$. We consider the subproblem of (LP) denoted (LP^F) (with dictionary D^F,) that only contains the rows of D indexed by F. Then $r \in F \cup N$ is nonredundant in (LP) if and only if it is nonredundant in (LP^F).*

Proof. We only need to show the "if" part. Let $r \in F \cup N$ be nonredundant in (LP^F) with certificate $\overline{D}^F$. Then there exists a sequence of pivot steps from D^F to $\overline{D}^F$. Using the same ones on D and obtaining dictionary $\overline{D}$, this is a nonredundancy certificate for r, since $\overline{d}_{ig} = d_{ig} > 0$ for all $i \in B \setminus F$ by the definition of F. □

For a dictionary $D = [b, -A]$, we define $D^0 = [0, b, -A]$ as the dictionary obtained by adding an all-zero column as the first column. This dictionary now has an additional nonbasic variable and the all-zero column is associated with the g-column.

Lemma 3.5.5. *Let $D = [b, -A]$ be the dictionary of an LP of form (3.3). Then a variable $r \in E$ is nonredundant in the LP given by D if and only if it is nonredundant in the LP with dictionary $D^0 = [0, b, -A]$.*

Proof. If $D(B)$ is a redundancy certificate for r for some basis B, then $D^0(B)$ is a redundancy certificate for r as well.

For the converse, let $D = D(B)$ be a nonredundancy certificate for r for some basis B. For simplicity assume that $B = \{1, 2, \ldots, n\}$.

For now assume that $b_i > 0$ for all $i \in B$ and let D^i the dictionary obtained from D^0 by pivoting on b_i, $i = 1, 2, \ldots, n$. We will show that at least one of the D^i, $i \in \{0, 1, \ldots, n\}$ is a nonredundancy certificate for r. Since after any pivot the first column of D^i stays zero, D^i is a nonredundancy certificate if and only if $D^i_{\bullet r} \leq 0$, i.e., the r-th column of D^i is nonpositive. Let $R^i = (r^i_1, r^i_2, \ldots r^i_n)^T := D^i_{\bullet r}$ for $i \geq 1$ and $R^0 = (r_1, r_2, \ldots, r_n)^T := D^0_{\bullet r}$.

Claim: Assume that $r^i_i < 0$ for any fixed i and there are at least $i - 1$ additional nonpositive entries (w.l.o.g. we assume them to be $r^i_1, r^i_2, \ldots, r^i_{i-1}$). If R^i has a positive entry (which w.l.o.g. we assume to be r^i_{i+1}), then $r^{i+1}_{i+1} < 0$ and $r^{i+1}_1, r^{i+1}_2, \ldots, r^{i+1}_i$ are nonpositive.

If D^0 is not a certificate for r, then w.l.o.g. $r_1 > 0$ and hence $r^1_1 = -\frac{r_1}{b_1} < 0$. Therefore by induction the lemma follows from the claim.

It remains to prove the claim. Assume that $r^i_1, r^i_2, \ldots, r^i_{i-1} \leq 0$, $r^i_i < 0$ and $r^i_{i+1} > 0$. Then we have $r_i > 0$ and

$$r^i_{i+1} = r_{i+1} - \frac{r_i b_{i+1}}{b_i} > 0 \Leftrightarrow r_i b_{i+1} < r_{i+1} b_i \Rightarrow r_{i+1} > 0, \tag{3.11}$$

$$\forall j < i : r^i_j = r_j - \frac{r_i b_j}{b_i} \leq 0 \Leftrightarrow r_j b_i \leq r_i b_j. \tag{3.12}$$

The following calculations show the claim.

$r^{i+1}_{i+1} = -\frac{r_{i+1}}{b_{i+1}} < 0 \Leftrightarrow r_{i+1} > 0$. Hence by (3.11) $r^{i+1}_{i+1} < 0$.

$r^{i+1}_i = r_i - \frac{r_{i+1} b_i}{b_{i+1}} \leq 0 \Leftrightarrow r_i b_{i+1} \leq r_{i+1} b_i$. Again by (3.11) $r^{i+1}_i \leq 0$.

$\forall j < i : r^{i+1}_j = r_j - \frac{r_{i+1} b_j}{b_{i+1}} \leq 0 \Leftrightarrow r_j b_{i+1} \leq r_{i+1} b_j$.

By (3.11) and (3.12), $r_j b_{i+1} = (r_j b_i) \cdot (r_i b_{i+1}) \cdot \frac{1}{r_i b_i} \leq r_{i+1} b_j$,

hence $r^{i+1}_j \leq 0$.

Now suppose that $b_i = 0$ for some i. Then by the nonredundancy certificate $r_i \leq 0$, and it is easy to see that $r^j_i = r_i \leq 0$ for all admissible pivots on b_j. Hence we can use the above construction on the

nonzero entries of b, the rows corresponding to the zero entries satisfy the nonredundancy certificate conditions trivially. □

Proof of Lemma 3.5.3. Suppose that during the execution of the algorithm, r is nonredundant w.r.t. the current set S, and redundant w.r.t. $E \setminus R$, with *feasible* redundancy certificate $D = [b, -A]$, which exists by Corollary 3.4.4. If $b > 0$, then all nonbasic indices in N are nonredundant by Theorem 3.4.5. Choose $S^F = N$, $R^F = \emptyset$. It holds that $S^F \not\subseteq S$, since otherwise r would be redundant w.r.t. S. The running time of the inner loop in this case is $LP(n, d)$.

Now if there exists $i \in B$ such that $b_i = 0$, define $F = \{i \in B \mid b_i = 0\}$, LP^F and D^F as in Lemma 3.5.4. We now recursively find all redundant and nonredundant constraints in the LP^F using Lemma 3.5.5 as follows. From LP^F we construct another LP, denoted LP^- with one less nonbasic variable, by deleting $D^F_{\bullet g}$ (the column of all zeros), choosing any element $t \in N$ and setting $t = g$, i.e., setting $x_t = 1$ in the corresponding LP. Finding all redundancies and nonredundancies in LP^- takes time $R(|F|, d-1, s)$. By Lemma 3.5.5 redundancies and nonredundancies are preserved for LP^F.

Therefore finding them in LP^F takes time $R(|F|, d-1, s) + LP(n, d) \leq R(n, d-1, s) + LP(n, d)$, where the $LP(n, d)$ term is needed to check separately whether t is redundant. Choose S^F as the set of nonredundant indices of LP^F and R^F as the set of redundant ones. By Lemma 3.5.4 $S^F \subseteq S'$ and $R^F \subseteq R'$. By the same lemma r is redundant in LP^F, therefore it follows that $S^F \not\subseteq S$, as otherwise r would be redundant w.r.t. S. □

3.5.2 Strong Redundancy Detection

In this section we show how under certain assumptions the running time of the redundancy algorithm can be improved. If we allow the output to also contain some *weakly redundant* constraints (see definition below), it is basically the same as the running time of Clarkson's method.

A redundant variable r is called *strongly redundant* if for any basic feasible solution $\overline{x}$, $\overline{x}_r > 0$. In particular for any basic feasible solution, $r \in B$. If r is redundant but not strongly redundant r is called *weakly redundant.*

As before let s be the number of nonredundant constraints and let R_v, (with $|R_v| = r_v$,) and R_w, (with $|R_w| = r_w$,) be the set of strongly and weakly redundant constraints respectively.

Theorem 3.5.6. *Let S' be the set of nonredundant constraints. It is possible to find a set $S^* \supseteq S'$, $S^* \cap R_v = \emptyset$ in time*

$$O((n+d) \cdot LP(s + r_w, d) + (s + r_w) \cdot LP(n, d)).$$

The following corollary follows immediately.

Corollary 3.5.7. *If there are no weakly redundant constraints, the set S' of nonredundant constraints can be found in time $O((n+d) \cdot LP(s,d) + s \cdot LP(n,d))$.*

The theorem is proven using the following two lemmas, which can be verified with straightforward variable substitutions.

Lemma 3.5.8. *[8, Part 1 Section 3] Let (LP) be of form (3.3), where (LP) is not necessarily full-dimensional. W.l.o.g. $B = \{1, 2, \ldots, n\}$. For each $i \in \{1, 2, \ldots, n\}$ replace the nonnegativity constraint $x_i \geq 0$ by $x_i \geq -\epsilon^i$, for $\epsilon > 0$ sufficiently small number. Denote the resulting LP by (LP^ϵ). Let D^σ be the output of the dictionary oracle for an arbitrary dictionary D of (LP). Then (LP^ϵ) is full-dimensional. Furthermore in $D^{\sigma,\epsilon}$, the corresponding output for the ϵ-perturbed version, all signs can be determined by D^σ, and the column $D^{\sigma,\epsilon}_{\bullet g}$ has no zero entries.*

Note that the perturbation method mentioned above can be implemented symbolically without an explicit evaluation of ϵ. This combinatorial technique is known as the lexicographic method [8, Part 1 Section 3 Page 34].

Lemma 3.5.9. *[8, Part 1 Section 3] Let (LP) and (LP^ϵ) be as in Lemma 3.5.8. Then any nonredundant constraint in (LP) is nonredundant in (LP^ϵ) and any strongly redundant constraint in (LP) is strongly redundant in (LP^ϵ).*

Proof of Theorem 3.5.6. Replace the given LP by its ϵ-perturbed version as in Lemma 3.5.8 and run the redundancy removal algorithm, which is possible by the same lemma. By Lemma 3.5.9, $S^* \supseteq S'$ and $S^* \cap R_v = \emptyset$. Since by Lemma 3.5.8, the entries of the g-column of any dictionary $D^{\sigma,\epsilon}$ are strictly positive the algorithm never runs the recursive step and the running time follows. □

Remark: The ϵ-perturbation makes every feasible LP full-dimensional, therefore the full-dimensionality assumption can be dropped for Theorem 3.5.6.

3.6 Number of Dictionaries for all Certificates

In Section 3.4 we showed the existence of certificates in the dictionary oracle for both redundant and nonredundant variables. The main question discussed in this section is how many dictionaries are needed to detect all redundancies. As we will show below, this number depends on the given set of linear inequalities and lies between 1 and $n+d-s$, i.e., the number of redundant constraints. The number of dictionaries needed to detect all nonredundancies is not very interesting as usually $s \ll n$. Moreover a single dictionary is a certificate for at most d nonredundancies (the nonbasic variables), which implies that we always need between $\frac{s}{d}$ and s dictionaries in order to obtain all certificates.

It is not hard to find an example where a single basis B is r-redundant for all redundant constraints r. For instance the following basis is r-redundant for all $r \in B$ and r-nonredundant for all $r \in N$.

$$\begin{aligned} x_B &= \mathbf{1} + \mathbf{I}x_N \\ x_i &\geq 0, \quad \forall i \in E, \end{aligned}$$

where $\mathbf{1}$, $\mathbf{I}$ denote the all-one vector, identity matrix, respectively.

For the maximum number of dictionaries needed to detect all redundancies, we give an example of a system of linear equalities where every redundant constraint r has a unique r-redundant basis and all those bases are pairwise distinct. Therefore $n + d - s$ bases are needed to detect all redundancies. We consider the d-dimensional hypercube and its dual d-cross polytope, such that each vertex of the hypercube lies in the barycenter of its corresponding dual face of the d-cross polytope. We show that any constraint of the d-cross polytope is redundant for the cube and has a unique certificate corresponding to its dual vertex. (For an example in general position, one can move the constraints of the d-cross polytope away from the hypercube by some $\epsilon > 0$ small enough, and still obtain the same results.)

Formally the d-dimensional hypercube is given by

$$\begin{array}{rcll} x_i & \leq & 1, & \forall i \in \{1, \ldots, d\} \\ x_i & \geq & 0, & \forall i \in \{1, \ldots, d\}. \end{array} \tag{3.13}$$

The d-cross polytope has 2^d constraints given as follows: for $x = (x_1, x_2, \ldots, x_d)^T$ and each $p \in \{-1, +1\}^d$ we have the constraint

$$p^T x \leq k_p,$$

where k_p denotes the number of $+1$'s in p.

The system of linear inequalities in dictionary form is hence given by

$$\begin{array}{rcll} x_{i+d} & = & 1 - x_i, & \forall i \in \{1, \ldots, d\} \\ x_p & = & k_p - p^T x, & \forall p \in \{-1, +1\}^d \\ x_i & \geq & 0 & \forall i \in E. \end{array} \tag{3.14}$$

It is easy to see that $x_1, \ldots, x_{2d}$ are nonredundant.

Claim: For all $p \in \{-1, +1\}^d$, x_p is redundant and has unique p-redundant basis $B_p = \{i \in \{1, \ldots, d\} \mid p_i = -1\} \cup \{i + d \in \{d + 1, \ldots, 2d\} \mid p_i = 1\}$.

This implies that we need $2^d = n - d = n + d - s$ bases to detect all redundancies.

We first show that B_p is p -redundant. Observe that

$$\begin{aligned} x_p &= k_p - \sum_{i:p_i=-1} (-x_i) - \sum_{i:p_i=+1} x_i \\ &= k_p - \sum_{i:p_i=-1} (-x_i) - \sum_{i:p_i=+1} (1 - x_{i+d}) \\ &= \sum_{i:p_i=-1} x_i + \sum_{i:p_i=+1} x_{i+d}. \end{aligned}$$

and hence B_p is p-redundant. We show uniqueness for $p^- = (-1, \ldots, -1)^T$, the rest follows by symmetry. We prove that x_{p^-} is nonredundant in the system induced by $E \setminus \{i\}$, for all $i \in \{1, \ldots, d\}$, which implies that the unique p^--redundant basis is given by $B_{p^-} = \{1, \ldots, d\}$. Again by symmetry it is enough to prove this for $i = 1$.

Let D be the dictionary corresponding to the linear system (3.14). Let D' be the dictionary obtained by a pivot step on $(p^-, 1)$, and B' its basis. Then for all $i \in B' \setminus \{1\}$

$$d'_{ig} = d_{ig} - \frac{d_{i1} d_{p^- g}}{d_{p^- 1}} = d_{ig} > 0,$$

since $d_{p^- g} = k_{p^-} = 0$. Hence by the nonredundancy certificate, it follows that p^- is nonredundant w.r.t. $E \setminus \{1\}$ (and by symmetry also w.r.t. $E \setminus \{i\}$ for all $i = \{2, \ldots, d\}$), which concludes our proof.

To obtain an example with maximum number of unique pairwise distinct redundancy certificates we use McMullen's Upper Bound Theorem [36] (see also Theorem 5.2.1). It states that a polyhedron given by s constraints, can have at most $\Theta(s^{\lfloor \frac{d}{2} \rfloor})$ vertices and the bound is tight for the dual cyclic polytope. Similarly as in the hypercube case, we can define a redundant constraint through each of its vertices and get an example with $n + d - s \in \Theta(s^{\lfloor \frac{d}{2} \rfloor})$ redundant constraints that have unique, pairwise distinct redundancy certificates.

3.7 Alternative Nonredundancy Certificate

In this section we introduce two families of dictionaries that are an alternative nonredundancy certificate to the one described in Section 3.4.2. We show that from the latter r-nonredundancy certificate one can always obtain a certificate of the new forms (Section 3.7.1) in a single pivot step. The opposite does not hold in general.

3.7.1 Another Certificate

In this section we introduce an alternative certificate for nonredundancy, which is given by two different kinds of dictionaries. A basis B is called *r-nonredundant type I* if $r \in B$, $d_{rg} < 0$ and $d_{ig} \geq 0$ for all $i \in B \setminus \{r\}$ i.e. $D^\sigma(B)$ is of the form of Figure 3.5.

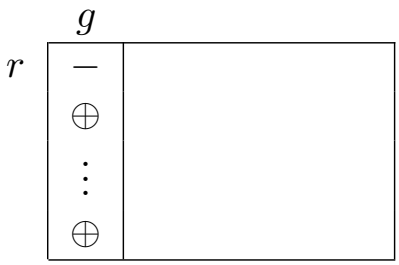

Figure 3.5: r-nonredundant type I

A basis B is called *r-nonredundant type II* if $r \in B$, and there exists $t \in N$ such that $d_{rt} < 0$ and $d_{it} \geq 0$ for all $i \in B \setminus \{r\}$ i.e. $D^\sigma(B)$ is of the form of Figure 3.6.

Note that here in both types of certificates r is a basic variable, whereas in the certificate of Theorem 3.4.5 r is nonbasic.

Theorem 3.7.1. *An inequality $x_r \geq 0$ is nonredundant for the system (3.3) if and only if there exists an r-nonredundant type I or r-nonredundant type II basis.*

Proof. Suppose x_r is nonredundant for the system (3.3). W.l.o.g. assume that $r \in B$ (by Assumption 3.4.2 in Section 3.4). Consider the

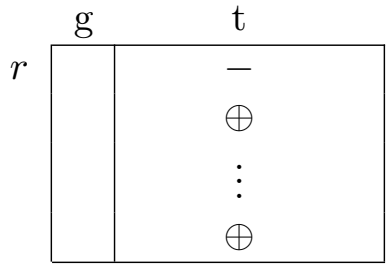

Figure 3.6: r-nonredundant type II

linear program (LP') obtained by considering $x_r = b_r - A_{r\bullet}x$ as the objective row instead of a constraint. As mentioned in Section 3.2.2 (LP') can be transformed into one of the terminal dictionaries, i.e., into an optimal, inconsistent or dual inconsistent dictionary. Since (LP') is feasible, the inconsistent case is void.

Suppose (LP') can be transformed into an optimal dictionary D. If $d_{rg} \geq 0$, then this implies redundancy of r, which is a contradiction. Hence $d_{rg} < 0$ and by considering $x_r = b_r - A_{r\bullet}x$ again as a constraint instead of the objective row, this is a r-nonredundant type I dictionary.

If (LP') can be transformed into an unbounded dictionary, this immediately gives us an r-nonredundant type II basis.

For the other direction suppose D is r-nonredundant type I. Then the corresponding basic solution, where $x_g = 1$, $x_N = 0$ and $x_B = D_{\bullet g}$, satisfies all constraints but $x_r \geq 0$. By definition r is nonredundant. If D is r-nonredundant type II, then by the nonemptyness assumption there exists a solution $y = (y_B, y_N)$ to the linear system. By setting $y_B^c = b - Ay_N + c \cdot D_{\bullet t}$, for c large enough, again all constraints but $x_r \geq 0$ are satisfied. □

We observe that the above certificates can be found with the algorithms described in Section 3.4.3. However the redundancy detection algorithm can not be applied in the given form to them. This can be seen already in general case, as we use that any r-redundant basis is t-nonredundant for $t \in N$.

3.7.2 Comparison of Certificates

The natural question that arises, is how the above certificates relate to the r-nonredundancy certificate of Section 3.4.2. This relation is given in the following theorem.

Theorem 3.7.2. *We can obtain a r-nonredundant type I or type II dictionary in one pivot step from any r-nonredundant dictionary. The opposite direction does not hold in general.*

Proof. Let D be an r-nonredundant dictionary. Assume there exists $i \in B$ such that $d_{ig} > 0$ and $d_{ir} > 0$. Choose $t \in B$ such that

$$\frac{d_{tg}}{d_{tr}} = \min \left\{ \frac{d_{kg}}{d_{kr}} \middle| k \in B : d_{kg}, d_{kr} > 0 \right\}.$$

The following calculation shows that the dictionary D', obtained by a pivot step on (t, r) is r-nonredundant type I.

First observe that $d'_{rg} = -\frac{1}{d_{tr}} < 0$, since $d_{tr} > 0$. For $i \in B, i \neq t$ we have: $d'_{ig} = d_{ig} - \frac{d_{ir}d_{tg}}{d_{tr}} \geq 0$. For $d_{ig} > 0$, this holds by choice of t and for $d_{ig} = 0$ we know that $d_{ir} < 0$, since D is r-nonredundant.

If there exists no $i \in B$ such that $d_{ig} > 0$ and $d_{ir} > 0$, then $d_{ir} \leq 0$ for all $i \in B$. Then for any $d_{tr} < 0$, the dictionary obtained from D by a pivot step on (t, r) is r-nonredundant type II.

On the other hand, consider the following 3-nonredundant type II dictionary.

	g	1	2
3	1	-1	0
4	1	0	-1
5	$-\frac{3}{2}$	1	1

As shown below none of the four possible pivots steps yield a 3-nonredundant basis (the dictionaries for the other two pivots follow by symmetry).

□

	g	1	4
3	1	-1	0
2	1	0	-1
5	$-\frac{1}{2}$	1	-1

pivot $(4, 2)$

	g	5	2
3	$-\frac{1}{2}$	-1	1
4	1	0	-1
1	$\frac{3}{2}$	1	-1

pivot $(5, 1)$

3.8 Discussion and Open Questions

In this chapter, we presented new combinatorial characterizations of redundancy and nonredundancy in linear inequality systems. We also presented a combinatorial algorithm for redundancy removal.

In contrast to the Clarkson algorithm our redundancy detection algorithm does not need the whole LP data but only the combinatorial information of the dictionaries. Although in general the running time is worse, assuming that we have no weak redundancies, our redundancy removal algorithm basically has the same running time as the Clarkson algorithm. Still, a natural goal is to improve the runtime of our algorithm in the general case and get it closer to that of Clarkson's method. We do have a first output-sensitive algorithm for combinatorial redundancy detection, but the exponential dependence on the dimension d is prohibitive already for moderate d.

Our algorithm works in a more general setting of oriented matroids [21, 45, 47]. This means one can remove redundancies from oriented pseudo hyperplane arrangements efficiently. Furthermore, the algorithm can be run in parallel. Yet, analyzing the performance may not be easy because checking redundancy of two distinct variables simultaneously may lead to the discovery of the same (non)redundant constraint. This is an interesting subject of future research.

Chapter 4

Redundancy Detection in Linear Systems with two Variables per Inequality

This chapter is based on [20] by K. Fukuda and M. Szedlák.

4.1 Introduction

Throughout this chapter we consider linear systems of inequalities of form $Ax \leq b$, for $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$, $d < n$.

We already mentioned that in general no strongly polynomial time algorithm (polynomial in d and n) to solve an LP is known. In this chapter we focus on the special case where every constraint has at most two variables with nonzero coefficients; we denote this family by $LI(2)$. Our main result is that for a full-dimensional system $Ax \leq b$ in $LI(2)$, we can detect all redundancies in strongly polynomial time(see Theorem 5.3.5), where we assume that a solution in the interior is given. To our knowledge, this is a first strongly polynomial time algorithm for redundancy detection in $LI(2)$. In the following chapter we show that the polytopes defined by systems in $LI(2)$ can still have high complexity.

For our algorithm we use an alternative version of Clarkson's algorithm, which solves feasibility problems instead of optimization problems. Moreover we make use of a modified version of Hochbaum and Naor's algorithm, which for a system in $LI(2)$ finds a feasible point or a

certificate for infeasibility in time $O(d^2 n \log n)$ [29]. This yields running time $O(nd^2 s \log s)$ for our redundancy detection algorithm. The result of Hochbaum and Naor is improvement of Megiddo's algorithm with running time $O(d^3 n \log n)$ [37]. Although their techniques are similar and both rely heavily on [2], the improved version is much simpler.

We will give a summary of the Hochbaum-Naor Algorithm in Section 4.4. In Section 4.5 we will give a stronger version of this algorithm which decides full-dimensionality and in the full-dimensional case outputs an interior point. Using this variant of the algorithm together with our modification of Clarkson's algorithm we get an output sensitive, strongly polynomial time redundancy detection algorithm. In Section 4.6 we show how the results extend to non-full-dimensional systems (see Theorem 4.6.1).

In all cases the preprocessing can also be done in strongly polynomial time. Moreover, we show that dimensionality testing of a polyhedron $P = \{x \mid Ax \leq b\}$ can be done with the same running time as the feasibility testing method of Hochbaum and Naor (see Corollary 4.6.4). Note that for general LP's one needs to solve up to d optimization problems (see Section 2.2.2).

Although in $LI(2)$ one can find a feasible solution fast, it is not known how to find an optimal solution in strongly polynomial time. For general LPs a standard technique for converting an optimization problem into a feasibility problem is to use the dual linear program. However, the dual of a system in $LI(2)$ is generally not in $LI(2)$. If the objective function is in $LI(2)$, one can apply binary search on the value of the optimal solution. However, this would lead to an algorithm that is not strongly polynomial because the number of iterations performed for the binary search depends on the sizes of the input numbers.

Note that Clarkson's algorithm relies on finding an optimal solution of a linear program. Since for $LI(2)$ we do not have a fast way to optimize, this is the reason why we modify the algorithm such that it only solves feasibility problems.

4.2 Definitions and Preliminaries

Again, we throughout consider *linear systems* of the form

$$Ax \leq b,$$

where $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$.

The set of inequalities of $Ax \leq b$ is denoted by G. Recall that a point $x^* \in \mathbb{R}^d$ is a *feasible solution* or *feasible point* of $Ax \leq b$ (or G) if $Ax^* \leq b$. It is called an *interior point solution* of $Ax \leq b$ (or G) if all inequalities are satisfied with strict inequality, i.e., $Ax^* < b$ (where "$<$" denotes the componentwise strict inequality). The system $Ax \leq b$ (or G) is called *feasible* if a feasible solution exists, otherwise it is called *infeasible*. If an interior point solution exists, the system is called *full-dimensional*. The system $Ax \leq b$ is called k-dimensional if the feasible region $\{x \mid Ax \leq b\}$ is k-dimensional.

For a subset $S \subseteq [n] := \{1, \ldots, n\}$ we denote by $A_S x \leq b_S$ the subsystem of $Ax \leq b$ containing only the inequalities indexed by S. In particular the j-th constraint is denoted by $A_j x \leq b_j$.

For a feasible system $Ax \leq b$ and a variable x_i let $[x_i^{\min}, x_i^{\max}]$ be the projection of the feasible region of $Ax \leq b$ to the x_i-axis, we call this the *range* of x_i. This interval is exactly the set of values of x_i for which a solution of the entire system can be constructed. It is possible that $x_i^{\min} = -\infty$ or $x_i^{\max} = \infty$.

In this chapter we are interested in sparse linear systems, in particular the family $LI(2)$. A linear system is in $LI(2)$, if every constraint has at most two variables with nonzero coefficients. That means all inequalities have form $\alpha x_i + \beta x_j \leq \gamma$ for some α, β, $\gamma \in \mathbb{R}$, $\alpha \neq 0$.

We define the *neighbors* of x_i in G, denoted $N(x_i, G)$, as the set of variables x_j, $j \in [d] \setminus \{i\}$ for which there exists an inequality in G containing x_i and x_j with nonzero coefficients.

The system $(Ax \leq b)|_{x_i = c}$ (or $G|_{x_i = c}$) is obtained from $Ax \leq b$ (or G) by substituting the variable x_i with the constant c, it hence has one variable less than the original system.

4.3 A Strongly Polynomial Time Redundancy Detection Algorithm for Linear Programs with two Variables per Inequality

In this section we will prove our main result, the running time of the strongly polynomial algorithm to detect all redundancies in $LI(2)$ (see Theorem 5.3.5).

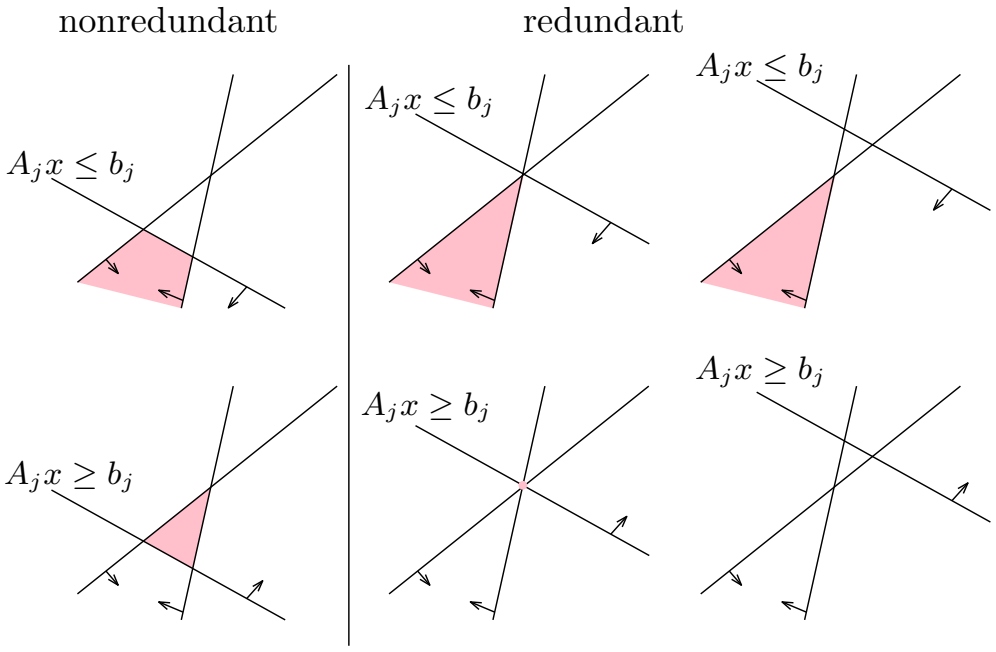

Figure 4.1: Redundancy Certificates

We make use of the following modified version of Hochbaum and Naor's result (Theorem 4.3.1). We will discuss their original result in Section 4.4 and the validity of the modification in Section 4.5. In Section 4.6 we will discuss how the results extend to non-full-dimensional systems.

Theorem 4.3.1. *For a system $Ax \leq b$ in $LI(2)$ one can decide in time $O(d^2 n \log n)$ whether the system is full-dimensional, and in the full-dimensional case output an interior point solution.*

For $S \subseteq [n]$, $r \in [n]$, let us denote by $G(S, -r)$ the system $\{A_{S\setminus\{r\}} x \leq b_{S\setminus\{r\}}, A_r x \geq b_r\}$.

Theorem 4.3.2. *Let $Ax \leq b$ a full-dimensional system in $LI(2)$. Let z be an interior point solution of $Ax \leq b$ and let $z^\epsilon := z + (\epsilon, \ldots, \epsilon^d)^T$ for some ϵ small enough, i.e., z^ϵ is a generic interior point. Then the following algorithm detects all redundancies in time $O(nd^2 s \log s)$.*

Algorithm Modified Clarkson (A,b,z);
begin
$R := \emptyset, S := \emptyset$;
while $R \cup S \neq [n]$ **do**
pick any $r \in [n] \setminus (R \cup S)$ *and use Theorem 4.3.1 on* $G(S, -r)$
if $G(S, -r)$ *not full-dimensional* **then**
$R = R \cup \{r\}$;
else *let* x^* *be an interior point solution of* $G(S, -r)$
$S = S \cup \{q\}$, *where* $q = RayShoot(A, b, z^\epsilon, x^* - z^\epsilon)$;
endif;
endwhile;
output S;
end.

The function RayShoot(A, b, z, t) returns the index q of the a hyperplane $\{x : A_q x = b_q\}$, which is hit first by the ray starting at z along the direction of t.

Note that since z^ϵ is generic, RayShoot$(A, b, z^\epsilon, x^* - z^\epsilon)$ is unique. Moreover, since z^ϵ is a feasible solution and by construction x^* is not a feasible solution, q must be nonredundant.

Note that Theorem 4.3.1 immediately implies that the interior point z of Theorem 5.3.5 can be found in strongly polynomial time $O(nd^2 \log n)$. It follows that finding all redundancies and the preprocessing can be achieved in strongly polynomial time.

The modified Clarkson algorithm is similar to the original version presented in Chapter 1. The main difference is the way we find the point

x^* through which we shoot the ray. The original version finds an appropriate point outside the feasible region using optimization, here we find such a point solving feasibility of an LP. We see in the proof, that the analyses of the algorithms are basically the same.

Using Theorem 4.3.1 and the following observation (see also Figure 4.1) we can prove Theorem 5.3.5.

Observation 4.3.3. *Let $Ax \leq b$ be a full-dimensional system in $LI(2)$. Then for $j \in [n]$ the following are equivalent.*

1. *$A_j x \leq b_j$ is nonredundant in $Ax \leq b$.*
2. *$A_j x \leq b_j$ is facet-inducing, i.e., defines a facet for*

$$P = \{x \mid Ax \leq b\}.$$

3. *The system $G([n] \setminus \{j\}, -j)$ is full-dimensional.*

Proof of Theorem 5.3.5. We have to show that the modified Clarkson Algorithm returns S^*, the indices of the set of nonredundant constraints. We first discuss correctness of the algorithm by induction. We claim that in every step $S \subseteq S^*$ and $R \subseteq [n] \setminus S^*$. This is trivially true in the beginning. Assume that in some step of the algorithm we have $S \subseteq S^*$, $R \subseteq [n] \setminus S^*$ and $r \in [n] \setminus (S \cup R)$. If $G(S, -r)$ is not full-dimensional, then $G(S^*, -r)$ is not full-dimensional and hence r is redundant by Observation 4.3.3.

If $G(S, -r)$ is full-dimensional and x^* is an interior point, then we do ray shooting from z^ϵ to x^*. Note that $A_r x^* > b_r$, hence x^* is not in the feasible region of $Ax \leq b$. Then the first constraint hit (with index q) is nonredundant. This constraint is unique, since z^ϵ is generic and x^* is not a feasible solution. Denote the intersection point of the hyperplane given by $A_q x = b_q$ with the ray by y^*. It follows that $q \notin S$ since $A_q y^* = b_q$ and we know that $A_S y^* < b_S$. This proves correctness of the algorithm.

It remains to discuss the running time. Since in every round we add either a variable to S or R, the outer loop is executed n times. In every

round we run the algorithm of Theorem 4.3.1 on at most s inequalities. This takes time $O(nd^2 s \log s)$ by Theorem 4.3.1. Moreover there are at most s stages of ray shooting which takes $O(nds)$ time in total. The running time follows. □

4.4 Revision of the Hochbaum-Naor Method

Since we modify Hochbaum and Naor's Method in the next section, for completeness we review the basic components and the key ideas of the algorithm.

Theorem 4.4.1. *[29] For a system $Ax \leq b$ in $LI(2)$ one can decide whether the problem is feasible in time $O(d^2 n \log n)$, and in the feasible case output a solution.*

In Section 4.4.1 we will give all relevant tools to prove the theorem. We then discuss the feasible case in Sections 4.4.2 and 4.4.3, and finally the infeasible case in Section 4.4.4

4.4.1 The Ingredients

The Hochbaum-Naor Theorem is a mix of an efficient implementation of the Fourier-Motzkin method and the result by Aspvall and Shiloach described below [2]. In general the Fourier-Motzkin method may generate an exponential number of inequalities, however in the $LI(2)$ case one can implement it efficiently.

The Fourier-Motzkin Method (for general LPs) (For more details please refer to [40, pp. 155 - 156]). Let G be a set of inequalities on the variables $x_1, \ldots, x_d$. The Fourier-Motzkin Method eliminates the variables one by one to obtain a feasible solution or a certificate of infeasibility. At step i, the LP only contains variables $x_i, \ldots, x_d$. Let us denote this system by G_i. To eliminate variable x_i, all inequalities that contain x_i are written as $x_i \leq h$ or $x_i \geq \ell$, where h and ℓ are

some linear functions in $x_{i+1}, \ldots, x_d$. Let us denote the two families of inequalities obtained by H and L, respectively. For each $h \in H$ and each $\ell \in L$ we add a new inequality $\ell \leq h$. This yields a set of inequalities G_{i+1}, on the variables $x_{i+1}, \ldots, x_d$. The method is feasibility preserving and given a solution to G_{i+1}, one can construct a solution of G_i in time $O(|G_i|)$.

The Aspvall-Shiloach Method Hochbaum and Naor's algorithm strongly relies on the following result by Aspvall and Shiloach [2].

For $G \in LI(2)$ let g, $h \in G$ be of form $g : \alpha_1 x + \beta_1 y \leq \gamma_1$, $h : \alpha_2 y + \beta_2 z \leq \gamma_2$, for $\alpha_1, \alpha_2, \beta_1 \neq 0$, i.e., g and h share a variable. If $\beta_1 > 0$ and $\alpha_2 < 0$, one can *update g with h* to get a bound on x in terms of z as follows:

Assume that $\alpha_1 > 0$, it follows that $x \leq \frac{\gamma_1 - \beta_1 y}{\alpha_1}$ and $y \geq \frac{\gamma_2 - \beta_2 z}{\alpha_2}$, and hence

$$x \leq \frac{\gamma_1 - \beta_1 y}{\alpha_1} \leq \frac{\gamma_1 - \beta_1 \frac{\gamma_2 - \beta_2 z}{\alpha_2}}{\alpha_1}.$$

In the case that $\alpha_1 < 0$ we get a lower bound on x in terms of z. If $\beta_1 > 0$ and $\alpha_2 < 0$ one can similarly update x. If the signs of β_1 and α_2 are the same, then no update on x is possible.

For an ordered sequence $(g_1, g_2, \ldots g_k)$ with of $g_i : \alpha_i x_i + \beta_i y_i \leq \gamma_i\}$, $g_i \in G$, $i \in [k]$ $(k \geq 1)$ we can do a sequence of updates to the bound on x_i, iff for all $i \in [k-1]$, $y_i = x_{i+1}$ and $\text{sign}(\beta_i) = -\text{sign}(\alpha_{i+1}) \neq 0$. If $\beta_k = 0$ this is called a *chain* of length k. If $\beta_k \neq 0$ and $y_k = x_1$, this is called *cycle* of length k. A chain or a cycle is called *simple*, if every inequality appears at most once. (This concept was first introduced in [42]).

Let us illustrate this on the following example of a perturbed cube, where $d = 3$ and $n = 6$ (see Figure 4.2).

1. $x - \frac{1}{3}y \geq \frac{1}{12}$
2. $x - \frac{1}{3}y \leq 1$

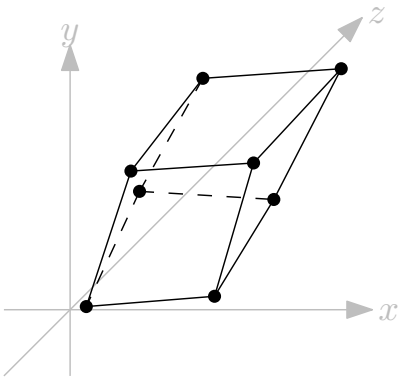

Figure 4.2: Perturbed cube, $x^{\min} = \frac{6}{72}$, $x^{\max} = \frac{204}{143}$

3. $y - \frac{1}{3}z \geq 0$
4. $y - \frac{1}{4}z \leq 1$
5. $z - \frac{1}{8}x \geq 0$
6. $z - \frac{1}{12}x \leq 1$

For a variable x_i we denote by x_i^{low} (x_i^{high}), the best lower (upper) bound that can be obtained by considering all simple chains and cycles of length at most d.

In our perturbed cube example using from chain $(2, 4, 6)$ we get the following upper bound for x.

$$\begin{aligned} x &\leq 1 + \frac{1}{3}y \\ &\leq 1 + \frac{1}{3}\left(1 + \frac{1}{4}z\right) \\ &= \frac{4}{3} + \frac{1}{12}z \\ &\leq \frac{4}{3} + \frac{1}{12}\left(1 + \frac{1}{12}x\right) \\ &= \frac{17}{12} + \frac{1}{144}x. \end{aligned}$$

It hence follows that $x \leq \frac{204}{143}$.

In the example all bounds for x that can be obtained by considering all simple chains and cycles of length at most d are the following. The numbers in the brackets denote the chains of inequalities we use to obtain the bound.

(a) $x \leq \frac{17}{12} + \frac{1}{144}x$, (2,4,6),

(b) $x \leq 8 + \frac{2}{3}x$, (5,6),

(c) $x \leq -6 + 72x$, (5,3,1),

(d) $x \geq -\frac{11}{12} + x$, (1,2),

(e) $x \geq \frac{1}{12} + \frac{1}{72}x$, (1,3,5),

(f) $x \geq -12 + \frac{3}{2}x$, (6,5),

(g) $x \geq -204 + 144x$, (6,4,2).

This translates to the system

(a') $x \leq \frac{204}{143}$, (2,4,6),

(b') $x \leq 24$, (5,6),

(c') $x \geq \frac{6}{71}$, (5,3,1),

(d') $0 \geq -\frac{11}{12}$, (1,2),

(e') $x \geq \frac{6}{71}$, (1,3,5),

(f') $x \leq 24$, (6,5),

(g') $x \leq \frac{204}{143}$, (6,4,2).

Note that (d') is a tautology and will be ignored in the calculations from now on. In our example $x^{\min} = x^{\text{low}} = \frac{6}{71}$ and $x^{\max} = x^{\text{high}} = \frac{204}{143}$. Recall that the interval $[x_i^{\min}, x_i^{\max}]$ denotes the range of x_i, $x_i^{\min}$ ($x_i^{\max}$) is the smallest (largest) value that x_i can take such that the system still has a feasible solution.

More generally, if a system $Ax \leq b$ is feasible, one can show that $[x_i^{\text{low}}, x_i^{\text{high}}] = [x_i^{\min}, x_i^{\max}]$ [2]. The idea of the proof is as follows. The direction $[x_i^{\min}, x_i^{\max}] \subseteq [x_i^{\text{low}}, x_i^{\text{high}}]$ follows immediately from the definitions. For the other direction let p be a point of $P = \{x \mid Ax \leq b\}$ such that $p_i = x_i^{\min}$. Looking at the simple chains and simple cycles of the halfspaces that contain p on their boundary, gives us the lower bound $x_i^{\min}$. The case for x_i^{high} and $x_i^{\max}$ is equivalent.

On the other hand, if the system is infeasible then two things can happen. If $x_i^{\text{high}} < x_i^{\text{low}}$, then the range of x_i is empty and this is a certificate for infeasibility. Such a certificate may not exist in general. This is the case for example if the linear system consists of two independent subsystems, one feasible and one infeasible. Also for the small (infeasible) example $y + z \leq -1$, $y + z \geq 1$, $x + y \leq 1$, we have that $x^{\text{low}} = -\infty$ and $x^{\text{high}} = \infty$.

From now on we will only discuss the case where $Ax \leq b$ is feasible. In Section 4.4.4 we will discuss how to get an infeasibility certificate using the same algorithm.

We have seen that considering all chains and cycles of length at most d, we can find the range of any variable. However, the problem is that this number is exponential. Hence instead of computing $[x_i^{\min}, x_i^{\max}]$ exactly we use the following result from Aspvall and Shiloach.

Theorem 4.4.2. *[2] Given a feasible system $Ax \leq b$ in $LI(2)$, a variable x_i and a value $\lambda \in \mathbb{R}$, one can decide in time $O(nd)$ whether $\lambda < x_i^{min}$, $\lambda = x_i^{min}$, $\lambda \in (x_i^{min}, x_i^{max})$, $\lambda = x_i^{max}$ or $\lambda > x_i^{max}$.*

In the feasible case this algorithm decides whether λ lies in the open range $(x_i^{\min}, x_i^{\max})$, on boundary of the range, or outside of the range (and on which side).

The proof of Theorem 4.4.2 requires many technical details. In the following we will summarize the method and provide an intuitive idea. For detailed proofs refer to [2, 29].

Let us start by considering once more the perturbed cube example. We

start with the inequality (a)

$$x \leq \frac{17}{12} + \frac{1}{144}x.$$

We rewrite this to

$$f(x) = \frac{17}{12} + \frac{1}{144}x.$$

We see that some value $\lambda \in \mathbb{R}$ satisfies (a), if and only if

$$\lambda \leq f(\lambda).$$

Now for some feasible system $Ax \leq b$ and we apply this construction to all inequalities obtained from the chains and loops. We obtain inequalities of form $x \leq f(x)$ and $x \geq g(x)$. Let us denote by $\mathcal{U}$ the set of functions f that we obtain from inequalities of form $x \leq f(x)$ and by $\mathcal{L}$ the set of functions g that we obtain from inequalities of form $x \geq g(x)$. Let

$$\underline{x}_i(\lambda) = \min\{f(\lambda) \mid f \in \mathcal{U}\},$$

and similarly

$$\overline{x}_i(\lambda) = \max\{g(\lambda) \mid g \in \mathcal{L}\}.$$

It follows that the function $\underline{x}_i(\lambda)$ is convex, whereas $\overline{x}_i(\lambda)$ is concave (see Figure 4.3). We can observe that $\lambda \in [x_i^{\min}, x_i^{\max}]$ if and only if $\underline{x}_i(\lambda) \leq \lambda \leq \overline{x}_i(\lambda)$.

For a fixed value λ, the Aspvall-Shiloach Algorithm can find $\underline{x}_i(\lambda)$ and $\overline{x}_i(\lambda)$ in time $O(nd)$ and by the above discussion can hence decide whether $\lambda \in [x_i^{\min}, x_i^{\max}]$. The algorithm can also find the slopes of $\underline{x}_i(\lambda)$ and $\overline{x}_i(\lambda)$ and use this to decide all cases of Theorem 4.4.2. We discuss this later.

For a variable x_i we say that c is the trivial upper bound on x_i, if there there exists a constraint $x_i \leq c$ in G and there is no constraints $x_i \leq c'$ with $c' < c$. Similarly the trivial lower bound can be defined. If there are no constraints of form $x_i \leq c$, $(x_i \geq c)$, we say that the trivial upper (lower) bound is ∞, $(-\infty)$. Let x_i be a fixed variable and $\lambda \in \mathbb{R}$. For all $j \in [n]$ let us denote by $\mathrm{lo}(x_j)$ $(\mathrm{up}(x_j))$ the trivial lower (upper) bound on x_j given by G, which may also be infinite.

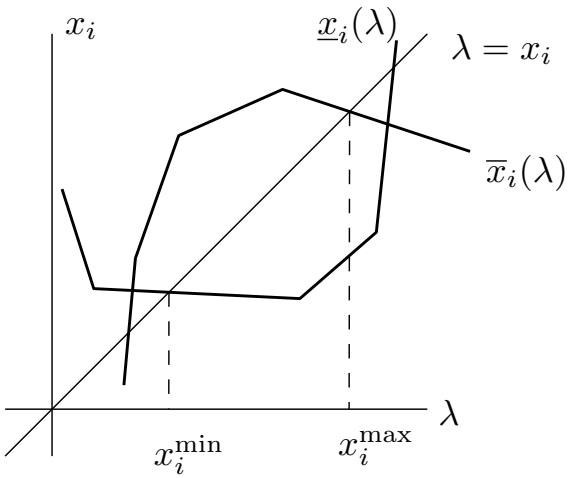

Figure 4.3: Output Aspvall-Shiloach

The following algorithm fixes $x_i = \lambda$ for the first round and returns upper and lower bounds on all x_j accordingly. In d rounds, it updates the lower and upper bounds, denoted by $\underline{x}_j$ and $\overline{x}_j$, respectively, on all variables, where initially we are given $\underline{x}_j := \text{lo}(x_j)$ and $\overline{x}_j := \text{up}(x_j)$. In the end the algorithm returns the bounds on x_i.

We observe that if $\lambda < \text{lo}(x_i)$ or $\lambda > \text{up}(x_i)$, then trivially $\lambda < x^{\min}$, $\lambda > x^{\max}$. Therefore we assume that the input λ is in $[\text{lo}(x_i), \text{up}(x_i)]$.

Algorithm Aspvall-Shiloach (G, i, λ);
begin
  for $j = 1, \ldots, d$ **do**
    $\underline{x}_j := \text{lo}(x_j), \overline{x}_j := \text{up}(x_j)$;
  endfor;
  $\underline{x}_i := \lambda$, $\overline{x}_i := \lambda$;
  for $m = 1, \ldots, d$ **do**
    for $g \in G$, with $g : ax_j + bx_k \leq c$, $a, b \neq 0$ **do**
      if $a > 0, b > 0$ **then**
        $x_j \leq \frac{c - b\underline{x}_k}{a} =: \overline{x}_j^g$, $x_k \leq \frac{c - a\underline{x}_j}{b} =: \overline{x}_k^g$;
      elseif $a > 0, b < 0$ **then**
        $x_j \leq \frac{c - b\overline{x}_k}{a} =: \overline{x}_j^g$, $x_k \geq \frac{c - a\overline{x}_j}{b} =: \underline{x}_k^g$;
      elseif $a < 0, b > 0$ **then**

$x_j \geq \frac{c-b\overline{x}_k}{a} =: \underline{x}_j^g$, $x_k \leq \frac{c-a\overline{x}_j}{b} =: \overline{x}_k^g$;
else /*$a < 0, b < 0$*/ **then**
$x_j \geq \frac{c-b\underline{x}_k}{a} =: \underline{x}_j^g$, $x_k \geq \frac{c-a\underline{x}_j}{b} =: \underline{x}_k^g$;
end for;
for $\ell = 1, \ldots, d$, $\ell \neq i$ **do**
$\underline{x}_\ell := \max_g\{\underline{x}_\ell, \underline{x}_\ell^g\}$, $\overline{x}_\ell := \min_g\{\overline{x}_\ell, \overline{x}_\ell^g\}$;
endfor;
if $m = 1$ **do**
$\underline{x}_i := \max_g\{\underline{x}_i^g\}$, $\overline{x}_i := \min_g\{\overline{x}_i^g\}$;
else /*$m > 1$ */ **then**
$\underline{x}_i := \max_g\{\underline{x}_i, \underline{x}_i^g\}$, $\overline{x}_i := \min_g\{\overline{x}_i, \overline{x}_i^g\}$;
endfor;
output $\underline{x}_i, \overline{x}_i$;
end.

The algorithm runs in d rounds of $O(n)$ steps each, which results in the running time of $O(nd)$.

Let us run the algorithm on the cube example. Let us set $\lambda = 1$. We know that $1 \in (x^{\min}, x^{\max})$. By plugging in $x = 1$ into inequalities (a) - (g) we see that

$$\underline{x}(1) = \frac{7}{12},$$

(by inequalilty (e)), and

$$\overline{x}(1) = \frac{205}{144},$$

(by inequality (a)).

The algorithm finds these values as follows. In the beginning $\underline{x} = \overline{x} = 1$, $\underline{y} = \underline{z} = -\infty$ and $\overline{y} = \overline{z} = \infty$

In the first round we hence get the following bounds. If we consider inequality 1. it follows that $y \leq 3x - \frac{1}{4} = 3 - \frac{1}{4} = \frac{11}{4}$. If we consider all inequalities 1. - 6. we obtain

1. $y \leq \frac{11}{4}$

2. $y \geq 0$

5. $z \geq \frac{1}{8}$

6. $z \leq \frac{13}{12}$.

Hence in the beginning of the second round we now have $\underline{x} = -\infty$, $\overline{x} = \infty$, $\underline{y} = 0$, $\overline{y} = \frac{11}{4}$, $\underline{z} = \frac{1}{8}$ and $\overline{z} = \frac{13}{12}$. For the second round we use these new bounds to obtain.

1. $x \geq \frac{1}{12}$,

2. $x \leq \frac{23}{12}$,

3. $y \geq \frac{1}{24}$, $z \leq \frac{33}{4}$,

4. $y \leq \frac{61}{48}$, $z \geq -4$,

5. $x \leq \frac{26}{3}$,

6. $x \geq -\frac{21}{2}$,

It follows that $\underline{x} = \frac{1}{12}$, $\overline{x} = \frac{23}{12}$, $\underline{y} = \frac{1}{24}$, $\overline{y} = \frac{61}{48}$, $\underline{z} = \frac{1}{8}$ and $\overline{z} = \frac{13}{12}$. Plugging in these values once more we see that indeed 1. implies $x \geq \frac{7}{72} = \underline{x}(1)$ and 2. implies $x \leq \frac{205}{144} = \overline{x}(1)$.

As already mentioned in general $\lambda \in [x_i^{\min}, x_i^{\max}]$, if and only if $\underline{x}_i(\lambda) \leq \lambda \leq \overline{x}_i(\lambda)$. It follows that $x_i^{\min} = \min\{\lambda \mid \underline{x}_i(\lambda) \leq \lambda \leq \overline{x}_i(\lambda)\}$ and $x_i^{\max} = \max\{\lambda \mid \underline{x}_i(\lambda) \leq \lambda \leq \overline{x}_i(\lambda)\}$. To distinguish between all cases of Theorem 4.4.2, we additionally need the (left and right) slopes of $\underline{x}_i(\lambda)$ and $\overline{x}_i(\lambda)$. It is not hard to modify the Aspvall-Shiloach Algorithm in such a way that it keeps track of the slopes as well. For instance if $\underline{x}_i(\lambda) = \lambda \leq \overline{x}_i(\lambda)$ and the slope of $\underline{x}_i$ is smaller than one at λ, then $\lambda = x_i^{\min}$. In the case where the slope is greater than one $\lambda = x_i^{\max}$. Using a careful case analysis one can show that the for given λ, the values of $\underline{x}_i(\lambda)$ and $\overline{x}_i(\lambda)$ and their (left and right) slopes at λ are enough to decide all cases of Theorem 4.4.2.

4.4.2 The Hochbaum-Naor Algorithm for the Feasible Case

The rough idea of the Hochbaum-Naor algorithm is the following. At step i we want to efficiently find λ in the current range $[x_i^{\min}, x_i^{\max}]$ and set $x_i = \lambda$ to obtain a system with one less variable. Whenever this is not possible, we eliminate x_i efficiently in a Fourier-Motzkin step. After this first part we set all variables that were eliminated to values in their current range in the normal Fourier-Motzkin backtracking step.

First Part The first part of the algorithm runs in d steps. In step i we update two linear systems G^{i+1} and H^{i+1} from G^i and H^i respectively, where initially $G^1 = H^1 = G$. The systems G^i (on $d - i + 1$ variables) and H^i (on d variables) do basically encode the same solution system, we will see later why a distinction is necessary. During the execution of the algorithm, G^i is used to do Fourier-Motzkin elimination method, H^i is used to run the algorithm of Theorem 4.4.2. We denote by $\mathrm{FM}(x_i, G)$ the set of inequalities obtained by eliminating x_i from G by using one step of the Fourier-Motzkin elimination method.

For any two variables x_j and x_k in G, with $j < k$ we represent the set of inequalities containing x_j and x_k (with nonzero coefficients) in the (x_j, x_k) plane as two envelopes, the upper envelope and the lower envelope. The feasible region of x_j and x_k is contained between the envelopes (in the highlighted region of Figure 4.4) and each envelope can be represented as a piecewise linear function with breakpoints. The projection of the breakpoints onto the j-axis is denoted by $B_k^j(G)$. If the envelope is unbounded in the x_j-direction we add points at infinity to $B_k^j(G)$. The range of x_j is hence contained in the interval given by the leftmost and rightmost point of $B_k^j(G)$.

Below follows the pseudo code and the explanation of the algorithm.

Algorithm Hochbaum-Naor (G);
begin
$G^1 = H^1 = G$;

for $i = 1, \ldots, d-1$ **do**
Generate $B^i = (b_1^i, \ldots, b_m^i)$, the sorted sequence of the points in $\bigcup_{x_j \in N(x_i, G_i)} B_j^i(G^i)$;
Use Theorem 4.4.2 to do binary search on B^i;
if $\exists \ell \in \{1, \ldots, m\}$ such that $x_i^{\min}(H^i) \leq b_\ell^i \leq x_i^{\max}(H^i)$
then
$G^{i+1} := G^i|_{x_i = b_\ell}$;
$H^{i+1} := H^i \cup \{x_i \leq b_\ell\} \cup \{x_i \geq b_\ell\}$;
elseif $\exists \ell (1 \leq \ell < m)$ s.t. $b_\ell^i < x_i^{\min}(H^i)$
and $x_i^{\max}(H^i) < b_{\ell+1}^i$ **then**
$G^i := \text{rel}(G^i) \cup \{x_i \geq b_\ell^i\} \cup \{x_i \leq b_{\ell+1}^i\}$;
$G^{i+1} := \text{FM}(x_i, G^i)$;
$H^{i+1} := H^i$;
else then
output *system infeasible*;
endif;
endfor;
end.

Here $\text{rel}(G^i)$ denotes the so called *relevant* inequalities in G_i which we obtain by removing some redundant inequalities. The exact definition follows in the description of the algorithm below.

In step i the algorithm has computed G^i and H^i, where originally $G^1 = H^1 = G$. For every pair (x_i, x_j), such that x_j is a neighbor of x_i in G^i, i.e., $x_j \in N(x_i, G_i)$, it computes the projections of the breakpoints $B_j^i(G^i)$. The union of those points are sorted and denoted by B^i. The idea is now to run a binary search on B^i using Theorem 4.4.2, in the hope of finding a point in the range of x_i.

If the algorithm finds a breakpoint $b_\ell^i \in B^i$ such that $x_i^{\min}(H^i) \leq b_\ell^i \leq x_i^{\max}(H^i)$, then we set $x_i := b_\ell^i$ (see Figure 4.4). We set $H^{i+1} = H^i \cup \{x_i \leq b_\ell^i\} \cup \{x_i \geq b_\ell^i\}$ and $G^{i+1} := G^i|_{x_i = b_\ell^i}$.

If there is no such b_ℓ^i, the algorithm finds b_ℓ^i, $b_{\ell+1}^i$ such that $b_\ell^i < x_i^{\min}(H^i)$ and $x_i^{\max}(H^i) < b_{\ell+1}^i$. In that case for any neighbor x_j of

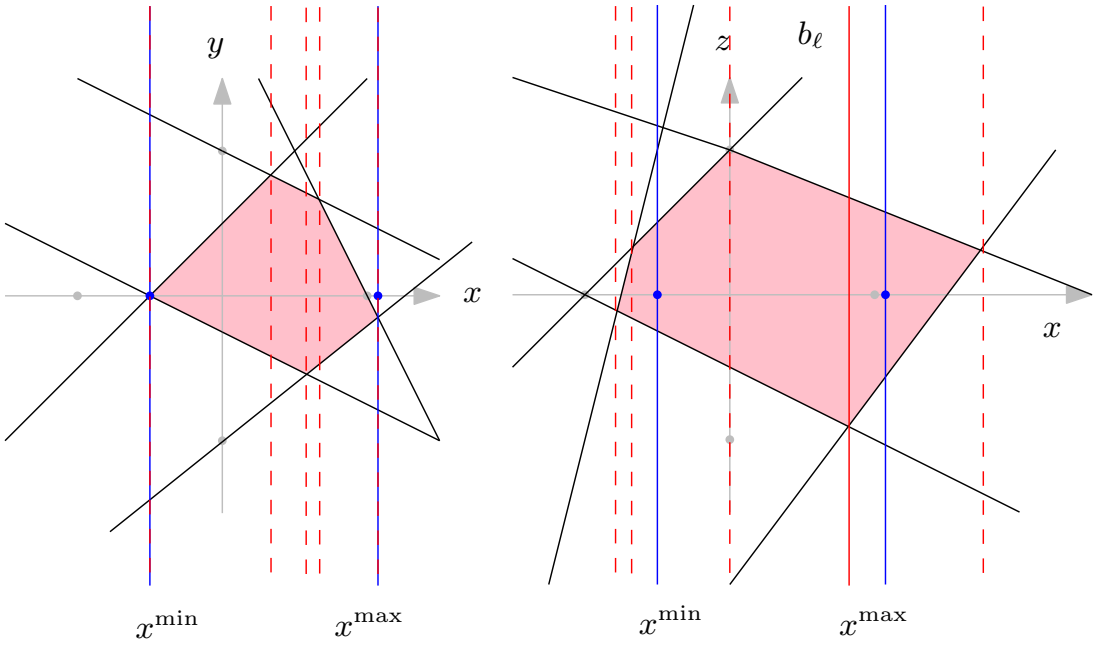

Figure 4.4: First case Hochbaum-Naor, example with 3 variables

x_i the number of inequalities containing both is reduced to at most two (bold lines in Figure 4.5), the ones that define the upper and lower envelope respectively on the interval $[b^i_\ell, b^i_{\ell+1}]$ (strip of Figure 4.5). This can be done since $[x^{\min}_i, x^{\max}_i] \subseteq [b^i_\ell, b^i_{\ell+1}]$ and therefore all other inequalities are redundant and can be removed. We denote the set of inequalities obtained after the removal of the redundant ones by $\text{rel}(G^i)$. The normal Fourier-Motzkin elimination is applied on $\text{rel}(G^i) \cup \{x_i \geq b^i_\ell\} \cup \{x_i \leq b^i_{\ell+1}\}$ to obtain G^{i+1}. By the above discussion $\text{rel}(G^i) \cup \{x_i \geq b^i_\ell\} \cup \{x_i \leq b^i_{\ell+1}\}$ has the same solution space as G^i. As the number of inequalities between x^i and any neighbor x^j is reduced to at most two, the algorithm adds at most four inequalities between any pair of neighbors of x^i. This prevents the usual quadratic blowup of the Fourier-Motzkin Method. The system H^{i+1} does not need to be updated, i.e., $H^{i+1} = H^i$.

Remark 4.4.3. We observe that in every step only a constant number of constraints are added to H^i, which guarantees the running time of the binary search to be $O(nd \log n)$. The size of G^i can be of order $\Theta(n + d^2)$ (as we may add up to $4d$ constraints in each step), hence running Theorem 4.4.2 on G^i would not guarantee the running time

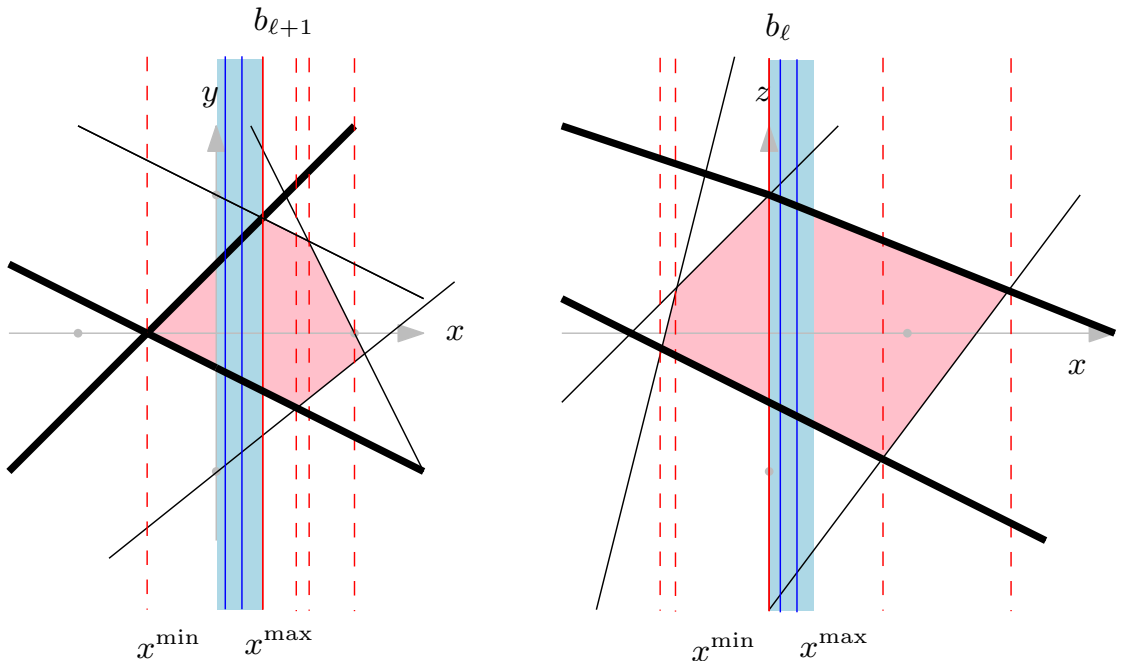

Figure 4.5: Second case Hochbaum-Naor, example with 3 variables

in case where $n = o(d^2)$. It follows that we indeed need to distinguish between the two systems G^i and H^i.

Second Part The second part of the algorithm is now the normal backtracking of the Fourier-Motzkin Method. Assume that the variables that were eliminated (in the elseif-step) are $x_{i_1}, \ldots, x_{i_k}$, where $k \leq d$ and $i_1 < i_2 < \cdots < i_k$. In the end of part one we are left with the system G^d on variable x_d. By the properties of the Fourier-Motzkin elimination G^d is feasible and the range of x_d is the same as its range in H^d. Now choose a feasible value of x_d and continue inductively by backtracking through $G^{i_k}, \ldots, G^{i_1}$. The geometric interpretation is similar to the first part: for each variable we pick a value in its current range.

4.4.3 Discussion of the Hochbaum Naor Algorithm

We briefly discuss the main points of the proof of Theorem 4.4.1 (for more detail see [29]).

Proof sketch of Theorem 4.4.1 for the feasible case. Building and updating all envelopes takes $O(dn \log n)$ time per step, hence $O(d^2 n \log n)$ in total. Since for all i, the size of H^i is $O(n)$ and $|B^i| \leq n + 4d$, in each step the binary search runs Theorem 4.4.2 $O(\log n)$ times, where each evaluation takes time $O(dn)$. It follows that the first part of the algorithm takes time $O(d^2 n \log n)$. For the second part consider the step where we find a solution for a variable x_j in the backtracking step in G_i. Then x_j shares at most two inequalities with each of its neighbors, therefore the whole second part only takes time $O(d^2)$.

During the whole algorithm, the variable x_i is set to some value λ if and only if λ is in the current bounds $[x_i^{\min}, x_i^{\max}]$. Therefore in the feasible case, it correctly outputs a feasible point of $Ax \leq b$. □

4.4.4 Discussion of the Hochbaum-Naor Algorithm in the Infeasible Case

In the previous section we showed that if $Ax \leq b$ is feasible, then the Hochbaum-Naor Method always correctly outputs a feasible point. We now show that in the infeasible case, infeasibility is always detected, which completes the proof of Theorem 4.4.1.

Proof sketch of Theorem 4.4.1 for the infeasible case. Assume that $Ax \leq b$ is infeasible. We now run the first part of the algorithm as in the feasible case. If during the execution at some point during binary search, we detect a contradiction in the form of $x_i^{\max} < x_i^{\min}$ this is a certificate for infeasibility and we are done. It is however possible that in every call of the algorithm of Theorem 4.4.2 we get some (wrong) output $\lambda < x_i^{\min}$, $\lambda \in [x_i^{\min}, x_i^{\max}]$, $\lambda > x_i^{\max}$. In that case G^d is an infeasible system on the variable x_d. This follows since the Fourier-Motzkin elimination is feasibility preserving and setting some variables to fixed values in an infeasible system, keeps the system infeasible. Detecting infeasibility in a system with one variable can be trivially done in linear time in the number of constraints. It follows that infeasibility is always detected, which concludes the proof of Theorem 4.4.1. □

4.5 Modification of Hochbaum-Naor Method

We now show how to modify the Hochbaum-Naor Method from Section 4.4, such that it decides full-dimensionality of the problem and in the full-dimensional case outputs an interior point (see Theorem 4.3.1). For this we need some preparatory lemmas.

Lemma 4.5.1. *Let $Ax \leq b$ be feasible and let $\lambda \in (x_1^{\min}, x_1^{\max})$. Then $y := (\lambda, y_2, \ldots, y_d)$ is an interior point solution of $Ax \leq b$ if and only if $y' := (y_2, \ldots, y_d)$ is an interior point solution of $A'x \leq b'$, where $A'x \leq b'$ is the system obtained by $(Ax \leq b)|_{x_1=\lambda}$.*

Proof. Let y be an interior point solution of $Ax \leq b$. Then by definition $Ay < b$ and obviously $A'y' < b'$. On the other hand let y' be an interior point solution of $A'x < b'$, i.e., $A'y' < b'$. Then y satisfies any inequalities containing some $x_i \neq x_1$ strictly. The only inequalities that might be satisfied with equality are the ones containing only x_1, but this is a contradiction to $\lambda \in (x_1^{\min}, x_1^{\max})$. □

Lemma 4.5.2. *In the Fourier-Motzkin Method G_{i+1} has an interior point solution, if G_i has one. Moreover if an interior point solution of $G_1 = G$ exists, it can be obtained in the running time of the Fourier-Motzkin algorithm.*

Proof. The first part follows by Lemma 4.5.1. For the second part it is enough to consider a slight variant of the Fourier-Motzkin elimination. Instead of running the algorithm on a system $Ax \leq b$, we run it on $Ax < b$. In each step the inequalities obtained are of form $l < h$ instead of $l \leq h$. By induction, using Lemma 4.5.1 one can see that finding a solution using this variant, is equivalent to finding an interior point of $Ax \leq b$. □

Proof of Theorem 4.3.1. Assume that $Ax \leq b$ is feasible. We run the Hochbaum-Naor Algorithm almost in the same way as described in Section 4.4. The only difference is in the if-loop of the algorithm. The original algorithm distinguishes between the cases $\lambda < x_i^{\min}$, $\lambda \in$

$[x_i^{\min}, x_i^{\max}]$ and $\lambda > x_i^{\max}$ of Theorem 4.4.2. Our algorithm however distinguishes between $\lambda \leq x_i^{\min}$, $\lambda \in (x_i^{\min}, x_i^{\max})$ and $\lambda \geq x_i^{\max}$.

In the first case we only set $x_i = b_\ell^i$ if there exists a breakpoint $b_\ell^i \in B$, such that $x_i^{\min}(H^i) < b_\ell^i < x_i^{\max}(H^i)$, that is, we only fix x_i to some value b_ℓ^i, if b_ℓ^i is in the *open* range $(x_i^{\min}(H^i), x_i^{\max}(H^i))$, (in the original Theorem we were considering the closed range). The second case accordingly changes to finding an interval $[b_\ell^i, b_{\ell+1}^i]$ $(1 \leq \ell < k)$ such that $b_\ell^i \leq x_i^{\min}(H^i)$ and $x_i^{\max}(H^i) \leq b_{\ell+1}^i$, (originally $b_\ell^i < x_i^{\min}(H^i)$ and $x_i^{\max}(H^i) < b_{\ell+1}^i$). We see that in this case, the number of inequalities on each variable adjacent to x_i is still reduced to at most two.

As Theorem 4.4.2 distinguishes the cases $\lambda < x_i^{\min}$, $\lambda = x_i^{\min}$, $\lambda \in (x_i^{\min}, x_i^{\max})$, $\lambda = x_i^{\max}$ $\lambda > x_i^{\max}$ and certificate for infeasibility in time $O(nd)$, the running time remains the same.

It remains to show that the modified algorithm detects full-dimensionality and in the full-dimensional case an interior point.

The discussion of the case where $Ax \leq b$ is infeasible is equivalent as in the proof of Theorem 4.4.1. Hence assume that the system is feasible.

Let $Ax \leq b$ be full-dimensional. By Lemma 4.5.1, after the first part of the algorithm H^d (and G^d) are associated with a system of inequalities whose interior point solutions can be extended to an interior point solution of $Ax \leq b$. The interior point solution of $Ax \leq b$ can now be found in the backtracking step using Lemma 4.5.2. Assume such a point can not be found. Then by Lemma 4.5.2 there is no interior point of $Ax \leq b$, which is a contradiction to full-dimensionality.

Let $Ax \leq b$ be feasible but non-full-dimensional. Then at some point of the backtracking the algorithm finds $x_i^{\min} = x_i^{\max}$ for the current bounds. Suppose this does not happen, then by Lemma 4.5.1 the algorithm finds an interior point, which is a contradiction to non-full-dimensionality.

□

4.6 The Non-Full-Dimensional Case

In the non full-dimensional case, redundancies are dependent on each other, meaning that a redundant constraint can become nonredundant after the removal of another redundant constraint. The problem is therefore to find a maximal set of nonredundant constraints.

Clarkson's Algorithm can be extended for redundancy removal in the non-full-dimensional case as follows: In a preprocessing step one can find the dimension k of the system $Ax \leq b$, by solving at most d linear programs as shown in Theorem 2.2.2. Of all the inequalities that are forced to equality, we can find a set of $(d-k)$ equalities that defines the k-dimensional space that $P = \{x \mid Ax \leq b\}$ lies in. Let us denote the remaining system of inequalities (the ones not forced to equality) by $A_2x \leq b_2$. One can now rotate the the system such that P lies in $\mathbb{R}^k$. Clarkson's algorithm can now be applied in $\mathbb{R}^k$, where the constraints are the intersections of the rotated system of $A_2x \leq b_2$ intersected with $\mathbb{R}^k$. After the preprocessing the running time is hence $O(n \cdot LP(s,k))$.

In the case of $LI(2)$ we observe that such a rotation may destroy the structure of two variables per inequality. But we are still able to match Clarkson's running time, using substitution of variables.

Theorem 4.6.1. *Let $Ax \leq b$ a k-dimensional system in $LI(2)$, for $0 \leq k \leq d$. Then given a relative interior point solution of $Ax \leq b$, all redundancies can be detected in time $O(nk^2 s \log s + d^2 n)$.*

The d^2n term comes from Gaussian elimination, which is dominated by the preprocessing time needed to find the relative interior point (see Proposition 4.6.3). Note that the typically larger term $nk^2 s \log s$ is dependent on the dimension k of the polyhedron and not d.

We need the following observation for the proof of Theorem 4.6.1.

Observation 4.6.2. *Let $Ax \leq b$ in $LI(2)$ and $\alpha x_i + \beta x_j \leq \gamma$, $\beta \neq 0$, an inequality of the system that is forced to equality, i.e., $\alpha x_i^* + \beta x_j^* = \gamma$, for all solutions x^*. Let $\overline{A}x \leq \overline{b}$ be the system obtained by substituting $x_j = \frac{\gamma}{\beta} - \frac{\alpha}{\beta} x_i$. Then the following holds.*

- $\overline{A}x \leq \overline{b}$ *is still in LI(2).*
- *A constraint is redundant in* $Ax \leq b$ *if and only if it is redundant in the system* $\overline{A}x \leq \overline{b}$.

Proof of Theorem 4.6.1. Given a relative interior point x^*, one can find $A_1x \leq b_1$, the subsystem of $Ax \leq b$ that is forced to be equality, in time $O(nd)$. The remaining system is denoted by $A_2x \leq b_2$. Finding a minimal subsystem $A_1^*x = b_1^*$ of $A_1x = b_1$ with $(d-k)$ linearly independent equalities that defines the k-dimensional space containing $P = \{x \mid Ax \leq b\}$, takes $O(d^2n)$ time using the Gaussian elimination. Using these equalities we can substitute $d-k$ variables of $A_2x \leq b_2$ in the same fashion as explained in Observation 4.6.2. Hence we get a k-dimensional representation of $Ax \leq b$ which is in $LI(2)$.

We can now run the algorithm given by Theorem 5.3.5 on $A_2'x \leq b_2'$, the system obtained from $A_2x \leq b_2$ after substitution. These detected nonredundant constraints together with $A_1^*x = b_1^*$ give us a minimal set of nonredundant inequalities. □

The following proposition shows that finding a relative interior point can also be done in strongly polynomial time.

Proposition 4.6.3. *Given* $Ax \leq b$*, one can find a relative interior point of* $Ax \leq b$ *or a certificate for infeasibility in time* $O(d^2n \log n)$.

Proof. Using a similar argument as in Lemma 4.5.1, one can show the following.

If $x_1^{\min} \leq \lambda \leq x_1^{\max}$, then $y := (\lambda, y_2, \ldots, y_d)$ is a relative interior point of $Ax \leq b$ if and only if $y' := (y_2, \ldots, y_d)$ is a relative interior point of $(Ax \leq b)|_{x_1=\lambda}$.

The algorithm for finding a relative interior point is very similar to the modified Hochbaum-Naor Method. The first part runs equivalently. In the second part of the backtracking if at some point $x_i^{\min} = x_i^{\max}$, we set $x_i := x_i^{\min} = x_i^{\max}$. Infeasibility is detected by the same argument as in Theorem 4.4.1. Correctness follows from the above argument, the running time is the same as in Theorem 4.3.1. □

Corollary 4.6.4. *The dimension of $Ax \leq b$ or a certificate for infeasibility can be found in time $O(d^2 n \log n)$, i.e., the same running time as finding a feasible point solution of a certificate for infeasibility.*

Proof. Consider the algorithm of the proof of Proposition 4.6.3. Since we know that infeasibility will be detected, assume that $Ax \leq b$ is feasible. Let us denote the current polyhedron by P, where initially $P = \{x \mid Ax \leq b\}$ the polyhedron defined by $Ax \leq b$. Every time x_i is set to a value in the open range $(x_i^{\min}, x_i^{\max})$, the dimension of the current polyhedron P decreases by 1, as we intersect it with a hyperplane not containing P. If $x_i^{\min} = x_i^{\max}$ and we set $x_i := x_i^{\min} = x_i^{\max}$, then the dimension of the current polyhedron stays the same, as we intersect it with a hyperplane containing P. Since after setting all to some value we end up with a point (polyhedron of dimension 0), the dimension of $Ax \leq b$ is exactly the number of times we set x_i to a value in the open range $(x_i^{\min}, x_i^{\max})$. □

4.7 Discussion and Open Questions

In the special case of linear programs with two variables per inequality, we have seen that all redundancies can be detected in strongly polynomial time $O(nd^2 s \log s)$ (Theorem 5.3.5). The first open question that arises is whether this running time can be improved. Moreover our algorithm uses geometric notions, such as ray shooting for Clarkson's algorithm, hence it would be interesting to see whether it is possible to get a combinatorial strongly polynomial time algorithm. Using the methods from Clarkson's algorithm or our alternative version, it is sufficient to find a strongly polynomial time algorithm that finds an optimal solution or an interior point solution.

It would be interesting to know in what kind of other families solving the feasibility problem is easier than optimizing. Furthermore, it would be interesting to find more subfamilies of LPs where either of this problems can be solved polynomially. Some results in this direction already exist (e.g. [44]), but there is still ample room for new research.

Chapter 5

Complexity of Polytopes with two Variables per Inequality

This chapter is based on joint work with K. Fukuda.

5.1 Introduction

As in the last chapter, we here consider linear systems with at most two variables per constraint. In this chapter we analyze the complexity of the polyhedron given by such systems. In particular, we want bounds on the number of vertices such a polyhedron can have. Since we want to analyze the maximum complexity, for simplicity in this chapter we only consider bounded polyhedra, the so-called polytopes. This is possible since every unbounded polyhedron with at least one vertex can be converted into a polytope by adding one more constraint. This can only increase the number of vertices and addition of one constraint does not change our complexity results.

Again, we assume that we are given a polytope by a system of n inequalities in d variables, of form $P = \{x \in \mathbb{R}^d \mid Ax \leq b\}$, where $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$.

Recall, that by $LI(2)$ we denote the family of linear systems $Ax \leq b$, that have at most two variables per inequality with nonzero coefficient. As described in the previous chapter for this particular family, Hochbaum and Naor's algorithm finds a feasible point or a certificate for infeasibility in time $O(d^2 n \log n)$ [29], i.e., it solves the feasibility problem in strongly polynomial time.

Because of this difference in running time, it is hence natural to ask, whether polytopes $LI(2)$ have an simpler structure than general polytopes. It was shown in [24], that polyhedra in $LI(2)$ can have shadows with high complexity.

It is known that in general the dual cyclic polytope maximizes the number of vertices for a polytope given by n constraints (see Theorem 5.2.1). In this chapter we examine a polytope in $LI(2)$, that has almost the same complexity as the dual cyclic polytope. This polytope was given in [1] in the context of deformed products. The number of vertices in this polytope is smaller by a factor that depends only on the dimension d and not on n, (see Lemma 5.3.2). A similar result can be shown not only for vertices but for all k-faces (see Theorem 5.3.3). This shows that polytopes in $LI(2)$ can have high complexity. If d is constant, then asymptotically the same complexity as the dual cyclic polytope.

We will also show in Theorem 5.4.2 that the dual cyclic polytope can not be realized in LI(2) for $d \geq 4$. In particular in the dual cyclic polytope any pair of the n facets are adjacent, however in $LI(2)$, there are at least $\Theta(n^2/\binom{d}{2})$ pairs that are not adjacent.

5.2 Definitions and Known Results

Let $P = \{x \in \mathbb{R}^d \mid Ax \leq b\}$ be a convex polytope in $\mathbb{R}^d$, where $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$. Recall that the rows of $Ax \leq b$ are called the *constraints*. The *dimension* of P, denoted $\dim(P)$, is defined as the number of affinely independent points in P minus one. A k-dimensional subset $F \subseteq P$ is a k-face of P, if F has dimension k and if there exists a hyperplane $h : ax \leq b$, such that $ax^* = b^*$ for all $x^* \in F$ and $ax^* < b^*$ for all $x^* \in P \setminus F$. The 0-dimensional faces are called the *vertices* of P, the $(d-1)$-dimensional faces are called *facets*. If F is a k-face, then $ax = b$ for at least $d - k$ constraints of $Ax \leq b$.

For $0 \leq k \leq d$ we denote by $f_k := f_k(P)$ the number of k-dimensional

faces of P. The *f-vector* of P is defined by

$$f(P) := (f_0, f_1, \ldots, f_d).$$

Theorem 5.2.1 (McMullen's Upper Bound Theorem [36, 18])**.** *The maximum number of k-faces in a d-dimensional polytope with n constraints is attained by the dual cyclic polytope $c^*(n,d)$ and is given by*

$$f_k(c^*(n,d)) = \sum_{r=\min\{k,\lceil d/2\rceil\}}^{\lceil d/2\rceil-1} \binom{n-d-1+r}{r}\binom{r}{k} + \sum_{r=\max\{k,\lceil d/2\rceil\}}^{d} \binom{n-r-1}{d-r}\binom{r}{k}.$$

In particular the number of vertices is given by

$$f_0(c^*(n,d)) = \binom{n-\lceil d/2\rceil}{n-d} + \binom{n-\lfloor d/2\rfloor-1}{n-d}.$$

Remark 5.2.2. For $\lceil d/2\rceil \le k \le d$ the formula can be simplified to

$$f_k(c^*(n,d)) = \binom{n}{d-k}.$$

This means that any $(d-k)$ constraints define a k-face.

For our calculation we will make use of the following well known formulas. Stirling's formula says that

$$n! = \Theta\left(\sqrt{n}\frac{n^n}{e^n}\right),$$

as n goes to infinity. It follows that

$$\binom{n}{k} = \mathrm{poly}(k)\cdot\frac{n^n}{k^k(n-k)^{n-k}}. \tag{5.1}$$

Furthermore we need the well known inequality

$$1 + x \leq e^x, \text{ for all } x \in R. \tag{5.2}$$

We conclude that

$$\binom{n}{k} = \text{poly}(k) \cdot \left(\frac{n}{k}\right)^k \cdot \left(1 + \frac{k}{n-k}\right)^{n-k} \leq \text{poly}(k) \cdot \left(\frac{n}{k}\right)^k \cdot e^k. \tag{5.3}$$

5.3 Lower Bound on Maximum Complexity of $LI(2)$

In the following we always assume that $\lfloor d/2 \rfloor$ is a divisor of n (if d is even) or $n-1$ (if d is odd). All results naturally extend to any $d < n$, but we would like to avoid to have even more floors and ceilings in the notation.

We want to construct a polytope in $LI(2)$, that has high complexity, i.e., with an f-vector of order close to the f-vector of the dual cyclic polytope.

In a first part let us assume that d is even. We pair the set of variables and define an $n/(d/2)$ polygon on each of the pairs. Formally, for $1 \leq i \leq d/2$, let

$$A_i \begin{pmatrix} x_{2i-1} \\ x_{2i} \end{pmatrix} \leq b_i,$$

be a polygon in the (x_{2i-1}, x_{2i})-plane, given by $n/(d/2)$ constraints with $n/(d/2)$ vertices (see Figure 5.1). We denote $P_i^* := P_i^*(n,d) = \{x \in \mathbb{R}^2 \mid A_i(x_{2i-1}, x_{2i})^T \leq b_i\}$ and by G_i the set of constraints of P_i^*.

Now $P^*(n,d)$ is defined as the d-dimensional polytope that we obtain from the union of G_i, $1 \leq i \leq d/2$. Since the P_i^*'s do not share any variables,

$$P^*(n,d) = \{x \in \mathbb{R}^d \mid (x_{2i-1}, x_{2i}) \in P_i, \text{ for all } 1 \leq i \leq d/2\}.$$

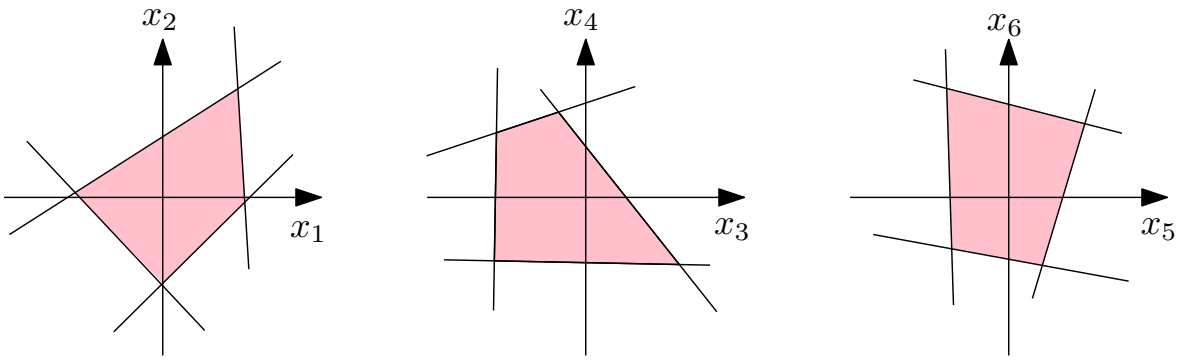

Figure 5.1: Example of $P^*(12, 3)$

For d odd, we pair the first $d-1$ variables and use the construction as above. Moreover we add the constraint $x_d \geq 0$, i.e.,

$$P^*(n, d) = \{x \in \mathbb{R}^d \mid (x_1, \ldots, x_{d-1}) \in P^*(n-1, d-1) \wedge x_d \geq 0\}.$$

Theorem 5.3.1. *[1] For d even, the polytope $P^*(n, d)$ in $LI(2)$ has the following number of vertices:*

$$\left(\frac{n}{d/2}\right)^{d/2}.$$

For d odd it is

$$\left(\frac{n-1}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}.$$

The proof of [1] is given in a much more general setting of deformed products, we will here give the proof for our special case.

Proof. Let us assume first that d is even. For $1 \leq i \leq d/2$ let $G_i = \{g_i^1, \ldots, g_i^{n/(d/2)}\}$, where the $g_i^j : a_i^j(x_{2i-1}, x_{2i})^T \leq b_i^j$, ordered in such a way that g_i^j and $g_i^{(j+1)}$, $1 \leq j \leq d/2$, define a vertex of P_i^*. Throughout the proof, $j+1$ is always considered modulo $n/(d/2)$. We will show that if for every P_i^* we choose two consecutive constraints g_i^j and g_i^{j+1}, these d constraints define a vertex of $P^*(n, d)$ and those are the only

sets of d constraints that define vertices (see also Figure 5.2). Let us denote the set of vertices of P^* by $V(P^*)$. Formally we show that

$$V(P^*) = \{x \in R^d \mid \exists(j_1, \ldots, j_{d/2}) : \\ a_i^{j_i}(x_{2i-1}, x_{2i})^T = b_i^{j_i} \wedge a_i^{j_i+1}(x_{2i-1}, x_{2i})^T = b_i^{j_i+1} \forall i\}.$$

Let us first show that the set on the right hand side is a subset of

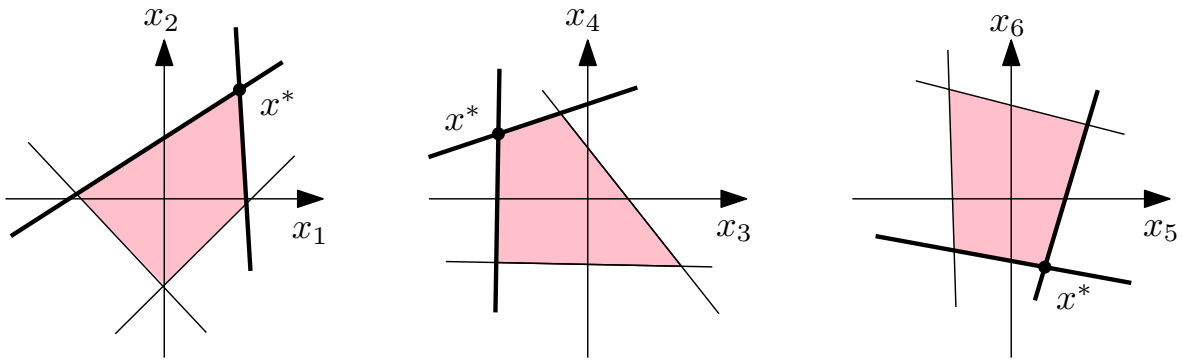

Figure 5.2: d constraints that define vertex

$V(P^*)$. We show that the g_i^1, g_i^2, $1 \leq i \leq d/2$, define a vertex, the rest follows from symmetry. Let us denote those d constraints by G' and x^* the intersection point of their boundaries. It follows that $x^* \in P^*$ because $(x_{2i-1}^*, x_{2i}^*) \in P_i^*$ for all i. We define the halfspace h by

$$h : \sum_{i=1}^{d/2}(a_i^1(x_{2i-1}, x_{2i})^T + a_i^2(x_{2i-1}, x_{2i})^T) \leq \sum_{i=1}^{d/2}(b_i^1 + b_i^2),$$

the halfspace obtained by the sum of all constraints in G'. Let us denote this halfspace by $h : a'x \leq b'$. Then by definition if follows that $a'x^* = b'$. Now let $y \in P^* \backslash x^*$. Since $y \in P^*$ it follows that $a_i^1(x_{2i-1}, x_{2i})^T \leq b_i^1$ and $a_i^2(x_{2i-1}, x_{2i})^T \leq b_i^2$ for all i. Moreover since $y \neq x^*$ there exists some k such that $a_k^1(x_{2i-1}, x_{2k})^T < b_k^1$ or $a_k^2(x_{2k-1}, x_{2k})^T < b_k^2$. It follows that $a'x < b'$, hence by definition of a 0-face, x^* is a vertex.

For the other direction we need to show that no other d constraints define a vertex (see also Figure 5.3). If we choose more than two constraints from some G_i, then the intersection of their boundaries is

empty. If we choose two constraints in G_i that are not adjacent, the point it defines in P_i^* violates some constraints of G_i. Hence, we need to choose two consecutive constraints.

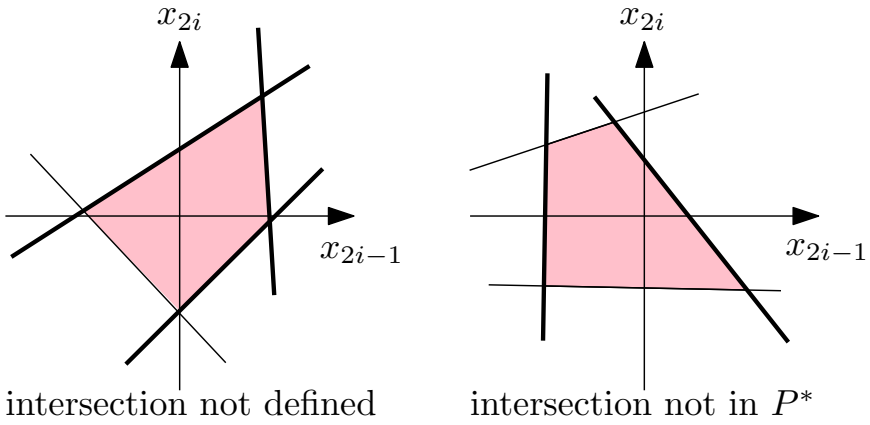

Figure 5.3: d constraints that do not define vertex

The case where d is odd is similar. The vertices of $P^*(n,d)$ are given by the constraints defining the vertices of $P^*(n-1,d-1)$ together with $x_d \geq 0$. $P^*(n-1,d-1)$ is the $d-1$ dimensional polytope defined by the constraints $G \setminus \{x_d \geq 0\}$.

The proof now follows by simple counting. □

We will compare the number of vertices between $P^*(n,d)$ and the dual cyclic polytope. Since we do not compare the exact values, but only the leading terms, we will not exactly compute the polynomial terms in d, but denote them by poly(d).

Lemma 5.3.2. *The dual cyclic polytope has a factor $O(poly(d) \cdot e^{\lfloor d/2 \rfloor})$ more vertices than $P^*(n,d)$, i.e.,*

$$f_0(c^*(n,d)) \leq poly(d) \cdot e^{\lfloor d/2 \rfloor} \cdot f_0(P^*(n,d)).$$

We see that this factor is independent of n, hence if d is constant then the number of vertices of $P^*(n,d)$ is asymptotically equal to the number of vertices of the dual cyclic polytope.

Proof. Considering only the leading term of $f_0(c^*(n,d))$ and using inequality (5.3) we get

$$\begin{aligned} f_0(c^*(n,d)) &= \binom{n-\lceil d/2\rceil}{n-d} + \binom{n-\lfloor d/2\rfloor-1}{n-d} \\ &\leq 2\cdot\binom{n-\lceil d/2\rceil}{\lfloor d/2\rfloor} \\ &\leq \mathrm{poly}(d)\cdot e^{\lfloor d/2\rfloor}\cdot\left(\frac{n-\lceil d/2\rceil}{\lfloor d/2\rfloor}\right)^{\lfloor d/2\rfloor} \\ &\leq \mathrm{poly}(d)\cdot e^{\lfloor d/2\rfloor}\cdot\left(\frac{n}{\lfloor d/2\rfloor}\right)^{\lfloor d/2\rfloor} \end{aligned}$$

Therefore

$$f_0(c^*(n,d)) \leq \mathrm{poly}(d)\cdot e^{\lfloor d/2\rfloor}\cdot f_0(P^*(n,d)).$$

□

In the following we do not only compare the number of vertices between $P^*(n,d)$ and $c^*(n,d)$, but also their f-vectors. We will see that if $k \leq \lceil d/2\rceil - 1$, then $f_k(P^*(n,d))$ is by a factor at most $e^{\lfloor d/2\rfloor}$ larger than $f_k(c^*(n,d))$. If $k \geq \lceil d/2\rceil$, then the factor is at most e^{d-k}.

Theorem 5.3.3. *For d even*

$$f_k(P^*(n,d)) = \sum_{r=\max\{0,d/2-k\}}^{d/2-\lfloor k/2\rfloor} \binom{d/2}{r}\binom{d/2-r}{d-k-2r}\left(\frac{n}{d/2}\right)^{d-k-r}.$$

For d odd and $0 < k < d$

$$\begin{aligned} &f_k(P^*(n,d)) \\ &= f_{k-1}(P^*(n-1,d-1)) + f_k(P^*(n-1,d-1)) \\ &= \sum_{r=\max\{0,\lfloor d/2\rfloor-(k-1)\}}^{\lfloor d/2\rfloor-\lfloor (k-1)/2\rfloor} \binom{\lfloor d/2\rfloor}{r}\binom{\lfloor d/2\rfloor-r}{d-k-2r}\left(\frac{n-1}{\lfloor d/2\rfloor}\right)^{d-k-r} \\ &+ \sum_{r=\max\{0,\lfloor d/2\rfloor-k\}}^{\lfloor d/2\rfloor-\lfloor k/2\rfloor} \binom{\lfloor d/2\rfloor}{r}\binom{\lfloor d/2\rfloor-r}{(d-1)-k-2r}\left(\frac{n-1}{\lfloor d/2\rfloor}\right)^{(d-1)-k-r}. \end{aligned}$$

The value of $f_0(P^*(n,d))$ follows from Theorem 5.3.1 and obviously $f_d(P^*(n,d)) = 1$.

Proof. The proof is similar to the proof of Theorem 5.3.1 and we use the same notation. Assume that d is even and $0 \leq k \leq d$. The k-faces of $P^*(n,d)$ are induced by certain intersections of $d-k$ constraints of G with $P^*(n,d)$. Let K be $d-k$ constraints from G such that the following holds. For every $1 \leq i \leq d/2$, G_i contains at most two constraints of K. If it contains two constraints h_ℓ and h_m then they are consecutive, i.e., they define a vertex in P_i^* (see also Figure 5.4). We show that the intersection of the boundaries of the constraints K with $P^*(n,d)$ is a k-face. This is the same argument as in the proof of Theorem 5.3.1. For $1 \leq i \leq d/2$ let $k_i \in \{0,1,2\}$, the number of

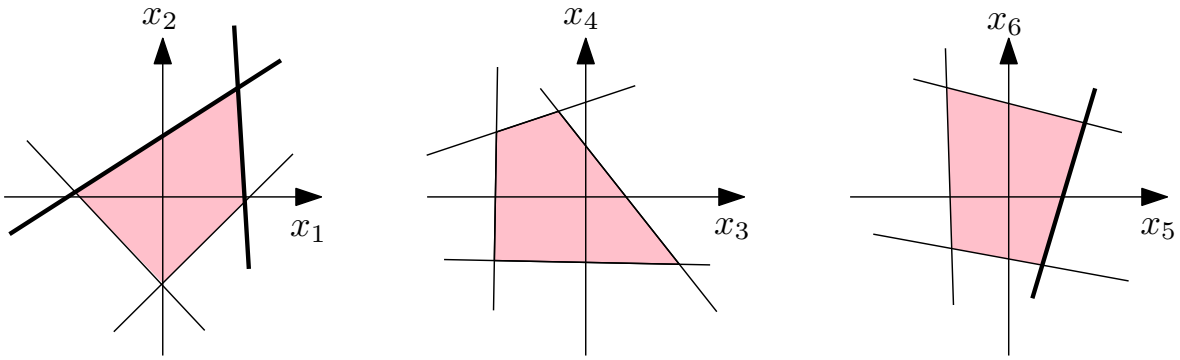

Figure 5.4: Example of 3-face in $P^*(12,6)$

constraints K contains from G_i. W.l.o.g. assume that if $k_i = 1$ then $K \cap G_i = \{g_i^1\}$ and if $k_i = 2$ then $K \cap G_i = \{g_i^1, g_i^2\}$. It is clear that $\sum_{i=1}^{d/2} k_i = d-k$.

As in the proof of Theorem 5.3.1 we add up all constraints involved and define the halfspace

$$h : \sum_{i=1}^{d/2} \sum_{\ell=1}^{k_i} a_i^\ell (x_{2i-1}, x_{2i})^T \leq \sum_{i=1}^{d/2} \sum_{\ell=1}^{k_i} b_i^\ell.$$

Denote h by $h : a'x \leq b'$ and let $F = h \cap P^*$. Since

$$P^*(n,d) = \{x \in R^d \mid (x_{2i-1}, x_{2i}) \in P_i, \text{ for all } 1 \leq i \leq d/2\},$$

it follows that

$$\dim(F) = \dim(h \cap P^*) = \sum_{i=1}^{d/2} \dim(h \cap P_i^*) = \sum_{i=1}^{d/2} (2 - k_i) = d - \sum_{i=1}^{d/2} k_i = k.$$

Moreover by definition of h for $x \in F \cap P^*$,

$$a'x = b'.$$

With the same argument as in the proof of Theorem 5.3.1 for $x \in P^* \backslash F$,

$$a'x < b'.$$

It follows that F is a k-face of P^*.

Now let K' be $d - k$ constraints of G that do not have the form as described above. Then K does not define a k-face of P^*, by the same argumentation as in the proof of Theorem 5.3.1.

It remains to count the number of faces that are induced by constraints of form K.

Let us consider the sets in K, such that there are exactly r many G_i's that contain two constraints of K. There are

$$\binom{d/2}{r} \binom{d/2 - r}{(d-k) - 2r} \left(\frac{n}{d/2}\right)^{(d-k)-r}$$

of those. Now if $(d - k) \leq d/2$, then r can be in $\{0, \ldots, \lfloor (d-k)/2 \rfloor\}$ and if $(d - k) > d/2$, then r is in $\{d/2 - k, \ldots, \lfloor (d-k)/2 \rfloor$. The claim for even d follows.

The case where d is odd is similar. We will not go into detail but only give the main idea. With the same kind of argumentation as above one can show that the k-dimensional faces are induced by $(d - k)$ constraints K of P^* in one of the following ways. In the first case K does not contain the constraint $x_d \geq 0$. Then the constraints in K must induce a $k - 1$ face in R^{d-1}, which then induces a k-face in R^d. There are $f_{k-1}(P^*(n-1, d-1))$ constraints of this form. In the second case K contains the constraint $x_d \geq 0$. Then the remaining $(d-k)-1$ constraints must induce a k face in R^{d-1}. There are $f_k(P^*(n-1, d-1))$ constraints of this form. □

Lemma 5.3.4. *The following tables show the leading terms of $P^*(n,d)$ and $c^*(n,d)$, if $d = o(n)$.*

	$P^*(n,d)$	$c^*(n,d)$
$k \leq d/2$	$\binom{d/2}{k} \cdot \left(\frac{n}{d/2}\right)^{d/2}$	$\binom{d/2}{k} \cdot \binom{n-d/2-1}{d/2}$
$k > d/2$	$\binom{d/2}{d-k} \cdot \left(\frac{n}{d/2}\right)^{d-k}$	$\binom{n-k-1}{d-k}$

d even

	$P^*(n,d)$	$c^*(n,d)$
$k \leq \lfloor d/2 \rfloor$	$\left(\binom{\lfloor d/2 \rfloor}{k} + \binom{\lfloor d/2 \rfloor}{k+1}\right) \cdot \left(\frac{n-1}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}$	$\binom{\lceil d/2 \rceil}{k} \cdot \binom{n-\lceil d/2 \rceil-1}{\lfloor d/2 \rfloor}$
$k \geq \lceil d/2 \rceil$	$\binom{\lfloor d/2 \rfloor}{d-k} \cdot \left(\frac{n-1}{\lfloor d/2 \rfloor}\right)^{d-k}$	$\binom{n-k-1}{d-k}$

d odd

The proof of the lemma follows by checking the formulas of $P^*(n,d)$ and $c^*(n,d)$.

Theorem 5.3.5. *For $k < \lceil d/2 \rceil$*

$$f_k(c^*(n,d)) \leq \mathit{poly}(d) \cdot e^{\lfloor d/2 \rfloor} \cdot f_k(P^*(n,d)),$$

and for $k \geq \lceil d/2 \rceil$

$$f_k(c^*(n,d)) \leq \mathit{poly}(d) \cdot e^{d-k} \cdot f_k(P^*(n,d)).$$

Proof. Let us consider the case $k \leq \lceil d/2 \rceil - 1$ first. We will prove the statement for odd d, the case where d is even follows immediately by replacing all $\lfloor d/2 \rfloor$ and $\lceil d/2 \rceil$ by $d/2$. First note that the leading term of $P^*(n,d)$ can be written as

$$\left(\binom{\lfloor d/2 \rfloor}{k} + \binom{\lfloor d/2 \rfloor}{k+1}\right) \cdot \left(\frac{n-1}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}$$
$$= \mathrm{poly}(d) \cdot \binom{\lceil d/2 \rceil}{k} \cdot \left(\frac{n-1}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}.$$

For the term of $c^*(n,d)$ we have

$$\begin{aligned}
&\binom{\lceil d/2\rceil}{k}\cdot\binom{n-\lceil d/2\rceil-1}{\lfloor d/2\rfloor}\\
&\quad\overset{(5.3)}{\leq}\ \mathrm{poly}(d)\cdot e^{\lfloor d/2\rfloor}\cdot\binom{\lceil d/2\rceil}{k}\cdot\left(\frac{n-\lceil d/2\rceil-1}{\lfloor d/2\rfloor}\right)^{\lfloor d/2\rfloor}\\
&\quad\leq \mathrm{poly}(d)\cdot e^{\lfloor d/2\rfloor}\cdot\binom{\lceil d/2\rceil}{k}\cdot\left(\frac{n-1}{\lfloor d/2\rfloor}\right)^{\lfloor d/2\rfloor}.
\end{aligned}$$

Therefore if $k\leq\lceil d/2\rceil-1$,

$$f_k(c^*(n,d))\leq \mathrm{poly}(d)\cdot e^{\lfloor d/2\rfloor}\cdot f_k(P^*(n,d)).$$

Now let us consider the case where $k\geq\lceil d/2\rceil$. We assume that d is even, the odd case follows by replacing $d/2$ with $\lfloor d/2\rfloor$. For $P^*(n,d)$ the leading term can be written as

$$\binom{d/2}{d-k}\cdot\left(\frac{n}{d/2}\right)^{d-k}\overset{(5.1)}{=}\mathrm{poly(d)}\frac{(d/2)^{d/2}}{(d-k)^{d-k}\cdot(k-d/2)^{k-d/2}}\left(\frac{n}{d/2}\right)^{d-k}$$

Similarly as in the last case for $c^*(n,d)$

$$\binom{n-k-1}{d-k}\leq \mathrm{poly}(d)\cdot e^{d-k}\cdot\left(\frac{n}{d-k}\right)^{d-k}$$

Hence

$$\begin{aligned}
\frac{\binom{n-k-1}{d-k}}{\binom{d/2}{d-k}\cdot\left(\frac{n}{d/2}\right)^{d-k}}&\leq \mathrm{poly}(d)\cdot e^{d-k}\cdot\underbrace{\left(\frac{k-d/2}{d/2}\right)^{k-d/2}}_{\leq 1}\\
&=\mathrm{poly}(d)\cdot e^{d-k}.
\end{aligned}$$

It follows that if $k\geq\lceil d/2\rceil$ and d even

$$f_k(c^*(n,d))\leq \mathrm{poly}(d)\cdot e^{d-k}\cdot f_k(P^*(n,d)).$$

□

5.4 Upper Bound on Maximum Complexity of $LI(2)$

In this section we show that no polytope in $LI(2)$ can achieve the complexity of the dual cyclic polytope. To our knowledge, this is the first time such bounds are given. In Lemma 5.4.1 we show that for all polytopes P in $LI(2)$, $d \geq 4$ and $\lceil d/2 \rceil \leq k \leq d-2$ it holds that $f_k(P) < f_k(c^*(n,d))$. Using this result in Theorem 5.4.2, we show that this holds for all $k \leq d-2$.

Lemma 5.4.1. *Let P be any polytope in $LI(2)$ given by n nonredundant constraints, $d \geq 4$ and denote by n' the number of constraints that contain exactly two variables per inequality. As for each index $i \in [d]$ there are at most two inequalities that contain only x_i it follows that $n - 2d \leq n' \leq n$. Then for all $\lceil d/2 \rceil \leq k \leq d-2$ we have*

$$f_k(P) < f_k(c^*(n,d)).$$

In particular

$$f_{d-2}(P) \leq \binom{n}{2} - \frac{\binom{n'}{2}}{\binom{d}{2}} + n' < \binom{n}{2} = f_{d-2}(c^*(n,d).)$$

Proof. Let us focus on the case of $f_{d-2}(P)$. In the dual cyclic polytope we know that any two facets are adjacent, i.e., their intersection defines a $(d-2)$-face. In $LI(2)$ however, not every two facets can be adjacent. Assume P is given by n constraints with index set E. For $i < j \in [d]$ let E_{ij} be the indices of the constraints that contain x_i and x_j and denote $|E_{ij}| = n_{ij}$. As in the proof of Theorem 5.3.3 we know that out of the $\binom{n_{ij}}{2}$ pairs only n_{ij} pairs are adjacent. Summing over all $i < j$ it follows that at least

$$\sum_{i<j} \left(\binom{n_{ij}}{2} - n_{ij} \right)$$

pairs of facets in P are not adjacent. Now using that $\sum_{i<j} n_{ij} = n'$ and that the sum is minimized if all n_{ij} have the same size $n'/\binom{d}{2}$, we

get that

$$\sum_{i<j}\left(\binom{n_{ij}}{2}-n_{ij}\right) \geq \binom{d}{2}\cdot\binom{\frac{n'}{\binom{d}{2}}}{2}-n'$$
$$= \frac{1}{\binom{d}{2}}\cdot\frac{n'\cdot\left(n'-\frac{1}{\binom{d}{2}}\right)}{2}-n'$$
$$\geq \frac{\binom{n'}{2}}{\binom{d}{2}}-n'.$$

The claim for $k = d-2$ follows. For other values of k one can similarly show that not all $(d-k)$-tuples of constraints define a k-face in P. □

Theorem 5.4.2. *Let P be any d-dimensional polytope in $LI(2)$ given by n nonredundant constraints, where $d \geq 4$. Then for all $k \leq d-2$ we have*

$$f_k(P) < f_k(c^*(n,d)).$$

Although asymptotically the bounds that we prove are the same as the bounds of the dual cyclic polytope, this shows that the dual cyclic polytope is not realizable in $LI(2)$.

Before proving this theorem we introduce a few notions used in the proof of McMullen's Upper Bound Theorem (for more details see [36, 18]). From now on we only consider simple d-dimensional polytopes given by n nonredundant constraints. A polytope P is called *simple*, if every vertex of P is satisfies exactly d inequalities with equality. We observe that by small perturbations, for any d-dimensional P' in $LI(2)$ given by n inequalities there exists a simple polytope P in $LI(2)$ with $f_k(P') \leq f_k(P)$ for all $k \in [d]$. Let us denote the family of simple d-dimensional polytopes in $LI(2)$ by $SLI(2)$.

Let P be any polytope in $SLI(2)$, given by n nonredundant constraints. We consider a linear program with objective value $c^T x$, subject to those constraints. We assume that c is generic, i.e., no edge of P is parallel to the hyperplane given by $c^T x = 0$. We now orient every edge of P w.r.t.

$c^T x$, towards the vertex with higher objective value. Let us denote the graph defined by those directed edges by $\overrightarrow{G}(P)$. Now for $i = 0, \ldots, d$ we denote by $h_i(\overrightarrow{G}(P))$ the number of vertices with indegree i.

We now show that $h_i(\overrightarrow{G}(P))$ is independent of the objective value, hence we can write $h_i(\overrightarrow{G}(P)) = h_i(P)$. Let k be fixed, we count the pairs (F, v) of k faces F with unique sink v. By definition of $\overrightarrow{G}(P)$ every face has a unique sink, hence there are exactly $f_k(P)$ many such pairs.

On the other hand by properties of simple polytopes it holds that for any k distinct edges to v, there exists a unique k-face containing the k edges. Let v be fixed and let r be the indegree of v. Summing over all indegrees $r \geq k$ it follows that for all $k = 0, \ldots, d$,

$$\sum_{r=k}^{d} h_r(\overrightarrow{G}(P)) \binom{r}{k} = f_k(P). \tag{5.4}$$

Solving this system of linear equalities one can show that for all $i = 0, \ldots, d$,

$$h_i(P) := h_i(\overrightarrow{G}(P)) = \sum_{k=i}^{d} (-1)^{k-i} \binom{k}{i} f_k(P). \tag{5.5}$$

Hence $h_i(P)$ is independent of the objective value.

To prove Theorem 5.4.2 we use the following strengthened version of McMullen's theorem, which holds for any simple polytopes. This strengthening was first given by Kalai in [30] with a small correction made by Fukuda in [18, Chapter 7]. Note that Theorem 5.4.3 implies McMullen's theorem (5.2.1), since by (5.4) we know that each $f_k(P)$ is a nonnegative linear combination of the $h_r(P)$'s.

Theorem 5.4.3 (Strengthened Upper Bound Theorem [36, 18])**.** *Let P be a simple polytope given by n nonredundant constraints. Then for all $i = 0, \ldots, d$ it holds that*

$$h_i(P) \leq h_i(c^*(n, d)).$$

Proof. Let P be a simple polytope given by n nonredundant constraints. We start by making two observations. First, for every $i < d$ and F a facet of P, it holds that

$$h_i(F) \leq h_{i+1}(P). \tag{5.6}$$

This is true if we choose the generic objective function c such that the objective value is higher in F than in the rest of the polytope. The inequality follows since the h-vector is independent of the choice of c.

Secondly for all $i < d$ it holds that

$$\sum_F h_i(F) = (i+1)h_{i+1}(P) + (d-i)h_i(P). \tag{5.7}$$

The left side of the equation sums over all facets of P. Notice that a facet with indegree i has indegree i or $i+1$ in P. Consider a vertex with indegree i in P. Then there are $(d-i)$ facets that contain this vertex and have indegree i. For a vertex with indegree $i+1$ in P there are exactly $(i+1)$ facets containing it that have indegree i.

For the proof of the theorem we use induction on i with decreasing values. The claim is trivial for $i = d$ and $i = 0$. Since the h vector is symmetric it is enough to show that for all $i = \lceil d/2 \rceil, \ldots, d$,

$$h_i(P) \leq h_i(c^*(n,d)) = \binom{n-i-1}{d-i}. \tag{5.8}$$

The last equality follows by substituting the values of the f-vector of $c^*(n,d)$ into (5.5).

Assume that the statement holds for $i = k+1 \leq d$. Then for $i = k$, by (5.6) and (5.7) we get that

$$(k+1)h_{k+1}(P) + (d-k)h_k(P) = \sum_F h_k(P) \leq nh_{k+1}(P).$$

Using the induction hypothesis it follows that

$$\begin{aligned} h_k(P) &\leq \frac{n-k-1}{d-k} h_{k+1}(P) \\ &\leq \frac{n-k-1}{d-k} \binom{n-k-2}{d-k-1} \\ &= \binom{n-k-1}{d-k} = h_k(c^*(n,d)). \end{aligned}$$

□

Proof of Theorem 5.4.2. Let P be any polytope in $SLI(2)$. By Lemma 5.4.1 the theorem holds for $\lceil d/2 \rceil \leq k \leq d-2$ (since $d \geq 4$ it holds in particular for $k = d-2$). We claim that

$$h_{d-2}(P) < h_{d-2}(c^*(n,d)).$$

The theorem then follows for all $k < d-2$ from equation (5.4) and Theorem 5.4.3. It hence remains to show the claim. By equation (5.5)

$$h_{d-2}(P) = f_{d-2}(P) - (d-1) f_{d-1}(P) + \binom{d}{d-2} f_d(P).$$

We know that

$$f_{d-1}(P) = f_{d-1}(c^*(n,d)) = n \text{ and } f_d(P) = f_d(c^*(n,d)) = 1.$$

Furthermore by Lemma 5.4.1 we know

$$f_{d-2}(P) < f_{d-2}(c^*(n,d)).$$

It follows that

$$\begin{aligned} h_{d-2}(P) &= f_{d-2}(P) - (d-1) f_{d-1}(P) + \binom{d}{d-2} f_d(P) \\ &< f_{d-2}(c^*(n,d)) - (d-1) f_{d-1}(c^*(n,d)) + \binom{d}{d-2} f_d(c^*(n,d)) \\ &= h_{d-2}(c^*(n,d)), \end{aligned}$$

which shows the claim. □

Remark 5.4.4. One can show that for $d = 3$, the bounds of McMullen's Upper Bound Theorem can be achieved in $LI(2)$. Let P be a polytope given by $n-2$ constraints in variables x_1 and x_2, such that they define a polygon with $n-2$ vertices in two dimensions. We furthermore add the constraints $x_3 \geq 0$ and $x_3 \leq 1$. We can easily observe that $f_0 = 2n - 4$ and $f_1 = 3n - 6$. Those are exactly the bounds achieved by the dual cyclic polytope.

5.5 Discussion and Open Questions

We saw in this chapter that $f_k(P^*(n,d))$ differs from $f_k(c^*(n,d))$ by a factor $O(e^{\lfloor d/2 \rfloor})$ if $k < \lceil d/2 \rceil$ and $O(e^{d-k})$ otherwise. In particular, if d is constant, then $f_k(P^*(n,d))$ is of the same order as $f_k(c^*(n,d))$. The high complexity of $P^*(n,d)$ shows us that although $LI(2)$ has a much simpler structure than general linear programs, it is still a powerful and complex tool. We also showed that the dual cyclic polytope is not realizable in $LI(2)$. However in the upper bound we showed, the asymptotic complexity remains the same. It would be interesting to get a deeper understanding of $LI(2)$ and how it is different from general linear programs.

In particular for this chapter, the main open question that remains is how large the complexity of $f(P)$ can be for a polytope P in $LI(2)$. Is it possible to have higher complexity than the complexity of $P^*(n,d)$? If yes, what is the maximum complexity that can be achieved? Is it asymptotically the same as the complexity of the a dual cyclic polytope? This is an interesting direction for future research.

Part II

Sampling with Removal in Generalizations of Linear Programming

Chapter 6

Sampling with Removal

This chapter is based on [25] by B. Gärtner, J. Lengler and M. Szedlák.

6.1 Introduction

On a high level, random sampling can be described as an efficient way of learning something about a problem, by first solving a subproblem of much smaller size. A classical example is the problem of finding the smallest element in a *sorted compact list* [12, Problem 11-3]. Such a list stores its elements in an array, but in arbitrary order. Additional pointers are used to link each element to the next smaller one in the list. Given a sorted compact list of size n, the smallest element can be found in expected time time $O(\sqrt{n})$ as follows: sample a set of $\lfloor\sqrt{n}\rfloor$ array elements at random. Starting from their minimum, follow the predecessor pointers to the global minimum. The key fact is that the expected number of pointers to be followed is bounded by $\sqrt{n}$, and this yields the expected runtime.

In the setting of linear programs one can show that sampling $O(\sqrt{n})$ constraint at random and optimizing w.r.t. said constraints, we only expect $O(d\sqrt{n})$ constraints to be not satisfied. Hence by solving a problem of sublinear size, we get a solution of the LP that only violates a sublinear number of constraints. Clarkson's linear program solver (to solve an LP exactly), which is linear in n but exponential in d, relies on this result [11].

On an abstract level, the situation can be modeled as follows. Let H

be a set of size n that we can think of as the set of constraints in an optimization problem, for example the elements in a sorted compact list. Let $V : 2^H \to 2^H$ be a function that assigns to each subset $R \subseteq H$ of constraints a set $V(R) \subseteq H \setminus R$. We can think of $V(R)$ as the set of constraints violated by the optimal solution subject to only the constraints in R. In this abstract setting, (H, V) is called a *consistent space (without minus infinity)*, a notion introduced here for the first time (for a formal definition see Definition 6.2.3 below.) In the sorted compact list example, $V(R)$ is the set of elements that are smaller than the minimum of R. In the setting of linear programs $V(R)$ are the constraints that change the optimal solution if added to R.

In this example, the above "key fact" is a concrete answer to the following general question: Suppose that we sample a set $R \subseteq H$ of size $r \leq n$ uniformly at random. What can we say about the quantity v_r, the expected size of $V(R)$? What are conditions on V under which v_r is small?

The main workhorse in this context is the *Sampling Lemma* [9, 28]. It states that $v_r = \frac{n-r}{r+1} \cdot x_{r+1}$, where x_r is the expected size of $X(R) = \{h \in R : h \in V(R \setminus \{h\})\}$. In other words, $h \in X(R)$ is a constraint that is not automatically satisfied if the problem is solved without enforcing it. In the sorted compact list example, every nonempty set R has one such "extreme" constraint, namely its minimum. Consequently, we have $x_{r+1} = 1$, and hence $v_r = (n - r)/(r + 1)$. With $r = \lfloor \sqrt{n} \rfloor$, $v_r < \sqrt{n}$ follows. In the setting of linear programs, the optimal solution is given by at most d constraints. These are the only ones that can be extreme, hence the result for linear programs follows.

The Sampling Lemma has many other applications in computational geometry when x_{r+1} can be bounded; in a number of relevant cases, we do not only know the expected value v_r but the complete probability distribution $p_\ell = \Pr[|V(R)| = \ell], \ell \leq n$, and tail estimates for $|V(R)|$ [28].

In this chapter, we address the following more general question in the abstract setting: Suppose that we sample a set $R \subseteq H$ of size $r \leq n$ uniformly at random, but then we remove a subset $K_R \subseteq R$ of a fixed

size k, according to an arbitrary rule. What can we still say about the expected size of $V(R \setminus K_R)$? If K_R is a random subset of R, the expectation is v_{r-k}, but if K_R is chosen by another (deterministic) rule, then $R \setminus K_R$ is no longer a uniformly random subset, and the Sampling Lemma does not apply.

Our work is originally motivated by *chance-constrained optimization*. In this setting, we have a probability distribution over a (possibly infinite) set of constraints. The goal is to compute a *sufficiently feasible* solution, one that satisfies a randomly chosen constraint with high probability. Such a solution can be obtained by optimizing over a finite sample of constraints drawn from the distribution [5]. Here, a-posteriori removal of constraints is shown to yield a tradeoff between solution quality and violation probability [6]. We are trying to understand the combinatorial essence of this tradeoff.

Intuitively, one would think that if k is constant, the change in the expected number of violated constraints under the removal of k constraints is small. This intuition was proved to be correct if the pair (H, V) is induced by a *nondegenerate LP-type problem* of fixed dimension δ [23] (for the definition of dimension see Definition 6.2.5 below). LP-type problems have been introduced and analyzed by Matoušek, Sharir and Welzl as a combinatorial framework that encompasses linear programming and other geometric optimization problems [41, 34]. The quantitative result is that under removal of k elements, the expected number of violated constraints increases by a factor of δ^k at most, which is constant if both δ and k are constant. It was left open whether this factor can be improved for interesting sample sizes (for very specific and rather irrelevant values of δ, r, k, it was shown to be best possible).

In this chapter, we improve over the results in [23] in several respects. In Section 6.3, Theorem 6.3.4 we show that the increase factor δ^k can be replaced by $\ln r + k$, which is a vast improvement for a large range of values of k. Moreover, the new bound neither requires the machinery of LP-type problems, nor nondegeneracy. It holds in the completely abstract setting of consistent spaces considered above. In this setting, we can also show that the bound is best possible for all sample sizes

of the form $r = n^\alpha, 0 < \alpha < 1$ (see Section 6.5). We also show that this bound is best possible for *violator spaces*, in the case where $k = \Omega(\delta \ln r)$. In general, the gap to the lower bound is $\ln r$.

Hence, if anything can be gained over the new bound, additional properties of the violator function V have to be used. Indeed, for nondegenerate LP-type problems and small values of k, the increase factor in [23] is better than our new bound, and most notably, it does not depend on the problem size n. We show in Section 6.4, Theorem 6.4.7 that the same factor can be derived under the much weaker conditions of a *violator space*, and with a much simpler proof, based on a "removal version" of the Sampling Lemma. Furthermore the proof of [23] is given for a specific rule to remove k, whereas our proof works for any rule.

Intuitively, violator spaces are LP-type problems without objective function, and they were introduced to show that many combinatorial properties of LP-type problems and algorithms for LP-type problems do not require the objective function at all [26, 4].

In Section 6.7, we give a complete characterization of violator spaces of dimension 1. For general δ and small (in particular constant) k, the quest for the best bound on the increase factor remains open. In particular, it is not clear whether the exponential growth in k actually happens.

6.2 Basics and Definitions

Throughout we will work with three combinatorial concepts, the LP-type problem, the violator space and the consistent space. The LP-type problem was first introduced by Sharir and Welzl [41], the generalized concept of violator spaces by Gärtner, Matoušek, Rüst, and Škovroň [26], based on the doctoral thesis of Škovroň [46]. We here introduce an even more general concept of consistent spaces.

6.2.1 LP-type Problems

Definition 6.2.1. An *LP-type* problem is a triple $\mathcal{P} = (H, \Omega, \omega)$ that satisfies the following. H is a finite set (the constraints), Ω a totally ordered set with a smallest element $-\infty$ and $\omega : 2^H \to \Omega$ a function that assigns an objective function value to $G \subseteq H$. For all $F \subseteq G \subseteq H$ and $h \in H$, the following hold.

1. $\omega(F) \leq \omega(G)$.
2. If $\omega(F) = \omega(G) > -\infty$, then $\omega(G \cup \{h\}) > \omega(G) \Rightarrow \omega(F \cup \{h\}) > \omega(F)$.

Observe that using the second property, called *locality*, by simple induction one can show that if $\omega(F \cup \{h\}) = \omega(F) > -\infty$ for all $h \in G \backslash F$, then $\omega(F) = \omega(G)$.

The classic example of an LP-type problem is the problem of computing the smallest enclosing ball of a finite set of points P in $\mathbb{R}^d$. Let us denote this problem by SEB. We can write this as an LP-type problem by setting $H = P$ and $\Omega = \mathbb{R} \cup \{-\infty\}$. For $G \subseteq H$, $\omega(G)$ is defined as the radius of the smallest enclosing ball of G, with the convention that the smallest enclosing ball of the empty set has radius $-\infty$. Since the smallest enclosing ball of a nonempty set of points exists and is unique [48], ω is well defined.

The first property is clear by definition. To see locality, observe that for $F \subseteq G$, $\omega(F) = \omega(G)$ means that both F and G have the same smallest enclosing ball. If $\omega(G \cup \{h\}) > \omega(G)$, then h is outside this ball, so $\omega(F \cup \{h\}) > \omega(F)$.

Another example for an LP-type problem is, as the name already suggests, linear programs. Let us consider a feasible linear program of form

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & Ax & \leq \quad b, \end{array}$$

where $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$ and $c \in \mathbb{R}^d$. We assume that c is a generic direction, in particular if there exists an optimal solution w.r.t. some

subsystem of $Ax \leq b$, then this solution is unique. We set H to be the set of constraints of this LP, i.e., the rows of $Ax \leq b$, and $\Omega = \mathbb{R} \cup \{-\infty\}$. For $G \subseteq H$ let $A_G x \leq b_G$, be the constraints that are indexed by G. Then $\omega(G)$ is defined as the optimal objective value of the LP w.r.t. $A_G x \leq b_G$. A constraint $h \in H$ is a violator of G, if the minimum objective value w.r.t. $A_{G\cup\{h\}} x \leq b_{G\cup\{h\}}$ is larger than w.r.t. $A_G x \leq b_G$. It is again clear that the first property holds. For locality, assume that for $F \subseteq G$, $\omega(F) = \omega(G) = c > -\infty$. Then this value c is defined by a unique set of d constraints $B \subseteq F \subseteq G$, i.e., $\omega(B) = \omega(F) = \omega(G)$. Now if $\omega(G \cup \{h\}) > \omega(G)$, then $\omega(B \cup \{h\}) > \omega(B)$ and hence $\omega(F \cup \{h\}) \geq \omega(B \cup \{h\}) > \omega(B) = \omega(F)$.

Definition 6.2.2. A constraint $h \in H \setminus G$ is violated by G if $\omega(G \cup \{h\}) > \omega(G)$. We denote the set of violated constraints by $V(G)$.

For SEB, the violated constraints of G are exactly the points lying outside the smallest enclosing ball of G. For linear programming, the violators are the constraints, whose addition changes the value of the optimal solution.

6.2.2 Violator Spaces

Intuitively a violator space is an LP-type problem without an objective function. The rationale behind this concept is that many things one can prove about LP-type problems do not require the concept of order.

Definition 6.2.3. A *violator space without minus infinity* is a pair (H, V), $|H| = n$ finite and V a function $2^H \to 2^H$ such that the following is satisfied for all $F \subseteq G \subseteq H$.

1. $G \cap V(G) = \emptyset$ (**consistency**).
2. If $G \cap V(F) = \emptyset$, then $V(G) = V(F)$ (**locality**).

Observe that the locality condition implies that if $E \subseteq F \subseteq G$ and $V(E) = V(G)$, then $V(E) = V(F) = V(G)$.

We note that this is a generalization of the SEB problem as (H, Ω, ω) can naturally be converted into a violator space through $V(G) = \{x \in H \setminus G \mid \omega(G \cup \{x\}) \neq \omega(G)\}$, for $G \subseteq H$. However this does not hold for all LP-type problems, as locality does not necessarily hold for sets G with $\omega(G) = -\infty$ [46]. In particular it does not always hold in linear programing.

The following definition of violator space with minus infinity is a generalization of all LP-type problems [46]. The two notions are not equivalent, not every violator space can be converted into an LP-type problem. Any unique sink orientation (USO) [43] of a cube or the grid [27] corresponds to a violator space, but not to an LP-type problem in general [26].

Definition 6.2.4. A *violator space with minus infinity*, or short a *violator space*, is a triple $(H, V, \mathcal{U})$, $|H| = n$ finite, V a function $2^H \to 2^H$, $\mathcal{U} \subseteq 2^H$, such that the following is satisfied.

1. For every $G \subseteq H$, $G \cap V(G) = \emptyset$ (**consistency**).
2. For every $F \subseteq G \subseteq H$ with $F \notin \mathcal{U}$ and $G \cap V(F) = \emptyset$, we have $V(G) = V(F)$ (**locality**).
3. For every $F \subseteq G \subseteq H$ with $G \in \mathcal{U}$, we have $F \in \mathcal{U}$ (**monotonicity**).
4. For every $G \in \mathcal{U}$, we have $V(G) = \{h \in H \mid G \cup \{h\} \notin \mathcal{U}\}$ (**matching of V and $\mathcal{U}$**).

We call a set $G \subseteq H$ *unbounded* if $G \in \mathcal{U}$, and otherwise *bounded*.

It is not hard to see that linear programming satisfies these conditions.

Definition 6.2.5. Let $(H, V, \mathcal{U})$ be a violator space.

1. $B \subseteq H$, $B \neq \emptyset$ is called a *basis* in $(H, V, \mathcal{U})$, if $B \notin \mathcal{U}$ and for all $F \subsetneq B$, $F \notin \mathcal{U}$, it holds that $B \cap V(F) \neq \emptyset$ (or equivalently, $V(F) \neq V(B)$).

   The empty set is always a basis.

2. For $G \subseteq H$, $G \notin \mathcal{U}$ a basis of G is an inclusion-minimal subset $B \subseteq G$, $B \notin \mathcal{U}$, such that $V(B) = V(G)$. For $G \in \mathcal{U}$ the empty set is the unique basis of G. (In particular, a basis of G is a basis in $(H, V, \mathcal{U})$, and every basis in $(H, V, \mathcal{U})$ is a basis of itself.)

3. The *(combinatorial) dimension* of $(H, V, \mathcal{U})$, $\delta := \delta(H, V, \mathcal{U})$, is defined as the size of the largest basis in $(H, V, \mathcal{U})$.

Again, let us illustrate this on the SEB problem. A basis of G is a minimal subset of points with the same smallest enclosing ball as G. In particular all points of the basis are on the ball's boundary. In d-dimensional space, the combinatorial dimension of any SEB-instance is at most $d + 1$ [15]. However, a basis can be smaller than the combinatorial dimension, and a point set can have more than one basis: in $\mathbb{R}^2$ the set of four corners of a square has two bases, the two pairs of diagonally opposite points. For a linear program, let $G \subseteq H$, with $G \notin \mathcal{U}$. Then the optimal objective value is bounded and hence defined by a set of at most d (number of variables) constraints. The dimension of a linear program hence coincides with the usual definition of the dimension of an LP.

Remark 6.2.6. In the following we will always assume for violator spaces, that the empty set is unbounded. Consider the case of a violator space (H, V), where the empty set is bounded, hence by monotonicity $\mathcal{U} = \emptyset$. We set $V'(\emptyset) = H$ and $V'(R) = V(R)$ for all other $R \subseteq H$. Then $(H, V', \{\emptyset\})$ is a violator space. If in (H, V), the empty set is a basis for some $R \subseteq H$, i.e., $V(R) = V(\emptyset)$, then by locality it follows that $V(R) = V(x)$ for any $x \in R$. Hence x is a basis of R in $(H, V', \{\emptyset\})$. The dimension hence does not change, unless it was 0 (which is not an interesting case, since $V(R) = \emptyset$ for all $R \subseteq H$). The violator space $(H, V', \{\emptyset\})$ is therefore almost the same as (H, V), they only differ in the violators of the empty set and the bases of the sets that had the empty set as their basis.

Note that for the examples of the LP-type problems, such as SEB and linear programming it holds that $\emptyset \in \mathcal{U}$.

Definition 6.2.7. Let $(H, V, \mathcal{U})$ be a violator space, $G \subseteq H$. The set

of *extreme constraints* $X(G) \subseteq G$ is defined by

$$r \in X(G) \Leftrightarrow r \in V(G \setminus \{r\}).$$

In the SEB case, h is extreme in G, if its removal allows for a smaller enclosing ball. Therefore h is necessarily on the boundary of the smallest enclosing ball, but this is not sufficient. For the case $\mathbb{R}^2$, if G consists of the four corners of a square, then G has no extreme point.

It is not hard to see that $X(G)$ is the intersection of all bases of G, hence $|X(G)| \leq \delta$ (see also Lemma 6.4.1). To bound the expected number of violators, the following result from [28] can be used.

Lemma 6.2.8. *[Sampling Lemma] Let $(H, V, \mathcal{U})$ be a violator space with combinatorial dimension δ. Let $R \subseteq H$ a set of size r, chosen uniformly at random, $v_r = \mathbb{E}[|V(R)|]$ and $x_r = \mathbb{E}[|X(R)|]$. Then*

$$v_r = \frac{n-r}{r+1} \cdot x_{r+1} \leq \frac{n-r}{r+1} \cdot \delta.$$

We will show a generalized proof of this lemma in the proof of Lemma 6.4.6.

In the SEB case in $\mathbb{R}^d$, the combinatorial dimension is $d+1$, hence $v_r \leq \frac{n-r}{r+1} \cdot (d+1)$. If $d = 2$, then the smallest enclosing ball of a random sample of size $\sqrt{n}$ has in expectation at most $3\sqrt{n}$ points outside.

Figure 6.2.2 shows an instance of SEB, on the 2×3 regular grid, in particular $n = 6$. We fix $r = 3$ and therefore look at the violators of sets of size three and extreme points of sets of size four, which are all depicted (up to symmetry). It is not hard to see that there are only eight sets of size three with no violators (corresponding to the first two in the figure) and all others have two. This implies $v_3 = 1.2$. Looking at the extreme points (Figure 6.2), the sets $\{1,3,4,6\}, \{1,2,4,5\}$ and $\{2,3,5,6\}$ have no extreme elements and all other sets of size four have exactly two extreme elements. Therefore $v_3 = \frac{6-3}{3+1} x_4 = \frac{3}{4} \cdot 1.6 = 1.2$. For six points in general position every set of four points has either two or three extreme points hence $v_3 \in [\frac{3}{4} \cdot 2, \frac{3}{4} \cdot 3] = [1.5, 2.25]$.

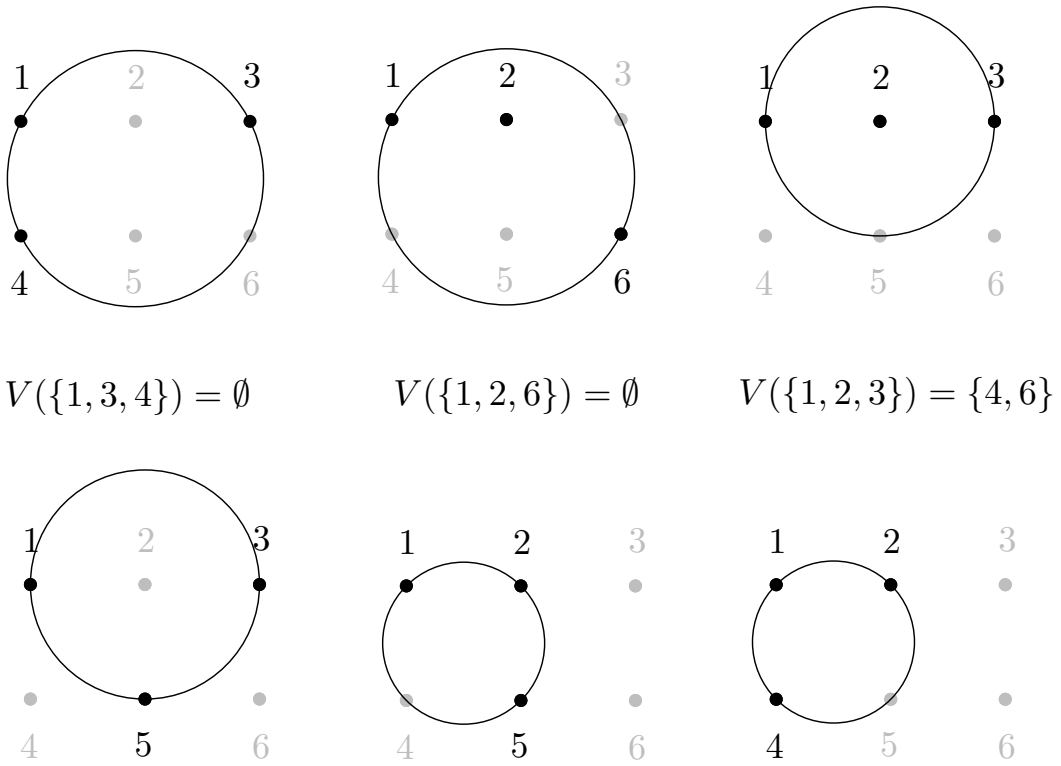

Figure 6.1: 2×3-grid violators

Definition 6.2.9. A violator space $(H, V, \mathcal{U})$ is called *nondegenerate* if every set $G \subseteq H$ has only one basis.

Note that the violator spaces resulting from SEB instances may be degenerate, since in $\mathbb{R}^2$, four points on a square have two bases. But for example, every *d-smallest number* violator space (Definition 6.5.2 below) is nondegenerate.

In geometric settings such as SEB, one can usually get rid of degeneracies by perturbations. It is shown in [35] that this does not work in abstract settings. This is shown by constructing LP-type problems of arbitrarily large dimension δ such that one has to increase the dimension to at least 2δ in oder to "remove degeneracies" (a notion that can suitably be defined in the abstract setting).

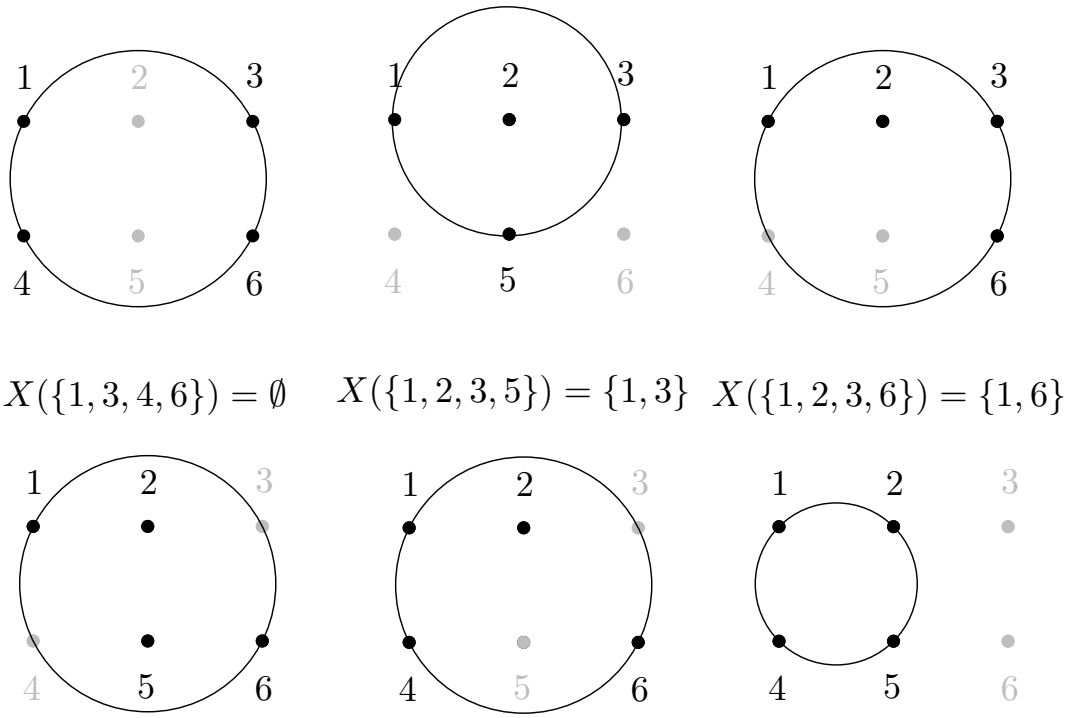

Figure 6.2: 2×3-grid extreme points

6.2.3 Consistent Spaces

We now move on to the even more general concept of consistent spaces.

Definition 6.2.10. A *consistent space (with minus infinity)* is a triple $(H, V, \mathcal{U})$, $|H| = n$ finite and V a function $2^H \to 2^H$ such that the following is satisfied for all $G \subseteq H$.

1. $G \cap V(G) = \emptyset$ (**consistency**).
2. For every $F \subseteq G \subseteq H$ with $G \in \mathcal{U}$, we have $F \in \mathcal{U}$ (**monotonicity**).
3. For every $G \in \mathcal{U}$, we have $V(G) = \{h \in H \mid G \cup \{h\} \notin \mathcal{U}\}$ (**matching of V and $\mathcal{U}$**).

Hence a consistent space is a violator space without the locality condition. If $\mathcal{U} = \emptyset$, we call it a consistent space without minus infinity.

In that case consistency is the only property that needs to hold. The basis, combinatorial dimension and extreme constraints of a consistent space can be defined equivalently as in the violator space.

Remark 6.2.11.

1. In consistent spaces we can not always assume that the empty set is unbounded, as we do in the violator spaces. Let (H, V) be a consistent space with $\mathcal{U} = \emptyset$ of dimension δ defined as follows. For the empty set we define $V(\emptyset) = \emptyset$. For all other sets B of size at most δ we define $V(B) = H \setminus B$. For all other $R \subseteq H$ we define $V(R) = \emptyset$. Now if we set $V'(\emptyset) = H$ as before, the dimension of the consistent space increases: sets of size larger than δ, do not have a basis of size at most δ.

2. In consistent spaces, the first equality $v_r = \frac{n-r}{r+1} \cdot x_{r+1}$ of the Sampling Lemma 6.2.8 still holds. However, in general it does not hold that $|X(R)| \leq \delta$ for all $R \subseteq H$. We give an example of a consistent space of dimension 1, with $\mathcal{U} = \emptyset$, such that for some $R \subseteq H$ each element is extreme. Let $R = \{1, 2, \ldots, 2m\} \subseteq [n]$ be of even size. For $i \in [m]$ let $V(\{i\}) = \{i + m\}$ and $V(\{i + m\}) = \{i\}$. For every $x \in R$ define $V(R \backslash \{x\}) = \{x\}$ and for all other sets define the violators as the empty set. Then this is a consistent space. By definition it also follows that every element in R is extreme. For $x \in R$ it holds that $V(R \setminus \{x\}) = \{x\} = V(\{x + m \mod 2m\})$ and since $x + m \mod 2m \in R$ it follows that the combinatorial dimension is 1.

6.2.4 Sampling with Removal

As already introduced in [23] for LP-type problems, we are interested in sampling with removal. We define the concept here for the most general case of consistent spaces. All results will then also hold for violator spaces and LP-type problems. Suppose we sample uniformly at random $R \subseteq H$ of size r. By some fixed rule P_k, we remove $k < r$ elements of R and obtain a set R_{P_k} of size $r - k$. We are interested in $\mathbb{E}[|V(R_{P_k})|]$, for which we will give different bounds in the next two chapters.

Before proceeding to the bounds, we discuss some possible rules for the removal of the k elements. If k elements are removed uniformly at random from R, then $\mathbb{E}[|V(R_{P_k})|] = v_{r-k}$. Another way to remove k elements is to maximize the number of violators after the removal. In the case of LP-type problems it is intuitive to remove in such a way that the objective function is minimized [23]. For this last rule [23] establishes a bound of $\mathbb{E}[|V(R_{P_k})|] = O(\frac{n-r}{r+1} \cdot \delta^{k+1})$ for fixed k, if the LP-type problem is nondegenerate.

6.3 An Upper Bound for Consistent Spaces

The main result of this section is Theorem 6.3.4, where we show an upper bound on $\mathbb{E}[|V(R_{P_k})|]$ for consistent spaces. In Lemma 6.5.1 and Lemma 6.5.3 we show asymptotically matching lower bounds for most relevant values of r.

We start with the following technical lemma, to obtain bounds on the bounded sets of the consistent space.

Lemma 6.3.1. *Let $(H, V, \mathcal{U})$, with $|H| = n$, a consistent space of dimension δ and P_k some fixed rule to remove k points. Let R be chosen uniformly at random from all sets of size $r \leq n$; let $c = 33$ and $x = c \cdot \max\left\{\frac{n}{r}\delta \ln r, \frac{n}{r}k\right\}$. We also assume $\delta, k \leq r/6$ and $x \leq n - \delta$. Then*

1. $$Pr[|V(R_{P_k})| \geq x \wedge R \notin \mathcal{U}] \leq \sum_{i=0}^{k} \sum_{\alpha=0}^{\delta} \frac{\binom{n}{\alpha}}{\binom{n}{r}} \binom{x}{i} \binom{n-x-\alpha}{r-\alpha-i}.$$

2. *For all $0 \leq \alpha \leq \delta$ and $0 \leq i \leq k$,*
$$\frac{\binom{n}{\alpha}}{\binom{n}{r}} \binom{x}{i} \binom{n-x-\alpha}{r-\alpha-i} \leq r^{-3}.$$

Proof of Lemma 6.3.1. For $R \subseteq H$, $R \notin \mathcal{U}$, define

$$\mathcal{B}(R_{P_k}) := \{B \subseteq H \mid B \text{ basis of } R_{P_k}\} \neq \emptyset,$$

the set of all bases of R_{P_k}. Let $R \subseteq H$ uniformly at random of size r. Then

$$\begin{aligned}
&\Pr[|V(R_{P_k})| \geq x \wedge R \notin \mathcal{U}] \\
&\quad = \Pr[\exists B \in \mathcal{B}(R_{P_k}) : |V(B)| \geq x \wedge R \notin \mathcal{U}] \\
&\quad \leq \Pr[\exists B \subseteq H : |V(B)| \geq x \wedge V(B) = V(R_{P_k}) \\
&\qquad\qquad \wedge |B| \leq \delta \wedge B \subseteq R \wedge R \notin \mathcal{U}] \\
&\quad \leq \Pr[\exists B \subseteq H : |V(B)| \geq x \wedge |V(B) \cap R| \leq k \\
&\qquad\qquad \wedge |B| \leq \delta \wedge B \subseteq R \wedge R \notin \mathcal{U}].
\end{aligned}$$

The last inequality follows since $V(R_{P_k}) = V(B)$ implies $R_{P_k} \cap V(B) = \emptyset$ (consistency), and because R_{P_k} is obtained from R by removing k elements, we have $|R \cap V(B)| \leq k$. Now by just dropping the last condition $R \notin \mathcal{U}$ we obtain

$$\begin{aligned}
&\Pr[|V(R_{P_k})| \geq x \wedge R \notin \mathcal{U}] \\
&\qquad \leq \Pr[\exists B \subseteq H : |V(B)| \geq x \wedge |V(B) \cap R| \leq k \wedge |B| \leq \delta \wedge B \subseteq R].
\end{aligned}$$

Using union bound we obtain

$$\begin{aligned}
&\Pr[|V(R_{P_k})| \geq x \wedge R \notin \mathcal{U}] \\
&\quad \leq \sum_{i=0}^{k} \sum_{\alpha=0}^{\delta} \Pr[\exists B \subseteq H : |V(B)| \geq x \wedge |V(B) \cap R| = i \\
&\qquad\qquad \wedge |B| = \alpha \wedge B \subseteq R] \\
&\quad \leq \sum_{i=0}^{k} \sum_{\alpha=0}^{\delta} \sum_{\substack{B \in \binom{H}{\alpha} \\ |V(B)| \geq x}} \Pr[|V(B) \cap R| = i \wedge B \subseteq R] \\
&\quad = \sum_{i=0}^{k} \sum_{\alpha=0}^{\delta} \sum_{\substack{B \in \binom{H}{\alpha} \\ n-\alpha \geq |V(B)| \geq x}} \frac{1}{\binom{n}{r}} \underbrace{\binom{|V(B)|}{i} \binom{n - |V(B)| - \alpha}{r - \alpha - i}}_{(*)}.
\end{aligned}$$

We now claim that $(*)$ is maximized if $|V(B)| = x$ which concludes the proof of part 1. This is clear if $x = n - \alpha$. If $x < n - \alpha$, we prove the claim by first observing (elementary calculations) that for $0 \leq y < n - \alpha$,

$$\binom{y}{i}\binom{n-y-\alpha}{r-\alpha-i} \geq \binom{y+1}{i}\binom{n-(y+1)-\alpha}{r-\alpha-i}$$

if and only if

$$y \geq \frac{i(n-\alpha)-r+i+\alpha}{r-\alpha}.$$

The claim follows if we can show that $y := x$ satisfies the latter inequality. Indeed, using $\alpha, i \leq r/6$, we have

$$\begin{aligned}\frac{i(n-\alpha)-r+i+\alpha}{r-\alpha} &< \frac{in}{r-\alpha} \leq \frac{6}{5}\frac{in}{r} \\ &< c\frac{kn}{r} \leq c \cdot \max\left\{\frac{n}{r}\delta \ln r, \frac{n}{r}k\right\} = x.\end{aligned}$$

For part 2, we argue as follows.

$$\begin{aligned}&\frac{\binom{n}{\alpha}}{\binom{n}{r}}\binom{x}{i}\binom{n-x-\alpha}{r-\alpha-i} \\ &\quad= \binom{x}{i} \cdot \frac{n!}{\underbrace{\alpha!}_{\geq 1}(n-\alpha)!} \cdot \frac{r!(n-r)!}{n!} \cdot \frac{(n-x-\alpha)!}{(r-\alpha-i)!(n-x-r+i)!} \\ &\quad\leq \binom{x}{i} \cdot \underbrace{\frac{r\cdots(r-\alpha-i+1)}{(n-\alpha)\cdots(n-\alpha-i+1)}}_{\leq r^\alpha\left(\frac{r}{n}\right)^i} \cdot \frac{(n-r)!}{(n-\alpha-i)!} \cdot \frac{(n-x-\alpha)!}{(n-x-r+i)!} \\ &\quad\leq \binom{x}{i} \cdot r^\alpha \left(\frac{r}{n}\right)^i \underbrace{\frac{(n-x-\alpha)\cdots(n-x-r+i+1)}{(n-\alpha-i)\cdots(n-r+1)}}_{\leq\left(\frac{n-x}{n-i}\right)^{r-\alpha-i}}.\end{aligned}$$

The upper bounds on the terms in the second and third line of this chain of inequalities employ the fact that $\frac{a-t}{b-t} \leq \frac{a}{b}$ if $0 \leq t < a \leq b$, using $r \leq n$ and $i \leq k < ck \leq x$.

We further have that

$$\begin{aligned}\left(\frac{n-x}{n-i}\right)^{r-\alpha-i} &= \left(1-\frac{x-i}{n-i}\right)^{r-\alpha-i}\\ &\le \exp\left(-\frac{(x-i)(r-\alpha-i)}{n-i}\right)\\ &< \exp\left(-\frac{xr}{2n}\right),\end{aligned}$$

using $\alpha, i \le r/6$ and $i \le k \le x/c \le n/c$ with $c = 33$.

Hence, we have now shown that

$$\frac{\binom{n}{\alpha}}{\binom{n}{r}}\binom{x}{i}\binom{n-x-\alpha}{r-\alpha-i} \le \binom{x}{i}\cdot r^{\alpha}\left(\frac{r}{n}\right)^{i}\exp\left(-\frac{xr}{2n}\right).$$

For $i = 0$, this is bounded by

$$r^{\delta}\exp\left(-\frac{c\delta\ln r}{2}\right) = r^{\delta-c\delta/2} \le r^{-3},$$

as desired. For $i > 0$, we now use the estimate $\binom{x}{i} \le (xe/i)^i$. In the first case we assume that $k \le \delta\ln r$, hence $x = c\frac{n}{r}\delta\ln r$. It follows that

$$\begin{aligned}&\frac{\binom{n}{\alpha}}{\binom{n}{r}}\binom{x}{i}\binom{n-x-\alpha}{r-\alpha-i}\\ &\quad\le \left(\frac{xe}{i}\right)^{i}\cdot r^{\alpha}\left(\frac{r}{n}\right)^{i}\exp\left(-\frac{xr}{2n}\right)\\ &\quad= \left(\frac{c\frac{n}{r}\delta\ln re}{i}\right)^{i}\cdot r^{\alpha}\left(\frac{r}{n}\right)^{i}\exp(-\frac{c}{2}\delta\ln r)\\ &\quad\le \left(\frac{c\delta\ln re}{i}\right)^{i}\cdot r^{\alpha}\exp(-\frac{c}{2}\delta\ln r)\\ &\quad= \exp(\overbrace{i\ln c}^{\le\delta\ln r\ln c} + i\ln\delta + i\ln\ln r + \overbrace{i}^{\le\delta\ln r} - i\ln i + \overbrace{\alpha\ln r}^{\le\delta\ln r} - \frac{c}{2}\delta\ln r)\\ &\quad\le \exp(i\ln\delta + i\ln\ln r - i\ln i - \left(\frac{c}{2} - 2 - \ln c\right)\delta\ln r).\end{aligned}$$

To show the claim it remains to show that

$$i \ln \delta + i \ln \ln r - i \ln i - \left(\frac{c}{2} - 2 - \ln c\right) \delta \ln r \leq -3 \ln r.$$

Since $i \leq k \leq \delta \ln r$ we can write $i = \beta \delta \ln r$ for some $\beta \in (0, 1]$. Then

$$\begin{aligned} i \ln \delta + i \ln \ln r - i \ln i - \left(\frac{c}{2} - 2 - \ln c\right) \delta \ln r \\ &= \beta \delta \ln r \ln \delta + \beta \delta \ln r \ln \ln r - \beta \delta \ln r (\ln \beta + \ln \delta + \ln \ln r) \\ &\quad - \left(\frac{c}{2} - 2 - \ln c\right) \delta \ln r \\ &= -\beta \ln \beta \delta \ln r - \left(\frac{c}{2} - 2 - \ln c\right) \delta \ln r. \end{aligned}$$

It remains to bound $\beta \ln \beta$ from below. By taking the derivative we observe that $\beta \ln \beta$ attains its minimum when $\beta = \frac{1}{e}$ and hence $\beta \ln \beta \geq -\frac{1}{e}$. It therefore follows that

$$\begin{aligned} i \ln \delta + i \ln \ln r - i \ln i - \left(\frac{c}{2} - 2 - \ln c\right) \delta \ln r \\ &\leq -\left(\frac{c}{2} - 2 - \ln c - \frac{1}{e}\right) \delta \ln r \leq -3 \ln r \text{ for } c \geq 33. \end{aligned}$$

In the second case $k \geq \delta \ln r$, hence $x = c \cdot \frac{n}{r} \cdot k$. Again, for $i \geq 1$ it follows that

$$\begin{aligned} \frac{\binom{n}{\alpha}}{\binom{n}{r}} \binom{x}{i} \binom{n - x - \alpha}{r - \alpha - i} &\leq \left(\frac{xe}{i}\right)^i \cdot r^\alpha \left(\frac{r}{n}\right)^i \exp(-\frac{x}{2n} r) \\ &= \left(c \frac{n}{r} k \frac{e}{i}\right)^i r^\alpha \left(\frac{r}{n}\right)^i \exp(-\frac{c}{2} k) \\ &= \exp(\overbrace{i \ln c}^{\leq k \ln c} + i \ln k + \overbrace{i}^{\leq k} - i \ln i + \overbrace{\alpha \ln r}^{\leq \delta \ln r \leq k} - \frac{c}{2} k) \\ &\leq \exp(i(\ln k - \ln i) - (\frac{c}{2} - 2 - \ln c) k). \end{aligned}$$

Now as before it suffices to show that

$$i(\ln k - \ln i) - \left(\frac{c}{2} - 2 - \ln c\right) k \leq -3 \ln r.$$

Since $i \leq k$ we can write $i = \beta k$ for some $\beta \in (0, 1]$. Then

$$\begin{aligned} & i(\ln k - \ln i) - \left(\frac{c}{2} - 2 - \ln c\right) k \\ & \quad = \beta k(\ln k - \ln k - \ln \beta) - \left(\frac{c}{2} - 2 - \ln c\right) k \\ & \quad \leq -\left(\frac{c}{2} - 2 - \ln c - \frac{1}{e}\right) \underbrace{k}_{\geq \delta \ln r} \leq -3 \ln r \text{ for } c \geq 33, \end{aligned}$$

where we again used that $\beta \ln \beta \geq -\frac{1}{e}$. □

The following lemma shows, that for unbounded sets removal after sampling only decreases the number of violators. Therefore for unbounded sets, $k = 0$ is the worst case.

Lemma 6.3.2. *Let $(H, V, \mathcal{U})$ be a consistent space and $F \subseteq G \subseteq H$, $G \in \mathcal{U}$. Then $V(F) \subseteq V(G)$.*

Proof. Since $G \in \mathcal{U}$, by monotonicity it follows that $F \in \mathcal{U}$. Now assume that $h \in H \setminus V(G)$. Then by the matching of V and $\mathcal{U}$ it follows that $G \cup \{h\} \in \mathcal{U}$ and therefore by monotonicity $F \cup \{h\} \in \mathcal{U}$. Now using the matching again we get that $h \in H \setminus V(F)$. It follows that if $h \notin G$, then $h \in V(F)$ implies $h \in V(G)$. If $h \in G$, then $F \cup \{h\} \subseteq G$ and hence by monotonicity $F \cup \{h\} \in \mathcal{U}$. By the matching it follows that $h \notin V(F)$. The claim of the lemma follows. □

As mentioned in the beginning of the chapter, in consistent spaces the Sampling Lemma is not very helpful as the number of extreme constraints can not be bounded by the dimension. However only considering the unbounded sets, there is the following version of the Sampling Lemma.

Lemma 6.3.3. *[Sampling Lemma for Unbounded Sets in Consistent Spaces] Let $(H, V, \mathcal{U})$ be a consistent space. Let R be uniformly at random (u.a.r.) from all sets of size r. Then*

$$\mathbb{E}[|V(R)| \mid R \in \mathcal{U}] \cdot Pr[R \in \mathcal{U}] \leq \frac{n - r}{r + 1} \cdot \delta.$$

Note that for violator spaces this follows directly by the Sampling Lemma (Theorem 6.2.8) since by total probability

$$\begin{aligned}&\mathbb{E}[|V(R)| \mid R \in \mathcal{U}] \cdot \Pr[R \in \mathcal{U}] + \mathbb{E}[|V(R)| \mid R \notin \mathcal{U}] \cdot \Pr[R \notin \mathcal{U}] \\ &\quad = \mathbb{E}[|V(R)|] = \frac{n-r}{r+1} \cdot x_{r+1} \leq \frac{n-r}{r+1} \cdot \delta.\end{aligned}$$

Proof. Assume that there exists $R \in \mathcal{U}$, otherwise this is clear. Let $G \subseteq H$. We say that $h \in G$ is *unbounded extreme*, if $h \in V(G \setminus \{h\})$ and $G \setminus \{h\} \in \mathcal{U}$. Hence h is extreme and $G \setminus \{h\}$ is unbounded. We denote the set of unbounded extreme constraints of G by $Y(G)$ and by y_r the expected size of $Y(R)$.

We first claim that $|Y(G)| \leq \delta$. If $G \in \mathcal{U}$, then by monotonicity and the matching of V and $\mathcal{U}$, $Y(G) = \emptyset$. Hence assume that $G \notin \mathcal{U}$. Then there exists a basis $B \subseteq G$ of G, such that $|B| \leq \delta$ and $B \notin \mathcal{U}$. For any $h \in G \setminus B$, $B \subseteq G \setminus \{h\}$ and therefore by monotonicity it follows that $G \setminus \{h\} \notin \mathcal{U}$. By definition of unbounded extreme constraints it follows that $h \notin Y(G)$. Hence the claim follows.

Similar as in the proof of the Sampling Lemma we now consider a bipartite graph. On the left side we have all *unbounded* sets of size r, i.e.,

$$A := \{R \subseteq H \mid |R| = r \wedge R \in \mathcal{U}\}.$$

On the right side we have *all* sets of size $r+1$. We connect two sets R and $R \cup \{h\}$ if $h \in V(R)$ or equivalently of $h \in Y(R \cup \{h\})$. The expected degree on the left side can be written as

$$\mathbb{E}[|V(R)| \mid R \in \mathcal{U}] = \frac{1}{|A|} \cdot \sum_{\substack{R \in \mathcal{U} \\ |R|=r}} |V(R)|.$$

On the right side similarly the expected degree can written as

$$\mathbb{E}[y_{r+1}] = \frac{1}{\binom{n}{r+1}} \cdot \sum_{\substack{S \subseteq H \\ |S|=r+1}} |Y(S)|.$$

Since $\sum_{\substack{R\in\mathcal{U}\\|R|=r}} |V(R)| = \sum_{\substack{S\subseteq H\\|S|=r+1}} |Y(S)|$ it follows that

$$\begin{aligned}\mathbb{E}[|V(R)| \mid R\in\mathcal{U}]\cdot\Pr[R\in\mathcal{U}] &= \frac{1}{|A|}\cdot\sum_{\substack{R\in\mathcal{U}\\|R|=r}}|V(R)|\cdot\frac{|A|}{\binom{n}{r}}\\ &= \frac{1}{\binom{n}{r}}\cdot\sum_{\substack{S\subseteq H\\|S|=r+1}}\underbrace{|Y(S)|}_{\leq\delta}\\ &\leq \frac{\binom{n}{r+1}}{\binom{n}{r}}\cdot\delta = \frac{n-r}{r+1}\cdot\delta.\end{aligned}$$

□

Theorem 6.3.4. *Let $(H,V,\mathcal{U})$, with $|H|=n$, a* consistent space *of dimension δ and P_k some fixed rule to remove k points. Let $R\subseteq H$ u.a.r. of all sets of size r, for some $r\leq n$. Then*

$$\mathbb{E}[|V(R_{P_k})|] \leq c\cdot\max\left\{\frac{n}{r}\delta\ln r, \frac{n}{r}k\right\} + \frac{n}{r}\cdot(\delta+1),$$

where c is some suitable constant from Lemma 6.3.1 (e.g. $c=33$).

Observe that compared to Lemma 6.2.8, for most relevant r (e.g. $r=n^\beta$, $\beta\in(0,1)$) and $k=o(\delta\ln r)$, there is an additional $\ln r$ term.

Proof of Theorem 6.3.4. We may assume $\delta\leq\frac{r}{6}$, $k\leq\frac{r}{c}$, and $r+x\leq n$, since otherwise the bound is trivial and there is nothing to prove.

By total expectation we get

$$\begin{aligned}\mathbb{E}[|V(R_{P_k})|] = \mathbb{E}[|V(R_{P_k})| \mid R\notin\mathcal{U}]\cdot\Pr[R\notin\mathcal{U}]\\ + \mathbb{E}[|V(R_{P_k})| \mid R\in\mathcal{U}]\cdot\Pr[R\in\mathcal{U}].\end{aligned}$$

Let us first consider the second term. By Lemma 6.3.2, for all unbounded R, $|V(R_{P_k})|\leq|V(R)|$. Therefore

$$\mathbb{E}[|V(R_{P_k})| \mid R\in\mathcal{U}] \leq \mathbb{E}[|V(R)| \mid R\in\mathcal{U}].$$

By Lemma 6.3.3 it follows that

$$\mathbb{E}[|V(R_{P_k})| \mid R \in \mathcal{U}] \cdot \Pr[R \in \mathcal{U}] \leq \frac{n-r}{r+1} \cdot \delta \leq \frac{n}{r} \cdot \delta.$$

Now consider the first term. Let $x = c \cdot \max\left\{\frac{n}{r}\delta \ln r, \frac{n}{r}k\right\}$. By definition of expectation

$$\begin{aligned}
&\mathbb{E}[|V(R_{P_k})| \mid R \notin \mathcal{U}] \cdot \Pr[R \notin \mathcal{U}] \\
&\quad \leq (\Pr[|V(R_{P_k})| < x \mid R \notin \mathcal{U}] \cdot (x-1) \\
&\qquad + \Pr[|V(R_{P_k})| \geq x \mid R \notin \mathcal{U}] \cdot n) \cdot \Pr[R \notin \mathcal{U}] \\
&\quad \leq x - 1 + \Pr[|V(R_{P_k})| \geq x \mid R \notin \mathcal{U}] \cdot \Pr[R \notin \mathcal{U}] \cdot n.
\end{aligned}$$

We will now show that $\Pr[|V(R_{P_k})| \geq x \mid R \notin \mathcal{U}] \cdot \Pr[R \notin \mathcal{U}] \leq r^{-1}$ which concludes the proof. By definition of conditional probability and Lemma 6.3.1,

$$\begin{aligned}
\Pr[|V(R_{P_k})| \geq x \mid R \notin \mathcal{U}] \cdot \Pr[R \notin \mathcal{U}] &= \Pr[|V(R_{P_k})| \geq x \wedge R \notin \mathcal{U}] \\
&\leq \sum_{i=0}^{k} \sum_{\alpha=0}^{\delta} \frac{\binom{n}{\alpha}}{\binom{n}{r}} \binom{x}{i} \binom{n-x-\alpha}{r-\alpha-i} \\
&\leq \sum_{i=0}^{k} \sum_{\alpha=0}^{\delta} r^{-3} \leq r^{-1}.
\end{aligned}$$

□

6.4 An Upper Bound for Violator Spaces

In this section we give an upper bound on $\mathbb{E}[|V(R_{P_k})|]$ for violator spaces, Theorem 6.4.7. This is an improvement of the bound given in [23], which stated the same bound for nondegenerate LP-type problems and the specific rule P_k to minimize the objective function after removal. Matching lower bounds for special nondegenerate cases are known [23]. We will give a lower bounds for the degenerate case in Section 6.6, which shows that other methods need to be applied to get a better upper bound. The bound in Theorem 6.4.7 is stronger than the bound in Theorem 6.3.4 if δ and k are very small, e.g., if $\delta^{2k} < \ln r$; for large δ and k, Theorem 6.3.4 is stronger.

6.4.1 Extreme Constraints after Removal

Let $(H, V, \mathcal{U})$ be a violator space of combinatorial dimension δ. In particular, every set has at most δ extreme constraints. For a given natural number k, we want to understand the following quantity:

$$\Delta_k(H, V, \mathcal{U}) := \max_{R \subseteq H} |\{X(R \setminus K) : K \subseteq R, |K| = k\}| .$$

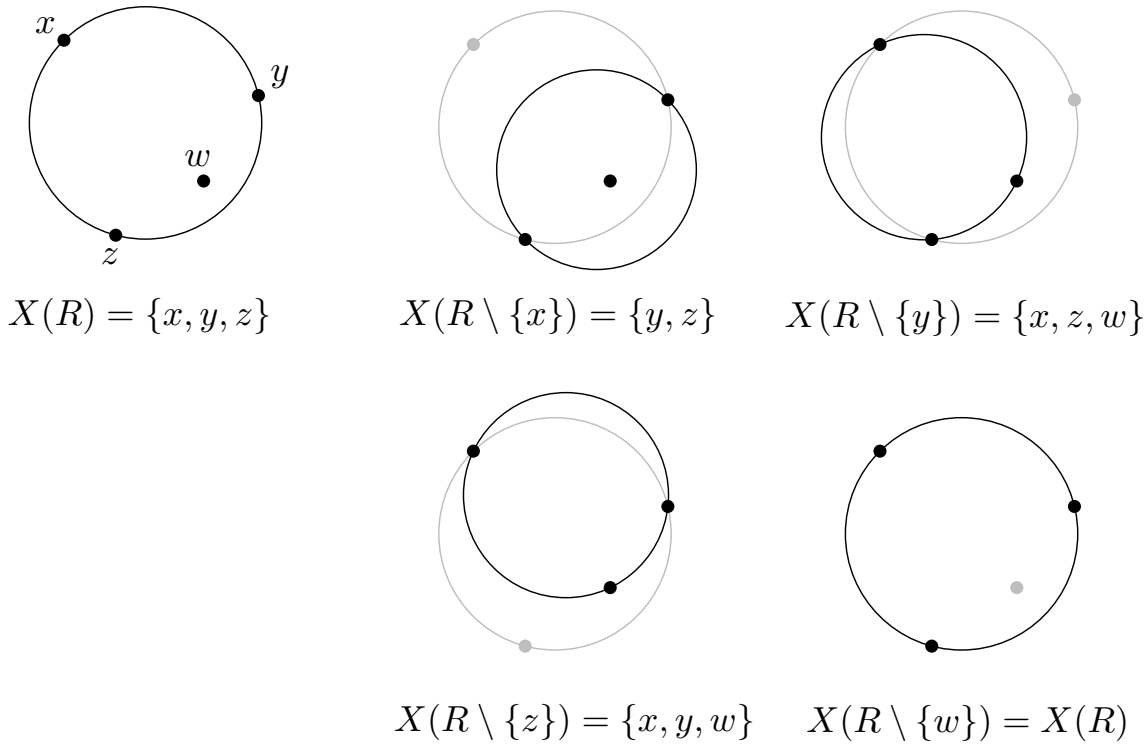

Figure 6.3: Example for $\Delta_1(H, V, \mathcal{U})$

In other words, how many sets of extreme constraints can we get by removing k elements from some set R?

Figure 6.3 shows an instance of SEB with $R = \{x, y, z, w\}$, where by removing one point from R we can get four different sets of extreme constraints ($\{x, y, z\}$, $\{y, z\}$, $\{x, z, w\}$ and $\{x, y, w\}$). Therefore we conclude that $\Delta_1(H, V, \mathcal{U}) \geq 4$. We will see below that for nondegenerate violator spaces $\Delta_1(H, V, \mathcal{U}) \leq \delta + 1$, so this bound is actually tight.

Let's bound the easy cases of $\Delta_k(H, V, \mathcal{U})$ first. We obviously have $\Delta_0(H, V, \mathcal{U}) = 1$ for any violator space $(H, V, \mathcal{U})$. We note that in the

nondegenerate case it holds that $X(R) = B(R)$, where $B(R)$ denotes the unique basis of R. Moreover for $(H, V, \mathcal{U})$ nondegenerate we have

$$\Delta_1(H, V, \mathcal{U}) \leq \delta + 1.$$

Indeed, if we remove a non-extreme element x from R, (by Lemma 6.4.1 below) we end up with the same set $X(R \setminus \{x\}) = X(R) = B(R)$ of extreme elements, so only in at most δ cases, we will get a different set. Note that in general this bound does not hold. Consider SEB and assume we have four points in general position on a circle and one point in the middle. It is not hard to see that for each point its removal generates a different set of extreme points. Hence $\Delta_1(H, V) \geq 5 > \delta + 1 = 4$. We will treat the general case in Lemma 6.4.4.

We start with a basic lemma about violator spaces.

Lemma 6.4.1. *Let $(H, V, \mathcal{U})$ be a violator space with $\emptyset \in \mathcal{U}$. For $R \subseteq H$ denote by $\mathcal{B}(R) := \{B \subseteq R \mid B \text{ basis of } R\}$, the set of all bases of R. Then the following holds.*

1. $X(R) = \bigcap_{B \in \mathcal{B}(R)} B$.
2. *Let $x \in R \setminus X(R)$. Then $\mathcal{B}(R \setminus \{x\}) = \{B \in \mathcal{B}(R) \mid x \notin B\}$.*

Proof. 1. If $R \in \mathcal{U}$, then by definition the empty set is the unique basis. Moreover by monotonicity and the matching of V and $\mathcal{U}$ it follows that $X(R) = \emptyset$.

Hence assume that $R \notin \mathcal{U}$. Let $x \in R \setminus \bigcap_{B \in \mathcal{B}(R)} B$. Then there exists $B' \in \mathcal{B}(R)$ such that $x \notin B'$. By locality it follows that $V(B') = V(R \setminus \{x\}) = V(R)$. In particular $x \notin V(R \setminus \{x\})$ and hence by definition $x \notin X(R)$.

For the other direction assume that $x \in R \setminus X(R)$. Then by the matching of V and $\mathcal{U}$ it follows that $R \setminus \{x\} \notin \mathcal{U}$. It follows by locality that $V(R) = V(R \setminus \{x\}) = V(B^*)$, where B^* is some basis of $R \setminus \{x\}$. Note that B^* is also a basis of R and since $x \notin B^*$ it follows that $x \notin \bigcap_{B \in \mathcal{B}(R)} B$.

2. For $R \in \mathcal{U}$ this is clear as $\mathcal{B}(R \setminus \{x\}) = \mathcal{B}(R) = \{\emptyset\}$.

   Hence assume $R \notin \mathcal{U}$ and let $x \in R$ such that $x \notin B$ for some $B \in \mathcal{B}(R)$. As $B \subseteq R \setminus \{x\}$, and B is bounded it follows by monotonicity that $R \setminus \{x\}$ is bounded. Since $x \notin V(R \setminus \{x\})$, by locality it follows that $B \in \mathcal{B}(R \setminus \{x\})$. For the other direction let $B \in \mathcal{B}(R \setminus \{x\})$. Assume that $B \notin \mathcal{B}(R)$. Then $V(B) = V(R \setminus \{x\}) \neq V(R)$, hence by locality $x \in V(R \setminus \{x\})$. Therefore by definition of extreme points and the first part $x \in X(R) = \bigcap_{B \in \mathcal{B}(R)} B$, which is a contradiction to the choice of x.

□

For $R \subseteq H$ define $\beta(R) = |\bigcap_{B \in \mathcal{B}(R)} B|$, the size of the intersection of all of its bases. Let $b_0(R) = \min_{B \in \mathcal{B}(R)} |B|$, the minimum size of a basis in R. In particular in the nondegenerate case we have $\beta(R) = b_0(R)$.

Lemma 6.4.2. *Let $R \subseteq H$ and $\mathcal{B}(R)$ the bases of R. Then there exists a set $\overline{\mathcal{B}}(R) \subseteq \mathcal{B}(R)$, such that*

$$\bigcap_{B \in \overline{\mathcal{B}}(R)} B = \bigcap_{B \in \mathcal{B}(R)} B,$$

and

$$|\bigcup_{B \in \overline{\mathcal{B}}(R)} B| \leq (\delta - \beta(R)) \cdot (b_0(R) - \beta(R)) + b_0(R) \leq \delta^2 + \delta.$$

Proof. Fix $B_0 \in \mathcal{B}(R)$ of minimum size, i.e., $|B_0| = b_0(R)$. Let us label the points in B_0 such that $B_0 = (\bigcap_{B \in \mathcal{B}(R)} B) \cup \{1, 2, \ldots, b_0(R) - \beta(R)\} = (\bigcap_{B \in \mathcal{B}(R)} B) \cup [b_0(R) - \beta(R)]$. Then for any $i \in [b_0(R) - \beta(R)]$ there exists $B_i \in \mathcal{B}$ such that $i \notin B_i$. Define

$$\overline{\mathcal{B}}(R) = \{B_0, B_1, \ldots, B_{b_0(R)-\beta(R)}\}.$$

It follows by construction that $\bigcap_{B \in \overline{\mathcal{B}}(R)} B = \bigcap_{B \in \mathcal{B}(R)} B$. Moreover $\overline{\mathcal{B}}(R)$ contains at most $b_0(R) - \beta(R) + 1$ elements, where each $B_i \in \mathcal{B}(R)$,

$i \neq 0$ contains at most $\delta - \beta(R)$ elements that are not in B_0. It hence follows that

$$|\bigcup_{B \in \overline{\mathcal{B}}(R)} B| \leq (\delta - \beta(R)) \cdot (b_0(R) - \beta(R)) + b_0(R) \leq \delta^2 + \delta,$$

where the last step holds since $b_0(R) \leq \delta$. We see that this (not necessarily unique) set $\overline{\mathcal{B}}(R)$ satisfies all the conditions. □

Lemma 6.4.3. *Let $K \subseteq R$ such that $K \cap \left(\bigcup_{B \in \overline{\mathcal{B}}(R)} B\right) = \emptyset$. Then $X(R \setminus K) = X(R)$.*

Proof. Using the second part of Lemma 6.4.1 repeatedly, by our choice of K we get

$$\mathcal{B}(R \setminus K) = \{B \in \mathcal{B}(R) \mid K \cap B = \emptyset\}.$$

It therefore follows that $\overline{\mathcal{B}}(R) \subseteq \mathcal{B}(R \setminus K) \subseteq \mathcal{B}(R)$. We get that

$$\bigcap_{B \in \mathcal{B}(R)} B \subseteq \bigcap_{B \in \mathcal{B}(R \setminus K)} B \subseteq \bigcap_{B \in \overline{\mathcal{B}}(R)} B = \bigcap_{B \in \mathcal{B}(R)} B,$$

where the last step follows from the definition of $\overline{\mathcal{B}}(R)$. Using the first part of Lemma 6.4.1 we get $X(R \setminus K) = X(R)$. □

Lemma 6.4.4. *Let $(H, V, \mathcal{U})$ be a violator space with $\emptyset \in \mathcal{U}$. Then*

$$\Delta_k(H, V, \mathcal{U}) \leq \sum_{i=0}^{k} (\delta^2 + \delta)^i = O(\delta^{2k} + k).$$

Note that $\Delta_k(H, V, \mathcal{U})$ is independent of the number of bases of R.

Proof of Lemma 6.4.4. Let us fix R and a set $K \subseteq R$, $|K| = k$ to be removed. We claim that we can order the elements of K as $e_1, \ldots, e_k$ such that for some $\ell \in \{0, \ldots, k\}$,

$$e_i \in \overline{\mathcal{B}}(R \setminus \{e_1, \ldots, e_{i-1}\}), i \leq \ell,$$
$$X(R \setminus K) = X(R \setminus \{e_1, \ldots, e_\ell\}).$$

Indeed, we can do this greedily: as long as we can remove an element of $\bigcup_{B\in\overline{\mathcal{B}}(S)} B$ of the current set S, we do so. At some point none of the elements that remain to be removed from the current set S are in $\bigcup_{B\in\overline{\mathcal{B}}(S)} B$. By Lemma 6.4.3 at this point the removal of any subset of them does not change the extreme elements anymore.

If follows that all sets $X(R\setminus K)$ can be obtained from R by repeatedly removing an element of $\bigcup_{B\in\overline{\mathcal{B}}(S)} B$ from the current set S, up to k times. In the first round we have at most $|\bigcup_{B\in\overline{\mathcal{B}}(R)} B| \leq \delta^2+\delta$ choices, and for each of them, we have at most $\delta^2+\delta$ in the second round, and so on. The bound follows. □

For the nondegenerate case the corollary below, (a strengthening of the result of [23]) follows.

Corollary 6.4.5. *For $(H,V,\mathcal{U})$ nondegenerate $\emptyset\in\mathcal{U}$*

$$\Delta_k(H,V,\mathcal{U}) \leq \sum_{i=0}^{k} \delta^i = O(\delta^k + k).$$

Proof. Since in the nondegenerate case $\beta(R) = b_0(R)$ by Lemma 6.4.2 it follows that $|\bigcup_{B\in\overline{\mathcal{B}}(S)} B| \leq \delta$ for all R. This can also be seen directly as $\mathcal{B}(R)$ has contains only one set, which is of size at most δ. We then apply the same argument as in Lemma 6.4.4. □

6.4.2 Sampling Lemma after Removal

Let $(H,V,\mathcal{U})$ be a violator space. For $R\subseteq H$ and a natural number k, we define the following two quantities.

$$\begin{aligned} V_k(R) &= \{x\in H\setminus R : x\in V(R\setminus K) \text{ for some } K\subseteq R, |K|=k\},\\ X_k(R) &= \{x\in R : x\in X(R\setminus K) \text{ for some } K\subseteq R, |K|=k\}. \end{aligned}$$

Clearly, $V(R) = V_0(R)$ and $X(R) = X_0(R)$.

Furthermore, we let $v_{r,k}$ denote the expected size of $V_k(R)$ over a randomly chosen set of size r. Similarly, $x_{r,k}$ is the expected size of $X_k(R)$.

Lemma 6.4.6. *[Sampling Lemma after Removal]*

$$v_{r,k} = \frac{n-r}{r+1} x_{r+1,k}.$$

Proof. This goes like for the "normal" Sampling Lemma 6.2.8 [28]. We define a bipartite graph on the vertex set $\binom{H}{r} \cup \binom{H}{r+1}$, where we connect R and $R \cup \{x\}$ with an edge if and only if $x \in V_k(R)$. Let $x \in H \setminus R$. We have the following equivalences:

$$\begin{aligned}
x \in V_k(R) &\Leftrightarrow x \in V(R \setminus K) \text{ for some } K \subseteq R, |K| = k \\
&\Leftrightarrow x \in X((R \setminus K) \cup \{x\}) \\
&\qquad \text{for some } K \subseteq R, |K| = k \\
&\Leftrightarrow x \in X((R \cup \{x\}) \setminus K) \\
&\qquad \text{for some } K \subseteq R, |K| = k,\ x \notin K \\
&\Leftrightarrow x \in X((R \cup \{x\}) \setminus K) \\
&\qquad \text{for some } K \subseteq R \cup \{x\}, |K| = k,\ x \notin K \\
&\Leftrightarrow x \in X((R \cup \{x\}) \setminus K) \\
&\qquad \text{for some } K \subseteq R \cup \{x\}, |K| = k \\
&\Leftrightarrow x \in X_k(R \cup \{x\}),
\end{aligned}$$

where in the fifth step we used that $x \in X((R \cup \{x\}) \setminus K)$ can only occur if $x \in (R \cup \{x\}) \setminus K$.

So we can also define the graph as having an edge between R and $R \cup \{x\}$ if and only if $x \in X_k(R \cup \{x\})$. Since the sum of the degrees of the vertices in $\binom{H}{r}$ is the same as the sum of degrees of the vertices in $\binom{H}{r+1}$, we get

$$\binom{n}{r} v_{r,k} = \sum_{a \in \binom{H}{r}} \deg(a) = \sum_{b \in \binom{H}{r+1}} \deg(b) = \binom{n}{r+1} x_{r,k},$$

which is the claimed result. □

Note that as in the "normal" Sampling Lemma, the result holds as well for consistent spaces.

6.4.3 Violators after Removal

Suppose we sample R at random, and then remove an arbitrary set of k elements K_R according to some fixed rule P_k, and obtain the set $R_{P_k} = R \setminus K_R$. The expected number of violators of R_{P_k} is bounded by $v_{r,k} + k$. This follows since $v_{r,k}$ counts the expected number of violators in $H \setminus R$ that we can possibly get by removing *any* set of k elements and the removed points K_R can also be in $V(R_{P_k})$. Therefore $E[|V(R_{P_k})|] \leq v_{r,k} + k$.

Theorem 6.4.7. *Let $(H, V, \mathcal{U})$ be a violator space of dimension δ, $\emptyset \in \mathcal{U}$, and let R be sampled u.a.r. from all subsets of H of size r. Let P_k be a fixed rule to remove k elements from the random sample. Then*

$$\begin{aligned}\mathbb{E}[|V(R_{P_k})|] \leq v_{r,k} + k &\leq \left(\sum_{i=0}^{k} (\delta^2 + \delta)^i\right) \cdot \delta \cdot \frac{n-r}{r+1} + k \\ &= \frac{n-r}{r+1} \cdot O(\delta^{2k+1} + k) + k.\end{aligned}$$

Proof of Theorem 6.4.7. By Lemma 6.4.6, we need to bound $x_{r,k}$. To this end, we show that for all R,

$$|X_k(R)| \leq \left(\sum_{i=0}^{k} (\delta^2 + \delta)^i\right) \cdot \delta.$$

This holds, since by Lemma 6.4.4, at most $\sum_{i=0}^{k} (\delta^2 + \delta)^i$ many *sets* of extreme elements can be obtained by removing k elements from R, and each of these sets has at most δ elements. □

Corollary 6.4.8. *For $(H, V, \mathcal{U})$ a nondegenerate violator space it holds that*

$$\mathbb{E}[|V(R_{P_k})|] = \frac{n-r}{r+1} \cdot O(\delta^{k+1} + k) + k.$$

Proof. The statement follows by the same argument as in the proof of 6.4.7 using the result of Corollary 6.4.5. □

6.5 A Lower Bound for Consistent Spaces

In this section we show the matching lower bound of Theorem 6.3.4 for consistent spaces for most relevant sizes of r, δ and k.

Lemma 6.5.1. *Let $r = n^\alpha$, let $\alpha \in (0,1)$, $0 < \epsilon < \alpha$, $\gamma < \alpha - \epsilon$ be constants, and $1 \leq \delta \leq n^\gamma$. Let k, P_k as in Theorem 6.3.4. Let $x = \epsilon \frac{n}{r} \delta \ln n = o(n)$. Then there exists a consistent space without minus infinity (H, V) of dimension δ, such that*

$$\mathbb{E}[|V(R_{P_k})|] = (1 + o(1))\epsilon \frac{n}{r} \delta \ln n = (1 + o(1))x.$$

Note that here $\ln n = \alpha \ln r$.

Proof. Define (H, V) consistently (with $\mathcal{U} = \emptyset$) as follows. The violator set of the empty set is defined as the empty set, $V(\emptyset) = \emptyset$. For all $B \subseteq H$ with $0 < |B| \leq \delta$, its violators are chosen u.a.r. of size $\epsilon \frac{n}{r} \delta \ln n$ from $H \setminus B$.

For $R \subseteq H$ of size r we define the violators as follows. If there exists $B \subseteq R$, $0 < |B| \leq \delta$, such that $V(B) \cap R = \emptyset$, then $V(R) = V(B)$. If there exists more than one such B, choose the lexicographically smallest. If no such B exists then set $V(R) = V(\emptyset) = \emptyset$. Therefore for all R we have a basis of size at most δ. Denote the basis for R by B_R.

First we show that it suffices to treat the "worst case" $k = 0$. For $k > 0$ we can reduce the problem to the case $k = 0$ by the following construction, where for all R of size r it holds that $|V(R_{P_k})| \geq |V(R)|$: For R with $V(R) \neq \emptyset$, fix P_k such that none of the k removed elements are in B_R, i.e., $B_R \subseteq R_{P_k}$. Since $R \cap V(B) = \emptyset$ it follows that $R_{P_k} \cap V(B) = \emptyset$ and we can choose $V(R_{P_k}) = V(R)$. If there exists multiple sets of size r with nonempty violator set that are mapped to the same R_{P_k}, choose $V(R_{P_k})$ arbitrary from the set of their violator spaces. For all other sets of size $r - k$, choose their violators as the empty set. It follows that for all R of size r, $|V(R_{P_k})| \geq |V(R)|$. For all other $S \subseteq H$ define $V(S) = \emptyset$.

Hence we may assume that $k = 0$.

$$\begin{aligned}\mathbb{E}[|V(R)|] &= \Pr[|V(R)| = \epsilon\frac{n}{r}\delta \ln n] \cdot \epsilon\frac{n}{r}\delta \ln n \\ &= (1 - \Pr[|V(R)| = 0]) \cdot \epsilon\frac{n}{r}\delta \ln n.\end{aligned}$$

We now show that $\Pr[|V(R)| = 0] = o(1)$, which concludes the proof. Because we chose the violators of the bases independently

$$\begin{aligned}\Pr[|V(R)| = 0] &= \Pr[\forall B \subseteq R, 0 < |B| \leq \delta \mid V(B) \cap R \neq \emptyset] \\ &= \prod_{\substack{B \subseteq R \\ 0<|B|\leq\delta}} \Pr[V(B) \cap R \neq \emptyset] \\ &= \prod_{\substack{B \subseteq R \\ 0<|B|\leq\delta}} (1 - \Pr[V(B) \cap R = \emptyset]).\end{aligned}$$

Now we bound $\Pr[V(B) \cap R = \emptyset]$ from below. For B of size β with $0 < \beta \leq \delta$ and $x = \epsilon\frac{x}{r}\delta \ln n$, we get

$$\begin{aligned}\Pr[V(B) \cap R = \emptyset] &= \frac{\binom{n-x-\beta}{r-\beta}}{\binom{n-\beta}{r-\beta}} = \frac{(n-x-\beta)\cdots(n-x-r+1)}{(n-\beta)\cdots(n-r+1)} \\ &= e^{\left((1+o(1))\frac{x}{n}+\Theta(\frac{x^2}{n^2})\right)r} = n^{-(1+o(1))\epsilon\delta}.\end{aligned}$$

Using that $\sum_{i=1}^{\delta}\binom{r}{i} \geq \frac{(r-\delta)^\delta}{\delta^\delta}$ and $\delta = o(r)$ we get

$$\begin{aligned}\Pr[|V(R)| = 0] &\leq \prod_{\substack{B \subseteq R \\ 0<|B|\leq\delta}} (1 - n^{-(1+o(1))\epsilon\delta}) \leq (1 - n^{-(1+o(1))\epsilon\delta})^{\frac{(r-\delta)^\delta}{\delta^\delta}} \\ &\leq \exp\left(-(1+o(1))n^{-(1+o(1))\epsilon\delta} \cdot \frac{(1+o(1))^\delta r^\delta}{\delta^\delta}\right).\end{aligned}$$

Plugging in $r = n^\alpha$, we observe that is sufficient to show that

$$n^{(\alpha-\epsilon+o(1))\delta}(1+o(1))^\delta \cdot \frac{1}{\delta^\delta} = \omega(1)$$

. By using $\delta \leq n^\gamma$ we get

$$\frac{n^{(\alpha-\epsilon+o(1))\delta} \cdot (1+o(1))^\delta}{\delta^\delta} \geq \left(n^{(\alpha-\epsilon-\gamma+o(1))} \cdot (1+o(1))\right)^\delta = \omega(1),$$

since $\gamma < \alpha - \epsilon$. □

We show that using one of the simplest violator spaces, namely the d-smallest number problem, we obtain the bound of $\mathbb{E}[|V(R_{P_k})|] = \Theta(\frac{n}{r} \cdot (\delta + k))$.

Definition 6.5.2. We define the *d-smallest number problem* as follows. Let $H = [n] = \{1, 2, \ldots, n\}$. For $R \subseteq H$, $R \neq \emptyset$ define $\min_d(R)$ as the d-smallest number in R. Let $V(R) = \{r \in H \setminus R \mid r < \min_d(R)\}$, i.e. all elements smaller than the d-smallest. For the empty set define $V(\emptyset) = H$.

We observe that $(H, V, \{\emptyset\})$ is a violator space, with combinatorial dimension d. The basis of R consists of the d smallest elements of R.

Lemma 6.5.3. *Let $H = [n]$ and $\delta + k \leq r$. For the δ-smallest number problem there exists a rule P_k such that $\mathbb{E}[|V(R_{P_k})|] = \frac{n-r}{r+1} \cdot (\delta+k) + k$, and therefore for $r = o(n)$, $\mathbb{E}[|V(R_{P_k})|] = \Theta(\frac{n}{r} \cdot (\delta + k))$.*

Proof of Lemma 6.5.3. Let $R \subseteq H$. To maximize the number of violators after the removal of k elements, we remove the k-smallest elements of R. We denote the removed set by R_k. Then

$$V(R_{P_k}) = \{r \in H \setminus R \mid r < \min_{\delta+k}(R)\} \cup R_k.$$

We observe that $\{r \in H \setminus R \mid r < \min_{\delta+k}(R)\}$ is exactly the set of violators of R, for the $\delta + k$ smallest problem, whose expected size we know by the Sampling Lemma 6.2.8. Hence

$$\mathbb{E}[|V(R_{P_k})|] = \frac{n-r}{r+1}(\delta + k) + k.$$ □

Lemma 6.5.1 and Lemma 6.5.3 show that for consistent spaces the bound of Theorem 6.3.4 is tight up to a constant factor for most relevant values of r, δ and k, (i.e., if r, δ and k satisfy the conditions of Lemma 6.5.1 or Lemma 6.5.3). Furthermore by Lemma 6.5.3 if $k \geq \delta \ln r$, then the upper bound of Theorem 6.3.4 is tight for violator spaces.

6.6 Lower Bounds for Violator Spaces

By [23, Section 7.2], there exists an LP-type problem and a rule P_k, such that $|X_k(R)| = \Theta(\delta^{k+1})$, for $|R| = n - 1$. However, the behavior of the bound is unknown for general r. In Lemma 6.6.2 we show that the bounds of Lemma 6.4.4 are tight up to a factor depending on k. We furthermore show in Lemma 6.6.3, that the bound of Corollary 6.4.5 for nondegenerate violator spaces is also tight up to a factor depending on k. It is not clear whether the bounds of Theorem 6.4.7 are tight for violator spaces, but this section shows that using our methods better bounds on the expectation $x_{r,k}$ can not be obtained.

The following lemma shows how given a set H and subsets of size at most δ, we can construct a violator space of dimension at most δ such that said subsets are exactly the bases of H.

Lemma 6.6.1. *Let H be a set of size n. Let $B_1, \ldots, B_m \subseteq H$, with $\max_{i \in [m]} |B_i| = \delta$ and there is no pair $i \neq j$ such that $B_i \subseteq B_j$. Then there exists a function $V : 2^H \mapsto 2^H$ and $\mathcal{U} \subseteq 2^H$, such that*

- *$(H, V, \mathcal{U})$ is a violator space,*
- *$\mathcal{B}(H) = \{B_1, \ldots, B_m\}$,*
- *$\dim(H, V, \mathcal{U}) = \delta$.*

Proof. We first define $\mathcal{U}$ to be the sets that do not contain any of the B_i, i.e.,

$$\mathcal{U} = \{R \subseteq H \mid B_i \nsubseteq R, \forall i \in [m]\}.$$

For a set $R \in \mathcal{U}$ we define the set of violators as

$$V(R) = \{x \in H \mid \exists i \in [m] \text{ such that } B_i \subseteq R \cup \{x\}\}.$$

For all sets $R \notin \mathcal{U}$ we define $V(R) = \emptyset$. We first observe that this is a violator space. By construction, consistency is satisfied. Locality is also trivially satisfied, since all $R \notin \mathcal{U}$ have the same set of violators. Monotonicity and the matching of V and $\mathcal{U}$ also hold by construction.

We now show the remaining two properties. By construction indeed $V(H) = V(B_i) = \emptyset$ for all $i \in [m]$. Since the only other sets S with $V(S) = V(H)$ contain some B_i, it follows that

$$\mathcal{B}(H) = \{B_1, \ldots, B_m\}.$$

By the second part of Lemma 6.4.1, we know that for all sets $R \notin \mathcal{U}$, $\mathcal{B}(R) \subseteq \mathcal{B}(H)$, hence it follows that the dimension is δ. □

Lemma 6.6.2. *There exists a violator space $(H, V, \mathcal{U})$ where*

$$\Delta_k(H, V, \mathcal{U}) \geq \binom{\frac{\delta}{2}}{2k} = \Omega\left(\frac{\delta^{2k}}{(4k)^{2k+1/2}}\right).$$

Proof. Note that the last equality follows from Stirling's formula.

Let H be of size $\Theta(\frac{\delta^2}{8})$ where the points of H are labeled by $\{1, \ldots, \frac{\delta}{2}\}$ and $\{ij \mid i, j \in \left[\frac{\delta}{2}\right], i < j\}$. Suppose H has $\frac{\delta}{2}$ bases where B_i, $i \in \left[\frac{\delta}{2}\right]$ is given by

$$B_i = \left(\left[\frac{\delta}{2}\right] \setminus \{i\}\right) \cup \{ji \mid j < i\} \cup \{ij \mid j > i\},$$

i.e., all points in $\left[\frac{\delta}{2}\right]$ except i, plus all pairs containing i. This can be extended to a violator space by Lemma 6.6.1. We observe that

$$|B_i| = \frac{\delta}{2} - 1 + \frac{\delta}{2} - 1 < \delta.$$

Let us first consider the case where $k = 1$. By the second part of Lemma 6.4.1, for $i \in \left[\frac{\delta}{2}\right]$

$$X(H \setminus \{i\}) = B_i.$$

For $i < j$,

$$X(H \setminus \{ij\}) = \bigcap_{k \neq i,j} B_k = \{i, j\}.$$

Since removal of each element in H induces a different extreme set it follows that $\Delta_1(R) = \frac{\delta^2}{4}$ and hence

$$\Delta_1(H, V, \mathcal{U}) \geq \frac{\delta^2}{4}.$$

The case $k > 1$ works similarly. Let $A \subseteq [\frac{\delta}{2}]$ of size $2k$. We will show that $A = X(H \setminus K)$ for some $K \subseteq H$, $|K| = k$, which implies the claim. W.l.o.g. assume that $A = \{1, 2, \ldots, 2k\}$. Let $K = \{12, 34, \ldots, 2(k-1)2k\}$. Using Lemma 6.4.1 repeatedly we get that

$$X(H \setminus K) = \bigcap_{i \in [\frac{\delta}{2}] \setminus [2k]} B_i = A.$$

□

Lemma 6.6.3. *There exists a* nondegenerate *violator space* $(H, V, \{\emptyset\})$, *where*

$$\Delta_k(H, V, \{\emptyset\}) \geq \binom{\delta}{k} = \Omega\left(\frac{\delta^k}{k^{k+1/2}}\right).$$

This shows that the bound of Corollary 18 is tight up to a factor depending on k.

Proof. Again we explicitly construct a violator space that satisfies the claim. Let $(H, V, \{\emptyset\})$ as follows. Let $H = B \cup [n - \delta]$, where B is a set of size δ. We define the violators as follows.

- Let $V(\emptyset) = H$.
- For $R \subseteq H$ with $R \cap B = C \neq \emptyset$, let $V(R) = B \setminus C$. In particular $V(B) = V(H) = \emptyset$.
- For $R \subseteq H$ with $R \cap B = \emptyset$ (i.e. $R \subseteq [n - \delta]$), let $V(R) = \{x \in [n - \delta] \mid x < \min_{i \in R} i\} \cup B$.

We claim that this is a nondegenerate violator space of dimension δ. Note that this implies the lemma: Consider some $k < \delta$ and let $B' \subseteq B$ of size k. Then $V(H \setminus B') = V(B \setminus B')$. Since the violator space is nondegenerate it follows that $B \setminus B'$ is the unique basis of $H \setminus B'$ and therefore by Lemma 6.4.1 $X(H \setminus B') = B \setminus B'$. Hence for every subset of B of size k we get a different set of extreme points which proves the lemma.

It remains to prove the claim. Note that consistency, monotonicity and matching of V and $\mathcal{U}$ are clear. It remains to show locality, nondegeneracy and that the dimension is δ.

For locality assume first that $G \subseteq H$ with $G \cap B = C \neq \emptyset$. Now for $F \subseteq G$, $V(F) \cap G = \emptyset$ holds if and only if $C \subseteq F$. In that case $V(F) = V(G) = B \setminus C$, hence locality holds. For the second case assume that $G \subseteq H$ with $G \cap B = \emptyset$. Then for $F \subseteq G$, $V(F) \cap G = \emptyset$ holds if and only if $\min_{i \in G} i \in F$. In this case again $V(F) = V(G) = \{x \in [n - \delta] \mid x < \min_{i \in G} i\} \cup B$. This shows locality.

It remains to show that every set has a unique basis of size at most δ. Let $G \subseteq H$ with $G \cap B = C \neq \emptyset$. We observe that for $F \subseteq G$, $V(F) = V(G)$ holds if and only if $C \subseteq F$, hence it follows that C is the unique basis of G. For the other case let $G \subseteq H$ with $G \cap B = \emptyset$. Since for $F \subseteq G$, $V(F) = V(G)$ holds if and only if $\min_{i \in G} i \in F$, it follows that $\min_{i \in G} i$ is the unique basis of F. It follows that $(H, V, \{\emptyset\})$ is a nondegenerate violator space of dimension δ. □

6.7 Characterization of Violator Spaces with Combinatorial Dimension 1

In general it is open whether (or when) the upper bound of Theorem 6.4.7 and Theorem 6.3.4 (for $k < \delta \ln r$) is tight. However for $\delta = 1$ Theorem 6.4.7 and Lemma 6.5.3 show tightness of this upper bound.

Corollary 6.7.1. *For $\delta = 1$, $\mathbb{E}[|V(R_{P_k})|] = O(\frac{n}{r}k)$, and this bound is tight.*

In the following we give a complete characterization of violator spaces with combinatorial dimension 1. We prove that there exists only one class of violator spaces of dimension 1, namely the class of the *smallest number with repetitions* violator space.

Definition 6.7.2. We define the class of *smallest number with repetitions* violator space as follows. Let $|H| = n$ and H a multiset of $[n]$, i.e., every element of H is in $[n]$ and there might be repetitions. For $R \subseteq H$, $R \neq \emptyset$, let $V(R) = \{x \in H \mid x < \min_{i \in R} i\}$. Finally we require that either $V(\emptyset) = H$ or $V(\emptyset) = V(\max_{i \in H} i)$.

Observe that this a a violator space of dimension 1 and similarly as in the proof of Lemma 6.5.3, we can show that $\mathbb{E}[|V(R_{P_k})|] = O(\frac{n}{r}k)$. The following two lemmas show how the violator spaces with dimension 1 have a somewhat simple structure.

Lemma 6.7.3. *Let $(H, V, \mathcal{U})$ be a violator space of dimension 1 with $\emptyset \in \mathcal{U}$. Then there exist H_1, H_2, such that $H = H_1 \cup H_2$, $H_1 \cap H_2 = \emptyset$ and $\mathcal{U} = 2^{H_2}$. Moreover for all $R \in \mathcal{U}$, $V(R) = H_1$.*

Proof. Define $H_2 := \{x \in H \mid \{x\} \in \mathcal{U}\}$, the set of all unbounded singletons. Now let $R \subseteq H$, such that $h \in R \cap H_1 \neq \emptyset$. Since $h \notin \mathcal{U}$, by monotonicity it follows that $R \notin \mathcal{U}$. This shows that $\mathcal{U} \subseteq 2^{H_2}$. Now let $R \subseteq H_2$ and assume for contradiction that R is bounded. Since the dimension is 1, it follows that $V(R) = V(i)$ for some bounded $i \in R$. This is a contradiction, since all elements of R are unbounded. Therefore $\mathcal{U} = 2^{H_2}$. The second part of the lemma follows directly from the matching of V and $\mathcal{U}$. □

Lemma 6.7.4. *Let $(H, V, \mathcal{U})$ be a violator space of dimension 1 with $\emptyset \in \mathcal{U}$, where H is a multiset of $[n]$. If for all $i \in H$, $V(i) = \{x \in H \mid x < i\}$, then for all $R \subseteq H$, $R \neq \emptyset$ we have $V(R) = \{x \in H \mid x < \min_{i \in R} i\}$. Moreover $V(\emptyset) = H$ or $V(\emptyset) = V(\max_{i \in H} i)$. This shows that the violators of sets of size 1 uniquely define the violators of larger sets.*

Proof. Let H_1, H_2 as in Lemma 6.7.3. By Lemma 6.7.3, for all $j \in H_2$ we know that $V(j) = H_1 = \{x \in H \mid x < j\}$. Therefore $j = \max_{i \in H} i$.

We first discuss the role of the empty set. Assume that $V(\emptyset) \neq H$. Since $\emptyset \in \mathcal{U}$ it follows that $V(\emptyset) = H_1 \neq H$. Hence $V(\emptyset) = V(j)$ for any $j \in H_2 \neq \emptyset$. By the above observation we know that $j = \max_{i \in H} i$.

Assume that $R \subseteq H$ is bounded, $|R| \geq 2$ with $y := \min_{i \in R} i$. Because the dimension of the violator space is 1 we have $V(R) = V(x)$ for some $x \in R$. Now for all $x > y$, we have $y \in V(x)$, hence $V(R) = V(y)$.

Let $R \subseteq H$ be unbounded. By Lemma 6.7.3, $V(R) = H_1 = V(i)$ for any $i \in R$. In particular $V(R) = V(\min_{i \in R} i)$. □

Theorem 6.7.5. *Every violator space $(H, V, \mathcal{U})$ of dimension 1 with $|H| = n$, $\emptyset \in \mathcal{U}$, is isomorphic to an instance of smallest number with repetitions, i.e., those are the only violator spaces of dimension* 1 *that exist.*

Note that the smallest number with repetition violator space can always be defined as a violator space without minus infinity. Hence if the dimension is 1, every violator space is homeomorphic to one without minus infinity.

Proof. Let $(H, V, \mathcal{U})$ be a violator space with $|H| = n$ and let H_1, H_2 as in Lemma 6.7.3.

The following holds for all $i \neq j \in H_1$.

1. If $V(i) \neq V(j)$ then $i \in V(j)$ or $j \in V(i)$.
2. $V(i) \subseteq V(j)$ or $V(j) \subseteq V(i)$.
3. $V(i) \cap H_2 = \emptyset$.

For the first part assume that $i \notin V(j)$ and $j \notin V(i)$. Then by locality $V(i, j) = V(i) = V(j)$.

For the second part assume that there exists $k \neq l$ such that $k \in V(i) \setminus V(j)$ and $l \in V(j) \setminus V(i)$. We consider $V(i, j, k, l)$. Since $\delta = 1$ we have that $V(i, j, k, l) = V(m)$ for some $m \in \{i, j, k, l\}$. By consistency we know $V(i, j, k, l) \neq V(i)$ since $k \in V(i)$. Similarly $V(i, j, k, l) \neq V(j)$.

Therefore w.l.o.g. assume that $V(i,j,k,l) = V(k)$. Then $j \notin V(k)$ and $k \notin V(j)$ and hence by the first part $V(j) = V(k) = V(i,j,k,l)$, which is a contradiction.

For the third part assume for contradiction that there exists $k \in H_2$ such that $k \in V(i)$. Since $\{i\}$ is bounded, by monotonicity $\{i,k\}$ is bounded. Hence $\{i,k\}$ must have a bounded basis of size 1, therefore $k \in V(i,k) = V(i)$, which is a contradiction.

We now construct a mapping $f : H \to [n]$, such that $j \in V(i)$ if and only if $f(j) < f(i)$. By Lemma 6.7.4 this concludes the proof.

We construct a sequence of pairwise disjoint nonempty sets

$$V_1, V_2, \ldots V_m \subseteq H_1, m \leq n, V_1 \cup \cdots \cup V_m = H_1$$

such that the following holds for all $i \in [m]$: For all $x \in V_i$ we have $V(x) = V_1 \cup \cdots \cup V_{i-1}$. Moreover we set $V_{m+1} = H_2$. By Lemma 6.7.3 we know that for all $x \in V_{m+1}$ we have $V(x) = H_1 = V_1 \cup \cdots \cup V_m$. By setting $f^{-1}(i) = V_i$ for all $i \in [m+1]$, this is one instance of minimum number with repetitions violator space.

Suppose that for some $i \geq 1$ we have constructed $V_1, \ldots, V_{i-1}$ and $H_1 \setminus (V_1, \ldots, V_{i-1}) \neq \emptyset$. Let V_i be the subset of $H_1 \setminus (V_1, \ldots, V_{i-1})$ with inclusion-minimal violator sets, i.e., $x \in H_1 \setminus (V_1, \ldots, V_{i-1})$ is in V_i if and only if there exists no $y \in H \setminus (V_1, \ldots, V_{i-1})$ such that $V(y) \subsetneq V(x)$. Then obviously V_i is nonempty. We need to show that for such x, $V(x) = V_1 \cup \cdots \cup V_{i-1}$. By condition 3 we know that $V(x) \subseteq H_1$, hence we only need to consider H_1. Let $y \in V(x) \subseteq H_1$. Since $y \notin V(y)$ condition 2 implies that $V(y) \subsetneq V(x)$, hence $y \in V_1 \cup \cdots \cup V_{i-1}$. Now let $y \in H_1 \setminus V(x)$. If $x \notin V(y)$ then by condition 1. it follows that $V(x) = V(y)$ and hence by definition of V_i, $y \in V_i$. Otherwise $x \in V(y)$ and therefore by condition 2. $V(x) \subsetneq V(y)$. It follows that $y \notin V_1 \cup \cdots \cup V_{i-1}$. □

6.8 Discussion and Open Questions

In this chapter we have given two upper bounds for the expected number of violators after removal. The first bound that holds for consistent spaces (Theorem 6.3.4) gives us a bound of $O(\frac{n}{r} \cdot (\delta \ln r + k))$, whereas the second bound for violator spaces is $O(\frac{n}{r} \cdot \delta^{2k+1})$ (Theorem 6.4.7). For both cases we argue that with our methods, no better bounds can be obtained. The main open question is therefore whether and when those bounds are tight. In the setting of violator spaces the best known lower bound is $\Theta(\frac{n}{r} \cdot (\delta + k))$, hence it is possible that neither the multiplicative $\ln r$, nor the δ^{2k} increase happens.

Bibliography

[1] N. Amenta and G. Ziegler. Deformed products and maximal shadows of polytopes. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 57–90. American Mathematical Society, 1999.

[2] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *Siam Journal on Computing*, 9:827–845, 1980.

[3] A. Björner, M. L. Vergnas, B. Sturmfels, N. White, and G. Ziegler. *Oriented Matroids*. Cambridge University Press, 1993.

[4] Y. Brise and B. Gärtner. Clarkson's algorithm for violator spaces. *Computational Geometry*, 44(2):70 – 81, 2011. Special issue of selected papers from the 21st Annual Canadian Conference on Computational Geometry.

[5] M. C. Campi and S. Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM J. Optim.*, 19:1211–1230, 2008.

[6] M. C. Campi and S. Garatti. A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality. *J. Optim. Theory Appl.*, 148:257–280, 2011.

[7] T. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996.

[8] V. Chvatal. *Linear Programming*. W.H. Freeman and company, 1980.

[9] K. Clarkson and P. Shor. Applications of random sampling in computational geometry, ii. *Discrete Comput. Geom.*, 4:387–421, 1989.

[10] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 695–702, 1994.

[11] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the Association for Computing Machinery*, 42:488–499, 1995.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA., 1990.

[13] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[14] J. H. Dulá, R. V. Helgason, and N. Venugopal. An algorithm for identifying the frame of a pointed finite conical hull. *INFORMS J. Comput.*, 10(3):323–330, 1998.

[15] K. Fischer and B. Gärtner. The smallest enclosing ball of balls: Combinatorial structure and algorithms. *International Journal of Computational Geometry and Applications (IJCGA)*, 14(4–5):341–387, 2004.

[16] K. Fukuda. Lecture: Introduction to optimization. https://www.inf.ethz.ch/personal/fukudak/lect/opt2011/, 2011.

[17] K. Fukuda. Walking on the arrangement, not on the feasible region, 2011. http://helper.ipam.ucla.edu/publications/sm2011/sm2011_9630.pdf.

[18] K. Fukuda. Lecture: Polyhedral computation. http://www-oldurls.inf.ethz.ch/personal/fukudak/lect/pclect/notes2016/, 2016.

[19] K. Fukuda, B. Gärtner, and M. Szedlák. Combinatorial redundancy detection. *Annals of Operations Research*, pages 1–19, 2016.

[20] K. Fukuda and M. Szedlák. Redundancies in linear systems with two variables per inequality. *CoRR*, abs/1610.02820, 2016.

[21] K. Fukuda and T. Terlaky. Linear complementarity and oriented matroids. *Journal of the Operations Research Society of Japan*, 35:45–61, 1992.

[22] K. Fukuda and T. Terlaky. Criss-cross methods: A fresh view on pivot algorithms. *Mathematical Programming*, 79:369–395, 1997.

[23] B. Gärtner. Sampling with removal in LP-type problems. *Journal of Computational Geometry*, 6(2):93–112, 2015.

[24] B. Gärtner, C. Helbing, Y. Ota, and T. Takahashi. Large shadows for sparse inequalities. *arXiv:1308.2495*, 2013.

[25] B. Gärtner, J. Lengler, and M. Szedlák. Random Sampling with Removal. In S. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[26] B. Gärtner, J. Matoušek, L. Rüst, and P. Škovroň. Violator spaces: Structure and algorithms. *Discrete Appl. Math.*, 156(11):2124–2141, June 2008.

[27] B. Gärtner, W. D. Morris, Jr., and L. Rüst. Unique sink orientations of grids. In *Proc. 11th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 3509 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 2005.

[28] B. Gärtner and E. Welzl. A simple sampling lemma: Analysis and applications in geometric optimization. *Discrete & Computational Geometry*, 25(4):569–590, 2001.

[29] D. S. Hochbaum and J. Naor. Simple and fast algorithm for linear and integer programs with two variables per inequality. *Siam Journal on Computing*, 23:1179–1192, 1994.

[30] G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Math. Prog. (Ser. B)*, 79:217–234, 1997.

[31] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[32] L. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 244:1093–1096, 1979. (Translated in Sovjet Mathematics Doklady 20, 191-194, 1979).

[33] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

[34] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[35] J. Matoušek. Removing degeneracy in LP-type problems revisited. *Discrete & Computational Geometry*, 42(4):517–526, 2009.

[36] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17:179–184, 1970.

[37] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *Siam Journal on Computing*, 12:347–353, 1983.

[38] T. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating extreme points in higher dimensions. In E. Mayer and C. Puech, editors, *STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 900, pages 562–570. Springer-Verlag, 1995.

[39] C. Roos. An exponential example for Terlaky's pivoting rule for the criss-cross simplex method. *Mathematical Programming*, 46:79–84, 1990.

[40] A. Schrijver. *Theory of linear and integer programming.* John Wiley, New York, 1986.

[41] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '92, pages 569–579, London, UK, UK, 1992. Springer-Verlag.

[42] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the Association for Computing Machinery*, 28:769–679, 1981.

[43] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 547–555, 2000.

[44] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34, 1986.

[45] T. Terlaky. A finite criss-cross method for the oriented matroids. *Journal of Combinatorial Theory Series B*, 42:319–327, 1987.

[46] P. Škovroň. *Abstract Models of Optimization Problems*. PhD thesis, Charles University in Prague, 2007.

[47] Z. Wang. A finite conformal-elimination free algorithm over oriented matroid programming. *Chinese Annals of Math.*, 8B:120–125, 1987.

[48] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.

[49] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.