Anti-Cycling Pivot Rules in Linear Optimization

Filiz Bilen

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Applied Mathematics and Computer Science

Eastern Mediterranean University
February 2023
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Applied Mathematics and Computer Science.

Prof. Dr. Nazim Mahmudov
Chair, Department of Mathematics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Applied Mathematics and Computer Science.

Prof. Dr. Tibor Illés
Co-Supervisor

Prof. Dr. Nazim Mahmudov
Supervisor

Examining Committee

1. Prof. Dr. Tibor Illés
2. Prof. Dr. Nazim Mahmudov
3. Prof. Dr. Hayri Sever
4. Prof. Dr. Mehmet Reşit Tolun
5. Prof. Dr. Sonuç Zorlu Oğurlu
6. Assoc. Prof. Dr. Arif Akkeleş
7. Asst. Prof. Dr. Mehmet Ali Tut

ABSTRACT

Pivot algorithms for solving linear optimization problems traverse the set of basic solutions or bases of the inequality system describing the model, searching for a feasible solution, or an optimal feasible solution if the model also contains a cost function to be minimized or maximized. Feasibility preserving pivot algorithms for solving linear optimization problems, often called simplex-type methods, preserve primal feasibility while trying to achieve dual feasibility or vice versa. Monotonic Build Up algorithm of Anstreicher and Terlaky is a simplex-type algorithm with interesting properties. We develop a pivot algorithm with similar properties for solving the feasibility problem of linear optimization in particular. To guarantee finiteness of our Monotonic Build Up simplex algorithm we incorporate s-monotone index selection rules into the general framework of the algorithm which are to be utilized whenever there is competition among the basic variables to leave the basis and among the nonbasic variables to enter the basis. We also use a specialized recursive procedure for handling strongly degenerate bases. We prove finiteness of the algorithm and analyze its computational complexity under some assumptions. As opposed to simplex-type methods criss-cross pivot algorithm preserves neither primal nor dual feasibility. We use criss-cross algorithm together with s-monotone index selection rules to solve feasibility problem of oriented matroids and and also prove its finiteness.

Keywords: pivot algorithms, finiteness, oriented matroids, feasibility

ÖZ

Doğrusal optimizasyon problemlerini çözmek için pivot algoritmaları, modeli açıklayan eşitsizlik sisteminin baz çözümleri veya bazları arasında gezinerek uygun bir çözüm ararlar. Model aynı zamanda minimize veya maksimize edilmesi gereken bir maliyet fonksiyonu içeriyorsa bu kez en uygun çözümü ararlar. Genellikle simpleks tipinde yöntemler olarak adlandırılan doğrusal optimizasyon problemlerini çözmek için fizibiliteyi koruyan pivot algoritmaları, dual fizibilite elde etmeye çalışırken primal fizibiliteyi korur (primal simpleks) veya bunun tersi de geçerlidir (dual simpleks). Anstreicher ve Terlaky'nin Monotonic Build Up (monoton inşa) algoritması ilginç özelliklere sahip simpleks tipi bir algoritmadır. Özellikle doğrusal optimizasyonun fizibilite problemini çözmek için benzer özelliklere sahip bir pivot algoritması geliştirdik. Monoton inşa simpleks algoritmamızın sonluluğunu garanti etmek için, temel değişkenler arasında bazdan ayrılmak ve temel olmayan değişkenler arasında baza girmek için rekabet olduğunda kullanılacak olan s-monoton dizin seçim kurallarını algoritmanın genel çerçevesine dahil ederek tanımladık. Ayrıca, güçlü bir şekilde dejenere olmuş bazları işlemek için özel bir özyinelemeli prosedür geliştirdik. Algoritmanın sonluluğunu kanıtladık ve hesaplama karmaşıklığını bazı varsayımlar altında analiz ettik. Simpleks tipi yöntemlerin aksine, criss-cross pivot algoritması ne primal ne de dual fizibiliteyi korur. Yönlendirilmiş matroidlerin fizibilite problemini çözmek ve sonluluğunu kanıtlamak için s-monoton indeks seçim kuralları ile birlikte criss-cross algoritmasını kullanıyoruz.

Anahtar Kelimeler: pivot algoritmaları, sonluluk, yönlendirilmiş matroidler, fizibilite

To Cemile and Havva

ACKNOWLEDGMENTS

This has been a long journey and I have many people to thank. First and foremost, my supervisor Prof. Dr. Tibor Illés, thank you for your continuous support and not giving up on me even at the times when I've given up on myself. I am ever so grateful to have met you and have had the chance to learn linear optimization from you.

To all the great teachers I was fortunate to cross paths with at Geçitkale Primary School and 20 Temmuz and Northeast High Schools, and at Eastern Mediterranean University, thank you. My dear teachers Daoud S. Daoud and Peter Kas, thank you.

I would like to express my appreciation to my supervisor Prof. Dr. Nazim Mahmudov for his support during my studies.

My grandmother Cemile, my mother Havva, my father Hüseyin, my sister Yeliz and my nieces Derin and Deniz; I am so grateful to have your support and love. My dear Pervin and Mayra, thank you for your love, support, and all the sacrifice for trying to keep quiet while I was working.

Last but not not least, I thank to my friends for their companionship, genuine support, long talks during long walks and many creative motivational tactics.

During the last stages of writing my thesis I lost a very good friend. I would like to dedicate this thesis in memory of Rıza Tuncel who will always be greatly missed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BFS	Basic Feaible Solution
FS	Feasible Subsystem
IIS	Irreducibly Infeasible Subsystem
LFP	Linear Feasibility Problem
LIFO/LOFI	Last In First Out/Last Out First In
LOP	Linear Optimization Problem
MAX-FS	Maximum Feasible Subsystem
MBU	Monotonic Build Up
MOSV	Most Often Selected Variable
OM	Oriented Matroid
OMFP	Oriented Matroid Feasibility Problem
SIV	Smallest Indexed Variable

Chapter 1

INTRODUCTION

Linear optimization, which has its roots in the study of linear inequalities dating as far back as 1826 to the work of Fourier, has been a prominent field of study right from its inception in the mid nineteen forties. Its applicability to solve a wide range of problems coming from many diverse fields has contributed to its status as one of the mostly studied research subjects as well as one of the mostly utilized models to solve real world problems. There are different types of methods for solving linear optimization problems. The first practically useful method is the simplex method developed by George Dantzig in 1947 [19, 20]. Simplex method belongs to the general category of solution techniques called pivot methods. Many pivot methods have been developed in the quest for finding more efficient solution algorithms for solving the linear optimization problem. Some of these are variants of the simplex method because they maintain primal (or dual) feasibility during the solution procedure. Original simplex method and its variants seek for an optimal solution in the set of feasible solutions (which is a convex polyhedron) by travelling the vertices or extreme points of the feasible region. Other type of pivot methods called exterior point methods walk through a path to the optimal vertex from the exterior of the feasible region using infeasible bases. Criss-cross method combines the two paradigms and searches for the optimal solution by visiting both feasible and infeasible bases along the way. The total number of vertices of the feasible region can be as large as $\binom{n}{m}$ where n is the number of variables (dimension of the solution

space) and m is the number of constraints. Since this number can be huge, the brute force approach of enumerating all the vertices and solving the optimization problem by searching for an optimal vertex is not only inefficient but not a viable option for even moderately sized problems.

Linear optimization was shown to belong to the class of polynomially solvable problems in 1979 when ellipsoid method was formulated for solving linear optimization problems by Khachian [31]. The ellipsoid method, possessing good theoretical properties, is not a practically efficient method so it is not useful in practice.

The second main category of solution methods widely used in practice, the interior point methods, find an approximation to optimum solution by following a path in the interior of the feasible region. Interior point methods, first introduced in 1984 by Karmarkar [30] are not only practically efficient but also have polynomial complexity.

Linear optimization deals with the problem of finding the optimum (or optimal) value of a linear function subject to linear equality or inequality constraints.

Fundamental to solving a linear optimization problem is finding a feasible solution or showing that none exists. The problem of finding a feasible solution precedes that of finding an optimal solution simply because if there is no feasible solution, then there is no optimal solution.

The *linear feasibility problem* (LFP) differs from LOP in that it has no objective function to be minimized or maximized. We give formal definitons of both problems

in the next chapter.

Although linear optimization belongs to the class of polynomially solvable problems, we do not yet know whether there is a variant of simplex algorithm which has polynomial complexity. This open problem and the strong belief that there exists a variant of the simplex algorithm which is strongly polynomial are the main motivations for studying pivot-based algorithms.

In the context of oriented matroids it is possible to study the problem of solving a system of linear inequalities as a purely combinatorial one, thereby clearing off the problem from the difficulties which arise when its arithmetic properties are considered. Providing an abstraction based only on sign structures, oriented matroids give a geometric language for pivot algorithms. In fact, most combinatorial pivot rules are originally developed in the context of oriented matroid programming [46].

When it comes to solving practical linear optimization problems arising from real life applications, variants of the simplex method and interior-point methods dominate the field. In this thesis we study monotonic build up simplex method and criss-cross method with different index selection rules which lead to finite algorithms for solving the linear feasibility and the oriented matroid feasibility problems.

Chapter 3 is devoted to linear feasibility problem. We explain a simplex type algorithm in which the number of feasible variables is monotonically built up and we give a complexity analysis for this algorithm. Furthermore, we show that the algorithm is finite by utilizing a recursive procedure when a sequence of strongly degenerate bases is encountered. Finiteness is also established using anti-cycling

rules which are categorized as s-monotone. In this chapter, we also give a general overview of methods for analyzing infeasibility by finding irreducible infeasible subsystems of a given infeasible problem.

In Chapter 4 the linear feasibility problem is viewed in an abstract setting using oriented matroids. A constructive proof of the Farkas lemma is given using pivot algorithms which solve the oriented matroid feasibility problem in a finite number of steps. We use s-monotone pivot rules with criss-cross algorithm for solving oriented matroid feasibility problem and prove that criss-cross with some specific s-monotone rules does not cycle.

Chapter 5 suggests some possible further research problems related to the ones studied.

Chapter 2

PRELIMINARIES

In this chapter we present the necessary mathematical foundations required to follow the thesis. These include basic definitions and theorems from linear algebra (section 2.4), the most basic definitions necessary for a general geometric understanding of linear inequality systems and pivot based methods (section 2.2) and a short introduction to matroids and oriented matroids (section 2.7) which will be necessary to follow chapter 4. The rest of the chapter is devoted to the relevant fundamental concepts in linear optimization.

2.1 Notation and Problems of Interest

First we fix the notation that we use throughout the thesis and then state the main problems of interest.

Vectors are denoted by lowercase boldface letters and sets and matrices with uppercase letters. All vectors are assumed to be column vectors. We indicate that a vector is a row vector by writing it as the transpose of another (column) vector if it is not clear from the context. $\mathbf{e}_j$ denotes the unit vector whose jth component is 1, and all other components are zeroes, and $\mathbf{e}$ denotes the vector of all ones.

Index sets are denoted by upper case caligraphic letters. Given that $\mathcal{M}$ is the row index set and $\mathcal{N}$ the column index set of a matrix A, for $\mathcal{I} \subseteq \mathcal{M}$ and $\mathcal{J} \subseteq \mathcal{N}$, $A_{\mathcal{I}\mathcal{J}}$ denotes the submatrix of A induced by those rows and columns of A whose indices belong to $\mathcal{I}$ and $\mathcal{J}$, respectively. $A_{\cdot\mathcal{J}}$ and $A_{\mathcal{I}\cdot}$ mean $A_{\mathcal{M}\mathcal{J}}$ and $A_{\mathcal{I}\mathcal{N}}$ respectively. Likewise, for a

vector $\mathbf{x}$ indexed by $\mathcal{N} = \{1,\ldots,n\}$, $\mathbf{x}_{\mathcal{J}}$ denotes the subvector of $\mathbf{x}$ indexed by $\mathcal{J}$ for any $\mathcal{J} \subseteq \mathcal{N}$. For a matrix $A \in \mathbb{R}^{mxn}$, $A_{i.}$ denotes the row of A indexed by i, and $A_{.j}$ or alternatively $\mathbf{a}_j$ denote the column of A indexed by j.

The cardinality of a set X is denoted by $|X|$.

A *linear optimization problem* (LOP) is about finding the minimum or maximum value of a linear objective function subject to a set of linear equations and inequalities, called the *constraints* of the problem. A *linear feasibility problem* (LFP) is also a search problem in which there is no explicit objective function and the goal is to find *any* point satisfying all the constraints which are also linear. The LFP model is simply a specialized formulation of the LOP model.

To be precise, a linear optimization problem (LOP) has the (standard) form

$$\text{minimize} \quad \mathbf{c}^T\mathbf{x} \tag{2.1}$$

$$\text{subject to} \quad A\mathbf{x} = \mathbf{b}, \tag{2.2}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{2.3}$$

and, a linear feasibility problem (LFP) has the (standard) form

$$A\mathbf{x} = \mathbf{b}, \tag{2.4}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{2.5}$$

where A is an $m \times n$ real matrix, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$.

$A\mathbf{x} = \mathbf{b}$ is a compact way of representing the m equality constraints

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1,\ldots,m$$

in n variables $x_1, x_2, \ldots, x_n$. In a general linear optimization problem, constraints can be of $\leq$ or $\geq$ type, variables can be unbounded or they could each have its own bounds, lower and/or upper. All the possible variations of a linear optimization problem can be transformed into the standard form, therefore we base all our discussion on the standard form problems. Furthermore, we can assume without loss of generality that the constraint matrix A has full row rank m because the feasibility check of the linear system and the removal of redundant equations can be done simultaneously with the Gauss-Jordan elimination.

An algorithm which is designed to solve a linear optimization problem can be used to solve a linear feasibility problem and vice versa. For example, given an optimization problem we can solve it by solving successive feasibility problems by imposing weaker constraints on the objective until feasibility is obtained. In this sense the two problems are equivalent. Although we will be mainly focusing on algorithms particularly designed for solving the linear feasibility problem, the most fundamental parts of the theoretical background of linear optimization will be reviewed in this chapter in order to have a more unified presentation. In the next section we give the most relevant and insightful results regarding the geometry of linear optimization.

We also study the feasibility problem in an abstract combinatorial setting called oriented matroids which is a certain system of sign vectors. Simply stated, the *oriented matroid feasibility problem* (OMFP) is finding a feasible circuit or cocircuit in an oriented matroid or its dual, respectively. For the necessary background, see section 2.7 and for a finite algorithm for finding a feasible circuit or cocircuit, see chapter 4.

2.2 The Geometry of Linear Optimization

Definition 2.1: Let **a** be a nonzero vector in $\mathbb{R}^n$ and let b be a scalar.

a) The set $H = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T\mathbf{x} = b\}$ is called a **hyperplane**.

b) The set $H_{\leq} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T\mathbf{x} \leq b\}$ is called a **halfspace**.

Definition 2.2: A **polyhedron** P is a set that can be described in the form

$$P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$$

where A is an $m \times n$ real matrix and **b** is a vector in $\mathbb{R}^m$.

Hyperplanes and halfspaces are polyhedra determined by a single constraint.

The feasible set of any linear optimization problem can be described by inequality constraints of the form $A\mathbf{x} \leq \mathbf{b}$, and is therefore a polyhedron. In particular, a set of the form $F = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$ is also a polyhedron.

Given two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $0 \leq \lambda \leq 1$, any point of the form $\mathbf{z} = \lambda\mathbf{x} + (1-\lambda)\mathbf{y}$ is called a convex combination of **x** and **y**.

Definition 2.3: Let P be a polyhedron. A vector $\mathbf{x} \in P$ is an **extreme point** or **vertex** of P if we cannot find two vectors $\mathbf{y}, \mathbf{z} \in P$, both different from **x**, and a scalar $\lambda \in [0,1]$, such that $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$.

By this definition, an extreme point of a polyhedron is a point that cannot be expressed as a convex combination of two other points of the polyhedron.

When the simplex method is used, the search for optimal solutions to linear

optimization problems is restricted to the set of vertices of the feasible region. Although the number of basic feasible solutions (algebraic equivalent of vertices, see section 2.3) of a linear optimization problem, therefore, the number of vertices of the corresponding polyhedron is finite, this number can be very large, ruling out a brute force approach for solving general linear optimization problems.

An inequality type constraint is called *active* whenever it is satisfied at equality. For example, a constraint in the form $A_{i.}\mathbf{x} \leq b_i$ is active at the solution vector $\mathbf{x}_0$ if $A_{i.}\mathbf{x}_0 = b_i$.

Two vertices $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ of a polyhedron are called *adjacent* if the line segment joining $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ is an edge of the polyhedron. Two adjacent vertices of the polyhedral feasible region differ in at least one active constraint.

2.3 Basic Feasible Solutions

In this section we give an algebraic characterization of vertices of the feasible region of a linear optimization problem. Although a geometric understanding is necessary and might be more intuitive, an algebraic description is certainly necessary for computational purposes.

Consider the polyhedron $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$.

Let A_B be an $m \times m$ nonsingular submatrix of the matrix A, that is, A_B is a basis for the column space of A. By using the decompositions $A = [A_B \;\; A_N]$ and $\mathbf{x} = [\mathbf{x}_B \;\; \mathbf{x}_N]$, we can rewrite the equation $A\mathbf{x} = \mathbf{b}$ in the form $A_B\mathbf{x}_B + A_N\mathbf{x}_N = \mathbf{b}$. Then, the (general) solution of the system of linear equations is

$$\mathbf{x}_N = \mathbf{u} \in \mathbb{R}^{n-m} \quad \text{and} \quad \mathbf{x}_B = A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{u} \in \mathbb{R}^m.$$

The above general solution expresses the basic variables $\mathbf{x}_B$ in terms of the nonbasic variables $\mathbf{x}_N$. A particular solution with $\mathbf{x}_N = 0$ and $\mathbf{x}_B = A_B^{-1}\mathbf{b}$ is called a *basic* solution. If, furthermore, $\mathbf{x}_B \geq \mathbf{0}$ then it is called a *basic feasible solution* (a BFS). In the decomposition of $\mathbf{x} = [\mathbf{x}_B \ \mathbf{x}_N]$, $\mathbf{x}_B$ are called the basic variables and $\mathbf{x}_N$ the nonbasic variables. Likewise, A_B are the basic columns and A_N are nonbasic columns of A. Every different basis of A captures a unique basic solution in the way defined above but the same basic solution could correspond to multiple bases. Speaking in terms of vertices; a vertex of P lies at the intersection of at least n hyperplanes, that is, at least n of the $m+n$ equations and nonnegativity constraints defining P are active at a vertex of P. If there exists exactly n active constraints at each vertex then the problem is not degenerate. In other words, if every vertex of an n-polyhedron belongs to exactly n edges then the associated linear optimization problem is non-degenerate. Polyhedra corresponding to non-degenerate LOPs are called simple. The real problem with degeneracy is the prospect of repeating the same sequence (this phenomenon is called *cycling*) of bases infinitely many times when solving the problem with a pivot based algorithm like the simplex method.

The following theorem states the condition for a LOP or LFP to have basic feasible solutions.

Theorem 2.1: If the system of constraints $A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ has a feasible solution, then it also has a basic feasible solution.

The simplex method and its variants basically search for a solution in the set of all BFSs. In the next section we explain pivoting operation with some examples and state related theorems.

2.4 Pivoting

Pivoting is the basic operation used in the algorithms studied in this thesis. Pivoting, as we are familiar with from elimination methods such as Gauss-Jordan elimination for solving systems of linear equations, is an operation which transforms a given linear system into an equivalent one which is ideally easier to solve so that the transformation takes us closer to solving the system. In the case of linear feasibility problems and linear optimization problems, a pivot operation will move us from one basic solution to another, or in geometric terms, from one vertex to a neighboring one.

We define pivoting in a general way using generating tableaux as it was done in [34]. In the cited paper, Klafszky and Terlaky use pivoting as a tool to give constructive proofs for some fundamental theorems of linear algebra.

Let $\{\mathbf{a}_1,\ldots,\mathbf{a}_n\} \subset \mathbb{R}^m$ be a set of arbitrary real vectors indexed by $\mathcal{J} = \{1,\ldots,n\}$ and let $\mathcal{J}_G \subset \mathcal{J}$ denote a spanning set (or a generating system) for this set of vectors. Any vector $\mathbf{a}_j, j \in \bar{\mathcal{J}}_G = \mathcal{J} \setminus \mathcal{J}_G$ can be written as a linear combination of the vectors indexed by $\mathcal{J}_G$ as

$$\mathbf{a}_j = \sum_{i \in \mathcal{J}_G} t_{ij}\mathbf{a}_i$$

We can record all the dependence data with respect to a given generating set into a tableau T as shown in figure 2.1.

It is apparent that the pivot tableau describes how any vector $\mathbf{a}_i$ for $i \in \mathcal{J}$ can be expressed as a linear combination of the vectors in $\mathcal{J}_G$. If we restrict the tableau to represent only vectors outside the generating system ($\bar{\mathcal{J}}_G = \mathcal{J} \setminus \mathcal{J}_G$) in terms of the vectors in the generating system we obtain the *short pivot tableau*. In the following we

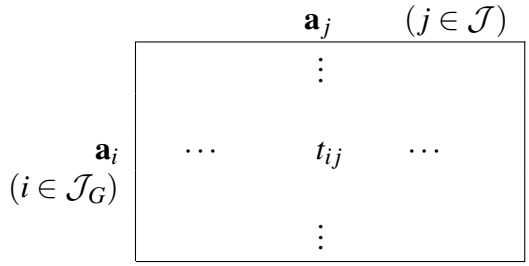

Figure 2.1: Tableau representation of vectors indexed by $\mathcal{J}$ with respect to the generating system $\mathcal{J}_G$.

provide a simple example.

Example 2.1: Consider the following set of vectors $\{\mathbf{a}_1,\ldots,\mathbf{a}_6\} \subset \mathbb{R}^3$: $\mathbf{a}_1 = (1,0,0), \mathbf{a}_2 = (0,1,0), \mathbf{a}_3 = (0,0,1), \mathbf{a}_4 = (5,2,-1), \mathbf{a}_5 = (4,0,1), \mathbf{a}_6 = (-2,1,3)$. $\mathcal{J}_G = \{1,2,3,4\}$ is a generating system for this set of vectors. A possible representation of $\mathbf{a}_5$ and $\mathbf{a}_6$ with respect to $\mathcal{J}_G$ are as follows:

$\mathbf{a}_5 = (4,0,1) = 9(1,0,0) + 2(0,1,0) - (5,2,-1) = 9\mathbf{a}_1 + 2\mathbf{a}_2 - \mathbf{a}_4$ and

$\mathbf{a}_6 = (-2,1,3) = -2(1,0,0) + 1(0,1,0) + 3(0,0,1) = -2\mathbf{a}_1 + \mathbf{a}_2 + 3\mathbf{a}_3$.

Note that neither of these representations is unique. The pivot tableau corresponding to $\mathcal{J}_G$ is obtained by collecting the coefficients of each vector's representation in a tabular form as described above.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$
$\mathbf{a}_1$	1	0	0	0	9	-2
$\mathbf{a}_2$	0	1	0	0	2	1
$\mathbf{a}_3$	0	0	1	0	0	3
$\mathbf{a}_4$	0	0	0	1	-1	0

If we exclude the first four columns from this tableau we get the short pivot tableau corresponding to $\mathcal{J}_G$.

The main computational tools one uses when working with pivot tableaux are the elementary row operations and pivot operation (which consists of a series of elementary row operations). A pivot operation (or a pivot step) is specified by selecting a pivot row $r \in \mathcal{J}_G$ and a pivot column $s \in \bar{\mathcal{J}}_G$, therefore, at the intersection of these a pivot element t_{rs}, which should be nonzero. A pivot operation where row r is chosen as the pivot row and column s is chosen as the pivot column updates the pivot tableau as follows:

Pivot Operation If $t_{rs} \neq 0$ then $\mathbf{a}_r, r \in \mathcal{J}_G$ can be exchanged with $\mathbf{a}_s, s \in \bar{\mathcal{J}}_G = \mathcal{J} \setminus \mathcal{J}_G$ in the following way:

(1) $t'_{ij} = t_{ij} - \frac{t_{rj}t_{is}}{t_{rs}} \qquad i \neq s, j \neq r, i \in \mathcal{J}_{G'}, j \in \bar{\mathcal{J}}_{G'}$

(2) $t'_{sj} = \frac{t_{rj}}{t_{rs}} \qquad j \neq r, j \in \bar{\mathcal{J}}_{G'}$

(3) $t'_{ir} = -\frac{t_{is}}{t_{rs}} \qquad i \in \mathcal{J}_{G'} \setminus \{s\}$

(4) $t'_{sr} = \frac{1}{t_{rs}}$

where $\mathcal{J}'_G = \mathcal{J}_G \setminus \{r\} \cup \{s\}$ is the index set of the new generating system and the scalars are t'_{ij} in the expressions.

See the following example for an illustration of a pivot step.

Example 2.2: Let us take the pivot tableau of the previous example:

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$
$\mathbf{a}_1$	1	0	0	0	9	-2
$\mathbf{a}_2$	0	1	0	0	2	1
$\mathbf{a}_3$	0	0	1	0	0	3
$\mathbf{a}_4$	0	0	0	1	-1	0

A pivot operation can be performed on any nonzero entry of this tableau. As an illustration, let us consider the element on row 3 (pivot row) and column 6 (pivot column) as the pivot element. The first row operation is to multiply row 3 by $\frac{1}{3}$. This operation is performed to obtain a 1 in the pivot column and pivot row and gives us the following modified tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$
1	0	0	0	4	-2
0	1	0	0	0	1
0	0	$\frac{1}{3}$	0	$\frac{1}{3}$	1
0	0	0	1	0	0

The next two row operations are performed to transform the nonzero entries that belong to the pivot column and not the pivot row to zero. The tableau after the pivot operation becomes:

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$
$\mathbf{a}_1$	1	0	$\frac{2}{3}$	0	9	0
$\mathbf{a}_2$	0	1	$-\frac{1}{3}$	0	2	0
$\mathbf{a}_6$	0	0	$\frac{1}{3}$	0	0	1
$\mathbf{a}_4$	0	0	0	1	-1	0

The pivot operation on (r,s) can be interpreted on the full pivot tableau as follows: the pivot operation consists of a series of row operations to convert the pivot column into a unit vector with entry 1 in the pivot row.

Definition 2.4: A set of vectors $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k\} \subset \mathbb{R}^m$ is called linearly independent if $c_1\mathbf{a}_1 + c_2\mathbf{a}_2 + \ldots + c_k\mathbf{a}_k = 0$ has the unique solution $c_1 = c_2 = \ldots = c_k = 0$, where $c_i \in \mathbb{R}$ for $i = 1, \ldots, k$.

Definition 2.5: The set of vectors $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k\}$ is called a basis of $\mathbb{R}^m$ if it is a minimal generating system for $\mathbb{R}^m$. (It can be shown that this is equivalent to $k = m$ and $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are linearly independent)

The relation between generating and independent systems can easily be illustrated using pivoting. This is the essence of the proof of the Steinitz's theorem.

Theorem 2.2: (Steinitz) If $\mathcal{J}_F \subset \mathcal{J}$ is an independent and $\mathcal{J}_G \subset \mathcal{J}$ is a generating system, then $|\mathcal{J}_F| \leq |\mathcal{J}_G|$.

In the following we state the orthogonality theorem which is an important tool used in finiteness proofs of pivot based algorithms. First, we give some necessary definitions.

Definition 2.6: The inner (or dot) product of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$ is a real number defined as

$$\mathbf{a}^T\mathbf{b} = \sum_{j=1}^{m} a_j b_j.$$

Definition 2.7: Two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$ are called *orthogonal* if $\mathbf{a}^T\mathbf{b} = 0$.

Let us call a row of the full pivot tableau corresponding to $i \in \mathcal{J}_B$ as $\mathbf{t}^{(i)}$. That is,

$$\mathbf{t}^{(i)} = \begin{cases} t_{ik}, & \text{if } k \in \bar{\mathcal{J}}_B \\ 1, & \text{if } k = i \\ 0, & \text{if } k \in \mathcal{J}_B, \ \ k \neq i. \end{cases}$$

The vector $\mathbf{t}_j$ corresponding to a nonbasic index $j \in \bar{\mathcal{J}}_B$ is an extension of a column of the pivot tableau.

$$\mathbf{t}_j = \begin{cases} t_{kj}, & \text{if } k \in \bar{\mathcal{J}}_B \\ -1, & \text{if } k = j \\ 0, & \text{if } k \in \bar{\mathcal{J}}_B, \quad k \neq j. \end{cases}$$

Theorem 2.3: (Orthogonality Theorem) Let $B^{'}$ and $B^{''}$ be two arbitrary bases of the finite set of vectors $\{\mathbf{a}_j | j \in \mathcal{J}\} \subset \mathbb{R}^m$. Then

$$\mathbf{t}^{(i)}\mathbf{t}_j = 0 \quad \text{for } i \in \mathcal{J}_{B'} \text{ and } j \in \bar{\mathcal{J}}_{B''}$$

where $\mathcal{J}_{B'}$ and $\mathcal{J}_{B''}$ are the index sets of $B^{'}$ and $B^{''}$, respectively.

Theorem 2.4: (Matrix Rank) Let $A \in \mathbb{R}^{m \times n}$ be an arbitrary matrix, where columns and rows of A are denoted by $\mathbf{a}_j, j = 1, \ldots, n$ and $\mathbf{a}^{(i)}, i = 1, \ldots, m$, respectively. Then $\text{rank}(\mathbf{a}_1, \ldots, \mathbf{a}_n) = \text{rank}(\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)})$.

The two theorems that follow state early results related with the solvability of linear systems.

Theorem 2.5: (Rouché-Capelli Lemma) $A\mathbf{x} = \mathbf{b}$ is solvable if and only if $rank(A) = rank([A, \mathbf{b}])$.

Theorem 2.6: (Rouché-Kronecker-Capelli) Let a matrix $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ be given. Then, exactly one of the following alternatives holds:

a) $A\mathbf{x} = \mathbf{b}$ has a solution, or

b) $\mathbf{y}^T A = \mathbf{0}$, $\mathbf{y}^T\mathbf{b} = 1$ has a solution.

Lemma 2.1: (Farkas) Let a matrix $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ be given. Then, exactly one of the following alternatives holds:

a) the system $A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ has a solution, or

b) the system $\mathbf{y}^T A \leq \mathbf{0}, \mathbf{y}^T\mathbf{b} = 1$ has a solution.

Proof. (first part) Suppose on the contrary that both systems have a solution and let the solution pair be $(\mathbf{x}, \mathbf{y})$. Then we have $\mathbf{y}^T A\mathbf{x} = \mathbf{y}^T\mathbf{b} = 1$ which cannot hold since $\mathbf{y}^T A \leq \mathbf{0}$ and $\mathbf{x} \geq \mathbf{0}$. □

To show that at least one of the alternative systems always has a solution, we define an algorithm that terminates only if it solves one of the problems. Showing that this algorithm is finite completes the proof.

Theorem 2.7: (Farkas-Minty) Let the vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n, \mathbf{b} \in \mathbb{R}^m$ be given. Then from the following two tableaux, where B' and B'' are two different bases of $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n, \mathbf{b}, \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m\}$, exactly one can occur.

	$\bar{\mathcal{J}}'_B$	b	$\bar{\mathcal{I}}'_B$
$\mathcal{J}'_B$			
$\mathcal{I}'_B$	0	0 $\vdots$ 0	

	$\bar{\mathcal{J}}_{B''}$		$\bar{\mathcal{I}}_{B''}$
$\mathcal{J}_{B''}$			
b	0 $\cdots$	0	
$\mathcal{I}_{B''}$	0		

2.5 Duality

To every linear optimization problem corresponds another linear optimization problem called its dual. Duality captures the symmetry inherent in a linear optimization problem and provides important insights about its solvability. Let us consider a linear optimization problem in standard form and call it the primal problem (P-LOP).

$$\text{minimize} \quad \mathbf{c}^T\mathbf{x} \tag{2.6}$$

$$\text{subject to} \quad A\mathbf{x} = \mathbf{b}, \tag{2.7}$$

$$\mathbf{x} \geq \mathbf{0} \tag{2.8}$$

The solution vector $\mathbf{x}$ which should be nonnegative and should be transformed to $\mathbf{b}$ by the matrix A (feasibility requirements), is expected to be a minimizer of $\mathbf{c}^T\mathbf{x}$ (optimality requirement). The dual problem is written with the aim of finding the best lower bound for the primal problem's objective function value $\mathbf{c}^T\mathbf{x}$, and is itself a linear optimization problem. The dual problem for (2.6) - (2.8) can be constructed as follows: Multiplying both sides of primal constraints $A\mathbf{x} = \mathbf{b}$ on the left by $\mathbf{y}^T$, we have $\mathbf{y}^T A\mathbf{x} = \mathbf{y}^T\mathbf{b}$. Since we would like to have a lower bound for $\mathbf{c}^T\mathbf{x}$, imposing the constraints $\mathbf{y}^T A \leq \mathbf{c}$ and using the nonnegativity constraints $\mathbf{x} \geq \mathbf{0}$ of the primal problem, we obtain $\mathbf{y}^T A\mathbf{x} = \mathbf{y}^T\mathbf{b} \leq \mathbf{c}^T\mathbf{x}$. Finally, $\mathbf{y}^T\mathbf{b}$ is maximized to find as good a lower bound for $\mathbf{c}^T\mathbf{x}$ as possible. The dual problem constructed in this way (D-LOP)

has the following precise form:

$$\text{maximize} \quad \mathbf{y}^T\mathbf{b} \tag{2.9}$$

$$\text{subject to} \quad \mathbf{y}^T A \leq \mathbf{c} \tag{2.10}$$

In general, the dual of a minimization problem is a maximization problem and vice versa. m primal constraints (2.7) are each associated with one dual variable and each of n dual constraints (2.10) with one primal variable.

There is a well developed body of theory around the concept of duality and the symmetric structure of the primal-dual pairs of problems are well exploited in the process of solving these problems, e.g., in developing pivot-based algorithms. Now we present important duality results. The sets of primal feasible and dual feasible solutions are,

$$P = \{\mathbf{x} \,|\, A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$$

$$D = \{\mathbf{y} \,|\, \mathbf{y}^T A \leq \mathbf{c}\},$$

and the sets of respective optimal solutions are,

$$P^* = \{\mathbf{x}^* \in P \,|\, \mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x} \quad \forall\, \mathbf{x} \in P\}$$

$$D^* = \{\mathbf{y}^* \in D \,|\, \mathbf{y}^{*T}\mathbf{b} \geq \mathbf{y}^T\mathbf{b} \quad \forall\, \mathbf{y} \in D\},$$

Theorem 2.8: (Weak Duality Theorem) For every feasible solution $\mathbf{x} \in P$ and every feasible solution $\mathbf{y} \in D$ the inequality $\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y}$ holds.

Proof of the weak duality theorem is direct by construction of the dual problem.

Theorem 2.9: (Strong Duality Theorem) For any primal-dual pair of linear

optimization problems at least one of the following holds:

1. $P = \emptyset$,
2. $D = \emptyset$,
3. if $P \neq \emptyset$ and $D \neq \emptyset$, then $P^* \neq \emptyset$ and $D^* \neq \emptyset$ and for any pair of optimal solutions $\mathbf{x}^* \in P^*$ and $\mathbf{y}^* \in D^*$, $\mathbf{c}^T\mathbf{x}^* = \mathbf{y}^{*T}\mathbf{b}$.

The main theorem of linear optimization duality, relating a primal problem to its dual can be seen as a statement about sign patterns of vectors in complementary subspaces of $\mathbb{R}^n$. This observation, first made by R. T. Rockafellar in the late sixties, led to the introduction of certain systems of sign vectors, called oriented matroids. Indeed, when oriented matroids came into being in the early seventies, one of the main issues was to study the fundamental principles underlying duality in this abstract setting [3].

2.6 The Simplex Method

Starting from its introduction by Dantzig in 1947, the simplex method has always been one of the main solution methods for linear optimization problems. From a theoretical point of view the simplex method is not really promising because its worst case computational complexity is shown to be exponential in the dimensions of the problem. The reasons of practical effectiveness of the simplex method despite its theoretical inefficiency has been studied as a research problem in its own right. The fact that the set of problems on which the simplex method executes exponentially many steps are not representative of the real-life problems usually encountered in practice and that these pathological problems are rather manufactured for the sole purpose of disproving polynomiality of a simplex pivot rule [see, e.g. Klee and Minty, 1972] provides a clue but does not prove anything about practical viability. For that, extensive studies have been performed which show that the average case complexity of the simplex method is a linear function of m [11]. These studies back up the

practical efficiency of the method.

With so many variants developed since its inception, "simplex method" is now used as an umbrella term that covers pivoting methods which behave in simplex-like ways. More precisely, a pivot method is called a simplex method if it preserves the (primal or dual) feasibility of basic solutions. Here, we present the primal simplex method for solving a primal linear optimization problem in standard form (P-LOP).

Primal simplex algorithm

Input data: Pivot tableau T_B corresponding to a primal feasible basis A_B
Begin
  $\mathcal{I}_- := \{i \in \mathcal{I}_N \,|\, \bar{c}_i < 0\}$;
  While ($\mathcal{I}_- \neq \emptyset$) **do**
    Let $q \in \mathcal{I}_-$ be arbitrary; (entering variable)
    If ($\mathbf{t}_q \leq \mathbf{0}$) **then** *STOP:* $\mathcal{D} = \emptyset$ (dual infeasibility)
    Else
      Let $\theta := \min\{\frac{x_i}{t_{iq}} : i \in \mathcal{I}_B \text{ and } t_{iq} > 0\}$; (ratio test)
      Let $p \in \mathcal{I}_B$ be arbitrary such that $\frac{x_p}{t_{pq}} = \theta$; (leaving variable)
    Pivoting: $\mathcal{I}_B := \mathcal{I}_B \cup \{q\} \setminus \{p\}$;
    Determine the new index set $\mathcal{I}_-$ corresponding to the updated basis;
  Optimal solution is found.
End

Figure 2.2: The primal simplex algorithm

The primal simplex method terminates with an optimal solution or a certificate of dual infeasibility (primal unboundedness) if it does not cycle.

2.7 Matroids and Oriented Matroids

In this section we give some basic definitions and examples necessary to understand the treatment of linear feasibility problem in oriented matroids in chapter 4. Some

familiarity with matroids is necessary in order to understand oriented matroids. Therefore we give basic examples for both concepts.

Let $E = \{e_1, ..., e_n\}$ be a finite set and $P(E)$ denote the power set of E. Furthermore, let $\mathcal{I} \subset P(E)$.

Definition 2.8: The sets $I \in \mathcal{I}$ are called *independent sets* and $M = (E, \mathcal{I})$ is a *matroid* if the following axioms are satisfied:

1. $\emptyset \in \mathcal{I}$.

2. If $I_1 \in \mathcal{I}$ and $I_2 \subset I_1$, then $I_2 \in \mathcal{I}$.

3. If $I_1, I_2 \in \mathcal{I}$ and $|I_1| > |I_2|$, then there exists $e \in I_1 \setminus I_2$, such that $I_2 \cup \{e\} \in \mathcal{I}$.

The set E is called the ground set of the matroid M.

A subset of E that is not in $\mathcal{I}$ is called dependent when $\mathcal{I}$ is a collection of independent sets. The family of dependent sets is denoted by $\mathcal{D}$. We next introduce the concept of a circuit.

Definition 2.9: Let $M = (E, \mathcal{I})$ be a matroid. The set $C \subset E$ is called a *circuit*, if $C \notin \mathcal{I}$, but $I \in \mathcal{I}$ holds for all $I \subsetneq C$. That is, a minimal (with respect to set inclusion) dependent set in a matroid M is called a *circuit* of M.

Example 2.3: Let $E = \{1,2,3,4,5,6\}$ and

$\mathcal{I} = \{X \subset E : |X| \leq 4\} \setminus \{\{1,2,3,4\}, \{1,2,5,6\}, \{3,4,5,6\}\}$. Thus the set of dependent sets for this matroid is

$\mathcal{D} = \{\{1,2,3,4\}, \{1,2,5,6\}, \{3,4,5,6\}\} \cup \{X \subseteq E : |X| \geq 5\}$. Then the circuits are

$\{1,2,3,4\}, \{3,4,5,6\}$, and $\{1,2,5,6\}$.

Let us denote a set of circuits defined on E by $\mathcal{C}$. The following is an alternative definition of a matroid based on circuits.

Definition 2.10: The pair $M = (E, \mathcal{C})$ is a *matroid* if the following axioms are satisfied:

1. For all $C_1, C_2 \in \mathcal{C}$ and $C_1 \subset C_2$, $C_1 = C_2$ follows.

2. For all $C_1, C_2 \in \mathcal{C}$ and $e_i \in C_1 \backslash C_2$, $e_j \in C_1 \cap C_2$, there exists $C_3 \in \mathcal{C}$, such that $e_i \in C_3 \subset (C_1 \cup C_2) \backslash \{e_j\}$.

The independent sets of the matroid $M = (E, \mathcal{C})$ are those sets $I \subset E$ which do not contain any circuits as subsets. A maximal (with respect to set inclusion) independent set in M is called a *basis* of M. There is a third way to define a matroid using bases.

Definition 2.11: The pair $M = (E, \mathcal{B})$ is called a *matroid* and the sets $B \in \mathcal{B}$ are called bases of M if the following axioms are satisfied:

1. For all $B_1, B_2 \in \mathcal{B}$ the cardinality of B_1 and that of B_2 are equal.

2. For all $B_1, B_2 \in \mathcal{B}$ and $e_i \in B_1$ there exists $e_j \in B_2$ such that $(B_1 \backslash \{e_i\}) \cup \{e_j\} \in \mathcal{B}$.

Example 2.4: It is easy to check that

$\mathcal{B}$ = { {1,2,3,5}, {1,2,3,6}, {1,2,4,5},{1,2,4,6}, {1,3,4,5}, {1,3,4,6},
{1,3,5,6}, {1,4,5,6}, {2,3,4,5}, {2,3,4,6}, {2,3,5,6}, {2,4,5,6} }

is the set of bases of the matroid defined in the previous example.

It is known that all given definitions of a matroid are equivalent [48].

It is easy to prove that $M^* = (E, \mathcal{B}^*)$ is a matroid, where

$$\mathcal{B}^* = \{B^* : B^* = E \backslash B \text{ for all } B \in \mathcal{B}\}.$$

The matroid $M^* = (E, \mathcal{B}^*)$ is called the *dual* of $M = (E, \mathcal{B})$. The circuits of the dual matroid are called *cocircuits*. The cardinality of any $B \in \mathcal{B}$ is called the *rank* of the matroid.

A signed set in E is a pair $X = (X^+, X^-)$, with $X^+ \subseteq E, X^- \subseteq E$, and $X^+ \cap X^- = \emptyset$. A pair $X = (X^+, X^-)$ is called a *signed set*, if $X^+, X^- \subset E$ and $X^+ \cap X^- = \emptyset$. We use the following notation $\bar{X} = X^+ \cup X^-$ and $-X = ((-X)^+, (-X)^-) = (X^-, X^+)$. So, (X^+, X^-) is a partition of $\bar{X}$ into ts positive and negative elements. Furthermore, $Y = \pm X$ means that either $Y = X$ or $Y = -X$. If $O = \{X_1, ..., X_p\}$ denotes a family of signed sets on E, then $\bar{O} = \{\bar{X}_1, ..., \bar{X}_p\}$ denotes the corresponding family of not signed sets on E. Bland studied the simplex method in the context of oriented matroids in [8]. We now introduce the concept of oriented matroids based on this paper and also the works by Folkman and Lawrence [23] and Bland and Las Vergnas [10]. Informally, an oriented matroid is a matroid where in addition every basis is equipped with an orientation.

Definition 2.12: Let O and O^* be families of signed sets defined on E. The pairs $M = (E, O)$ and $M^* = (E, O^*)$ are called dual pairs of oriented matroids if the following four conditions are satisfied:

1. $\bar{O}$ and $\bar{O}^*$ are collections of circuits and cocircuits of a dual pair of matroids $\bar{M} = (E, \bar{O})$ and $\bar{M}^* = (E, \bar{O}^*)$.

2. $X \in O \Rightarrow -X \in O$ and $Y \in O^* \Rightarrow -Y \in O^*$.

3. $X_1, X_2 \in O$ and $\bar{X}_1 = \bar{X}_2 \Rightarrow X_1 = \pm X_2, \quad Y_1, Y_2 \in O^*$ and $\bar{Y}_1 = \bar{Y}_2 \Rightarrow Y_1 = \pm Y_2$.

4. $X \in O, Y \in O^*$ and $\bar{X} \cap \bar{Y} \neq \emptyset$ imply

$$(X^+ \cap Y^+) \cup (X^- \cap Y^-) \neq \emptyset \text{ and } (X^+ \cap Y^-) \cup (X^- \cap Y^+) \neq \emptyset.$$

Condition 4 is called *orthogonality* condition.

Example 2.5: Let us introduce an oriented matroid $M = (E, O)$ with the sets of signed circuits and signed cocircuits as follows

$$O = \{\pm(+,+,-,-,0,0), \pm(+,+,0,0,-,-), \pm(0,0,+,+,-,-)\}$$

and

$$\begin{aligned} O^* = \{ & \pm(0,0,0,0,+,-), \pm(0,0,+,-,0,0), \pm(0,+,0,+,0,+), \pm(0,+,0,+,+,0), \\ & \pm(0,+,+,0,0,+), \pm(0,+,+,0,+,0), \pm(+,0,0,+,0,+), \pm(+,0,0,+,+,0), \\ & \pm(+,0,+,0,0,+), \pm(+,0,+,0,+,0), \pm(+,-,0,0,0,0)\}. \end{aligned}$$

The underlying matroid of $M = (E, O)$ is the matroid of Examples 2.3 and 2.4. Here a $+(-)$ sign in ith position of a circuit X (cocircuit Y) indicates that

$$e_i \in X^+ (e_i \in X^-) \ (e_i \in Y^+ (e_i \in Y^-)).$$

For further reading on matroids and oriented matroids we suggest the books by Gordon and McNulty [29] and Bjorner et. al. [7].

Chapter 3

FEASIBILITY PROBLEM OF LINEAR OPTIMIZATION

Feasibility problem of linear optimization or the linear feasibility problem is concerned with finding any feasible solution given a set of linear inequalities. Linear optimization and linear feasibility problems can be regarded as essentially the same problems because they can be reduced to one another [13, 15, 37]. So, a feasibility problem is not particularly easier than an optimization problem, and developing efficient methods for solving linear feasibility problems is as essential as developing methods for solving linear optimization problems. There are three main aims of this chapter. Firstly, we present a monotone scheme pivot algorithm for solving the linear feasibility problem (LFP) and prove its finiteness. The algorithm uses a similar ratio rule to Monotonic Build Up pivot rule developed by Anstreicher and Terlaky [1] in determining the variable that will enter the basis. Selecting pivot positions under the guidance of this ratio rule guarantees the monotonic increase of the feasibility status of the problem at each step. By monotonic increase of the feasibility status we mean either an increase in the number of (primal) feasible variables or an increase in the value of the driving variable with no change in the set of (primal) feasible variables.

Secondly, we analyze the presented algorithm and compute an upper bound for its computational complexity under the assumption that the problem is either nondegenerate or if degenerate bases occur during the solution process they are of the weak degenerate type. In other words, we assume that the bases visited by the algorithm are not strongly degenerate. Strong degeneracy, if it happens, calls for extra

measures be taken so that cycling will be prevented.

A method developed for solving the feasibility problem is expected, naturally, to return a point of the feasibility set or report that the problem is infeasible if the feasibility set turns out to be empty. If what we are trying to solve is a real-world problem by modeling it as a linear feasibility problem, merely reporting an infeasibility status is often not satisfactory because there remains a real problem to be solved behind the model. This point brings us to the third goal of the current chapter, which is investigating the cause of infeasibility.

The main results in this chapter are based on the published papers [5] and [6].

Linear feasibility problem in standard form is

$$A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}, \tag{3.1}$$

where $A \in \mathbb{R}^{m \times n}$ and $\mathbf{x}, \mathbf{b}$, and $\mathbf{0}$ are vectors of conforming sizes. We denote the index set of all variables by $\mathcal{I}$.

3.1 Monotonic Build Up Simplex Pivot Rule for LFP

Anstreicher and Terlaky in [1] define a pivot based algorithm they call Monotonic Build Up (MBU) Simplex algorithm and study its properties. MBU Simplex algorithm solves a linear programming problem starting with a primal feasible basis and *builds up* the feasibility of the dual variables monotonically while preserving primal feasibility. The algorithm may visit basic solutions which are outside the feasible region, that is, basic solutions which are neither primal nor dual feasible, but always returns to a primal feasible basis with an increased number of dual feasible variables. In this way, an optimal solution (if one exists), given by a basis which is both primal and dual

feasible, is achieved.

In this section we present a version of MBU to solve the linear feasibility problem 3.1 and analyze the resulting algorithm under a weak degeneracy assumption. The proposed algorithm possesses similar properties with Anstreicher and Terlaky's MBU in the way that it has the number of primal feasible variables increasing monotonically in the process due to a similar pivot selection rule.

For a given basis B, the set of basic variables $\mathcal{I}_B$ can be partitioned based on their values as follows:

$$\mathcal{I}_B = \mathcal{I}_B^+ \cup \mathcal{I}_B^0 \cup \mathcal{I}_B^-,$$

where

$$\mathcal{I}_B^+ = \{i \in \mathcal{I}_B : \bar{x}_i > 0\},\ \mathcal{I}_B^0 = \{i \in \mathcal{I}_B : \bar{x}_i = 0\} \text{ and } \mathcal{I}_B^- = \{i \in \mathcal{I}_B : \bar{x}_i < 0\}.$$

A specific variable x_i is feasible if $i \in \mathcal{I}_B^\oplus$ where $\mathcal{I}_B^\oplus = \mathcal{I}_B^+ \cup \mathcal{I}_B^0$. From the set of basic feasible variables corresponding to a given basis, let us call the ones which are equal to zero as degenerate basic variables.

A sample basic tableau according to the above partition is shown in Figure 3.1, where each symbol $(-, 0, +)$ represents either a right hand side value of the appropriate sign or zero, or, a possible pivot position of the appropriate sign.

In most pivot algorithms, pivots are made only on positive elements in primal feasible rows (like the simplex algorithm), and only on negative elements in primal infeasible rows (like the dual simplex algorithm), or the two strategies are combined (like the

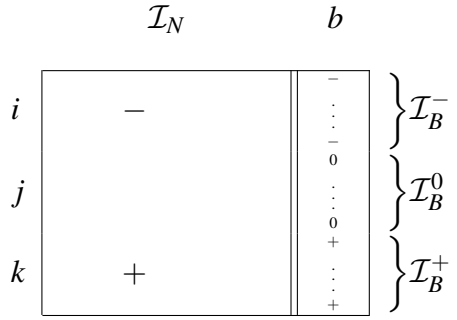

Figure 3.1: Partition of a basis.

criss-cross method). To conceptualize this, Fukuda and Terlaky [27, 28] have defined *admissible pivot operations*. Our algorithm carries out more general pivot operations and for this reason we will define *generalized* admissible pivots. A pivot on a positive value in the row of a strictly feasible (positive) variable can be viewed as the primal feasibility equivalent of the dual side admissible pivots. While handling degeneracy, our algorithm makes pivots on both negative and positive values in degenerate rows and hence the following definition of generalized admissible pivots contains those cases as well.

Definition 3.1: For a given basis B and pivot tableau T, a pivot element t_{ij} is called a *generalized admissible pivot* if

1. $i \in \mathcal{I}_B^-$ and $t_{ij} < 0$, or
2. $i \in \mathcal{I}_B^\oplus$ and $t_{ij} > 0$, or
3. $i \in \mathcal{I}_B^0$ and $t_{ij} \neq 0$.

Before formulating our algorithm which uses generalized admissible pivots, we need to refine the concept of degeneracy. For $s \in \mathcal{I}_N$ let us introduce the set

$$\mathcal{K}_s = \left\{ i \in \mathcal{I}_B^0 : t_{is} > 0 \right\}.$$

The set $\mathcal{K}_s$ isolates those degenerate basic variables whose tableau entries for some specific column s are positive.

Definition 3.2: A basis B is called *degenerate* if the corresponding basic solution $\bar{\mathbf{x}}_{\mathcal{I}_B} = B^{-1}\mathbf{b}$ has at least one zero component, and it is called *non-degenerate* otherwise.

The degeneracy concept defined above is often referred as *primal degeneracy* and analogously *primal non-degeneracy*. The phenomenon of degeneracy is a local property depending on the actual basis. A basis is degenerate if and only if the right hand side vector is the linear combination of less number of columns of the basis matrix than its dimension. We distinguish between two kinds of degeneracies.

Definition 3.3: A degenerate basis B is called *weakly degenerate* with respect to index $s \in \mathcal{I}_N$ if $\mathcal{K}_s = \emptyset$, and *strongly degenerate* with respect to index s if $\mathcal{K}_s \neq \emptyset$.

Let us assume that for a given basis B, we have chosen the index s as a column of a generalized admissible pivot in a non-degenerate row. Observe that such a pivot does not make any feasible variable infeasible if and only if $\mathcal{K}_s = \emptyset$, i.e., if and only if the basis B is weakly degenerate with respect to index s. Such a weakly degenerate tableau is shown in Figure 3.2.

Definition 3.4: Let a basis B and an index $r \in \mathcal{I}_B^-$ be given. We call a pivot operation on t_{ij} an x_r *increasing* pivot if

1. the pivot made on t_{ij} is a generalized admissible pivot,
2. $\mathcal{I}_B^{\oplus} \subseteq \mathcal{I}_{B'}^{\oplus}$, and

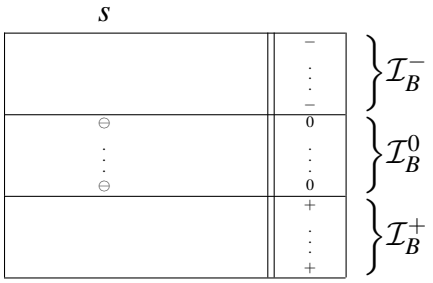

Figure 3.2: A weakly degenerate pivot tableau.

3. $\hat{x}_r > \bar{x}_r$ holds,

where $B' = B \cup \{j\} - \{i\}$ denotes the new basis, and $\hat{\mathbf{x}}$ denotes the new basic solution.

Our aim is to formulate an algorithm based on x_r increasing pivots. To the best of our knowledge, the only such pivot algorithm for feasibility (or general linear programming) problems known from the literature is the dual version of the algorithm of Anstreicher and Terlaky [1] (which starts from a dual feasible basic solution).

Unfortunately, there are basic tableaux where no x_r increasing pivot exists, as shown by the example of Figure 3.3. This problem has a feasible solution of $x_1 = x_2 = 0, x_3 = x_4 = 1$, but has no increasing pivot for the only infeasible row.

	x_3	x_4	
x_1	-1	0	-1
x_2	1	-1	0

Figure 3.3: A strongly degenerate tableau with no x_r increasing pivot, where $r = 1$.

This pathological example shows that one may expect that there is no such algorithm that performs only x_r increasing pivots since the only possible non-degenerate pivot would make the variable x_2 infeasible.

Our algorithm follows an intuitive path in order to achieve monotonicity. Starting with a given or constructed basic solution, the algorithm first chooses an infeasible variable x_r and designates it as the *driving variable*. The aim of the subsequent iterations is to reduce the infeasibility of the driving variable by means of pivot operations we aptly call x_r increasing pivots, and to eventually change the status of x_r from infeasible to feasible. We will show that this can always be done if the pivot tableau is non-degenerate or weakly degenerate with respect to the pivot column. The algorithm then selects another infeasible variable as the driving variable and applies a sequence of pivot operations until its status becomes feasible. The procedure continues until there is no more infeasible variable or the problem is detected to be infeasible.

The pseudocode of our algorithm is given in Figure 3.4. Note that the algorithm outputs a feasible solution or a message that no feasible solution exists, that is, the corresponding polyhedron is empty. The index set $\mathcal{J}$ used in the algorithm represents the set of basic variables. Therefore, $\mathcal{J}$ contains the m row indices - by associating the basic variable appearing in each constraint to the corresponding row (there is precisely one basic variable in every constraint)- or, in other words, indices of the m basic variables.

In case of strongly degenerate tableaux, the algorithm uses a recursive anti-degeneracy method we call *DegProc*. Naturally, degeneracy could be handled by using classical approaches known from the literature (like minimal index rule, lexicographic ordering, etc.). Our anti-degeneracy method is different from those known in the literature and will ensure finiteness too. Handling strong degeneracy using DegProc is explained in Section 3.3.

MBU type simplex algorithm for feasibility problems

Input data: $A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m$, basis B;
Begin
  $T := B^{-1}A, \bar{\mathbf{b}} := B^{-1}\mathbf{b}, \quad \mathcal{I}_B^- := \{i \in \mathcal{J} \mid \bar{b}_i < 0\}$
  While $(\mathcal{I}_B^- \neq \emptyset)$ **do**
    Let $r \in \mathcal{I}_B^-$ be arbitrary (driving variable), $\quad rDone := false$
    While $(rDone = false)$ **do**
      $\mathcal{J}_r^- := \{j \in \mathcal{I}_N \mid t_{rj} < 0\}$
      If $(\mathcal{J}_r^- = \emptyset)$ **then**
        no feasible solution exists, **Return**
      Endif
      Let $s \in \mathcal{J}_r^-$ be arbitrary, $\quad \mathcal{K}_s := \{i \in \mathcal{I}_B^0 \mid t_{is} > 0\}$
      If $(\mathcal{K}_s \neq \emptyset)$ **then**
        $(T, l) = DegProc(T, \mathcal{I}_B^0, r)$
        If $(l \in \mathcal{I}_N)$ **then**
          $s := l$
        else
          no feasible solution exists, **Return**
        Endif
      Endif
      $\theta_1 := \frac{\bar{b}_r}{t_{rs}}, \quad \theta_2 := \min\left\{\frac{\bar{b}_k}{t_{ks}} \mid k \in \mathcal{I}_B^{\oplus}, t_{ks} > 0\right\}$
      If $(\theta_1 \leq \theta_2)$ **then**
        pivot on t_{rs}, $\quad rDone := true$
      else
        $q := \arg\min\left\{\frac{\bar{b}_k}{t_{ks}} \mid k \in \mathcal{I}_B^+, t_{ks} > 0\right\}$, pivot on t_{qs}
      Endif
    Endwhile
    $\mathcal{I}_B^- := \{i \mid \bar{b}_i < 0\}$
  Endwhile
  Return: $\bar{\mathbf{x}}$ is a feasible solution
End

Figure 3.4: The MBU type simplex algorithm

In order to find an x_r increasing pivot, a ratio test is performed in the selected column. The result of this test determines whether the driving variable can be made feasible in one pivot step or more than one x_r increasing pivot steps will be necessary.

Two ratios which take part in the test are defined by

$$\theta_1 := \frac{\bar{b}_r}{t_{rs}}, \text{ and } \theta_2 := \min\left\{ \frac{\bar{b}_k}{t_{ks}} \mid k \in \mathcal{I}_B^{\oplus}, t_{ks} > 0 \right\}.$$

From their definition, it is easy to see that $\theta_1 > 0$ and $\theta_2 \geq 0$. Furthermore, if the basis is non-degenerate or weakly degenerate, then $\theta_2 > 0$ holds. We use the convention that the minimum taken over the empty set is infinity. The aim of the inner cycle of the algorithm is to make the driving variable feasible.

We prove that if the given basis B is non-degenerate or weakly degenerate, then the algorithm makes only x_r increasing pivots. First we investigate the case when the driving variable leaves the basis.

Proposition 3.1: For a given basis B, let $r \in \mathcal{I}_B^-$ and $q \in \mathcal{I}_B^{\oplus}$, $t_{qs} > 0$. Suppose that

$$\frac{\bar{b}_q}{t_{qs}} = \theta_2 \geq \theta_1 = \frac{\bar{b}_r}{t_{rs}} > 0,$$

where $\bar{b}_r < 0$, $t_{rs} < 0$, $\bar{b}_q \geq 0$ and $t_{qs} > 0$. In this case a pivot is carried out on t_{rs}. Let us denote the new basis by B', then

$$\mathcal{I}_B^{\oplus} \cup \{s\} \subseteq \mathcal{I}_{B'}^{\oplus}$$

holds, thus

$$\left|\mathcal{I}_{B'}^{\oplus}\right| > \left|\mathcal{I}_B^{\oplus}\right|.$$

Note that $\theta_2 > 0$ also implies $q \in \mathcal{I}_B^+$.

Proof. When t_{rs} is the pivot position, the variable x_r leaves the basis, while variable x_s enters it. Let us denote the new basic solution (thus the new right hand side of the pivot tableau) by $\mathbf{b}'$. We distinguish between the following cases.

a) For the index s the right hand side becomes $b'_s = \frac{\bar{b}_r}{t_{rs}} = \theta_1 > 0$, making the driving

variable $x'_r = 0$ feasible.

b) For $i \in \mathcal{I}_B^+$, $i \neq s$ we have that $b'_i = \bar{b}_i - \frac{t_{is}\bar{b}_r}{t_{rs}}$. If $t_{is} \leq 0$ then by using $\bar{b}_i \geq 0$, $\bar{b}_r < 0$ and $t_{rs} < 0$ we have $b'_i > 0$, because we add a nonnegative number to an already positive $\bar{b}_i$. Otherwise if $t_{is} > 0$ then by $b'_i = t_{is}\left(\frac{\bar{b}_i}{t_{is}} - \frac{\bar{b}_r}{t_{rs}}\right)$ using the condition $\frac{\bar{b}_i}{t_{is}} \geq \theta_2 \geq \theta_1 = \frac{\bar{b}_r}{t_{rs}} > 0$ we get that $b'_i \geq 0$.

c) For $i \in \mathcal{I}_B^0$ we have $\bar{b}_i = 0$, so $b'_i = -\frac{t_{is}\bar{b}_r}{t_{rs}}$, and by $\theta_2 > 0$ either $\mathcal{I}_B^0 = \emptyset$ or $t_{is} \leq 0$, thus $b'_i = -\frac{t_{is}\bar{b}_r}{t_{rs}} \geq 0$.

As we have seen, no feasible basic variable turns infeasible, while the infeasible driving variable x_r leaves the basis, thus becoming feasible, and the entering variable x_s enters at a feasible level, as well. It follows that $\left|\mathcal{I}_{B'}^{\oplus}\right| > \left|\mathcal{I}_B^{\oplus}\right|$. □

In the next proposition we investigate the case when a pivot is made outside the row of the driving variable. If the basis is non-degenerate or weakly degenerate for the entering nonbasic variable, the pivot made by the algorithm is still x_r increasing.

Proposition 3.2: In a given iteration of the algorithm, for the actual basis B let $r \in \mathcal{I}_B^-$ and $q \in \mathcal{I}_B^{\oplus}$, $t_{qs} > 0$. Suppose that

$$0 \leq \frac{\bar{b}_q}{t_{qs}} = \theta_2 < \theta_1 = \frac{\bar{b}_r}{t_{rs}},$$

where $\bar{b}_r < 0$, $t_{rs} < 0$, $\bar{b}_q \geq 0$ and $t_{qs} > 0$. In this case a pivot is carried out on t_{qs}. Let us denote the new basis by B'. Then $\mathcal{I}_B^{\oplus} \setminus \{q\} \subseteq \mathcal{I}_{B'}^{\oplus} \setminus \{s\}$ and $0 > b'_r \geq \bar{b}_r$. Furthermore, if $\theta_2 > 0$, then $0 > b'_r > \bar{b}_r$ holds.

Proof. Using the notations introduced in the proof of Proposition 3.1, one can see that due to the ratio test, for any index $i \in \mathcal{I}_B^{\oplus}$ we have $i \in \mathcal{I}_{B'}^{\oplus}$ if $i \neq q$, as proved in Proposition 3.1.

Furthermore, $b'_s = \frac{\bar{b}_q}{t_{qs}} \geq 0$, so $s \in \mathcal{I}_{B'}^{\oplus}$ thus

$$\mathcal{I}_B^{\oplus} \setminus \{q\} \subseteq \mathcal{I}_{B'}^{\oplus} \setminus \{s\}$$

proving that the already feasible variables remain feasible. For the index of the leading variable $b'_r = \bar{b}_r - \frac{t_{rs}\bar{b}_q}{t_{qs}}$, where $-\frac{t_{rs}\bar{b}_q}{t_{qs}} \geq 0$, using that $t_{rs} < 0$, $t_{qs} > 0$ and $\bar{b}_q \geq 0$. By the condition $\theta_2 < \theta_1$ we have $0 \geq \frac{t_{rs}\bar{b}_q}{t_{qs}} > \bar{b}_r$ thus

$$0 > b'_r = \bar{b}_r - \frac{t_{rs}\bar{b}_q}{t_{qs}} \geq \bar{b}_r.$$

If the basis is non-degenerate, or weakly degenerate, then $\theta_2 > 0$ holds by definition, so $\frac{\bar{b}_q}{t_{qs}} > 0$, thus $-t_{rs}\frac{\bar{b}_q}{t_{qs}} > 0$, and

$$0 > b'_r = \bar{b}_r - t_{rs}\frac{\bar{b}_q}{t_{qs}} > \bar{b}_r,$$

completing the proof. □

Geometrically, Proposition 3.2 tells us that the new solution is closer to the nonnegativity constraint of the driving variable.

Summarizing Propositions 3.1 and 3.2 we obtain the following result.

Corollary 3.1: If the MBU type simplex algorithm performs only non-degenerate or weakly degenerate pivots, then the algorithm makes only x_r increasing pivots, and thus it is finite.

Proof. The number of different bases is finite; therefore, it suffices to prove that the algorithm is not cycling, or in other words, that a basis may not occur twice. Since we assumed that the algorithm does not visit strongly degenerate bases, it follows from Propositions 3.1 and 3.2 that at each iteration, the algorithm makes x_r increasing pivots. In each step a new variable becomes feasible, or the value of the driving variable

increases, thus the same basis may not return. □

Propositions 3.1 and 3.2 present important results for our MBU type simplex algorithm. Anstreicher and Terlaky in their paper [1] have proved similar results for their primal algorithm for linear programming problems.

In the next section we give a lower bound on the increment of the value of the driving variable, and consequently we provide an upper bound on the iteration number of the algorithm. Most classical and primal MBU simplex algorithms can be analyzed in a similar way to what is presented in the next section.

3.2 Complexity Analysis

In this section we first assume that our algorithm visits only non-degenerate or weakly degenerate tableaux. Degeneracy is handled in Section 3.3.

By the definition of the pivot tableau and the basic solution, for the s^{th} column of the tableau we have $\mathbf{t}_s = B^{-1}\mathbf{a}_s$ and $\bar{\mathbf{b}} = B^{-1}\mathbf{b}$; thus the vectors $\mathbf{t}_s$ and $\bar{\mathbf{b}}$ can be considered as the unique solutions of the linear equations $B\mathbf{u} = \mathbf{a}_s$ and $B\mathbf{v} = \mathbf{b}$. For any index $i \in \mathcal{I}_B$, Cramer's rule yields

$$t_{is} = \frac{\det(B_{is})}{\det(B)} \quad \text{and} \quad \bar{b}_i = \frac{\det(B_i)}{\det(B)},$$

where matrix $B_{is} \in \mathbb{R}^{m\times m}$ is the modification of the regular basic matrix B such that its i^{th} column is replaced by vector $\mathbf{a}_s$, and similarly matrix B_i is obtained from B by replacing its i^{th} column by vector $\mathbf{b}$. For an x_r increasing pivot that does not make the driving variable feasible, we have

$$b'_r = \bar{b}_r - \frac{t_{rs}\bar{b}_q}{t_{qs}} = \frac{\det(B_r)}{\det(B)} - \frac{\frac{\det(B_{rs})}{\det(B)}\frac{\det(B_q)}{\det(B)}}{\frac{\det(B_{qs})}{\det(B)}} = \frac{\det(B_r)}{\det(B)} - \frac{\det(B_{rs})\det(B_q)}{\det(B_{qs})\det(B)},$$

where

$$-\frac{\det(B_{rs})}{\det(B_{qs})}\frac{\det(B_q)}{\det(B)} > 0$$

holds by the fact that the basis is not strongly degenerate, as seen in Proposition 3.2. Let

$$\Delta_A := \min\left\{ -\frac{\det(B_{rs})\det(B_q)}{\det(B_{qs})\det(B)} : \begin{array}{l} B \text{ is a regular submatrix of A, and} \\ \frac{\det(B_{rs})}{\det(B)} < 0,\ \frac{\det(B_q)}{\det(B)} > 0,\ \frac{\det(B_{qs})}{\det(B)} > 0 \end{array} \right\}$$

be the minimal increase of the driving variable's value. Assuming that in all pivot transformations of the tableau $\theta_2 > 0$ holds, we have that $\Delta_A > 0$ is a finite number and

$$b'_r = \bar{b}_r - \frac{t_{rs}\bar{b}_q}{t_{qs}} = \frac{\det(B_r)}{\det(B)} - \frac{\det(B_{rs})\det(B_q)}{\det(B_{qs})\det(B)} \geq \frac{\det(B_r)}{\det(B)} + \Delta_A$$

thus an x_r increasing pivot either makes the leading variable feasible, or increases its value by at least Δ_A. We now bound the maximum absolute value that an infeasible variable can take during the algorithm. Let

$$\Delta_{\max} := \max\left\{ -\frac{\det(B_r)}{\det(B)} : \begin{array}{l} \operatorname{sgn}(\det(B_r)) = -\operatorname{sgn}(\det(B)), \\ B \in \mathbb{R}^{m\times m} \text{ is a regular submatrix of } A \end{array} \right\}$$

be the maximal possible RHS value determined by the help of Cramer's rule. If there is any basis for which there is a negative right hand side value, then the number $\Delta_{\max}$ is positive and finite. Let $K \in \mathbb{Z}$ be such that $K = \left\lceil \frac{\Delta_{\max}}{\Delta_A} \right\rceil$, thus $K \in \mathbb{N}$.

We are now ready to bound the number of pivots necessary to make the driving variable feasible.

Proposition 3.3: Assume that the algorithm visits only non-degenerate or weakly degenerate pivot tableaux. Let $r \in \mathcal{I}_B^-$ be the index of the driving variable. There can be at most K pivot operations before the driving variable becomes feasible.

Proof. By the definition, the value of the driving variable cannot be smaller than

$-\Delta_{\max}$. The value of the driving variable increases by at least Δ_A in every iteration; thus, there cannot be more than K iterations before the next x_r increasing pivot makes the driving variable feasible. □

We are now ready to prove the bound on the complexity of the algorithm.

Theorem 3.1: Consider the feasibility problem (3.1). Assume that the MBU type pivot algorithm visits only non-degenerate or weakly degenerate pivot tableaux in solving (3.1). Then the algorithm is finite, and there can be at most mK pivots.

Proof. By Proposition 3.3, there can be at most K pivot operations before the algorithm reaches feasibility in the row of the driving variable or proves infeasibility. The number of driving variables during the algorithm is bounded by the number of rows, because by Propositions 3.1 and 3.2 the number of infeasible variables decreases monotonically; thus, the algorithm may not cycle, and there can be at most mK pivots before solving the problem, or proving that it is infeasible. □

We have proved under the non-degeneracy assumption that the algorithm is finite, and we can bound the required number of pivot operations. This upper bound is generally not tight.

By Corollary 3.1, none of the bases can be visited twice by our MBU type algorithm. Therefore, the number of pivot iterations is at most

$$\min\left\{ mK, \binom{n}{m} \right\}.$$

Furthermore, if we assume that $A \in \mathbb{Z}^{m\times n}$ and $\mathbf{b} \in \mathbb{Z}^m$, then it is easy to show that the following inequality

$$K = \left\lceil \frac{\Delta_{\max}}{\Delta_A} \right\rceil = \left\lceil \frac{\det(B_r)}{\det(B)} \frac{\det(B'_{qs}) \det(B')}{\det(B'_{rs}) \det(B'_q)} \right\rceil \leq |\det(B^*)|^3$$

holds, where $B^* \in \mathbb{Z}^{m \times m}$ is such a submatrix of the matrix $[A\,\mathbf{b}]$ which has, in absolute value, maximal determinant. Denoting by $L = L(A, \mathbf{b})$, the classical bit length description of the matrix A and vector $\mathbf{b}$, and applying the well known Hadamard inequality, we can derive that

$$K = \left\lceil \frac{\Delta_{\max}}{\Delta_A} \right\rceil \leq |\det(B^*)|^3 \leq 2^{3L}.$$

This shows once more that we do not know a good (polynomial size) bound for the constant K for an arbitrary feasibility problem. However, the proof of the existence of such a constant K is new. Although the ratio of $\Delta_{\max}$ to Δ_A can be very large in general, the constructed bound is an incentive to search for bounds for classes of problems for which this ratio is small.

3.3 Handling Strong Degeneracy

In this section, we discuss the *DegProc*, the procedure handling degeneracy in our MBU type algorithm presented in Figure 3.4.

Algorithmically, degeneracy is often handled by perturbation or index selection rules (lexicographic, minimal index). This is also possible for MBU type simplex algorithm, but we follow a different approach to handle problems caused by degeneracy.

The key issue of the analysis presented in the previous section was that the tableaux visited by the algorithm were all non-degenerate or weakly degenerate. The procedure for handling degeneracy selects pivot positions based on the row of the driving variable and the degenerate submatrix (consisting of all degenerate rows). These pivots do not change the current basic solution, but transform the basic tableau in such a way

that it either becomes primal infeasible, or there will be at least one column that has negative entry in the row of the driving variable, and is weakly degenerate. During the solution process of the primal (dual) degenerate subproblem, we only make pivots in the degenerate rows. Furthermore, for solving the subproblems we use the dual (primal) version of our MBU type simplex algorithm. This way, the solution process of the subproblems carries the same already shown basic properties over the iterations as the MBU type simplex algorithm.

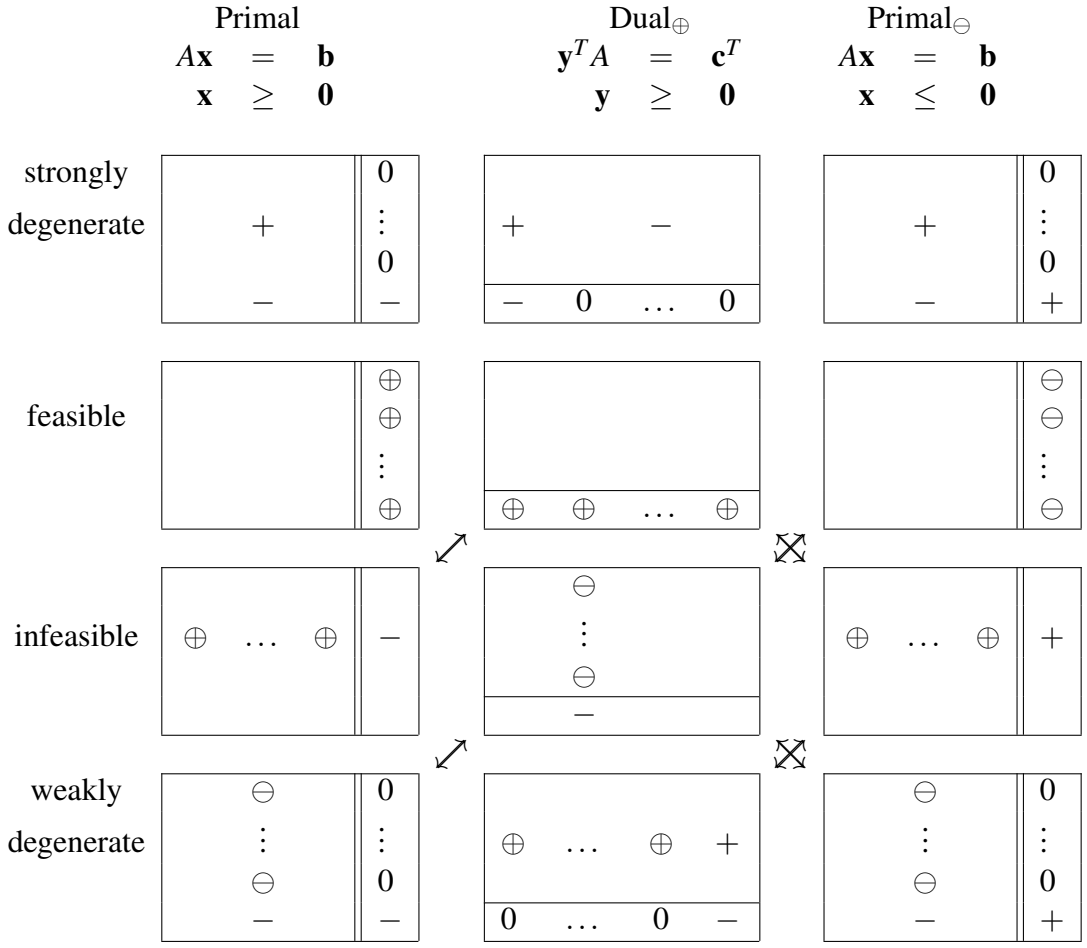

Figure 3.5: Key tableaux of the algorithm.

The anti-degeneracy method described in the sequel uses a recursive structure. We use the tableaux illustrated in Figure 3.5. Suppose that a strongly degenerate tableau of the

primal problem is obtained. In this case, the algorithm defines a so called *dual side subproblem* $\text{Dual}_{\oplus}$. The tableau of this subproblem consists of all the degenerate rows without the primal right hand side and the row of the driving variable, as dual right hand side. Constructed this way, the dual side subproblem $\text{Dual}_{\oplus}$ has the following form:

$$\mathbf{y}^T A = \mathbf{c}^T, \ \ \mathbf{y} \geq \mathbf{0}$$

where the size of $\mathbf{y}$ is $|\mathcal{I}_B^0|$. Obviously, the size of the subproblem is smaller than the size of the original primal problem. Now suppose that the dual subproblem is solved. If the dual subproblem is feasible, its structure is the same as the structure of an infeasible tableau of the original primal problem corresponding to the degenerate rows and the driving variable; while if the dual subproblem is infeasible, its structure is the same as the structure of a weakly degenerate tableau of the original primal problem, restricted again to the degenerate rows and the row of the driving variable.

A similar interrelated connection may be observed between problems $\text{Dual}_{\oplus}$ and $\text{Primal}_{\ominus}$, as illustrated in Figure 3.5. The problem $\text{Primal}_{\ominus}$ is an analogous version of the original problem, where all the variables are required to be non-positive, instead of being non-negative. The accurate definition for the analogous algorithm will be given in Figure 3.6, while the connections used by the recursions will be illustrated in Figure 3.9.

We will use the dual version of our MBU type simplex algorithm to solve the dual subproblems as the two algorithms have symmetric descriptions. Any pivot method could as well be used to solve the subproblems.

We prove that while solving the degenerate subproblem, the actual basic solution of

the algorithm does not change.

Proposition 3.4: Let a degenerate pivot tableau $T \in \mathbb{R}^{m \times n}$ be given, and denote the index set of degenerate rows by $\mathcal{D} = \mathcal{I}_B^0$. Then any pivot made on the elements of submatrix $T_{\mathcal{D}\mathcal{I}}$ does not change the current basic solution.

Proof. Let the chosen generalized admissible pivot element be $t_{ij} \in T_{\mathcal{D}\mathcal{I}}$. We show that the right-hand side of the tableau does not change after the pivot, namely $b'_i = \frac{b_i}{t_{ij}} = 0 = b_i$ and $b'_k = b_k + t_{kj}\frac{b_i}{t_{ij}} = b_k + 0 = b_k$, where $k \neq i$. □

Because of the nature of the procedure handling degeneracy, we formulate it as a recursive method. The pseudo-codes of the subprocedures are summarized in Figures 3.6. and 3.7. The subprocedures make pivots on the whole tableaux, but consider only the subtableau defined by the index sets $\mathcal{F}$ and $\mathcal{G}$. The column indexed by b, and row indexed by c play the roles of primal and dual right-hand sides respectively, of the subproblems taken into consideration.

Because the primal subproblems called by *DegProc* are instances of type Primal$_\ominus$ where the right hand side values are required to be non-positive, in what follows we call a primal variable feasible if its sign is adequate for the corresponding feasibility problem or zero, and otherwise infeasible. For the sake of clarity, in the discussion of the subproblems, we talk only of values instead of variables, when referring to the right hand side values; because only for the original problem do variables actually correspond to right hand side values.

The analysis of the procedure for solving the dual subproblems is completely analogous to the primal version. Using Figure 3.5, the concepts of dual side weak and

strong degeneracy, as well as the concept of the x_r increasing pivot can be defined analogously as in Propositions 3.2, 3.3 and 3.4.

Proposition 3.5: For the PrimalSubMBU and DualSubMBU procedures, if the corresponding tableau is non-degenerate or weakly degenerate, then the pivot steps carried out are increasing pivots for the column indexed by b or for the row indexed by c, respectively.

Proof. The first part of the proposition follows immediately from Proposition 3.1, while the part corresponding to the dual subprocedure can be proved similarly. □

We stated the main idea behind the recursive algorithm; now, we formalize it for both subprocedures.

We show that the primal and dual subalgorithms have similar properties as the primal algorithm working on non-degenerate problems.

Proposition 3.6: Suppose that the algorithm is started from problem P_0, then because of repeated strongly degenerate tableaux, the following recursive calling sequence occurs: $P_1 := DualSubMBU(T_1, k_1, \mathcal{F}_1, \mathcal{G}_1)$,

$P_2 := PrimalSubMBU(T_2, k_2, \mathcal{F}_2, \mathcal{G}_2)$,

$\ldots$,

$P_l := PrimalSubMBU(T_l, k_l, \mathcal{F}_l, \mathcal{G}_l)$ (or $P_l := DualSubMBU(T_l, k_l, \mathcal{F}_l, \mathcal{G}_l)$). Then the pivots carried out while solving subproblem P_l do not change any right hand side of problems P_i, where $(i = 1, \ldots, l-1)$ and $k_i \notin \mathcal{F}_l \cup \mathcal{G}_l$.

Proof. The recursive steps involve only degenerate rows and columns, thus by Proposition 3.4 our statement holds. □

Procedure DualSubMBU

Input: $(T, c, \mathcal{F}, \mathcal{G})$.
Output: (T modified pivot tableau, l status flag).
Begin
  $\mathcal{J}_B^- := \{j \in \mathcal{G} : t_{cj} < 0\}$
  While $(\mathcal{J}_B^- \neq \emptyset)$ **do**
    Let $r \in \mathcal{J}_B^-$ be arbitrary (driving variable), $rDone := false$.
    While $(rDone = false)$ **do**
      $\mathcal{I}_r^+ := \{i \in \mathcal{F} \mid t_{ir} > 0\}$
      If $(\mathcal{I}_r^+ = \emptyset)$ **then**
        The calling pivot tableau is weakly degenerate, **Return**(T, r).
      Endif
      Let $s \in \mathcal{I}_r^+$, $\mathcal{K}_s = \{k \in \mathcal{G} \mid t_{ck} = 0, t_{sk} < 0\}$.
      If $(\mathcal{K}_s \neq \emptyset)$ **then** (when subtableau $T_{\mathcal{FG}}$ is strongly degenerate)
        $(T, s) :=$ PrimalSubMBU$(T, r, \mathcal{F}, \{j \in \mathcal{G} \mid t_{cj} = 0\})$.
        If $(s = -1)$ **then**
          The calling tableau is weakly degenerate, **Return**(T, r).
        Endif
      Endif
      $\theta_1 := \frac{t_{cr}}{t_{sr}}, \quad \theta_2 := \max\left\{\frac{t_{ck}}{t_{sr}} \mid k \in \mathcal{G}, t_{ck} > 0, t_{sk} < 0\right\}$
      If $(\theta_1 \leq \theta_2)$ **then**
        pivot on t_{sr}, $rDone := true$.
      else
        $q := \arg\max\left\{\frac{t_{ck}}{t_{sr}} \mid k \in \mathcal{G}, t_{ck} > 0, t_{sk} < 0\right\}$.
        pivot on t_{sq}.
      Endif
    Endwhile
  Endwhile
  The calling tableau is infeasible, **Return**$(T, -1)$**.**
End

Figure 3.6: The dual subprocedure starts with the solution of a subproblem of the form Dual$_\oplus$.

Procedure PrimalSubMBU

Input: $(T, b, \mathcal{F}, \mathcal{G})$.
Output: (T modified pivot tableau, l status flag).
Begin
  $\mathcal{I}_B^+ := \{i \in \mathcal{F} : t_{ib} > 0\}$
  While $(\mathcal{I}_B^+ \neq \emptyset)$ **do**
    Let $r \in \mathcal{I}_B^+$ be arbitrary (driving variable), $rDone := false$.
    While $(rDone = false)$ **do**
      $\mathcal{J}_r^- := \{j \in \mathcal{G} \mid t_{rj} < 0\}$
      If $(\mathcal{J}_r^- = \emptyset)$ **then**
        The calling tableau is weakly degenerate, **Return**(T, r).
      Endif
      Let $s \in \mathcal{J}_r^-$, $\mathcal{K}_s := \{k \in \mathcal{F} : t_{kb} = 0, t_{ks} > 0\}$.
      If $(\mathcal{K}_s \neq \emptyset)$ **then** (when subtableau $T_{\mathcal{FG}}$ is strongly degenerate)
        $(T, s) :=$ DualSubMBU$(T, r, \{i \in \mathcal{F} \mid t_{ib} = 0\}, \mathcal{G})$.
        If $(s = -1)$ **then**
          The calling tableau is weakly degenerate, **Return**(T, r).
        Endif
      Endif
      $\theta_1 := \frac{|t_{rb}|}{t_{rs}}, \quad \theta_2 := \min\left\{\frac{|t_{kb}|}{t_{ks}} \mid k \in \mathcal{F}, t_{kb} < 0, t_{ks} > 0\right\}$.
      If $(\theta_1 \leq \theta_2)$ **then**
        pivot on t_{rs}, $rDone := true$
      else
        $q := \arg\min\left\{\frac{|t_{kb}|}{t_{ks}} \mid k \in \mathcal{F}, t_{kb} < 0, t_{ks} > 0\right\}$.
        pivot on t_{qs}.
      Endif
    Endwhile
  Endwhile
  The calling tableau is infeasible, **Return**$(T, -1)$.
End

Figure 3.7: The *PrimalSubMBU* starts with the solution of a subproblem of form Primal$_{\ominus}$.

The procedure starting the recursion and handling degeneracy can easily be formalized as shown in Figure 3.8.

Procedure DegProc

Input: $(T, \mathcal{I}_B^0, r)$.
Output: (T modified pivot tableau, l status flag).
Begin
(T, l):=(PrimalSubMBU$(T, \mathcal{I}_B^0, \{1, \ldots, n\}, n+1)$).
Return(T, l).
End

Figure 3.8: A possible anti degeneracy procedure, DegProc.

The relationship of the different subproblems is shown in Figure 3.9. The figure shows a possible primal-dual-primal calling sequence. Phrases *PrimalSubMBU* and *DualSubMBU* refer to the type of the subproblem. The structure of the basic tableaux corresponding to strongly degenerate bases has already been presented in Figure 3.5.

We now prove the finiteness of the MBU algorithm without any non-degeneracy assumption.

Theorem 3.2: The MBU type simplex algorithm is finite for any feasibility problem.

Proof. While the algorithm visits only non-degenerate or weakly degenerate problems, the algorithm carries out x_r increasing pivots according to Proposition 3.5; thus, the same basis may not return. Then the algorithm may not cycle. If the corresponding pivot tableau is strongly degenerate for a choice of a nonbasic variable, the algorithm calls the PrimalSubMBU or DualSubMBU subprocedures for strictly smaller problems, thus the depth of recursion is at most $2m \leq n+m$.

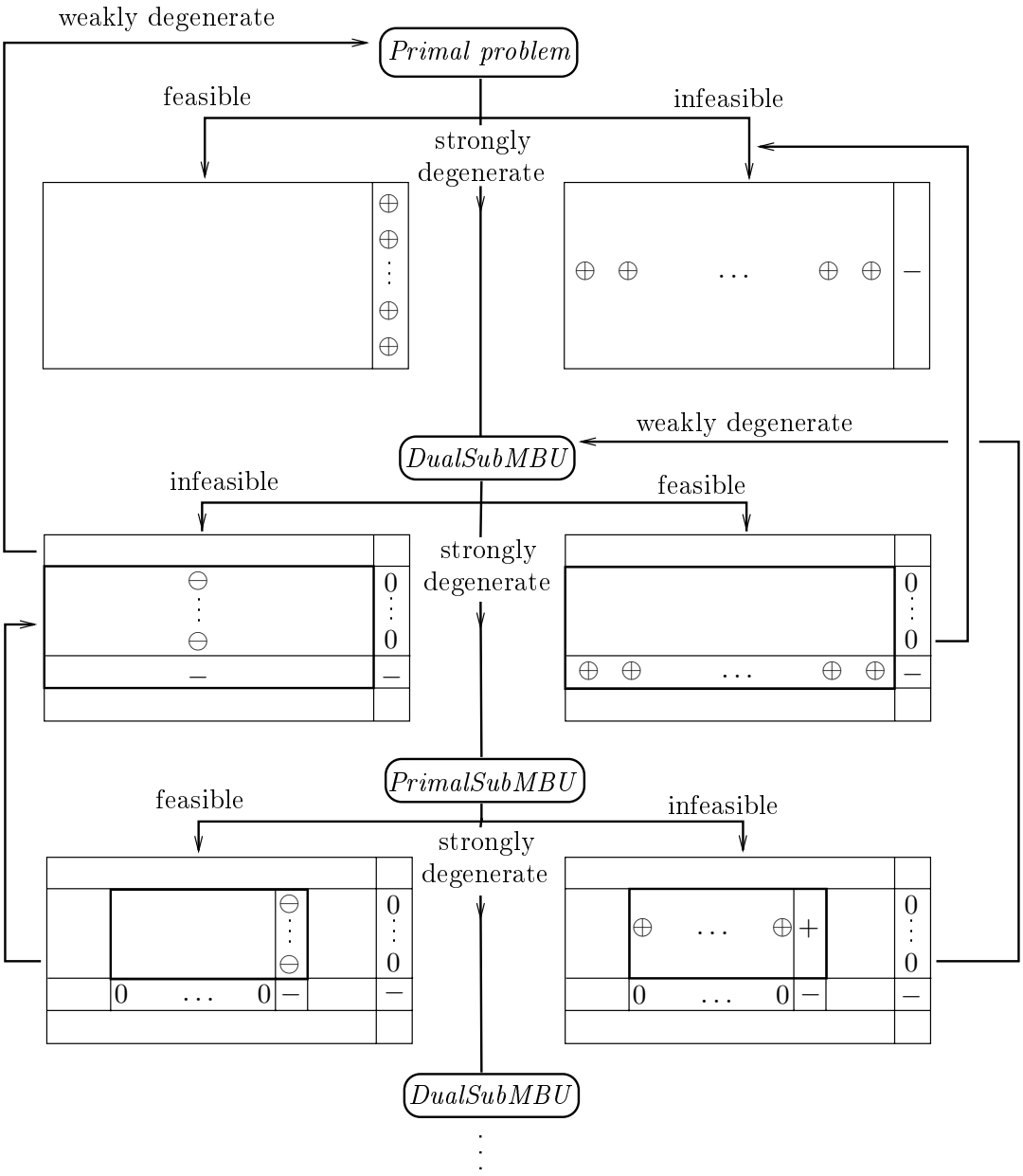

Figure 3.9: A Primal-Dual-Primal sequence of subproblems.

Consider the case when the recursively called *DualSubMBU* solves the subproblem. When it stops with an infeasible subtableau, then the corresponding calling (sub)problem became weakly degenerate to the proper nonbasic variable; thus, the procedure continues with an increasing pivot. When the *DualSubMBU* stops with a feasible subtableau, then the primal (sub)problem above became infeasible. This means the infeasibility of the original problem when the calling procedure was the *DegProc* procedure. Otherwise, the calling dual subproblem (one level above) has been transformed into a weakly degenerate form, thus it continues with an increasing pivot.

Similar connections hold when the *PrimalSubMBU* solves the corresponding subproblem.

Because the depth of recursion is bounded, and the returning sub-procedures provide the possibility of an increasing pivot for the calling procedure, no basis may occur twice. The number of different bases is finite, thus the algorithm is finite. □

Observe that both the algorithm and its recursive subalgorithms make increasing pivots to the corresponding (sub)problem and use recursion. Thus, it is possible to generalize the complexity bound of the non-degenerate and weakly degenerate case; however, because of the recursion, the implied bound greatly depends on the number of degenerate subproblem calls.

Although the analysis presented in Section 3.2 can be carried out for the first phase of the simplex algorithm, the presented anti cycling recursion procedure cannot be naturally applied to the simplex algorithm. The main reason for this is, when a

subproblem is feasible, we make use of the infeasibility of the driving variable to reach the conclusion that the calling problem is infeasible.

As already stated, it would be possible to solve the degenerate subproblems with an arbitrary pivot method, similarly to the case of the Hungarian method for linear programming [33], in which the criss-cross method was used [44].

In practice, the efficiency of the algorithm could be increased by exploring the freedom on the choice of the variable to enter the basis. This freedom may help, especially for numerically challenging problems.

3.4 Anti-Cycling Index Selection Rules

The prospect of cycling when using a pivot algorithm for solving an LFP or an LOP led researchers to study various methods to overcome this problem. Since cycling is possible only in the case of degeneracy and degeneracy is highly related with the data in the problem, perturbation of the data is a common practice to guarantee that cycling will not occur.

Bland discovered a surprisingly simple pivot strategy that never leads to cycling while studying linear systems in oriented matroid setting and anti-cycling rules have been of considerable interest since then. Bland's smallest indexed variable (SIV) rule, Zhang's last in first out / last out first in (LIFO/LOFI) and most often selected variable (MOSV) rules are shown to be anti-cycling rules when used with the simplex, MBU simplex and criss-cross methods. The common properties of these rules that serve their anti-cycling aspect are isolated and then captured in the concept of *s-monotonicity*.

Let i_j denote the index of the variable that enters (goes in) the basis and o_j denote the

index of the variable that leaves (goes out) of the basis at iteration j (the pivot position is (o_j, i_j)). A pair consisting of the indices of a basic and a nonbasic variable can be a possible pivot position if the corresponding basic tableau entry is nonzero. The sequence of possible pivot positions is defined as follows:

$$S = ((o_j, i_j): \quad o_j \in B_{j-1}, i_j \in N_{j-1}, t_{o_j,i_j} \neq 0, j = 1,2,\ldots)$$

A possible pivot sequence is a sequence showing all possible candidate pairs of entering/leaving variables regardless of any pivot selection rule. We can assume without loss of generality that a possible pivot sequence is infinite. This assumption may feel counterintuitive since what we aim is finite pivot rules, but a possible pivot sequence just stresses the *possibility* of an infinite sequence of pivot operations. The assumption can indeed be made w.l.o.g. since for any pivot operation transforming a tableau T to T', there is an inverse operation which transforms T' to T and we can always find a pivot position in any pivot tableau T. However, the number of bases for a given LFP is finite, therefore an infinite possible pivot sequence implies that some of the bases should appear infinitely many times. We denote the set of indices which appear in S infinitely many times by I^*.

An index selection rule is called s-monotone if the index of an entering or leaving variable at each iteration is selected according to a dynamic index priority vector $\mathbf{s} \in \mathbb{N}_0^n$ (created and updated according to the rule) which possesses the following properties:

1. $\mathbf{s}$ has an initial value of $\mathbf{s}_0$ (according to which the indices o_1 and i_1 will be selected) and is updated after every pivot operation. That is, $\mathbf{s}_1$ is defined after pivot operation in iteration 1 and is used to determine o_2 and i_2. In general, index priority vector $\mathbf{s}_{j-1}$ is used to determine o_j and i_j and $\mathbf{s}$ is updated to $\mathbf{s}_j$ after the pivot operation at position (o_j, i_j) takes place.

2. $\mathbf{s}_j > \mathbf{s}_{j-1}$.

3. For any iteration j, there exists an iteration $r \geq j$ in which the index $u \in I^* \cap B_{r-1}$ (u appears in S infinitely many times and is basic at iteration r) with minimal value in $\mathbf{s}_{r-1}$ is unique and there exists an iteration t where u will appear in S for the first time again and u will have the lowest priority among the indices in I^* according to $\mathbf{s}$ from iteration r to t.

An index selection rule can be regarded as having the effect of reorganizing variables by assigning them priorities and this reorganization is recorded in the vector $\mathbf{s}$. In short, the $\mathbf{s}$ vector for an index selection rule R records the way variables are reorganized according to R.

Here are the definitions of index priority vectors for three s-monotone index selection rules.

SIV: $\mathbf{s}_0 = (n, n-1, \ldots, 1), \quad \mathbf{s}_j = \mathbf{s}_{j-1} + \mathbf{1}$

LIFO/LOFI: $\mathbf{s}_0 = \mathbf{0}, \quad \mathbf{s}_j(k) = \begin{cases} k, & \text{if } k \in \{o_{j-1}, i_{j-1}\} \\ \mathbf{s}_{j-1}(k), & \text{otherwise.} \end{cases}$

MOSV: $\mathbf{s}_0 = \mathbf{0}, \quad \mathbf{s}_j(k) = \begin{cases} \mathbf{s}_{j-1}(k) + 1, & \text{if } k \in \{o_{j-1}, i_{j-1}\} \\ \mathbf{s}_{j-1}(k), & \text{otherwise.} \end{cases}$

The proofs that SIV, LIFO/LOFI and MOSV are s-monotone rules can be found in [17, 18] and will not be repeated here. For each of the above rules, a symmetric counterpart can be defined naturally. The largest indexed variable rule for SIV, the first-in-first-out

rule for LIFO/LOFI and the least-often-selected-variable rule for MOSV. By a slight change in the definition of index priority vectors from their counterparts, each of these can also be shown to be s-monotone rules.

3.4.1 Finiteness Proof Based on s-monotone Rules

When using the MBU simplex algorithm for solving a feasibility problem, if $\theta_2 = 0$, then a degenerate pivot is encountered. From among the candidates choosing the one with the largest s-vector value (in the case of SIV that means the variable with the minimal index is selected, in the case of MOSV, the variable which has moved the most number of times up to current iteration is selected) guarantees that the method will not cycle. The proof [17, 18, 38] uses the properties given in the definition of s-monotone index selection rule, therefore to show that an index selection rule has the anti-cycling property it is sufficient to show that it is s-monotone.

s-monotone rules can be incorporated into the frames provided by other pivot based methods to produce finite algorithms for LFP as well as LOP. For example, Adrienn Csizmadia recently studied criss-cross algorithm with s-monotone index selection rules for solving linear optimization problems [16].

3.5 Infeasibility Analysis for LFP

In a general optimization problem the goal is to find the maximum (or minimum) of a function over a set of constraints imposed on the variables of the problem. It is clear that in order to start seeking for an optimum solution, the system must be feasible in the first place, that is, the set of constraints must have a nonempty intersection. For the case of linear optimization this means that the polyhedron defined by the set of linear inequalities must be nonempty.

What happens then when a solver applied to an optimization problem returns that the

model is infeasible? Clearly, there is no solution to the problem as it is presented, but what does that mean about the real-world problem which still needs to be solved? For the real-world problem behind the model, this is an indication that something went wrong during the modeling stage (ill modeling). Possible reasons for infeasibility could be an erroneous data entry, a mis-specification in the direction of inequality constraints, or perhaps a conflict of interests. It is important to know which kind of problem occurred in order to help the model repairing process.

Therefore, when a solver terminates with a certificate of infeasibility, the model needs to be reconsidered in order to detect the cause of infeasibility and fix the model in the best way possible.

Example 3.1: (A small infeasible problem) A trivial example of an infeasible model would be one consisting of two obviously conflicting inequalities, such as,

$$a_1x_1 + a_2x_2 \leq b$$

$$a_1x_1 + a_2x_2 \geq b + 1$$

Let us look at a small but less obvious infeasible system. Consider the following system of inequalities:

$$3x_1 - x_2 \leq 0$$

$$x_1 - 3x_2 \geq 0$$

$$x_1 + x_2 \geq 4$$

A sketch of this system reveals that the three halfspaces have empty intersection, in other words, the system is infeasible.

We can also verify computationally that the linear system given above is infeasible by

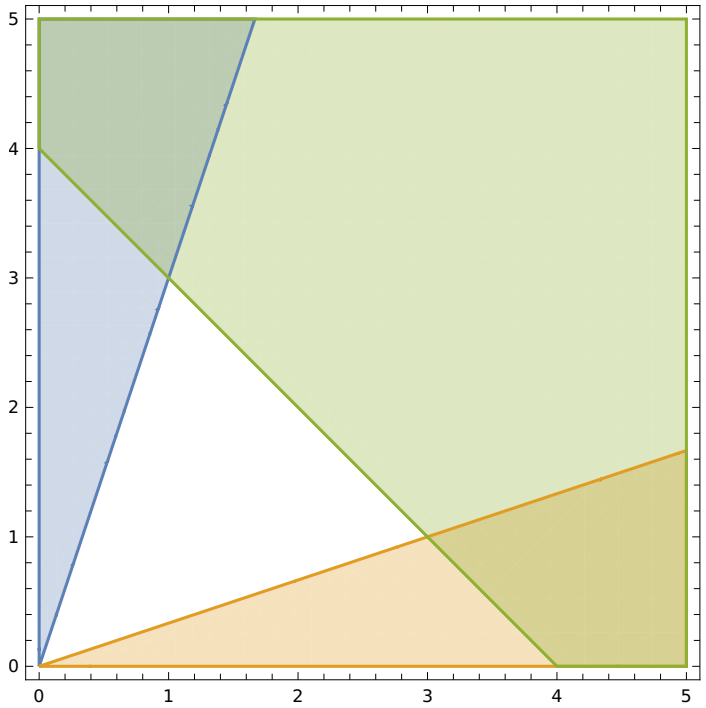

Figure 3.10: An infeasible system

applying any algorithm which is directly applicable to solve the general linear feasibility problem, like, for example, the MBU algorithm explained in Section 3.1 or the version of criss-cross algorithm for solving linear feasibility problems [44]. Furthermore, the verification can be done using any pivot algorithm for solving general linear optimization problems, such as the simplex or MBU simplex methods; but in that case we need to introduce a phase 1 objective function which minimizes the sum of the violations of each constraint at any point. A nonzero objective function value for the phase 1 objective function then indicates the infeasibility of the system.

Let us illustrate the application of our MBU type algorithm (Figure 3.4) on the above problem. Note that for the sake of simplicity the algorithm is applied without calling the anti-degeneracy procedure DegProc even when a strongly degenerate tableau is encountered. The ties for selecting entering/leaving variables will be broken by using the smallest indexed variable rule. Multiplying the $\geq$ type constraints by -1 on both

sides and then adding slack variables to all three constraints produces the following initial tableau:

	s_1	s_2	s_3	x_1	x_2	b
s_1	1	0	0	3	-1	0
s_2	0	1	0	-1	3	0
s_3	0	0	1	-1	-1	-4

The only candidate for the driving variable is s_3. Note that this tableau is strongly degenerate. After the first ratio test the algorithm proceeds with the pivot operation on $(1,4)$, that is, x_1 enters and s_1 leaves the basis. This pivot operation results in the following tableau:

	s_1	s_2	s_3	x_1	x_2	b
x_1	0.3333	0	0	1	-0.3333	0
s_2	0.3333	1	0	0	2.6667	0
s_3	0.3333	0	1	0	-1.3333	-4

The second pivot position is $(2,5)$ (x_2 enters and s_2 leaves the basis) and following the corresponding pivot operation we get the tableau:

	s_1	s_2	s_3	x_1	x_2	b
x_1	0.375	0.125	0	1	0	0
x_2	0.125	0.375	0	0	1	0
s_3	0.5	0.5	1	0	0	-4

This is the final tableau as the last row gives a certificate that the given linear system is infeasible.

Nowadays with the size and complexity of real-life linear optimization problems it is not uncommon to detect infeasibility. And when that happens the model must be analyzed and repaired. Before the infeasible model can be repaired an analysis of infeasibility and an explanation of the cause of infeasibility are needed. Because of the scale and complexity of models, automated assistance is used in diagnosing infeasibility. A common analysis technique which is utilized by most commercial packages is to find a useful isolation such as an irreducible infeasible subset (IIS) or a maximal feasible subsystem (MAX FS). An IIS is a subset of the original set of constraints which is itself infeasible, but all its proper subsets are feasible. A MAX FS, on the other hand, is the subsystem obtained when the least number of constraints are removed from the infeasible system in order to render it feasible. It is widely accepted that finding IISs (ideally small ones) and hence repairing a model based on the information obtained from such subsets is the best approach and it is in full use practically as well. Most commercial linear optimization packages, such as CPLEX and XPRESS-MP base their infeasibility analysis on IIS detection algorithms.

3.5.1 Irreducible Infeasible Subsets

An irreducible infeasible subset (IIS) is a set of constraints which is itself infeasible but every proper subset of which is feasible which means that all of the constraints in an IIS contribute to infeasibility. There could be more than one IIS in the model and it is possible that a single error presents itself in different IISs. Therefore in order to render the model feasible we must remove or modify at least one of the constraints in each IIS. An infeasible model is repaired in a cyclic fashion as follows:

1. Isolate an IIS
2. Repair the infeasibility in the IIS
3. Check whether the entire model is feasible yet; if not go to Step 1.

Steps 1 and 3 of the above procedure are fully automatized and we rely on computers for their outcomes. However, the second step requires human understanding of the model and therefore must be performed by an expert. Step 1 as well may require human intervention in case there are multiple IISs within the same infeasible model. In that case selection of which IIS to repair by a skilled analyzer could prove useful. Research results and experience with IISs show that the desired size of the IIS set to be found is as small as possible, because the lesser the number of constraints, the easier it is to repair infeasibility. However, when more than one IIS exists, it is usually not practical to find the minimum IIS due to the computational burden involved. Summarizing, we can say that an IIS is, ideally, a small set of conflicting constraints on which we can concentrate the analysis in order to diagnose and repair infeasibility.

The following result is important as it gives an upper bound on the number of inequalities that can be identified as an IIS [15].

Theorem 3.3: If there are n variables in an infeasible linear system, then the maximum cardinality of any IIS is $n+1$.

We now discuss some algorithms for identifying IISs of infeasible linear systems. The deletion filter and the additive method are general methods and they could be applied in analyzing infeasible nonlinear optimization problems or mixed integer problems as well as linear systems. However, the elastic filter is currently applicable to only linear systems due to computational limitations.

Given an infeasible set of constraints the deletion filter starts with the whole set and passes through each constraint once. So there are m main steps in the algorithm. In

Deletion filter

Input data: an infeasible set C of linear constraints
Output: set of constraints constituting a single IIS
Begin
  For ($c \in C$) **do**
    Temporarily drop c from C. $C' = C \setminus \{c\}$
    Test the feasibility of C'
    If (C' is infeasible) **then**
      $C = C'$
    Endif
  Endfor
  Return: C is a set constituting a single IIS
End

Figure 3.11: The deletion filter

each cycle it takes a constraint out of the set and tests the feasibility of the remaining constraints. If the remaining constraints are feasible this means the removed constraint affects the feasibility status and therefore is put back. On the other hand, if removing a particular constraint does not change the feasibility status, i.e., the system is still infeasible, then it is removed permanently. The deletion filter, after it passes through all the constraints once, returns a single IIS. In case there are more than one IISs within the infeasible system, which one is returned by the deletion filter depends on the order of the constraints tested.

Let us illustrate the algorithm on an example.

Example 3.2: Given that $C = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ is an infeasible set of constraints, assume that there are two (overlapping) IISs in C, $\{C_3, C_4, C_6\}$ and $\{C_4, C_5, C_7, C_8\}$. Let's run the algorithm on C: In the first iteration we drop C_1 temporarily and the remaining set of constraints $\{C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ is still infeasible. Therefore we drop C_1 permanently. $C = \{C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$. In the

second iteration, dropping C_2 gives us $\{C_3, C_4, C_5, C_6, C_7, C_8\}$ which is infeasible. Drop C_2 permanently. $C = \{C_3, C_4, C_5, C_6, C_7, C_8\}$. Third iteration: drop C_3 and test the feasibility status of $\{C_4, C_5, C_6, C_7, C_8\}$. This set is infeasible (since it contains an IIS), therefore drop C_3 permanently. $C = \{C_4, C_5, C_6, C_7, C_8\}$. Fourth iteration: now dropping C_4 from C gives $\{C_5, C_6, C_7, C_8\}$ whose feasibility status is "feasible". Therefore C_4 is returned back to C keeping it intact at the end of fourth iteration. In the fifth iteration we drop C_5 and obtain $\{C_4, C_6, C_7, C_8\}$ which is feasible. Returning C_5 back gives $C = \{C_4, C_5, C_6, C_7, C_8\}$. Sixth iteration: drop C_6 from C and test the feasibility status of $\{C_4, C_5, C_7, C_8\}$. This set is infeasible, therefore we drop C_6 permanently and update C: $C = \{C_4, C_5, C_7, C_8\}$. In the next iteration dropping C_7 gives the feasible set $\{C_4, C_5, C_8\}$ therefore we return C_7 back to C. $C = \{C_4, C_5, C_7, C_8\}$. In the last iteration we remove C_8 temporarily and get $C = \{C_4, C_5, C_7\}$ which is feasible. We return C_8 back and the algorithm terminates with the IIS $C = \{C_4, C_5, C_7, C_8\}$.

As it is seen from this example, from the two IISs in C the one whose first member is tested last is returned by the deletion filter. If the order in which the constraints are tested is changed then the deletion filter might isolate the other IIS. As an illustration, let us reorder the constraints in C as follows: $C = \{C_8, C_7, C_6, C_5, C_4, C_3, C_2, C_1\}$. Applying the deletion filter on C now isolates the IIS $\{C_3, C_4, C_6\}$.

Theorem 3.4: Deletion filter isolates the IIS whose first constraint is tested last.

Next we describe the additive method originally introduced by Tamiz et. al. [43] by using methods from LP goal programming. In their algorithm they use "deviational variables" similar to elastic variables (see Elastic Filter) and an elastic objective

function. In the following we give a simpler version of the algorithm due to Chinneck [13] without elastic variables and elastic objective function.

Additive method

Input data: an infeasible set of linear constraints C
Output: An IIS, I
Begin
  $T := \emptyset, I := \emptyset, i = 1$
  While $(i \leq |C|)$ **do**
    $T = T \cup c_i$
    If (T is infeasible) **then**
      $I = I \cup c_i$
      $T = I$
      $i = 1$
      If (I is infeasible) **then**
        Return I
      Endif
    Else
      $i = i + 1$
    Endif
  Endwhile
End

Figure 3.12: The additive method

Note that in the algorithm's description, C is the ordered set of constraints in the infeasible model, T is the current test set of constraints and I is the set of IIS members identified so far. Similarly to the deletion filter, additive method returns a single IIS upon termination. The IIS it isolates depends on the order of the constraints tested in case there are more than one IISs in the problem.

Theorem 3.5: Additive method isolates the IIS whose last constraint is tested first.

Both the deletion filter and the additive method have the major drawback of having to check the feasibility status of a linear system many times over.

3.5.2 An Elastic Model

Useful information about an infeasible model can be obtained if the constraints can be violated in a graceful manner. For example, in the simplex method a basic feasible solution is required to start the procedure. If the initial basic solution is not feasible a phase 1 procedure is used in which nonnegative artificial variables a_i are added to all equality and $\geq$ type constraints, which allows those constraints to be violated so that an initial basic "feasible" solution can be established. This initial solution is feasible in the space consisting of the original and the artificial variables, but it is not feasible in the space consisting of just the original variables. In order to find a basic feasible solution for the original problem (if there exists one), a phase 1 objective is to minimize the sum of the artificial variables, that is, minimize $V = \sum a_i$. If V reaches a minimum value of zero, then all of the artificial variables are zeroes, hence a feasible solution has been found for the original model, and the simplex method can now proceed to find the solution of the original problem. If the minimum value of V is not zero, then at least one of the artificial variables cannot be forced to zero, so the corresponding constraint remains violated in the original variable space, and the LOP as a whole is determined to be infeasible.

In this way, the linear equality and $\geq$ constraints are able to stretch, or violate their original bounds and the value of the associated artificial variable corresponds directly to the size of the adjustment of the right hand side needed to provide a feasible solution in the original variable space.

The elastic model used by Chinneck [13] extends this idea to allow all forms of constraints to adjust in all directions, as originally described by Brown and Graves [12]. A fully elastic program adds a nonnegative elastic variable (or variables)

s_i (or s_i' and s_i'') to every constraint. This allows a solver to find a "feasible" solution for the original infeasible problem. The rules for adding elastic variables are as follows:

Nonelastic Constraint	Elastic Version
$\sum_j a_{ij}x_j \geq b_i$	$\sum_j a_{ij}x_j + s_i \geq b_i$
$\sum_j a_{ij}x_j \leq b_i$	$\sum_j a_{ij}x_j - s_i \leq b_i$
$\sum_j a_{ij}x_j = b_i$	$\sum_j a_{ij}x_j + s_i' - s_i'' = b_i$

An elastic constraint stretches to violate its original bounds when one of its elastic variables takes on a positive value. Stretching is resisted by the elastic objective function (minimize $\sum_i s_i$) which replaces the original objective function.

This is the essence of the classical elastic model description; elastic variables are added to all constraints, and equality constraints are elasticized in both directions. Note that integer restrictions cannot be elasticized, so elastic filtering can be applied only to LPs, NLPs, and the linear part of MIPs. In this sense it is slightly less general than the deletion filter and the additive method.

The elastic filter of Chinneck and Dravnieks [14] makes use of the elastic model described above. At the beginning all constraints are elasticized, but since the original model is infeasible, at least one constraint must stretch to achieve a feasible solution for the elastic program. The elastic variables are removed from any constraints that stretch, this enforces the constraint in the next round. The cycle repeats until enough elastic variables have been removed that the partly-elastic model becomes infeasible. At this point the de-elasticized constraints constitute a small infeasible set that is not necessarily an IIS, but that has some very desirable properties.

In the case that the MBU algorithm terminates with a certificate of infeasibility for a LFP, we propose a specialized elastic model which can be built and solved with a modified MBU algorithm to obtain an IIS for the infeasible problem. Although we have yet to complete the definition of this procedure and prove its correctness, we give an explanation of the proposed elastic model in the following.

Suppose that we solve the following feasibility problem and obtain a certificate that it is infeasible. The system could contain inequality and/or equality constraints.

(1) $A\mathbf{x}(\leq,\geq,=)\mathbf{b}$

(2) $\mathbf{x} \geq \mathbf{0}$

In (1) we fix $\mathbf{b}$ to be nonnegative without loss of generality for if there is a constraint i where $b_i < 0$, we can multiply both sides of that inequality or equation with -1.

With $\mathbf{b} \geq 0$, the inequalities of the form $\sum_j a_{ij}x_j \leq b_i$ are feasible at the origin but the inequalities of the form $\sum_j a_{ij}x_j = b_i$ and $\sum_j a_{ij}x_j \geq b_i$ are not feasible at the origin (except when $b_i = 0$).

We can "relax" the problem by introducing slack variables for $\leq$ constraints and elasticity variables for $\geq$ and $=$ constraints as follows:

$$\sum_j a_{ij}x_j + s_i = b_i, \quad s_i : \text{slack variable}$$

$$\sum_j a_{ij}x_j + u_i = b_i, \quad \sum_j a_{ij}x_j + v_i = b_i, \quad u_i, v_i : \text{elasticity variables}$$

In the new model including slack and elasticity variables, all variables are nonnegative,

i.e., $\mathbf{x}, \mathbf{s}, \mathbf{u}, \mathbf{v} \geq 0$. Furthermore, we impose a new constraint $\mathbf{e}^T\mathbf{u} + \mathbf{e}^T\mathbf{v} = \mathbf{0}$ to resist stretching of elasticized constraints.

At this stage we have an "almost feasible" solution for our elastic model (except for the last constraint) which is:

$$\mathbf{x} = \mathbf{0}, \quad s_i = b_i, \quad u_i = b_i, \quad v_i = b_i$$

We introduce an elastic variable for this last constraint:

$$\mathbf{e}^T\mathbf{u} + \mathbf{e}^T\mathbf{v} + \lambda = 0, \quad \lambda \geq 0$$

Now the starting tableau for the elastic problem has the following structure:

The only infeasible variable in the starting tableau is λ. Therefore, all the effort can be concentrated on finding λ-increasing pivots trying to push λ to 0 (feasible status, which it will never have) as much as possible. The use of MBU simplex algorithm to solve this special elastic model to obtain useful information (such as an IIS) about the original infeasible model is one of the open problems we think can be solved successfully.

Chapter 4

ORIENTED MATROID FEASIBILITY PROBLEM

In this chapter we consider the linear feasibility problem and its solution by finite pivot rules in an abstract setting by using oriented matroids. Rockafellar was the first to observe that the strong duality theorem of linear optimization can be seen as a statement about sign patterns of vectors in complementary subspaces of $\mathbb{R}^n$ [41]. Following this idea, many aspects of linear optimization, including the simplex and criss-cross methods with finite pivot rules were described in the context of oriented matroids.

As Björner et. al. [7] point out, "Oriented matroid framework can add to the understanding of the combinatorics and the geometry of the simplex method for linear programming. In fact, the oriented matroid approach gives a geometric language for pivot algorithms, interpreting linear programs as oriented matroid search problems."

We first describe the oriented matroid feasibility problem (OMFP) and then present a finite criss-cross algorithm with LIFO/LOFI index selection rule that solves the problem. The other s-monotone index selection rules described in section 3.4 can also be used within the criss-cross pivot method's framework in order to solve the OMFP. We close this chapter with two proofs. The first one is the finiteness proof of the presented algorithm and the second one is a constructive proof of the Farkas lemma for oriented matroids.

The results in this chapter are mainly based on the paper [4].

4.1 OM Feasibility Problem

From the perspective of linear feasibility problems, an oriented matroid can be seen as a certain system of signed vectors that captures important properties of a system of linear inequalities. Basic background material on matroids and oriented matroids are given in section 2.7.

Let $E = \{e_1, ..., e_n\}$ be a finite set and $e_1 \in E$ be a fixed element. Also, let $M = (E, O)$ and $M^* = (E, O^*)$ be dual pairs of oriented matroids. First we need to introduce the concept of feasible circuits and cocircuits of oriented matroids.

Definition 4.1: The oriented circuit $X = (X^+, X^-) \in O$ is called feasible, if $e_1 \in X^+$ and $X^- = \emptyset$. Similarly, the oriented cocircuit $Y = (Y^+, Y^-) \in O^*$ is called dual feasible, if $e_1 \in Y^+$ and $Y^- = \emptyset$

Just like in the case of solving an LFP, we need a proof or a certificate of solvability of an OMFP. Next, we state an alternative theorem related to the feasibility problem of oriented matroids. This theorem is a generalization of the Farkas lemma for oriented matroids. A proof of the theorem is based on the finiteness of a criss-cross type algorithm. The details of the algorithm, its finiteness and the application to the feasibility problem of oriented matroids will be discussed in section 4.1.1.

Theorem 4.1: Let $M = (E, O)$ and $M^* = (E, O^*)$ be a dual pair of oriented matroids. Then exactly one of the following statements hold

(a) there is a feasible circuit in $M = (E, O)$,

(b) there is a feasible cocircuit in $M^* = (E, O^*)$.

Klafszky and Terlaky [34] proved a variant of the Farkas lemma using Terlaky's criss-cross method [44, 45]. Theorem 4.1 is a generalization of that variant of the Farkas lemma. Theorem 4.1 was first proved by Bland [8], while the first constructive, algorithmic proof is due to Klafszky and Terlaky [32]. For underlying the connection between Theorem 4.1 and the variant of the Farkas lemma presented in Klafszky and Terlaky [34], we introduce the concept of tableau for oriented matroids as it is done in Bland [8], Terlaky [45] and Klafszky and Terlaky [32].

Let $\mathcal{B}$ be the set of all bases of the matroid $\bar{M}$ underlying the oriented matroid M and r denote the rank of $\bar{M}$. Then for any $B = \{e_{b_1}, \ldots, e_{b_r}\} \in \mathcal{B}$ there is for each $i = 1, \ldots, r$ a unique cocircuit $D_i \in \bar{O}^*$ of the underlying matroid $\bar{M}$ having $D_i \cap B = \{e_{b_i}\}$.

Let Y_{b_i} be the oriented cocircuit associated with the basic element $e_{b_i}, i = 1, \ldots r$, in the way that $e_{b_i} \in Y_{b_i}^+$ and $\bar{Y}_{b_i} \cap B = \{e_{b_i}\}$. The set $\{Y_{b_1}, \ldots, Y_{b_r}\}$ is called the *fundamental set* of oriented cocircuits with respect to the base B. Then the matrix having its b_ith row equal to the signed incidence vector of Y_{b_i} is denoted by $T(B)$ and called the tableau corresponding to the base B. That is, the entry t_{ij} of $T(B)$ is the sign of e_j in $Y_i \in \{Y_{b_1}, \ldots, Y_{b_r}\}$. One important property of $T(B)$ is that for $e_k \notin B$, the e_k column of $T(B)$ corresponds to the oriented circuit $X = (X^+, X^-)$ having $X^+ = \{e_{b_i} : t_{ik} = +1\}$ and $X^- = \{e_{b_i} : t_{ik} = -1\} \cup \{e_k\}$.

Example 4.1: Continuing with the previous example, consider the base $B = \{1, 2, 4, 6\}$. Then the following set of oriented cocircuits

$$\{(+,0,+,0,+,0), (0,+,+,0,+,0), (0,0,-,+,0,0), (0,0,0,0,-,+)\}$$

is the set of fundamental cocircuits corresponding to the base B. The tableau corresponding to B is

+	0	+	0	+	0
0	+	+	0	+	0
0	0	-	0	0	0
0	0	-	+	0	0
0	0	0	0	-	0
0	0	0	0	-	+

Figure 4.1: The tableau corresponding to the base given in example 4.1

4.1.1 A Criss-Cross Type Algorithm and Its Finiteness

Now we present a finite algorithm, which is a variant of criss-cross algorithm [44, 45] using Zhang's LIFO/LOFI rule [49, 50] for solving the feasibility problem of oriented matroids. This algorithm produces either a feasible circuit $X \in O$ or a dual feasible cocircuit $Y \in O^*$.

The vector $\mathbf{u}_r : E \to \mathbf{N}_0$ takes place in the algorithm. Its definition is

$$\mathbf{u}_r(i) = \begin{cases} r, & \text{if the } i^{th} \text{ element moves in the } r^{th} \text{ iteration} \\ \mathbf{u}_{r-1}(i), & \text{otherwise.} \end{cases}$$

Initially, $\mathbf{u}_0 = (0, \ldots, 0)$.

Note that the vector $\mathbf{u}$ is defined in exactly the same way as the vector $\mathbf{s}$ for the anti-cycling LIFO/LOFI index selection rule of chapter 3.

Our algorithm differs from that of Klafzky and Terlaky [32] in the way that there is no fixed ordering at the beginning of the algorithm between the indices of the elements but an ordering depending on the pivot selection rule is built up until the last element was moved for the first time. This ordering will guarantee the finiteness of this modified criss-cross algorithm.

criss-cross algorithm for OMFP

Input data: A base B_0 and the corresponding tableau $T(B_0)$,
$\tilde{I} = \{1,2,\ldots,n\}$ as the index set of elements in E, $r = 0$.

Step 0: Let $B := B_r$.
If $e_1 \notin B$ **then** go to **Step 1**,
else go to **Step 2.**

Step 1: **If** $t_{i1} \in \{-1,0\}$ for all $e_i \in B$ **then**
$(-X_1)$ is a primal feasible circuit, **Return**;
else let $J = \{i \in \tilde{I} : t_{i1} = +1 \text{ for } e_i \in B\}$, $a_J := \max_{i \in J} \mathbf{u}_r(i)$ and
$\hat{J} = \{j \in J : \mathbf{u}_{r-1}(i) = a_J \text{ for all } i \in J\}$
Choose an arbitrary index $k \in \hat{J}$
Make a pivot operation (e_1 enters and e_k leaves the base)
Let $\mathbf{u}_r(k) := r$, $r := r+1$ and continue with **Step 0**.

Step 2: **If** $t_{1j} \in \{0,+1\}$ for all $e_j \notin B$ **then**
Y_1 is a dual feasible cocircuit, **Return**;
else let $K := \{j \in \tilde{I} : t_{1j} = -1 \text{ for } e_j \notin B\}$, $a_K := \max_{i \in K} \mathbf{u}_r(i)$ and
$\hat{K} := \{i \in K : \mathbf{u}_{r-1}(i) = a_K \text{ for all } i \in K\}$
Choose an arbitrary index $k \in \hat{K}$
Make a pivot operation (e_k enters and e_1 leaves the base)
Let $\mathbf{u}_r(k) := r$, $r := r+1$ and continue with **Step 0**.

Figure 4.2: Criss-cross algorithm with LIFO/LOFI Index Selection Rule

In this section we will prove that criss-cross with LIFO/LOFI index selection rule is finite which implies that theorem 4.1 holds. The finiteness proof of the algorithm follows the main steps of Klafszky and Terlaky's proof [32].

Lemma 4.1: Criss-cross algorithm with LIFO/LOFI index selection rule for solving OMFP is finite.

Proof. Let us suppose the contrary that the algorithm is not finite. The fact that the number of different bases is finite implies that the algorithm cycles, that is, starting from a base B, the algorithm produces the same base again after a certain number of steps.

Let $E^c = \{e_i : e_i$ changes its place with e_1 through the cycle$\}$.

Let us consider the sequence of pivot tableaus generated by the algorithm and let us denote by $T(B')$ the base which satisfies the criteria; (a) there is a variable $e_q \in E^c$ which changes its basic status for the first time (i.e. $u_r(q) = 0$ at base B') and (b) after this pivot tableau all the variables in E^c have changed their basic status at least once. Here we need to make the following observation; after the base B' each member of $E^c \cap B$ would have different $u_r(i)$ values at each forthcoming iteration. This is due to the fact that in each iteration an e_k element and the e_1 element has changed their basic status.

We can suppose without loss of generality that e_q enters B'.

Let B'' be the base corresponding to the iteration in which e_q is chosen for the first time to leave the base thus $q \in \{i \in \tilde{I} : e_i \in B''$ and $t_{i1} = +1\}$ and according to the pivot rule $u_r(q) > u_r(i)$ at the iteration corresponding to base B''. This means that $X_1^+ \subset \{e_q\} \cup (B'' \cup B')$. The oriented cocircuit Y_1 as row e_1 of B' and the oriented circuit X_1 as column e_1 of B'' have the following properties

$1'.\ e_1 \in Y_1^+, e_q \in Y_1^-$ $\qquad$ $1''.\ e_1 \in X_1^-, e_q \in X_1^+$

$2'.\ Y_1^- \subset \{e_q\} \cup [(E - B') - E^c]$ $\qquad$ $2''.\ X_1^- \subset B'' - \{e_q\} \cup \{e_1\}$

$3'.\ Y_1^+ \subset (E - B') - \{e_q\} \cup \{e_1\}$ $\qquad$ $3''.X_1^+ \subset \{e_q\} \cup (B'' \cap B')$

Properties $1'$ and $1''$ imply that $X_1 \cap Y_1 \neq \emptyset$.

Let us consider $2'$ and $2''$. We have $e_q \in Y_1^-$ and $e_q \notin X_1^-$. Let us consider an element $e_i \neq e_q$. If $e_i \in Y_1^-$, i.e., $e_i \in [(E - B') - E^c]$, this means e_i is nonbasic at B' and it never

changes its basic status, therefore e_i cannot belong to $B'' - \{e_q\} \cup \{e_1\}$, i.e., $e_i \notin X_1^-$. Thus properties $2'$ and $2''$ imply that $X_1^- \cap Y_1^- = \emptyset$.

Now consider the properties $3'$ and $3''$. We have $e_q \notin Y_1^+$ and $e_q \in X_1^+$. Let us consider an element $e_i \neq e_q$. If $e_i \in X_1^+$, i.e., $e_i \in B'' \cap B'$. Therefore $e_i \notin E - B'$. Thus properties $3'$ and $3''$ imply that $X_1^+ \cap Y_1^+ = \emptyset$; which contradict orthogonality. □

4.1.2 A Constructive Proof of the Farkas Lemma

Now we can prove the Farkas lemma.

Proof of the Theorem 4.1: The orthogonality of oriented circuits and cocircuits imply that both (a) and (b) cannot occur simultaneously. The algorithm stops either in Step 1 with a feasible circuit or in Step 2 with a feasible cocircuit. Since the algorithm is finite, the proof is complete. □

Replacing the LIFO pivot rule with the most-often-selected-variable (MOSV) pivot rule or any s-monotone pivot rule the algorithm 4.2 remains finite.

Chapter 5

FURTHER RESEARCH SUGGESTIONS

In section 3.2 we derive an upper bound for the number of iterations MBU makes for solving and LFP. Further investigations are needed to find such problem classes where the new upper bound can easily be determined; that is, the values of Δ_A and $\Delta_{\max}$ are easily computable. A naturally arising problem class would be one for which the matrix A is totally unimodular. Most of the problems coming from combinatorial optimization are highly degenerate; thus, the complexity estimate would only bound the number of non-degenerate and weakly degenerate pivot steps. If A is totally unimodular and $\mathbf{b} \in \mathbb{Z}^m$, we get $K \leq \|\mathbf{b}\|_1$. This means that in this specific case K depends pseudo-polynomially on the (binary) input size. It would also be interesting to find proper perturbations for given problem classes to handle strong degeneracy. Perturbation techniques known from the literature (like the ε-perturbation technique) do not present themselves as efficient choices because they, usually drastically, affect the values of Δ_A and $\Delta_{\max}$ as well.

In section 3.4 we explain the concept of s-monotonicity and we give three index selection rules which are s-monotone. A possible research direction from here on is to develop other s-monotone rules and study their practical as well as theoretical characteristics. Another possibility is to investigate hybrid methods and their behavior on special problem classes such as the network flow problem.

In section 3.1 we state MBU simplex algorithm for solving LFPs. For solving LOPs a primal-dual version of MBU can be defined so that the algorithm could start from a primal feasible or a dual feasible basis. Also, inspired by the fact that MBU visits both primal and dual infeasible bases (even though this is temporary and one of the two feasibilities is eventually restored), we think that a generalized MBU simplex algorithm which starts from a neither primal nor dual feasible basis (just like the criss-cross method) can be developed.

The linear feasibility problem has some interesting applications stemming from diverse fields. These by themselves constitute of important research topics some of which we would like to mention. For example, the search for a linear classifier consistent with all available examples is a problem of finding a feasible solution to a linear optimization problem.

As mentioned in section 3.5.2, we think a special elastic model can be built for infeasible LFPs and MBU simplex algorithm developed for solving feasibility problems can be utilized in a way to isolate IISs or in the worst case gather useful information about the original infeasible model.

Writing programs to test for any computational aspect of oriented matroids seem to be difficult in a procedural language such as Python. A functional language like Haskell would be more suited for testing the algorithms for oriented matroids. We think such an endeavour might be beneficial in order to gain more insight and be able to test quickly for some heuristics in search for more anti-cycling pivot rules.

Given an infeasible linear system, is the corresponding oriented matroid necessarily

infeasible? Can we generate feasible oriented matroids starting from an infeasible one by using a generation approach that incrementally extends oriented matroids by adding single elements? Solving this more abstract problem could also prove useful in analyzing infeasibility of linear systems.

REFERENCES

[1] Anstreicher, K. M., & Terlaky, T. (1994). A monotonic build-up simplex algorithm for linear programming. *Operations Research, 42*(3), 556–561.

[2] Avis, D., & Chvátal, V. (1978). Notes on Bland's pivoting rule. *Mathematical Programming Study, 8*, 24–34.

[3] Bachem, A., & Kern, W. (1992). *Linear Programming Duality*, Springer-Verlag.

[4] Balogh, L., Bilen, F., & Illés, T. (2002). A simple proof of the generalized Farkas lemma for oriented matroids. *Pure Mathematics and Applications, 13*, 423–431.

[5] Bilen, F., Csizmadia, Z., & Illés, T. (2007). Anstreicher-Terlaky type monotonic simplex algorithms for linear feasibility problems. *Optimisation Methods and Software, 22*(4), 679–695.

[6] Bilen, F., Csizmadia, Z., & Illés, T. (2007). A new analysis for monotonic type simplex algorithms for feasibility problems (in Hungarian). *Alkalmazott Matematikai Lapok, 24*, 163–185.

[7] Björner, A., Las Vergnas, M., Sturmfels, B., White, N., & Ziegler, G. (1999). *Oriented Matroids, Second Edition*, Cambridge University Press.

[8] Bland, R. G. (1977). A combinatorial abstraction of linear programming. *Journal of Combinatorial Theory (B), 23*, 33–57.

[9] Bland, R. G. (1977). New finite pivoting rules for the simplex method. *Mathematics of Operations Research, 2*, 103–107.

[10] Bland, R. G., & las Vergnas, M. (1978). Orientability of matroids. *Journal of Combinatorial Theory (B), 24*, 94–123.

[11] Borgwardt, K.H. (1987). The simplex method: A probabilistic analysis. *In the series: Algorithms and Combinatorics, Vol. 1*, Springer, Berlin.

[12] Brown, G., & Graves, G. (1975). Elastic programming: A new approach to large-scale mixed integer optimization. *ORSA/TIMS Conference*, Las Vegas.

[13] Chinneck, J. W. (2008). *Feasibility and Infeasibility in Optimization*. Springer.

[14] Chinneck, J. W., & Dravnieks, E. W. (1991). Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing, 3*, 157–168.

[15] Chvátal, V. (1983). *Linear Programming*. W. H. Freeman and Company, New York.

[16] Csizmadia, A. (2022). Finiteness of the criss-cross algorithm for the linear programming problem with s-monotone index selection rules. *Pure Mathematics and Applications, 30*(2), 58–70.

[17] Csizmadia, Z. (2007). New pivot based methods in linear optimization, and an application in the petroleum industry. *Ph.D Thesis*, Eötvös Loránd University.

[18] Csizmadia, Z., Illés, T., & Nagy, A. (2012). The s-monotone index selection rules for pivot algorithms of linear programming. *European Journal of Operations Research, 221*(3), 491–500.

[19] Dantzig, G. B. (1948). Programming in a linear structure. Comptroller, United States Air Force, Washington DC.

[20] Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey.

[21] Farkas, Gy. (1894). A Fourier-féle mechanikai elv alkalmazásai (The applications of the mechanical principle of Fourier). *Mathematikai és Természettudományi Értesitö, 12*, 457–472.

[22] Farkas, Gy. (1896). A Fourier-féle mechanikai elv alkalmazásainak algebrai alapjáról (On the algebraic background of the applications of the mechanical principle of Fourier). *Mathematikai és Fizikai Lapok, 5*, 49–54.

[23] Folkman, J., & Lawrence, J. (1978). Oriented matroids. *Journal of Combinatorial Theory (B), 25*, 199–236.

[24] Illés, T. (1999). Linear programming. *Lecture Notes*, Eastern Mediterranean University, Department of Mathematics, Famagusta.

[25] Illés, T., & Molnar-Szipai, R. (2014). On strongly polynomial variants of the MBU simplex algorithm for a maximum flow problem with non-zero lower

bounds. *Optimization, 63*(1), 39–47.

[26] Jensen, D. L. (1985). Coloring and duality: Combinatorial augmentation methods. *Ph.D Thesis*, Cornell University.

[27] Fukuda, K., & Terlaky, T. (1997). Criss-cross methods: A fresh view on pivot algorithms. *Mathematical Programming, 79*, 369–395.

[28] Fukuda, K., & Terlaky, T. (1999). On the existence of a short admissible pivot sequence for feasibility and linear optimization problems. *Pure Mathematics with Applications, 10*, 431–447.

[29] Gordon, G., & McNulty, J. (2012). *Matroids: A Geometric Introduction*. Cambridge University Press.

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

[31] Khachiyan, L. G. (1979). A polynomial algorithm in linear programming (in Russian). *Dokl. Akad. Nauk SSSR, 244*(5), 1093–1096.

[32] Klafszky, E., & Terlaky, T. (1987). Remarks on the feasibility problem of oriented matroids. *Ann. Univ. Sci. Budapestiensis de Rolando Eötvös Nominatae, Sectio Computatorica, 7*, 155–157.

[33] Klafszky, E., & Terlaky, T. (1989). Variants of the Hungarian method for solving linear programming problems. *Optimization, 20*(1), 79–91.

[34] Klafszky, E., & Terlaky, T. (1991). The role of pivoting in proving some fundamental theorems of linear algebra. *Linear Algebra and its Applications, 151*, 97–118.

[35] Klee, V., & Minty, G. J. (1972). How good is the simplex algorithm?. In *Inequalities*, Academic Press, New York.

[36] Maros, I. (2003). *Computational Techniques of the Simplex Method*. Kluwer Academic Publishers, Boston.

[37] Murty, K. G. (1983). *Linear Programming*. John Wiley & Sons, New York.

[38] Nagy, A. (2014). On the theory and applications of flexible anti-cycling index selection rules for linear optimization problems. *Ph.D Thesis*, Eötvös Loránd University.

[39] NETLIB Internet Repository of Linear Programming and Linear Feasibility Problems. *https://netlib.org/lp/data/*

[40] Prékopa, A. (1980). On the development of optimization. *American Mathematical Monthly, 87*, 527–542.

[41] Rockafellar, R. T. (1969). The elementary vectors of a subspace of R^n. In *Combinatorial Mathematics and its Applications,* Proceedings of the Chapel Hill Conference (1967), eds. R. G. Bore and T. A. Dowling, University of North Carolina Press, 104–127.

[42] Smale, S. (1998). Mathematical problems for the next century. *The Mathematical Intelligencer, 20*, 7–15.

[43] Tamiz, M., Mardle, S. J., & Jones, D. F. (1996). Detecting IIS in infeasible linear programmes using techniques from goal programming. *Computers and Operations Research, 23*, 113–119.

[44] Terlaky, T. (1985). A convergent criss-cross method. *Optimization, 16*, 683–690.

[45] Terlaky, T. (1987). A finite criss-cross method for oriented matroids. *Journal of Combinatorial Theory (B), 42*, 319–327.

[46] Terlaky, T., & Zhang, S. (1993). Pivot rules for linear programming: A survey on recent theoretical developments. *Annals of Operations Research, 46*, 203–233.

[47] Todd, M. J. (1986). Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Mathematical Programming, 35*, 173–192.

[48] Tutte, W. T. (1965). Lectures on matroids. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics, 69*, 1.

[49] Zhang, S. (1991). On anti-cycling pivoting rules for the simplex method. *Operations Research Letters, 10*, 189–192.

[50] Zhang, S. (1999). New variants of finite criss-cross pivot algorithms for linear programming. *European Journal of Operational Research, 116*(3), 607–614.