2016

Exploring the Power of Rescaling

Dan Li
Lehigh University

Exploring the Power of Rescaling

by

Dan Li

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Industrial Engineering

Lehigh University

January 2016

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Dissertation Advisor

Committee Members:

_____
Tamás Terlaky, Committee Chair

_____
Robert H. Storer

_____
Brian D. Davison

_____
Javier Peña

Acknowledgements

First of all, I am deeply grateful to my Ph.D. advisor Dr. Tamás Terlaky. He is a great teacher, researcher, and person. He has very broad knowledge of the subjects and research insights. He is always full of passion and willing to impart what his knowledge and wisdom to me unreservedly. He is always patient even when I ask very fundamental questions and discuss details with him. Learning from him and watching the way he approaches problems is an invaluable experience. His strict requirements inspire me all along. He always supports me even in my most difficult time. It has been a privilege to do research with him. I am grateful to my thesis committee members professor Robert Storer, Brian Davison, Javier Peña, and Alexandre d'Aspremont, for taking the time to provide insightful comments and constructive advice.

I would like to thank all the Lehigh ISE professors, especially Ted Ralphs, Larry Snyder, Imre Pólik, George Wilson, and Jitamitra Desai. I have learned a great deal from their wonderful lectures. Working as TA for Tamás Terlaky, Frank Curtis, Ted Ralphs, Larry Snyder, and Stuart Paxton has been great experience for me to practice not only teaching but also communicating skills. I am also thankful to Rita Frey, Kathy Rambo, and Brianne Lisk who were always there to guide us smoothly through all administration problems.

I am thankful to all my dear friends here who make me feel home at Lehigh. The time I have spent together with them will always be a treasured memory for me. My heart is full of gratitude for my parents, Kai Wang and Yunfei Li. I cannot thank enough to their love and consistent support all these years. Finally, I thank my husband Fenglin

Yan, who is always being there for me, and my son Eric Yan, a little angel who brings all the sunshine to my life.

Contents

List of Tables

List of Figures

List of Symbols

Δ_n	The unit simplex in $\mathbb{R}^n$, page 11.
$\mathbb{S}^{m-1}$	The unit shpere in $\mathbb{R}^m$, page 21.
$\mathcal{F}$	The feasible cone (region) of the perceptron problem, page 20.
$\mathcal{N}_A$	The null space of the matrix A, page 71.
$\mathcal{R}_A$	The row space of the matrix A, page 71.
$\text{conv}(A)$	The convex hull of the points a_i, the column vectors of matrix A, page 6.
ρ	The radius of the largest inscribed ball, page 10.
${}^p\rho$	ρ for the perceptron problem, page 10.
${}^v\rho$	ρ for the von Neumann problem, page 10.
e	The vector of all ones, page 6.
I	Identity matrix, page 17.
i, j, s	Indices for vectors or matrices.
k	Iteration counter. The notations with k as their superscript means the corresponding value in the current iteration.
L	The bit-length (size) of an LO instance with integer data, page 7.

m, n Problem dimension.

P_A, Q_A The orthogonal projection matrix of $\mathbb{R}^n$ onto $\mathcal{N}_A$ and $\mathcal{R}_A$, respectively, page 71.

$x(y)$ $\underset{x \in \Delta_n}{\operatorname{argmin}} \langle A^T y, x \rangle$, page 11.

$x^{\mathcal{N}}, x^{\mathcal{R}}$ The orthogonal decomposition of vector x in the spaces $\mathcal{N}_A$ and $\mathcal{R}_A$, respectively, page 71.

$\text{Vol}(\mathcal{S})$ The volume of a measurable set $\mathcal{S}$, page 21.

General Notations

Lower-case Greek letters Scalars, such as ϵ, α, and θ.

Lower-case Roman letters Vectors unless otherwise specified, such as x, y, b, u, and v. They are corresponding to normalized vectors if they are "over-lined", such as $\bar{y}$..

Notations with $'$ The corresponding values after a rescaling step.

Upper-case Roman letters Matrices, for example A represents the coefficient matrix of either the perceptron or the von Neumann problem.

List of Algorithms

Abstract

The goal of our research is a comprehensive exploration of the power of rescaling to improve the efficiency of various algorithms for linear optimization and related problems. Linear optimization and linear feasibility problems arguably yield the fundamental problems of optimization. Advances in solving these problems impact the core of optimization theory, and consequently its practical applications. The development and analysis of solution methods for linear optimization is one of the major topics in optimization research. Although the polynomial time ellipsoid method has excellent theoretical properties, however it turned out to be inefficient in practice. Still today, in spite of the dominance of interior point methods, various algorithms, such as perceptron algorithms, rescaling perceptron algorithms, von Neumann algorithms, Chubanov's method, and linear optimization related problems, such as the colorful feasibility problem – whose complexity status is still undecided – are studied.

Motivated by the successful application of a rescaling principle on the perceptron algorithm, our research aims to explore the power of rescaling on other algorithms too, and improve their computational complexity. We focus on algorithms for solving linear feasibility and related problems, whose complexity depend on a quantity ρ, which is a condition number for measuring the distance to the feasibility or infeasibility of the problem. These algorithms include the von Neumann algorithm and the perceptron algorithm. First, we discuss the close duality relationship between the perceptron and the von Neumann algorithms. This observation allows us to transit one algorithm as a variant of the other, as well as we can transit their complexity results. The discovery of

this duality not only provides a profound insight into both of the algorithms, but also results in new variants of the algorithms.

Based on this duality relationship, we propose a deterministic rescaling von Neumann algorithm. It computationally outperforms the original von Neumann algorithm. Though its complexity has not been proved yet, we construct a von Neumann example which shows that the rescaling steps cannot keep the quantity ρ increasing monotonically. Showing a monotonic increase of ρ is a common technique used to prove the complexity of rescaling algorithms. Therefore, this von Neumann example actually shows that another proof method needs to be discovered in order to obtain the complexity of this deterministic rescaling von Neumann algorithm. Furthermore, this von Neumann example serves as the foundation of a perceptron example, which verifies that ρ is not always increasing after one rescaling step in the polynomial time deterministic rescaling perceptron algorithm either.

After that, we adapt the idea of Chubanov's method to our rescaling frame and develop a polynomial-time column-wise rescaling von Neumann algorithm. Chubanov recently proposed a simple polynomial-time algorithm for solving homogeneous linear systems with positive variables. The Basic Procedure of Chubanov's method can either find a feasible solution, or identify an upper bound for at least one coordinate of any feasible solution. The column-wise rescaling von Neumann algorithm combines the Basic Procedure with column-wise rescaling to identify zero coordinates in all feasible solutions and remove the corresponding columns from the coefficient matrix. This is the first variant of the von Neumann algorithm with polynomial-time complexity. Furthermore, compared with the original von Neumann algorithm which returns an approximate solution, this rescaling variant guarantees an exact solution for feasible problems.

Finally, we develop the methodology of higher order rescaling and propose a higher-order perceptron algorithm. We implement the perceptron improvement phase by utilizing parallel processors. Therefore, in a multi-core environment we may obtain several rescaling vectors without extra wall-clock time. Once we use these rescaling vectors in

a single higher-order rescaling step, better rescaling rates may be expected and thus computational efficiency is improved.

Publications

- Dan Li and Tamás Terlaky. The Duality Between the Perceptron and the von Neumann Algorithms. *In Modeling and Optimization: Theory and Applications*, volume 62, pp. 113 - 136. 2013.

- Dan Li, Cornelis Roos and Tamás Terlaky. A Polynomial Column-Whised Rescaling von Neumann Algorithm. Submitted to *Mathematical Programming*.

- Dan Li and Tamás Terlaky. An Example with Decreasing Largest Inscribed Ball for Deterministic Rescaled Algorithms. Submitted to *INFORMS Journal on Computing*.

- Dan Li and Tamás Terlaky. A Higher-Order Rescaling Perceptron Algorithm. Working paper.

Chapter 1

Introduction

This chapter reviews basic concepts of linear optimization and linear feasibility problems in Section 1.1: problem equivalence, complexity theory, related algorithms, and condition numbers. Section 1.2 introduces the details of the algorithms which we focus on. Our motivation is presented in Section 1.3 and the structure of the thesis is given in Section 1.4.

1.1 Linear Optimization and Linear Feasibility Problem

It is well known that Linear Optimization (LO) and Linear Feasibility Problems (LFPs) are the fundamental problems of optimization. We start with important concepts considered frequently during the study of these problems.

1.1.1 Problem Equivalence

LO is the problem of minimizing or maximizing a linear objective function subject to a system of linear inequalities and equations. The goal of the LFP is to find a feasible solution to a linear inequality system. Considering an LO problem with zero as its objective function, then we have that every feasible solution is optimal. From this point of view the LFP is a special case of LO. On the other hand, it is well known [4, 42] that

by the LO duality theorem, any LO problem can be transformed to an equivalent LFP.

Given an LO problem, it can be transformed to the following canonical form [42]:

$$\max \{c^T x \ : \ Ax = b, \ x \geq 0\}, \tag{1.1}$$

where matrix $A \in \mathbb{R}^{m \times n}$, vectors $c, x \in \mathbb{R}^n$, and vector $b \in \mathbb{R}^m$. Consider (1.1) as a primal problem, then its dual problem is given by

$$\min \{b^T y \ : \ A^T y \geq c\}. \tag{1.2}$$

There are two important duality theorems [4]. The weak duality theorem shows that for problems in the forms of (1.1) and (1.2), $b^T y \geq c^T x$ holds, i.e., the objective function value of the primal problem at any feasible solution is always less than or equal to the objective function value of the dual problem at any feasible solution. If x^* and y^* are optimal solutions for (1.1) and (1.2) respectively, then by the strong duality theorem we have $b^T y^* = c^T x^*$. Consider the following inequalities system:

$$\begin{aligned} Ax &\geq b, \quad x \geq 0, \\ -Ax &\geq -b, \\ A^T y &\geq c, \\ c^T x - b^T y &\geq 0. \end{aligned} \tag{1.3}$$

Given any feasible solution (x, y) to the linear inequality system (1.3), the first two inequalities guarantee that x and y are feasible to (1.1) and (1.2), respectively. By the weak duality theorem, the third inequality can only be satisfied with equality, which implies that x is an optimal solution to (1.1) and y is an optimal solution to (1.2). Thus, any LO problem (1.1) can be solved by solving the corresponding LFP (1.3). In other words, LO problems and LFPs are equivalent to one another and they are equivalently solvable.

LFPs can be written in various forms. First we consider the form [1]

$$A^T y > 0, \tag{1.4}$$

where $A \in \mathbb{R}^{m\times n}$ with its column vectors $a_1, a_2, \ldots, a_n \in \mathbb{R}^m$ and $y \in \mathbb{R}^m$. We may assume without loss of generality that $\|a_i\|_2 = 1$ for all $i = 1, 2, \ldots, n$ and rank(A)=m. This assumption is not changing the feasibility or infeasibility of problem (1.4). An LFP can also have a non-zero right hand side constant $c \in \mathbb{R}^n$ in the form

$$A^T y \geq c. \tag{1.5}$$

Since each equation is equivalent to two inequalities, any linear inequality and equation system can be transformed to the form of problem (1.5). We also consider LFPs in the form

$$\begin{aligned} Ax &= 0 \\ e^T x &= 1 \\ x &\geq 0, \end{aligned} \tag{1.6}$$

where $x \in \mathbb{R}^n$ and $e \in \mathbb{R}^n$ is the vector of all ones. For convenience of our discussion, without loss of generality [2], we may assume that matrix A has the same properties as matrix A in problem (1.4). Observe that problem (1.6) yields a standard form of LFPs with a convexity constraint [14]. Let conv(A) represent the convex hull of the points a_i. If the origin $0 \in$ conv(A), then problem (1.6) is feasible and can be considered as a weighted center problem [14], i.e., the problem of assigning nonnegative weights x_i to the points a_i so that their weighted center of gravity is the origin 0.

[1] For convenience matrix A will be universally used as the coefficient matrix of the actually considered LFP or LO problems. We will make clear when any relationship exists between the coefficient matrices.

1.1.2 Complexity Theory for Linear Optimization

The purpose of complexity theory is to give an upper bound for the number of arithmetic operations needed to solve a given problem, and this way one can quantify how efficient an algorithm is. The complexity estimates are worst case bounds, thus they do not always indicate the actual behavior of the algorithms for particular problem instances. However, complexity theory is a main instigator for algorithmic research [44].

In complexity theory, the word "problem" is used to represent the form of general questions without concrete values. If all the parameters of a problem are set to certain values, we define it an instance of the problem. That is, a problem is a collection of instances [4, 44]. Given any LO instance, an LO algorithm is able to determine whether the constraints for the instances are consistent, i.e., whether the instance is feasible, and compute an optimal solution if it exists.

A measure of the difficulty of an LO instance, when given by integer data, is its "bit-length", which is usually denoted by L. It is also referred to as the size of the instance. Given an LO instance, L is the number of bits that are necessary to represent the instance in the binary number system, i.e., the size of the input data, or input length. We assume that all the data of the given instances are integer. Consider a problem in the form of (1.5), then the size of an instance is

$$L = \sum_{i=1}^{m}\sum_{j=1}^{n}\lceil \log_2(|\alpha_{ij}|+1)\rceil + \sum_{i=1}^{n}\lceil \log_2(|\gamma_i|+1)\rceil + 1 + mn + n,$$

where $A = (\alpha_{ij})_{m\times n}$ and $c = (\gamma_i)_n$. Let $T(L)$ be the number of arithmetic operations of an algorithm in the worst case over all instances of size L. An algorithm is polynomial-time if $T(L)$ is bounded by some function that is polynomial in m, n and L.

1.1.3 Linear Optimization Algorithms

Since the discovery of the simplex method [4, 44] in 1947, the development and analysis of solution methods for Linear Optimization (LO) are in the center of optimization

research. Still today, the simplex method is one of the most powerful algorithms for solving LO problems in computational practice. However, in 1972, Klee and Minty [33] showed that the simplex method may take an exponential number of iterations, i.e., the performance of the simplex method in the worst case is poor. Thus, the simplex method is not a polynomial time algorithm.

The era of polynomial time algorithms for LO was launched in 1979 by Khachiyan [31] who published the first polynomial time algorithm – the ellipsoid method – to solve LO problems. Although the polynomial time ellipsoid method has excellent theoretical properties, it turned out to be inefficient in computational practice [31, 32]. Regardless of its weak performance in practice, Khachiyan's algorithm inspired a flood of research in LO and established the foundations of the polynomial-time solvability of LO problems.

In 1984, Karmarkar [30] introduced a new polynomial time algorithm which belongs to the class of Interior Point Methods (IPMs). IPMs are considered to be the first family of algorithms which are both theoretically and computationally efficient for solving LO problems. Since Karmarkar's remarkable work, the study of IPMs as polynomial time algorithms have dominated the optimization literature, in particular LO, in the past quarter century [4, 44, 42]. In spite of the dominance of IPMs, various algorithms, such as variants of the perceptron algorithm (PA) [5, 6, 45] and the von Neumann algorithm (vNA) [12, 14] were studied.

The PA [43] was originally invented in the field of machine learning. It is used to solve data classification problems by learning a linear threshold function. It is designed to solve LFPs in the form (1.4). In its original form, the PA is not a polynomial-time algorithm. Two rescaling variants were proposed with the goal to speed up the PA. Dunagan and Vempala [17] proposed a randomized rescaling PA. Its complexity is polynomial with high probability. Recently, Peña and Soheili [37] proposed a deterministic rescaling PA. It guarantees that the algorithm always stops in polynomial time, however, the total complexity is not as good as the one of the stochastic version and that of IPMs. For these two variants of rescaling PAs, the authors used different terminologies to describe the

rescaling process. In order to keep consistent terminology we choose to use "rescaling", regardless what the original term the authors used.

The vNA was first published by Dantzig [12, 14], and the vNA can be interpreted as a special case of the Frank-Wolfe algorithm, which is an iterative method for solving linearly constrained convex optimization problems [20, 3]. The complexity of finding an approximate solution, in the worst case, could require exponentially many arithmetic operations. Therefore, the vNA is not a polynomial-time algorithm. Although, several variants of the vNA are proposed by Gonçalves, Storer, and Gondzio [28], none of them is proved to have polynomial complexity.

Chubanov [9, 10, 11] has recently proposed a novel polynomial-time algorithm for solving homogeneous linear systems with positive variables. It is a divide-and-conquer algorithm which can be considered as a generalization of the classic relaxation method [1, 36]. It projects the current solution not only onto the half-spaces corresponding to original constraints, but also onto those corresponding to induced inequalities. The so-called elementary procedure or Basic Procedure (BP) is the core of the method. The BP generates induced inequalities if it neither finds a solution, nor provides an evidence of infeasibility. One of the advantages of Chubanov's method is its polynomial-time complexity. For certain problems, it runs in strongly polynomial time.

More details about the various algorithms will be introduced in Section 1.2.

1.1.4 Condition Number for Linear Feasibility Problems

For solving LFPs, we focus on variants of the perceptron algorithm and variants of the von Neumann algorithm. They consider LFPs in the standard forms (1.4) and (1.6), respectively. Therefore, we also call problem (3.1) the perceptron problem and problem (3.2) the von Neumann problem. Both of these algorithms aim to find a feasible solution for an LFP. They either deliver a feasible solution, or provide an evidence of infeasibility. A common feature of these algorithms is that their iteration complexity depends on a condition number ρ [5, 19, 32]. Since LO problems and LFPs are equivalent to solve,

the concept of condition number for LO problems also applies to LFPs. A variety of condition numbers for LFPs have been defined, e.g., by Renegar [39, 40], Epelman and Freund [18], Epelman and Vera [22, 23], and Cheung and Cucker [8]. As one of the condition numbers, ρ quantifies how far the given instance is from the boundary of infeasibility and feasibility. Given an LFP, we define ρ in general terms as follows.

Definition 1.1.1. *Given an LFP, if the problem is feasible, ρ is defined as the radius of the largest inscribed ball contained in the feasible region. Otherwise, ρ measures the distance to feasibility.*

For different algorithms and problem forms, ρ may have slightly different definitions. We will discuss them in details in Section 2.2.2. In order to distinguish ρ in different algorithms, we use ${}^{p}\rho$ to represent ρ in the perceptron algorithm and ${}^{v}\rho$ for the von Neumann algorithm. If the problem is feasible (infeasible), this quantity ρ indicates how far the constraints can be shifted or rotated so that the problem becomes infeasible (feasible). The larger ρ is, the harder to turn a feasible problem to infeasible, or vice versa. Therefore, ρ can be seen as a measure of the robustness of feasibility or infeasibility.

In addition, as one of the condition measures of LFPs, the quantity ρ effects the convergence of algorithms directly. Freund and Vera [23] tried to build a relationship between the geometry of a convex feasible set and the computational complexity of an algorithm applied to the problem. Quantity ρ may be considered as a measure of the goodness of problem geometry. They showed that the problems with favorable geometry have better computational complexity. The complexity results of the perceptron and von Neumann algorithms confirm this fact: the larger ρ is, the better the algorithm behaves. The details about the complexity of algorithms are also discussed in Section 1.2.

1.2 Elementary Algorithms

Among the algorithms for solving LFPs, there are several of them which have a common feature: each iteration involves only simple computations. Here, simple computation

means that unlike conventional Newton-based algorithms, there are no more complicated operations than matrix-vector multiplication in each iteration. This class of algorithms is referred to as elementary algorithms. The PA and vNA are elementary algorithms.

1.2.1 Perceptron Algorithms

In this section, we introduce the Classical PA and a modified version with their complexity results.

1.2.1.1 The Classical Perceptron Algorithm

The Classical PA used in machine learning [45] is designed to solve the following data classification problem: Given a set of points in the m-dimensional space. Each point is labeled as positive or negative. The problem is to find a separating hyperplane, which separates all positive points from the negative ones. By some transformations [45] those problems lead to LFPs in the form of (1.4) that consists of n constraints in dimension m. The Classical PA assumes that problem (1.4) is feasible. It starts from the origin and performs a classical perceptron update at each iteration. This update step tries to find a violated constraint and moves the current iterate y by one unit perpendicularly towards the violated constraint. The algorithm terminates once the iterate y is in the feasible region and y is a feasible solution.

Let Δ_n be the unit simplex, i.e., $\Delta_n = \{x : x \geq 0, \|x\|_1 = 1\}$. For $y \in \mathbb{R}^m$, define $x(y) = \underset{x\in\Delta_n}{\operatorname{argmin}}\langle A^T y, x\rangle$. Thus, we have $Ax(y) = a_s$ if and only if $a_s^T y = \min\limits_{i=1,\cdots,n} a_i^T y$. Observe that a_s and so $x(y)$ are not necessarily unique. Now we are ready to describe the Classical PA.

Remember that we have assumed that all a_i vectors have unit length. The PA has the following complexity result.

Theorem 1.2.1. *[35] Assume that the LFP (1.4) is strictly feasible. Then after at most*

$$\frac{1}{^p\rho^2}$$

Algorithm 1.1 The Classical Perceptron Algorithm

1: **Initialization**: Let y^0 be the all-zero vector. $k = 0$.
2: **while** True **do**
3: $Ax(y^k) = a_s$.
4: **if** $a_s^T y^k > 0$ **then**
5: STOP, and return y^k.
6: **else**

$$\begin{aligned} y^{k+1} &= y^k + a_s, \\ k &= k + 1. \end{aligned}$$

7: **end if**
8: **end while**

iterations, the PA terminates with a feasible solution.

Note that this theorem assumes that problem (1.4) is strictly feasible. According to the general definition of the quantity ρ in Definition 1.1.1, ${}^p\rho$ refers to the radius of the largest inscribed ball contained in the feasible region and centered on the unit sphere. In the worst case, radius ${}^p\rho$ can be exponentially small in the input length L. Thus, the iteration complexity of the Classical PA is not polynomial. In the dual view, ${}^p\rho$ is called the wiggle room for a feasible solution y^* [5], which indicates the minimum distance of any column a_i of matrix A to the hyperplane $a_i^T y^* = 0$. It can be calculated by ${}^p\rho = \min_i \frac{|a_i^T y^*|}{\|a_i\|\|y^*\|}$. Theorem 1.2.1 shows that if an LFP (1.4) has a non-zero wiggle room, then the PA produces a feasible solution, and the smaller the ball inside the feasible region is, the more iterations the PA needs. To improve the geometry of LFPs, and improve the complexity of PAs, rescaling is applied.

1.2.1.2 A Modified Perceptron Algorithm

A modified version of the PA was presented by Blum, et al. [6]. This algorithm returns a nearly feasible solution to problem (1.4). A nearly feasible solution means that some constraints in problem (1.4) might be violated but none of them is violated much. Formally, a nearly feasibility solution is defined as follows.

Definition 1.2.2. *A vector y is a nearly feasible solution to problem (1.4) if $a_i^T y \geq -\sigma\|y\|$ for all $i = 1, \ldots, n$, where a_i is the ith column of matrix A and σ is a small positive number.*

The Modified PA is stated in Algorithm 1.2. It always starts from a random unit

Algorithm 1.2 The Modified Perceptron Algorithm

Initialization: Choose any random unit vector y^0 in $\mathbb{R}^m$. Let $k = 0$.
while $k \leq \lceil \frac{\ln m}{\sigma^2} \rceil$ **do**
    $Ax(y^k) = a_s^k$.
    if $(a_s^k)^T y^k \geq -\sigma\|y^k\|$ **then**
        STOP and return y^k.
    else

$$\begin{aligned} y^{k+1} &= y^k - ((a_s^k)^T y^k) a_s^k, \\ k &= k + 1. \end{aligned}$$

        if $y^{k+1} = 0$ **then**
            Go back to **Initialization** and restart.
        end if
    end if
end while
if the algorithm does not stop at this point **then**
    Go back to **Initialization** and restart the algorithm.
end if

vector, which brings a nondeterministic factor to its complexity result.

Theorem 1.2.3. *[6] Assume that problem (1.4) is feasible. With high probability, after at most*

$$O\left(\frac{1}{\sigma^2}\log(m)\right)$$

iterations, the Modified PA returns a nearly feasible solution. "High probability" is defined as the probability of at least $1 - e^{-m}$.

1.2.2 The von Neumann Algorithm

The vNA was published by Dantzig [12, 14] in 1991. It is an algorithm for solving LFPs in the form of (1.6). The vNA can also be interpreted as a special case of the Frank-Wolfe algorithm [20, 3], which is an iterative method for solving linearly constrained convex optimization problems. Unlike the perceptron algorithm, the vNA gives an approximate solution in a finite number of iterations. Thus, before presenting the vNA, we need to define what an ϵ-*approximate solution* (or ϵ-solution for short reference) of problem (1.6) is.

Definition 1.2.4. *Given $0 < \epsilon < 1$. An ϵ-solution of problem (1.6) is a solution $x \geq 0$ satisfying $e^T x = 1$ and $\|Ax\| \leq \epsilon$.*

The vNA terminates once it obtains an ϵ-solution. Consequently, this algorithm can be interpreted as an algorithm for solving an optimization problem with minimizing $\|Ax\|$ as its objective function. This interpretation makes the vNA to fall in the framework of the Frank-Wolfe algorithm [20]. Actually, when the Frank-Wolfe algorithm is applied to the problem $\min\{\|Ax\| : e^T x = 1, x \geq 0\}$ with exact line search for calculating the step length at each iteration, the Frank-Wolfe algorithm exactly reduces to the vNA.

In Algorithm 1.3, vector b is also called the residual at the current iterate. We have the following lemma to justify the statement in Step 7.

Lemma 1.2.5. *For any vector b, let $s = \underset{i=1,\ldots,n}{\operatorname{argmin}} a_i^T b$. If $a_s^T b > 0$, then problem (1.6) is infeasible.*

Proof. Let $b \in \mathbb{R}^m$ be an arbitrary vector. Since $s = \underset{i=1,\ldots,n}{\operatorname{argmin}} a_i^T b$ and $a_s^T b > 0$, we have $a_i^T b > 0$ for all $i = 1, \ldots, n$. Thus, all the points $a_i, i = 1, \ldots, n$ lie in the open halfspace $\mathcal{C}$ defined by $\mathcal{C} = \{y | b^T y < 0\}$. By the definition of the convex hull, it is obvious that $\text{conv}(A) \subset \mathcal{C}$. We also have $0 \notin \mathcal{C}$ because $b^T 0 = 0$, therefore $0 \notin \text{conv}(A)$. Thus, problem (1.6) is infeasible and the separating hyperplane $b^T w = 0$ is a certificate for it. □

Algorithm 1.3 The von Neumann Algorithm

Initialization:
Choose any $x^0 \geq 0$ with $e^T x^0 = 1$.
Let $b^0 = Ax^0$ and $k = 0$.
while $\|b^k\| \geq \epsilon$ **do**
We have an approximate solution x^k, such that $x^k \geq 0$, $e^T x^k = 1$. Let $\mu_k = \|b^k\|$.
Find the vector a_s which makes the largest angle with the vector b^k:

$$s = \underset{i=1,\ldots,n}{\operatorname{argmin}} \, a_i^T b^k.$$

Let $\nu_k = a_s^T b^k$.
if $\nu_k > 0$ **then**
STOP, problem (1.6) is infeasible.
end if
Let e_s be the unit vector corresponding to index s and let

$$\begin{aligned} \lambda &= \frac{1 - \nu_k}{\mu_k^2 - 2\nu_k + 1}, \\ \mu_{k+1}^2 &= \lambda\nu_k + (1 - \lambda), \\ x^{k+1} &= \lambda x^k + (1 - \lambda)e_s, \\ b^{k+1} &= Ax^{k+1}, \\ k &= k + 1. \end{aligned}$$

end while

Dantzig assumed that problem (1.6) is feasible and derived an upper bound for the number of iterations of the vNA [14] as follows.

Theorem 1.2.6. *[14] Let $\epsilon > 0$ and assume that problem (1.6) is feasible. Then after at most*

$$\left\lceil \frac{1}{\epsilon^2} \right\rceil$$

iterations, the von Neumann Algorithm provides an ϵ-solution for problem (1.6).

Epelman and Freund [19] gave a different complexity analysis, and showed that when problem (1.6) is strictly feasible or strictly infeasible, then the iteration complexity of the vNA is linear in $\ln(1/\epsilon)$ and $1/{}^v\rho^2$.

Theorem 1.2.7. *[19] Let $\epsilon > 0$ and assume that ${}^v\rho > 0$.*

1) If problem (1.6) is strictly feasible, then after at most

$$\left\lceil \frac{2}{{}^v\rho^2} \ln \frac{1}{\epsilon} \right\rceil$$

iterations, the vNA obtains an ϵ-solution of problem (1.6).

2) If problem (1.6) is strictly infeasible, then after at most

$$\left\lfloor \frac{1}{{}^v\rho^2} \right\rfloor$$

iterations, the vNA returns a certificate of infeasibility.

When problem (1.6) is feasible, both the complexity bounds proved by Dantzig (Theorem 1.2.6) and Epelman and Freund (Theorem 1.2.7) are valid for obtaining an ϵ-approximate solution. Neither of them is dominant. When ${}^v\rho$ is large, the complexity bound proved by Epelman and Freund is better. Otherwise, the one by Dantzig is better.

Theoretically the vNA does not provide an exact solution, it only converges to a solution. Dantizig [13] proposed a "bracketing" procedure to identify an exact solution

in finite number of iterations. By applying the vNA to $m+1$ perturbed problems, $m+1$ approximate solutions are generated. A weighted sum of these approximate solutions yields an exact solution to the original unperturbed problem. This requires the solution of an $(m+1) \times (m+1)$ system of linear equations. This procedure has the following complexity.

Theorem 1.2.8. *[13] Assume that problem (1.6) is strictly feasible. Then after at most*

$$\frac{4(m+1)^3}{{}^v\rho^2}$$

iterations, Dantzig's "bracketing" procedure returns an exact feasible solution.

1.2.3 Rescaling Perceptron Algorithms

There are two successful versions of rescaling PAs. Both of them lead to polynomial-time complexity by applying a rescaling procedure. The major difference is whether the polynomial complexity is deterministic.

1.2.3.1 The Stochastic Rescaling Perceptron Algorithm

Dunagan and Vempala [17] proposed a rescaling PA in 2004. In order to seperate it from another one, we call it Stochastic Rescaling PA. It rescales the linear inequality system (1.4) at each step. A *step* of the algorithm consists of three phases. The perceptron phase employs the Classic PA with a restricted number of iterations. The perceptron improvement phase uses a modified version of the algorithm proposed in Section 1.2.1.2 in order to obtain a nearly feasible solution, which is then used in the rescaling phase to widen the cone of feasible solutions. The Rescaling PA is as follows.

Figure 1.1 illustrates a constraint system before and after rescaling. In the rescaling phase, a nearly feasible solution y, found in the perceptron improvement phase, is used to perform a linear transformation on problem (1.4); and consequently, increase ${}^p\rho$. An important result for the rescaling phase is derived in [17].

Algorithm 1.4 The Stochastic Rescaling Perceptron Algorithm

Initialization: Let $B \in \mathbb{R}^{m\times m}$, $B = I$, and $\sigma = \frac{1}{32m}$.
while True **do**
  Phase 1. The Perceptron Phase
    Run *The PA* for at most $\lceil \frac{1}{\sigma^2} \rceil$ iterations, then output y.
    if $A^T y \geq 0$ **then**
      STOP and return By as a feasible solution.
    end if
  Phase 2. The Perceptron Improvement Phase
    Run *The Modified PA*, then output y.
    if $A^T y \geq 0$ **then**
      STOP and return By as a feasible solution.
    end if
  Phase 3. The Rescaling Phase
    Let $\bar{y} = \frac{y}{\|y\|}$. Set $A = (I + \bar{y}\bar{y}^T)A$ and $B = (I + \bar{y}\bar{y}^T)B$.
end while
Output: A point y such that $A^T y \geq 0$ and $y \neq 0$.

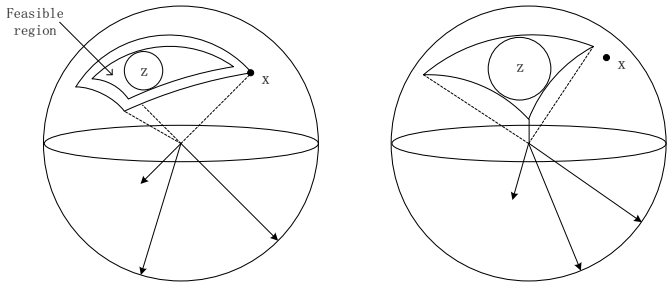


Figure 1.1: Illustration of the impact of rescaling.

Lemma 1.2.9. *[17] Let $\rho, \sigma \leq 1/32m$. Let A' be obtained from A by one iteration of the algorithm (when the problem was not solved). Let ${}^p\rho'$ and ${}^p\rho$ be the radii of A' and A, respectively. Then,*

(a) ${}^p\rho' \geq (1 - \frac{1}{32m} - \frac{1}{512m^2}){}^p\rho$.

(b) With probability at least $\frac{1}{8}$, ${}^p\rho' \geq (1 + \frac{1}{3m}){}^p\rho$.

Lemma 1.2.9 shows the fact that the rescaling phase increases ${}^p\rho$ by at least a fixed factor with probability at least 1/8. Thus, at each iteration, the Rescaling PA either returns a feasible solution to problem (1.4), or with probability at least 1/8 inflates the largest inscribed ball. The iteration complexity of the Rescaling PA is presented in the following theorem.

Theorem 1.2.10. *[17] Assume that problem (1.4) is strictly feasible. Then after at most*

$$O\left(m \ln\left(\frac{1}{{}^p\rho}\right)\right)$$

iterations, with high probability, the Rescaling PA returns a feasible solution.

Now compare the complexity results in Theorem 1.2.1 and Theorem 1.2.10. The iteration complexity of the Classical PA is $O(1/{}^p\rho^2)$, while the computational complexity of the Rescaling PA is $O(m \ln(1/{}^p\rho))$. As ${}^p\rho$ may be as small as 2^{-L}, see e.g. [24], the iteration complexity of the Classical Perceptron Algorihm is $O(2^L)$, while with high probability the complexity of the Rescaling PA is $O(mL)$).

1.2.3.2 The Deterministic Rescaling Perceptron Algorithm

Recently, Peña and Sohèili [38] proposed a Deterministic Rescaling PA, shown in Algorithm 1.5. It consists of an outer loop with two main phases, the perceptron phase and the rescaling phase. Same as the Stochastic Rescaling PA, the perceptron phase also invokes the Classic PA. This phase either finds a feasible solution within a predefined number of perceptron updates (iterations), or identifies one column of matrix A, called

a_j as the rescaling vector. Therefore, this Deterministic Rescaling PA does not need additional phase to find rescaling vectors. This is one of the major differences compared to the Stochastic Rescaling PA.

Algorithm 1.5 The Deterministic Rescaling Perceptron Algorithm

Initialization: Let $B = I$, $N = 6mn^2$.
while True **do**
  Phase 1. Perceptron Phase
  Let $x^0 = 0$, $y^0 = 0$.
  for $k = 0, 1, \cdots, N-1$ **do**
    if $A^T y^k > 0$ **then**
      Stop and return By^k as a feasible solution.
    else
      Let $j \in \{1, \cdots, n\}$ be such that $a_j^T y^k \leq 0$.

$$\begin{aligned} x^{k+1} &= x^k + e_j, \\ y^{k+1} &= y^k + a_j. \end{aligned}$$

    end if
  end for
  Phase 2. Rescaling Phase
  Let $j = \underset{i=1,\cdots,n}{\operatorname{argmax}} \{e_i^T x^N\}$.
  Set $B = B(I - \frac{1}{2} a_j a_j^T)$ and $A = (I - \frac{1}{2} a_j a_j^T)A$.
  Normalize the columns of A.
end while

Lemma 1.2.11. *[38] If the perceptron phase in the Deterministic Rescaling PA does not find a solution to $A^T y > 0$, then the vector a_j in the first step of the rescaling phase satisfies*

$$\{y : A^T y \geq 0\} \subseteq \{y : 0 \leq a_j^T y \leq \frac{1}{\sqrt{6m}} \|y\|\}. \tag{1.7}$$

Geometrically, Lemma 1.2.11 states that if the perceptron phase does not solve the problem, then it identifies a column a_j of A which is nearly perpendicular to the feasible cone $\mathcal{F} = \{y : A^T y \geq 0\}$, which is contained in a narrow band.

The rescaling phase applies the linear transformation

$$A' = \left(I - \frac{1}{2}a_j a_j^T\right) A \tag{1.8}$$

on matrix A at each rescaling step, where a_j is a rescaling vector obtained in the perceptron phase. In order to monitor how the volumn of $\mathcal{F}$ changes after rescaling, the volume of the intersection of $\mathcal{F}$ and the unit sphere is considered because these two volumes changes coincidently and the latter is easier to be calculated. The next lemma shows that the volume of this intersection will increase if utilizing such a_j as the rescaling vector.

Let $\mathbb{S}^{m-1} = \{y \in \mathbb{R}^m : \|y\| = 1\}$ denote the unit shpere. Given a measurable set $\mathcal{S} \in \mathbb{S}^{m-1}$, let $\text{Vol}(\mathcal{S})$ denote its volume in $\mathbb{S}^{m-1}$.

Lemma 1.2.12. *[38] Assume that $\mathcal{F} \subseteq \{y : 0 \leq a_j^T y \leq \frac{1}{\sqrt{6m}}\}$, i.e., (1.7) holds. If A is rescaled by (1.8), then*

$$\text{Vol}(\mathcal{F}' \cap \mathbb{S}^{m-1}) \geq 1.5\text{Vol}(\mathcal{F} \cap \mathbb{S}^{m-1}),$$

where $\mathcal{F}' = \{y : (A')^T y \geq 0\}$ is the feasible cone after rescaling.

Lemma 1.2.13. *[38] Assume $\mathcal{F} \subseteq \mathbb{R}^m$ is a closed convex cone. Then*

$$\text{Vol}(\mathcal{F} \cap \mathbb{S}^{m-1}) \geq \left(\frac{{}^p\rho}{\sqrt{1 + {}^p\rho^2}}\right)^{m-1} \frac{1}{2\sqrt{\pi}}\text{Vol}(\mathbb{S}^{m-1}).$$

Lemma 1.2.12 implies that after each rescaling phase, the quantity $\text{Vol}(\mathcal{F} \cap \mathbb{S}^{m-1})$ increases by a factor of 1.5 or more. Lemma 1.2.13 states that the volume of the inital $\text{Vol}(\mathcal{F} \cap \mathbb{S}^{m-1})$ is bounded below by a factor of ${}^p\rho^{m-1}$. Furthermore, the set $\text{Vol}(\mathcal{F} \cap \mathbb{S}^{m-1})$ is always contained in a hemisphere. Therefore, this algorithm will terminates within finite number of rescaling steps. It has the following complexity result.

Theorem 1.2.14. *Assume that problem (1.4) is strictly feasible. Then after at most*

$$O\left(m \ln\left(\frac{1}{{}^{p}\rho}\right)\right)$$

iterations, the Rescaling PA finds a feasible solution.

Compared to the Stochastic Rescaling PA, the one by Peña and Sohèili has a weaker, but deterministic polynomial complexity. Utilizing a monotonically increasing spherical cap instead of ${}^{p}\rho$ to prove the complexity makes another major difference between these two versions of rescaling PAs.

1.2.4 Chubanov's Method

Chubanov [9, 10, 11] has recently proposed a novel polynomial-time algorithm for solving homogeneous linear systems with positive variables. It is a divide-and-conquer algorithm which can be considered as a generalization of the relaxation method [1, 36]. We refer to this algorithm as Chubanov's Method. The general form of Chubanov's problem is as follows.

$$Ax = b, \quad \hat{A}x \leq \hat{b}, \tag{1.9}$$

where the coefficient matrix A is an $m \times n$ matrix, $\hat{A}$ is an $\hat{m} \times n$ matrix, and all the entries of $A, \hat{A}$ and the right side vector $b, \hat{b}$ are integers. We can assume that matrix A is full rank, i.e., $\text{rank}(A) = m$. Otherwise, the problem can be easily transformed into another one with the same structure and a full rank coefficient matrix without affecting the set of feasible solutions.

Chubanov's Method either finds a feasible solution or determine the problem (1.9) has no integer solutions in polynomial time. Furthermore, if the inequalities take the form $0 \leq x \leq 1$, then the Chubanov's Method runs in strongly polynomial time. The idea of Chubanov's Method is to use new induced inequalities, or sometimes called implicit

inequalities. Those valid induced inequalities are constructed by a procedure called Elementary Procedure or Basic Procedure (BP) [41]. As the core of Chubanov's method, if the BP neither finds a feasible solution, nor identifies the infeasibility of the system, then the BP identifies an upper bound for at least one coordinate of any possible feasible solution by projecting the current solution onto induced inequlites. According to this upper bound for the identified coordinates, the corresponding columns of the coefficient matrix are multiplied by a scalar. Therefore, Chubanov's method can be also considered as a rescaling procedure.

For our purpose, we consider the Chubanov's Method on the following problem form

$$Ax = 0, \quad x > 0, \tag{1.10}$$

where A is an $m \times n$ full rank matrix. Note that any solution of problem (1.10) can be transferred to a feasible solution of the von Neumann problem (1.6) by the normalization $x := \frac{x}{e^T x}$. However, only strictly positive solutions of problem (1.6) are also solutions of problem (1.10). For problem (1.10), if the problem is not solved, then the BP will generate an induced inequality: an upper bound for some coordinate x_i if such a solution x exists within $4n^3$ iterations, which makes the BP strongly polynomial. After applying this new inequality, problem (1.10) is reduced to another one similar to itself but with different A and then the BP is called again. The Chubanov's Method has following complexity result.

Theorem 1.2.15. *[10] For problem (1.10), after at most $O(nL)$ iterations, where L denotes the bit-length size of A, Chubanov's Method either finds a solution or identifies its infeasibility.*

Note that if problem (1.10) is modified to allow some of the coordinates of x be 0 as follows, then problem

$$Ax = 0, \quad 0 \neq x \geq 0. \tag{1.11}$$

and the von Neumann problem (1.6) are equivalently solvable.

1.3 Motivation

Rescaling is a linear transformation on the linear system with the aim of improving the condition of the problem, which is measured by the quantity ${}^{p}\rho$ in our cases. In other words, rescaling might enlarge the largest inscribed ball in the feasible region if the problem is feasible or increase the distance to the feasibility if the problem is infeasible. We observed that by successfully applying rescaling onto the PA, polynomial-time complexity can be achieved.

Analogous to the PA, the vNA is another elementary algorithm whose performance depends on the similar condition number ${}^{v}\rho$. As far as we are aware, there is no variant of the vNA has been proved to have polynomial-time complexity. Therefore, our goal is to explore the power of the rescaling methodology on designing rescaling vNA. The duality relationship between the PA and vNA, and Chubanov's method are two major directions we explore. The duality relationship provides possibility to design new variants of the vNA by transiting the existing variants of the PA. Also, the PA might benefit from the results of the vNA. Chubanov's method provides another way to apply linear transformation onto the linear problem. Therefore, Chubanov's method can be also considered as containing a rescaling procedure.

The third direction of our exploration is higher-order rescaling. Recall that for the stochastic rescaling PA, the procedure of generating the rescaling vector is independent of either the other phases of the algorithm or any results of previous iterations. This property makes the stochastic rescaling PA good for multi-core environment. Therefore, we study the effect of higher-order rescaling on the computational efficiency.

1.4 Structure of the Thesis

The structure of this thesis is built in the following way as we explore the power of rescaling from three different aspects. First, we start with Chapter 2 to explore the relationship between the perceptron and the von Neumann problems. The existing duality relationship builds a dual connection between these two families of algorithms and provides the theory basis for transiting algorithms from one side to the other side. Chapter 3 is a further study on the duality relationship. A deterministic rescaling vNA is presented by utilizing the duality on the Deterministic Rescaling PA. In addition, a percepton example is obtained by constructing its corresponding dual von Neumann example. These two examples show that the condition number ρ is not going to increasing monotonically in the deterministic rescaling algorithms. Secondly, Chapter 4 combines rescaling and Chubanov's Method to derive a polynomial-time column-wise rescaling vNA, which is the first variant of polynomial vNA. Finally, we explore effects of higher-order rescaling on the PA in Chapter 5. The computational results show that the practical performance is improved significantly under the multi-core environment by utilizing higher-order rescaling. Conclusions and future research directions are presented in Chapter 6.

Chapter 2

The Duality Between the Perceptron and the von Neumann Algorithm

2.1 Introduction

The PA and the vNA were developed to solve LFPs. In this chapter, we investigate and reveal the duality relationship between these two algorithms. The specific forms of LFPs solved by the PA and the vNA are a pair of alternative systems by the Farkas Lemma. Based on this observation, we interpret variants of the PA as variants of the vNA, and vice-versa; as well as transit the complexity results from one family to the other. A solution of one problem serves as an infeasibility certificate of its alternative system. Further, an Approximate Farkas Lemma enables us to derive bounds for the distance to feasibility or infeasibility from approximate solutions of the alternative systems. Based on Farkas Lemma, Section 2.2 discusses the duality relationship in general including the relationship between problems, condition numbers, and interpretation of solutions. Section 2.3 and Section 2.4 show the procedures of transiting variants of algorithms into their dual side.

2.2 Duality Relationship

In this section, we first employ the Farkas Lemma to analyze the duality relationship between problem (1.4) and problem (1.6). This observation provides the foundation for the duality between the PA and the vNA. Then we extend the definition of ρ to infeasible problems and give ρ meaningful explanations for different problems. At last, we propose to utilize an Approximate Farkas Lemma to interpret an approximate solution from its dual perspective.

2.2.1 Alternative Systems

Recall that conv(A) represents the convex hull of the points a_i. Assume that problem (1.4) is feasible and y is a feasible solution. In this case, the hyperplane with normal vector y separates conv(A) from the origin, which implies that problem (1.6) is infeasible. Conversely, if problem (1.6) is infeasible, then there exists at least one separating hyperplane that can separate conv(A) from the origin. In other words, there exists a vector y such that $A^T y > 0$, which means that problem (1.4) is feasible. Therefore, problem (1.4) and problem (1.6) are a pair of alternative systems. This conclusion can also be verified by the Farkas Lemma [4, 42, 44]. According to the Farkas Lemma, the alternative system of problem (1.6) is

$$\begin{aligned} A^T y + e\eta &\geq 0 \\ \eta &< 0. \end{aligned} \tag{2.1}$$

Problem (2.1) can equivalently be written as $A^T y > 0$, which is the form of problem (1.4). Thus, problems (1.4) and (1.6) are alternative systems to each other, i.e., exactly one of them is solvable. Since the PA and vNA solve problems (1.4) and (1.6), respectively, the duality relationship between these two problems leads to a duality between the two algorithms.

2.2.2 Calculation of Condition Number

Definition 1.1.1 in Section 1.1.4 gives a general definition of the condition number ρ. In this section, we discuss its special forms for the different problem forms in the different algorithms. The Classical PA shown in Section 1.2.1.1 assumes that problem (1.4) is strictly feasible. Thus, ${}^{p}\rho$ is only defined for feasible problems in [17]. In order to make the discussions about the duality relationship complete, we extend the definition of ${}^{p}\rho$ to infeasible cases.

Definition 2.2.1. *Consider the LFP (1.4).*

1. *If problem (1.4) is feasible [17], then the condition number ${}^{p}\rho$ is the radius of the largest inscribed ball that fits in the feasible region, and the center of the ball is on the unit sphere. It is calculated by*

$$ {}^{p}\rho = \max_{\|y\|=1} \min_{i} \{a_i^T y\}. \tag{2.2} $$

2. *If problem (1.4) is infeasible, then ${}^{p}\rho$ is the distance to feasibility, i.e.,*

$$ {}^{p}\rho = \min_{\|y\|=1} \max_{i} \{-a_i^T y\}. \tag{2.3} $$

Note that when problem (1.4) is feasible, then ${}^{p}\rho > 0$ if and only if problem (1.4) is strictly feasible. On the other hand, the specific ${}^{v}\rho$ for problem (1.6) in the vNA is defined as follows.

Definition 2.2.2. *[19] The condition number ${}^{v}\rho$ is the distance from the origin to the boundary $\partial(\mathrm{conv}(A))$ of the feasible set $\mathrm{conv}(A)$, i.e.,*

$$ {}^{v}\rho = \inf\{\|b\| : b \in \partial(\mathrm{conv}(A))\}. \tag{2.4} $$

Definition 2.2.2 also defines condition number ${}^{v}\rho$ with two different meanings depending on the feasibility or infeasibility of problem (1.6).

1. If problem (1.6) is feasible, then ${}^v\rho$ is the radius of the largest inscribed ball in conv(A) centered at the origin. It can be calculated [34] by

$$ {}^v\rho = \min_{\|y\|=1} \max_i \{-a_i^T y\}. \tag{2.5} $$

   When problem (1.6) is feasible but not strictly feasible, i.e., the origin is on the boundary of conv(A), then ${}^v\rho = 0$.

2. If problem (1.6) is infeasible, then ${}^v\rho$ is the distance from the origin to conv(A), i.e., ${}^v\rho$ is the radius of the largest separating ball centered at the origin. It can be calculated as

$$ {}^v\rho = \max_{\|y\|=1} \min_i \{a_i^T y\}. \tag{2.6} $$

Comparing (2.2), (2.3), (2.5), and (2.6), it is easy to see that when problem (1.6) is infeasible (feasible), the condition number ρ is computed in the same way as the one when problem (1.4) is feasible (infeasible). This observation originates from the alternative systems relationship of these two problems.

2.2.3 Interpretation of Approximate Solutions

Instead of providing an exact feasible solution as the PA does, the vNA returns an ϵ-solution when it terminates in a finite number of iterations. Analogously, the Modified PA [6] – a variant of the PA – returns a σ-feasible solution when the perceptron problem (1.4) is close to the boundary of feasibility and infeasibility. A σ-feasible solution is also an approximate solution which we will discuss later. When ϵ or σ is a fixed number, an ϵ-solution or a σ-feasible solution is not sufficient to draw a firm conclusion about feasibility of the problem. In this section and also in the following section, our goal is to give some interpretations of these approximate solutions from their alternative perspective, and answer the following questions.

- How to derive meaningful information from these approximate solutions?
- What conclusion can be drawn about the feasibility status of the problems?

The duality relationship discussed in Section 2.2.1 is directly derived from the Farkas Lemma. The two problems (1.4) and (1.6) are a pair of alternative systems and therefore, exactly one of them is solvable. Recall that both the Classical PA and the vNA are non-polynomial algorithms. When the problems are close to the boundary of feasibility and infeasibility, both the these two algorithms are inefficient. It takes exponentially many iterations for the algorithms to obtain a clear answer about solvability of the problems. Therefore, we would like that some variants of the algorithms could provide an approximate solution, or some indications of approximate feasibility or infeasibility. Due to the alternative system relationship between problems (1.4) and (1.6), a proof of infeasibility for one problem can be given by giving a solution to the other one. We are interested in exploring approximate solutions to this pair of alternative systems and their interpretations for their alternative systems.

We first discuss σ-feasible solutions [6]. Recall that in Definition 1.2.2, a vector y is defined as a nearly feasible solution to problem (1.4) if $A^T y \geq -\sigma\|y\|e$. Thus, we also call y a σ-feasible solution (or σ-solution for short reference). According to Definition 1.2.2, a σ-solution allows slight violations to the constraints in problem (1.4); and thus it is an approximate solution. Recall the analysis of the meaning of condition number ρ in Section 2.2.2. From Definition 2.2.2, we obtain the following theorem.

Theorem 2.2.3. *The following two statements are equivalent.*

(a) Perceptron problem (1.4) has a σ-solution.

(b) There is no ball in $\text{conv}(A)$ *centered at the origin with radius larger than σ.*

This theorem is directly derived from (2.5). Theorem 2.2.3 shows that a σ-feasible solution to perceptron problem (1.4) indicates that the corresponding von Neumann problem (1.6) is either infeasible, or if it is feasible then it is close to infeasibility. As a

result, we define such a σ-solution as a σ-infeasibility certificate for the von Neumann problem (1.6).

Definition 2.2.4. *A vector y is a σ-infeasibility certificate for the von Neumann problem (1.6) if $A^T y \geq -\sigma\|y\|e$.*

By combining Theorem 2.2.3 and Definition 2.2.4 we derive the following corollary.

Corollary 2.2.5. *A σ-infeasibility certificate indicates that the von Neumann problem (1.6) is either infeasible or feasible with ${}^v\rho \leq \sigma$.*

2.2.4 Approximate Farkas Lemma

In the previous section, a σ-feasible solution to the preceptron problem (1.4) is interpreted as a σ-infeasibility certificate of the von Neumann problem (1.6). In this section, we explore whether we can obtain an analogous result about an ϵ-solution. The major tool we employ is the Approximate Farkas Lemma [48].

Approximate Farkas lemmas are derived by Todd and Ye [48] from the general gauge duality results of Freund [21]. These lemmas are extensions of the Farkas Lemma [44] and quantify how certain approximate feasible solutions to a system of inequalities indicate the infeasibility of its alternative system. In order to adapt the Approximate Farkas Lemma, we first transfer a strictly feasible problems (1.4) to an optimization problem. Consider the following optimization problem

$$\begin{array}{rll} \alpha_{\tilde{y}} = & \min & \|\tilde{y}\| \\ & \text{s.t.} & A^T\tilde{y} \geq e, \end{array} \tag{2.7}$$

where $\tilde{y} \in \mathbb{R}^m$, matrix A has the same definition as in problem (1.4), and $\alpha_{\tilde{y}}$ denote the optimal value. This optimization problem aims to find a feasible solution to inequality system $A^T\tilde{y} \geq e$ with the shortest length. Comparing problem (2.7) and problem (1.4), any feasible solution to problem (2.7) is also a feasible solution to problem (1.4). On the other hand, if y^* is a strict feasible solution to problem (1.4), i.e., all coordi-

nates of $A^T y^* > 0$, then $\tilde{y}^* = \frac{y^*}{(y^*)^T A x(y^*)}$ is a feasible solution to problem (2.7), where $(y^*)^T A x(y^*)$ gives the smallest coordinate of $A^T y^*$. Thus, feasibility of problem (2.7) is equivalent to strictly feasibility of problem (1.4). When problem (1.4) is strictly feasible, we can put a ball into the feasible region of $A^T y > 0$. If the radius of the ball is fixed to 1, then $\alpha_{\tilde{y}}$ measures the minimal distance from the center of this unit ball to the origin. Recall that ${}^p\rho$ measures the radius of the maximal ball put in the feasible region and centered on the unit sphere. Comparing $\alpha_{\tilde{y}}$ with ${}^p\rho$, we can conclude that the closer problem (1.4) is to infeasibility, the narrower the feasible region is, i.e., the smaller ${}^p\rho$ is; and the further the unit ball is from the origin, i.e., the larger $\alpha_{\tilde{y}}$ is. Thus, $\alpha_{\tilde{y}}$ is seen as another measure of the robustness of problem (1.4). We obtain the following result by geometrical relationship.

$$\alpha_{\tilde{y}} = \frac{1}{{}^p\rho}. \tag{2.8}$$

By adapting the Approximate Farkas Lemma [48] to our problem, we obtain:

Lemma 2.2.6. *(Approximate Farkas Lemma) Consider the optimization problems*

$$\text{(GP)}: \quad \alpha_{\tilde{y}} = \min\left\{\|\tilde{y}\| \mid A^T\tilde{y} \geq e\right\}, \qquad \text{and}$$

$$\text{(GD)}: \quad \beta_b = \min\left\{\|b\| \mid Ax = b, e^T x = 1, x \geq 0\right\}.$$

Then $\alpha_{\tilde{y}}\beta_b = 1$.

The special case $0 \cdot +\infty$ is interpreted as 1. It is easy to see that problem (GD) is the perturbed problem of problem (1.6). When problem (1.4) is strictly feasible, then β_b gives the minimal distance between the origin and conv(A), which is the radius ${}^v\rho$ of the largest separating ball defined by (2.6). β_b also indicates the minimum corrections needed to make problem (1.6) feasible. By Lemma 2.2.6 and (2.8), we have $\beta_b = \frac{1}{\alpha_{\tilde{y}}} = {}^p\rho$. Therefore, the Approximate Farkas Lemma verifies that ${}^p\rho$ for feasible perceptron problem (1.4) is equivalent to ${}^v\rho$ for infeasible von Neumann problem (1.6). Thus, any

feasible solution to problem (GP) is an infeasibility certificate of problem (1.6), and gives a lower bound for the distance to feasibility.

On the other side, if problem (1.6) is feasible, then any feasible solution is an optimal solution to optimization problem (GD) with $\beta_b = 0$. According to the Approximate Farkas Lemma, $\alpha_{\tilde{y}} = +\infty$. It implies that problem (GP) is infeasible, then Lemma 2.2.6 reduces to the exact Farkas Lemma. In this case, problem (1.4) is either infeasible or feasible but not strictly feasible.

The following theorem utilizes the Approximate Farkas Lemma to interpret an approximate solution.

Theorem 2.2.7. *The following three statements are equivalent.*

(a) The von Neumann problem (1.6) has an ϵ-solution.

(b) A unit ball cannot be put closer than $1/\epsilon$ to the origin in the feasible region of problem (1.4).

(c) There is no ball in the feasible region of problem (1.4) centered on the unit sphere and its radius is larger than ϵ.

Proof. Problem (1.6) has an ϵ-solution x' such that $\beta'_b = \|b'\| = \|Ax'\| \leq \epsilon$, if and only if $\beta_b \leq \beta'_b \leq \epsilon$. By Lemma 2.2.6, this holds if and only if $\frac{1}{\epsilon} \leq \frac{1}{\beta_b} = \alpha_{\tilde{y}}$. The inequality $\frac{1}{\epsilon} \leq \alpha_{\tilde{y}}$ holds if and only if

$$\nexists \tilde{y} \text{ such that } \|\tilde{y}\| < \frac{1}{\epsilon} \text{ and } A^T\tilde{y} \geq e. \tag{2.9}$$

By scaling, (2.9) is equivalent to

$$\nexists y \text{ such that } \|y\| \leq 1 \text{ and } A^T y > \epsilon e. \tag{2.10}$$

Statement (2.9) is the statement (b). Recall the definition of ${}^p\rho$, (2.10) indicates that ${}^p\rho \leq \epsilon$. Thus, statements (a), (b), and (c) are equivalent. □

Theorem 2.2.7 shows that an ϵ-solution to the von Neumann problem (1.6) implies that the corresponding perceptron problem (1.4) is either infeasible, or if it is feasible then it is close to infeasibility. Therefore, we define such an ϵ-solution as an ϵ-infeasibility certificate for problem (1.4).

Definition 2.2.8. *A vector y is an ϵ-infeasibility certificate for the perceptron problem (1.4) if there exist a vector $x \in \Delta_n$ such that $Ax = y$ and $\|y\| \leq \epsilon$.*

We can derive following corollary from Theorem 2.2.7.

Corollary 2.2.9. *An ϵ-infeasibility certificate indicates that the perceptron problem (1.4) is either infeasible, or feasible with ${}^p\rho \leq \epsilon$.*

Since ϵ is a small positive number, it means that the norm of any feasible solution $\tilde{y}'$ of problem (GP), if it exists, is at least as large as $\frac{1}{\epsilon}$. Thus, if problem (GP) is close to infeasibility, its feasible solutions has to be large. For example, if problem (GP) is feasible and the distance to infeasibility is as small as 10^{-10}, then the magnitude of a feasible solution $\tilde{y}'$ has to be at least 10^{10}.

We utilized the definition of the condition number ρ and the Approximate Farkas Lemma to interpret approximate solutions so that we can draw more definitive conclusions about the solutions or feasibility of the problems. In addition, when the respective variants of the PA and vNA terminate at a certain point, then the Approximate Farkas Lemma allows a more precise interpretation of the output and provides meaningful information about the solution.

Inspired by the alternative system relationship of problems (1.4) and (1.6), we investigate the duality of the PA and the vNA. In Section 2.3, different versions of the PA are interpreted as variants of the vNA as they are applied to problem (1.6). In Section 2.4, we interpret variants of the vNA from the perspective of the PA. By exploring the intriguing duality of these algorithms, we not only gain new insight into the intimate relationship of these algorithms, but also derive several novel variants of these algorithms with their corresponding complexity results.

2.3 From Perceptron to von Neumann

Since problem (1.4) and (1.6) are alternative systems to each other, the PA can be operated on problem (1.6) with proper adjustments. The complexity results for the feasible case of the PA are adaptable for the infeasible case of the vNA. Since the PA has several variants, we discuss them in the following subsections.

In order to make our discussions more transparent, we rename the two spaces. The PA solves problems in form (1.4) to get a solution y if the problem is feasible. Thus, we call the space $\mathbb{R}^m$ in which the vector y lives the *perceptron space*. Similarly, because the vNA solves problem (1.6), we call $\mathbb{R}^n$ the *von Neumann space*. Note that the vector $b^k = Ax^k$ in the vNA, presented as Algorithm 1.3, is in the perceptron space. This reflects the duality of the two problems and also indicates some close relationships between the two algorithms. Matrix A can be seen as a linear operator between the perceptron and the von Neumann spaces.

2.3.1 The Normalized Perceptron Algorithm

We first revisit, Algorithm 1.1, the Classical PA. It starts from $y^0 = 0$ and at iteration k, from the point y^k it makes a unit step in the direction of a violated constraint a_j. Let x^k be the corresponding vector that satisfies $y^k = Ax^k$. We have $x^0 = 0$ and $x^{k+1} = x^k + e_j$, where $Ae_j = a_j$. According to this relationship, x^k is a sequence of vectors in the von Neumann space with $x^k \geq 0$ and $\|x^k\|_1 = k$ for all $k \in \mathbb{N}$. On the other hand, observe that the last two constraints in problem (1.6), $e^T x = 1, x \geq 0$ restrict vector x to be in the unit simplex Δ_n. The comparison of x^k at iteration k in the Classical PA and x in problem (1.6) leads to a normalized version of the PA [47]. Assume that problem (1.4) is feasible. The Normalized PA is as follows.

Note that at the end of each iteration, iterate y^k is inspected. The algorithm terminates if y^k is an ϵ-infeasibility certificate. This stopping criterion is derived from the von Neumann side after we successfully explain an approximate solution. In the Normalized

Algorithm 2.1 The Normalized Perceptron Algorithm

Initialization: Let $y^0 = 0$ and $k = 0$.
while True **do**
  Find a column $a_s, s \in \{1, 2, \cdots, n\}$ such that $a_s^T y^k \leq 0$.
  if such an a_s does not exist **then**
    STOP and return y^k.
  else

$$\begin{aligned} \lambda_k &= \frac{1}{k+1}, \\ y^{k+1} &= (1-\lambda_k)y^k + \lambda_k a_s, \\ k &= k+1. \end{aligned}$$

    if $\|y^k\| \leq \epsilon$ **then**
      STOP and return y^k as an ϵ-infeasibility certificate.
    end if
  end if
end while

PA, the k-th iterate y^k is divided by k to satisfy $y^k = Ax^k$ for some $x^k \in \Delta_n$. Thus, x^k is a vector x in the von Neumann space, as well as y^k can be interpreted as the corresponding b^k vector in the vNA. If the Normalized PA starts from $x^0 = 0$ and x^k can be updated to satisfy $x^k \in \Delta_n$ and $y^k = Ax^k$, then we get a variant of the vNA. To ease understanding, the derived Normalized vNA is described in full details as follows.

When applying to problem (1.6), the Normalized vNA has the following complexity result.

Theorem 2.3.1. *Let $\epsilon > 0$.*

1) If problem (1.6) is feasible, then after at most

$$\left\lceil \frac{1}{\epsilon^2} \right\rceil$$

iterations, the Normalized vNA provides an ϵ-solution.

2) If problem (1.6) is strictly infeasible, then after at most

$$\left\lceil \frac{1}{v\rho^2} \right\rceil$$

Algorithm 2.2 The Normalized von Neumann Algorithm

1: **Initialization**: Let $x^0 = 0$, $b^0 = Ax^0$, and $k = 0$.
2: **while** $\|b^k\| \geq \epsilon$ **do**
3: Find an $a_j, j \in \{1, 2, \cdots, n\}$ such that $a_j^T b^k \leq 0$.
4: **if** such an a_j does not exist **then**
5: STOP and return b^k as an infeasibility certificate of problem (1.6).
6: **else**

$$\begin{aligned} \lambda_k &= \frac{1}{k+1}, \\ x^{k+1} &= (1-\lambda_k)x^k + \lambda_k a_j, \\ b^{k+1} &= Ax^{k+1}, \\ k &= k+1. \end{aligned}$$

7: **end if**
8: **end while**
9: Return x^k as an $\epsilon-$solution.

iterations, an infeasibility certificate is given.

Proof. When problem (1.6) is strictly infeasible, then its alternative problem (1.4) is strictly feasible. Since y^k, generated by the Normalized PA, is exactly the same as y^k in the Classical PA divided by k, the complexity result of the Classical PA stated in Theorem 1.2.1 is also valid for Algorithm 2.1, the Normalized PA. Thus, the complexity result for strictly infeasible problems is an immediate corollary of Theorem 1.2.1.

Now we prove the complexity when problem (1.6) is feasible. By using induction on k, we prove that $\|b^k\| \leq \frac{1}{\sqrt{k}}$. For $k = 1$, since the algorithm starts with $b^0 = 0$,

$$\|b^1\| = \|(1-\lambda_0)b^0 + \lambda_0 Ax(b^0)\| = \|Ax(b^0)\| = 1,$$

where the last equality results from $\|a_i\| = 1$ for $i = 1, \ldots, n$.

Now, suppose that we have $\|b^{k-1}\| \leq \frac{1}{\sqrt{k-1}}$. At the iteration k, we obtain

$$\begin{aligned}
\|b^k\|^2 &= \|(1-\lambda_{k-1})b^{k-1} + \lambda_{k-1}a_j\|^2 \\
&= (1-\lambda_{k-1})^2\|b^{k-1}\|^2 + \lambda_{k-1}^2\|a_j\|^2 + 2\lambda_{k-1}(1-\lambda_{k-1})(a_j^T b^{k-1}) \\
&\leq \frac{1}{k^2}\left[(k-1)^2\|b^{k-1}\|^2 + 1\right] \leq \frac{1}{k}
\end{aligned}$$

The first inequality must be true because $a_j^T b^{k-1} \leq 0$ when problem (1.6) is feasible. The second inequality holds due to the inductive hypothesis $\|b^k\| \leq 1/\sqrt{k}$. Thus, to obtain an ϵ-solution, after k iterations, it is sufficient to have

$$\epsilon = \|b^k\| \leq 1/\sqrt{k}.$$

Therefore, the algorithm needs at most $\lceil 1/\epsilon^2 \rceil$ iterations. □

When problem is feasible, Theorem 2.3.1 shows that the complexity result of the Normalized vNA is the same as Dantzig's result (Theorem 1.2.6) for the vNA. However, there are two major differences between the Normalized vNA and the original vNA. At each iteration, the original vNA searches a_s which has the largest angle with b^k, and computes step-length λ based on the current iterate b^k and a_s. The Normalized vNA, which is transformed from the Classical PA, uses any a_j which satisfies $a_j^T b^k \leq 0$. Its update step-length only depends on k. Thus, the cost per iteration of the vNA is $2n^2$ more than that of the Normalized vNA.

Recall that Theorem 1.2.1 provides the complexity result for feasible perceptron problems. If problem (1.4) is strictly feasible, then the Classical PA returns a feasible solution in at most $1/^p\rho^2$ iterations. However, there is no published result for infeasible perceptron problems. Now, by transiting Theorem 2.3.1 back to the perceptron problems, we obtain the following new result for the Classical PA.

Theorem 2.3.2. *Let $\epsilon > 0$. Assume that problem (1.4) is infeasible, then after at most*

$$\left\lceil \frac{1}{\epsilon^2} \right\rceil$$

iterations, the Classical/Normalized PA provides an ϵ-infeasibility certificate, which indicates that there is no ϵ-ball in the feasible region.

The complexity bound in Theorem 2.3.2 only depends on the value ϵ, the accuracy of the infeasibility certificate, but does not depend on the geometry of the problem.

2.3.2 The Smooth Perceptron Algorithm

Soheili and Peña [47] proposed a smooth version of the PA and showed that it can be seen as a smooth first-order algorithm. This deterministic variant retains the original simplicity of the PA and its complexity is improved by almost a factor of $1/^p\rho$ compared to the Classical PA. The improved complexity result is given in Theorem 2.3.3. We first introduce the Smooth PA.

Given $\varphi > 0$, $x(y)$ is smoothed by the entropy prox-function

$$x_\varphi(y) = \frac{e^{\frac{-A^T y}{\varphi}}}{\left\| e^{\frac{-A^T y}{\varphi}} \right\|_1}, \tag{2.11}$$

where the expression $e^{\frac{-A^T y}{\varphi}}$ denotes the n-dimensional vector

$$e^{\frac{-A^T y}{\varphi}} = \left[e^{\frac{-a_1^T y}{\varphi}}, e^{\frac{-a_2^T y}{\varphi}}, \ldots, e^{\frac{-a_n^T y}{\varphi}} \right]^T.$$

The Smooth PA is as follows.

Compared to the complexity of the Classical PA stated in Theorem 1.2.1, Theorem 2.3.3 shows that the Smooth PA has a complexity result with $\frac{1}{^p\rho\sqrt{\log(n)}}$ improvement.

Algorithm 2.3 The Smooth Perceptron Algorithm

1: **Initialization**: Let $y^0 = \frac{Ae}{n}$, $\varphi_0 = 1$, and $x^0 = x_{\varphi_0}(y^0)$. $k = 0$.
2: **while** True **do**
3: Let $a_s = Ax(y^k)$.
4: **if** $a_s^T y^k > 0$ **then**
5: STOP and return y^k.
6: **else**

$$\begin{aligned} \lambda_k &= \frac{2}{k+3}, \\ y^{k+1} &= (1-\lambda_k)(y^k + \lambda_k A x^k) + \lambda_k^2 A x_{\varphi_k}(y^k), \\ \varphi_{k+1} &= (1-\lambda_k)\varphi_k, \\ x^{k+1} &= (1-\lambda_k)x^k + \lambda_k x_{\varphi_{k+1}}(y^{k+1}), \\ k &= k+1. \end{aligned}$$

7: **end if**
8: **end while**

Theorem 2.3.3. *[47] Assume that problem (1.4) is strictly feasible. Then after at most*

$$\frac{2\sqrt{\log(n)}}{{}^p\rho} - 1$$

iterations, the Smooth PA returns a feasible solution.

Analogous to the Normalized PA, the Smooth PA can also be applied to problem (1.6) when it is infeasible. Iterate y^k in the perceptron space plays the role of vector b^k in the vNA. Since b^k is updated so that $Ax^k = b^k$, we derive the corresponding vector x^k.

Compare Algorithm 2.3 and 2.4. Iterate y^k in Algorithm 2.3 is the same as vector b^k in Algorithm 2.4, and its corresponding x^k satisfying $y^k = Ax^k$ is the vector x in problem (1.6). It is easy to see that $x^k \in \Delta_n$ for all iterations. Therefore, the complexity result of Theorem 2.3.3 applies to Algorithm 2.4 as well when problem (1.6) is infeasible. We derive the following corollary from Theorem 2.3.3.

Algorithm 2.4 The Smooth von Neumann Algorithm

1: **Initialization**: Let $x^0 = \frac{e}{n}$, $b^0 = Ax^0 = \frac{Ae}{n}$, $\varphi_0 = 1$, and $\tilde{x}^0 = x_{\varphi_0}(y^0)$. $k = 0$.
2: **while** True **do**
3: Let $a_s = Ax(b^k)$.
4: **if** $a_s^T b^k > 0$ **then** STOP and return b^k as an infeasibility certificate.
5: **else**

$$\begin{aligned}
\lambda_k &= \frac{2}{k+3}, \\
b^{k+1} &= (1-\lambda_k)(b^k + \lambda_k A\tilde{x}^k) + \lambda_k^2 A x_{\varphi_k}(b^k), \\
\varphi_{k+1} &= (1-\lambda_k)\varphi_k, \\
\tilde{x}^{k+1} &= (1-\lambda_k)\tilde{x}^k + \lambda_k x_{\varphi_{k+1}}(b^{k+1}), \\
x^{k+1} &= (1-\lambda_k)(x^k + \lambda_k \tilde{x}^k) + \lambda_k^2 x_{\varphi_k}(b^k), \\
k &= k+1.
\end{aligned}$$

6: **end if**
7: **end while**

Corollary 2.3.4. *Assume that problem (1.6) is strictly infeasible. Then after at most*

$$\frac{2\sqrt{\log(n)}}{{}^v\rho} - 1$$

iterations, the Smooth vNA returns a certificate of infeasibility b^k such that $A^T b^k > 0$.

Recall that the Smooth PA has an improved complexity result compared to the Normalized PA when problem (1.4) is feasible. Thus, if problem (1.6) is infeasible, then after interpreted in the von Neumann space, the Smooth vNA also enjoys an almost $1/{}^v\rho$ complexity improvement compared to the one presented in Corollary 2.3.1.

Independently of our work, Soheili and Peña [46] proposed a version of smooth vNA called Iterated Smooth Perceptron-von Neumann (ISPVN) Algorithm. It is also based on the duality relationship between problems (1.4) and (1.6). Instead of using the entropy prox-function as the Smooth PA, it employs the Euclidean prox-function to smooth $x(y)$. The merit of the ISPVN Algorithm is that when problem (1.4) is infeasible with ${}^p\rho > 0$, the ISPVN Algorithm solves its alternative system (1.6). Thus, the ISPVN Algorithm could handle both problems (1.4) and (1.6) simultaneously. It either finds a feasible

solution to problem (1.4) in $O\left(\frac{\sqrt{n}}{\rho}\log\left(\frac{1}{\rho}\right)\right)$ iterations, or finds an ϵ-solution to the corresponding problem (1.6) in $O\left(\frac{\sqrt{n}}{\rho}\log\left(\frac{1}{\epsilon}\right)\right)$ iterations. Both of the iteration complexity of the ISPVN Algorithm are better than these of the PA and the vNA. However, in the case when problem (1.6) is infeasible, the Smooth vNA stated in Algorithm 2.4 has a better complexity bound.

2.4 From von Neumann to Perceptron

After interpreting variants of the perceptron algorithm from its dual perspective in Section 2.3, in this section we show how to interpret the vNA as a variant of the perceptron algorithm and how to apply it to problem (1.4).

2.4.1 The Original von Neumann Algorithm

The vNA was reviewed in Section 1.2.2. Note that in the vNA, iterates x^k are always in the unit simplex. The vector $b^k = Ax^k$ in the vNA can play the role of vector y in the perceptron space.

Algorithm 2.5 The von Neumann Algorithm Interpreted in the Perceptron Space

1: **Initialization**
Choose any $x^0 \in \Delta_n$. Let $y^0 = Ax^0$ and $k = 0$.
2: **while** $\|y^k\|_2 \geq \epsilon$ **do**
3: Let $\varphi_k = \|y^k\|_2$ and $\nu_k = (Ax(y^k))^T y^k$, where $x(y^k) = \underset{x\in\Delta_n}{\operatorname{argmin}}\{(y^k)^T Ax\}$.
4: **if** $\nu_k > 0$ **then** STOP and return y^k as a solution.
5: **else**

$$\begin{aligned} \lambda &= \frac{1-\nu_k}{\varphi_k^2 - 2\nu_k + 1}, \\ \varphi_{k+1}^2 &= \lambda\nu_k + (1-\lambda), \\ y^{k+1} &= \lambda y^k + (1-\lambda)Ax(y^k), \\ k &= k+1. \end{aligned}$$

6: **end if**
7: **end while**

According to Theorem 1.2.7, there are two possible outcomes of the vNA. If prob-

lem (1.6) is strictly infeasible, the vNA provides an infeasibility certificate. Then the alternative case in the perceptron space is that problem (1.4) is strictly feasible. Thus, applying the vNA to problem (1.4) will provide a feasible solution y^k. On the other hand, if problem (1.6) is strictly feasible, then the vNA will return an ϵ-solution with $\|b^k\| < \epsilon$, and $m+1$ applications of the vNA allows to get an exact solution [13], which is interpreted as an exact infeasibility certificate for problem (1.4). However, if problem (1.6) is neither strictly feasible with an ϵ-ball in the feasible set, nor strictly infeasible with at least ϵ distance to feasibility, then an ϵ-solution interpreted in the perceptron space implies an ϵ-infeasibility certificate of problem (1.4). An ϵ-solution/ϵ-infeasibility certificate could have two possible meanings.

1. If problem (1.6) is feasible, then problem (1.4) is infeasible.

2. If problem (1.6) is infeasible, then an ϵ-solution implies that the distance to the infeasibility of problem (1.6) is less than ϵ, i.e., ${}^v\rho < \epsilon$; and the radius of the largest inscribed ball in the feasible region of problem (1.4) is ${}^p\rho < \epsilon$. This means that though problem (1.4) is feasible, it is almost infeasible. The distance to the infeasibility is less then ϵ.

Thus, problem (1.4) is either infeasible or ϵ-close to infeasibility. From Theorem 1.2.7 the following complexity result can be derived for Algorithm 2.5.

Theorem 2.4.1. *Let $\epsilon > 0$.*

1) If problem (1.4) is strictly feasible, then after at most

$$\left\lfloor \frac{1}{\rho_p^2} \right\rfloor$$

iterations the vNA finds a feasible solution to problem (1.4).

2) If problem (1.4) is strictly infeasible, then after at most

$$\left\lceil \frac{2}{\rho_p^2} \ln \frac{1}{\epsilon} \right\rceil$$

iterations the vNA provides an ϵ-infeasibility certificate.

2.4.2 The Optimal Pair Adjustment Algorithm

Gonçalves et al. [27] introduced three variants of the vNA named as Weight-Reduction, Optimal Pair Adjustment (OPA), and Projection Algorithms. Among these three variants, the OPA Algorithm has the best performance in computational experiments. The basic idea of the OPA Algorithm is to move the residual b^k in Algorithm 1.3 closer to the origin 0 as much as possible at each update step. It gives the maximum possible freedom to two weights: the one in column a_{s^+} which has the largest angle with b^k and the one in column a_{s^-} which has the smallest angle with b^k. At each iteration, it finds the optimal value for these two coordinates and adjusts the remaining ones. The OPA Algorithm is as follows.
The OPA Algorithm has a better performance than the original vNA in practice [27], and Gonçalves proves that in theory it is at least as good as the original vNA.

Theorem 2.4.2. *[27] The decrease in $\|b^k\|$ obtained by an iteration of the OPA Algorithm is at least as large as that obtained by one iteration of the vNA.*

Therefore, the OPA and the vNAs share the same theoretical complexity as given in Theorem 1.2.7.

In the dual space, the residual b^k is the normalized iterate y^k in the perceptron algorithm. The column which has the largest angle with b^k corresponds the "most infeasible" constraint for y^k. Since a feasible solution to the von Neumann problem is an infeasibility certificate for the corresponding perceptron problem, the faster the residual b^k moves closer to 0, the sooner infeasibility of the perceptron problem is detected. Therefore, the OPA Algorithm outperforms the vNA when problem (1.4) is infeasible. In this section, we describe the equivalent OPA PA.

The subproblem in line 9 can be solved by enumerating all possibilities that satisfy the KKT conditions [27]. Analogous to Algorithm 2.5, if problem (1.4) is feasible and

Algorithm 2.6 The Optimal Pair Adjustment Algorithm

1: **Initialization**: Choose any $x^0 \in \Delta_n$. Let $b^0 = Ax^0$. Given a small positive number ϵ.

2: **while** $\|b^k\| \geq \epsilon$ **do**

3: **Find** the vectors a_{s^+} and a_{s^-} which make the largest and smallest angles with the current iterate y^k:

$$\begin{aligned} s^+ &= \underset{i=1,\ldots,n}{\operatorname{argmin}} \{a_i^T b^k\}, \\ s^- &= \underset{i=1,\ldots,n}{\operatorname{argmin}} \{a_i^T b^k | x_i > 0\}, \\ \nu_k &= a_{s^+}^T b^k. \end{aligned}$$

4: **if** $\nu_k > 0$ **then**

5: STOP, and return b^k as a feasible solution to problem (1.6).

6: **end if**

7: **Solve** the subproblem

$$\begin{aligned} \min \quad & \|\tilde{b}\|^2 \\ \text{s.t.} \quad & \lambda_0(1 - x_{s^+}^k - x_{s^-}^k) + \lambda_1 + \lambda_2 = 1, \\ & \lambda_i \geq 0, \text{ for } i = 1, 2, \end{aligned}$$

where $\tilde{b} = \lambda_0(b^k - x_{s^+}^k a_{s^+} - x_{s^-}^k a_{s^-}) + \lambda_1 a_{s^+} + \lambda_2 a_{s^-}$.

8: **Update**

$$\begin{aligned} b^{k+1} &= \lambda_0(b^k - x_{s^+}^k a_{s^+} - x_{s^-}^k a_{s^-}) + \lambda_1 a_{s^+} + \lambda_2 a_{s^-}, \\ x_i^{k+1} &= \begin{cases} \lambda_0 x_i^k & i \neq s^+, s^-, \\ \lambda_1, & i = s^+, \\ \lambda_2, & i = s^-. \end{cases} \\ k &= k + 1. \end{aligned}$$

9: **end while**

Algorithm 2.7 The Optimal Pair Adjustment Perceptron Algorithm

Initialization
Choose any $x^0 \in \Delta_n$. Let $y^0 = Ax^0$ and $u^0 = \|y^0\|$. $\iota = 1$.
Given a small positive number ϵ.
while $u^k \geq \epsilon$ **do**
    Find the vectors a_{s^+} and a_{s^-} which make the largest and smallest angles with the current iterate y^k:

$$\begin{aligned} s^+ &= \underset{i=1,\ldots,n}{\operatorname{argmin}} \{a_i^T y^k\}, \\ s^- &= \underset{i=1,\ldots,n}{\operatorname{argmin}} \{a_i^T y^k | x_i > 0\}, \\ \nu_k &= a_{s^+}^T y^k. \end{aligned}$$

    if $\nu_k > 0$ **then**
        STOP, and return y^k as a feasible solution to problem (1.4).
    end if
    Solve the subproblem

$$\begin{aligned} \min \quad & \|\tilde{y}\|^2 \\ \text{s.t.} \quad & \lambda_0(1 - x_{s^+}^k - x_{s^-}^k) + \lambda_1 + \lambda_2 = 1, \\ & \lambda_i \geq 0, \text{ for } i = 1, 2, \end{aligned}$$

    where $\tilde{y} = \lambda_0(\frac{y^k}{\iota} - x_{s^+}^k a_{s^+} - x_{s^-}^k a_{s^-}) + \lambda_1 a_{s^+} + \lambda_2 a_{s^-}$.
    Update
    if $\lambda_0 = 0$ **then**

$$\begin{aligned} \iota &= 1, \\ y^{k+1} &= \lambda_1 a_{s^+} + \lambda_2 a_{s^-}. \end{aligned}$$

    else

$$\begin{aligned} \iota &= \frac{\iota}{\lambda_0} \\ y^{k+1} &= y^k + (\iota\lambda_1 - x_{s^+}^k) a_{s^+} + (\iota\lambda_2 - x_{s^-}^k) a_{s^-}. \end{aligned}$$

    end if

$$\begin{aligned} u^{k+1} &= \frac{1}{\iota}\|y^{k+1}\|, \\ x_i^{k+1} &= \begin{cases} \lambda_0 x_i^k & i \neq s^+, s^-, \\ \lambda_1, & i = s^+, \\ \lambda_2, & i = s^-. \end{cases} \\ k &= k + 1. \end{aligned}$$

end while

Algorithm 2.7 terminates with $u^k < \epsilon$, then there is no ϵ-ball contained in the feasible cone centered on the unit sphere. In this case, problem (1.4) is close to infeasibility and y^k is an ϵ-infeasibility certificate. After interpreted as a variant of the PA, the OPA PA has the complexity result as stated in Theorem 2.4.1.

2.5 Summary

The perceptron and the vNAs are used to solve LFPs in different forms. In this chapter, we reveal the duality relationship between these algorithms. This observation is based on the fact that the forms of the LFPs these algorithms deal with are a pair of Farkas alternative systems. This relationship enables us to interpret variants of the prceptron algorithm as variants of the vNA, and vice versa. The dual interpretation of the algorithms allows us to transit the complexity results to the new algorithms too. The interpretation of an approximate solution is crucial during the algorithms transit. By utilizing the Approximate Farkas Lemma to make the solution meaningful for the alternative system and the transit complete. A major difference of these two algorithm families is that the PA assumes that the problem is feasible while the vNA solves both feasible and infeasible problems. Therefore, in this paper, we show that the infeasibility of perceptron problems are detected by the interpreted vNA (Algorithm 2.5) and the OPA PA (Algorithm 2.7); and the Normalized vNA (Algorithm 2.2) – interpreted from the Normalized PA is applicable for both strictly infeasible and feasible von Neumann problems. Furthermore, when problem (1.4) is infeasible, we derive a complexity result for the Classical PA from the perspective of the von Neumann space.

There is another variant of the PA – the Modified PA [6]. It starts from a random vector y. In order to interpret it from the von Neumann perspective, finding the corresponding vector x is a critical step. In addition, the Modified PA returns a σ-feasible solution – which is also an approximate solution – when the perceptron problem is feasible with small quantity ${}^p\rho$ – smaller than a given threshold. Therefore, it is worth to

design a modified version of the vNA which can return an infeasibility certificate when the problem is almost infeasible. Its interpreted variant will benefit detecting infeasibility of perceptron problems when ${}^{p}\rho$ is small.

Chapter 3

On Deterministic Rescaling Algorithms

3.1 Introduction

In this chapter, we further explore the application of the duality relationship between the perceptron and the von Neumann problems, also the duality between these two algorithms. We propose a Deterministic Rescaling von Neumann Algorithm which is a direct transformation of the Deterministic Rescaling Percepton Algorithm. Though the complexity of this new variant of the von Neumann algorithm is not proved yet, we show by constructing a von Neumann example that ${}^{v}\rho$ does not increase monotonically after each rescaling step. Therefore, proving its complexity cannot be based on monotonic expansion of ${}^{v}\rho$. Computational results show that the performance of the rescaling algorithm is improved compared to the original von Neumann Algorithm when solving ill-conditioned von Neumann problems.

Furthermore, due to the duality, the von Neumann example serves as the foundation of a perceptron example. Analogously, this perceptron example shows that with the Deterministic Rescaling Perceptron Algorithm by Peña and Sohèili, ${}^{p}\rho$ may decrease after one rescaling step.

Recall that the standard forms of the perceptron problem and the von Neumann problems are given by (1.4) and (1.6) respectively in Section 1.1. In general, being a pair of dual problems, their coefficient matrices are denoted by the same letter A. Without loss of generality [2], we can assume that matrix A in (1.4) has the same properties as A in (1.6). However, in this chapter, since we construct two respective examples based on the duality, superscripts are used for the purpose of clarification. The forms of these two problems are rewritten as follows. The perceptron problem is

$$ {}^{p}A^{T}y \geq 0,\ y \neq 0, \tag{3.1} $$

where ${}^{p}A \in \mathbb{R}^{m\times n}$ with its column vectors ${}^{p}a_1, {}^{p}a_2, \ldots, {}^{p}a_n \in \mathbb{R}^m$ and $y \in \mathbb{R}^m$. Without loss of generality, we may assume that $\|{}^{p}a_i\|_2 = 1$ for all $i = 1, 2, \ldots, n$. The von Neumann problem is

$$ \begin{aligned} {}^{v}Ax &= 0 \\ e^{T}x &= 1 \\ x &\geq 0, \end{aligned} \tag{3.2} $$

where ${}^{v}A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, and $e \in \mathbb{R}^n$ is the vector of all ones.

Recall that we use notation following several rules: (1) the superscript on the left indicates which problem the notation is used for, e.g., p for the perceptron problem and v for the von Neumann problem; (2) prime denotes the corresponding notation after rescaling; (3) the superscript on the right is either the iteration counter or an arithmetic operation depending on the context; (4) positive number subscript is the index of vectors, points, or coordinates.

3.2 A Deterministic Rescaling von Neumann Algorithm

Chapter 2 discusses the duality relationship between the perceptron and the von Neumann algorithms; and consequently interpreted variants of the perceptron algorithm as

variants of the von Neumann algorithm and vice versa. This relationship leads us to formalize an analogous Deterministic Rescaling von Neumann Algorithm according to the Deterministic Rescaling Perceptron Algorithm [38] introduced in Section 1.2.3.2.

3.2.1 A Deterministic Rescaling von Neumann Algorithm

We propose the following rescaled variant of the von Neumann algorithm.

Polynomial complexity of this algorithm still needs to be proved. In order to get closer to this result, we ask the following questions: can the complexity be proved based on the increase of ${}^v\rho$ as was done in the proof of the Stochastic Rescaling Perceptron Algorithm by Dunagan and Vempala [17]? Or else, analogous to the Deterministic Rescaling Perceptron Algorithm, is it possible to identify some increasing cap? Towards answering these questions, we construct an example of the von Neumann problem in the next section. This example not only shows that ${}^v\rho$ is not going to increase monotonically after each rescaling, but also helps us to generate an analogous perceptron example as presented in Section 2.

3.2.2 The Precision of Solutions

Before introducing the example of the von Neumann problem, we first discuss how rescaling steps change the precision of a solution.

Lemma 3.2.1. *Run Algorithm 3.1 on a von Neumann problem (3.2). Assume that starting from this original von Neumann problem, the algorithm has done t times rescaling steps (rescaling phase) and the current iterate in the von Neumann phase is b^k. If $\|b^k\| \leq \frac{\epsilon}{2^t}$, then x^* calculated by (3.3) is an ϵ-solution to the original von Neumann problem, i.e., $\|{}^vAx^*\| \leq \epsilon$.*

Proof. For one single rescaling step, matrix ${}^vA$ is rescaled by formula (3.4) and then each column is normalized back to the unit sphere. Let $B = I - \frac{1}{2}{}^va_j{}^va_j^T$ and $D = \text{diag}\left(\frac{1}{\|{}^va_1'\|}, \frac{1}{\|{}^va_2'\|}, \cdots, \frac{1}{\|{}^va_n'\|}\right)$. We have ${}^vA' = B{}^vAD$, where ${}^vA'$ is the matrix after rescaling. Assume that after rescaling $\|{}^vA'x\| = \|b\| \leq \epsilon$ and x is on the unit simplex. Observe

Algorithm 3.1 The Deterministic Rescaling von Neumann Algorithm

Initialization: Let $N = 6nm^2$, $D = I$, and let $t = 0$.
while True **do**
  Phase 1. von Neumann Phase [12, 14]
  (**Run** the von Neumann algorithm for N iterations)
  Choose any $x^0 \in \Delta_n$. Let $b^0 = {}^vAx^0$ and $k = 0$.
  for $k = 0, 1, \cdots, N-1$ **do**
    if $\|b^k\| \leq \frac{\epsilon}{2^t}$ **then** STOP, and return

$$x^* = \frac{Dx^k}{\sum_{i=1}^n (\phi_i x_i^k)} \tag{3.3}$$

as an ϵ-solution, where x_i^k is the i-th coordinate of x^k and ϕ_i is the i-th diagonal entry of D.

    else
      Find ${}^va_s$ which makes the largest angle with the vector b^k, i.e., ${}^va_s = {}^vAx(b^k)$. Let $\nu^k = {}^va_s^T b^k$.
      if $\nu^k > 0$ **then** STOP, problem (3.2) is infeasible.
      else
        Let e_s be the unit vector corresponding to index s. Update

$$\begin{aligned} \lambda &= \frac{1-\nu^k}{\|b^k\|^2 - 2\nu^k + 1}, \\ x^{k+1} &= \lambda x^k + (1-\lambda)e_s, \\ b^{k+1} &= {}^vAx^{k+1}, \\ k &= k+1. \end{aligned}$$

      end if
    end if
  end for
  Phase 2. Rescaling Phase
  Let $j = \underset{i=1,\cdots,n}{\operatorname{argmax}} \{e_i^T x^N\}$.
  Utilize ${}^va_j$ as the rescaling vector, do the linear transformation

$${}^vA = \left(I - \frac{1}{2} {}^va_j {}^va_j^T\right) {}^vA. \tag{3.4}$$

  Let

$$D = D\text{diag}\left(\frac{1}{\|{}^va_1\|}, \frac{1}{\|{}^va_2\|}, \cdots, \frac{1}{\|{}^va_n\|}\right),$$

where $\text{diag}(\phi_1, \phi_2, \cdots, \phi_n)$ means an $n \times n$ diagonal matrix whose diagonal entries are $\phi_1, \phi_2, \cdots, \phi_n$.

  Normalize each column of ${}^vA$ back to the unit sphere and let $t = t + 1$.
end while

that matrix B is invertible and its inverse can be computed according to the Sherman-Morrison formula [26]

$$B^{-1} = (I - \frac{1}{2}{}^{v}a_j {}^{v}a_j^T)^{-1} = I + {}^{v}a_j {}^{v}a_j^T.$$

Since ${}^{v}A'x = B {}^{v}ADx = b$, we have

$$\|{}^{v}ADx\| = \|B^{-1}b\| = \|(I + {}^{v}a_j {}^{v}a_j^T)b\| \leq \|b\| + \|({}^{v}a_j {}^{v}a_j^T)b\| \leq 2\|b\| \leq 2\epsilon. \tag{3.5}$$

It means that x is a solution of $\|{}^{v}ADx\| \leq 2\epsilon$. In order to recover a solution for the original problem, we need to bound $\|{}^{v}Ax^*\|$ above. Notice that x^* is also on the unit simplex and

$$\|{}^{v}Ax^*\| = \left\| \sum_{i=1}^{n} {}^{v}a_i x_i^* \right\| = \left\| \frac{\sum_{i=1}^{n} {}^{v}a_i \phi_i x_i}{\sum_{i=1}^{n}(\phi_i x_i)} \right\| = \frac{\|\sum_{i=1}^{n} {}^{v}a_i \phi_i x_i\|}{\sum_{i=1}^{n}(\phi_i x_i)}. \tag{3.6}$$

Since we also have the fact that

$$\frac{1}{\phi_i} = \|{}^{v}a_i'\| = \left\| {}^{v}a_i - \frac{1}{2}({}^{v}a_i^T {}^{v}a_j) {}^{v}a_j \right\| = \sqrt{1 - \frac{3}{4}\|{}^{v}a_i^T {}^{v}a_j\|} \leq 1, \tag{3.7}$$

which shows that rescaling always shrinks the length of column vectors of ${}^{v}A$. Combining (3.3), (3.6), and (3.7), we have after one rescaling step

$$\|{}^{v}Ax^*\| = \frac{\|{}^{v}ADx\|}{\sum_{i=1}^{n}(\phi_i x_i)} \leq \|{}^{v}ADx\| \leq 2\epsilon.$$

Therefore, ϵ needs to be reduced by a factor $\frac{1}{2}$ after each rescaling phase in order to keep the final solution x^* as an ϵ-solution to the original problem. If the total number of calling the rescaling phase is t, then in the worst case we need to reduce ϵ to $\frac{\epsilon}{2^t}$. The lemma is proved. □

3.3 Construction of a von Neumann Example with a Decreasing Ball

For an example that ${}^v\rho$ is not increasing monotonically, the constraint matrix ${}^v\!A$ has to satisfy the following properties.

Property 1. *Among all column vectors ${}^va_i$, there is at least one ${}^va_j$ such that after applying (3.4), ${}^v\rho' < {}^v\rho$.*

Property 2. *After running the von Neumann algorithm, ${}^va_j$ has the largest weight in the returned linear combination, i.e., the largest coordinate of x is corresponding to ${}^va_j$.*

In order to obtain these two properties, an LFP example is generated according to the following idea. First, create an initial convex hull with a known ${}^v\rho_0$, where ${}^v\rho_0$ is a small positive number. Second, identify vectors ${}^va_j$ from the columns of ${}^v\!A$ which shrink ${}^v\rho$ after rescaling. If no such column vector exist, then add new columns to ${}^v\!A$. As a result, matrix A satisfies Property 1. At last, if ${}^va_j$ obtained in the previous step does not satisfy Property 2, then add new perturbed points around ${}^va_i$ which have larger weight after running the von Neumann algorithm but would increase ${}^v\rho$ with rescaling. The function of these new perturbed points is to introduce perturbation by creating more small facets around the corner of those ${}^va_i$ and evenly share (distribute) the large weight when running the von Neumann algorithm, and consequently make Property 2 holds for ${}^va_j$.

We construct an example ${}^v\!A \in \mathbb{R}^3$ with 13 column vectors. Each column vector represents a point on the unit sphere. Let $\xi = 0.01$, which gives the initial ${}^v\rho_0$. Let $\zeta = \sqrt{1-2\xi^2}$ to simplify expressions. First, we construct a symmetric convex hull

defined by eight points (columns) as follows:

$$[p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8] = \begin{bmatrix} \zeta & \zeta & \zeta & \zeta & -\zeta & -\zeta & -\zeta & -\zeta \\ \xi & \xi & -\xi & -\xi & \xi & \xi & -\xi & -\xi \\ \xi & -\xi & \xi & -\xi & \xi & -\xi & \xi & -\xi \end{bmatrix}.$$

These points are symmetrically distributed on four hyperplanes. The distances between the origin and these four hyperplanes are all equal to $\xi = 0.01$. Figure 3.1 shows the positions of these initial points. For better illustration, the distances in Figure 3.1 are not drawn to scale. The unit sphere is presented for scale, while the four sub-circles are pushed much further away from the origin. The real distance is much smaller. We call these eight points major points.

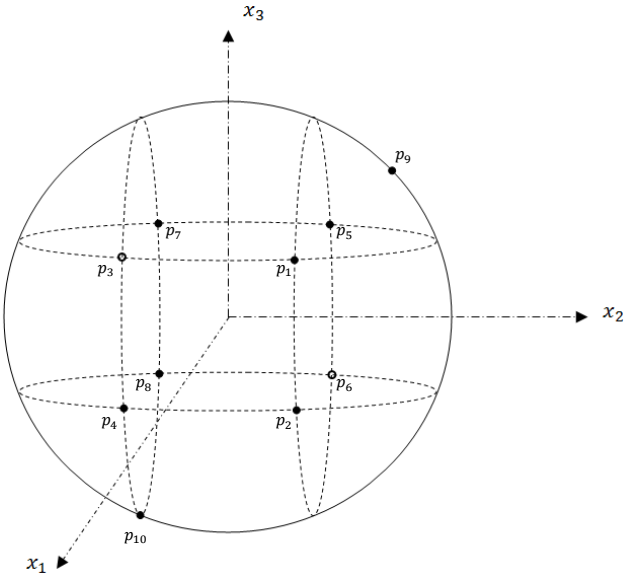


Figure 3.1: Illustration of the initial major points, p_9, and p_{10}.

In order to obtain a rescaling vector ${}^{v}a_j$, we add two points

$$[p_9, p_{10}] = \begin{bmatrix} 0 & 0 \\ \sqrt{1-\left(\frac{2\xi}{3}\right)^2} & -\xi \\ \frac{2\xi}{3} & -\sqrt{1-\xi^2} \end{bmatrix}.$$

Figure 3.1 also illustrates these two points. Computational experiment shows that with points p_9 and p_{10}, ${}^{v}\rho$ will decrease after rescaling by p_9. However, after running the von Neumann algorithm, p_9 does not take the largest weight. Therefore, we need more points (columns) of matrix A.

The goal of adding new points is to decrease large weights on other points so that after running the von Neumann algorithm p_9 has the largest weight. A point with larger weight indicates that the point has been used more for updating the iterate. Thus, after identifying those major points with large weight, we consider to add perturbed points near them. The perturbed points will diffuse the update process so that instead of the major points the perturbed points are used to update at some iterations. As a result, the perturbed points share weight with the major points. To prevent that the new perturbed points are dominated, which means the new perturbed points take all the weights from the major points, the perturbation should be small enough compared to the distance between the major points, which is $O(10^{-2})$. We set the magnitude of perturbation to $\delta = 10^{-6}$.

We present two methods to perturb a major point. The first method is to move the point along certain small circle on the unit sphere. We perturb point p_7 by this method to obtain p_{12} and p_{13} as follows.

$$p_{12} = \begin{bmatrix} -\sqrt{1-2\xi^2-\frac{17}{16}\delta^2-\frac{5}{2}\xi\delta} \\ -\xi-\delta \\ \xi+\frac{\delta}{4} \end{bmatrix},$$

$$p_{13} = \begin{bmatrix} -\sqrt{1 - 2\xi^2 - \frac{17}{16}\delta^2 - \frac{3}{2}\xi\delta} \\ -\xi - \delta \\ \xi - \frac{\delta}{4} \end{bmatrix}.$$

Observe that the second coordinates of p_{12} and p_{13} are more negative than the one of p_7, which means the direction of perturbation is pointing away from the initial convex hull. This is also the rule when we for the rest of perturbations. The second method to generate perturbed point is to move point along a given direction d with a step length $\delta = 10^{-6}$, then normalize it back to the unit sphere. Points p_3, p_4, and p_{12} are perturbed by the second method to generate p_{11}, p_{14}, and p_{15} respectively.

$$p_{11} = p_3 + \delta d_3 = \begin{bmatrix} x_1 \\ -\xi \\ \xi \end{bmatrix} + \delta \begin{bmatrix} 0 \\ -4 \\ 1 \end{bmatrix},$$

$$p_{14} = p_4 + \delta d_4 = \begin{bmatrix} x_1 \\ -\xi \\ -\xi \end{bmatrix} + \delta \begin{bmatrix} -10^{-4} \\ 0 \\ -0.01 \end{bmatrix},$$

$$p_{15} = p_{12} + \delta d_{12} = \begin{bmatrix} -\sqrt{1 - 2\xi^2 - \frac{17}{16}\delta^2 - \frac{3}{2}\xi\delta} \\ -\xi - \delta \\ \xi - \frac{\delta}{4} \end{bmatrix} + \delta \begin{bmatrix} -10^{-4} \\ 0 \\ -0.01 \end{bmatrix}.$$

After normalization and rearrangement, the new perturbed points can be expressed as

follows.

$$p_{11} = \left(1 + 10\xi\delta + 17\delta^2\right)^{-\frac{1}{2}} \begin{bmatrix} \zeta \\ -\xi - 4\delta \\ \xi + \delta \end{bmatrix},$$

$$p_{14} = \left(1 - 2 \times 10^{-4}\delta\zeta + (10^{-4} + 10^{-8})\delta^2 + 2 \times 10^{-2}\xi\delta\right)^{-\frac{1}{2}} \begin{bmatrix} \zeta - 10^{-4}\delta \\ -\xi \\ -\xi - 10^{-2}\delta \end{bmatrix},$$

$$p_{15} = \left(1 + \left(10^{-8} - \frac{49}{1.6 \times 10^5}\right)\delta - 2 \times 10^{-2}\xi\delta - 2 \times 10^{-4}\delta\sqrt{1 - 2\xi^2 - \frac{17}{16}\delta^2 - \frac{5}{2}\xi\delta}\right)^{-\frac{1}{2}} \times \begin{bmatrix} -\sqrt{1 - 2\xi^2 - \frac{17}{16}\delta^2 - \frac{5}{2}\xi\delta} + 10^{-4}\delta \\ -\xi - \delta \\ \xi + \frac{6}{25}\delta \end{bmatrix}.$$

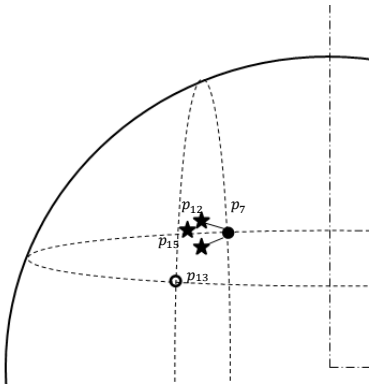
Figure 3.2 illustrates the perturbation of point p_7. After removing p_3 and p_6, we obtain our example (*Eg.vN*): problem (*vNPb*) with

$${}^vA = [{}^va_1, {}^va_2, \cdots, {}^va_{13}] = [p_1, p_2, p_4, p_5, p_7, p_8, \cdots, p_{15}].$$

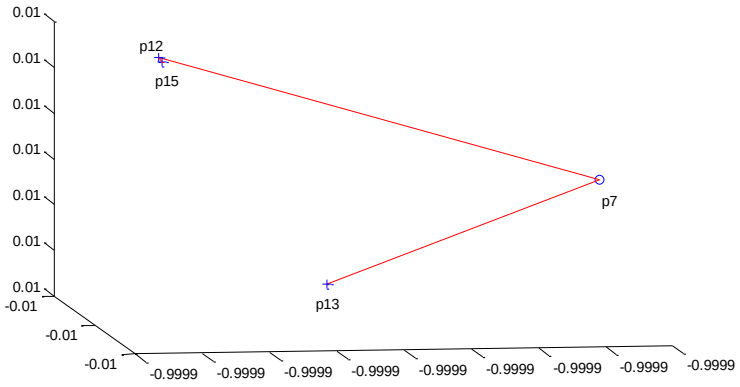
The reason that we remove points p_3 and p_6 is that p_9 will have the largest weight without them. We have the following two claims on this example (*Eg.vN*) and they will be verified in Section 3.4.

Claim 1. *For the von Neumann problem (Eg.vN), the radius of the largest inscribed ball ${}^v\rho$ will decrease if the problem is rescaled by (3.4) using ${}^va_7$ as the rescaling vector.*

Claim 2. *The von Neumann phase of the Deterministic Rescaling von Neumann Algorithm will identify ${}^va_7$ as the rescaling vector when applying the algorithm on (Eg.vN).*



(a) Relative positions.



(b) Drawn to scale.

Figure 3.2: Illustration of the perturbations of point p_7.

3.4 Verification of the von Neumann Example

In this section, we will verify Claim 1 in Section 3.4.1 – 3.4.2, and Claim 2 in Section 3.4.3 both theoretically and numerically.

3.4.1 The Initial Condition Number

To estimate ${}^{v}\rho$, we start from an initial convex hull comprised by the following ten columns p_i, where $i = 1, 2, \cdots, 10$. Figure 3.3 shows this initial convex hull. By construction, $p_1, p_2, \cdots, p_8$ compose a cube with an edge length of 0.02. It is easy to check that the radius of the largest inscribed ball in this initial convex hull is ${}^{v}\rho_0 = 0.01$. Then

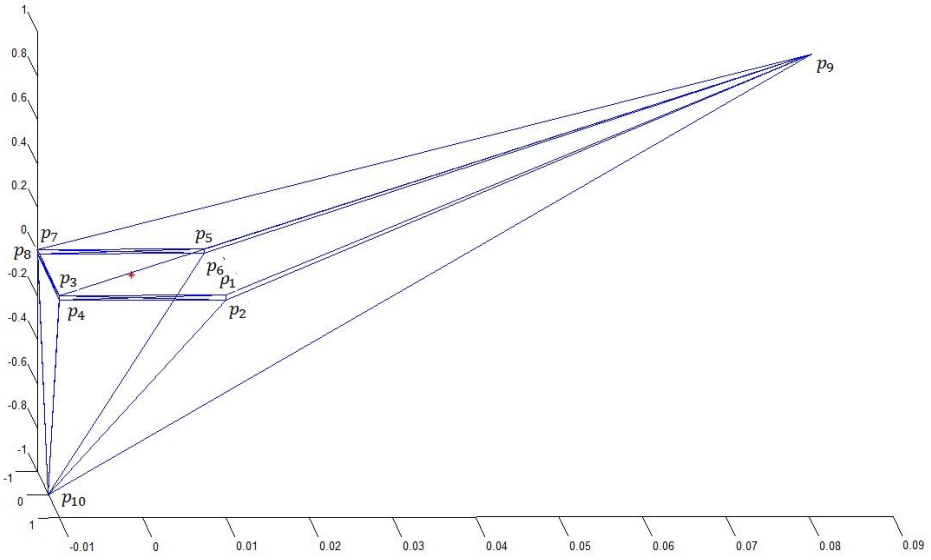


Figure 3.3: Illustration of the initial convex hull.

for the radius ${}^{v}\rho$, we have the following conclusions.

Lemma 3.4.1. *(a) The quantity ${}^{v}\rho_0$ is a lower bound of the largest inscribed ball in* $\text{conv}([{}^{v}a_1, {}^{v}a_2, \cdots, {}^{v}a_9])$. *Then (b) it also provides a lower bound for ${}^{v}\rho$, i.e., ${}^{v}\rho_0 \leq {}^{v}\rho$.*

Proof. The lemma can be proved by the procedure of construction, which is based on the initial convex hull shown in Figure 3.3.

(a) Due to the special positions of p_9 and p_{10}, removing p_6 only causes small changes of some inessential facets which compared with 0.01 have relatively larger distance to the origin. Thus, removing p_6 from the convex hull does not effect ${}^v\rho$. However, removing p_3 will generate a new facet defined by p_4, p_7, and p_9. This facet is closer to the origin than 0.01. Thus, we continue our constrcution with replacing p_3 by p_{11} instead of removing p_3 directly.

Recall that in the previous section, p_{11} is generated by perturbing p_3 along the direction of $[0; -4; 1]$ with a step size 10^{-6}. Point p_{11} is very close to p_3 compared to the distance among the facets and the origin. In the original convex hull, the facets containing p_3 as vertex are (p_3, p_1, p_4), (p_3, p_4, p_7), (p_3, p_7, p_9), and (p_3, p_1, p_9). Replacing p_3 by p_{11} rotate facets (p_3, p_4, p_7), (p_3, p_7, p_9), and (p_3, p_1, p_9) towards outside of the original convex hull and generates new facets with p_8 and p_{10}. Since (p_3, p_4, p_7) is the facet which defines ${}^v\rho$ in the original convex hull, the rotation relaxes this constraint and makes ${}^v\rho$ larger than 0.01. The replacement also brings the facet (p_3, p_1, p_4) closer to the origin. However, the original distance from this facet to the origin is almost one and the change is in the magnitude of 10^{-6}. Thus, it does not have effect on ${}^v\rho$. Figure 3.4 illustrates this replacement without drawing to scale.

Therefore, after removing p_6 and replacing p_3 by p_{11}, we obtain a convex set comprised by nine columns $\text{conv}([{}^va_1, {}^va_2, \cdots, {}^va_9]) = \text{conv}([p_1, p_2, p_4, p_5, p_7, p_8, p_9, p_{10}, p_{11}])$ and ${}^v\rho_0$ is a lower bound for the radius of the largest inscribed ball.

(b) Since $\|{}^va_i\| = 1$ for all i, $\text{conv}([{}^va_1, {}^va_2, \cdots, {}^va_9])$ is in the interior of the unit ball except nine vertexes ${}^va_1, {}^va_2, \cdots, {}^va_9$. All the new points $p_{12}, \cdots, p_{15}$ are on the unit sphere and different from ${}^va_1, {}^va_2, \cdots, {}^va_8$. Introducing them to matrix ${}^v\mathcal{A}$ will expand the convex hull, i.e., $\text{conv}([{}^va_1, {}^va_2, \cdots, {}^va_9]) \subset \text{conv}({}^v\mathcal{A})$. Therefore, ${}^v\rho \geq {}^v\rho_0 = 0.01$. □

To confirm this conclusion, we have calculated ${}^v\rho$ in MATLAB using IEEE double precision arithmetic. The unit roundoff error is $O(10^{16})$. Numerical calculation returns ${}^v\rho = 0.010002475$, which confirms that initially ${}^v\rho$ is larger than 0.01.

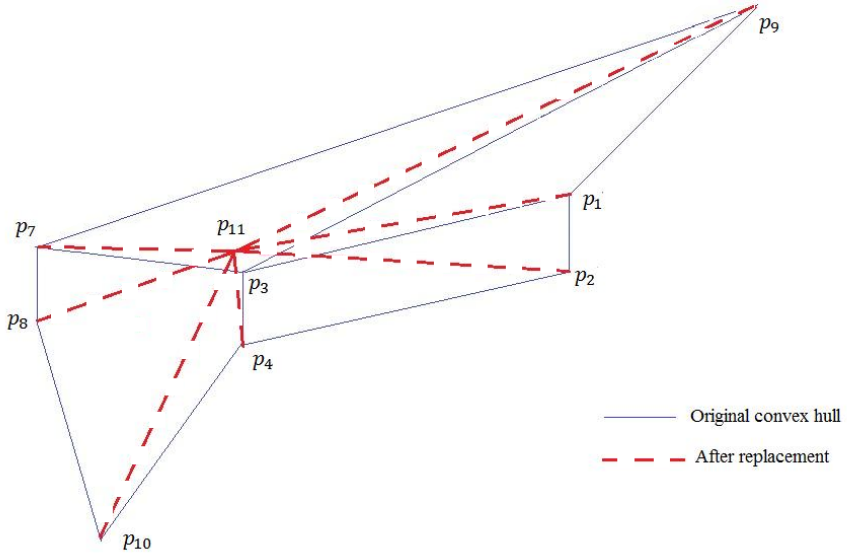


Figure 3.4: Illustration of replacing p_3 by p_{11}.

3.4.2 The Condition Number After One Rescaling Step

Numerical experiment shows that rescaling by using ${}^v a_7$ will decrease the size of the inscribed ball, i.e., ${}^v a_j = {}^v a_7 = p_9$ in (3.4). The hyperplane defined by ${}^v a_8, {}^v a_9, {}^v a_{10}$ restricts the ball. The distance from the origin to this hyperplane is 0.009964594, which gives an upper bound for ${}^v\rho'$, i.e., ${}^v\rho' \leq 0.009964594$.

We verify this number by multiple methods. First we solve the problem

$$ {}^v\rho' = \min_{\|y\|=1} \max_i \{-y^T ({}^v a')_i\} \tag{3.8}$$

by the *fminmax* function in MATLAB. The solution returned is also ${}^v\rho' = 0.009964594$. By the reasons stated in Section 3.5.2 for the function *fminmax*, we numerically enumerate all the facets of the convex hull and calculate the minimal distance from the origin to those facets and use both of LU and QR factorizations to solve linear equation systems in the process of calculating the hyperplanes. All the calculations are done in double precision arithmetic. The returned results are within the same order of $O(10^{-15})$ precision. The difference between ${}^v\rho$ and ${}^v\rho'$ is on the order of $O(10^{-5})$, which is much

larger than the roundoff errors. Thus we claim that ${}^v\rho > {}^v\rho'$. Therefore, utilizing ${}^va_7$ to rescale this von Neumann problem ($Eg.vN$) will shrink the inscribed ball. Claim 1 is verified.

3.4.3 Choosing the Rescaling Vector

In the desired rescaling von Neumann algorithm, we run the von Neumann algorithm to identify the rescaling vector, which need to be ${}^va_7$ in this example. The rescaling vector should be the corresponding column vector of the largest coordinate of x when the von Neumann algorithm stops after $6mn^2$ iterations [38]. Let x_i be the i-th coordinate of x, so $x = [x_1, x_2, \cdots, x_{13}]$. In our example, the von Neumann algorithm initiated with $x^0 = e_7$, where e_7 is the unit vector corresponding to index 7. After 3042 iterations, the numerical experiment shows that ${}^va_7$ and ${}^va_8$ have the same largest weight $x_7 = x_8 = x_{\max}$. Now we verify x_7 and x_8 are theoretically equal. Since we start from $x_7^0 = 1, x_8^0 = 0$. The superscript denotes the iteration counter. At the first iteration, ${}^va_8$ is utilized to update x. Thus, $x_7^1 = x_8^1 = 0.5$. After that, neither of ${}^va_7$ and ${}^va_8$ are used again to update. Throughout the following 3041 iterations, ${}^va_7^T({}^pAx^k)$ is always positive, and the minimal difference between ${}^va_7^T({}^pAx^k)$ and ${}^va_s^T({}^pAx^k)$ are in the order of $O(10^{-5})$ versus the numerical error is $O(10^{-15})$ for double digit accuracy. Thus, we recognize that there is enough separation between ${}^va_7$ and ${}^va_s$, and ${}^va_7$ is not overlooked. Consequently, x_7 and x_8 have exactly the same updates starting from the second iteration [28]

$$x_7^k = x_8^k = \lambda^k x_7^{k-1} = \cdots = \frac{1}{2}\prod_{i=2}^{k}\lambda^i.$$

Therefore, x_7 and x_8 remain equal, thus we can choose ${}^va_7$ as the rescaling vector. The computational experiment also shows that ${}^va_7$ is chosen when the von Neumann phase terminates. Claim 2 is confirmed.

3.5 A Perceptron Example with a Decreasing Ball Example

In this section, we explain how we derived the example for the Deterministic Rescaling Perceptron Algorithm which is stated in Section 1.2.3.2, followed by the example and its verification.

3.5.1 From the von Neumann Example to the Perceptron Example

Since constructing a von Neumann example in dimension three can be visualized, we start from the von Neumann example in spite the fact that the complexity of the Deterministic Rescaling von Neumann Algorithm has not been proved yet. After obtaining an example for the von Neumann algorithm for which the inscribed ball decreases, we adopt the following steps.

1. Identify all the facets of conv(^{v}A) and calculate their normal vectors.

2. Lift these normal vectors to a one dimension higher space. Since we already know that conv(^{v}A) only contains a small ball inside, lifting will lead to a narrow feasible cone.

3. Run the perceptron algorithm and remove the redundant constraints that are not used during updates.

4. Identify a constraint which can shrink $^{p}\rho$ when rescaling is done by its normal vector $^{p}a_j$.

5. Analogous to constructing the von Neumann example, adding perturbed constraints to balance the weight among all vectors so that $^{p}a_j$ is used the most frequently during the preceptron updates.

With the above five steps, we obtain the example presented in Section 3.5.2.

3.5.2 The Percetron Example

The example is as follows:

$$(Eg.p) \quad {}^pA^T y \geq 0,\ y \neq 0,$$

where ${}^pA \in \mathbb{R}^{4\times 9}$ and

$${}^pA^T = \begin{bmatrix} -0.000003029342674 & -2.019699173751262 & -0.000004999001640 & 0.020000000000000 \\ -0.019798999974999 & 0.019997999899990 & -0.019997999899990 & 0.020000000000000 \\ 0.001431631766736 & 0.019997999899990 & 0 & 0.020000000000000 \\ -0.000003028146773 & -1.973134679085590 & 0.183149852715338 & 0.020000000000000 \\ 0.019737351052173 & 0.000000950063128 & 0.020002002579065 & 0.020000000000000 \\ -0.052561097586474 & 1.592477159015729 & -0.091573429520343 & 0.024000000000000 \\ -0.052561703455009 & 1.188537324265476 & -0.091574429320671 & 0.028000000000000 \\ -0.052560491717939 & 1.996416993765981 & -0.091572429720015 & 0.020000000000000 \\ 0.050728859951203 & 1.996416993765981 & -0.091572429720015 & 0.020000000000000 \end{bmatrix}.$$

Each column vector ${}^pa_i$ of ${}^pA$ defines a hyperplane in $\mathbb{R}^4$ and there are nine hyperplanes in total. We have the following claims.

Claim 3. *For the perceptron problem (Eg.p), the radius of the largest inscribed ball ${}^p\rho$ will decrease if the problem is rescaled by (1.8) using ${}^pa_1$ as the rescaling vector.*

Claim 4. *The perceptron phase of the rescaling perceptron algorithm [38] will identify column ${}^pa_1$ as the rescaling vector when applying the algorithm on problem (Eg.p).*

3.5.3 Verification of the Perceptron Example

In order to verify these two claims, we also implement the example and the algorithm in MATLAB using IEEE double precision arithmetic. Recall that the unit roundoff error is $O(10^{-16})$.

We have the following observations. The initial ${}^p\rho = 0.00999988$. After running the rescaling perceptron algorithm [38], the perceptron phase does not solve ${}^pA^T y \geq 0, y \neq 0$. It identifies column ${}^pa_1$ as the rescaling vector, which is nearly perpendicular to the

feasible cone. In the rescaling phase, ${}^{p}a_1$ is used to rescale the matrix ${}^{p}A$. The radius of the largest inscribed ball after rescaling becomes ${}^{p}\rho' = 0.00961856$, which yields a factor of $O(10^{-4})$ decrease.

Verify Claim 3: the correctness of ${}^{p}\rho$ and ${}^{p}\rho'$ are checked first by solving

$$ {}^{p}\rho = \max_{\|y\|=1, {}^{p}A^T y \geq 0} \min_i \{ {}^{p}a_i^T y \} \tag{3.9} $$

in MATLAB using the *fminmax* function. The *fminmax* function uses a Sequential Quadratic Programming method [7] and might only return a local optimal solution. To dismiss this situation, we also verify the results by the following steps.

Step 1. Identify the hyperplanes that touch/support the current ball (${}^{p}\rho$ or ${}^{p}\rho'$).

Step 2. Project the normal vectors of the hyperplanes found in Step 1 to a three dimensional subspace. Denote these three-dimension vectors as ${}^{l}a_i$.

Step 3. Employ Dantzig's method [12] to solve the von Neumann problem

$$ \begin{aligned} {}^{l}A x &= 0, \\ e^T x &= 1, \\ x &\geq 0, \end{aligned} $$

where ${}^{l}A$ is composed by the vectors ${}^{l}a_i$ as its columns. If an exact solution is found, then this von Neumann problem is feasible, which proves that there is no direction in which the ball would grow. Dantzig's method yields to run the von Neumann algorithm multiple times and solve a linear equation system to obtain an exact solution to the von Neumann problem. The von Neumann algorithm is presented as the von Neumann phase of Algorithm 3.1 in Section 3.2. The most complex arithmetical operations in the process of verification involve vector normalization, matrix-vector multiplication, and solving linear equation systems. At each iteration of the von Neumann algorithm, the column

vectors ${}^{l}a_s$ which has the largest angle with the current iterate ${}^{l}Ax^k$ is chosen for update, where k is the iteration counter. The inner product values of ${}^{l}a_i^T({}^{l}Ax^k)$ are compared for all i. The minimal difference between ${}^{l}a_s^T({}^{l}Ax^k)$ and all the other ${}^{l}a_i^T({}^{l}Ax^k)$ values is $O(10^{-5})$ versus the numerical error is $O(10^{-16})$ in the double precision arithmetic. Thus, we recognize that the vectors ${}^{l}a_s$ are chosen correctly due to sufficient separation between the vectors. Regarding solving the linear equation systems, since the systems for our example are 4×4 dimensional, we use decomposition methods to solve them. Both LU and QR factorizations are applied to test our results. Though LU factorization is commonly used and needs less computation, however, QR factorization is more reliable in numerical computations. The accuracy of QR factorization is sufficient for most purposes [49]. The results of our experiment show that the values of ${}^{p}\rho$ and ${}^{p}\rho'$ are consistent while using different methods and factorizations. Therefore, executing Steps 1-3, we verify that ${}^{p}\rho = 0.00999988$ and ${}^{p}\rho' = 0.00961856$ are the radii of the largest inscribed balls before and after rescaling, respectively. Rescaling using ${}^{p}a_1$ makes the ball shrink. which verifies Claim 3.

Verify Claim 4: we have already noticed that the perceptron phase of the rescaled algorithm is actually the same as the perceptron algorithm. The minimal difference between ${}^{p}a_1^T({}^{p}Ax^k)$ and all the other ${}^{p}a_i^T({}^{p}Ax^k)$ is in the order of $O(10^{-5})$, which is much larger than the numerical error $O(10^{-16})$. Therefore, the vector ${}^{p}a_1$ is chosen correctly as the rescaling vector after running the rescaling perceptron algorithm. Claim 4 is verified.

3.6 Computational Results

The description of the Deterministic Rescaling von Neumann Algorithm is given in Section 3.2. As we stated, the theoretical complexity result of this algorithm is not proved yet. Regardless, we present some computation results in this section to show that the performance of the von Neumann algorithm is notably improved after applying the rescaling phase.

To generate ill-conditioned von Neumann problems which have small ${}^v\rho$, we adapt the tube generator [16, 29]. It places $n-1$ points on the spherical cap concentrated around $[0\ 0\ \ldots 0\ 1]^T$ or $[0\ 0\ \ldots 0\ -1]^T$. The n-th point is generated as a positive combination of the antipodes of the first $n-1$ points, so that it is on the opposite spherical cap. This generator guarantees that the von Neumann problem is feasible. At the mean time, since all the n points lie in the tube around the last coordinate axis, we can control ${}^v\rho$ by adjusting the width of the tube. The performance of the Deterministic Rescaling von Neumann Algorithm is compared with the original von Neumann algorithm. For each size of A, we randomly generated 20 von Neumann problems using tube generator. In Table 3.1, ${}^v\rho$ is controlled in the interval $[0.0015, 0.0025]$ by selecting proper width of the tube. When the dimensions of problems increase, the total number of updates remains at the same magnitude and the running time increases because the cost of each update step increases. The number of updates for finding a rescaling vector also depends on the dimension. Therefore, the number of rescaling step gets decreased. Table 3.2 shows for each size of problems, how the number of rescaling step increases while ${}^v\rho$ is decreasing.

The results in Table 3.1 and Table 3.2 are obtained by using Matlab R2014a on a

Table 3.1: Comparison of the performance of Algorithm 3.1 and the original von Neumann algorithm with ${}^v\rho \in [0.0015, 0.0025]$.

Size $m \times n$		Original		Deterministic rescaling			
		Sec.	No.update	Sec.	No.update	No.res	Speedup
5×10	Min	15.1439	6.0E+5	0.1852	9.0E+3	5	70.01
	Avg	20.7475	7.7E+5	0.2128	9.9E+3	6.05	93.00
	Max	26.9956	1.0E+6	0.2541	1.5E+4	7	144.97
10×20	Min	10.4675	2.4E+5	1.3518	4.8E+4	3	5.57
	Avg	16.6360	5.2E+5	1.4768	5.4E+4	3.8	10.91
	Max	25.3937	8.1E+5	1.9368	7.2E+4	5	15.93
20×40	Min	13.8834	2.3E+5	9.8932	1.9E+5	2	1.58
	Avg	30.6556	6.2E+5	13.2555	2.7E+5	2.1	2.45
	Max	58.6731	1.2E+6	15.1924	3.4E+5	3	3.79
40×80	Min	62.3095	3.7E+5	61.3109	3.7E+5	0	1.00
	Avg	114.9813	8.2E+5	98.7332	7.7E+5	1	1.12
	Max	142.4147	1.1E+6	101.8646	8.2E+5	1	1.43

Table 3.2: Comparison of the performance of Algorithm 3.1 and the original von Neumann algorithm with different ${}^{v}\rho$.

Size $m \times n$	${}^{v}\rho$ ($\times 10^{-3}$)	Original		Deterministic rescaling			
		Sec.	No.update	Sec.	No.update	No.res.	Speedup
10×20	$[1.5, 3]$	9.6168	3.1E+5	1.3234	4.8E+4	3.29	7.03
	$[3, 6]$	3.4683	1.1E+5	0.9868	3.6E+4	2.37	3.31
	$[6, 12]$	1.5130	4.8E+4	0.7997	2.9E+4	1.8	1.88
	> 12	0.7252	2.4E+4	0.5231	2.0E+4	1	1.39
20×40	$[0.5, 1]$	124.0785	3.7E+6	12.3569	4.3E+5	3.88	9.55
	$[1, 2]$	33.9353	1.0E+6	8.9600	3.1E+5	2.6	3.62
	$[2, 4]$	10.1445	2.8E+5	6.4094	2.0E+5	1.41	1.54
	> 4	7.2659	2.0E+5	6.3480	2.0E+5	1.5	1.1

Windows 7 desktop (Intel(R) Xeon(R) CPU, 2.5GHz) with 4Gb RAM.

3.7 Summary

Peña and Sohèili presented a Deterministic Rescaling Perceptron Algorithm. We construct an example showing that even though the algorithm eventually expands the feasible cone, ${}^{p}\rho$ may decrease after one rescaling step. By the duality relationship between the perceptron and the von Neumann algorithms, we apply the Peña-Sohèili rescaling method to the von Neumann algorithm. Driven by the desire of proving its complexity, we explore how ${}^{v}\rho$ will change after rescaling. We construct an example in $\mathbb{R}^3$ to show that there is no guarantee of monotonic increasing of ${}^{v}\rho$. Therefore, the complexity cannot be proved by increasing ${}^{v}\rho$ and another method need to be discovered. Computational results shows that the Deterministic Rescaling von Neumann Algorithm can solve the test problems faster then the original von Neumann algorithm.

Chapter 4

A Polynomial Column-wise Rescaling von Neumann Algorithm

4.1 Introduction

Recall that in Chubanov's Method, once the BP identifies an upper bound for at least one coordinate of any possible feasible solution, the corresponding columns of the coefficient matrix are multiplied by a scalar. Therefore, Chubanov's Method can be considered as s rescaling procedure. Utilizing this idea, in this chapter we propose a deterministic Column-wise Rescaling vNA and prove its polynomial-time complexity. We rename the BP as von Neumann Procedure (vNP) because it uses von Neumann-like update steps. The outline of this chapter is as follows. In the following of this section we introduce some notation and important lemmas that serve as the foundation of Chubanov's Method. In Section 4.3, we present the details of the column-wise rescaling von Neumann algorithm. In Section 4.2.1, we introduce different ways to compute upper bounds for some coordinates of any feasible x, if problem (1.10) has feasible solutions. These bounds are utilized to construct a rescaling matrix. The complexity analysis is presented in Section 4.4, and

computational results are given in Section 4.5.

Before presenting the details of the Column-wise Rescaling von Neumann Algorithm, we first introduce important notations and lemmas which are the foundation of Chubanov's Method. Let $\mathcal{N}_A$ denote the null space of the matrix A and $\mathcal{R}_A$ its row space, i.e.,

$$\mathcal{N}_A := \{x \in \mathbb{R}^n : Ax = 0\}, \quad \mathcal{R}_A := \{A^T y : y \in \mathbb{R}^m\}.$$

We define matrices P_A and Q_A as the orthogonal projection matrices of $\mathbb{R}^n$ onto $\mathcal{N}_A$ and $\mathcal{R}_A$, respectively, as follows:

$$P_A := I - A^T(AA^T)^{-1}A, \quad Q_A := A^T(AA^T)^{-1}A = I - P_A.$$

Our assumption that matrix A is full rank guarantees that AA^T is invertible. So P_A and Q_A are well defined. Let $x^{\mathcal{N}}$ and $x^{\mathcal{R}}$ denote the orthogonal decomposition of vector x in the spaces $\mathcal{N}_A$ and $\mathcal{R}_A$, respectively, i.e.,

$$x^{\mathcal{N}} := P_A x, \quad x^{\mathcal{R}} := Q_A x.$$

Obviously we have

$$AP_A = 0, \quad P_A Q_A = 0, \quad x = x^{\mathcal{N}} + x^{\mathcal{R}}.$$

According to the properties of the orthogonal decomposition [25], $P_A x = 0$ holds if and only if $x \in \mathcal{R}_A$, i.e., $x = A^T y$ holds for some y. In other words, problem (1.4) is equivalently solvable to the following problem

$$P_A x = 0, \quad x > 0. \tag{4.1}$$

Since problem (4.1) is homogeneous and x is strictly positive, without loss of generality,

we may assume that $e^T x = 1$. The concept of the orthogonal decomposition plays a crucial role in Chubanov's Method. The following lemma summarize the relationship between the orthogonal components and the solutions of problems.

Lemma 4.1.1. *For a vector $x \in \mathbb{R}^n$, if we have $0 \neq P_A x \geq 0$, then $x^{\mathcal{N}}$ is a solution to problem (1.11) and problem (1.6) is also solvable; if $P_A x = 0$ for some $x > 0$, i.e., x is a solution to problem (4.1), then problem (1.4) is solvable, i.e., $x = A^T y$ holds for some y.*

Proof. The first statement immediately follows from the definitions of $\mathcal{N}_A$ and P_A. For the second statement, if x is a solution to problem (4.1), then we have $x = x^{\mathcal{R}} + x^{\mathcal{N}} = x^{\mathcal{R}} \in \mathcal{R}_A$, which implies $x = A^T y > 0$ has a solution, i.e., problem (1.4) is feasible. By Farkas Lemma, problem (1.6) has no solution. □

Lemma 4.1.1 shows that the value of $P_A x$ for some x can be used to solve problem (1.6) when $0 \neq P_A x \geq 0$ or identify the feasibility when $P_A x = 0$. Therefore, as we show in the next section, Lemmas 4.1.1 serves as stopping criteria for the vNP in Chubanov's Method.

4.2 The von Neumann Procedure

Recall that Chubanov's problem (1.10) is homogeneous, we may assume without loss of generality that $0 < x \leq e$, where e denotes the all-one vector. Thus, we may equivalently consider the problem

$$Ax = 0, \quad x \in (0, 1]^n, \tag{4.2}$$

whose solution set is in the unit cube. The major difference between (1.6) and (4.2) is that every solution of (4.2) has to be strictly positive, while solutions of (1.6) still may have zero coordinates.

4.2.1 Bounds for Feasible Solutions

The core of Chubanov's Method is the vNP. The vNP is a von Neumann-like algorithm and works on problem (4.1). For the purpose of clarification, vector $u \in \mathbb{R}^n$ is used to denote the variable in problem (4.1) in the rest of this paper. Vector x is only used in the problems whose coefficient matrix is A, such as problems (1.6), (1.11), and (4.2). With the new notation, problem (4.1) can be rewritten as follows.

$$P_A u = 0, \quad e^T u = 1, \quad u > 0. \tag{4.3}$$

Recall that $u^{\mathcal{N}} = P_A u$ and $u^{\mathcal{R}} = Q_A u$. Due to the fact that P_A and Q_A are orthogonal projection matrices, with the assumption that problem (1.6) or (4.2) is feasible, an upper bound of every feasible solution x may be obtained from a given vector u. Let vector $d > 0$ denote this upper bound for x and its i-th coordinate d_i represent the upper bound for x_i, i.e., $x_i \leq d_i$ holds for every feasible solution x and every coordinate i. We will show later that vector d is crucial for rescaling. First we have the following observation.

Lemma 4.2.1. *Let vector d be an upper bound for every feasible solution x of problem (4.2). If $\max(d) < 1$, then problem (4.2) is infeasible.*

Proof. Observe that problem (4.2) is homogenous. Assume that it is feasible and x' is a feasible solution, then $x = \frac{x}{\max(x)}$ is another feasible solution which has at least one coordinate equal to 1. In other words, we have $x_j = 1$ for some j. According to the definition of vector d, $x_i \leq d_i$ holds for every feasible solution x and every coordinate i. Therefore, if problem (4.2) is feasible, then d_j has to be at least 1 for some j. □

Lemma 4.2.1 is utilized as an evidence of infeasibility in Algorithm 4.3 in Section 4.3. There are several ways to compute such an upper bound for x. Chubanov's original method uses the bound [9, 11]

$$x_i \leq d_i = \frac{\sqrt{n}\|u^{\mathcal{N}}\|}{u_i}, \tag{4.4}$$

where the subscript i is the index for coordinates. This inequality provides a useful bound only if the right hand side expression is smaller than 1, because we have made the assumption is that x is in the unit cube for problem (4.2). To determine if there is a bound (4.4) not greater than $\frac{1}{2}$ for some i, we can simply test the inequality

$$2\sqrt{n}\|u^{\mathcal{N}}\| \leq \max_i(u_i). \tag{4.5}$$

We have the following result.

Lemma 4.2.2. *[9, 11] Let u satisfy $0 \neq u \geq 0$ and (4.5), and let j be such that $u_j = \max_i(u_i)$. Let x be a solution for (4.2). Then x_j is bounded above by $d_j = \frac{1}{2}$.*

It will be convenient to call u *small* if it satisfies (4.5), and *large* otherwise. Note that $u^{\mathcal{N}} \neq 0$ if u is large, and u is small if (4.1) is feasible. For future use we also state the following result.

Lemma 4.2.3. *[41] If u satisfies $2\sqrt{n}\|u^{\mathcal{N}}\| \leq e^T u$, then u is small.*

Lemma 4.2.2 shows that any small vector u induces a bound $x_j \leq \frac{1}{2}$ for some j for problem (4.2). Recently more study shows that some large vectors u may also provide useful bounds for x. Roos [41] proposed a modified vNP using the following bound.

Lemma 4.2.4. *[41] Let x be a solution for (4.2). Then x_i is bounded by*

$$x_i \leq d_i = \min\left\{1, e^T\left[\frac{u^{\mathcal{R}}}{-u_i^{\mathcal{R}}}\right]^+\right\}, \text{ for } i = 1, \cdots, n, \tag{4.6}$$

where $[a]^+$ is derived from a by replacing its negative entries by zero, i.e., $[a]_j^+ = \max\{0, a_j\}$

By using the duality theorem of LO, Chubanov [11] also derived another bound in Lemma 4.2.5.

Lemma 4.2.5. *[41] Let x be a solution for (4.2). Then x_i is bounded by*

$$x_i \leq d_i = \min\left\{1, e^T\left[e_i - \frac{u^{\mathcal{R}}}{u_i}\right]^+\right\}, \text{ for } i = 1, \cdots, n, \tag{4.7}$$

where e_i is the i-th unit vector.

Among these three bounds (4.4), (4.7), and (4.6), Roos [41] concludes that for each nonzero $u \geq 0$ and for each i, one has

$$\min\left\{1, e^T\left[\frac{u^{\mathcal{R}}}{-u_i^{\mathcal{R}}}\right]^+\right\} \leq \min\left\{1, e^T\left[e_i - \frac{u^{\mathcal{R}}}{u_i}\right]^+\right\} \leq \min\left\{1, \frac{\sqrt{n}\|u^{\mathcal{N}}\|}{u_i}\right\}. \tag{4.8}$$

Bound (4.6) is the tightest upper bound for x. In the vNP, we only need to compute the smallest bound among all coordinates, so we define

$$d_{\min} \quad := \quad \min_i d_i = d_j,$$

where j is as follows: for the bounds (4.7) and (4.6), j is the index such that $u_j^{\mathcal{R}} = \max_i(u_i^{\mathcal{R}})$ if $e^T u^{\mathcal{R}} > 0$ and $u_j^{\mathcal{R}} = \min_i(u_i^{\mathcal{R}})$ if $e^T u^{\mathcal{R}} < 0$; for the bound (4.4), j is the index such that $u_j = \max_i(u_i)$. Note that j might not be unique.

4.2.2 The von Neumann Procedure

By iteratively updating vector u and the value of $P_A u$, the vNP aims to find a vector u which either satisfies one of the two conditions in Lemma 4.1.1, or if such an u is not found, then $u^{\mathcal{N}} \neq 0$ and there is at least one nonpositive coordinate of $u^{\mathcal{N}}$. Let S denote a nonempty set of indices such that

$$\sum_{s \in S} u_s^{\mathcal{N}} \leq 0.$$

Let p_s denote the s-th column of P_A, i.e., $p_s = P_A e_s$. We define

$$e_S := \frac{1}{|S|}\sum_{s\in S} e_s, \quad p_S := P_A e_S = \frac{1}{|S|}\sum_{s\in S} p_s.$$

The vNP is shown in Algorithm 4.1. Recall that vector u in problem (4.3) is analogous

Algorithm 4.1 [$\tilde{u}$, u, $u^{\mathcal{N}}$, $\tilde{J}$, $\tilde{d}$, CASE]=von Neumann Procedure(P_A, u)

1: Initialize: $\tilde{u} = 0$, $u^{\mathcal{N}} = P_A u$, $\tilde{J} = \emptyset$, $d_{\min} = 1$, CASE $= 0$, $\frac{1}{2} < \theta < 1$ (e.g. $\theta = 0.8$).
2: **while** $d_{\min} > \frac{1}{2}$ **and** CASE $= 0$ **do**
3: **if** $0 \neq u^{\mathcal{N}} \geq 0$ **then**
4: CASE $= 1$ ▷ Problem (1.6) is feasible.
5: Return
6: **else**
7: **if** $u^{\mathcal{N}} = 0$ **then**
8: CASE $= 2$ ▷ Problem (1.6) is infeasible.
9: Return
10: **else**
11: $\tilde{u} = u$
12: Find an index set S such that $\sum_{s\in S} u_s^{\mathcal{N}} \leq 0$
13: $\lambda = \frac{p_S^T(p_S - u^{\mathcal{N}})}{\|u^{\mathcal{N}} - p_S\|^2}$
14: $u = \lambda u + (1-\lambda)e_S$
15: $u^{\mathcal{N}} = \lambda u^{\mathcal{N}} + (1-\lambda)p_S$
16: **end if**
17: **end if**
18: Compute $d_{\min}$ by using (4.6)
19: **end while**
20: **if** CASE $= 0$ **then**
21: Compute d_i for all i by using (4.6)
22: $\tilde{d} = \{d_i : d_i \leq \theta\}$ ▷ Upper bound(s).
23: $\tilde{J} = \{i : d_i \leq \theta\}$ ▷ Corresponding coordinate index (indices).
24: **end if**

to vector x in problem (1.6). To solve problem (4.3), Algorithm 4.1 starts with u as a point from the unit simplex. Vector $u^{\mathcal{N}} = P_A u$ is analogous to vector b in Algorithm 1.3. Line 12-15 in Algorithm 4.1 is the update step. It moves $u^{\mathcal{N}}$ along a direction, which is a combination of one or more columns of P_A, with step size λ. The updating maintains at every iteration the conditions that u is from the unit simplex and the corresponding $u^{\mathcal{N}}$ is a convex combination of columns of P_A, i.e., $u \in \Delta_n$ and $u^{\mathcal{N}} \in \text{conv}(P_A)$. Since

this update step is analogous to the von Neumann update step in the vNA, we name this procedure as the vNP. As you will learn later in Section 4.3 that the vNP is the core subroutine of this proposed rescaling algorithm, therefore, we classify it as a variant of the vNA.

The vNP updates vector u until one of the following three cases occurs:

CASE = 1: We have $u^{\mathcal{N}} = P_A u$ as a solution of problem (4.2);

CASE = 2: We have $u^{\mathcal{N}} = 0$ that means that problem (4.2) is infeasible. Consequently u is a certificate of infeasibility;

CASE = 0: We have an index set $\tilde{J}$ and the corresponding bounds $\tilde{d}$ such that $x_{\tilde{J}} < \tilde{d}$ for all feasible solutions x of problem (4.2), and $\min(\tilde{d}) \leq \frac{1}{2}$. In other words, we have at least one coordinate j of x such that $x_j \leq \frac{1}{2}$ in every possible solution of (4.2).

As we will show in Section 4.3, $\tilde{d}$ is going to be used as the rescaling factor. In the case of rescaling, the vNP terminates when $d_{\min} \leq \frac{1}{2}$, i.e. the minimum value in $\tilde{d}$ is less than $\frac{1}{2}$. We also require that the maximum value of $\tilde{d}$ should not exceed a threshold $\theta \in (\frac{1}{2}, 1)$. Therefore, the vNP only records those $d_i \leq \theta$ into $\tilde{d}$ and their corresponding indices into $\tilde{J}$.

4.2.3 Complexity of the von Neumann Procedure

Roos has proved that the vNP in Algorithm 4.1 has strong polynomial-time complexity.

Theorem 4.2.6. *[41] After at most $4n^2$ iterations, the vNP either (a) provides a solution to problem (4.2), or (b) provides an evidence of infeasibility, or (c) identifies at least one coordinate of x which is smaller than or equal to $\frac{1}{2}$ in every feasible solution of (4.2).*

Each vNP iteration needs $O(n)$ arithmetic operations. Therefore, the vNP has $O(n^3)$ time complexity. Note that this is a strongly polynomial-time complexity.

4.3 The Column-wise Rescaling von Neumann Algorithm

In Section 4.2, we introduced the vNP to calculate an upper bound d for every feasible solution x for problem (4.2). In this section, we start with the idea of utilizing this upper bound as a rescaling vector. Then the Column-wise Rescaling vNA is discussed in details.

4.3.1 Rescaling

Since $x \leq d \leq e$ holds for every feasible solution x of problem (4.2), then $x'_i = \frac{x_i}{d_i} \leq 1$. This means that x' is a feasible solution to the following problem:

$$ADx = 0, \quad x \in (0, 1]^n, \tag{4.9}$$

where $D = \text{diag}(d)$, i.e., D is the diagonal matrix whose i-th diagonal entry is d_i. Observe that problems (4.2) and (4.9) are the same, if we replace A by AD. Since D is a diagonal matrix, AD is a rescaled version of A, where the i-th column of A is scaled by the factor d_i. This rescaling preserves the problem's form because e remains the upper bound for the variables.

When the vNP stops with an upper bound d, then the columns of A are rescaled by their corresponding d_i bound, respectively. The condition $d_{\min} \leq \frac{1}{2}$ ensures that at least one column is divided by at least a factor of $\frac{1}{2}$. This fact is used when proving the complexity result. Note that in Algorithm 4.1, the vNP only records the bounds which are less than a threshold θ, e.g., 0.8. After rescaling the vNP is called again to solve the rescaled problem, which has the same form but a different coefficient matrix. By repeating this vNP-rescaling procedure, a sequence of vectors d is constructed. The coordinate wise multiplication of these d vectors is denoted by $\hat{d}$ in Algorithm 4.3, as the final upper bound for every feasible solutions of problem (4.2).

It is well known that if problem (4.2) has rational data, there exists a positive number τ such that it is a uniform lower bound for all the positive coordinates in any basic

solutions. As discussed e.g. in [42], we have $\tau^{-1} = O(2^L)$, where L denotes the binary input length of matrix A [31]. After calling the vNP-rescaling procedure at most $O(nL)$ times, the upper bound for at least one coordinate of x will become smaller than τ, which is not possible if the problem has positive solution. Therefore, we can conclude that then problem (4.2) is infeasible.

4.3.2 Removing Columns

Compare the von Neuman problem (1.11) and problem (4.2). Every solution of (4.2) is restricted to be strictly positive. However, solutions of (1.11) may have zero coordinates. This difference leads to different conclusions in the case of $x_i < \tau$ for some i. As we stated in the previous section, when solving problem (4.2), we can conclude that if $x_i < \tau$, then problem (4.2) is infeasible. When solving problem (1.11), in such a case x_i has to be zero if the problem is feasible. We call such i a "must-be-zero" coordinate. Once a "must-be-zero" coordinate is identified, x_i is fixed to 0, and the corresponding column is removed from A without changing the feasibility of the problem.

Recall that in order to guarantee that P_A is well defined, we may assume that matrix A has full row rank. Removing columns from A may destroy this assumption. Therefore, a preprocessing step is needed before running the vNP-rescaling procedure again on the new problem. The preprocessing procedure eliminates any redundant rows to bring A back to a full rank matrix and reduces problem (4.2) to a similar problem with A replaced by a reduced matrix of A. The preprocessing procedure is stated as Algorithm 4.2. There are three possible outcomes of the preprocessing procedure:

CASE = 0: A is full rank and not a square matrix;

CASE = 2: A is full rank and square, then problem (1.11) is infeasible;

CASE = 3: $\text{rank}(A) = 0$, then problem (1.11) is feasible.

If the preprocessing procedure returns CASE = 3, then the non-zero coordinates of a feasible solution x can be any positive numbers. If CASE = 0, no action is needed.

Algorithm 4.2 [A,CASE]=PreProcessing(A, J_0)

if $J_0 \neq \emptyset$ **then**
  $A = A_{J_0}$ ▷ Remain column(s) J_0, remove others.
  if rank(A)=0 **then**
    Return CASE = 3 ▷ x_{J_0} (Line 35 in Algorithm 4.3) can be any positive numbers.
  end if
  if A is not full rank **then**
    Remove redundant row(s) of A to make it of full row rank
  end if
end if
Return A, CASE = 0

4.3.3 The Column-wise Rescaling von Neumann Algorithm

The Column-wise Rescaling vNA is stated as Algorithm 4.3. For convenience, the while loop in lines 9-32 is called inner loop, the while loop in lines 2-37 is called outer loop. The inner loop is the vNP-rescaling procedure for the actual matrix A. Once it identifies "must-be-zero" coordinates, the algorithm removes the corresponding columns from A, calls the preprocessing procedure, updates matrix A and P_A, and starts the vNP-rescaling procedure again.

4.4 Complexity

The following complexity result for the Column-wise Rescaling vNA shows that this is a polynomial time variant of the von Neumann algorithm.

Theorem 4.4.1. *After at most $O(n^5 \log_2 \tau^{-1}) = O(n^5 L)$ arithmetic operations, the Column-wise Rescaling vNA, as stated in Algorithm 4.3, either finds a solution to the von Neumann problem (1.11), or provides an evidence of its infeasibility.*

Proof. The number of inner-loop iterations is $O(n \log_2 \tau^{-1})$ for a given A. For each inner-loop iteration, the complexity of the vNP is $O(n^3)$ arithmetic operations. For each time calling the vNP, $O(n^3)$ arithmetic operations are needed for computing P_A. Therefore, the complexity of executing the inner loop is $O(n^4 \log_2 \tau^{-1})$. The complexity of the

Algorithm 4.3 The Column-wise Rescaling von Neumann Algorithm

Initialize: CASE $= 0$, $J = \emptyset$, and $J_0 = \{1, 2, \cdots, n\}$.
while CASE $= 0$ **do**
  $[A, \text{CASE}]$=PreProcessing(A, J_0) ▷ Check if A is full rank
  if A is square **then**
    CASE=2 ▷ System (1.11) is infeasible
    Break
  end if
  Set $\hat{d} = e$, $y = \frac{e}{n}$, $x = 0$ with corresponding dimension
  while CASE $= 0$ **do**
    $P_A = I - A^T(AA^T)^{-1}A$
    $[\tilde{y}, y, u^{\mathcal{N}}, \tilde{J}, \tilde{d}, \text{CASE}]$=von Neumann Procedure($P_A$, y)
    if CASE=0 **then**
      $\tilde{D} = \text{diag}(\tilde{d})$
      $\hat{d}_{\tilde{J}} = \tilde{D}\hat{d}_{\tilde{J}}$ ▷ d records the rescaling factors
      $A_{\tilde{J}} = \tilde{D}A_{\tilde{J}}$ ▷ Rescale matrix A
      if $\max(\hat{d}) < 1$ **then**
        CASE $= 2$ ▷ System (1.11) is infeasible
        Break
      end if
      if exists some coordinate set J such that $\hat{d}_J < \tau$ **then**
        $x_J = 0$
        $J_0 = J_0 \setminus J$
        Break
      else
        if $\tilde{y} \neq 0$ **then**
          $y = \tilde{y}$
        end if
        $y_J = y_J/2$
        $y = y/e^T y$
      end if
    end if
  end while
  if CASE $= 1$ **then** ▷ x is a solution of (1.1).
    $D = \text{diag}(\hat{d})$
    $x_{J_0} = Du^{\mathcal{N}}$
  end if
end while

Table 4.1: Comparison of the performance of Algorithm 4.3 and SeDuMi.

Size($m \times n$)	MA		SeDuMi		Speedup			
	Sec.	$\|Ax\|$	Sec.	$\|Ax\|$	Max	Min	%	Avg
5×10	0.0025	3.0e-13	0.0316	2.0e-9	58.6	0.2	99	16.9
25×50	0.0085	3.6e-12	0.1077	6.3e-9	89.1	0.9	98	10.8
125×250	0.1102	8.5e-11	0.7439	2.7e-8	27.4	0.6	99	6.0
250×500	0.2386	3.5e-10	3.8000	1.6e-7	41.1	3.0	100	10.6
500×1000	1.0553	8.3e-10	27.7594	4.4e-7	62.5	9.0	100	23.9
625×1250	2.4499	2.2e-10	61.6622	3.0e-8	81.2	15.0	100	32.4
1000×2000	7.6571	8.4e-10	555.3555	1.8e-7	114.5	25.1	100	50.6

preprocessing procedure is $O(n^3)$. The total number of executions of the outer loop is $O(n)$. Therefore, the total complexity of Algorithm 4.3 is $O(n^5 \log_2 \tau^{-1}) = O(n^5 L)$. $\square$

4.5 Computational Results

The performance of the Column-wise Rescaling vNA is compared to those of SeDuMi and Linprog. The bound used in the implementation is bound (4.6). For each size of A, we randomly generated 100 von Neumann problems with a dense matrix. The elements of A are randomly chosen in the intervals [-100,100].

Table 4.1 shows that for those randomly generated problems, the rescaling von Neumann algorithm outperforms SeDuMi. The running time shown has a significant reduction. The speedup columns compare the running time of SeDuMi versus the Column-wise Rescaling vNA for each problem, and shows the maximal, minimal, and average speedup ratios, as well as the percentage of problems which are solved faster by using this rescaling von Neumann algorithm. With averagely less than a tenth of the running time, the rescaling von Neumann algorithm returns solutions with higher accuracy then the ones obtained by SeDuMi. Table 4.2 compares the performance of the rescaling von Neumann algorithm and Linprog. For small problems, Linprog runs faster than the rescaling von Neumann algorithm. However, when the size is getting larger than 250×500, Linprog has a limited ability to solve all the problems. The numbers in the "Solved %" columns show how many problems out of 100 are solved successfully by each algorithm.

Table 4.2: Comparison of the performance of Algorithm 4.3 and Linprog.

Size($m \times n$)	MA			Linprog		
	Sec.	$\|Ax\|$	Solved %	Sec.	$\|Ax\|$	Solved %
5×10	0.0025	3.0e-13	100	0.0033	5.3e-11	100
25×50	0.0085	3.6e-12	100	0.0046	6.7e-11	100
125×250	0.1102	8.5e-11	100	0.0456	9.1e-10	100
250×500	0.2386	3.5e-10	100	0.3476	6.7e-9	48
500×1000	1.0553	8.3e-10	100	1.0407	9.4e-9	19
625×1250	2.4499	2.2e-10	100	–	–	0
1000×2000	7.6571	8.4e-10	100	–	–	0

The results in Table 4.1 and Table 4.2 are obtained by using Matlab R2014a on a Windows 7 desktop (Intel(R) Xeon(R) CPU, 3.07GHz) with 4Gb RAM. For the computation of the projection matrix P_A, we used the *factorize* function developed by Davis [15].

Chapter 5

A Higher-order Rescaling Perceptron Algorithm

5.1 Introduction

This chapter develops the methodology of a higher-order rescaling algorithm. We realize the perceptron improvement phase by utilizing parallel processors. In a multi-core environment, we can get several rescaling vectors without extra wall-clock time. Then we use them in a single higher order rescaling step. By this, a better rescaling rate may be achieved, and thus the complexity and computational efficiency is improved. Computational experiments shows that the practical efficiency of the rescaling algorithm improved by at least 40 percent compared to the one-order rescaling perceptron algorithm.

As we have introduced in Section 1.2.3.1, Dunagan and Vempala proposed a Stochastic Rescaling PA [17] to improve the complexity of the perceptron algorithm. By repeated rescaling of the linear system, radius ${}^{p}\rho$ increases so that with high probability (at least $1-e^{-m}$), the rescaling perceptron algorithm finds a feasible solution. Thus, the Stochastic Rescaling PA is a polynomial time algorithm with high probability.

A good characteristic of this algorithm is that the perceptron improvement phase always starts from a random vector to generate rescaling vectors at each cycle. Thus,

each improvement cycle is independent of any results from the previous cycles. This provides an opportunity for us to take advantage of parallel computing. Therefore, we propose a Higher-order Rescaling PA. We realize the perceptron improvement phase by utilizing parallel processors. In a multi-core environment we may obtain several rescaling vectors without extra wall-clock time. Then we use them in a single *higher order rescaling phase*. Numerical experiments show that the Higher-order Rescaling PA has notably improved efficiency in practice. Before introducing the Higher-order Rescaling PA, we first introduce several new notations which will be used frequently in this section.

- z: a unit vector such that

$$z = \underset{\{:y\in\mathcal{F},\|y\|=1\}}{\arg\max} \min_i a_i^T y.$$

- σ_1, σ_2: in the Stochastic Perceptron PA (Algorithm 1.4), the perceptron phase and the perceptron improvement phase share the same parameter σ in the upper bounds of total number of iterations. In addition, σ is also the parameter used to define the nearly feasible solution. In this chapter, these two phases might have σ with different values. Therefore, σ_1 and σ_2 replace the original σ in the perceptron phase and the perceptron improvement phase respectively, and σ_2 also replaces the parameter σ in the condition of the nearly feasible solution.

- $\Pr(\Omega)$: the probability of event Ω.

- $\bar{y}$: recall that in Algorithm 1.4, $\bar{y}$ is defined as the unit vector along y, i.e., $\frac{y}{\|y\|}$. For the convenience of our discussion, in this chapter, we define that vector "over-lined" is the corresponding unit vector.

5.2 The Higher-Order Rescaling Perceptron Algorithm

In this section we present the Higher-order Rescaling PA and analyze its theoretical properties.

In the rescaling phase of the Stochastic Rescaling PA (Algorithm 1.4), only one nearly feasible solution is used to make the linear transformation. Recall that at each iteration, the system is rescaled (Line 14) by $A = \left(I + \bar{y}\bar{y}^T\right) A$, where y is a nearly feasible solution obtained in the perceptron improvement phase. After rescaling, with probability at least $\frac{1}{8}$, the largest inscribed ball is enlarged. Since $\bar{y}\bar{y}^T$ is of rank-one, the Stochastic Rescaling PA is an order one rescaling algorithm. We propose a Higher-order Rescaling PA that uses more than one nearly feasible solutions and makes higher-order updates at each rescaling phase. The goal of the higher-order update is to utilize, for virtually no cost, in a multi-processor computing environment, to obtain a higher rescaling rate.

Similar to the Stochastic Rescaling PA, each outer loop of the Higher-order Rescaling PA consists of three phases, and both the perceptron phase and the perceptron improvement phase have an inner loop.

In the Stochastic Rescaling PA, the search for a nearly feasible solution in phase 2 always starts from a random vector, which is independent of any interim result and other status of the problem. Thus, the perceptron improvement phase can be run on the κ_0 processors in parallel without any communication between each other, and we may get several nearly feasible solutions at the same time. In such a computational model, no matter how many processors are used, the wall-clock time will not be increased. So the generation of multiple rescaling vectors simultaneously can be considered as a single step. Then defined by

$$A' = (I + \bar{y}_1\bar{y}_1^T + \cdots + \bar{y}_\kappa\bar{y}_\kappa^T)A, \tag{5.1}$$

a higher-order rescaling transformation is applied in the rescaling phase, where A' denotes the matrix after rescaling. We call (5.1) a κ-order rescaling/transformation.

Algorithm 5.1 The Higher-order Rescaling Perceptron Algorithm

Initialization
Assume that there are κ_0 parallel processors available.
Let $B \in \mathbb{R}^{m\times m}$, $B = I$, $\sigma_1 = \frac{1}{32m}$, and $\sigma_2 = \frac{1}{32\kappa_0 m}$.
while True **do**
  Phase 1. The Perceptron Phase
    Designate one processor as the main processor.
    On the main processor:
      run *the Classical PA* for at most $\left\lceil \frac{1}{\sigma_1^2} \right\rceil$ iterations;
      obtain vector y.
    if $A^T y \geq 0$ **then**
      STOP and return By as a feasible solution.
    end if

  Phase 2. The Perceptron improvement Phase
    On each processors:
      run lines 9 of Algorithm 1.4 with $\sigma = \sigma_2$;
      obtain κ_0 vectors.
    if there is any non-zero vector that satisfies $A^T y \geq 0$ **then**
      STOP and return By as a feasible solution.
    end if

  Phase 3. The Higher-Order Rescaling Phase
    Assume that $y_1, y_2, \cdots, y_\kappa$ are κ nearly feasible solutions among the total κ_0 vectors.
    Set

$$\begin{aligned} A &= (I + \bar{y}_1\bar{y}_1^T + \cdots + \bar{y}_\kappa\bar{y}_\kappa^T)A, \\ B &= (I + \bar{y}_1\bar{y}_1^T + \cdots + \bar{y}_\kappa\bar{y}_\kappa^T)B. \end{aligned}$$

end while
Output: A point y such that $A^T y \geq 0$ and $y \neq 0$.

5.3 Probability of Getting Good Recaling Vectors

Recall that in the perceptron improvement phase, the generation of nearly feasible solutions starts from random vectors, which causes the rescaling algorithms to have a stochastic property. Thus we discuss the probability of getting a good vector in the improvement phase.

Definition 5.3.1. *Let z be a unit vector such that ${}^p\rho = \max_{z \in \mathcal{F}} \min_i a_i^T z$, i.e., z is the unit vector pointing to the center of the largest inscribed ball. A vector y is called a* good vector *if*

$$z^T y \geq \frac{\omega}{\sqrt{m}},$$

where ω is a predetermined parameter. Further, y is a good nearly feasible solution *if y is both a good vector and a nearly feasible solution.*

Lemma 5.3.2. *Let $\Phi(\cdot)$ be the cumulative distribution function of the standard normal distribution. On a single processor, with probability at least $1 - \Phi(\omega)$, after at most $\frac{\ln(m/\omega^2)}{\sigma_2^2}$ iterations, the perceptron improvement phase returns a good nearly feasible solution.*

Proof. The proof is similar to the one in [17]. Let u be any unit vector. We first show that for a random unit vector y, the probability of $u^T y \geq \frac{\omega}{\sqrt{m}}$ is at least $1 - \Phi(\omega)$. We generate a random unit vector as follows: pick each coordinate independently from a standard normal distribution and normalize the vector to unit length. Let $Y_1, \cdots, Y_m$ be these coordinates. Since u can be any unit vector, we may assume without loss of

generality that $u = e_1$. Then the desired probability is

$$\begin{aligned} Pr &= Pr\left(u^T y \geq \frac{\omega}{\sqrt{m}}\right) \\ &= Pr\left(\frac{Y_1}{\sqrt{\sum_{i=1}^m Y_i^2}} \geq \frac{\omega}{\sqrt{m}}\right) \\ &= \frac{1}{2} Pr\left(\frac{Y_1^2}{\sum_{i=1}^m Y_i^2} \geq \frac{\omega^2}{m}\right) \\ &= \frac{1}{2} Pr\left(Y_1^2 \geq \frac{\omega^2(m-1)}{m-\omega^2} \cdot \frac{\sum_{i=2}^m Y_i^2}{m-1}\right). \end{aligned}$$

Each Y_i^2 has a χ-squared distribution. The probability is a monotonic decreasing function of m. If ω is a constant which is independent of m, then the limit as m increases is

$$Pr = \frac{1}{2} Pr(Y_1^2 \geq \omega^2) = Pr(Y_1 \geq \omega) = 1 - \Phi(\omega),$$

where Y_1 is standard normally distributed. Thus, the probability can be obtained by the cumulative distribution function of the standard normal distribution. Since z is a special case of u, the result holds for z.

Then we prove that the perceptron improvement phase will terminate if the starting random vector y is a good vector. Let y^1 be the vector after one update step, i.e., $y^1 = y - (a_s^T y)a_s$, where $a_s^T \bar{y} < -\sigma_2$. Consider the norm of y^1

$$\begin{aligned} \|y^1\|^2 &= (y - (a_s^T y)a_s)(y - (a_s^T y)a_s) \\ &= \|y\|^2 - (a_s^T y)^2 \\ &\leq \|y\|^2(1 - \sigma_2^2). \end{aligned}$$

Thus $\left\|y^1\right\|$ will decrease at least by a factor $(1-\sigma_2^2)^{\frac{1}{2}}$. At the same time, $z^T y^1$ does not

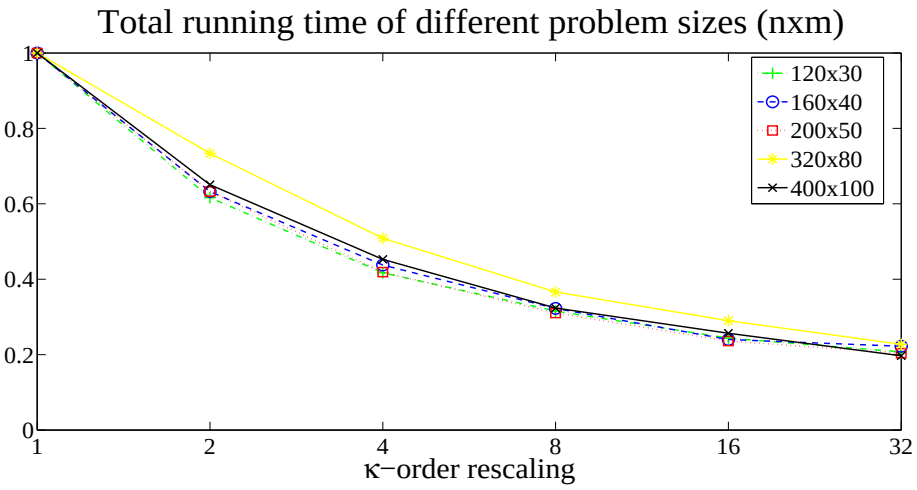
decrease because $a_s^T \bar{y} < -\sigma_2$ and $a_s^T z \geq 0$,

$$\begin{aligned} z^T y^1 &= (y - (a_s^T y) a_s)^T z \\ &= y^T z - (a_s^T y)(a_s^T z) \\ &\geq y^T z. \end{aligned}$$

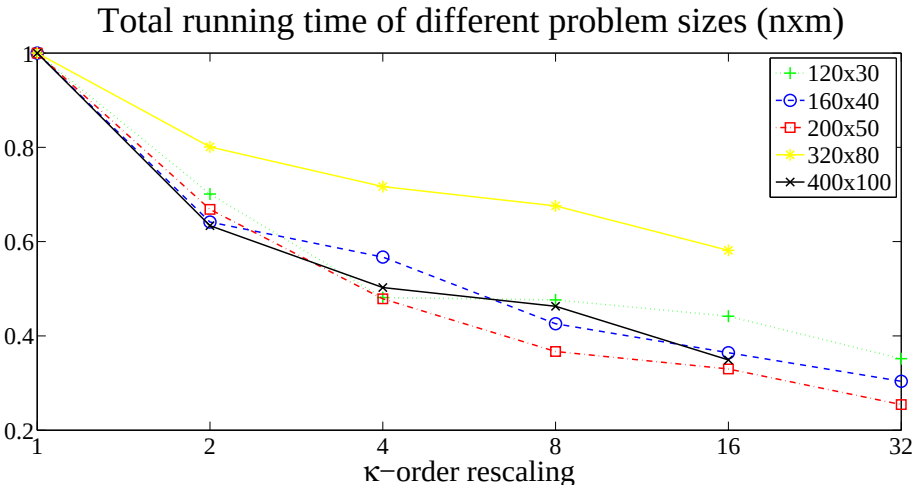
Thus, $\bar{y}^T z$ is increasing monotonously. Since the initial value of $\bar{y}^T z = y^T z \geq \frac{\omega}{\sqrt{m}}$ because of y being a good vector, after $\frac{\ln(m/\omega^2)}{\sigma_2^2}$ iterations, we would have $\bar{y}^T z > 1$ which cannot happen. Therefore, a good nearly feasible solution will return in at most $\frac{\ln(m/\omega^2)}{\sigma_2^2}$ iterations with probability at least $1 - \Phi(\omega)$. □

5.4 Computational Result

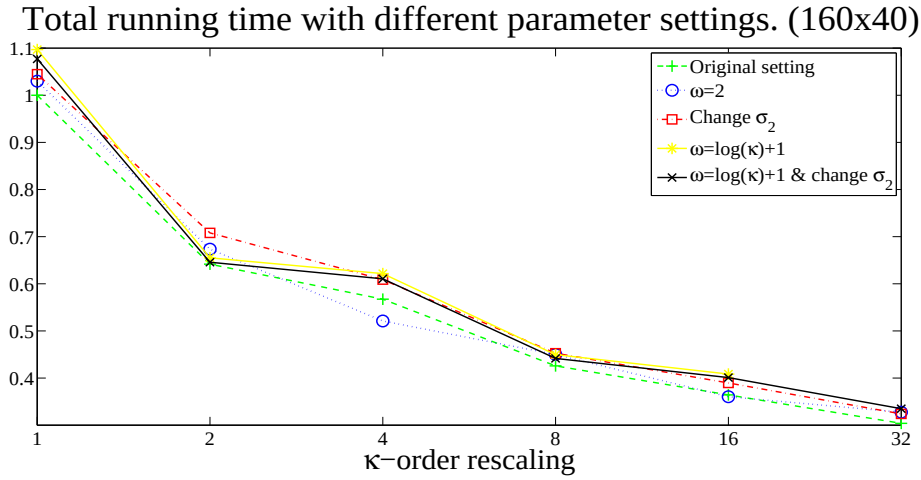
We have implemented the Higher-order Rescaling PA and made some limited computational experiments using MATLAB with Intel® Core™ Duo CPU T6570 2.1GHz, 4GB RAM [34]. The algorithm was applied to two types of problem sets. Problem set I contains simpler problems whose feasible regions are narrow only in one dimension. The problems in set II are more difficult since their feasible regions are narrow along $(n-1)$ dimensions. In Figure 5.1, we show the ratios of the total running time comparing higher-order rescaling variants and the one-order rescaling, with the original parameter setting $\omega = 1$ and $\sigma_2 = \frac{1}{32m}$. Figures 5.1(a) and 5.1(b) show for both types of test problems with different sizes that the running time is reduced significantly. For Figure 5.1, we have also tested the effect of different parameter settings on the total running time. The Higher-order Rescaling PA runs on problem set II with parameter options $\omega = 2$, $\omega = \ln \kappa + 1$, and $\sigma_2 = \frac{1}{32m\sqrt{m}}$. Figure 5.1(c) shows the results on the problems with size $m = 40$ and $n = 160$. With varied parameter settings, we found in Figure 5.1(c) that none of the settings is dominant. The effect of higher-order rescaling is robust, and increasing the rescaling order makes more improvement than changing the parameters.



(a) Problem set I with original parameter setting



(b) Problem set II with original parameter setting



(c) Set II, different parameter settings

Figure 5.1: Improving the running time by using the Higher-order Rescaling Perceptron Algorithm.

5.5 Summary

In this paper we propose a Higher-order Rescaling PA. The power of rescaling is enhanced by utilizing parallel computing in a multi-core environment. Realizing the perceptron improvement phase by parallel processors, we can obtain multiple nearly feasible solutions. By using them in one higher-order rescaling step, we get better rescaling rates when all the other parameters are properly adapted. In addition, the practical running time of solving LFPs is improved as well.

Chapter 6

Conclusions and Future Research

In this final chapter, we review the results of the thesis and highlight future research problems.

The PA and the vNA are two algorithms to solve LFPs. One of the important properties they share is that their computational complexity depends on the geometry of the problems. The idea of Rescaling is to improve the geometry by applying linear a transformation periodically, thereby improve the complexity of algorithms. In Chapter 2, we first explore the duality relationship between the perceptron and the von Neumann algorithms, which is the fundation of our research work. By the duality relationship, variants of the perceptron and the von Neumann algorithms together with their complexity result could be transit to the dual side. However, since all variants of perceptron algorithms assume that perceptron problem is feasible, after being transited to the dual (von Neumann) side, there is no immediate result for feasible von Neumann problems. *Future research problem 1 is to transit all variants of PAs to vNAs to solve both feasible and infeasible problems.*

In Chapter 3, we further apply the duality relationship on the Deterministic Rescaling PA to obtain the corresponding Deterministic Rescaling vNA. Computational experiments show a notable improvment of this rescaling vNA. However, the theoretical complexity is not proved yet. The example we created tells us that proving monotonic

increasing of ρ is not going to work and another method need to be discovered for the purpose of proving the complexity. *Future research problem 2 is to give a complete proof of the complexity of the Deterministic Rescaling vNA. We expect that this rescaling vNA is polynomial-time algorithm.*

In Chapter 4, we combine Chubanov's Method with the rescaling idea to obtain a column-wise rescaling vNA. We have proved that it is a polynomial-time algorithm[1]. In Chapter 5, we take advantage of a good characteristic of the Stochastic Rescaling PA: the perceptron improvement phase always starts with a random vector. The independence of this phase enables us to run the perceptron improvment phase on parallel procesors and obtain multiple rescaling vectors without extra wall-clock time. With multiple rescaling vectors on-hand, we proposed a Higher-order Rescaling PA. Computational experiments shows that the practical efficiency of this higher-order rescaling algorithm has a significant improvment compared to the original one-order Stochastic Rescaling PA (Algorithm 1.4). *Future research problem 3 is to complete the proof of the complexity of the Higher-order Rescaling PA. We expect that it is a polynomial-time algorithm with high probability.*

The largest theoretical impact of our research may be expected from developing rescaling algorithms for the Colorful Feasibility Problem (CFP) [2], that is a combinatorial generalization of problem (1.6). The Bárány and Onn Algorithms proposed by Bárány and Onn [2] are currently the best algorithms used for solving the CFP. However, they are not polynomial time algorithms. It is still an open problem whether the CFP is solvable in polynomial time. Due to the specialization-generalization relationship between the CFP and the LFP, as well as the undecided polynomial-solvability of the CFP, Bárány and Onn [2] marked the CFP "an outstanding problem on the border line between tractable and intractable computational problems".

Exploring the relationships between the vNA (Section 1.2.2) and the Bárány and Onn Algorithms (Section B.2) leads to the recognition of numerous common features:

[1] As far as we know, it is the first variant of vNA with polynomial complexity.

1. they aim to find the representation of a point as a convex combination of certain points;
2. all the given points can be normalized, thus all the iterates are in the unit ball;
3. both of them are first-order line-search algorithms;
4. the solution they return are ϵ-solutions;
5. their complexity depends on the quantity ρ which measures the distance to feasibility or infeasibility;
6. they are based on the analogous algorithmic logic and update procedure.

Thus, the polynomial-time Rescaling vNA proposed in Chapter 4 would highly increase the possibility of designing a polynomial-time rescaling Bárány and Onn algorithm used to solve the CFP and getting closer to answer the question that whether the CFP is polynomial solvable. *Future research problem 4 is to develop rescaling Bárány and Onn algorithms for CFP.*

Bibliography

[1] Shmuel Agmon. The relaxation method for linear inequlities. *Canadian Journal of Mathematics*, 1954.

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

[3] Amir Beck and Marc Teboulle. A conditional gradient method with linear rate of convergence for solving convex linear systems. *Mathematical Methods of Operations Research*, 59:235–247, 2004.

[4] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[5] Avrim Blum and John Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 905–914, 2002.

[6] Avrim Blum, Alan Frieze, Ravi Kannan, and Santosh Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22(1), 1998.

[7] Robert K. Brayton, Stephen W. Director, Gary D. Hachtel, and Luis M. Vidigal. A new algorithm for statistical circuit design based on quasi-newton methods and function splitting. *IEEE Trans. Circuits and Systems*, 26:784–794, 1979.

[8] Dennis Cheung and Felipe Cucker. A new condition number for linear programming. *Mathematical Programming*, 91:163–174, 2001.

[9] Sergei Chubanov. A polynomial relaxation type algorithm for linear programming. `http://www.optimization-online.org/DB_FILE/2011/02/2915.pdf`, 2012.

[10] Sergei Chubanov. A strongly polynomial algorithm for linear systems having a binary solution. *Mathematical Programming*, pages 533–570, 2012.

[11] Sergei Chubanov. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*, pages 1–27, 2014.

[12] George B. Dantzig. Converting a converging algorithm into a polynomially bounded algorithm. Technical Report SOL 91-5, Stanford University, 1991.

[13] George B. Dantzig. Bracketing to speed convergence illustrated on the von Neumann algorithm for finding a feasible solution to a linear program with a convexity constraint. Technical Report SOL 92-6, Stanford University, 1992.

[14] George B. Dantzig. An ϵ-precise feasible solution to a linear program with a convexity constraint in $1/\epsilon^2$ iterations independent of problem size. Technical Report SOL 92-5, Stanford University, 1992.

[15] Tim Davis. http://www.mathworks.com/matlabcentral/fileexchange/24119-don-t-let-that-inv-go-past-your-eyes–to-solve-that-system–factorize-.

[16] Antoine Deza, Sui Huang, Tamon Stephen, and Tamás Terlaky. The colourful feasibility problem. *Discrete Applied Mathematics*, 156:2166–2177, 2008.

[17] John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Mathematical Programming*, 114:101–114, 2008.

[18] Marina A. Epelman and Robert M. Freund. Condition number complexity of an elementary algorithm for computing a reliable solution of a conic linear system. *Mathematical Programming*, 88:451–485, 2000.

[19] Marina A. Epelman and Robert M. Freund. Condition number complexity of an elementary algorithm for resolving a conic linear system. *Mathematical Programming*, 88(3):451–485, 2000.

[20] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[21] Robert M. Freund. Dual gauge programs, with applications to quadratic programming and the minimum-norm problem. *Mathematical Programming*, 38:47–67, 1987.

[22] Robert M. Freund and Jorge R. Vera. Some characterizations and properties of the "distance to ill-posedness" and the condition measure of a conic linear system. *Mathematical Programming*, 86(2):225–260, 1999.

[23] Robert M. Freund and Jorge R. Vera. Equivalence of convex problem geometry and computational complexity in the separation oracle model. *Mathematics of Operations Research*, 34:869–879, 2009.

[24] Peter Gács and László Lovász. Khachian's algorithm for linear programming. *Mathematical Programming Study*, 14:61–68, 1981.

[25] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Numerical Linear Algebra and Optimization, Volume 1.* Addison-Wesley Publishing Company, 1991.

[26] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore and London, third edition, 1996.

[27] João P. M. Gonçalves. *A family of linear programming algorithms based on the von Neumann algorithm.* PhD thesis, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, 2004.

[28] João P. M. Gonçalves, Robert.H. Storer, and Jacek Gondzio. A family of linear programming algorithms based on an algorithm by von Neumann. *Optimization Methods and Software*, 24(3):461–478, 2009.

[29] Sui Huang. Colourful feasibility: algorithms, bounds and implications. Master's thesis, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, 2007.

[30] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[31] Leonid G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[32] Emil Klafszky and Tamás Terlaky. On the ellipsoid method. *Radovi Mathematicki*, 8:269–280, 1992.

[33] Victor Klee and George L. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–179, New York, 1972. Academic Press.

[34] Dan Li. On rescaling algorithms for linear optimization, 2011. Ph.D. Proposal, revised. Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, United States.

[35] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction To Computational Geometry*. MIT Press, 1969.

[36] Theodore S. Motzkin and Isaac J. Schoenberg. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:393–404, 1954.

[37] Javier Peña and Negar Sohèili. A deterministic rescaling perceptron algorithm. 2015. Accepted by Mathematical Programming. Appears online at `http://dx.doi.org/10.1007/s10107-015-0860-y`.

[38] Javier Peña and Negar Sohèili. A deterministic rescaling perceptron algorithm. *Mathematical Programming*, January 2015.

[39] James Renegar. Linear programming, complexity theory and elementary functional analysis. Technical Report 1090, School of Operations Research and Industrial Engineering College of Engineering, Cornell University, 1994.

[40] James Renegar. Some perturbation theory for linear programming. *Mathematical Programming*, 65:73–91, 1994.

[41] Cornelis Roos. An improved version of chubanov's method for solving a homogeneous feasibility problem. `http://www.optimization-online.org/DB_HTML/2015/01/4750.html`.

[42] Cornelis Roos, Tamás Terlaky, and Jean-Philippe Vial. *Interior Point Methods for Linear Optimization*. Springer, 2006.

[43] Frank Rosenblatt. The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[44] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[45] John Shawe-Taylor and Nello Cristianini. *Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[46] Negar Sohèili and Javier Peña. A primal-dual smooth perceptron-von Neumann algorithm. Working paper, Carnegie Mellon University, 2012.

[47] Negar Sohèili and Javier Peña. A smooth perceptron algorithm. *SIAM Journal on Optimization*, 22(2):728–737, 2012.

[48] Michael J. Todd and Yinyu Ye. Approximate Farkas Lemmas and stopping rules for iterative infeasible-point algorithms for linear programming. *Mathematical Programming*, 81:1–21, 1998.

[49] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, 1997.

Appendix A

Reformulation of LO Problems as LFPs

[1] In this section, we present how to reformulate a standard LO problem to a LFP that can be solved using the Higher-order Rescaling PA. In order to have positive ρ for any LO problem, we use the embedding model to ensure that the interior-point condition is satisfied [42]. Given an LO problem in the form of

$$\text{(P)} \quad \min \ \{c^T x : \ Ax \geq b, \ x \geq 0\}, \tag{A.1}$$

where matrix A is of size $m \times n$, the vectors $c, x \in \mathcal{R}^n$, and $b \in \mathcal{R}^m$. The dual problem of (P) is given by

$$\text{(D)} \quad \max \ \{b^T y : \ A^T y \leq c, \ y \geq 0\}, \tag{A.2}$$

[1] In appendix, we may use the same notation appeared in the text for different meanings. Please see the specified details.

where $y \in \mathcal{R}^m$ is the dual vector variable. Define the matrix $\bar{M}$ and M, the vector r and u by

$$\begin{aligned} \bar{M} &= \begin{bmatrix} 0 & A & -b \\ -A^T & 0 & c \\ b^T & -c^T & 0 \end{bmatrix}, \; M = \begin{bmatrix} M & r \\ -r^T & 0 \end{bmatrix}, \\ r &= \begin{bmatrix} e_m - Ae_n + b \\ e_n + A^T e_m - c \\ 1 - b^T e_m + c^T e_n \end{bmatrix}, \; z = \begin{bmatrix} y \\ x \\ \kappa \\ \vartheta \end{bmatrix}, \end{aligned} \tag{A.3}$$

where κ and ϑ are two scalars. Letting $\bar{n} = m + n + 2$ and q be the vector of length $\bar{n}$ given by $q = [0_{\bar{n}-1}^T \; \bar{n}]^T$, we consider the system

$$Mu \geq -q, \; u \geq 0. \tag{A.4}$$

Theorem I.5 in [42] yields the following results. Problems (P) and (D) have optimal solutions with zero duality gap if and only if system (A.4) has a solution with $\vartheta = 0$ and $\kappa > 0$. To reduce system (A.4) to a homogeneous linear system in the form of (1.4), define the matrix $\hat{M}$ and the vector $\hat{u}$ by

$$\hat{M} = \begin{bmatrix} \bar{M} & r & 0 \\ -r^T & 0 & -\bar{n} \\ 0 & -1 & \epsilon \end{bmatrix}, \; \hat{u} = \begin{bmatrix} \bar{u} \\ \vartheta \\ \beta \end{bmatrix}, \tag{A.5}$$

where $\bar{u} = [y^T \; x^T \; \kappa]^T$ and ϵ is a very small positive number, such as e^{-10}. A number is consider to be 0 if its value is not larger than ϵ. Letting $M = [\hat{M}^T \; I_{\bar{n}+1}]^T$, problems (P) and (D) have optimal solutions if and only if system $M\hat{u} \geq 0, \hat{u} \neq 0$ has a solution with $\kappa, \beta > 0$.

We have made some preliminary computational experiments on LFPs which are re-

duced from LO problems. These problems have small dimension not exceeding 10. The results show that by using the the Higher-order Rescaling PA to solve LO problems, we obtain analogous speedup to the one we get in Section 5.4. In spite of the fact that the Higher-order Rescaling PA improves the practical efficiency of the Stochastic Rescaling PA, its running time is not as good as the one of state of the art commercial solvers.

Appendix B

Colorful Feasibility Problem

B.1 Colorful Feasibility Problem

The Colorful Feasibility Problem (CFP) was presented by Bárány and Onn in 1997 [2, 16]. The CFP is a combinatorial generalization of the LFP [2] in the following form: Given $d+1$ colorful sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d+1}$, each containing $d+1$ points, referred to as colored sets, and a point $q \in \mathbb{R}^d$. The CFP is to find a colorful set $\mathcal{V} = \{v_1, v_2, \ldots, v_{d+1}\}$ containing $d+1$ points $v_i \in \mathcal{S}_i$ such that $q \in \text{conv}(\mathcal{V})$, where $\text{conv}(\mathcal{V})$ denotes the convex hull of the point set $\mathcal{V}$.

The core of a CFP is defined as $\bigcap_{i=1}^{d+1} \text{conv}(\mathcal{S}_i)$. If a point q is in the core, then a Colourful Feasiblility Problem is called a Colorful Core Feasibility Problem (CCFP). Without loss of generality, the following assumptions can be made [29].

(a) Point q is the origin, i.e., $q = 0$.

(b) Point $q \notin \mathcal{S}_i$ for all i.

(c) $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for all $i \neq j$.

(d) $\|s\| = 1$ for all $s \in \mathcal{S}_i$ and all i.

For any point q, the problem can be preprocessed by translating the point q to be the origin in $\mathbb{R}^d$. If q is a point in one of the point sets, then the solution of CFP is trivial.

Assumption (c) can be guaranteed by removing duplicates. We normalize all the points of $\mathcal{S}_i$'s so that they lie on the unit sphere in $\mathbb{R}^d$. With $q = 0$, the normalization will not change the feasibility or infeasibility of the problem. Based on these assumptions, we define CFP and CCFP formally.

Definition B.1.1. *[29]* ***Colorful Feasibility Problem.***

Given $d+1$ sets $\mathcal{S}_1, \ldots, \mathcal{S}_{d+1}$ in dimension d such that each size of the set is $d+1$, i.e., $|\mathcal{S}_i| = d+1$ for all i, decide if there is a set of points $\mathcal{V} = \{v_1, \ldots, v_{d+1}\}$ such that $v_i \in \mathcal{S}_i$ for each $i \in \{1, \cdots, d+1\}$ and $0 \in \text{conv}(\mathcal{V})$.

Definition B.1.2. *[29]* ***Colorful Core Feasibility Problem.***

Colourful Core Feasiblility Problem with point 0 in the core of the Colorful Feasibility Problem, i.e., $0 \in \bigcap_{i=1}^{d+1} \text{conv}(\mathcal{S}_i)$.

Observe that the monochrome version of a CFP, setting $\mathcal{S} = \mathcal{S}_1 = \cdots = \mathcal{S}_{d+1}$, is reduced to decide whether the point q is in the convex hull of the point set $\mathcal{S}$, and if it is, find one convex combination. This is a classical LFP [4]. Furthermore, with the assumptions made in [29], the monochrome version of a CFP defined in Definition B.1.1 is a problem (3.2) with $m = d$ and $n = d+1$, which can be solved using vNAs.

B.2 The Bárány and Onn Algorithms

The Bárány and Onn Algorithms (BOAs) proposed by Bárány and Onn [2] are currently the best algorithms used for solving the CFP. In this section, we introduce the BOAs and their complexity result.

Let ${}^b\rho$ denote the specialized quantity ρ in the Bárány-Onn algorithms: the radius of the largest ball contained in the core and centered at the origin 0, i.e., the minimal distance between 0 and the boundary of the core. The first BOA has the following complexity result [2].

Theorem B.2.1. *[2] Let $\epsilon > 0$. After at most $O(\frac{1}{{}^b\rho^2} \ln \frac{1}{\epsilon})$ iterations if ${}^b\rho > 0$, or $O(\frac{1}{\epsilon^2})$*

Algorithm B.1 The First Bárány-Onn Algorithm

initialization
  Pick any arbitrary colorful set $\mathcal{V}^1 = \{v_1, \ldots, v_{d+1}\}$ such that $v_i \in \mathcal{S}_i$ for all i.
  Let $w^1 = \text{argmin}_{v \in \text{conv}(\mathcal{V}^1)} \|v\|$, and $k = 1$.
while $\|x^k\| \leq \epsilon$ **do**
  Find a color i such that $w^k \in \text{conv}(\mathcal{V}^k \backslash \{v_i\})$.
  Set $v_i = \text{argmin}_{v \in \mathcal{S}_i}(v^T w^k)$, and update $\mathcal{V}^{k+1}$.
  Let $w^{k+1} = \text{argmin}_{v \in \text{conv}(\mathcal{V}^{k+1})} \|v\|$, and $k = k + 1$.
end while
output
  A set $\mathcal{V}^{k+1} = \{v_1, \ldots, v_{d+1}\}$ such that $\text{conv}(\mathcal{V})$ is ϵ-close to 0;
  And a point w^{k+1} which is within distance ϵ from the origin.

iterations if ${}^b\rho = 0$*, the Bárány-Onn Algorithm gets* $\text{conv}(\mathcal{V})$ ϵ*-close to* 0*, i.e., finds a vector* w *such that* $w \in \text{conv}(\mathcal{V})$ *and* $\|w\| \leq \epsilon$*.*

Theorem B.2.1 shows that the quantity ${}^b\rho$ plays an essential role in the complexity of the Bárány-Onn algorithm. The algorithm runs in polynomial time in the $1/{}^b\rho$. Since $\text{conv}(\mathcal{V})$ is warranted to be closer to 0 after each iteration, the first BOA does not visit the same set $\mathcal{V}$ twice which is an advantage of it. However, computing the nearest point w^{k+1} at each iteration is the complexity bottleneck of the algorithm. It involves a convex quadratic optimization problem which can be solved only approximately. Solving to an ϵ-precision solution needs a number of arithmetic operations polynomial in d and $\ln \frac{1}{\epsilon}$. Thus, the arithmetic operations complexity of Algorithm B.1 for getting $\text{conv}(\mathcal{V})$ ϵ-close to 0 is $O(\frac{d^{3.5}}{{}^b\rho^2} \ln \frac{1}{\epsilon} \ln \frac{1}{\epsilon})$ if ${}^b\rho > 0$ or $O(\frac{d^{3.5}}{\epsilon^2} \ln \frac{1}{\epsilon})$ if ${}^b\rho = 0$ [29]. Motivated by this disadvantage, Bárány and Onn proposed an alternative algorithm for the CFP which only involves linear algebra.

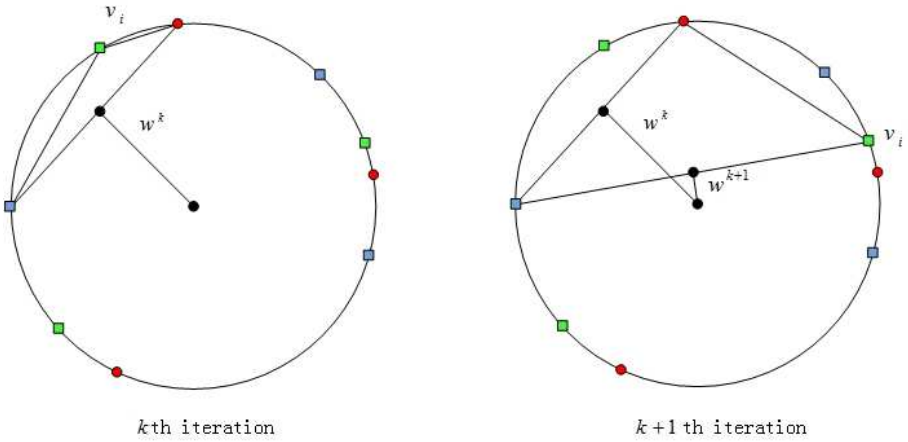
Algorithm B.2 avoids computing the point with minimum norm in $\text{conv}(\mathcal{V})$. Thus, it only needs $O(d^4)$ arithmetic operations at each iteration. The iteration complexity result for Algorithm B.2 is the same as for Algorithm B.1. Comparing these two BOAs, thought Algorithm B.2 is faster at each iteration than Algorithm B.1, it may revisit the same set $\mathcal{V}$ more than once, which is called oscillations in [29].

Figure B.1 illustrates the two BOAs. Both of the iteration complexity is not poly-

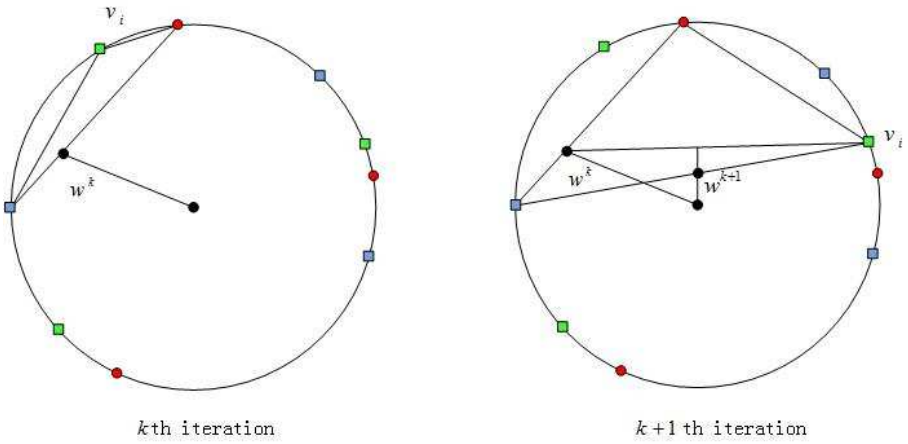
Algorithm B.2 The Second Bárány-Onn Algorithm

initialization
  Pick any arbitrary colorful set $\mathcal{V}^1 = \{v_1, \ldots, v_{d+1}\}$ such that $v_i \in \mathcal{S}_i$ for all i.
  Let $w^1 = v_1$, and $k = 1$.
while $\|w^k\| \leq \epsilon$ **do**
  Find a color i such that $w^k \in \text{conv}(\mathcal{V}^k \backslash \{v_i\})$.
  Set $v_i = \text{argmin}_{v \in \mathcal{S}_i}(v^T w^k)$, and update $\mathcal{V}^{k+1}$.
  Let w^{k+1} be the projection point of 0 onto the line segment $[w^k, v_i]$, i.e., $w^{k+1} = \text{proj}_{[w^k, v_i]}(0)$.
  Compute $\beta_{\min}$ such that $\beta_{\min} = \min\{\beta : \beta w^k \in \text{conv}(\mathcal{V}^{k+1})\}$.
  Let $w^{k+1} = \beta_{\min} w^k$ and $k = k + 1$.
end while
output
  A set $\mathcal{V}^{k+1} = \{v_1, \ldots, v_{d+1}\}$ such that $\text{conv}(\mathcal{V})$ is ϵ-close to 0;
  And a point x^{k+1} which is within distance ϵ from the origin.

nomial as ${}^b\rho$ can be exponentially small in the input length L of the system [24]. Recall that when $\mathcal{S} = \mathcal{S}_1 = \cdots = \mathcal{S}_{d+1}$, the CFP reduces to a LFP; and in this case, the BOAs specializes to the vNA. Furthermore, observe that the iteration bounds in Theorem B.2.1 are analogous to the bounds known for the vNA (see Theorem 1.2.7).



(a) First Bárány-Onn Algorithm



(b) Second Bárány-Onn Algorithm

Figure B.1: Illustration of the Bárány-Onn Algorithms.

Biography

Dan Li was born in Beijing, China in 1983. She received her B.S. and M.S. degrees in Control Science and Engineering in 2005 and 2007, from Department of Automation in Tsinghua University in China. Her master's thesis was on air traffic management system and conflict detection. She joined Lehigh University to pursue her doctoral degree. Her research focuses on designing algorithms for linear optimization and their complexity. She also has experience in different industry areas such as government planning and management, pharmaceuticals, airlines, and information technology. She is a Rossin Doctoral Fellow and has held the Gotshall fellowship for one semester at Lehigh. She is a member of Institute for Operations Research and the Management Sciences (INFORMS) and has served as the president of Lehigh INFORMS Student Chapter for one year.