Linear Programming: Pivoting on Polyhedra and Arrangements

Bohdan Lubomyr Kaluzny

Doctor of Philosophy

School of Computer Science

McGill University

Montreal,Quebec

2005-12-31

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Doctor of Philosophy

Your file Votre référence
ISBN: 978-0-494-25180-5
Our file Notre référence
ISBN: 978-0-494-25180-5

DEDICATION

This thesis is dedicated to my family - a tough-loving bunch that have pushed me to great heights.

ACKNOWLEDGEMENTS

Use what talent you possess: the woods would be very silent if no birds sang except those that sang best - Henry Van Dyke

Nine years have passed since I first entered the halls, classrooms, and labs of McGill University. I am indebted to many people for giving me assistance these last few years during my doctoral studies.

First and foremost, I thank my supervisor David Avis for his teaching, guidance, professionalism, patience, helpful discussions and motivational kicks to get me going. It is an honour to be his student and inconceivable that this thesis would come to fruition without his direction. I am also grateful that David provided financial assistance for the last year of my studies after my scholarships (for which I praise NSERC and FQRNT) came to term. Thanks boatman!

I thank members of my PhD advisory committee; David Avis, David Bryant, Xiao-Wen Chang, Komei Fukuda, and Adrian Vetta, for their time and useful suggestions. My coauthors during my stay, namely David Avis, Komei Fukuda, and David Titley-Péloquin, have benefitted me greatly with their ideas, recommendations, and expert editing. It has been a great learning experience to be a member of the computational geometry group and discrete mathematics group. In particular Luc Devroye, Godfried Toussaint, Adrian Vetta, and Sue Whitesides have provided exemplary teaching and an ideal research environment. Of course (uncle) Bruce Reed deserves special mention due to his encouragement, humorous discussions, and his famous "wagga waggas" which will be missed.

In my opinion McGill University has lived up to its name. The institution provided me with all the necessary tools and services to succeed in my research. It has been inspiring to work with so many brilliant minds accessible. The School

of Computer Science has collected outstanding individuals to run the department smoothly. Andrew Bochego, Ron Simpson, and the rest of the SOCS system staff have been exceptional in helping me with basic problems. Lucy St. James, Vicki Keirl, and the remainder of the administrative staff, have been of great assistance in all general matters - including unlocking my office door when I forgot my keys! My office mates of McConnell 232 have provided me with an optimal work environment, helping make every day productive and fun. In particular, I wish Conor and Perouz all the best in their studies and future endeavors. If I can do it, you can do it too!

External to McGill there have been many people influential in my life which have aided me while I prepared this thesis. From my brothers of the DBHL to my dearest friends, I thank you for giving me recluse, keeping me grounded and focused while providing great friendship that I truly cherish. Finally, I thank all my family, especially my parents, for the moral support, love, and endless encouragement they give me. They believe in me when I falter; they give me the courage and inspiration to sing. I share this exciting accomplishment with you!

STATEMENT OF ORIGINALITY

I certify that this thesis, and the research to which it refers, are original contributions to knowledge, and the product of my own work. The ideas and quotations from the work of other researchers are fully acknowledged, cited, and referenced. Some of the work presented has been previously published (or submitted for publication) as listed below.

- **Chapter 1, section 1.2.1**: (with David Avis) "Solving Inequalities and Proving Farkas' Lemma Made Easy," in *American Mathematics Monthly* [7].
- **Chapter 2, sections 1-3**: (with David Avis and David Titley-Péloquin) "Visualizing and Constructing Cycles in the Simplex Method," submitted to *Journal of Operations Research* [9].
- **Chapter 4**: (with Komei Fukuda) "The Criss-Cross Method Can Take $\Omega(n^d)$ Pivots," in *Proceedings of the 20th Annual Symposium on Computational Geometry* [33].
- **Chapter 7**: (with David Avis) "Computing Disjoint Paths on Polytopes," submitted to *Journal of Combinatorial Optimization special issue of the Franco-Canadian Workshop on Combinatorial Algorithms, McMaster, August 18-20, 2005* [8].

ABSTRACT

Linear programming is perhaps the most useful tool in optimization, much of it's success owed to the efficiency of the simplex method in practice - its ability to solve problems with millions of variables with relative ease. However, whether there exists a strongly polynomial algorithm to solve linear programming remains an open question. Pivot methods, including the simplex method, remain the best hope for finding such an algorithm, despite the fact that almost all variants have been shown to require exponential time on special instances. Fundamental questions about the path length (number of iterations) of pivot methods remain unanswered. Some, such as the related Hirsch Conjecture, are famous long-standing problems in polyhedral theory. How long can a pivot path be? How many distinct degenerate solutions (bases) can appear during a simplex method cycle? How long can a finite pivot rule stall without improving the solution? Can we enumerate all possible pivot paths to optimality? Can we compute monotone disjoint pivot paths? These are some of the questions we tackle in this thesis in a quest to better understand pivot methods.

ABRÉGÉ

Les méthodes de pivot, y compris la méthode simplexe, offrent le meilleur espoir pour trouver un algorithme fortement polynôme pour résoudre la programmation linéaire, malgré le fait que presque toutes les variantes ont été montrées pour avoir besoin du temps exponentiel pour des exemples spéciaux. Des questions fondamentales au sujet de la longueur de chemin (nombre d'itérations) des méthodes de pivot demeurent sans réponse. Certains, tels que la conjecture de Hirsch, sont des problèmes de longue date célèbres dans la théorie polyèdral. Quel est la longueur maximale d'un chemin de pivot? Combien de solutions dégénérées distinctes (bases) peuvent apparaître dans un cycle pendant la méthode simplexe? Combien de temps une règle de pivot finie peut-elle caler sans améliorer la solution? Pouvons-nous énumérer tous les chemins possibles de pivot à l'optimalité? Pouvons-nous calculer un ensemble de cardinalité maximal de chemins de pivot monotone et distincts? Voilà les questions que nous abordons dans cette thèse: une recherche pour mieux comprendre les méthodes de pivot.

MAIN CONTRIBUTIONS

Theoretical results:

→ We construct linear programs with $m + n$ inequalities in n dimensions on which the simplex method can cycle and visit $\Theta(n^m)$ different bases.

→ We study the behaviour of the simplex method under degeneracy. We show that $D_{Bl}(d, n)$, the maximal number of sequential degenerate bases that the simplex method with Bland's rule can visit on a linear program of dimension d with n inequalities, is $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$, and $\Omega(n^{n-d})$ for $d \leq n \leq \frac{4d}{3}$.

→ We prove that Bland's rule is not equivalent to a perturbation scheme when $d \geq 3$ leaving open the possibility that $D_{Bl}(d, n) = \Theta(n^d)$. However using a novel geometric interpretation of dictionary coefficients we prove that $D_{Bl}(2, n) = n$ and $D_{Bl}(3, n) = O(n^2)$.

→ We construct the longest admissible pivot path possible: defining deformed products of arrangements, we construct a family of linear programs with n inequalities in $\Re^d$ on which the least-index criss-cross method requires $\Theta(n^d)$ iterations to reach optimality.

Software Implementations:

→ We develop an algorithm and implementation to enumerate finite pivot paths over polyhedra, enabling us to compute the objective function and starting basis that yields longest pivot path taken by the simplex method for user input.

→ We present an algorithm and implementation based on the simplex method to compute the maximum cardinality set of vertex-disjoint strictly monotone paths from the source to the sink of a polytopal digraph of a $d-$polytope P directed by a linear objective function.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction & Preliminaries

1.1 Introduction

"Citius, Altius, Fortius" is the motto of the most popular sporting event in the world followed closely by billions of people, demonstrating man's almost unsatiable desire to improve, push limits, set new records - to optimize. This desire has sparked advances in science that have an enormous impact on all aspects of our daily lives. In particular, mathematics and computational sciences have provided us with solutions, if not best solutions, to many of our logistical problems: from scheduling to resource allocation to communications to transportation, etc. However, despite thousands of years of history, mathematics and human decision-making did not formally couple until the first half of the 20th century.

> *Operational research is the use of mathematical models, statistics, and algorithms to aid in decision-making with the goal of improving or optimizing performance.*[1]

Not surprisingly, operations research has its origins in the study of military operations. Its successful application during war-time eventually lead to its routine use today in industry, health care, banking, shipping... the list goes on. Without a

[1] Wikipedia Online Encyclopedia 2005.

doubt, the most useful and celebrated O.R. model is a *Linear Program*, the problem of maximizing a linear objective function subject to a system of linear inequalities, mathematically denoted as

$$\begin{aligned} &\text{maximize } z = \sum_{j=1}^{d} c_j x_j \\ &\text{subject to } \sum_{j=1}^{d} a_{ij} x_j \leq b_i \qquad \text{for } i = 1, ..., n, \end{aligned} \tag{1.1}$$

where the a_{ij}'s, b_i's, and c_j's are given real numbers, and $x_1, ..., x_d$ are variables to be solved for.

The history of linear programming dates back to Fourier [26], who presented an algorithm to solve linear inequalities; Gauss [40], Jordan [52], and de la Vallée Poussin [80], who provided methods to solve systems of linear equations. However it reached its pinacle in the 1930's and 40's with the work of Kantorovich [57] and Dantzig [19]. In 1947, Dantzig - the "father" of linear programming - introduced the *Simplex Method* for solving linear programs. After nearly sixty years of research and use, the simplex method remains the standard solution technique. It is highly efficient in practice; problems with thousands or even millions of variables are routinely solved on computers. Highly sophisticated implementations are available (eg. [12]), often calling the simplex method as a subroutine to help solve much harder *integer linear programs* where the variables are additionally constrained to be integers. Typically, the simplex method solves a linear program in a number of steps that is just a small multiple of the number of variables and inequalities. While its average running-time is theoretically justified ([1], [15], [48], [94], and more recently [88]), there remains no guarantee that the simplex method will compute a solution in reasonable time. In fact, Klee and Minty [61] constructed examples on which Dantzig's simplex method takes a number of iterations exponential in the number of inequalities and variables, prompting new refinements of the method

([14], [15], [19], [29], [45], [51], [54], [70], [74], [92], [99], [101],...), and almost an equal amount of worst-case constructions to accompany them ([2], [5], [45], [61], [74], [78],...). None have been proven to run in time bounded by a polynomial function of n and d.

Whether or not there exists a *strongly-polynomial* time algorithm for linear programming - a method where the maximum number of iterations is a polynomial function of n and d - remains one of the most important unsolved question in operations research theory. Khachian's ellipsoid method [59] and Karmakar's interior point method [58] prove that a linear program can be solved in weakly-polynomial time (in the total amount of data defining the linear program). But as computational scientists we are not satisfied with this result.

Simplex methods offer the possibility of finding a strongly polynomial-time algorithm for linear programming. As an initialization, the (primal) simplex method adds nonnegative *slack variables* to convert the inequalities to equations, then fixes enough of the variables at their bounds to reduce to a square system of linear equations which can be solved for unique values of the remaining variables to obtain a *basic solution.* The variables set to their bounds are called *cobasic* and the remaining are called *basic*, the index set of which is called a *basis.* Phase I of the simplex method either computes a *basic feasible solution*, a basic solution which satisfies the inequalities of (1.1), or proves that the inequalities are unsatisfiable. If the inequality system is feasible then the simplex method proceeds to phase II: computing the *optimal* basic feasible solution that maximizes the objective function. This is accomplished by local improvement: given a basic feasible solution the simplex method purposely selects and exchanges a cobasic variable with a basic variable, re-solves the system of equations for the new basis to obtain a new basic feasible solution whose objective value is greater than or equal to the previous solution. This operation is called a *(primal admissible) pivot*, and *pivot rules*

(refinements) determine which variables to exchange. Simplex methods may stall, pivot from basis to basis without improving the objective value. This phenomenon is known as *degeneracy*, a common occurence in practice, and pivot rules require an anti-cycling scheme to guarantee finiteness. The length of a pivot path is the number of distinct bases visited, starting at the basis representing the initial basic feasible solution.

Pivot paths have close ties to *polyhedral* theory as the inequalities of a linear program (1.1) represent a d-polyhedron P with up to n facets, and each basic feasible solution corresponds to a vertex of P. A polytope is a bounded polyhedron. The vertices and edges of a $d-$polytope P form an undirected graph $G(P)$. The linear objective function cx, when in *general position* with respect to P, gives an orientation on the edges of $G(P)$ allowing us to define an acyclic digraph $D_c(P)$. This digraph has a unique *source*, or vertex minimizing cx, and *sink*, or vertex maximizing cx. Linear programming is the problem of finding a vertex of P whose location in space maximizes the objective function, and the simplex method follows a path (*monotone* with respect to the objective function) in $D_c(P)$.

The *diameter* of $G(P)$ is the smallest number δ such that every two vertices in $G(P)$ are connected by a path with at most δ edges. The earliest, and most outstanding conjecture on the diameter of $G(P)$ is the following.

Conjecture 1 (Hirsch Conjecture, [102] p. 84). *Let P be a $d-$polytope with n facets, then the maximal diameter $\Delta(d,n)$ of $G(P)$ is at most $n-d$.*

The best known upperbound on the maximal diameter is subexponential, $\Delta(d,n) \leq n^{\log d+1}$, proved independently by Kalai [54] and Matoušek et al. [70]. For further discussion and results see [62] and [75]. Even if true, the Hirsch Conjecture would not imply a bound on the length of such a monotone path in $D_c(P)$. In fact, Todd [92] gave an example of $4-$polytope with 8 facets for which every monotone

path from a given vertex to the sink had at least 5 edges. A conjecture more directly related to paths taken by the simplex method is the following.

Conjecture 2 (Ziegler's Strict Monotone Hirsch Conjecture, [102] p.86). *Let P be a $d-$polytope with n facets and cx a linear function in general position, then there exists a path in $D_c(P)$ from source to sink of length at most $n-d$.*

The Hirsch-type conjectures and the closely related polynomial-time simplex algorithm mystery have motivated, but also baffled, researchers for the last half a century. We have and will continue to choose the simplex method to solve linear programs, despite the fact that many fundamental questions about the path length taken by pivot methods remain unanswered. This thesis tackles some of these questions: we study pivot paths in linear programs, particularly focusing on the worst-case performance of the simplex method under degeneracy, cycling; and the criss-cross method. We provide new algorithms and implementations for pivot path enumeration, and for computing disjoint paths on polytopes. The new functionalities of these packages add to a growing amount of software that exists (such as *polymake* [53]) to assist researchers in polyhedral theory.

Our main contributions, and their significance, are the following:

- In Chapter 2 we study cycling in the simplex method. We define products of arrangements, an extension of products of polytopes, and use them to construct linear programs with m inequalities and n nonnegative variables on which the simplex method cycles, achieving new bounds on cycle lengths. We show that the maximal number of distinct bases in a simplex cycle is $\Theta(n^m)$ for $n \geq 3m$ with $m \geq 2$ fixed and even; we can force the simplex method to cycle and visit nearly all $\binom{n+m}{m}$ bases of a linear program. The previous best known bound on the maximal length of a simplex cycle was $\Omega(n)$ for $m \geq 2, n \geq 6$ [67].

- In Chapter 3 we study finite pivot rules for the simplex method and degenerate stalling. Degeneracy is a common occurrence in practice, but little was known about how much time the simplex method can spend stalling. We show that Bland's anti-cycling pivot rule is not a perturbation scheme leading to the possibility that the number of degenerate bases visited by the simplex method with Bland's rule, $D_{Bl}(d,n)$, can exceed the maximal number of vertices that a $d-$polytope with n facets can have. However we prove that $D_{Bl}(2,n) = n$ and $D_{Bl}(3,n) = O(n^2)$. Our proofs employ a new geometric interpretation of (simplex method) dictionary coefficients (presented in Section 1.2.5).
- In Chapter 4 we construct the longest linear programming (admissible) pivot path possible. The lack of success with simplex methods, with respect to finding a strongly polynomial algorithm that solves linear programming, lead researchers to study *criss-cross methods* [103], [90], [95] which leave the boundary of the $d-$polytope and traverse the edges of the underlying oriented hyperplane arrangement (n inequalities) using *admissible pivots.* We prove that the least-index criss-cross method can visit nearly every vertex of a hyperplane arrangement and take $\Theta(n^d)$ pivots for $n \geq 2d$. In doing so, we extend the notion of deformed products of polytopes to oriented hyperplane arrangements by defining *deformed products of arrangements.* The result affirms observations made in practice, proving that the criss-cross method is worse than most refinements of the simplex method.
- In Chapter 5 we construct families of examples proving that $D_{Bl}(d,n) = \Omega(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$ (matching Amenta and Ziegler's [2] bound on nondegenerate input), and $D_{Bl}(d,n) = \Theta(n^{n-d})$ for $d \leq n \leq \frac{4d}{3}$. Most implementations of the simplex method suggest that when $n \geq 2d$ the dual simplex method should be used, thus the latter construction is of particular significance as it is

the first general worst-case construction for the primal simplex method when $n < 2d$.

- In Chapter 6 we present pivot path enumeration software: listing all possible pivot paths that the simplex method may follow to a given optimal basis. We compute the finite number of objective functions needed to enumerate all possible pivot paths taken by the simplex method following a combinatorial pivot rule. We present a pivot path enumeration algorithm and nontrivial implementation that allows us, for instance, to compute the objective function and starting basis yielding the longest pivot path on user inputed polyhedra.
- In Chapter 7 we present a new algorithm and implementation for computing vertex-disjoint strict monotone paths in polytopes. Holt and Klee [50] proved that there exist d vertex-disjoint strict monotone paths from source to sink of $D_c(P)$ for a $d-$polytope with n facets (with a linear function in general position with respect to P). Studying these paths has the potential to provide new insight into designing a polynomial-time simplex method, or proving none exists. We provide the tool to compute them. Experimental results show that our algorithm is particularly advantageous when only a few disjoint paths are required, but also excels when the input has little or no degeneracy, and is especially memory-efficient when the polytope has many vertices.

Our research has lead to new open problems, which we list at the end of each chapter.

We begin with a gentle introduction to solving inequalities, linear programming via the simplex method, and some basic polyhedral theory.

1.2 Preliminaries

Our only assumption is that the reader has learned how to solve a system of linear equations and is familiar with basic geometric concepts. We follow the

notation of Chvátal [18] and Ziegler [102], but also refer the reader to Grünbaum [47] and Matoušek [71] for background material.

1.2.1 Solving Linear Inequalities

Consider the following problem: given a matrix $A = [a_{ij}]$ in $R^{m \times n}$ and a column vector b in R^m, find $x = (x_1, x_2, ..., x_n)^T$ that satisfies the following linear system, or prove that no such vector x exists:

$$\begin{aligned} Ax &\leq b, \\ x &\geq 0. \end{aligned} \tag{1.2}$$

We illustrate a simple method for doing this with an example:

$$\begin{aligned} -x_1 - 2x_2 + x_3 &\leq -1 \\ x_1 - 3x_2 - x_3 &\leq 2 \\ -x_1 - 2x_2 + 2x_3 &\leq -2 \\ x_i &\geq 0 \qquad (i = 1, 2, 3). \end{aligned} \tag{1.3}$$

We first convert this system of inequalities into a system of equations by introducing a new nonnegative *slack* variable for each inequality. This slack variable represents the difference between the right- and left-hand sides of the inequality. In our example, we need three new variables, which we label x_4, x_5, and x_6. Putting these variables on the left-hand side, and the others on the right-hand side we have the following system:

$$\begin{aligned} x_4 &= -1 + x_1 + 2x_2 - x_3 \\ x_5 &= 2 - x_1 + 3x_2 + x_3 \\ x_6 &= -2 + x_1 + 2x_2 - 2x_3. \end{aligned} \tag{1.4}$$

It is easy to see that any nonnegative solution of (1.3) then extends to a nonnegative solution of (1.4) by assigning the slack variables values via their respective

equations. Conversely, a nonnegative solution of (1.4) when restricted to x_1, x_2, and x_3 gives a solution to (1.3). We call a system of equations such as (1.4) a *dictionary.* The variables on the left-hand side are called *basic*, and the variables on the right-hand side are called *cobasic.* We get a *basic solution* to the equations in (1.4) by setting all the cobasic variables to zero, which gives $x_4 = -1, x_5 = 2, x_6 = -2$. Unfortunately this is not a nonnegative solution. The algorithm proceeds as follows: it finds the smallest-indexed basic variable that is set to a negative value. In this case it is x_4. In the equation for x_4 it identifies the cobasic variable with the smallest index that has a positive coefficient (in this case it is x_1), solves this equation for x_1, and substitutes the result for x_1 in the other equations. This yields a new dictionary:

$$\begin{aligned} x_1 &= 1 - 2x_2 + x_3 + x_4 \\ x_5 &= 1 + 5x_2 - x_4 \\ x_6 &= -1 - x_3 + x_4. \end{aligned} \tag{1.5}$$

The step we just performed is called a *pivot* operation, and it is the basic step of the algorithm. In fact it is the only step: we simply repeat this operation. In (1.5), we first set the cobasic (i.e., right-hand) variables to zero and get the basic solution $x_1 = 1, x_5 = 1, x_6 = -1$. Again, we find the basic variable with the smallest index and negative value, namely, x_6. In the equation for x_6 we find the smallest-indexed cobasic variable with a positive coefficient, here x_4. We pivot by solving this equation for x_4 and substituting for x_4 in the other equations, obtaining the new dictionary:

$$\begin{aligned} x_1 &= 2 - 2x_2 + 2x_3 + x_6 \\ x_4 &= 1 + x_3 + x_6 \\ x_5 &= 0 + 5x_2 - x_3 - x_6. \end{aligned} \tag{1.6}$$

We are now in luck. The basic solution is nonnegative, aptly named a *basic feasible solution* as its restriction to our original three variables gives a feasible solution to (1.3): $x_1 = 2, x_2 = 0, x_3 = 0$. So far so good. An immediate question raises itself: What happens if there is no solution to the original problem? Consider the following problem:

$$\begin{aligned} -x_1 + 2x_2 + x_3 &\leq 3 \\ 3x_1 - 2x_2 + x_3 &\leq -17 \\ -x_1 - 6x_2 - 23x_3 &\leq 19. \end{aligned} \tag{1.7}$$

We get an initial dictionary by introducing three slack variables and letting them be the basic variables:

$$\begin{aligned} x_4 &= 3 + x_1 - 2x_2 - x_3 \\ x_5 &= -17 - 3x_1 + 2x_2 - x_3 \\ x_6 &= 19 + x_1 + 6x_2 + 23x_3. \end{aligned} \tag{1.8}$$

The algorithm proceeds as before by choosing the equation for x_5 and solving for x_2:

$$\begin{aligned} x_2 &= 17/2 + (3/2)x_1 + (1/2)x_3 + (1/2)x_5 \\ x_4 &= -14 - 2x_1 - 2x_3 - x_5 \\ x_6 &= 70 + 10x_1 + 26x_3 + 3x_5. \end{aligned} \tag{1.9}$$

Here we encounter something new. We select the equation for x_4, as we should, but find that there is no cobasic variable with a positive coefficient. We rewrite this equation with all variables on the left-hand side, including those with zero coefficients, getting

$$2x_1 + 0x_2 + 2x_3 + \mathbf{1}x_4 + \mathbf{1}x_5 + \mathbf{0}x_6 = -14. \tag{1.10}$$

This is an example of an *inconsistent equation.* Note that the coefficients of all variables are nonnegative, but the right-hand side is negative. Therefore this equation cannot be satisfied by choosing any combination of nonnegative values for the variables. This equation was derived from the original system by standard operations that do not change the solution set for the equations. Therefore (1.8), hence (1.7), has no nonnegative solution. In fact, (1.10) provides a simple proof of this encoded in the boldface coefficients of the slack variables. We multiply each inequality in (1.7) by the coefficient of its corresponding slack variable

$$\begin{aligned} \mathbf{1} * (-x_1 + 2x_2 + x_3 \leq 3) \\ +\mathbf{1} * (3x_1 - 2x_2 + x_3 \leq -17) \\ +\mathbf{0} * (-x_1 - 6x_2 - 23x_3 \leq 19) \end{aligned} \tag{1.11}$$

and add the inequalities in (1.11) to get

$$\mathbf{2x_1 + 2x_3 \leq -14.} \tag{1.12}$$

The final inequality, (1.12) is called an *inconsistent inequality*: all the variables have nonnegative coefficients, yet the right-hand side is negative. The multipliers given by the coefficients of the slack variables are said to furnish a *certificate of infeasibility* for the original system.

We now have a complete description of the algorithm that we christen the "$b-$rule"[2] for solving problems of form (1.2):

[2] The $b-$rule is a dual form of Bland's least-index rule for linear programming [13].

Step 1: Introduce m slack variables $x_{n+1}, ..., x_{n+m}$ and use these as the basis (left-hand side) of an initial dictionary:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad (i = 1, ..., m). \tag{1.13}$$

Step 2: Set the cobasic (right-hand) variables to zero. Find the smallest index of a basic (left-hand side) variables with a negative value. If there is none, terminate with a feasible solution.

Step 3: Find the cobasic variable in the equation chosen in step 2 that has the smallest index and a positive coefficient. If there is none, terminate, for the problem is infeasible, and the coefficients of the slack variables represent a certificate of infeasibility. Otherwise, solve this equation for the indicated cobasic variable, and substitute the result in all of the other equations. Go to step 2.

In what follows we prove:

- the algorithm that we have described halts after a finite number of steps;
- if it halts in step 2, then the basic solution is feasible for (1.2);
- if it halts in step 3, then the system (1.2) is infeasible and the slack coefficients "certify" this.

Theorem 1.1. *The b−rule is finite.*

Our proof is based on ideas found in Fukuda and Terlaky [36].

Proof. Given an input system (1.2), we construct the initial dictionary (1.13) and run the b−rule algorithm. Since there are at most $\binom{n+m}{m}$ possible choices of a basis, if the algorithm is not finite (in the sense of halting after finitely many steps), then some bases must be repeated, a process called *cycling*. Assume that this can happen, and choose a system of equations that cycles.

Suppose first that x_{n+m} ($n + m$ being the largest index) enters and leaves the basis during the cycle. When x_{n+m} is chosen to enter the basis we must have an

equation of the following form, where B and N denote the set of basic and cobasic indices, respectively:

$$x_k = -b'_k - \sum_{j \in N \setminus \{n+m\}} a'_{kj} x_j + a'_{k,n+m} x_{n+m} \qquad (k \in B). \tag{1.14}$$

The choice of x_{n+m} as entering variable in this equation implies that $-b'_k < 0$, $a'_{k,n+m} > 0$, and $a'_{kj} \geq 0$ for j in $N \setminus \{n+m\}$. This shows that every solution to the full system of equations with $x_1, ..., x_{n+m-1} \geq 0$ must have $x_{n+m} > 0$, i.e. solving (1.14) for x_{n+m}.

Now consider the stage at which x_{n+m} is chosen to leave the basis. The dictionary has the form:

$$\begin{aligned} x_i &= b'_i + \sum_{j \in N} a'_{ij} x_j \qquad (i \in B \setminus \{n+m\}) \\ x_{n+m} &= -b'_{n+m} + \sum_{j \in N} a'_{n+m,j} x_j. \end{aligned} \tag{1.15}$$

The choice of x_{n+m} ensures that $-b'_{n+m} < 0$ and $b'_i \geq 0$ for i in $B \setminus \{n+m\}$. By setting the cobasic variables to zero, dictionary (1.15) shows that there exists a solution to the system of equations with $x_1, ..., x_{n+m-1} \geq 0$ and $x_{n+m} < 0$. Clearly not both (1.14) and (1.15) can hold, so there cannot exist a cycle during which the largest-indexed variable enters and leaves the basis.

Now suppose that there exists a cycle in which x_{n+m} always stays in the basis. Then we can remove x_{n+m} and its corresponding equation without changing the pivot decisions made during the cycle. Similarly, if there exists a cycle where x_{n+m} always stays in the cobasis, then we can remove x_{n+m} from all of the equations without influencing the cycle. Either way we can reduce the original example that cycles to an equivalent example with a cycle during which the largest-indexed variable both enters and leaves the basis. This leads to the two conflicting situations that we met earlier, so a cycle cannot exist: the algorithm is finite. □

Since the algorithm is finite, it must halt in either step 2 or step 3. If it terminates in step 2, we have a nonnegative solution to the original system. This follows from the fact that the only operations we performed on the initial dictionary were standard operations for manipulating a system of equations. If the algorithm stops in step 3, we have a certificate of infeasibility that, when stated in general terms, is a variant of the Farkas lemma [22].

Theorem 1.2 (Farkas Lemma). *Either there exists x in R^n with $x \geq 0$ such that $Ax \leq b$ or there exists y in R^m with $y \geq 0$ such that $y^T A \geq 0$ and $y^T b < 0$.*

Proof. We begin by showing that there cannot exist both a vector x and a vector y satisfying the conditions of the theorem. For otherwise, $0 > y^T b \geq y^T Ax \geq 0$. If such a vector x does not exist, the finiteness of the b—rule implies that the algorithm must halt in step 3. The algorithm returns an inconsistent equation:

$$\sum_{\substack{j=1,\\ j\neq k}}^{n+m} a'_{kj} x_j + x_k = -b'_k, \tag{1.16}$$

where $b'_k > 0$ and all of the coefficients $a'_{kj} \geq 0$. Set $y_i = a'_{k,n+i} \geq 0$ for $i = 1, ..., m$. We observe that equation (1.16) is obtained from the initial dictionary (1.13) by multiplying the equation for x_{n+i} by y_i and summing. This is because variable x_{n+i} appears only once in the initial dictionary, as the left-hand side of its defining equation. (1.16) is a nonnegative combination of the original rows. This shows that $y^T b = -b'_k < 0$ and that

$$\sum_{i=1}^{m} y_i a_{ij} = a'_{kj} \geq 0 \qquad (j = 1, ..., n), \tag{1.17}$$

again by the finiteness of the algorithm. Hence $y^T A \geq 0$. □

For constructive proofs of other classical theorems of linear algebra using pivoting see [60].

1.2.2 Linear Programming and the Simplex Method

Linear programming is the problem of maximizing a linear objective function subject to a system of linear inequalities, written in standard form as

$$\begin{aligned} \max z &= \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j &\leq b_i \qquad i = 1, ..., m \\ x_j &\geq 0 \qquad j = 1, ..., n. \end{aligned} \tag{1.18}$$

There are three possible outcomes: *optimality*, *infeasibility*, or *unboundedness*. The *Simplex Method* [19] is a two phase method for solving linear programs. *Phase I* either computes a basic feasible solution to the system of inequalities or provides a certificate of infeasibility. We will not elaborate on phase I as for the remainder of this thesis we will assume that the linear program is feasible and that an initial basic feasible solution is given. Of course we can add nonnegative *slack* variables $x_{n+1}, ..., x_{n+m}$ to the inequalities of (1.18) to obtain a system of linear equations,

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} &= b_i \qquad i = 1, ..., m, \\ z &= \sum_{j=1}^{n} c_j x_j, \end{aligned} \tag{1.19}$$

solve these $m + 1$ equations for $m - n$ variables and z to obtain a dictionary,

$$\begin{aligned} x_i &= b'_i - \sum_{j \in N} a'_{ij} x_j \qquad \text{for } i \in B \\ z &= z' + \sum_{j \in N} c'_j x_j, \end{aligned} \tag{1.20}$$

where B is the set of indices of the basic variables, hereforth referred to as the "basis" (and N the index set of the remaining indices - termed "cobasis"), and then use the $b-$rule presented in the previous section to compute an initial basic feasible

solution for the linear program (instead of phase I), with the value of z being the objective value. For example, the linear program

$$\begin{aligned} \max\ z &= 10x_1 - 57x_2 - 9x_3 - 24x_4 \\ 0.5x_1 - 5.5x_2 - 2.5x_3 + 9x_4 &\leq 0 \\ 0.5x_1 - 1.5x_2 - 0.5x_3 + x_4 &\leq 0 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned} \tag{1.21}$$

is encoded by the dictionary

$$\begin{aligned} x_5 &= 0 - 0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 \\ x_6 &= 0 - 0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 \\ z &= 0 + 10x_1 - 57x_2 - 9x_3 - 24x_4. \end{aligned} \tag{1.22}$$

Phase II of the simplex method attempts to increase the objective value by exchanging a cobasic variable x_s with positive coefficient in the $z-$row with a basic variable x_r while maintaining a basic feasible solution. If all the coefficients of the cobasic variables are nonpositive, then the solution is optimal. Otherwise an exchange is initiated and feasibility is maintained via a *ratio test*:

$$r = ratio(s) = argmin\{b'_i/a'_{is} : i \in B, a'_{is} > 0\},$$

where *argmin* returns the index i minimizing the given ratio. (If $a'_{is} \leq 0$ for all $i \in B$ then the linear program is unbounded.) The new basic feasible solution has objective value $c'_s \frac{b'_r}{a'_{rs}}$ more than previous solution. This exchange of variables is called an *primal admissible pivot*, also known as a *simplex pivot*. In fact it is the only operation of the simplex method; we repeat it until an optimal solution is found or until we find a system where the coefficient of x_s is nonnegative in all equations, yielding a direction of unboundedness. The sequence of bases visited from the first feasible basis to the terminal basis is called a *pivot path*. For the

remainder of this thesis when "simplex method" will refer to only phase II, and when we say "simplex pivot" we talk about a "primal admissible pivot".

Pivot rules (or refinements) dictate which variables to select. Combinatorial pivot rules select an entering variable based solely on the sign of the coefficient. Coefficient-based pivot rules select a variable depending on the magnitude of its coefficient. There are many documented pivot rules, for example [13], [14], [15], [19], [29], [45], [51], [54], [70], [74], [93], [99], and [101] (for a survey on pivot rules see Terlaky and Zhang [91]). Most of the pivot rules proposed can be made to follow, on carefully constructed input, pivot paths whose length is an exponential function of n and m (see [2], [5], [45], [61], [74], and [78]), and whether there exists a pivot rule guaranteed to follow a path whose length is polynomial in n and m remains the most important unsolved problem in the theory of linear programming. Despite this mystery, the simplex method almost always terminates very quickly in practice, its average-case behaviour theoretically justified in [1], [15], [48], [94], and more recently [88].

The first pivot rule proposed, *Dantzig's pivot rule* [19], is widely used. It chooses the variable in the $z-$row with largest positive coefficient, and breaks ties in the ratio test arbitrarily (usually by selecting the variable with the smallest index). In our example we would pivot on variables x_1 and x_5, the former chosen as it has the largest positive coefficient in the $z-$row and the latter arbitrarily chosen from x_5 and x_6 due to a ratio test tie - caused by *degeneracy* - (we chose x_5) yielding the dictionary:

$$\begin{aligned} x_1 &= 11x_2 + 5x_3 - 18x_4 - 2x_5 \\ x_6 &= -4x_2 - 2x_3 + 8x_4 + x_5 \\ z &= 53x_2 + 41x_3 - 204x_4 - 20x_5. \end{aligned} \tag{1.23}$$

Degeneracy may cause the simplex method to *stall*, pivot to a new dictionary without increasing the objective value. The simplex method *cycles* if it stalls indefinitely, revisiting previously computed dictionaries. We will return to this example, and cycling in general, in Chapter 2. A pivot rule is *finite* if it avoids cycling, guaranteeing termination. As we will see in Chapter 3, there are several techniques that can be employed to make a pivot rule finite.

Every maximization linear programming problem is associated to a *dual* minimization linear program. The *primal LP* of (1.18) gives rise to the dual *LP*:

$$\begin{aligned} \min w &= \sum_{i=1}^{m} b_i y_i \\ \sum_{i=1}^{m} a_{ij} y_i &\geq c_j \qquad j = 1, ..., n \\ y_i &\geq 0 \qquad i = 1, ..., m. \end{aligned} \tag{1.24}$$

Theorem 1.3 (The Duality Theorem). *If the primal (1.18) has an optimal solution* $(x_1^*, x_2^*, ..., x_n^*,)$ *then the dual (1.24) has an optimal solution* $(y_1^*, y_2^*, ..., y_m^*)$ *such that*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

We refer to Chvátal's linear programming book [18] for the proof of the duality theorem, and for a further (in-depth) introduction to the theory of linear programming.

1.2.3 Polyhedra

We refer to Ziegler [102] for a detailed introduction into polytope theory, but provide the essential background for this thesis in this section.

Points, lines, planes, and hyperplanes in $d-$dimensions are *affine subspaces* of dimension $0, 1, 2$ and $d-1$. A point set $K \subseteq \Re^d$ is *convex* if with any two points $k_1, k_2 \in K$ it also contains the straight line segment $[k_1, k_2] = \{\lambda k_1 + (1-\lambda)k_2 :$

$0 \leq \lambda \leq 1\}$ between them. The intersection of convex sets is convex, and for any $K \in \Re^d$, the *convex hull* of K, conv$\{K\}$, is defined as the "smallest" convex set containing K; the intersection of all convex sets that contain K.

An $H-$*polyhedron* P is the intersection of n closed halfspaces in $\Re^d$: $P = \{x \in \Re^d : a_i^T x \leq \alpha_i \quad \text{for } 1 \leq i \leq n\}$ where $a_i \in \Re^d$, and $\alpha_i \in \Re$. A $V-$*polyhedron* is the *Minkowski sum* of the convex hull of a finite set of points in $\Re^d$ with a finite number of rays in $\Re^d$: $P = \text{conv}\{p_1, ..., p_m\} + \text{cone}\{r_1, ..., r_{\overline{m}}\}$ where each ray r_i for $i = 1, ..., \overline{m}$ is defined as $r_i = \{\overline{x}_i + t\overline{y}_i : t \geq 0\}$ for any $\overline{y}_i \neq 0$ $(\overline{x}_i, \overline{y}_i \in \Re^d)$. Throughout this thesis we assume full dimensionality of polyhedra. A bounded polyhedron (that contains no rays) is called a *polytope.* The *faces* of a polyhedron P are all the subsets of the form $F = \{x \in P : a^T x = \alpha\}$ for some $a \in \Re^d$, and $\alpha \in \Re$, where $a^T x \leq \alpha$ is a *valid* inequality for P; meaning that $a^T x \leq \alpha$ is satisfied for all $x \in P$. The faces of P are themselves polyhedra and the faces of dimensions $0, 1, ..., d-1$, and k are called *vertices*, *edges*, ..., *facets,* and $k-$*faces* of P.

The inequalities of a linear program define an $H-$polyhedron P. To relate results from polyhedral theory more easily, we will stray from the standard form of (1.18) and instead use the following form (with Chapter 2 being the only exception):

$$\begin{aligned} \text{maximize } z = \varphi(x) &= \sum_{j=1}^{d} c_j x_j \\ \text{subject to } \sum_{j=1}^{d} a_{ij} x_j &\leq b_i \qquad \text{for } i = 1, ..., n, \end{aligned} \tag{1.25}$$

and dictionary

$$\begin{aligned} x_i &= b_i' - \sum_{j \in N} a_{ij}' x_j \qquad \text{for } i \in B \\ z &= z' + \sum_{j \in N} c_j' x_j \end{aligned} \tag{1.26}$$

with $\{1, ..., d\} \in B$. The cobasic variables of a basic feasible solution correspond to inequalities of P set to equality. The intersection of a set of these inequalities defines a face of P, in particular the d inequalities corresponding to d cobasic variables of a basic feasible solution intersect to form a vertex of P. A $d-$polytope is *simple* if every vertex lies on exactly d facets, and *degenerate* otherwise. A linear function cx in R^d is in *general position* with respect to P if it attains different values at each vertex of P. Thus the geometric interpretation of the simplex method is that it tries to solve a linear program by pivoting along the boundary edges of P from vertex $p_i \in P$ to vertex $p_j \in P$ such that $\varphi(p_i) < \varphi(p_j)$, but possibly stalling on degenerate vertices.

The main theorem for polyhedra states that each $H-$polyhedron is a $V-$polyhedron and vice versa. For simplicity we give the proof for the polytope version:

Theorem 1.4 (Main Theorem for Polytopes). *Each $H-$polytope is a $V-$polytope, and each $V-$polytope is an $H-$polytope.*

Proof (Minkowski-Weyl-Farkas). Let $P = \{x \in \Re^d : a_i^t x \leq \alpha_i \quad \text{for } 1 \leq i \leq n\}$, and let X be the set of vertices of P. We must prove that $P = \text{conv}(X)$:

- $\text{conv}(X) \subset P$ as each element of X, a vertex of P, must satisfy the inequalities of P.
- To prove $P \subset \text{conv}(X)$ we proceed by contradiction. Let $X = \{x_1, ..., x_m\}$ and assume that P is not entirely contained in $\text{conv}(X)$, that there is a point $p \in P$ that is not a convex combination of $x_1, ..., x_m$: no $\lambda_1, \lambda_2, ..., \lambda_m \geq 0$ exist such that

$$(u) : \sum_{i=1}^{m} \lambda_i x_i = p$$
$$(v) : \sum_{i=1}^{m} \lambda_i = 1.$$

Then by Farkas Lemma (variant of Theorem 1.2) there exist multipliers u and v such that $u \cdot p + v < 0$ and $u \cdot x_i + v \geq 0$ for $i = 1, ..., m$. So the following linear program has negative optimum objective value.

$$\begin{aligned} \min z = & u \cdot x + v \\ & a_i^T x \leq \alpha_i \quad \text{for } 1 \leq i \leq n. \end{aligned}$$

The simplex method therefore returns a vertex $q = x_j$ of P with $uq + v = ux_j + v < 0$, a contradiction. This implies that $p \in \text{conv}(X)$. This proves that every $H-$polytope is a $V-$polytope. The proof that every $V-$polytope is a $H-$polytope follows by polytope polarity (see [102]).

□

Theorem 1.5 (Upper Bound Theorem, McMullen [72]). *A $d-$dimensional polytope with n facets has no more than*

$$M(d,n) = \binom{n - \lceil \frac{d}{2} \rceil}{\lfloor \frac{d}{2} \rfloor} + \binom{n - 1 - \lceil \frac{d-1}{2} \rceil}{\lfloor \frac{d-1}{2} \rfloor} \tag{1.27}$$

vertices, where equality is attained only by the polars of neighborly polytopes (for example, by the polars of cyclic polytopes - see [102, p. 15]).

This upper bound is a polynomial in n of degree $\lfloor \frac{d}{2} \rfloor$ in the case of fixed dimension:

$$M(d,n) = \Theta(n^{\lfloor \frac{d}{2} \rfloor}) \quad \text{for fixed } d. \tag{1.28}$$

Definition 1.1 (Product of Polytopes). *The product of two polytopes $P \subseteq \Re^d$ and $Q \subseteq \Re^e$ is given by*

$$P \times Q = \left\{ \begin{pmatrix} u \\ v \end{pmatrix} : \begin{matrix} u \in P \\ v \in Q \end{matrix} \right\}. \tag{1.29}$$

The vertices of the product are given by

$$\text{vert}(P \times Q) = \left\{ \begin{pmatrix} p_i \\ q_j \end{pmatrix} : \begin{matrix} p_i \in \text{vert}(P) \\ q_j \in \text{vert}(Q) \end{matrix} \right\}, \tag{1.30}$$

and the facet-defining inequalities for $P \times Q$ are the inequalities of P together with the inequalities of Q. Thus taking the product of two polytopes multiplies the number of vertices and sums the number of facets.

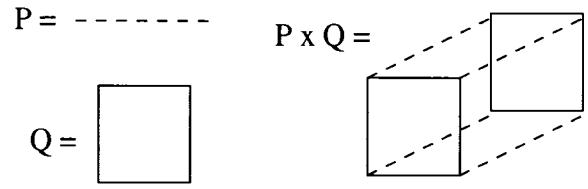


Figure 1–1: Product of polytopes

Definition 1.2 (Combinatorially Equivalent Polytopes). *Two polytopes P and Q are combinatorially equivalent if there is a bijection between their vertices, $vert(P) = \{p_1, ..., p_m\}$ and $vert(Q) = \{q_1, ..., q_m\}$, such that for any subset $I \subseteq \{1, ..., m\}$, the convex hull $conv\{p_i : i \in I\}$ is a face of P if and only if $conv\{q_i : i \in I\}$ is a face of Q.*

Definition 1.3 (Normally Equivalent Polytopes). *Two polytopes P and Q are normally equivalent if they are combinatorially equivalent and each facet $conv\{p_i : i \in I\}$ of P is parallel to the corresponding facet $conv\{q_i : i \in I\}$ of Q.*

Definition 1.4 (Deformed Products of Polytopes [2]). *Let $P \subseteq \Re^d$ be a convex polytope, and $\varphi : P \to \Re$ a linear functional with $\varphi(P) \subseteq [0, 1]$. Let V, $W \subseteq \Re^e$ be convex polytopes. Then the deformed product of (P, φ) and of (V, W) is*

$$(P, \varphi) \bowtie (V, W) := \left\{ \begin{pmatrix} x \\ v + \varphi(x)(w - v) \end{pmatrix} : \begin{matrix} x \in P \\ v \in V, w \in W \end{matrix} \right\} \subseteq \Re^{d+e}.$$

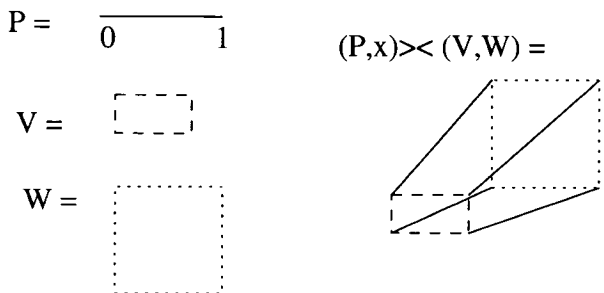


Figure 1–2: Deformed product of polytopes

Definition 1.5 (Deformed Product Programs). *For* $\alpha : \Re^e \to \Re$, *define the deformed product program as*

$$\max \widehat{\alpha}\begin{pmatrix} x \\ u \end{pmatrix} = \alpha(u) : \quad \begin{pmatrix} x \\ u \end{pmatrix} \in Q = (P, \varphi) \bowtie (V, W), \tag{1.31}$$

where $\widehat{\alpha} : \Re^{d+e} \to \Re$ *is a linear function. The resulting linear program is the deformed product polytope* Q *with objective function* $\max \alpha(u)$.

1.2.4 Hyperplane Arrangements

A *hyperplane* is a set $h = \{u \in \Re^d : a^T u = \alpha\}$ for some nonzero $a \in \Re^d$, and $\alpha \in \Re$. A finite set of hyperplanes H in $\Re^d$ induces a decomposition of $\Re^d$ into connected cells called an *arrangement* A_H. The $0, 1, 2, (d-1)$, and $k-$dimensional cells of A_H are termed vertices, edges, faces, facets, and $k-$cells. Two vertices of A_H are *adjacent* if they share $d-1$ hyperplanes, in other words they share an edge. An arrangement is *central* if $\alpha = 0$ for each hyperplane (passes through the origin). We will assume that the hyperplanes are labeled as $H = \{h_1, h_2, \ldots, h_n\}$, and that hyperplanes are *oriented*: each has a positive side and negative side that are given by $\{u \in \Re^d : a^T u > \alpha\}$ and $\{u \in \Re^d : a^T u < \alpha\}$. An arrangement of oriented hyperplanes is an example of an *oriented matroid*, and shares all of its features. In particular, each cell of the arrangement is represented as a *signed vector* in $\{+, 0, -\}^n$ indicating the position of the cell with respect to the hyperplanes of H. For a study on the combinatorial structure of arrangements, see [100] and [81].

Proposition 1.1. *A polytope* $P \subseteq \Re^d$ *induces an arrangement of oriented hyperplanes* A_P.

1.2.5 Geometric Interpretation of Dictionary Coefficients

Given a linear program,

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{d} c_j x_j \\ &\text{subject to} \sum_{j=1}^{d} a_{ij} x_j \leq b_i \text{ for } i = 1, ..., n, \end{aligned} \tag{1.32}$$

add slack variables $x_{d+1}, ..., x_{d+n}$ and compute an (extended) dictionary with basis B and cobasis N:

$$\begin{aligned} x_1 &= \overline{b}_1 - \sum_{j \in N} \overline{a}_{1j} x_j \\ x_2 &= \overline{b}_2 - \sum_{j \in N} \overline{a}_{2j} x_j \\ &\vdots \\ x_d &= \overline{b}_d - \sum_{j \in N} \overline{a}_{dj} x_j \\ &- - - - - - - - \\ x_i &= b'_i - \sum_{j \in N} a'_{ij} x_j \text{ for all } i \in B \setminus \{1, ..., d\} \\ z &= z' + \sum_{i \in N} c'_j x_j. \end{aligned} \tag{1.33}$$

(The original variable indices $1, ..., d$ always remain in the basis in an extended dictionary)

Definition 1.6 (Cobasic Gradient Matrix). *Let G_N be the matrix whose rows consist of the (ordered set of) gradient vectors of the inequalities of (1.32) whose slack variables are cobasic in dictionary (1.33),*

$$G_N = \left[\begin{array}{cccc} a_{j1} & ... & a_{jd} & : \ \{j+d\} \in N \end{array} \right].$$

Theorem 1.6.

$$-a'_{rs} = \frac{\left| \begin{smallmatrix} a_{r-d} \\ G_{N \setminus s} \end{smallmatrix} \right|}{\left| \begin{smallmatrix} a_{s-d} \\ G_{N \setminus s} \end{smallmatrix} \right|}, \text{ and } c'_s = -\frac{\left| \begin{smallmatrix} c \\ G_{N \setminus s} \end{smallmatrix} \right|}{\left| \begin{smallmatrix} a_{s-d} \\ G_{N \setminus s} \end{smallmatrix} \right|},$$

where $a_k = (a_{k1}, a_{k2}, ..., a_{kd}), c = (c_1, c_2, ..., c_d), r \in B \setminus \{1, ..., d\}$, *and* $s \in N$.

Interpreting the coefficients of a dictionary as a ratio of determinants is useful because the determinant of a matrix has a geometric meaning. The determinant is the signed hypervolume of the hyperparallelepiped generated by the vectors of the matrix. The sign of a determinant is the orientation of the arrangement of the vectors. Swapping two rows changes the order of sweeping out the volume, and will hence turn a positive volume to negative or vice-versa. For example, consider the matrix $\begin{bmatrix} u \\ v \end{bmatrix}$ with $u, v \in \Re^2$ as two 2−dimensional points u and v on the plane, and complete the parallelogram that includes those two points and the origin. The signed area of this parallelogram is the determinant. If you sweep clockwise from u to v, the determinant $\begin{vmatrix} u \\ v \end{vmatrix}$ is negative; otherwise, positive. This geometry is especially useful for understanding the geometry of a degenerate pivot. In this thesis we will apply this geometric interpretation of dictionary coefficients: In Chapter 3 we use it to improve upper bounds on the maximal number of stalling iterations of the simplex method, in Chapter 6 we use it to compute objective functions for pivot path enumeration. The proof of Theorem 1.6 is provided in the Appendix.

CHAPTER 2
Cycling the Simplex Method

"Cycling is certainly not completely understood." - Hoffman, 1951.

2.1 Introduction

What is the magic behind cycling in the simplex method? Every textbook on linear programming dedicates a section to cycling in the simplex method; however, no single textbook provides the reader with a geometric intuition behind cycling. This list includes the classic textbooks of Chvátal [18] and Dantzig [19], and recent books such as Sierksma [86]. The examples are usually presented as a sequence of dictionaries. Such presentations are sure to convince the reader that simplex methods can cycle, but do not provide any insight behind the construction. Consider the first cycling example, concocted by Hoffman in 1951 [49], [19, p.229],

with starting dictionary

$$
\begin{aligned}
x_0 &= 1 \\
x_1 &= -\cos\varphi x_3 + \omega\cos\varphi x_4 - \cos 2\varphi x_5 + 2\omega\cos^2\varphi x_6 \\
&\quad - \cos 2\varphi x_7 - 2\omega\cos^2\varphi x_8 - \cos\varphi x_9 - \omega\cos\varphi x_{10} \\
x_2 &= -[(\tan\varphi\sin\varphi)/\omega]x_3 - \cos\varphi x_4 - [(\tan\varphi\sin 2\varphi)/\omega]x_5 - \cos 2\varphi x_6 \\
&\quad + 2(\sin^2\varphi/\omega)x_7 - \cos 2\varphi x_8 + [(\tan\varphi\sin\varphi)/\omega]x_9 - \cos\varphi x_{10} \\
z &= [(1-\cos\varphi)/\cos\varphi]x_3 - \omega x_4 - 2\omega x_6 - 4\sin^2\varphi x_7 \\
&\quad + 2\omega\cos 2\varphi x_8 - 4\sin^2\varphi x_9 - \omega(1-2\cos\varphi)x_{10}.
\end{aligned}
$$

If we set $\varphi = 2\pi/5, and \omega > (1-\cos 2\varphi)/(1-2\cos\varphi) \approx 4.74$ the simplex method using Dantzig's rule cycles through a sequence of 10 dictionaries[1] . The left hand side variables cycle repeatedly through the sequence $\{x_0, x_1, x_2\}$, $\{x_0, x_2, x_3\}$, ... , $\{x_0, x_9, x_{10}\}$, $\{x_0, x_{10}, x_1\}$.

How was this example constructed? Strangely, even Hoffman forgot!

> *"Finally, we regret that we are unable at this date to recall any details of the considerations that led to the construction of the example, beyond the fact that the geometric meaning of [a pivot rule for the simplex method] was very much in the foreground."* - Hoffman, 1951.
>
> *"Two months after I made up the example, I lost the mental picture which produced it..."* - Hoffman, 1994.

This is a shame, as geometry can certainly help students understand the simplex method through a visual interpretation. In 1997 Lee [67] successfully untangled

[1] Note that in [19], [39], [49], and [67] the bound on ω reads $\omega > (1-\cos\varphi)/(1-2\cos\varphi) \approx 1.81$. In fact, the tighter bound presented here is required for Hoffman's LP to cycle using Dantzig's pivot rule. See [9] for details.

the algebraic derivation of Hoffman's example, and using *column geometry* (originally known as the *simplex interpretation* [19, p.160]) produced a three-dimensional picture. In Figure 2–1 we exhibit Hoffman's example in two-dimensions using the geometry of the dual simplex method, which was first explained and used to construct cycles by Beale [11], but not used to analyze Hoffman's example:

> *"Unfortunately, the geometric motivation behind Hoffman's example is not easy to grasp"* - Beale, 1955.

In the next section we see how cycling works in this setting, by giving a simplified version of this geometrical interpretation. We show that cycling examples obtained from various linear programming texts have a similar structure, and many are combinatorially equivalent. In Section 2.3 we give a simple method of constructing examples that contain cycles of arbitrary even length. In Section 2.4 we use the method to generate linear programs with $m + n$ inequalities in n dimensions on which the simplex method can cycle and visit $\Theta(n^m)$ different bases.

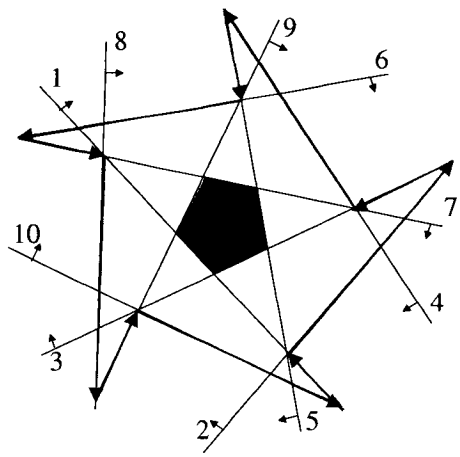


Figure 2–1: A picture of Hoffman's cycle

2.2 Visualizing Cycles

For the remainder of this chapter we will deal with linear programs in the standard form (1.18) with n nonnegative variables and m additional inequalities.

Let's continue with our example from Chapter 1, the linear program

$$\begin{aligned} \max\ z &= 10x_1 - 57x_2 - 9x_3 - 24x_4 \\ 0.5x_1 - 5.5x_2 - 2.5x_3 + 9x_4 &\leq 0 \\ 0.5x_1 - 1.5x_2 - 0.5x_3 + x_4 &\leq 0 \\ x_1, x_2, x_3, x_4 &\geq 0, \end{aligned} \tag{2.1}$$

and starting dictionary

$$\begin{aligned} x_5 &= 0 - 0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 \\ x_6 &= 0 - 0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 \\ z &= 0 + 10x_1 - 57x_2 - 9x_3 - 24x_4. \end{aligned} \tag{2.2}$$

The simplex method with Dantzig's rule visits the dictionaries

$$\begin{aligned} x_1 &= 11x_2 + 5x_3 - 18x_4 - 2x_5 \\ x_6 &= -4x_2 - 2x_3 + 8x_4 + x_5 \\ z &= 53x_2 + 41x_3 - 204x_4 - 20x_5, \end{aligned} \tag{2.3}$$

$$\begin{aligned} x_1 &= -0.5x_3 + 4x_4 + 0.75x_5 - 2.75x_6 \\ x_2 &= -0.5x_3 + 2x_4 + 0.25x_5 - 0.25x_6 \\ z &= 14.5x_3 - 98x_4 - 6.75x_5 - 13.25x_6, \end{aligned} \tag{2.4}$$

$$\begin{aligned} x_2 &= x_1 - 2x_4 - 0.5x_5 + 2.5x_6 \\ x_3 &= -2x_1 + 8x_4 + 1.5x_5 - 5.5x_6 \\ z &= -29x_1 + 18x_4 + 15x_5 - 93x_6, \end{aligned} \tag{2.5}$$

$$
\begin{aligned}
x_3 &= 2x_1 - 4x_2 - 0.5x_5 + 4.5x_6 \\
x_4 &= 0.5x_1 - 0.5x_2 - 0.25x_5 + 1.25x_6 \\
z &= -20x_1 - 9x_2 + 10.5x_5 - 70.5x_6,
\end{aligned} \tag{2.6}
$$

$$
\begin{aligned}
x_4 &= -0.5x_1 + 1.5x_2 + 0.5x_3 - x_6 \\
x_5 &= 4x_1 - 8x_2 - 2x_3 + 9x_6 \\
z &= 22x_1 - 93x_2 - 21x_3 + 24x_6,
\end{aligned} \tag{2.7}
$$

and returns to dictionary (2.2), cycling through the sequence of bases $\{5,6\}$, $\{1,6\}$, $\{1,2\}$, $\{2,3\}$, $\{3,4\}$, $\{4,5\}$, $\{5,6\}$... The six inequalities of the linear program (2.1) define a four-dimensional polyhedron, and the simplex method cycles among the bases of a single vertex. This geometry cannot help us visualize the cycle. In 1955, Beale [11] constructed a cycle for the simplex method by considering the geometry of the dual problem. The dual of (2.1) is

$$
\begin{aligned}
\min\ w &= 0y_1 + 0y_2 \\
(1) &: 0.5y_1 + 0.5y_2 \geq 10 \\
(2) &: -5.5y_1 - 1.5y_2 \geq -57 \\
(3) &: -2.5y_1 - 0.5y_2 \geq -9 \\
(4) &: 9y_1 + y_2 \geq -24 \\
(5) &: y_1 \geq 0, \\
(6) &: y_2 \geq 0
\end{aligned} \tag{2.8}
$$

which has six inequalities in two dimensions. We can also write the dual problem in dictionary form. In fact, the dual dictionary is simply a disguised version of the (primal) dictionary for the primal: the transposed system with signs reversed.

Table 2–1: Primal and dual dictionaries

$$
\begin{array}{lcl}
 & & y_3 = -10 + 0.5y_1 + 0.5y_2 \\
x_5 = 0 - 0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 & & y_4 = 57 - 5.5y_1 - 1.5y_2 \\
x_6 = 0 - 0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 & \longleftrightarrow & y_5 = 9 - 2.5y_1 - 0.5y_2 \\
z = 0 + 10x_1 - 57x_2 - 9x_3 - 24x_4 & & y_6 = 24 + 9y_1 + y_2 \\
 & & w = 0 + 0y_1 + 0y_2.
\end{array}
$$

In the dual simplex method the $w-$row is assumed to have all coefficients nonpositive (dual feasibility). The leaving basic variable is chosen from the basic variables with negative constant on the right hand side of the dictionary. A ratio test is used to choose the entering variable so that the $w-$row remains nonpositive. It is easy to verify that the dual simplex method applied to the dual problem will, with a suitably chosen pivot rule, follow the same sequence of pivots as the primal simplex method does on the primal problem. The sequence of bases for the primal problem becomes the sequence of cobases for the dual problem, and vice versa. Graphing (2.8), the geometry behind a simplex method cycle begins to unveil. We obtain an arrangement of lines, each bounding an inequality of (2.8). Figure 2–2 depicts this arrangement.

The two-dimensional geometric interpretation of the dual simplex method is simple when the objective function is zero. For the time being, let's ignore specific pivot rules and consider simplex pivots in general. Every line in the arrangement corresponds to a variable in the dual dictionary. The slack variable y_i for $i = 3, ..., n$ corresponds to line $i-2$, while y_1 and y_2 correspond to lines $n-1$ and n respectively.

Each intersection of lines corresponds to a basic dual solution obtained by setting the two corresponding variables to zero. These are the two dual cobasic variables for a corresponding dual dictionary. For example, the dual dictionary in Table 2–1 with cobasis {1,2} corresponds to the intersection of lines labelled 5 and 6. (Notice that variable x_i of a primal dictionary conveniently corresponds to line i of the dual diagram!) The basic variable chosen in a pivot corresponds to a violated inequality: in our case the intersection point lies on the wrong side of line 1, so

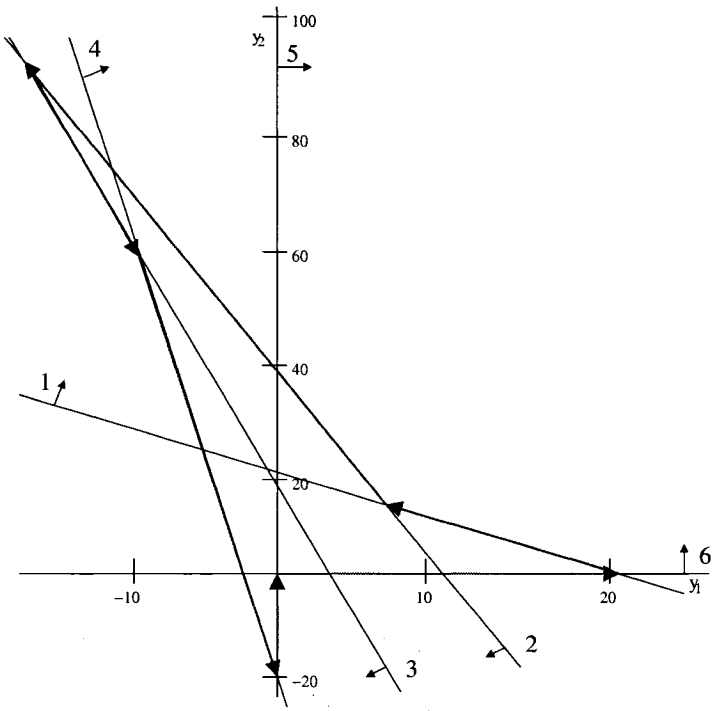


Figure 2–2: Chvátal's example [18]

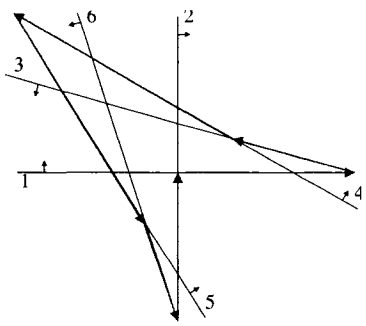
y_3 is the leaving basic variable. The entering variable is chosen from those with a positive coefficient in the equation for the leaving variable, so that that the new basic solution will satisfy the inequality corresponding to the entering variable. In the example we could choose either y_1 or y_2 as the entering variable. However from the cobasis {1,6} we could pivot to the cobasis {1,2} since this is on the correct side of line 4. We could not pivot to cobasis {3,6} as this is on the wrong side of line 5. A negative basic variable indicates a degree of infeasibility, for example $y_3 = -10 + 0.5y_1 + 0.5y_2$ indicates that at the intersection of line 5 and line 6, line 1 is infeasible by $\frac{10}{0.5}$ units along line 5 and by $\frac{10}{0.5}$ units along line 6 with respect to the vector space basis $\{y_1, y_2\}$. The dual simplex method terminates when it reaches a point where either no inequalities are violated (optimality) or when it cannot pivot to a violated inequality without violating one of the two intersecting lines (dual infeasible).

It is now routine to check that the cycles shown in Figure 2–1 for Hoffman's example, and Figure 2–2 for our example correspond to valid dual simplex pivots. It follows that the primal problems pivot through the same cycles, with bases/cobases interchanged. It is more convenient to remember that variable x_i of a primal dictionary conveniently corresponds to line i of the dual diagram.

Our example is derived from the following example found in Chvátal's book. The first dictionary of the cycle is presented as

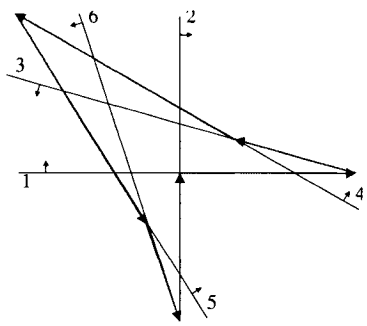
$$\begin{aligned} x_5 &= -0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 \\ x_6 &= -0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 \\ x_7 &= 1 - x_1 \\ z &= 10x_1 - 57x_2 - 9x_3 - 24x_4, \end{aligned} \tag{2.9}$$

and consists of seven variables and three equations. However, variable x_7 never leaves the basis as the cycle only pivots on the variables x_1 through x_6. The equation $x_7 = 1 - x_1$ only serves to bound the linear program but can be removed from dictionary (2.9) without affecting the cycle. It turns out that many other examples found in literature, including Hoffman's, can be reduced in this way and then graphed in two-dimensions. In fact, many are then combinatorially equivalent.



Yudin and Gol'shtein [98]

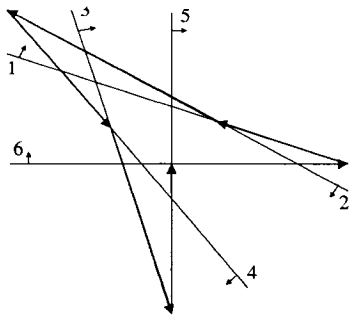
$$\begin{aligned} x_1 &= 0 - 2x_3 + 3x_4 + 5x_5 - 6x_6 \\ x_2 &= 0 - 6x_3 + 5x_4 + 3x_5 - 2x_6 \\ z &= 0 + x_3 - x_4 + x_5 - x_6 \end{aligned}$$



Yudin and Gol'shtein [98]

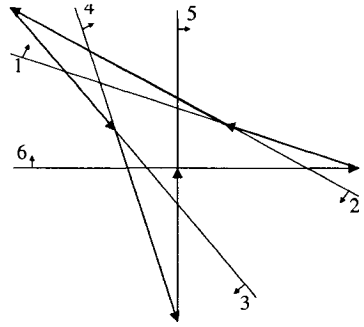
$$\begin{aligned} x_1 &= 0 - x_3 + 2x_4 + 3x_5 - 4x_6 \\ x_2 &= 0 - 4x_3 + 3x_4 + 2x_5 - x_6 \\ z &= 0 + x_3 - x_4 + x_5 - x_6 \end{aligned}$$

Table 2–2: Some examples of cycling found in literature



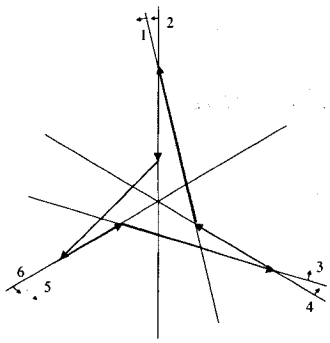
Beale [11]

$$x_5 = 0 - \tfrac{1}{4}x_1 + 60x_2 + \tfrac{1}{25}x_3 - 9x_4$$
$$x_6 = 0 - \tfrac{1}{2}x_1 + 90x_2 + \tfrac{1}{50}x_3 - 3x_4$$
$$z = 0 + \tfrac{3}{4}x_1 + 150x_2 + \tfrac{1}{50}x_3 - 6x_4$$



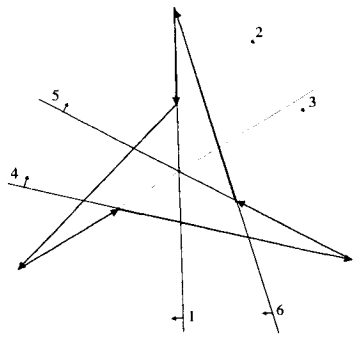
Sierksma [86]

$$x_5 = 0 - x_1 + 32x_2 + 4x_3 - 36x_4$$
$$x_6 = 0 - x_1 + 24x_2 + x_3 - 6x_4$$
$$z = 0 + 3x_1 - 80x_2 + 2x_3 - 24x_4$$



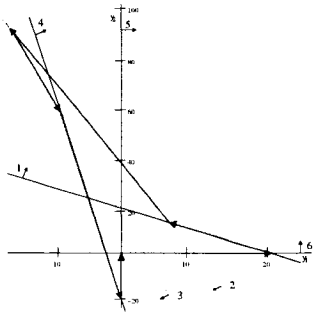
Solow [87]

$$x_1 = 0 + 7x_3 + 3x_4 - 7x_5 - 2x_6$$
$$x_2 = 0 - 2x_3 - x_4 + 3x_5 + x_6$$
$$z = 0 + 2x_3 + 2x_4 - 8x_5 - 2x_6$$



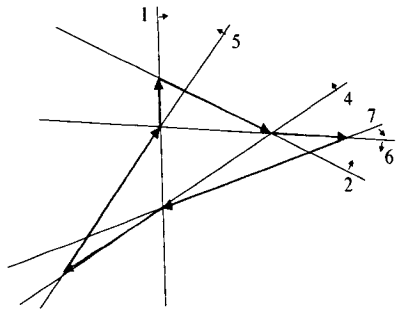
Marshall and Suurballe [68]

$$x_1 = 0 - \tfrac{1}{2}x_3 + \tfrac{11}{2}x_4 + \tfrac{5}{2}x_5 - 9x_6$$
$$x_2 = 0 - \tfrac{1}{2}x_3 + \tfrac{3}{2}x_4 + \tfrac{1}{2}x_5 - x_6$$
$$z = 0 - x_3 + 7x_4 + 1x_5 - 2x_6$$



Chvátal [18]

$$x_5 = 0 - 0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4$$
$$x_6 = 0 - 0.5x_1 + 1.5x_2 + 0.5x_3 - x_4$$
$$z = 0 + 10x_1 - 57x_2 - 9x_3 - 24x_4$$



Kuhn [10]

$$x_1 = 0 + 2x_4 + 9x_5 - x_6 - 9x_7$$
$$x_2 = 0 - \tfrac{1}{3}x_4 - x_5 + \tfrac{1}{3}x_6 + 2x_7$$
$$z = 0 + 2x_4 + 3x_5 - x_6 - 12x_7$$

In all eight examples in Table 2–2, the primal linear program (after reduction to 2 equations) is unbounded. Thus, by duality theory, all six dual linear programs above are infeasible. Nonetheless, the simplex method cycles without detecting this fact. (Note that the shaded region in Figure 2–1 corresponds to the feasible set of Hoffman's dual linear program).

In no way is the geometry described in this section constrained to cycles with $m = 2$, as when $m = d$ oriented lines become $d-$dimensional oriented hyperplanes and a dual simplex pivot has an analogous description to its $m = 2$ counterpart. The geometry can be used to explain cycling examples with $m \geq 3$ such as [32], [68] and [77]. For the time being we choose to restrict ourselves to the two-dimensional interpretation for simplicity. In Section 2.4 we will elevate to $m = d$ for any d.

2.3 Constructing Cycles

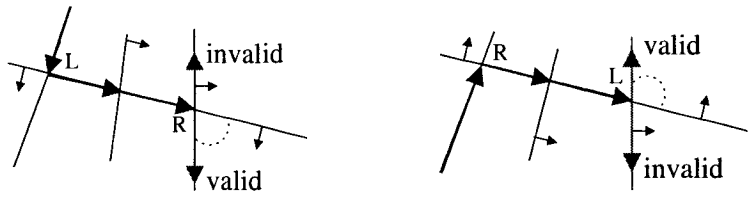
> *"Constructing an LP problem on which the simplex method may cycle is difficult."* - Chvátal [18, p. 33], 1982.

Every two-dimensional arrangement of k inequalities can be encoded as a primal dictionary with k equations and $k + 2$ variables. For the time being let's ignore specific pivot rules and consider simplex pivots in general. Only two basic rules need to be followed to build a simplex method cycle:

1. Each pivot must be a simplex pivot: the pivot step from the intersection of lines r and s to the intersection of lines r and t is a simplex pivot if and only if the inequality bounded by t is violated at $\{r, s\}$ and the inequality bounded by s is satisfied at $\{r, t\}$.
2. The cycle must be represented by closed loop in the diagram.

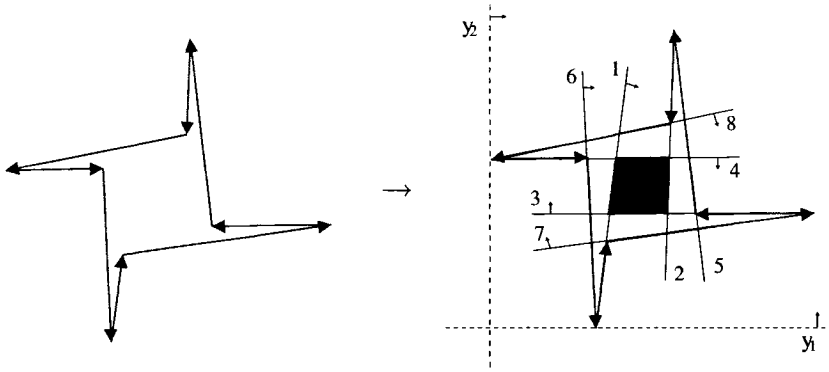
The path consists of an alternating series of left and right turns in the line arrangement, as the next point must always satisfy the two currently intersecting lines.

Table 2–3: Left and right turns



We use this fact to construct a cycle from scratch and the reader can follow the construction in Table 2–4. First, using line segments we draw a path consisting of an alternating series of left and right turns that forms a closed loop. Note that the number of points must be even to form a closed loop, and the loop can indeed be self-crossing (as in Table 2–5). Next, we choose coordinate axes (y_1, y_2), and lines bounding $y_1 \geq 0, y_2 \geq 0$ so that the loop lies in the positive quadrant.

Table 2–4: Constructing cycles from scratch



$$
\begin{aligned}
& \min 0y_1 + 0y_2 \\
& (1) : -y_1 \leq -10 \\
& (2) : y_1 \leq 15 \\
& (3) : -y_2 \leq -10 \\
\rightarrow \quad & (4) : y_2 \leq 15 \\
& (5) : 10y_1 + y_2 \leq 175 \\
& (6) : -10y_1 - y_2 \leq -100 \\
& (7) : y_1 - 10y_2 \leq -70 \\
& (8) : -y_1 + 10y_2 \leq 150 \\
& (9), (10) : y_1, y_2 \geq 0
\end{aligned}
$$

We then fit a line through each line segment in the loop. For each line assign a direction of feasibility which coincides with the path. Compute these inequalities

with respect to (y_1, y_2), and add a zero objective function to obtain a dual LP. By duality, we convert to a primal LP, add slack variables, and compute an initial dictionary:

$$\begin{aligned} x_2 &= 0 + x_1 - 10x_3 + 10x_4 - 101x_7 + 101x_8 + x_9 - 10x_{10} \\ x_5 &= 0 + x_3 - x_4 + x_6 + 10x_7 - 10x_8 + x_{10} \\ z &= -5x_1 - 15x_3 + 10x_4 - 75x_6 - 165x_7 + 85x_8 - 15x_9 - 25x_{10}. \end{aligned} \tag{2.10}$$

Now simplex pivots in our diagram correspond to simplex pivots in the primal linear program, cycling repeatedly through the sequence $\{x_2, x_5\}$, $\{x_2, x_8\}$, $\{x_4, x_8\}$, $\{x_4, x_6\}$, $\{x_1, x_6\}$, $\{x_1, x_7\}$, $\{x_3, x_7\}$, $\{x_3, x_5\}$[2] .

In order for the cycle to follow a specific pivot rule, an additional step is required. Note that for each inequality, $a_{i1}y_1 + a_{i2}y_2 \leq b_i$ for $i = 1, ..., 8$, we can scale both sides by a positive constant C_i at will as $a_{i1}y_1 + a_{i2}y_2 \leq b_i \Leftrightarrow C_i(a_{i1}y_1 + a_{i2}y_2) \leq C_i b_i$. These constants in turn scale the primal variables, yielding the dictionary,

$$\begin{aligned} x_2 &= 0 + \frac{1}{C_2}x_1 - \frac{10}{C_2}x_3 + \frac{10}{C_2}x_4 - \frac{101}{C_2}x_7 + \frac{101}{C_2}x_8 \\ x_5 &= 0 + \frac{1}{C_5}x_3 - \frac{1}{C_5}x_4 + \frac{1}{C_5}x_6 + \frac{10}{C_5}x_7 - \frac{10}{C_5}x_8 \\ z &= -5C_1x_1 - 15C_3x_3 + 10C_4x_4 - 75C_6x_6 - 165C_7x_7 + 85C_8x_8. \end{aligned} \tag{2.11}$$

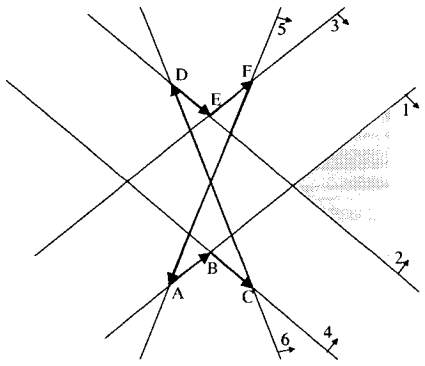
By pivoting along the desired cycle, the corresponding dictionaries constrain the values of C_i we can choose so that the pivot rule is followed. For example,

[2] We can ignore variables x_9 and x_{10} as they are not involved in the cycle. They correspond, respectively, to inequalities $y_1 \geq 0$ and $y_2 \geq 0$ of the dual LP which are only used to specify the coordinate axes

for the first pivot to follow Dantzig's rule we need $85C_8 > 10C_4$ in the primal dictionary. We accumulate all the constraints on C_i in this manner, and only if we find satisfying values for the C_i's then following the pivot rule yields the cycle. (The reader can easily check that for any dictionary the coefficient for variable x_j in the $z-$row is strictly a multiple of C_j.) In our example we need $85C_8 > 10C_4$, $85C_6 > 10C_1$, $80C_7 > 10C_3$, $135C_5 > 15C_2$, thus setting $C_i = 1$ for $i = 1, ..., 8$ results in a simplex method cycle following Dantzig's pivot rule.

This process has a geometric analogue that can guide us to construct a diagram to ensure that Dantzig's rule (or any other coefficient based rule) will follow the loop. The sufficient condition is as follows: given a diagram, for each point along the loop we list the pivot rule requirements for the cycle to be followed. For example, at point A in Table 2–5 we want to pivot to violated line 4 instead of line 2 or 6 which are also violated at A. This translates into constraints $C_4 > C_2$ and $C_4 > C_6$. Similar constraints are obtained from all points B, C, D, E and F along the cycle. If the set of constraints are satisfiable, as is the case here, then there exist C_i's large enough such that Dantzig's rule will cycle on the resulting LP. Of course the challenge is to draw a diagram with this in mind!

Table 2–5: Crossing loop



point A : C_4 bigger than C_2 and C_6
point B : C_6 bigger than C_2
point C : only line 2 violated
point D : C_3 bigger than C_1 and C_5
point E : C_5 bigger than C_1
point F : only line 1 violated

2.4 New Bounds on Cycle Lengths

The length of a simplex method cycle is defined as the number of different bases involved in the cycle. Given a linear program with n nonnegative variables and m inequalities ($m+n$ inequalities in n dimensions), how long can a cycle be?

Theorem 2.1 (Beale [11], Yudin and Gol'shtein [98]). *The minimum length of a simplex method cycle is at least six.*

Theorem 2.2 (Marshall and Suurballe [68]). *A cycling example must have* $m \geq 2, n \geq 6$ *and* $n \geq m+3$.

Many of the cycling examples found earlier in this chapter have length six, and in our initial example (2.1) $m = 2$ and $n = 6$. It is an interesting problem to determine the maximum length of a simplex method cycle, both for special refinements (such as Dantzig's rule) and in general (choose the entering variable arbitrarily). Let $S(n,m)$ be the maximal length of a simplex method cycle for a linear program with n nonnegative variables and m inequalities. If we define $S_\gamma(n,m)$ to be the maximal length of a simplex method cycle following pivot rule γ, then $S_\gamma(n,m) \leq S(n,m)$, and in particular $S_{Dan}(n,m) \leq S(n,m)$ for Dantzig's rule.

Theorem 2.3. $S_{Dan}(n,m)$ *is* $\Omega(n)$ *for* $m \geq 2, n \geq 6$.

Proof. In the previous section we provided examples with $n = 6, 8$ on which the simplex method cycles following Dantzig's rule with cycle length n. The family of cycles of length $n \geq 10$ derived by Lee [67] can be made to follow Dantzig's rule. We show that Lee's cycles can be made to satisfy the sufficient condition presented in Section 2.3.

Lee's examples consists of $2k$ inequalities, for $k \geq 5$, k of which define a regular $k-$polygon. The polygon inequalities are given odd labels in an anti-clockwise order, such that the vertices of the polygon are the intersection of lines i and $(i+2) \bmod 2k$ for $i = 1, ..., 2k-1$ and odd. We call the intersection of lines i and $(i+4) \bmod 2k$, denoted $\{i,(i+4) \bmod 2k\}$, for odd $i = 1, ..., 2k-1$ a *star point.* Let

c_1 be the circle that contains all k *star points*, and r_1 its radius. Let c_2 be the circle sharing center with c_1 with radius $r_2 > r_1$.

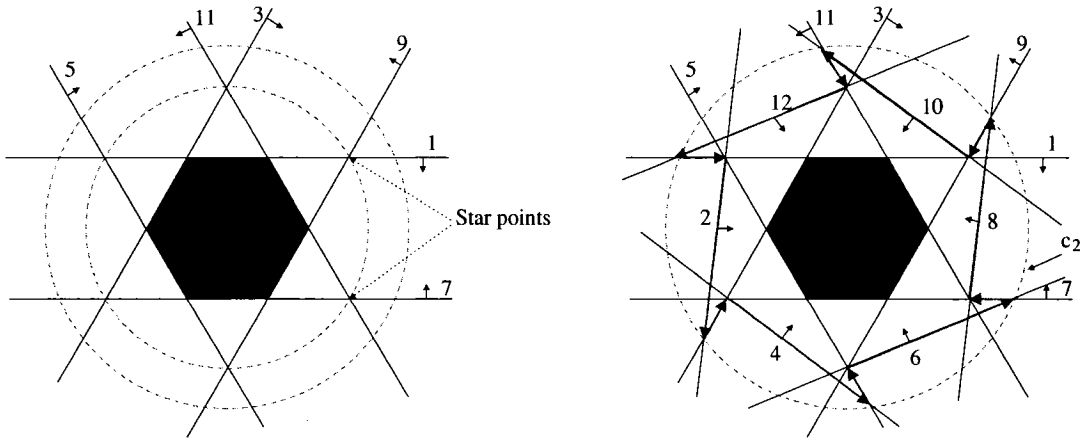


Figure 2–3: Lee's cycle with $k = 6$

The k inequalities with even label are defined as follows: inequality i, for $i = 2, ..., 2k$ and even, is bounded by the line through the two points a) *star point* $\{(i+3) \bmod 2k, (i-1) \bmod 2k\}$ and b) the intersection point of line $(i+1) \bmod 2k$ and circle c_2 on the violated side of inequality $(i+5) \bmod 2k$. The direction of all the inequalities are chosen so that the regular $k-$polygon represents the dual feasible set. Figure 2–3 depicts the case when $k = 6$. The simplex method cycles: from a point on c_1, $\{i, (i+3) \bmod 2k\}$, to a point on c_2, $\{i, (i+1) \bmod 2k\}$, to a point on c_1, $\{(i+2) \bmod 2k, (i+1) \bmod 2k\}$, and so on... for odd $i = 1, ..., 2k-1$. We can translate this cycle into dictionary notation with scaling parameters $C_1, C_2, ..., C_{2k}$ on the variables. Then whenever we lie on an intersection point on c_2, $\{i, (i+1) \bmod 2k\}$ for odd i, we require $C_{(i+3) \bmod 2k}$ to be greater than $C_{(i+4) \bmod 2k}$ and $C_{(i+6) \bmod 2k}$ in order for the cycle to follow Dantzig's rule. These are the only constraints on the cycle and they do not conflict, satisfying the sufficient condition presented in the previous section. (For $n = 10$, Hoffman's cycle, ω is the scaling parameter). □

$S(n,m)$ is $O(n^m)$ by the trivial upper bound given by the maximum number of bases: $\binom{n+m}{m}$. We will now show constructions which prove that $S(n,2)$ is $\Theta(n^2)$, and more generally that $S(n,m)$ is $\Theta(n^m)$ for $n \geq 3m$ with $m \geq 2$ fixed and even. For all our constructions we will continue to use the geometry described in Section 2.2.

Lemma 2.1 ($S(n,2)$ **is** $\Omega(n^2)$). *When $m = 2$ the simplex method can cycle through $\Omega(n^2)$ bases.*

Proof. The idea is to use $n = 3k + 3$ bounding lines, $2k$ to form a $k \times k$ grid, $k - 1$ lines to act as *turn lines*, and then 4 *loop lines* to close the loop (for an example with $k = 7$ see Figure 2–4):

$$\begin{aligned} \textit{grid lines:} \quad & (i)^{th} : y_1 \geq (i-1)\tfrac{10}{k} \quad \text{for } i = 1, ..., k, \\ & (k+i)^{th} : y_2 \leq (i-1)\tfrac{10}{k} \end{aligned}$$

$$\textit{turn lines:} \quad (2k+i)^{th} : 12y_1 - y_2 \geq 1 + 12i\tfrac{k}{10} \text{ for } i = 1, ..., k-1,$$

$$\begin{aligned} \textit{loop lines:} \quad & (3k)^{th} : 12y_1 - y_2 \geq -11 \\ & (3k+1)^{th} : -y_1 + 12y_2 \leq -11 \\ & (3k+2)^{th} : 6y_1 - 4y_2 \leq -3 \\ & (3k+3)^{th} : -4y_1 + 6y_2 \geq -3. \end{aligned}$$

Starting at the intersection of lines 1 and $3k$ we perform k simplex pivots down line 1 from $\{1, 3k\}$ to $\{1, 2k\}$ to $\{1, 2k-1\}$ ... to $\{1, k+1\}$, each time satisfying the violated $(k+i)^{th}$ inequality for $i = 1, ..., k$. We then pivot to the first *turn* line, the intersection point $\{1, 2k+1\}$, from which we *turn* (pivot) to $\{2, 2k+1\}$ and perform k simplex pivots down line 2 using grid lines. Then we *turn* using line $2k + 2$, and so on... We repeat this process, turning $k - 1$ times and traversing the grid until we reach the grid point $\{k, k+1\}$. We then use the four loop lines to close the cycle (just as in Figure 2–4 - the loop lines are invariant with respect to parameter k)

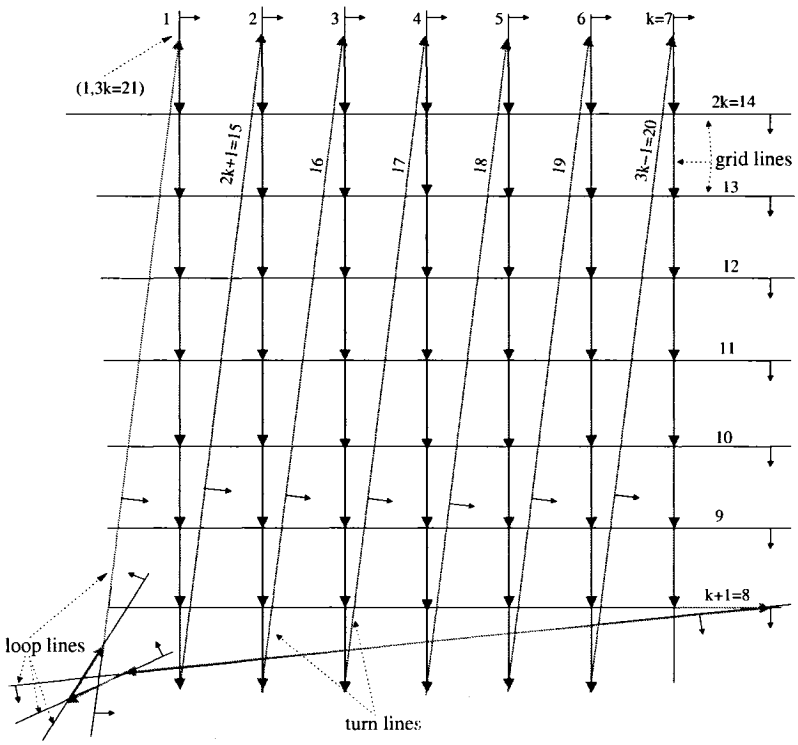


Figure 2–4: $S(n, 2)$ is $\Omega(n^2)$

and return to $\{1, 3k\}$. Note that each pivot is a valid dual simplex pivot and that there are $k^2 + 2k + 3$ bases along the cycle. □

2.4.1 Products of Arrangements

Our goal is to construct a family of linear programs on which the simplex method cycles and visits almost all $\binom{n+m}{m}$ bases. Our approach is to take products of linear programs with long cycles; to use our low-dimensional constructions to generalize to higher dimensions. We first define *products of arrangements* - an extension of products of polytopes.

Proposition 2.1. *Each cell of an hyperplane arrangement is a polyhedron.*

Definition 2.1 (Product of Arrangements). *Given two arrangements A_P and A_Q*

$$A_P = \{u \in \Re^d : a_i u \leq \alpha_i \text{ for } 1 \leq i \leq r\}, \text{ and}$$

$$A_Q = \{v \in \Re^e : b_j v \leq \beta_j \text{ for } 1 \leq j \leq s\},$$

the product $A_P \times A_Q$ is the arrangement

$$A_W = \left\{(u,v) \in \Re^{d+e} : \begin{array}{l} a_i u \leq \alpha_i \text{ for } 1 \leq i \leq r \\ b_j v \leq \beta_j \text{ for } 1 \leq j \leq s \end{array}\right\}. \tag{2.12}$$

Proposition 2.2. *The face poset of a product of arrangements is independent of the hyperplane orientations.*

Note that if we set a bounded cell P of A_P and a bounded cell Q of A_Q to have positive sign vectors, then the product of arrangements $A_P \times A_Q$ is the induced hyperplane arrangement of the polytope product $P \times Q$. We now extend some of the facts about products of polytopes to express the properties of products of arrangements. We facilitate understanding by providing an example (see Figure 2–5) on which the reader can verify the theorem's statements.

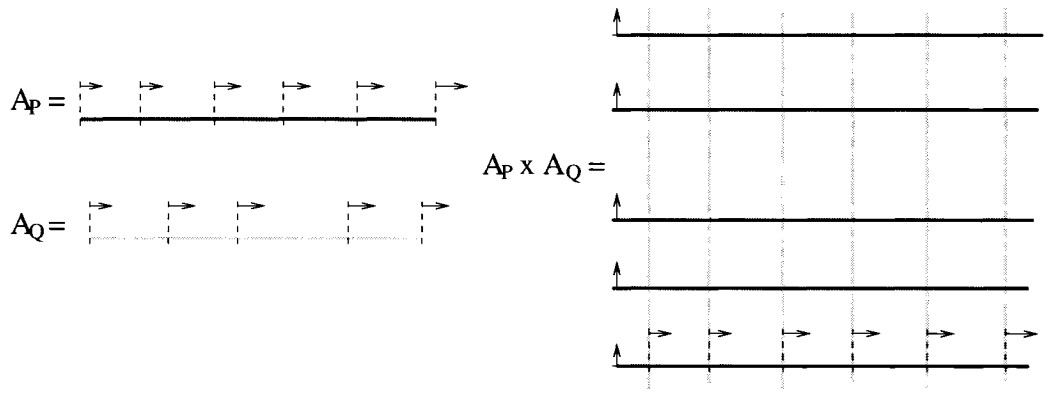


Figure 2–5: Product of arrangements

Theorem 2.4. *Let A_W be the product of arrangements $A_P \subseteq \Re^d$ and $A_Q \subseteq \Re^e$, then:*

- *If A_P has $\overline{p}$ vertices and r hyperplanes, and if A_Q has $\overline{q}$ vertices and s hyperplanes, then $A_W := A_P \times A_Q$ is an arrangement $\subseteq \Re^{d+e}$ and has $\overline{p} \cdot \overline{q}$ vertices and $r + s$ hyperplanes.*
- *Specifically, if $\{p_1, ..., p_{\overline{p}}\}$ are vertices of A_P, $\{q_1, ..., q_{\overline{q}}\}$ the vertices of A_Q, then we can define the $\overline{p} \cdot \overline{q}$ the vertices of A_W, denoted $w(i,j)$, as:*

$$w(i,j) = \binom{p_i}{q_j} : \quad \begin{matrix} 1 \leq i \leq \overline{p} \\ 1 \leq j \leq \overline{q}. \end{matrix} \tag{2.13}$$

- *Given the set of cells C_P and C_Q corresponding respectively to arrangements A_P and A_Q, then the cells $C_W^i \in A_W$ are defined as the product of cells $C_P^j \in A_P$ with $C_Q^k \in A_Q : C_W^i = C_P^j \times C_Q^k$ for all j, k.*

The edge set of A_W consists of two types of edges:

- *A_P-edges* of the form $[w(i',j), w(i'',j)]$ for $1 \leq j \leq \overline{q}$ and for any $1 \leq i', i'' \leq \overline{p}$ such that $p_{i'}$ and $p_{i''}$ are adjacent vertices of A_P and
- *A_Q-edges* of the form $[w(i,j'), w(i,j'')]$ for $1 \leq i \leq \overline{p}$ and for any $1 \leq j', j'' \leq \overline{q}$ such that $q_{j'}$ and $q_{j''}$ are adjacent vertices of A_Q.

2.4.2 The Construction

We are almost ready to explain how to build a long cycles in higher dimensions using a product of two arrangements on which the dual simplex method cycles respectively. First let's generalize the geometric description of the dual simplex method presented in Section 2.2 to higher dimensions. Starting at a vertex defined by a set H of d intersecting hyperplanes, a *pivot* is the operation of exchanging a hyperplane $h_i \in H$ for a hyperplane $h_j \notin H$ that intersects the edge $H \backslash \{h_i\}$. This results in a second vertex defined by $H' := H \backslash \{h_i\} \cup \{h_j\}$. There exists a one-to-one correspondence between the n hyperplanes in dimension $d = m$ and n nonnegative variables of a linear program with m inequalities. Each vertex of the arrangement corresponds to a basis.

Definition 2.2 (Infeasible Vertex). *A vertex p_i of an oriented hyperplane arrangement is infeasible if some oriented hyperplane (inequality) $h \in H$ is violated at p_i.*

Definition 2.3 (Dual Simplex Pivot). *For every infeasible vertex p defined by a set H of d intersecting hyperplanes, there exists an oriented hyperplane $h_j \notin H$ that is violated at p. A pivot from p to vertex p', defined by $H' := H \backslash \{h_i\} \cup \{h_j\}$, is a simplex pivot if p' lies on the nonnegative side of h_i.*

When the objective function is zero, the dual simplex method performs dual simplex pivots and terminates when it reaches a vertex where either no oriented hyperplanes are violated (optimality) or when it cannot pivot to a violated hyperplane without violating one of the d intersecting hyperplanes defining the current vertex (dual infeasible / primal unbounded).

We now describe a *product of cycles.* Let A_P be an arrangement of r oriented hyperplanes in $\Re^d$ with a cycle of length l from p_1 to p_l to p_1... Similarly let A_Q be an arrangement of s oriented hyperplanes in $\Re^e$ with a cycle of length $\overline{k}$ from q_1 to $q_{\overline{k}}$ to q_1... Compute the product $A_W = A_P \times A_Q$. Starting at vertex $w(1,1)$ of A_W we can take $l-1$ simplex pivots along A_P−edges to $w(l,1)$, then pivot once along an A_Q−edge to $w(l,2)$, and pivot once to $w(1,2)$ (which would complete the cycle if were were solely on A_P). From $w(1,2)$ we pivot once along an A_Q−edge to $w(1,3)$, and then take $l-1$ simplex pivots along A_P−edges to $w(l,3)$, and continue in this manner: $\rightarrow w(l,4) \rightarrow w(1,4) \rightarrow w(1,5) \rightarrow ... \rightarrow w(l,5) \rightarrow w(l,6) \rightarrow w(1,6) \rightarrow w(1,7) \rightarrow ... \rightarrow w(l,7) \rightarrow ...$ We keep pivoting until we reach $w(1,\overline{k})$ at then we close the loop by pivoting to $w(1,1)$. The length of the cycle in A_W is $(l+2)\left\lfloor \frac{\overline{k}}{2} \right\rfloor + (\overline{k}+1) \bmod 2$. The cycle in A_W corresponds to a cycle in a linear program with $r+s$ nonnegative variables and $d \cdot e$ inequalities.

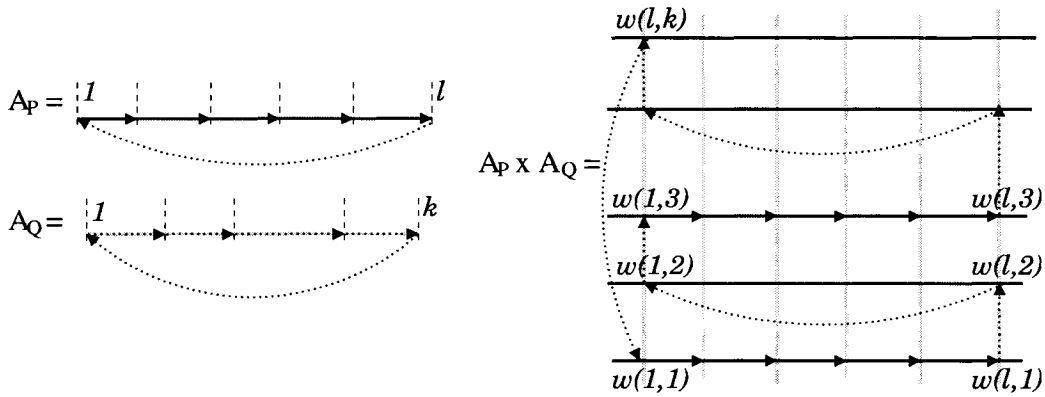


Figure 2–6: (Imaginary) product of cycles

To visualize this product of cycles, let's make believe that the dual simplex method can cycle in one dimension. If this were to be true, then Figure 2–6 would depict the product of two one-dimensional cycles yielding a cycle of dimension two.

Theorem 2.5. *For $s \geq 6, n \geq 6, n \geq m+3, m \geq 2$, and some constant $C > 1$,*

$$S(n+s, m+2) \geq \frac{s^2}{C} \cdot S(n,m). \tag{2.14}$$

Proof. Take a linear program with n nonnegative variables and m inequalities which has a cycle of length $l = S(n,m)$. The corresponding arrangement A_P consists of n oriented hyperplanes in dimension m. Now take the product of arrangements A_P and A_Q where A_Q is defined as in Lemma 2.1 with a total of s lines. By our product of cycles construction, the resulting product yields a simplex method cycle of length greater than $(S(n,m)+2)\left\lfloor \frac{s^2}{2K} \right\rfloor + (\frac{s^2}{K}+1) \bmod 2 \geq \frac{s^2}{C} \cdot S(n,m)$ for some constants $C, K > 1$. □

Corollary 2.1. *For $n \geq 3m, m \geq 2$ and even, $S(n,m) = \Omega(n^m)$. More specifically, for some constant $C > 1$:*

$$S(n,m) \geq \left\lfloor \frac{2n}{mC} \right\rfloor^m. \tag{2.15}$$

Proof. (By induction)

$$\begin{aligned} S(st, 2t) &\geq \frac{s^2}{C} S(s(t-1), 2(t-1)) \\ &\vdots \quad t \text{ times} \\ &\geq \left(\frac{s^2}{C}\right)^{t-1} S(s, 2) \\ &= \frac{s^{2t}}{C^t} \qquad \text{by Lemma 2.1.} \end{aligned}$$

Substituting for $m = 2t$ and $s = \frac{2n}{m}$, we get

$$S(n, m) \geq \left(\frac{2n}{mC}\right)^m. \tag{2.16}$$

□

Corollary 2.2 (Main Theorem). *For fixed m, the function $S(n, m)$ grows like a polynomial of degree m in n:*

$$S(n, m) \text{ is } \Theta(n^m) \text{ for } n \geq 3m \text{ for fixed and even } m \geq 2. \tag{2.17}$$

$$S(n, m) \text{ is } \Omega(n^{m-1}) \text{ for } n \geq 3(m-1) \text{ for fixed and odd } m \geq 3. \tag{2.18}$$

In order to tighten our bound when m is odd, we require a base case cycling example with odd m of length $\Theta(n^m)$. By Theorem 2.2 $S(n, 1) = 0$, and so our next best hope is to find an example with $m = 3$ of length $\Theta(n^3)$.

Remark 2.1. *We require $n \geq 3m$ when we take iterative products of the basic construction presented in Lemma 2.1. So while we can construct cycles of length $\Theta(n^m)$, when $n \geq 3m$ $S(n, m)$ is asymptotically less than the maximal number of vertices that a $n-$dimensional polytope with $n + m \leq \lfloor \frac{4n}{3} \rfloor$ facets can have, which is $\Theta((n+m)^{\lfloor \frac{n}{2} \rfloor})$ by McMullen's Upper Bound Theorem [72], since $O(n^m) < \Omega((n+m)^{\lfloor \frac{3m}{2} \rfloor})$.*

Open Problem 2.1. *Construct a family of cycling examples with odd m with cycle length $\Theta(n^m)$.*

Open Problem 2.2. *Construct cycles whose length (asymptotically) exceeds* $\Theta((n+m)^{\lfloor \frac{n}{2} \rfloor})$.

Open Problem 2.3. *Tighten the bounds on* $S_{Dan}(n,m)$*, currently* $\Omega(n)$ *and* $O(n^m)$*, and for other pivot rules.*

CHAPTER 3
Stalling the Simplex Method - Part I

3.1 Introduction

While degeneracy can cause most refinements of the simplex method to cycle, there exist employable techniques which, when used, guarantee that the simplex method will terminate. The most primitive device for avoiding degeneracy, let alone cycling, is the *perturbation method*[1] [17]. Degeneracy is avoided by perturbing the linear program by adding a small ϵ_i to the i^{th} inequality with $\epsilon_i << \epsilon_{i+1}$ so that the resulting inequalities define a simple polyhedron P, and the simplex method then follows a strict (objective value increasing) monotone path along the edges of P. The ϵ_i's must be small, usually in the order of 10^{-6}, and require extra special arithmetic precaution during an iteration of the simplex method or they may wreak numerical havoc. The *lexicographic method* [20], [96] is a more sophisticated scheme, an implementation of the perturbation method where the ϵ_i's are treated as symbols [18, p. 34]. It does not modify the input, rather it defines an order on a subset of the bases, the *lex-positive* bases, such that with each degenerate pivot the objective

[1] We only study perturbation of the b-vector of a linear program. Perturbations of matrix A are also possible, used in matrix analysis [63].

function increases lexicographically. The (lexicographic) ratio test requires extra computation to ensure that subsequent bases remain lex-positive.

The simplest way of avoiding cycling, which wins with respect to finding

"a way of avoiding [cycling] that involved as little extra work as possible"

- G. Dantzig [19, p.231],

is Bland's least-index pivot rule [13]. For a linear program

$$\begin{aligned} &\text{maximize } z = \sum_{j=1}^{d} c_j x_j \\ &\text{subject to } \sum_{j=1}^{d} a_{ij} x_j \leq b_i \qquad \text{for } i = 1, ..., n, \end{aligned} \tag{3.1}$$

and a dictionary,

$$\begin{aligned} x_i &= b'_i - \sum_{j \in N} a'_{ij} x_j \qquad \text{for } i \in B \\ z &= z' + \sum_{j \in N} c'_j x_j, \end{aligned} \tag{3.2}$$

with feasible basis B and cobasis N, an iteration of the simplex method with Bland's rule selects the least-indexed cobasic variable x_s with a positive coefficient in the $z-$row as the entering variable and chooses the leaving variable, x_r, by performing a ratio test,

$$r = ratio(s) = argmin\{b'_i/a'_{is} : i \in B, a'_{is} > 0\},$$

where $argmin$ returns the smallest index i minimizing the given ratio. Recall that if no such x_s is found, then the basis is optimal. If $a'_{is} \leq 0$ for all $i \in B$, then the linear program is unbounded.

Theorem 3.1 (Bland [13]). *The simplex method with Bland's rule is finite.*

Proof. (By contradiction.) Assume that the simplex method with Bland's rule fails to terminate on a particular linear program LP, that it cycles on a degenerate

sequence of bases never increasing the objective value. The dictionaries in this cycle will have the form

$$\begin{aligned} x_i &= 0 - \sum_{j \in N} a'_{ij} x_j \qquad \text{for } i \in \overline{B} \\ x_i &= b'_i - \sum_{j \in N} a'_{ij} x_j \qquad \text{for } i \in B \setminus \overline{B} \\ z &= z' + \sum_{j \in N} c'_j x_j, \end{aligned} \tag{3.3}$$

where the basic variables with indices in $B \setminus \overline{B}$ remain basic throughout the cycle. We can omit these rows of the dictionaries without altering the cycle. Thus we can reduce the LP into the following form:

$$\begin{aligned} \max z = \sum_{j=1}^{d} c_j x_j & \\ \sum_{j=1}^{d} \overline{a}_{ij} x_j \leq 0 \qquad & i = 1, ..., (n-d) \\ x_j \geq 0 \qquad & j = 1, ..., d. \end{aligned} \tag{3.4}$$

The dual of (3.4) is:

$$\begin{aligned} \min w = \sum_{i=1}^{n-d} 0 \cdot y_i & \\ \sum_{i=1}^{n-d} \overline{a}_{ij} y_i \geq c_j \qquad & j = 1, ..., d \\ y_i \geq 0 \qquad & i = 1, ..., (n-d). \end{aligned} \tag{3.5}$$

If the primal simplex method with Bland's rule cycles on (3.4), then the dual simplex method with Bland's rule will cycle on (3.5). Note that the objective function of (3.5) is zero, making each pivot step of Bland's rule equivalent to a pivot step of the $b-$rule presented in Chapter 1. Here lies the contradiction as

the b–rule, and thus the simplex method with Bland's rule, is finite by Theorem 1.1. □

(Also see Fujishige [28] for another simple proof of the anticycling nature of Bland's rule.)

Most linear programs arising in practice are degenerate to some degree and, despite anti-cycling theory, degeneracy remains a problem in practice. For example, linear relaxations of many combinatorial optimization problems are highly degenerate. Algorithms employing the lexicographic (perturbation) method or Bland's rule might perform an exponentially long sequence of degenerate pivots. This phenomenon is known as *stalling.* In 1995 Boyd [16] noted practical examples where *"the choice of anti-cycling scheme meant the difference between solving or not solving the linear program."* In particular, Ryan and Osborne [83] discuss aircrew scheduling problems where prolonged stalling prevented them from computing a solution. Even state-of-the-art linear programming codes, usually employing heuristics to combat stalling, throw caution at the prospect of degeneracy and suggest user interruption in the case of excessive stalling:

> *"The majority of LP problems solve best using Cplex's state of the art modified primal simplex algorithm... ...[but] highly degenerate problems with little variability in the right-hand-side coefficients but significant variability in the cost coefficients often solve much faster using dual simplex... ...consider trying dual simplex if numerical problems occur while using primal simplex."* - Cplex 7.5 User Notes.

Avis and Chvátal [5], and later Megiddo [73], have proved that the problem of leaving a degenerate vertex of a linear program with n inequalities in d dimensions is equivalent to solving a nondegenerate linear program with parameters linear in n and d. In other words, any refinement of the simplex method that could guarantee polynomial-length stalling at a degenerate vertex in the worst-case would also solve

a linear program in strongly polynomial time. Until such a pivot rule is discovered, we must continue to live with degeneracy and strive to better understand stalling. For starters we should improve the theoretical bounds on the maximum number of degenerate iterations at a vertex.

In the case of the lexicographic method, the maximum number of iterations required for the simplex method (with any refinement) to terminate on a degenerate linear program with n inequalities in d dimensions is $O(n^{\lfloor \frac{d}{2} \rfloor})$ by McMullen's Upper Bound Theorem [72] as each lex-positive basis is in a one-to-one correspondence with a vertex of a simple $d-$polyhedron defined by n facets. Amenta and Ziegler [2] have shown that there exist nondegenerate constructions for most refinements of the simplex method where the number of iterations is $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$. It follows that there exists constructions requiring $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$ degenerate iterations: Let a completely degenerate linear program be defined as in (3.1) with $b_i = 0$ for $i = 1, ..., n$. Define $D(d, n)$ to be the maximum number of bases that the simplex method visits on a completely degenerate linear program with n inequalities in dimension d. ($D(d, n)$ represents the maximum number of stalling iterations on a degenerate vertex that lies at the intersection of n facets in d dimensions.)

Proposition 3.1. $D(d, n) = \Omega(n^{\lfloor \frac{d}{2} \rfloor})$

Proof. Take a nondegenerate worst-case example LP_{non} of the form (3.1) attaining $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ iterations [2]. Assign $b'_i = b_i$ for $i = 1, ..., n$. Now alter LP_{non} by setting $b_i = 0$ for $i = 1, ..., n$ to get a completely degenerate problem LP_{deg}. LP_{non} represents a perturbation of LP_{deg} requiring $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ iterations, so let's represent this perturbation symbolically: add symbols ϵ_i to the right hand side of inequality i. Let ϵ_i have coefficient b'_i and apply the simplex method. The simplex method will follow the same sequence of pivots on LP_{deg} as in LP_{non}, however it chooses the leaving variable due to the symbolic perturbation rather than just the ratio test. □

It is unclear whether the lexicographic simplex method can visit $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ bases. Similarly, if we choose to follow Bland's rule to ensure finiteness, little is known. How long can the simplex method with Bland's rule stall on a degenerate vertex? Define $D_{Bl}(d, n)$ to be the maximum number of bases that the simplex method with Bland's rule visits on a completely degenerate linear program with n inequalities in dimension d. Trivially $D_{Bl}(d, n) = O(n^d)$ as there are at most $\binom{n}{d}$ bases, and Avis and Chvátal [5] proved that $D_{Bl}(d, n) = \Omega(1.618 \cdots^d)$ for $n \geq 2d$. Tightening the upper and lower bounds on $D_{Bl}(d, n)$ is the focus of this chapter and Chapter 5. It is useful to note that $D_{Bl}(d, n)$ also represents the maximum number of bases visited by the simplex method with Dantzig's rule [2, Observation 2.6].

3.2 Upper Bounds on $D_{Bl}(d, n)$

Problem 3.1. *What is the upper bound on the maximum number of bases that the simplex method with Bland's rule visits on a a completely degenerate linear program with n inequalities in dimension d?*

In this section we present three results. First we introduce a perturbation finding algorithm which we use to show that Bland's rule is not equivalent to a perturbation scheme, leaving open the possibility that $D_{Bl}(d, n) = \Theta(n^d)$. However we then prove that $D_{Bl}(2, n) = n$ and $D_{Bl}(3, n) = O(n^2)$. Our proofs make use of the geometric interpretation of dictionary coefficients presented in Section 1.2.5.

3.2.1 Bland's Rule is not a Perturbation Scheme

In 1990 Yap [97] (and private communication) asked if Bland's rule is equivalent to a perturbation: can we perturb a degenerate linear program to obtain a nondegenerate linear program on which Bland's rule follows the same sequence of bases on both inputs? If this is true then $D_{Bl}(d, n) = \Theta(n^{\lfloor \frac{d}{2} \rfloor})$. We address a more general problem.

Problem 3.2 (Perturbation). *Given a linear program and a finite pivot rule R, let S be a degenerate sequence of bases that the simplex method encounters when*

following pivot rule R. Can we perturb the linear program so that simplex method with rule R follows S nondegenerately?

The following algorithm finds a perturbation if one exists, otherwise returns a certificate showing no such perturbation exists.

Algorithm 3.1 (Perturb(LP,R,S)). *Let LP be maximize* $\sum_{j=1}^{d} c_j x_j$ *subject to* $\sum_{j=1}^{d} a_{ij} x_j \leq b_i$, *for* $i = 1, ..., n$, *and S a sequence of degenerate bases following pivot rule R on LP.*

(1) Construct $\widetilde{LP}$: *maximize* $\sum_{j=1}^{d} c_j x_j$ *subject to* $\sum_{j=1}^{d} a_{ij} x_j \leq b_i + \epsilon_i$, *for* $i = 1, ..., n$ *where the* ϵ_i*'s are variables. Let F be a set of linear inequalities initially empty.*

(2) Compute the dictionary for each basis s of S for $\widetilde{LP}$. *Add to F the (linear) constraints on* $\epsilon_1, \epsilon_2, ..., \epsilon_n$ *so that the dictionary is nondegenerate.*

(3) Find values for ϵ_i, *for* $i = 1, ..., n$ *that satisfy the constraints of F.*

F is a system of strict linear inequalities with variables ϵ_i for $i = 1, ..., n$. F has the form $D\epsilon > 0$ (which has a solution if and only if $D\epsilon \geq 1$ has a solution). If there exists a solution $\epsilon_1^*, \epsilon_2^*, ..., \epsilon_n^*$ then maximize $\sum_{j=1}^{d} c_j x_j$ subject to $\sum_{j=1}^{d} a_{ij} x_j \leq b_i + \epsilon_i^*$, for $i = 1, ..., n$ is a linear program on which the simplex method with pivot rule R follows a sequence S of nondegenerate dictionaries. Otherwise we cannot perturb the linear program and follow sequence S.

Proposition 3.2. *Bland's rule is not equivalent to a perturbation when* $d \geq 3$.

Proof. Consider the following sequence of degenerate dictionaries following the simplex method with Bland's rule.

$$\begin{aligned} x_1 &= \mathbf{0} - x_4 + 9x_5 + 17x_6 \\ x_2 &= \mathbf{0} - 23x_4 + 7x_5 + 31x_6 \\ x_3 &= \mathbf{0} - 30x_4 + 6x_5 + 6x_6 \\ z &= \mathbf{0} + 62x_4 - 14x_5 + 2x_6 \end{aligned}$$

Pivot x_4 with x_1:

$$
\begin{aligned}
x_2 &= \mathbf{0} + 23x_1 - 200x_5 - 360x_6 \\
x_3 &= \mathbf{0} + 30x_1 - 264x_5 - 504x_6 \\
x_4 &= \mathbf{0} - x_1 + 9x_5 + 17x_6 \\
z &= \mathbf{0} - 62x_1 + 544x_5 + 1056x_6
\end{aligned}
$$

Pivot x_5 with x_2:

$$
\begin{aligned}
x_3 &= \mathbf{0} - \tfrac{9}{25}x_1 + \tfrac{33}{25}x_2 - \tfrac{144}{5}x_6 \\
x_4 &= \mathbf{0} + \tfrac{7}{200}x_1 - \tfrac{9}{200}x_2 + \tfrac{4}{5}x_6 \\
x_5 &= \mathbf{0} + \tfrac{23}{200}x_1 - \tfrac{1}{200}x_2 - \tfrac{9}{5}x_6 \\
z &= \mathbf{0} + \tfrac{14}{25}x_1 - \tfrac{68}{25}x_2 + \tfrac{384}{5}x_6
\end{aligned}
$$

Pivot x_1 with x_3:

$$
\begin{aligned}
x_1 &= \mathbf{0} + \tfrac{11}{3}x_2 - \tfrac{25}{9}x_3 - 80x_6 \\
x_4 &= \mathbf{0} + \tfrac{1}{12}x_2 - \tfrac{7}{72}x_3 - 2x_6 \\
x_5 &= \mathbf{0} + \tfrac{5}{12}x_2 - \tfrac{23}{72}x_3 - 11x_6 \\
z &= \mathbf{0} - \tfrac{2}{3}x_2 - \tfrac{14}{9}x_3 + 32x_6
\end{aligned}
$$

Pivot x_6 with x_1:

$$
\begin{aligned}
x_4 &= \mathbf{0} + \tfrac{1}{40}x_1 - \tfrac{1}{120}x_2 - \tfrac{1}{36}x_3 \\
x_5 &= \mathbf{0} + \tfrac{11}{80}x_1 - \tfrac{7}{80}x_2 + \tfrac{1}{16}x_3 \\
x_6 &= \mathbf{0} - \tfrac{1}{80}x_1 + \tfrac{11}{240}x_2 - \tfrac{5}{144}x_3 \\
z &= \mathbf{0} - \tfrac{2}{5}x_1 + \tfrac{4}{5}x_2 - \tfrac{8}{3}x_3
\end{aligned}
$$

Pivot x_2 with x_4:

$$
\begin{aligned}
x_2 &= \mathbf{0} + 3x_1 - \tfrac{10}{3}x_3 - 120x_4 \\
x_5 &= \mathbf{0} - \tfrac{1}{8}x_1 - \tfrac{17}{48}x_3 + \tfrac{21}{2}x_4 \\
x_6 &= \mathbf{0} + \tfrac{1}{8}x_1 - \tfrac{3}{16}x_3 - \tfrac{11}{2}x_4 \\
z &= \mathbf{0} + 2x_1 - \tfrac{16}{3}x_3 - 96x_4
\end{aligned}
$$

Pivot x_1 with x_5:

$$
\begin{aligned}
x_1 &= \mathbf{0} + \tfrac{17}{6}x_3 + 84x_4 - 8x_5 \\
x_2 &= \mathbf{0} + \tfrac{31}{6}x_3 + 132x_4 - 24x_5 \\
x_6 &= \mathbf{0} + \tfrac{1}{6}x_3 + 5x_4 - x_5 \\
z &= \mathbf{0} + \tfrac{1}{3}x_3 + 72x_4 - 16x_5.
\end{aligned}
$$

Now let's attempt to perturb the example so that the same sequence of dictionaries is nondegenerate. We add ϵ_1, ϵ_2 and ϵ_3 to the rows of the initial dictionary. If $\epsilon_i > 0$ for $i = 1, 2$, and 3, then the initial dictionary is nondegenerate.

$$\begin{array}{l} x_1 = \epsilon_1 - x_4 + 9x_5 + 17x_6 \\ x_2 = \epsilon_2 - 23x_4 + 7x_5 + 31x_6 \\ x_3 = \epsilon_3 - 30x_4 + 6x_5 + 6x_6 \\ z = 62x_4 - 14x_5 + 2x_6 \end{array} \longrightarrow \begin{array}{l} (1) : \epsilon_1 > 0 \\ (2) : \epsilon_2 > 0 \\ (3) : \epsilon_3 > 0 \end{array}$$

Pivot x_4 with x_1:

$$\begin{array}{l} x_2 = -\mathbf{23}\epsilon_1 + \epsilon_2 + 23x_1 - 200x_5 - 360x_6 \\ x_3 = -\mathbf{30}\epsilon_1 + \epsilon_3 + 30x_1 - 264x_5 - 504x_6 \\ x_4 = +\epsilon_1 - x_1 + 9x_5 + 17x_6 \\ z = +\mathbf{62}\epsilon_1 - 62x_1 + 544x_5 + 1056x_6 \end{array} \longrightarrow \begin{array}{l} (4) : -23\epsilon_1 + \epsilon_2 > 0 \\ (5) : -30\epsilon_1 + \epsilon_3 > 0 \\ (6) : \epsilon_1 > 0 \end{array}$$

Pivot x_5 with x_2:

$$\begin{array}{l} x_3 = +\frac{\mathbf{9}}{\mathbf{25}}\epsilon_1 - \frac{\mathbf{33}}{\mathbf{25}}\epsilon_2 + \epsilon_3 - \frac{9}{25}x_1 + \frac{33}{25}x_2 - \frac{144}{5}x_6 \\ x_4 = -\frac{\mathbf{7}}{\mathbf{200}}\epsilon_1 + \frac{\mathbf{9}}{\mathbf{200}}\epsilon_2 + \frac{7}{200}x_1 - \frac{9}{200}x_2 + \frac{4}{5}x_6 \\ x_5 = -\frac{\mathbf{23}}{\mathbf{200}}\epsilon_1 + \frac{\mathbf{1}}{\mathbf{200}}\epsilon_2 + \frac{23}{200}x_1 - \frac{1}{200}x_2 - \frac{9}{5}x_6 \\ z = -\frac{\mathbf{14}}{\mathbf{25}}\epsilon_1 + \frac{\mathbf{68}}{\mathbf{25}}\epsilon_2 + \frac{14}{25}x_1 - \frac{68}{25}x_2 + \frac{384}{5}x_6 \end{array} \longrightarrow \begin{array}{l} (7) : \frac{9}{25}\epsilon_1 - \frac{33}{25}\epsilon_2 + \epsilon_3 > 0 \\ (8) : -\frac{7}{200}\epsilon_1 + \frac{9}{200}\epsilon_2 > 0 \\ (9) : -\frac{23}{200}\epsilon_1 + \frac{1}{200}\epsilon_2 > 0 \end{array}$$

Pivot x_1 with x_3:

$$\begin{array}{l} x_1 = +\epsilon_1 - \frac{\mathbf{11}}{\mathbf{3}}\epsilon_2 + \frac{\mathbf{25}}{\mathbf{9}}\epsilon_3 + \frac{11}{3}x_2 - \frac{25}{9}x_3 - 80x_6 \\ x_4 = -\frac{\mathbf{1}}{\mathbf{12}}\epsilon_2 + \frac{\mathbf{7}}{\mathbf{72}}\epsilon_3 + \frac{1}{12}x_2 - \frac{7}{72}x_3 - 2x_6 \\ x_5 = -\frac{\mathbf{5}}{\mathbf{12}}\epsilon_2 + \frac{\mathbf{23}}{\mathbf{72}}\epsilon_3 + \frac{5}{12}x_2 - \frac{23}{72}x_3 - 11x_6 \\ z = +\frac{\mathbf{2}}{\mathbf{3}}\epsilon_2 + \frac{\mathbf{14}}{\mathbf{9}}\epsilon_3 - \frac{2}{3}x_2 - \frac{14}{9}x_3 + 32x_6 \end{array} \longrightarrow \begin{array}{l} (10) : \epsilon_1 - \frac{11}{3}\epsilon_2 + \frac{25}{9}\epsilon_3 > 0 \\ (11) : -\frac{1}{12}\epsilon_2 + \frac{7}{72}\epsilon_3 > 0 \\ (12) : -\frac{5}{12}\epsilon_2 + \frac{23}{72}\epsilon_3 > 0 \end{array}$$

Pivot x_6 with x_1:

$$\begin{array}{l} x_4 = -\frac{\mathbf{1}}{\mathbf{40}}\epsilon_1 + \frac{\mathbf{1}}{\mathbf{120}}\epsilon_2 + \frac{\mathbf{1}}{\mathbf{36}}\epsilon_3 + \frac{1}{40}x_1 - \frac{1}{120}x_2 - \frac{1}{36}x_3 \\ x_5 = -\frac{\mathbf{11}}{\mathbf{80}}\epsilon_1 + \frac{\mathbf{7}}{\mathbf{80}}\epsilon_2 - \frac{\mathbf{1}}{\mathbf{16}}\epsilon_3 + \frac{11}{80}x_1 - \frac{7}{80}x_2 + \frac{1}{16}x_3 \\ x_6 = +\frac{\mathbf{1}}{\mathbf{80}}\epsilon_1 - \frac{\mathbf{11}}{\mathbf{240}}\epsilon_2 + \frac{\mathbf{5}}{\mathbf{144}}\epsilon_3 - \frac{1}{80}x_1 + \frac{11}{240}x_2 - \frac{5}{144}x_3 \\ z = +\frac{\mathbf{2}}{\mathbf{5}}\epsilon_1 - \frac{\mathbf{4}}{\mathbf{5}}\epsilon_2 + \frac{\mathbf{8}}{\mathbf{3}}\epsilon_3 - \frac{2}{5}x_1 + \frac{4}{5}x_2 - \frac{8}{3}x_3 \end{array} \begin{array}{l} (13) : -\frac{1}{40}\epsilon_1 + \frac{1}{120}\epsilon_2 + \frac{1}{36}\epsilon_3 > 0 \\ (14) : -\frac{11}{80}\epsilon_1 + \frac{7}{80}\epsilon_2 - \frac{1}{16}\epsilon_3 > 0 \\ (15) : +\frac{1}{80}\epsilon_1 - \frac{11}{240}\epsilon_2 + \frac{5}{144}\epsilon_3 > 0 \end{array}$$

Pivot x_2 with x_4:

$$\begin{array}{l} x_2 = -\mathbf{3}\epsilon_1 + \epsilon_2 + \frac{\mathbf{10}}{\mathbf{3}}\epsilon_3 + 3x_1 - \frac{10}{3}x_3 - 120x_4 \\ x_5 = -\frac{\mathbf{1}}{\mathbf{8}}\epsilon_1 - \frac{\mathbf{17}}{\mathbf{48}}\epsilon_3 - \frac{1}{8}x_1 - \frac{17}{48}x_3 + \frac{21}{2}x_4 \\ x_6 = -\frac{\mathbf{1}}{\mathbf{8}}\epsilon_1 + \frac{\mathbf{3}}{\mathbf{16}}\epsilon_3 + \frac{1}{8}x_1 - \frac{3}{16}x_3 - \frac{11}{2}x_4 \\ z = -\mathbf{2}\epsilon_1 + \frac{\mathbf{16}}{\mathbf{3}}\epsilon_3 + 2x_1 - \frac{16}{3}x_3 - 96x_4 \end{array} \longrightarrow \begin{array}{l} (16) : -3\epsilon_1 + \epsilon_2 + \frac{10}{3}\epsilon_3 > 0 \\ (17) : \frac{1}{8}\epsilon_1 - \frac{17}{48}\epsilon_3 > 0 \\ (18) : -\frac{1}{8}\epsilon_1 + \frac{3}{16}\epsilon_3 > 0 \end{array}$$

Pivot x_1 with x_5:

$$
\begin{array}{l}
x_1 = +\epsilon_1 - \frac{17}{6}\epsilon_3 + \frac{17}{6}x_3 + 84x_4 - 8x_5 \\
x_2 = +\epsilon_2 - \frac{31}{6}\epsilon_3 + \frac{31}{6}x_3 + 132x_4 - 24x_5 \\
x_6 = -\frac{1}{6}\epsilon_3 + \frac{1}{6}x_3 + 5x_4 - x_5 \\
z = -\frac{1}{3}\epsilon_3 + \frac{1}{3}x_3 + 72x_4 - 16x_5
\end{array}
\quad \longrightarrow \quad
\begin{array}{l}
(19) : \epsilon_1 - \frac{17}{6}\epsilon_3 > 0 \\
(20) : \epsilon_2 - \frac{31}{6}\epsilon_3 > 0 \\
(21) : -\frac{1}{6}\epsilon_3 > 0
\end{array}
$$

Alas we cannot find satisfying values for ϵ_1, ϵ_2 and ϵ_3 as constraint (3) indicates $\epsilon_3 > 0$ and constraint (21) requires $\epsilon_3 < 0$. The first and last dictionary cannot both be part of a nondegenerate sequence, so we cannot perturb this input so that the sequence of bases following Bland's rule is nondegenerate. □

This example shows that Bland's rule is not a perturbation scheme when $d \geq 3$, and leaves open the possibility that a degenerate sequence might have more than $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ bases. Algorithm 3.1 also has other applications (see Chapter 6).

3.2.2 $D(2,n) = n$ and $D_{Bl}(3,n) = O(n^2)$

In this section we apply our geometric interpretation of dictionary coefficients (recall Theorem 1.6 of Section 1.2.5) to improve upper bounds on $D_{Bl}(d,n)$.

The feasible region for the linear program,

$$
\begin{array}{l}
\text{maximize } c_1 x_2 + c_2 x_2 \\
\text{subject to } a_{i1} x_1 + a_{i2} x_2 \leq 0 \text{ for } i = 1, ..., n,
\end{array}
\tag{3.6}
$$

consists of a single vertex at the origin with up to $\binom{n}{2}$ bases if the feasible region is bounded and cone with vertex at the origin if the feasible region is unbounded. The

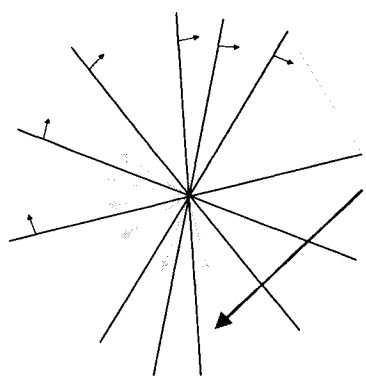


Figure 3–1: Completely degenerate polyhedron in $\Re^2$

bounding line, $a_{i1}x_1 + a_{i2}x_2 = 0$ for all n inequalities passes through the origin. We add slack variables $x_3, ..., x_{n+2}$ and compute a dictionary with basis B and cobasis $N = \{j, k\}$:

$$\begin{aligned} x_1 &= 0 - \overline{a}_{11}x_j - \overline{a}_{12}x_k \\ x_2 &= 0 - \overline{a}_{21}x_j - \overline{a}_{22}x_k \\ &\text{---------} \\ x_i &= 0 - a'_{i1}x_j - a'_{i2}x_k \text{ for all } i \in B \setminus \{1, 2\} \\ z &= 0 + c'_j x_j + c'_k x_k. \end{aligned} \tag{3.7}$$

Each slack variable x_i for $i = 3, ..., n + 2$ corresponds to bounding line $i - 2$. A dictionary with cobasic variables x_j and x_k corresponds to the intersection of line $j - 2$ and line $k - 2$. A pivot corresponds to swapping one of the two intersecting lines for a line corresponding to basic variable. In dictionary (3.7), if $c'_j > 0$ and $-a'_{ij} < 0$ then the pivot $B' = B - \{i\} + \{j\}$ ($N' = N - \{j\} + \{i\}$) is a simplex method pivot. Let $a^i = \binom{a_{i1}}{a_{i2}}$ be the gradient vector of inequality i, and let $c = \binom{c_1}{c_2}$, then the dual of (3.6), the set of constraints

$$y_1 a^1 + y_2 a^2 + ... + y_n a^n = c \tag{3.8}$$

$$y_1, y_2, ..., y_n \geq 0, \tag{3.9}$$

specifies that the objective gradient c be a nonnegative combination of the gradients a^i of the primal inequalities. The duality theorem for linear programming states that, given a feasible basis B and cobasis N, the linear program (3.6) is optimal if $\sum_{i \in N} y_i a^i = c$. In other words c is contained in the cone $C(a^i : i \in N)$. By Theorem 1.6 the coefficients of a dictionary (3.7) can be expressed in terms of the gradients of the original inequalities:

$$-a'_{ij} = \frac{\begin{vmatrix} a^{i-2} \\ a^{j-2} \end{vmatrix}}{\begin{vmatrix} a^{j-2} \\ a^{k-2} \end{vmatrix}}, c'_j = -\frac{\begin{vmatrix} c \\ a^{k-2} \end{vmatrix}}{\begin{vmatrix} a^{j-2} \\ a^{k-2} \end{vmatrix}}.$$

We will now show that the simplex method for a completely degenerate linear program in $\Re^2$ reduces to a problem involving n gradient vectors and cones in $\Re^2$. We use this geometry to compute new bounds for $D_{Bl}(2,n)$ and $D_{Bl}(3,n)$.

Consider n distinct two dimensional vectors a^i for $i = 1,...,n$, and a vector c, let N be a pair of indices, and let $B = \{1,...,n\} \setminus N$. Let a *vector pivot* be the exchange of vector $a^s, s \in N$ with $a^r, r \in B$: $N' = N - \{s\} + \{r\}$. For vectors u and v in the plane, let θ be the angle between u and v when sweeping clockwise from u to v. Define

$$\alpha(u,v) = \begin{cases} + & \text{if } \theta < \pi \\ - & \text{if } \theta > \pi \\ 0 & \text{if } \theta = \pi, \end{cases}$$

and call a pivot from $N = \{s,j\}$ to $N' = \{j,r\}$ an *admissible vector pivot* if

$$\alpha(a^s, a^j) \text{ and } \alpha(c, a^j) \text{ have opposite sign, } \quad \textit{and} \tag{3.10}$$

$$\alpha(a^s, a^j) \text{ and } \alpha(a^r, a^j) \text{ have opposite sign.} \tag{3.11}$$

Call $N = \{i,j\}$ an *optimal* set if c can be expressed as a nonnegative combination of a^i and a^j (c is contained in the cone $C(a^i,a^j) = \{\lambda \in \Re^2 : \lambda = y_1 a^i + y_2 a^j$, for $y_1, y_2 \geq 0\}$). Call N *terminal* when no further admissible vector pivots are possible.

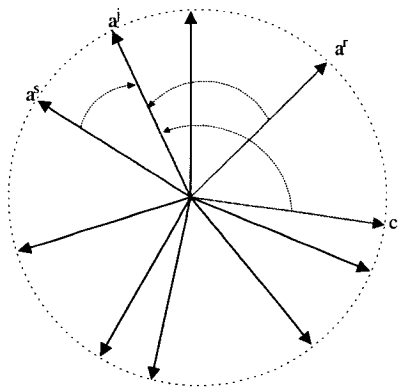


Figure 3–2: Vector sweeping in $\Re^2$

Define $G(n)$ to be the maximal number of admissible vector pivots, over all possible vector configurations, required to attain a terminal set N.

Lemma 3.1. $G(n) = n - 1$.

Proof. The n vectors a^i, for $i = 1, ..., n$, partition $\Re^2$ into n regions, one of which contains vector c. The cone $C(a^i, a^j)$ for a set $N = \{i, j\}$ spans one or more of these regions. When we perform an admissible vector pivot from a set $N^1 = \{s, j\}$ to $N^2 = \{j, r\}$ the corresponding cones $C^1(a^s, a^j)$ and $C^2(a^j, a^r)$ are adjacent, sharing vector a^j. If the pivot from N^1 to N^2 (from cone $C^1(a^s, a^j)$ to $C^2(a^j, a^r)$) is a clockwise (resp. counterclockwise) rotation in the plane, then all subsequent pivots will also be clockwise (counterclockwise), and in the worst-case we will have to pivot around 2π radians before we reach a cone that contains vector c (if such a cone exists). Let R be the set of regions covered by the sequence of cones encountered. With each pivot a new region is added to R, otherwise we have rotated around 2π radians and missed c. □

Theorem 3.2. $D_{Bl}(2, n) \leq 1 + G(n) = n$.

Proof. Consider the n gradient vectors and objective vector of a $2D$ linear program during the execution of the simplex method. A simplex pivot corresponds exactly to an admissible vector pivot with respect to the gradient vectors, the cobasis being N. If and only if $c'_j = -\frac{\left|a^{c}_{a^{k-2}}\right|}{\left|a^{j-2}_{a^{k-2}}\right|} > 0$, and $-a'_{ij} = \frac{\left|a^{i-2}_{a^{j-2}}\right|}{\left|a^{j-2}_{a^{k-2}}\right|} < 0$ are conditions (3.10) and (3.11) satisfied. The optimality and termination conditions for the simplex method and the gradient vector problem also correspond. By Lemma 3.1, $D_{Bl}(2, n) \leq 1 + G(n) = n$. □

Corollary 3.1. $D_{Bl}(2, n) = n$.

Proof. It is easy to construct nondegenerate linear programs on which the simplex method with Bland's rule visits n bases: Take n distinct points along a semicircular arc such that 2 points p_1 and p_n lie on the line that passes through the diameter

of the arc. Now label the remaining points in a clockwise manner from p_1 to p_n. The n points define an $n-$gon P. Consider the inequality description of P. Let the inequality defining edge (p_1, p_n) have index 1 and correspond to slack variable x_3. Let the inequality defining edge (p_{i-1}, p_i) have index i and correspond to slack variable x_{i+2}. Choose the objective function direction to be the vector from p_1 to p_n. Now the simplex method with Bland's rule starting at vertex p_1, cobasis $\{3, 4\}$, will leave inequality 1 (enter variable x_3), perform a ratio test and pivot to $\{4, 5\}$, then leave inequality 2, perform a ratio test and pivot to $\{5, 6\}$, then to $\{6, 7\}$ etc, ending at cobasis $\{n+1, n+2\}$ (vertex p_n). For each pivot, the ratio test

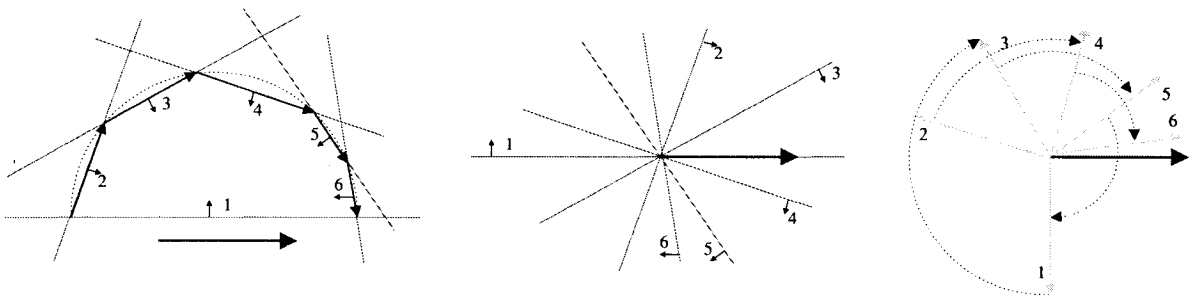


Figure 3–3: $D_{Bl}(2, n) = n$: example with $n = 6$

happens to select the basic variable with least-index. Hence if we set the right hand side of each inequality to zero (collapse the problem), then the linear program will be completely degenerate and the simplex method with Bland's rule, starting at cobasis $\{3, 4\}$, will follow the same path choosing the leaving variable with least-index each time - but this time because it breaks ratio test ties by selecting the variable with the lowest index. □

Corollary 3.2. *When $d = 2$ Bland's rule is equivalent to a perturbation scheme.*

Proof. Let $N_1, N_2, ..., N_m$ be the sequence of cobases that the simplex method with Bland's rule visits on a completely degenerate linear program

$$\begin{aligned} &\text{maximize } c_1x_2 + c_2x_2 \\ &\text{subject to } (i) : a_{i1}x_1 + a_{i2}x_2 \leq 0 \text{ for } i = 1, ..., n. \end{aligned} \tag{3.12}$$

If $\{i+2\} \in \cup_{j=1,...,m} N_j$, then let $b_i = \sqrt{(a_{i1})^2 + (a_{i2})^2}$, otherwise let $b_i = M\sqrt{(a_{i1})^2 + (a_{i2})^2}$ for some large positive constant M. Let the perturbed linear program be

$$\begin{aligned} &\text{maximize } c_1x_2 + c_2x_2 \\ &\text{subject to } (i) : a_{i1}x_1 + a_{i2}x_2 \leq b_i \text{ for } i = 1, ..., n. \end{aligned} \tag{3.13}$$

Each bounding line will have a distance of 1 or M from the origin. The linear program is nondegenerate as each bounding line is tangent to either the unit circle or circle with radius M. Consider the set of gradient vectors $a^i = \binom{a_{i1}}{a_{i2}}$ for

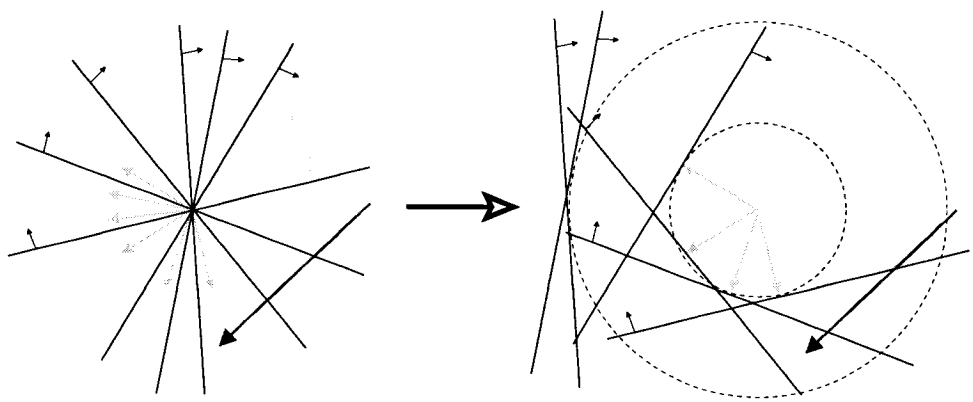


Figure 3–4: Example perturbation

$\{i+2\} \in \cup_{j=1,...,m} N_j$ and the sequence of cobases $N'_1, ..., N'_{\overline{m}}$ that simplex method with Bland's rule visits on (3.13), starting at cobasis $N'_1 = N_1$. With every pivot we exchange gradient vectors, rotating clockwise (or counterclockwise). Since each pivot is nondegenerate, during the sequence of pivots every gradient vector a^i for $i + 2 \in \cup_{j=1,...,m} N_j$ must be encountered in a clockwise (or counterclockwise)

order. Thus the nondegenerate sequence $N'_1, ..., N'_{\overline{m}}$ is equivalent to the degenerate sequence $N_1, N_2, ..., N_m$. □

We will prove that $D_{Bl}(3,n) = O(n^2)$, but first we need to translate properties of Bland's least-index rule to the gradient vectors interpretation of the simplex method.

Definition 3.1 (Least-Index Vector Pivot). *A vector pivot from $N = \{s, j\}$ to $N' = \{j, r\}$ is a least-index vector pivot if a^s has the smallest index s satisfying condition (3.10) and if a^r has the smallest index satisfying condition (3.11).*

Definition 3.2 (Least-Index Cones). *The cone $C(a^s, a^j)$ has the least-index property if*

(1) *there does not exist an admissible vector pivot from $N = \{s, j\}$ to $N' = \{j, r\}$ such that $r < j$, and*

(2) *there does not exist a gradient vector $a^r \in C(a^s, a^j)$ with $r < s$ or $r < j$.*

Proposition 3.3. *After at most two least-index pivots a least-index cone is reached.*

(The proof follows from the first item of Definition 3.2.)

Proposition 3.4. *No two least-index cones can overlap. (There are up to n least-index cones.)*

(The proof follows from the second item of Definition 3.2.)

Corollary 3.3. $D_{Bl}(3,n) = O(n^2)$.

Proof. Assume that $D_{Bl}(3,n) = \Omega(n^3)$ and consider the the sequence of bases for the worst-case example attaining the bound. A dictionary will have the form,

$$\begin{aligned} x_i &= 0 - a'_{ij}x_j - a'_{ik}x_k - a'_{il}x_l \text{ for } i \in B \\ z &= 0 + c'_j x_j + c'_k x_k + c'_l x_l. \end{aligned} \tag{3.14}$$

A variable x_s can only stay in the cobasis for a sequence of $O(n)$ bases as $D_{Bl}(2,n) = n$ by Theorem 3.2. The only way to attain a sequence of $\Omega(n^3)$

bases is for some variable x_s to enter and leave the cobasis $\Theta(n^2)$ times and each time remain in the cobasis for $\Theta(n)$ pivots. Consider the geometry of the gradient vectors of the linear program when x_s is cobasic (project to the plane $x_s = 0$). By Theorem 3.2, when x_s is cobasic then only $n - 1$ pivots are possible before optimality or before x_s needs to leave the cobasis. Let $C_{Bl}(x_s)$ be the set of least-index cones when x_s is cobasic (on the plane $x_s = 0$). Let $C_{Bl}^{t-1}(x_s)$ be the set of cones visited by Bland's rule before the t^{th} time x_s is cobasic ($|C_{Bl}^{0}(x_s)| = 1$). By Proposition 3.3, when x_s becomes cobasic for the t^{th} time at most 2 pivots are possible before a cone of $C_{Bl}(x_s)$ is encountered. Each time that x_s becomes cobasic at most 2 new bases (cones) can be encountered before a basis corresponding to a cone of $C_{Bl}(x_s)$ is reached. $|C_{Bl}(x_s)| \leq n$ by Proposition 3.4, and so the total number of bases possible when x_s is cobasic is $\Theta(n)$. □

A completely degenerate linear program has $\Theta(n^3)$ bases, but by Corollary 3.3, $D_{Bl}(3, n) = O(n^2)$.

Open Problem 3.1. *Prove that $D_{Bl}(3, n) = O(n)$, specifically $D_{Bl}(3, n) \leq 2n - 4$.*

In Chapter 5 we will continue our study on the stalling behaviour of Bland's least-index rule. We will construct families of examples on which the simplex method with Bland's rule stalls for a really long time. But first we interrupt to analyze the worst-case of a related pivot method, the least-index criss-cross method. The reason being that the next chapter also provides a gentler introduction to some key concepts that we will employ in Chapter 5.

CHAPTER 4
The Longest Criss-Cross Method Pivot Path

(The Criss-Cross Method can take $\Omega(n^d)$ Pivots)

4.1 Introduction

The lack of success with simplex methods, with respect to finding a *strongly polynomial* algorithm that solves linear programming, lead researchers to study *criss-cross methods* which leave the boundary of the polytope and traverse the edges of the underlying oriented hyperplane arrangement using *admissible pivots.* In 1999, Fukuda and Terlaky [37] proved the existence of a short (linear in n and d) admissible pivot path between any two vertices of a polytope, motivating the research community with the prospect of finding a polynomial criss-cross method. But how long can an admissible pivot path be? What is the maximal number of vertices, $C(d, n)$, along a path taken by the *least-index criss-cross method* to solve a linear program in dimension d with n inequality constraints? Clearly

$$C(d, n) \leq \binom{n}{d} \leq n^d \quad \text{for } n \geq d \tag{4.1}$$

since the maximal number of vertices of an arrangement is $\binom{n}{d}$. In 1978, even before the birth of the least-index criss-cross method, Avis and Chvátal [5] unknowingly proved its exponential worst-case behaviour by exhibiting an example where the

number of pivots taken by the simplex method with Bland's rule on a completely degenerate polytope is bounded from below by the d^{th} Fibonacci number which is of the order $(1.618\cdots)^d$.

$$C(d,n) \geq \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^d - \left(\frac{1-\sqrt{5}}{2}\right)^d\right) \quad \text{for } n \geq 2d.$$

The result follows from the observation that the least-index criss-cross method and the simplex method with Bland's rule follow the same pivot path on a completely degenerate polytope. In 1990, Roos [82] constructed an example where the least-index criss-cross method follows the boundary of a nondegenerate polytope and requires an exponential number of pivots[1] . Until now, Roos' result provided the best known lower bound,

$$C(d,n) \geq 2^d \quad \text{for } n \geq 2d, \tag{4.2}$$

which, asymptotically, leaves a significant gap with the upper bound,

$$C(d,n) \text{ is } \Omega(2^d) \text{ and } O(n^d) \quad \text{for } n \geq 2d \text{ where } d \text{ is fixed.}$$

In fact, it remained unclear whether the criss-cross method could take a path of length longer than $M(d,n)$ which is $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ for fixed d by the Upper Bound Theorem (McMullen, [72]).

$$C(d,n) \stackrel{?}{\geq} M(d,n) \tag{4.3}$$

In the present chapter we show how to construct a family of examples which not only answer this question affirmatively, but also show that $C(d,n)$ is $\Omega(n^d)$ for fixed d, implying that criss-cross methods can visit nearly every vertex of

[1] Both the Avis-Chvátal and Roos constructions are variants of the *Klee-Minty* examples [61].

the arrangement, and thus can perform even worse than simplex methods on nondegenerate input.

Theorem 4.1 (Main Theorem). *For fixed d, the function $C(d,n)$ grows like a polynomial of degree d in n:*

$$C(d,n) \text{ is } \Theta(n^d) \text{ for } n \geq 2d. \tag{4.4}$$

Our construction uses the powerful tool of a *deformed product of arrangements*, an extension of a *deformed product of polytopes* as defined recently by Amenta and Ziegler [2]. We begin by giving a geometric interpretation of the least-index criss-cross method. In Section 4.3 we define deformed products of arrangements and examine the behaviour of the least-index criss-cross method on them. By section 4.4 we are ready to construct a family of worst-case examples proving Theorem 4.1.

4.2 The Geometric Interpretation of the Criss-Cross Method

For the remainder of this chapter we consider linear programs, the problem of maximizing a linear functional φ over a polyhedron $P \subseteq \Re^d$:

$$\max \varphi(x) : x \in P, \tag{4.5}$$

where P is non-empty, P is simple, and that φ is bounded on P. The inequalities of P are a finite set of oriented hyperplanes H which together define an arrangement $A_H \subseteq \Re^d$. Let $v_{\min}$ and $v_{\max}$ be the vertices of A_H that minimize resp. maximize φ with $0 \leq \varphi(v_{\min}) \leq \varphi(v_{\max}) \leq 1$, then we write $\varphi(vert(A_H)) \subseteq [0,1]$. We will assume that the hyperplanes are labeled as $H = \{h_1, h_2, \ldots, h_n\}$. Starting at a vertex defined by a set H of d intersecting hyperplanes, a *pivot* is the operation of exchanging a hyperplane $h_i \in H$ for a hyperplane $h_j \notin H$ that intersects the edge $H \backslash \{h_i\}$. This results in a second vertex defined by $H' := H \backslash \{h_i\} \cup \{h_j\}$ (henceforth abbreviated by $H - h_i + h_j$).

Definition 4.1 (Increasing Edges). *For a linear functional $\varphi : \Re^d \rightarrow \Re$, an edge $[v', v'']$ of $A(H)$ is $\varphi-$increasing if $\varphi(v'') > \varphi(v')$.*

Definition 4.2 (Increasing Rays). *Given a start point $s \in \Re^d$ and a vector $\overrightarrow{u} \in \Re^d$, a ray $r = (s, \overrightarrow{u})$ is the set of all points of the form $s + \lambda\overrightarrow{u}$ for all scalar $\lambda \geq 0$. For a linear functional $\varphi : \Re^d \rightarrow \Re$, a ray is $\varphi-$increasing if and only if $\varphi(\overrightarrow{u}) > 0$.*

Definition 4.3 (Primal Infeasible Vertex). *A vertex p_i of the arrangement induced by a linear program (P, φ) is primal infeasible if p_i is not a vertex of the polytope P. In other words, p_i violates at least one inequality of P.*

Definition 4.4 (Dual Infeasible Vertex). *A vertex p_i, defined by a set H of d intersecting hyperplanes of the arrangement induced by a linear program (P, φ), is dual infeasible if there is at least one ray starting at p_i that is $\varphi-$increasing and satisfies all $h \in H$ (every point on the ray lies on the nonnegative side of all hyperplanes $h \in H$).*

The *optimal* vertex which maximizes φ is a vertex that is both primal and dual feasible. *Criss-cross methods* are pivot methods for solving a linear program (P, φ) whose pivot path can leave the boundary of P. The first criss-cross method was baptized by Zoints [103], and the first finite criss-cross method, the *least-index criss-cross method*, was discovered independently by Terlaky [90], and Wang [95]. As the name suggests, criss-cross methods have two types of pivots (with respect to an objective function φ): *admissible type I* pivots and *admissible type II* pivots.

Definition 4.5 (Admissible Type I Pivot). *For every primal infeasible vertex p defined by a set H of d intersecting hyperplanes, there exists an oriented hyperplane $h_j \notin H$ that is violated at p. A pivot from p to vertex p', defined by $H' := H - h_i + h_j$, is an admissible type I pivot if p' lies on the nonnegative side of h_i.*

If h_j is selected such that j is minimized, followed by selecting h_i to minimize i, then the pivot is a *least-index admissible type I* pivot.

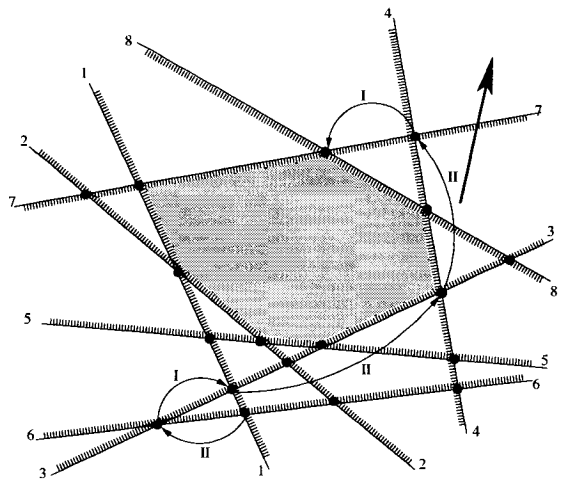


Figure 4–1: The least-index criss-cross method

Definition 4.6 (Admissible Type II Pivot). *For every dual infeasible vertex p defined by a set H of d intersecting hyperplanes, there exists a $\varphi-$increasing ray $(p, \overrightarrow{u})$ that lies on an edge $H\backslash\{h_i\}$ that is on the nonnegative side of $h_i \in H$. A pivot from p to vertex p', defined by $H' := H - h_i + h_j$, is an admissible type II pivot if there exists a point on $(p, \overrightarrow{u})$ that lies on the nonpositive side of h_j.*

If h_i is selected such that i is minimized, followed by selecting h_j to minimize j, then the pivot is a *least-index admissible type II* pivot.

We will denote a pivot, exchanging h_i for h_j, by $pivot(i, j)$. Using these notions, we provide the geometric interpretation of the least-index criss-cross method. The reader can follow the algorithm on the example shown in Figure 4–1.

Algorithm 4.1 (The Least-Index Criss-Cross Method). *Given a linear program $(P, \varphi) \subseteq \Re^d$, a linear ordering of the inequalities of P, and a vertex p of A_P:*

Criss-Cross:

If p is optimal (both primal feasible and dual feasible) then ***Stop;***

If p is primal feasible then let $f := +\infty$. Otherwise let $f := j$ such that $pivot(i, j)$ is the least-index admissible type I pivot from p to p'. If no pivot exists then the linear program is primal inconsistent, ***Stop;***

If p is dual feasible then let $g := +\infty$. Otherwise let $g := i'$ such that pivot(i', j') is the least-index admissible type II pivot from p to p''. If no pivot exists then the linear program is dual inconsistent, ***Stop;***

If $f < g$, then $\overline{p} := p'$. Otherwise $\overline{p} := p''$.

Pivot from p to $\overline{p}$, let $p := \overline{p}$ and go to ***Criss-Cross;***

The least-index criss-cross method is finite and solves a linear program (see [34] for simple proofs). From this point forward the criss-cross method will refer to the least-index criss-cross method.

4.3 Deformed Product of Arrangements

Our goal is to construct a family of deformed product programs on which the criss-cross method visits almost all vertices of the arrangement. We begin by analyzing the behaviour of the criss-cross method on the arrangement of hyperplanes of a deformed product program. (Deformed products of polytopes and deformed product programs are defined in Section 1.2.3, but for sake of clarity we reproduce the definitions here too.)

Definition 4.7 (Deformed Products of Polytopes [2]). *For a $d-$polytope P, $\varphi : P \to \Re$ a linear function with $\varphi(P) \subseteq [0,1]$, $e-$polytopes V and W, the deformed product of (P, φ) and of (V, W) is*

$$(P,\varphi) \bowtie (V,W) := \left\{ \begin{pmatrix} x \\ v + \varphi(x)(w-v) \end{pmatrix} : \quad \begin{matrix} x \in P \\ v \in V, w \in W \end{matrix} \right\} \subseteq \Re^{d+e}.$$

Definition 4.8 (Deformed Product Programs). *For $\alpha : \Re^e \to \Re$,*

$$\max \widehat{\alpha} \begin{pmatrix} x \\ u \end{pmatrix} = \alpha(u) : \quad \begin{pmatrix} x \\ u \end{pmatrix} \in Q = (P,\varphi) \bowtie (V,W), \tag{4.6}$$

where $\widehat{\alpha} : \Re^{d+e} \to \Re$.

We define the *deformed product of arrangements* to be the induced hyperplane arrangement of a deformed product of polytopes. We extend some of the facts about deformed products that are proved in [2] to express the properties of the

induced hyperplane arrangement of Q. We facilitate understanding by providing an example (see Figure 4–2) on which the reader is encouraged to verify the theorem's statements.

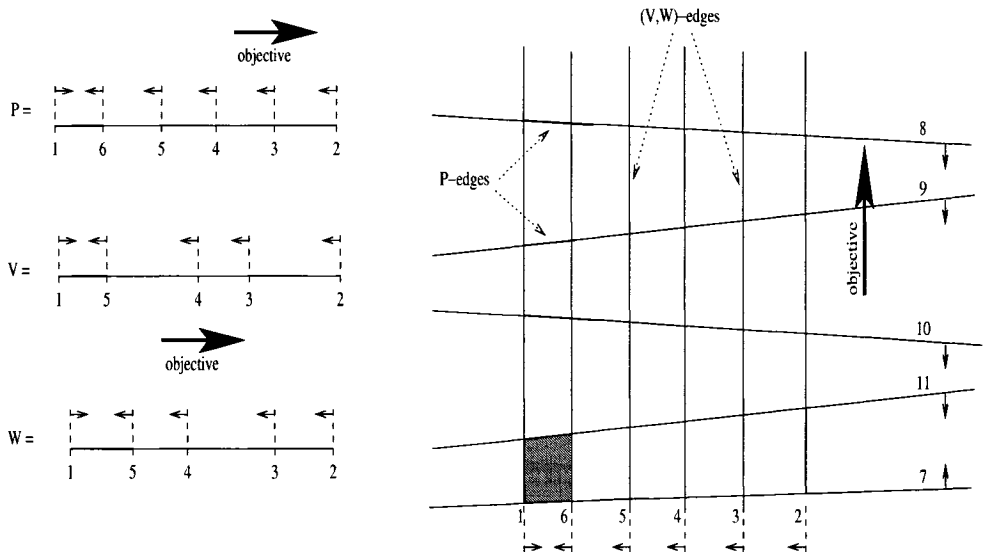


Figure 4–2: Deformed product of arrangements

Definition 4.9 (Combinatorially Equivalent Arrangements). *Two arrangements are combinatorially equivalent if the two sets of sign vectors of cells of the arrangements are exactly the same (i.e. the underlying oriented matroids are exactly the same).*

Definition 4.10 (Normally Equivalent Arrangements). *Two hyperplane arrangements are said to be normally equivalent if they are combinatorially equivalent and the corresponding unit hyperplane normals coincide.*

Theorem 4.2. *Let $P \subseteq \Re^d$ be a $d-$polyhedron, A_P be the underlying hyperplane arrangement of P, $\varphi : \Re^d \to \Re$ a linear function such that $\varphi(vert(A_P)) \subseteq [0,1]$, and $V, W \subseteq \Re^e$ be normally equivalent $e-$polyhedra inducing normally equivalent hyperplane arrangements A_V and A_W, then:*

- *If A_P has m vertices and s hyperplanes, and if A_V and A_W have n vertices and t hyperplanes each, then $Q := (P, \varphi) \bowtie (V, W)$ is a $(d+e)-$polytope*

whose underlying arrangement A_Q has at least $m \cdot n$ vertices and exactly $s + t$ hyperplanes.

- *Specifically, if $\{p_1, ..., p_m\}$ are vertices of A_P, $\{v_1, ..., v_n\}$ and $\{w_1, ..., w_n\}$ the vertices of A_V resp. A_W, then we can define $m \cdot n$ of the vertices of Q, denoted $\gamma(i, j)$, as:*

$$\gamma(i,j) = \begin{pmatrix} p_i \\ v_j + \varphi(p_i)(w_j - v_j) \end{pmatrix} : \quad \begin{matrix} 1 \le i \le m \\ 1 \le j \le n. \end{matrix} \tag{4.7}$$

- *If A_P, A_V, and A_W are given by*

$$\begin{aligned} A_P &= \{a_k x \le \alpha_k \text{ for } 1 \le k \le s\}, \\ A_V &= \{b_l u \le \beta_l \text{ for } 1 \le l \le t\}, \text{ and} \\ A_W &= \{b_l u \le \beta'_l \text{ for } 1 \le l \le t\}, \end{aligned} \tag{4.8}$$

then the arrangement of hyperplanes of the deformed product Q is given by

$$A_Q = \left\{ \begin{matrix} a_k x \le \alpha_k \text{ for } 1 \le k \le s \\ (\beta_l - \beta'_l)\varphi(x) + b_l u \le \beta_l \text{ for } 1 \le l \le t \end{matrix} \right\}. \tag{4.9}$$

- *A cell C_Q^i of A_Q is the deformed product of some cell C_P^j with (C_V^k, C_W^k): $C_Q^i = (C_P^j, \varphi) \bowtie (C_V^k, C_W^k)$. C_Q^i is convex if $0 \le \varphi(y) \le 1$ for $y \in C_Q^i$.*

Let's examine the edges of a hyperplane arrangement underlying a deformed product program with the assumptions of Theorem 4.2. Specifically, we are interested in these edges of A_Q:

- *A_P-edges* of the form $[\gamma(i', j), \gamma(i'', j)]$ for $1 \le j \le n$ and for any $1 \le i', i'' \le m$ such that $p_{i'}$ and $p_{i''}$ are adjacent vertices of A_P and
- *$A_{(V,W)}-$edges* of the form $[\gamma(i, j'), \gamma(i, j'')]$ for $1 \le i \le m$ and for any $1 \le j', j'' \le n$ such that $v_{j'}$ and $v_{j''}$ are adjacent vertices of A_V (equivalently, $w_{j'}$ and $w_{j''}$ are adjacent vertices of A_W).

Proposition 4.1. *Given a deformed product program (4.6), an A_P-edge $[\gamma(i', j), \gamma(i'', j)]$ is $\widehat{\alpha}-$increasing if and only if either $[p_{i'}, p_{i''}]$ is $\varphi-$increasing and $\alpha(w_j) > \alpha(v_j)$, or $[p', p'']$ is $\varphi-$decreasing and $\alpha(w_j) < \alpha(v_j)$.*

Proposition 4.2. *A $(V,W)-$edge $[\gamma(i,j'),\gamma(i,j'')]$ is $\widehat{\alpha}-$increasing if and only if $[v_{j'},v_{j''}]$ is $\alpha-$increasing.*

The proofs of the preceeding statements follow naturally from the proofs given in [2] for deformed products of polytopes (some aspects are also proved in Section 5.1.1). We are now ready to analyze the behaviour of the criss-cross method on deformed product programs:

Corollary 4.1. *Let p_1 and p_l be the vertices of P that minimize respectively maximize φ with $0 \leq \varphi(p_1) \leq \varphi(p_l) \leq 1$. Construct the deformed product program Q as defined in (4.6). If we number the inequalities of Q such that the inequalities of P get smaller indices than the inequalities corresponding to (V,W), then the criss-cross method prefers to pivot along A_P-edges rather than $A_{(V,W)}-$edges.*

The result is that if the criss-cross method on (P,φ) for the objective function φ takes a path of length l from the p_1 to p_l, and for $-\varphi$ takes a path of length l' from p_l to p_1, then for $(Q,\widehat{\alpha})$ the criss-cross method will follow a path of length l from $\gamma(1,j)$ to $\gamma(l,j)$ if $\alpha(v_j) < \alpha(w_j)$, and a path of length l' from $\gamma(l,j)$ to $\gamma(1,j)$ if $\alpha(v_j) > \alpha(w_j)$.

4.4 The Construction

We construct the worst-case example by first building low dimensional examples where the criss-cross method takes many pivots. We then show how to take deformed products of these base cases to construct linear programs in any dimension for which the criss-cross method behaves badly.

Definition 4.11. *Let $C(d,n)$ be the maximal number of (arrangement) vertices along a path taken by the least-index criss-cross method for some linear objective function φ on a $d-$dimensional polyhedron with at most n facets.*

Definition 4.12. *Starting at the vertex of P that minimizes φ, let $H(d,n)$ be the maximal number of (arrangement) vertices visited along a path taken by the least-index criss-cross method for some linear objective function max φ on the*

arrangement of hyperplanes induced by a d−dimensional polyhedron with at most n facets.

Clearly $C(d,n) \geq H(d,n)$.

Lemma 4.1. *For* $n \geq 2$, $H(1,n) = n$

Proof. Consider the following linear program defined by $max\ \varphi = x$ and the polytope given by the inequalities (indexed in order of appearance): $x \geq 0$ and $x \leq (n-i+1)\lambda$ for $2 \leq i \leq n$ and for some constant $\lambda > 0$. For example, for $n = 6$:

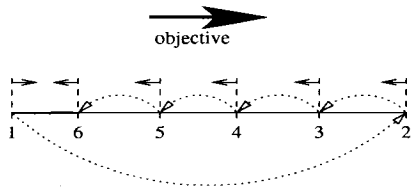


Figure 4–3: Worst-case example of the criss-cross method in one-dimension

Let v_i be the vertex defined by hyperplane h_i for $1 \leq i \leq n$. The criss-cross method takes a path of length n from vertex v_1 to vertex v_n when $\varphi = x$. The criss-cross method takes a path of length one from vertex v_n to vertex v_1 when $\varphi = -x$. □

Lemma 4.2. *There exists a pair of normally equivalent* $1-$*polytopes,* (V,W)*, defined by* k *inequalities each (hence* k *vertices), and a linear functional* α*, such that* $\alpha(v_i) > \alpha(w_i)$ *if* i *is even and* $\alpha(v_i) < \alpha(w_i)$ *if* i *is odd.*

Proof. Construct V as in Lemma 4.1. To build W, for each hyperplane h_i of V, construct h_i' of W by translating h_i in the positive $x-$direction by (some suitably small) $\epsilon > 0$ if i is odd and by $-\epsilon$ if i is even. The case when $k = 5$ in is illustrated in Figure 4–2. □

Note that $\alpha(v_k) < \alpha(w_k)$ when k is odd and $\alpha(v_k) > \alpha(w_k)$ if k is even. The following example illustrates the construction of a deformed product and the path that the criss-cross method takes on the underlying arrangement.

Example 4.1. *(See Figure 4–4) Construct P (6 inequalities, variable x_1, $\lambda = 0.1$) and V (5 inequalities, variable x_2) as in Lemma 4.1, and W (5 inequalities, variable x_2) as in Lemma 4.2. Let $Q = (P, x) \bowtie (V, W)$ and order the inequalities of Q so that the inequalities coming from P are indexed smaller than those from (V, W). Consider the path that the criss-cross method takes on the deformed product program $(Q, \alpha = x_2)$.*

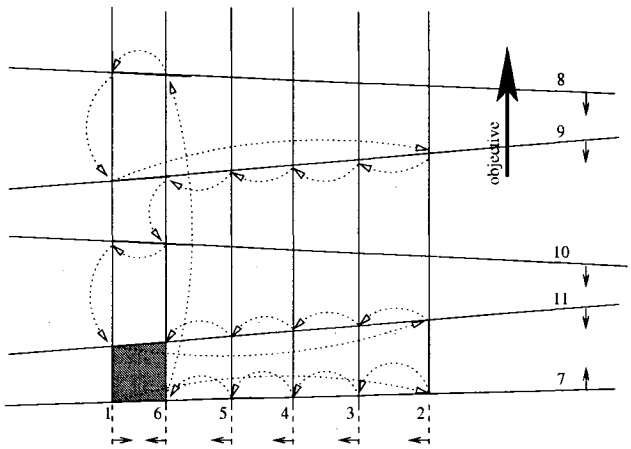


Figure 4–4: Path taken by the criss-cross method on a deformed product program

Theorem 4.3. *For $k \geq 2$ and $n > d \geq 0$,*

$$H(d+1, n+k) \geq \left\lceil \frac{k}{2} \right\rceil \cdot H(d, n) \tag{4.10}$$

Proof. Take a polytope $P \subseteq \Re^d$ with n inequalities for which the least-index criss-cross method for a linear functional $\varphi(x)$ (rescaled such that $\varphi(vert(A_P)) \subseteq [0, 1]$) follows a criss-cross method path of length $l = H(d, n)$ starting at vertex p_1 and ending at vertex p_l. Now construct the deformed product program, as in (4.6), where V, W and α are defined according to Lemma 4.2. By Corollary 4.1 we get that the criss-cross method applied to (Q, α) first follows a $P-$path with l vertices from $\gamma(1, 1)$ to $\gamma(l, 1)$, then after one $(V, W)-$pivot it follows a $P-$path of length one from $\gamma(l, 2)$ to $\gamma(1, 2)$, then after one $(V, W)-$pivot it follows a $P-$path with l vertices from $\gamma(1, 3)$ to $\gamma(l, 3)$, then after one $(V, W)-$pivot it follows a $P-$path

of length one from $\gamma(l,4)$ to $\gamma(1,4)$, etc... The complete path will visit $\lceil \frac{k}{2} \rceil l + \lfloor \frac{k}{2} \rfloor$ vertices arriving at $\gamma(1,k)$ or $\gamma(l,k)$, depending on whether k is even or odd. □

Remark 4.1. *We could use this result to construct examples, by induction, where $C(d,n)$ is $\Omega(n^d)$ asymptotically for fixed d. However we choose to postpone this analysis since iterative deformed products with the $1-$dimensional construction would contain a large number of redundant constraints, in fact $n-2d$ of them.*

Lemma 4.3. *For $n \geq 3$, $H(2,n)$ is $\Omega(n^2)$.*

Proof. Consider the following construction: let the i^{th} inequality of P be defined as

$$-(i-1)x_1 - (n-i)x_2 \leq -2(i-1)(n-i). \tag{4.11}$$

This construction ensures that the x_1 intercept of the i^{th} inequality is greater than the x_1 intercept of the $(i-1)^{th}$ while the x_2 intercept of the i^{th} inequality is less than that of the $(i-1)^{th}$ (see Figure 4–5 for an example with $n=7$). The least-index criss-cross method on the linear program $min\ \alpha = x_2$, subject to $\binom{x_1}{x_2} \in P$, starting at the vertex defined by the intersection of hyperplanes 1 and n (which we denote $(1,n)$), will take $n-1$ *type I* pivots $(1,n) \rightarrow (2,n) \rightarrow \cdots \rightarrow (n-1,n)$, and then from $(n-1,n)$ take one *type II* pivot to $(1,n-1)$, and then take $n-2$ *type I* pivots to $(n-2,n-1)$, and then one *type II* pivot to $(1,n-2)$, and then take $n-3$ *type I* pivots to $(1,n-3)$, etc... and ending with one *type II* pivot from $(2,3)$ to $(1,2)$, visiting a total of $\frac{n(n-1)}{2} = \binom{n}{2}$ vertices. □

Remark 4.2. *There are $n-2$ type II pivots, and $\frac{(n-1)(n-2)}{2}$ type I pivots. For every type I pivot from intersection (i,j) to (g,j), if i is odd then g is even, and if i is even then g is odd. Every type II pivot has the form (i,j) to $(1,i)$ where $j = i+1$.*

Lemma 4.4. *There exist normally equivalent $2-$dimensional polyhedra V and W with k facets ($k \geq 4$), and objective function $min\ \alpha$ for which the criss-cross method takes $\Theta(k^2)$ pivots such that corresponding vertices v of V and w of W, defined by*

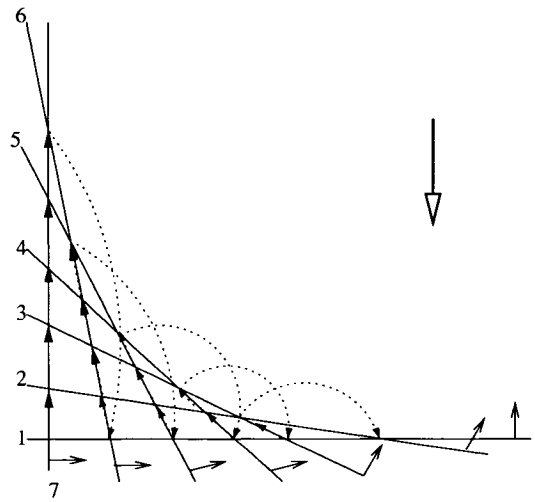


Figure 4–5: Worst-case example of the criss-cross method in two-dimensions

the intersection of hyperplanes h_i and h_j for $i < j$, have the following property: $\alpha(v) > \alpha(w)$ when i is odd, and $\alpha(v) < \alpha(w)$ when i is even.

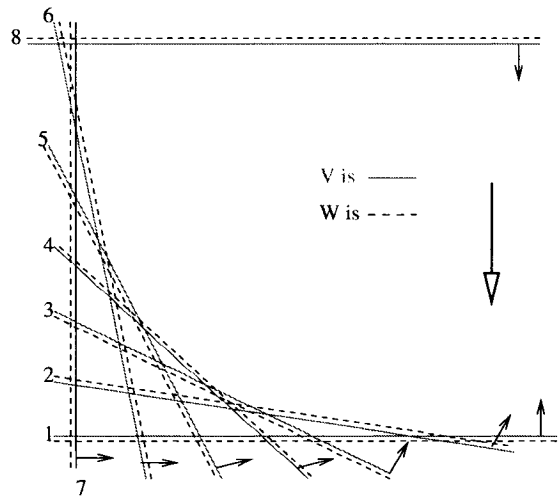


Figure 4–6: Normally equivalent worst-case arrangements

Proof. Let V be a 2−polyhedron with $k-1$ inequalities as defined in Lemma 4.3 and define the k^{th} inequality as

$$0x_1 + x_2 \leq 2(k-2).$$

Set $\alpha = x_2$. The k^{th} inequality ensures that V bounds both α and $-\alpha$, and that the x_2 intercept of the k^{th} inequality is greater than that of the $(k-2)^{th}$. Build W

as follows: for $1 \leq i \leq k-1$ take the i^{th} inequality of V, $b_i x \leq \beta$, and define the i^{th} inequality of W to be $b_i x \leq \beta'$ where $\beta' = \beta - \epsilon$ if i is odd and $\beta' = \beta + \epsilon$ if i is even (see Figure 4–6). ϵ is chosen to be positive and suitably small. Let the k^{th} inequality of W be $b_k x \leq \beta'$ where $\beta' = \beta + \epsilon$. Now let's examine corresponding vertices v of V and w of W defined by the intersection of hyperplanes h_i and h_j for $i < j$. Using basic trigonometry we can prove that if i is odd then $\alpha(v) > \alpha(w)$, and otherwise if i is even then $\alpha(v) < \alpha(w)$.

Case 1: i is odd and j odd. This case is illustrated in Figure 4–7. n_i and n_j represent the normals of h_i and h_j respectively, or if you wish the direction of translation by ϵ: $|n_i| = |n_j| = \epsilon$. Let $\delta = |d|, \theta_1 = angle(A)$, and $\theta_2 = angle(B)$. By construction, $0° \leq \theta_1 < \theta_2 \leq 90°$, and $\delta = \epsilon \sin\theta_1$ where $\sin\theta_1 \geq 0°$. Now $\alpha(w) < \alpha(v - \delta) \leq \alpha(v)$ which implies $\alpha(v) > \alpha(w)$.

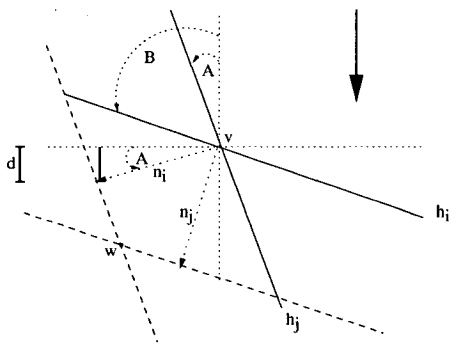


Figure 4–7: Case 1 and 2

Case 2: i is even and j even. This case is symmetric to *case 1*, hence $\alpha(v) < \alpha(w)$.

Case 3: i is odd and j even. This case is illustrated in Figure 4–8. n_i and n_j represent the normals of h_i and h_j respectively, the direction of translation by ϵ: $|n_i| = |n_j| = \epsilon$. Let $\delta = |d|, \theta_1 = angle(A)$, and $\theta_2 = angle(B)$. By construction, $0° \leq \theta_1 < \theta_2 \leq 90°$, and $\delta = \epsilon \sin(90° - \theta_2)$ where $\sin(90° - \theta_2) > 0$. Now $\alpha(w) \leq \alpha(v - \delta) < \alpha(v)$ which implies $\alpha(v) > \alpha(w)$.

Case 4: i is even and j odd. This case is symmetric to *case 3*, hence $\alpha(v) < \alpha(w)$.

□

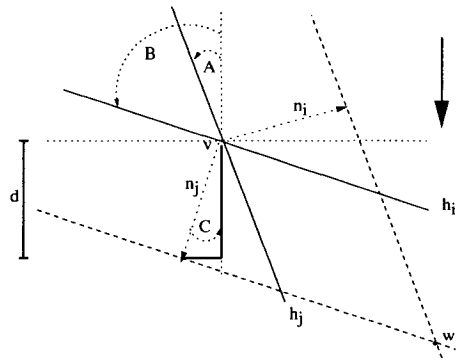


Figure 4–8: Case 3 and 4

Remark 4.3. *Starting at $(n-2, n)$ the criss-cross method on V (or W) will take one type II pivot to $(1, n)$ and then follow the path described in Lemma 4.3. There are $k-2$ type II pivots, and $\frac{(k-2)(k-3)}{2}$ type I pivots (see Remark 4.2 setting $n = k-1$ and adding one additional type II pivot from $(k-2, k)$ to $(1, k)$).*

Definition 4.13 (Switch Pivot). *Given two normally equivalent polyhedra V and W, and a linear objective function α we define a switch pivot to be a pivot from v_i to v_j (w_i to w_j) such that if $\alpha(v_i) > \alpha(w_i)$ then $\alpha(v_j) < \alpha(w_j)$, otherwise if $\alpha(v_i) < \alpha(w_i)$ then $\alpha(v_j) > \alpha(w_j)$.*

Lemma 4.5. *Let (V, W) be as defined in Lemma 4.4. Starting at the intersection of h_{k-1} and h_k on V (W) the least-index criss cross method takes a path to the intersection of h_1 and h_2 performing $\Theta(k^2)$ switch pivots.*

Proof. Every *type I* pivot is a switch pivot, and every second *type II* pivot is a switch pivot,

$$\#\text{ of switch pivots} \geq \frac{k^2}{K} = \Theta(k^2) \tag{4.12}$$

for some constant $K > 1$ and all $k \geq 3$. □

Theorem 4.4. *For $k \geq 3$, $n > d \geq 0$, and some constant $K > 1$,*

$$H(d+2, n+k) \geq \frac{k^2}{2K} \cdot H(d, n). \tag{4.13}$$

Proof. Take a polytope $P \subseteq \Re^d$ with n inequalities for which the least-index criss-cross method for a linear functional $\varphi(x)$ (rescaled such that $\varphi(vert(A_P)) \subseteq [0,1]$) follows a criss-cross method path of length $l = H(d,n)$ starting at vertex p_1 and ending at vertex p_l. Let l' be the length of the criss-cross method path from p_l to p_1 for $-\varphi$. Now construct the deformed product program, as in (4.6), where V, W and α are defined according to Lemma 4.4. Let v_{opt} be the optimal vertex of (V,W). By Corollary 4.1, we get that the criss-cross method applied to (Q,α) first follows a $P-$path with l vertices from $\gamma(1,1)$ to $\gamma(l,1)$, and then after a (V,W)-*switch pivot* it follows a $P-$path of length l', and then after a $(V,W)-$*switch pivot* it follows a $P-$path of length l, and then after a $(V,W)-$*switch pivot* it follows a $P-$path of length l', etc... The complete path will visit at least $\frac{1}{2}\frac{k^2}{K}l + \frac{1}{2}\frac{k^2}{K}l'$ vertices ending at (l, opt). □

Corollary 4.2. *For $n \geq 2d \geq 2$, $C(d,n) = \Omega((\frac{n}{d})^d)$. More specifically, for some constant $K > 1$:*

$$C(d,n) \geq \left\lfloor \frac{2n}{d\sqrt{2K}} \right\rfloor^d \qquad \text{if } d \text{ is even,} \tag{4.14}$$

and

$$C(d,n) \geq \left\lfloor \frac{2n}{(d+1)\sqrt{2K}} \right\rfloor^d \qquad \text{if } d \text{ is odd.} \tag{4.15}$$

Proof. (By induction) Let's begin with the even case, when $d = 2m$ for all $m \geq 0$, let $n = km$:

$$\begin{aligned} H(2m,km) &\geq \frac{k^2}{2K} H(2(m-1), k(m-1)) \\ &\vdots \qquad m \text{ times} \\ &\geq \left(\frac{k^2}{2K}\right)^{m-1} H(2,k) \\ &= \frac{k^{2m}}{(2K)^m} \qquad \text{by Lemma 4.3.} \end{aligned}$$

Substituting for $m = \frac{d}{2}$ and $k = \lfloor \frac{2n}{d} \rfloor$, we get

$$H(d,n) \geq \left\lfloor \frac{2n}{d\sqrt{2K}} \right\rfloor^d. \tag{4.16}$$

For the odd case, when $d = 2m+1$ for all $m \geq 0$, let $n = k(m+1)$:

$$\begin{aligned} H(2m+1, k(m+1)) &\geq \frac{k^2}{2K} H(2(m-1)+1, km) \\ &\vdots \quad m \text{ times} \\ &\geq \left(\frac{k^2}{2K}\right)^m H(1,k) \\ &= \frac{k^{2m+1}}{(2K)^m} \qquad \text{by Lemma 4.1.} \end{aligned}$$

Substituting for $m = \frac{d-1}{2}$ and $k = \lfloor \frac{2n}{d+1} \rfloor$, we get

$$H(d,n) \geq \left\lfloor \frac{2n}{(d+1)\sqrt{2K}} \right\rfloor^d. \tag{4.17}$$

The condition $n \geq 2d$ guarantees $k \geq 4$. □

Remark 4.4. *The construction has no redundant constraints when d is even, and $\lfloor \frac{2n}{d+1} \rfloor - 2$ redundant constraints when d is odd.*

Corollary 4.3 (Main Theorem). *For fixed dimension d, the function $C(d,n)$ grows like a polynomial of degree d in n:*

$$C(d,n) \text{ is } \Theta(n^d) \text{ for } n \geq 2d \text{ where } d \text{ is fixed.} \tag{4.18}$$

The least-index criss-cross method can take $\Omega(n^d)$ pivots to solve a $d-$dimensional linear program defined by n inequalities (when d is fixed). This result provides a tighter lower bound that asymptotically achieves the upperbound, and also shows that the least-index criss-cross method is worse than simplex methods employing lexicographic perturbation in the worst case. Despite this negative result, criss-cross methods remain perhaps the best hope of finding a strongly polynomial algorithm for linear programming (see [37]).

Open Problem 4.1. *Construct linear programs on which criss-cross variants [76] follow long paths.*

CHAPTER 5
Stalling the Simplex Method - Part II

Problem 5.1. *What is the lower bound on the maximal number of bases that the simplex method with Bland's rule visits on a completely degenerate linear program with n inequalities in dimension d?*

(Can we reproduce the bound achieved for the least-index criss-cross method in the previous chapter?)

In this chapter we continue our study on the worst-case behaviour of the simplex method with Bland's rule on degenerate input. We present new constructions that extend previous known bounds.

5.1 Lower Bounds on $D_{Bl}(d, n)$

Avis and Chvátal [5] constructed completely degenerate linear programs where the simplex method with Bland's rule follows a path whose length is bounded by the d^{th} Fibonacci number which is of the order $(1.618\cdots)^d$.

$$D_{Bl}(d,n) \geq \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^d - \left(\frac{1-\sqrt{5}}{2}\right)^d\right) \quad \text{for } n \geq 2d.$$

We provide two main results in this chapter. First we improve the lower bound by showing that $D_{Bl}(d,n) = \Omega(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$. Second we study linear programs with few inequalities (with respect to the dimension) and show that $D_{Bl}(d,n) = \Omega(n^{n-d})$ for $d \leq n \leq \frac{4d}{3}$. Most textbooks and implementations suggest that when $n \geq 2d$ the

dual simplex method should be used, thus the latter construction is of particular significance as it is the first general worst-case construction for the primal simplex method when $n < 2d$.

5.1.1 $D_{Bl}(d,n) = \Omega(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$

Amenta and Ziegler [2] have shown that the maximal length of a Bland rule path on a nondegenerate linear program with n inequalities in d dimensions is $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$. If we alter their construction, by defining a *collapsed deformed product construction*, we can build completely degenerate linear programs on which the simplex method with Bland's rule takes long degenerate paths, and tighten the lower bound for $D_{Bl}(d,n)$.

Definition 5.1. *Let $\varphi : \Re^d \to \Re, \alpha : \Re^e \to \Re$ be linear functions, $P \subseteq \Re^d$ be a completely degenerate $d-$polytope, and $V, W \subseteq \Re^e$ be normally equivalent $e-$polytopes given by*

$$P = \{x \in \Re^d : a_k x \leq 0 \qquad \text{for } 1 \leq k \leq s\}, \tag{5.1}$$

$$V = conv\{v_1, ..., v_n\} = \{u \in \Re^e : b_l u \leq \beta_l \qquad \text{for } 1 \leq l \leq t\}, \text{ and} \tag{5.2}$$

$$W = conv\{w_1, ..., w_n\} = \{v \in \Re^e : b_l u \leq \beta'_l \qquad \text{for } 1 \leq l \leq t\}, \tag{5.3}$$

then let the inequalities of the collapsed deformed product $\widehat{Q}$ of $Q = (P, \varphi) \bowtie (V, W) \subseteq \Re^{d+e}$ be given by

$$\widehat{Q} = \left\{ (x,u) \in \Re^{d+e} : \begin{array}{ll} a_k x \leq 0 & \text{for } 1 \leq k \leq s \\ (\beta_l - \beta'_l)\varphi(x) + b_l u \leq 0 & \text{for } 1 \leq l \leq t \end{array} \right\}. \tag{5.4}$$

Proposition 5.1. *If P has m feasible bases and is defined by s inequalities, and if V and W have n vertices and are defined by t inequalities each, then $\widehat{Q}$ is a $(d+e)-$polyhedron which has at least $n \cdot m$ feasible bases. In particular if $\{B_1^P, ..., B_m^P\}$ are feasible bases of P, $\{B_1^V, ..., B_n^V\}$ and $\{B_1^W, ..., B_n^W\}$ the feasible bases (vertices) of V resp. W, then we can define $m \cdot n$ feasible bases of $\widehat{Q}$*

(appropriately distinguishing variable indices), denoted $B^{\widehat{Q}}_{(i,j)}$, *as:*

$$B^{\widehat{Q}}_{(i,j)} = B^P_i \cup B^V_j = B^P_i \cup B^W_j \text{ for } 1 \leq i \leq m \text{ and } 1 \leq j \leq n. \tag{5.5}$$

Definition 5.2 (Basis graph $B(P)$ of a polyhedron P). *The vertices of* $B(P)$ *are bases of* P. *Two vertices of* $B(P)$ *share an edge if a single (allowable) exchange of variables separates the corresponding bases of* P *(adjacent bases).*

We will avoid using the term "vertex of $B(P)$" to avoid confusion with the the term "vertex of P". Instead we will refer to the "bases" of $B(P)$. An objective function φ directs the edges of the graph to obtain the digraph $B_\varphi(P)$. The simplex method follows paths in $B_\varphi(P)$. (The study of degeneracy graphs is relatively limited [42], [43], [44], [64], [104], [105], for a survey see Gal [41].)

Let's examine the directed basis graph $B_{\widehat{\alpha}}(\widehat{Q})$ of a collapsed deformed product program maximize $\widehat{\alpha} = \binom{0}{\alpha} : \Re^{d+e} \rightarrow \Re$ subject to $\widehat{Q} \subseteq \Re^{d+e}$. Specifically, we are interested in these edges of $B_{\widehat{\alpha}}(\widehat{Q})$:

- *$P-$edges* of the form $[B^{\widehat{Q}}_{(i',j)}, B^{\widehat{Q}}_{(i'',j)}]$ for $1 \leq j \leq n$ and for any $1 \leq i', i'' \leq m$ such that $B^P_{i'}$ and $B^P_{i''}$ are adjacent feasible bases of P and
- *$(V,W)-$edges* of the form $[B^{\widehat{Q}}_{(i,j')}, B^{\widehat{Q}}_{(i,j'')}]$ for $1 \leq i \leq m$ and for any $1 \leq j', j'' \leq n$ such that $B^V_{j'}$ and $B^V_{j''}$ are adjacent feasible bases of V (equivalently, $B^W_{j'}$ and $B^W_{j''}$ are adjacent vertices of W).

Definition 5.3. *An edge* $[B^P_i, B^P_j]$ *of a basis graph* $B_\varphi(P)$ *is* $\varphi-$*admissible if we can perform a simplex method pivot from* B^P_i *to* B^P_j.

Proposition 5.2. *A* $(V,W)-$*edge* $[B^{\widehat{Q}}_{(i,j')}, B^{\widehat{Q}}_{(i,j'')}]$ *is* $\widehat{\alpha}-$*admissible if and only if* $[B^V_{j'}, B^V_{j''}]$ *is* $\alpha-$*admissible.*

Proposition 5.3. *A* $P-$*edge* $[B^{\widehat{Q}}_{(i',j)}, B^{\widehat{Q}}_{(i'',j)}]$ *is* $\widehat{\alpha}-$*admissible if and only if either* $[B^P_{i'}, B^P_{i''}]$ *is* $\varphi-$*admissible and* $\alpha(w_j) > \alpha(v_j)$, *or* $[B^P_{i''}, B^P_{i'}]$ *is* $\varphi-$*admissible and* $\alpha(w_j) < \alpha(v_j)$.

Proof of Propositions 5.1, 5.2, and 5.3. These propositions are verifiable when we express the linear programs in dictionary form in matrix notation. After adding slack variables, linear functions over the inequality systems (5.1), (5.2), and (5.3) can be represented as

$$(P,\varphi): \quad \text{maximize } z_P = \varphi x \tag{5.6}$$
$$Ax = 0, \text{ with } x_{d+1}, ..., x_{d+s} \geq 0,$$

$$(V,\alpha): \quad \text{maximize } z_V = \alpha u \tag{5.7}$$
$$Du = \beta, \text{ with } u_{e+1}, ..., u_{e+t} \geq 0,$$

$$(W,\alpha): \quad \text{maximize } z_W = \alpha u \tag{5.8}$$
$$Du = \beta', \text{ with } u_{e+1}, ..., u_{e+t} \geq 0,$$

where $A \in \Re^{s+d}, B \in \Re^{t+e}$ are coefficient matrices, $\varphi \in \Re^s, \alpha \in \Re^t$ are row vectors, and $\beta, \beta' \in \Re^t$ are column vectors. Compute the dictionaries

for basis B^P and cobasis N^P
$$x_{B^P} = -A_{B^P}^{-1} A_{N^P} x_{N^P} \tag{5.9}$$
$$z_P = (\varphi_{N^P} - \varphi_{B^P} A_{B^P}^{-1} A_{N^P}) x_{N^P},$$

for basis B^V and cobasis N^V
$$u_{B^V} = D_{B^V}^{-1}\beta - D_{B^V}^{-1} D_{N^V} u_{N^V} \tag{5.10}$$
$$z_V = \alpha_{B^V} D_{B^V}^{-1}\beta + (\alpha_{N^V} - \alpha_{B^V} D_{B^V}^{-1} D_{N^V}) u_{N^V},$$

for basis B^W and cobasis N^W
$$u_{B^W} = D_{B^W}^{-1}\beta' - D_{B^W}^{-1} D_{N^W} u_{N^W} \tag{5.11}$$
$$z_W = \alpha_{B^W} D_{B^W}^{-1}\beta' + (\alpha_{N^W} - \alpha_{B^W} D_{B^W}^{-1} D_{N^W}) u_{N^W}.$$

Adding nonnegative slack variables $x_{d+1}, ..., x_{d+s}, u_{e+1}, ..., u_{e+t}$ to the system (5.4) we get

$$(\widehat{Q}, \widehat{\alpha}): \qquad \begin{aligned} Ax &= 0 \\ (\beta - \beta')\varphi x + Du &= 0 \\ z_{\widehat{Q}} &= \widehat{\alpha} u, \end{aligned} \tag{5.12}$$

where $A, D, \varphi, \beta, \beta'$ are as defined in (5.6),(5.7), and (5.8) with $\widehat{\alpha} = \binom{0}{\alpha} \in \Re^{s+t}$. Now let's compute a dictionary of $(\widehat{Q}, \widehat{\alpha})$ with basis $B^{\widehat{Q}} = B^P \cup B^V = B^P \cup B^W$ and cobasis $N^{\widehat{Q}} = N^P \cup N^V = N^P \cup N^W$:

$$\begin{aligned} x_{B^P} &= -A_{B^P}^{-1} A_{N^P} x_{N^P} \\ u_{B^V} &= -D_{B^V}^{-1} \left((\beta - \beta')(\varphi_{N^P} x_{N^P} + \varphi_{B^P} x_{B^P}) + D_{N^V} u_{N^V} \right) \\ z_{\widehat{Q}} &= \alpha_{B^V} u_{B^V} + \alpha_{N^V} u_{N^V}, \end{aligned} \tag{5.13}$$

and after substitutions:

$$\begin{aligned} x_{B^P} &= -A_{B^P}^{-1} A_{N^P} x_{N^P} \\ u_{B^V} &= (D_{B^V}^{-1}\beta' - D_{B^V}^{-1}\beta)(\varphi_{N^P} - \varphi_{B^P} A_{B^P}^{-1} A_{N^P}) x_{N^P} - D_{B^V}^{-1} D_{N^V} u_{N^V} \\ z_{\widehat{Q}} &= \alpha_{B^V}(D_{B^V}^{-1}\beta' - D_{B^V}^{-1}\beta)(\varphi_{N^P} - \varphi_{B^P} A_{B^P}^{-1} A_{N^P}) x_{N^P} + \left(\alpha_{N^V} - \alpha_{B^V} D_{B^V}^{-1} D_{N^V}\right) u_{N^V} \end{aligned} \tag{5.14}$$

(proving Proposition 5.1). Examining the dictionaries we see that when

- $(\varphi_{N^P} - \varphi_{B^P} A_{B^P}^{-1} A_{N^P})_j > 0$ and $(-A_{B^P}^{-1} A_{N^P})_{ij} < 0$ then the pivot $B^P = B^P - \{i\} + \{j\}$ is $\varphi-$admissible for (5.9),
- $(\alpha_{N^V} - \alpha_{B^V} D_{B^V}^{-1} D_{N^V})_j > 0$ and $(-D_{B^V}^{-1} D_{N^V})_{ij} < 0$ then the pivot $B^V = B^V - \{i\} + \{j\}$ is $\alpha-$admissible for (5.10), similarly for (5.11).

Now we can say the following about the *edges* of $B_{\widehat{\alpha}}(\widehat{Q})$:

- if and only if $(\alpha_{N^V} - \alpha_{B^V} D_{B^V}^{-1} D_{N^V})_j > 0$ and $(-D_{B^V}^{-1} D_{N^V})_{ij} < 0$ then the $((V,W)-$edge) pivot $B^{\widehat{Q}} = B^{\widehat{Q}} - \{i\} + \{j\}$ is $\widehat{\alpha}-$admissible for (5.14),

- if and only if $(\alpha_{B^V}(D_{B^V}^{-1}\beta' - D_{B^V}^{-1}\beta)(\varphi_{N^P} - \varphi_{B^P}A_{B^P}^{-1}A_{N^P}))_j > 0$ and $(-A_{B^P}^{-1}A_{N^P})_{ij} < 0$ then the ($P-$edge) pivot $B^{\widehat{Q}} = B^{\widehat{Q}} - \{i\} + \{j\}$ is $\widehat{\alpha}-$admissible for (5.14). Analyzing further, when $\alpha_{B^V}(D_{B^V}^{-1}\beta' - D_{B^V}^{-1}\beta)(\varphi_{N^P} - \varphi_{B^P}A_{B^P}^{-1}A_{N^P})_j > 0$ either

$$(\varphi_{N^P} - \varphi_{B^P}A_{B^P}^{-1}A_{N^P})_j > 0 \text{ and } \alpha_{B^V}D_{B^V}^{-1}\beta' > \alpha_{B^V}D_{B^V}^{-1}\beta$$
$$\text{or } (\varphi_{N^P} - \varphi_{B^P}A_{B^P}^{-1}A_{N^P})_j < 0 \text{ and } \alpha_{B^V}D_{B^V}^{-1}\beta' < \alpha_{B^V}D_{B^V}^{-1}\beta.$$

This proves Propositions 5.2 and 5.3, as $\alpha_{B_j^V}D_{B_j^V}^{-1}\beta' = \alpha(w_j)$ and $\alpha_{B_j^V}D_{B_j^V}^{-1}\beta = \alpha(v_j)$. □

We are now ready to analyze the behaviour of the simplex method with Bland's rule on collapsed deformed product programs. Let B_1^P and B_l^P be source and sink bases of the directed feasible basis graph of (P, φ). Construct the collapsed deformed product program $(\widehat{Q}, \widehat{\alpha})$ - a linear objective function over a collapsed deformed product. If we number the inequalities of $\widehat{Q}$ such that the inequalities of P get smaller indices than the inequalities corresponding to (V, W), then the simplex method with Bland's rule prefers to pivot along $P-$edges rather than $(V, W)-$edges. If the simplex method with Bland's rule on (P, φ) for the objective function φ takes a path of length l from the B_1^P to B_l^P, and for $-\varphi$ takes a path of length l' from B_l^P to B_1^P, then for $(\widehat{Q}, \widehat{\alpha})$ the simplex method with Bland's rule will follow a path of length l from $B_{(1,j)}^{\widehat{Q}}$ to $B_{(l,j)}^{\widehat{Q}}$ if $\alpha(w_j) > \alpha(v_j)$ and a path of length l' from $B_{(l,j)}^{\widehat{Q}}$ to $B_{(1,j)}^{\widehat{Q}}$ if $\alpha(w_j) < \alpha(v_j)$. When we collapse the deformed product construction of Amenta and Ziegler [2], we can show that the maximal length, $D_{Bl}(d, n)$, of a Bland rule pivot path on a completely degenerate linear program with n inequalities in $d-$dimensions is $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$.

Lemma 5.1 (Amenta and Ziegler [2] Lemma 4.6). *Let $\alpha(x) = 0x_1 + 1x_2$ denote a linear function on the plane (using the cartesian coordinate axes with*

$x = (x_1, x_2) \in \Re^2$). *For each* $k \geq 4$ *there exist normally equivalent* $k-$*gons*

$$V = conv\{v_1, ..., v_k\} \qquad and \qquad W = conv\{w_1, ..., w_k\}$$

(both labeled in clockwise order), such that

$$0 = \alpha(v_1) < \alpha(w_1) < \alpha(w_2) < \alpha(v_2) < \alpha(v_3) < \alpha(w_3) < \alpha(w_4) < ...$$

(The sequence ends with $... < \alpha(v_k) < \alpha(w_k) = 1$ *if* k *is odd, and with* $... < \alpha(w_k) < \alpha(v_k) = 1$ *if* k *is even.) Furthermore, there exists a labeling of the inequalities of* V *(and* W*) so that the simplex method with Bland's rule on the linear program defined by* $\alpha(x)$ *over* V *(or* W*) follows a path of length* k *starting at* v_1 *(or* w_1*) and ending at* v_k *(or* w_k*).*

Proof. Take $2k - 4 \geq 4$ points equally spaced on a semicircular arc and label them $w_2, v_2, v_3, w_3, w_4, ...$ ending with $..., v_{k-1}, w_{k-1}$ if k is odd and with $..., w_{k-1}, v_{k-1}$ if k is even. The additional points v_1, w_1, v_k, and w_k are then chosen appropriately on a line that is parallel to the diameter on which the semicircular arc was based. Figure 5–1 depicts the case when $k = 5$.

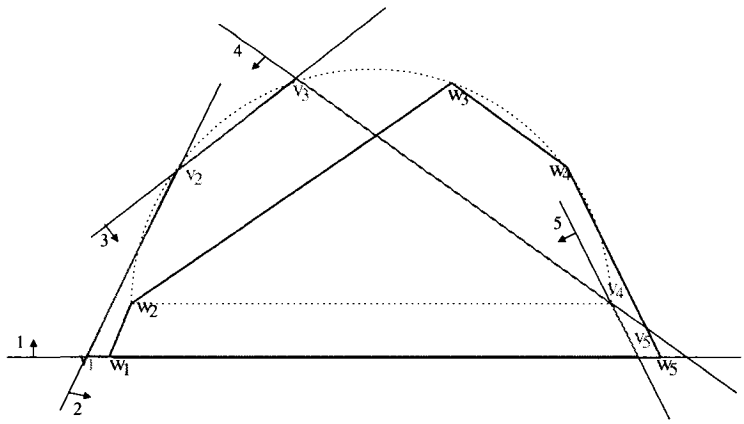


Figure 5–1: Amenta & Ziegler construction

Now consider the inequality description of V (and W), namely let the inequality defining edge (v_1, v_k) have index 1, and let the inequality defining edge

(v_{i-1}, v_i) for $i = 2, ..., k$ have index i. Starting at v_1 (the intersection of lines 1 and 2) the simplex method following Bland's rule will leave inequality 1 and follow the $\alpha-$increasing edge along line 2 to v_2. In general Bland's rule at v_i will leave line i and pivot to v_{i+1} for $i = 1, ..., k-1$, ending at v_k. □

Note that if we collapse V (or W) then the simplex method with Bland's rule will follow the same path as in the nondegenerate case from the intersection of lines 1 and 2 to the intersection of lines 1 and k. If we reverse the objective function, then Bland's rule on V (W) collapsed pivots twice: from the source basis (intersection of 1 and k) to the sink basis (intersection of 1 and 2) via the intersection of lines 2 and k.

Theorem 5.1. *For $k \geq 4$ and $n > d \geq 0$,*

$$D_{Bl}(d+2, n+k) \geq k D_{Bl}(d, n).$$

Proof. Take a completely degenerate linear program $(P, \varphi) \subseteq \Re^d$ on which the simplex method with Bland's rule follows a path of length $l := D_{Bl}(d, n)$ from source basis B_1^P to sink basis B_l^P, and path of length l' from B_l^P to B_1^P on $(P, -\varphi)$. Now construct the collapsing deformed product program $\widehat{Q}$ of $Q = (P, \varphi) \bowtie (V, W)$ where $V, W \subseteq \Re^e$ are convex $k-$gons built according to Lemma 5.1. Now Bland's rule applied to $(\widehat{Q}, \widehat{\alpha} = \alpha)$ first follows a $P-$path of length l from $B^{\widehat{Q}}_{(1,1)}$ to $B^{\widehat{Q}}_{(l,1)}$, then after one $(V, W)-$pivot to $B^{\widehat{Q}}_{(l,2)}$ it follows a $P-$path of length l' to $B^{\widehat{Q}}_{(1,2)}$, then after one $(V, W)-$pivot to $B^{\widehat{Q}}_{(1,3)}$ it follows a $P-$path of length l to $B^{\widehat{Q}}_{(l,3)}$, etc... The complete path will visit at least $\frac{k}{2}l$ bases, arriving at $B^{\widehat{Q}}_{(1,k)}$ if k is even or $B^{\widehat{Q}}_{(l,k)}$ if k is odd. □

Corollary 5.1. *For $n \geq 2d$, $D_{Bl}(d, n) = \Omega(n^{\lfloor \frac{d}{2} \rfloor})$. More specifically,*

$$D_{Bl}(d, n) \geq \left\lfloor \frac{n}{d} \right\rfloor^{\frac{d}{2}} \qquad \text{if } d \text{ is even,} \tag{5.15}$$

and

$$D_{Bl}(d,n) \geq \left\lfloor \frac{n}{(d+1)} \right\rfloor^{\lfloor \frac{d}{2} \rfloor} \qquad \text{if } d \text{ is odd.} \tag{5.16}$$

Proof. (By induction) Let's begin with the even case, when $d = 2m$ for all $m \geq 0$, let $n = km$:

$$\begin{aligned} D_{Bl}(2m, km) &\geq \frac{k}{2} D_{Bl}(2(m-1), k(m-1)) \\ &\vdots \quad m \text{ times} \\ &\geq \left(\frac{k}{2}\right)^{m-1} D_{Bl}(2,k) = \left(\frac{k}{2}\right)^{m} \qquad \text{by Lemma 5.1.} \end{aligned}$$

Substituting for $m = \frac{d}{2}$ and $k = \lfloor \frac{2n}{d} \rfloor$, we get

$$H(d,n) \geq \left\lfloor \frac{n}{d} \right\rfloor^{\frac{d}{2}}. \tag{5.17}$$

For the odd case, when $d = 2m+1$ for all $m \geq 0$, let $n = k(m+1)$:

$$\begin{aligned} D_{Bl}(2m+1, k(m+1)) &\geq \frac{k}{2} D_{Bl}(2(m-1)+1, km) \\ &\vdots \quad m \text{ times} \\ &\geq \left(\frac{k}{2}\right)^{m} D_{Bl}(1,k) = \left(\frac{k}{2}\right)^{m} \end{aligned}$$

Substituting for $m = \frac{d-1}{2}$ and $k = \lfloor \frac{2n}{d+1} \rfloor$, we get

$$D_{Bl}(d,n) \geq \left\lfloor \frac{n}{(d+1)} \right\rfloor^{\lfloor \frac{d}{2} \rfloor}. \tag{5.18}$$

(The condition $n \geq 2d$ guarantees $k \geq 4$). □

5.1.2 $D_{Bl}(d,n) = \Theta(n^{n-d})$ for $d \leq n \leq \frac{4d}{3}$

We now consider linear programs with few inequalities. Recall the geometry of the dual simplex method used in Chapter 2 to build completely degenerate linear programs on which the simplex method cycles with length $\Theta(n^{n-d})$ for $d \leq n \leq \lfloor \frac{4d}{3} \rfloor$. In particular, we note that we can relabel variables of the primal dictionary so that x_i corresponds to inequality i of the dual linear program for

$i = 1, ..., n$. A basis B of a primal dictionary corresponds to the cobasis of the dual dictionary - the intersection of bounding lines i for $i \in B$, and selecting a least-indexed line in the dual corresponds to selecting the least-indexed variable in the primal. Using the dual geometry we present a new construction where the (primal) simplex method with Bland's rule visits $\Theta(n^{n-d})$ bases for $d \leq n \leq \lfloor \frac{4d}{3} \rfloor$ with $d \geq 1$.

Lemma 5.2. *$D_{Bl}(d, n = d + 1) = \Theta(n)$ for $d \geq 2$.*

Proof. Consider the dual linear program

$$\text{maximize } 0y \tag{5.19}$$

$$\text{subject to : } \quad (i) : y \geq i \quad \text{for } i = 1, ..., k$$

with $k \geq 2$. The simplex method with Bland's rule will follow a path from $\{1\}$ to

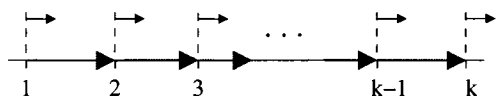


Figure 5–2: $D_{Bl}(k, k + 1) = \Theta(k)$

$\{2\}$ to $\{3\}$ ... to ...$\{k\}$ of length k. The primal linear program (dual of (5.19)) has dimension k with $k + 1$ inequalities. □

In 2001 Gärtner et al. [38] constructed examples where the simplex method (with no particular refinement) visits $\Theta(n^2)$ vertices (bases) for a linear program in dimension d with $n = d + 2$ inequalities. Our following construction duplicates their bound on completely degenerate linear programs, and in addition follows Bland's rule.

Lemma 5.3. *$D_{Bl}(d, n = d + 2) = \Theta(n^2)$ for $d \geq 5$.*

Proof. Consider the dual linear program for some integers $k \geq 2, l \geq 3$ with l odd,

$$\begin{aligned} &\text{maximize } 0y_1 + 0y_2 && (5.20)\\ &\text{subject to : } (i) : y_1 \geq i \qquad \text{for } i = 1, ..., k\\ &\qquad (k+1) : y_2 \geq 1\\ &\qquad\quad (i) : y_1 + (k+1)y_2 \geq \frac{(i-k)}{2} + 1 \qquad \text{for } i = k+2, k+4, ..., k+l-1\\ &\qquad\quad (i) : y_2 \geq \frac{(i-k-1)}{2} + 1 \qquad \text{for } i = k+3, k+5, ..., k+l. \end{aligned}$$

We illustrate the case when $k = 6, l = 9$ in Figure 5–3. The simplex method

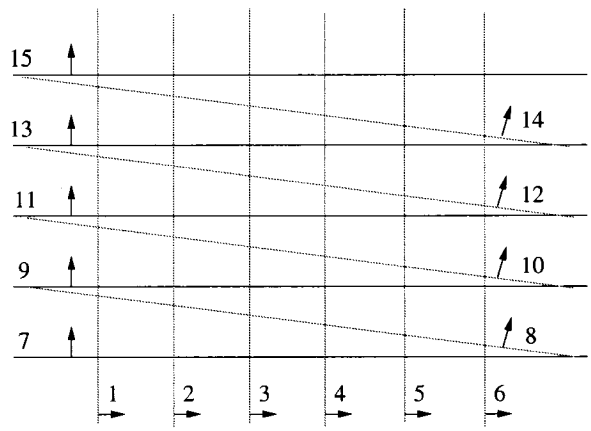


Figure 5–3: $D_{Bl}(d, n = d+2) = \Theta(n^2)$ (Inequalities)

with Bland's rule will follow a path (see Figure 5–4) of length k from $\{1, k+1\}$ to $\{k, k+1\}$ (by Lemma 5.2), then pivot to $\{k+2, k+1\}$, then 1 (turn) pivot to $\{k+2, k+3\}$, then 1 pivot to $\{1, k+3\}$ and follow a path of length k to $\{k, k+3\}$ (by Lemma 5.2), then pivot to $\{k+3, k+4\}$, then 1 (turn) pivot to $\{k+5, k+4\}$ then pivot to $\{1, k+5\}$ and follow a path of length k to $\{k+5, k+6\}$, and so on..., following a path of total length $k\left(\frac{l-1}{2}\right) + l - 3$ and ending at the basis $\{k, k+l\}$. The primal of (5.20) has dimension $(k+l)$ with $(k+l+2)$ inequalities. The construction requires that $k \geq 2$ and $l \geq 3$ requiring at least 5 dual inequalities in total. The lemma follows when we set $k = \lfloor \frac{d}{2} \rfloor$ and $l = \lceil \frac{d}{2} \rceil$ (subtracting 1 if $\lceil \frac{d}{2} \rceil$ is even). □

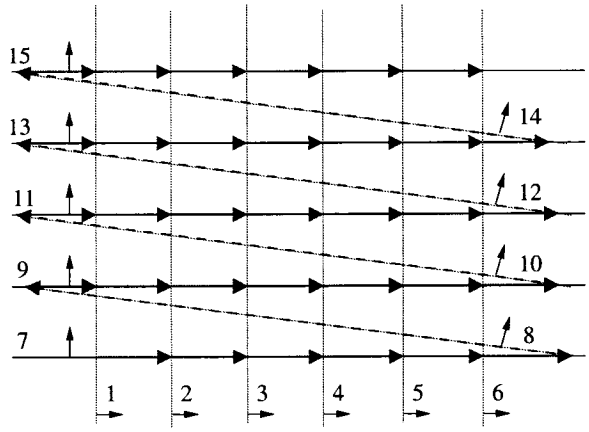


Figure 5–4: $D_{Bl}(d, n = d + 2) = \Theta(n^2)$ (Path)

Let's generalize these constructions to arbitrary dimensions.

Theorem 5.2. *$D_{Bl}(d, n) = \Omega(n^{n-d})$ for $n \leq \lfloor \frac{4d}{3} \rfloor$ for fixed $d \geq 1$.*

Proof. Let $Q(e, k_m)$ be the dual linear program with k_m inequalities in $\Re^e$ on which the dual simplex method with Bland's rule follows a path of length $\Omega((k_m)^e)$, where $Q(1, k_1)$ is constructed as in Lemma 5.2. Iteratively construct $Q(e + 1, k_{m+1} = k_m + l_m)$ by taking the k_m inequalities of $Q(e, k_m)$ and concatenating the l_m inequalities for some arbitrary $l_m \geq 3$ and odd:

$$
\begin{aligned}
&(k_m + 1) : y_{e+1} \geq 1 && (5.21)\\
&\quad (i) : y_e + (t + 1)y_{e+1} \geq \frac{i - k_m}{2} + 1 \qquad \text{for } i = k_m + 2, k_m + 4, ..., k_m + (l_m - 1)\\
&\quad (i) : y_{e+1} \geq \frac{i - k_m - 1}{2} + 1 \qquad \text{for } i = k_m + 3, k_m + 5, ..., k_m + l_m
\end{aligned}
$$

where t is the number of inequalities of $Q(e, k_m)$ of the form $y_e \geq b$.

The iterative construction reads like $Q(1, k_1) \rightarrow Q(2, k_2 = k_1 + l_1) \rightarrow ... \rightarrow Q(e - 1, k_{m-1} = k_{m-2} + l_{m-2}) \rightarrow Q(e, k_m = k_{m-1} + l_{m-1}) \rightarrow Q(e + 1, k_{m+1} = k_m + l_m)$ where $k > 2$ and $l_1, ..., l_m \geq 3$ with l_i odd for $i = 1, ..., m$.

Now if the simplex method with Bland's rule follow a path of length $\Omega((k_m)^e)$ on $Q(e, k_m)$ starting at basis $B \cup \{k_{m-1}\} \cup \{k_{m-1} + 1\}$ and ending at $B \cup \{k_{m-1}\} \cup \{k_m\}$ (where B is the set of indices shared by the starting and ending

bases of $Q(e-1, k_{m-1})$), then the simplex method with Bland's rule can be made to follow a path of length $\Omega((k_{m+1})^{e+1})$ on $Q(e+1, k_{m+1})$: starting at basis $B \cup \{k_{m-1}\} \cup \{k_{m-1}+1\} \cup \{k_m+1\}$ we follow a path from

$$
\begin{array}{ll}
B \cup \{k_{m-1}\} \cup \{k_{m-1}+1\} \cup \{k_m+1\} & \\
\quad \downarrow \text{ (to)} & \text{(of length) } \Theta((k_m)^e) \\
B \cup \{k_{m-1}\} \cup \{k_m\} \cup \{k_m+1\} & \\
\quad \downarrow & 1 \\
B \cup \{k_{m-1}\} \cup \{k_m+2\} \cup \{k_m+1\} & \\
\quad \downarrow & 1 \\
B \cup \{k_{m-1}\} \cup \{k_m+2\} \cup \{k_m+3\} & \\
\quad \downarrow & 1 \\
B \cup \{k_{m-1}\} \cup \{k_{m-1}+1\} \cup \{k_m+3\} & \\
\quad \downarrow & \Theta((k_m)^e) \\
B \cup \{k_{m-1}\} \cup \{k_m\} \cup \{k_m+3\} & \\
\quad \downarrow & 1 \\
B \cup \{k_{m-1}\} \cup \{k_m+4\} \cup \{k_m+3\} & \\
\quad \downarrow & 1 \\
B \cup \{k_{m-1}\} \cup \{k_m+4\} \cup \{k_m+5\} & \\
\quad \downarrow & 1 \\
B \cup \{k_{m-1}\} \cup \{k_{m-1}+1\} \cup \{k_m+5\} & \\
\quad \downarrow & \Theta((k_m)^e),
\end{array}
$$

etc..., following a path of total length $\Omega(l_m \cdot (k_m)^e)$ ending at $B \cup \{k_{m-1}\} \cup \{k_m\} \cup \{k_m + l\}$. Now choose $k_m = \lceil \frac{k_{m+1}}{2} \rceil$ and $l_m = \lfloor \frac{k_{m+1}}{2} \rfloor$ (subtracting 1 if $\lfloor \frac{k_{m+1}}{2} \rfloor$ is even) so that $\Omega(l_m \cdot (k_m)^e) = \Omega((k_{m+1})^{e+1})$. The theorem follows when we let the primal linear program of a specimen $Q(e, k_m))$ have $n = k_m + e$ inequalities in dimension

$d = k_m$. The construction of $Q(e, k_m)$ requires that $k_m \geq 3e \Rightarrow n \leq \lfloor \frac{4d}{3} \rfloor$, for fixed $d \geq 1$. □

When $n \leq \lfloor \frac{4d}{3} \rfloor$ there are $O(n^{n-d})$ bases, thus Theorem 5.2 shows there can be a maximum of $\Theta(n^{n-d})$ iterations.

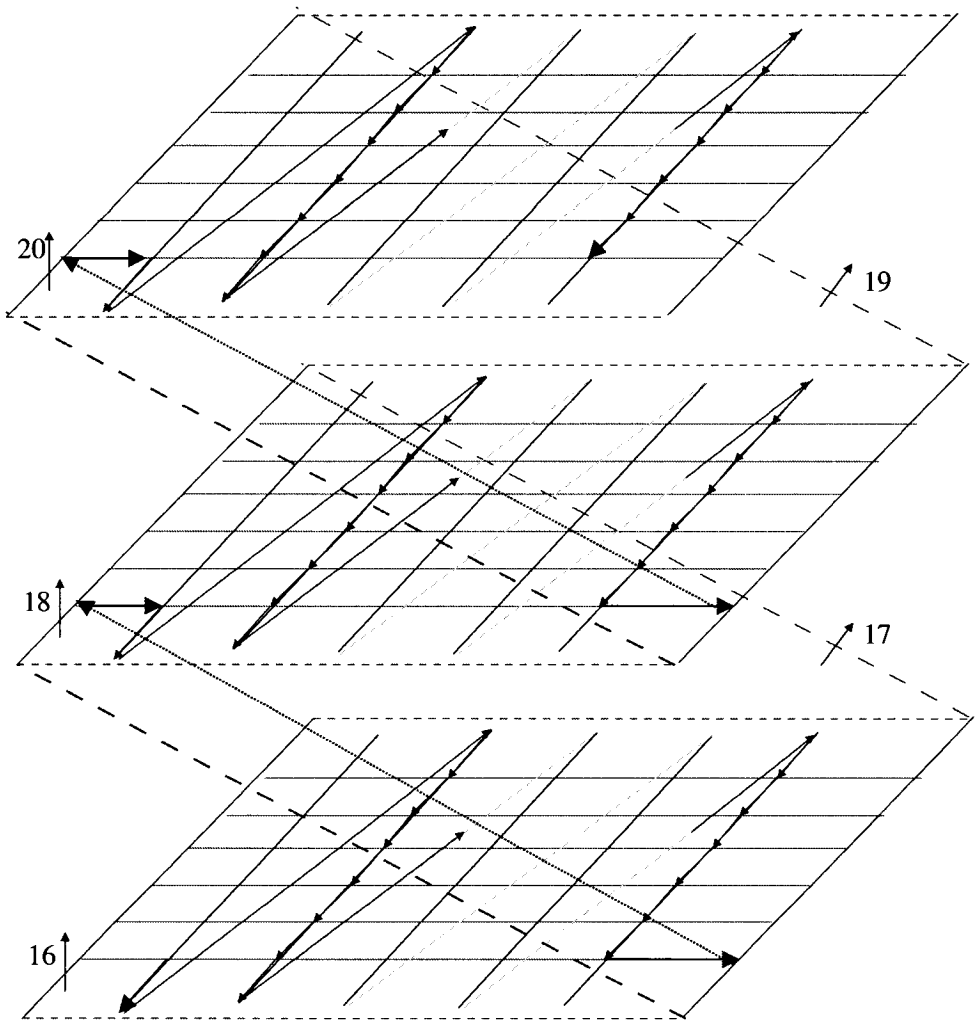


Figure 5–5: $D_{Bl}(d, n = d + 3) = \Theta(n^3)$

The construction of Lemma 5.3 follows the construction paradigm of Theorem 5.2. Figure 5–5 shows an instance of the construction when $D_{Bl}(d, n = d+3) = \Theta(n^3)$. It can be followed when reading the following corollary (Theorem 5.2 when $n = d + 3$):

Corollary 5.2. $D_{Bl}(d, n = d + 3) = \Theta(n^3)$ *for* $d \geq 8$.

Proof. Consider the dual linear program

$$\text{maximize } 0y_1 + 0y_2 + 0y_3 \tag{5.22}$$

$$\begin{aligned}
\text{subject to: } & (i) : y_1 \geq i \qquad \text{for } i = 1, ..., k \\
& (k+1) : y_2 \geq 1 \\
& \quad (i) : y_1 + (k+1)y_2 \geq \frac{(i-k)}{2} + 1 \qquad \text{for } i = k+2, k+4, ..., k+l_1-1 \\
& \quad (i) : y_2 \geq \frac{(i-k-1)}{2} + 1 \qquad \text{for } i = k+3, k+5, ..., k+l_1 \\
& (k+l_1+1) : y_3 \geq 1 \\
& \quad (i) : y_2 + \frac{l_1-1}{2} y_3 \geq \frac{(i-(k+l_1))}{2} + 1 \\
& \qquad \text{for } i = k+l_1+2, k+l_1+4, ..., k+l_1+l_2-1 \\
& \quad (i) : y_3 \geq \frac{(i-(k+l_1)-1)}{2} + 1 \\
& \qquad \text{for } i = k+l_1+3, k+l_1+3, ..., k+l_1+l_2
\end{aligned}$$

with $k \geq 2, l_1 \geq 3, l_2 \geq 3$. We illustrate the case when $k = 6, l_1 = 9, l_2 = 5$ in Figure 5–5. The simplex method with Bland's rule will follow a path of length $k\left(\frac{l_1-1}{2}\right) + l_1 - 3$ from $\{1, k+1, k+l_1+1\}$ to $\{k, k+l_1, k+l_1+1\}$ (by Lemma 5.3) then pivot to $\{k, k+l_1+2, k+l_1+1\}$ then (turn) pivot to $\{k, k+l_1+2, k+l_1+3\}$ then pivot to $\{k, k+1, k+l_1+3\}$ and follow a path of length $k\left(\frac{l-1}{2}\right) + l_1 - 3$ to $\{k, k+l_1, k+l_1+3\}$ (by Lemma 5.3) then pivot to $\{k, k+l_1+4, k+l_1+3\}$ then (turn) pivot to $\{k, k+l_1+4, k+l_1+5\}$ then pivot to $\{k, k+1, k+l_1+5\}$, etc... ending at the basis $\{k, k+l_1+1, k+l_1+l_2\}$ having followed a path of total length $\Theta(k \cdot l_1 \cdot l_2)$. The primal of (5.22) has dimension $(k+l_1+l_2)$ with $(k+l_1+l_2+3)$ inequalities. The construction requires that $k \geq 2, l_1 \geq 3, l_2 \geq 3$ requiring at least 8 dual inequalities in total. The corollary follows when we set $k = \lfloor \frac{d}{3} \rfloor, l_1 = \lfloor \frac{d-k}{2} \rfloor, l_2 = \lceil \frac{d-k}{2} \rceil$ (subtracting 1 if necessary to ensure that l_1 and l_2 are odd). □

Open Problem 5.1. *Can the simplex method visit more bases than the maximal number of vertices given by the Upper Bound Theorem?*

Open Problem 5.2. *Is* $D_{Bl}(d,n) = \Theta(n^d)$ *for* $n \geq 2d$, $d > 3$*?*

Open Problem 5.3. *Construct a family of linear programs with* $\frac{4d}{3} < n < 2d$ *on which the simplex method takes many pivots.*

CHAPTER 6
Enumerating Pivot Paths

6.1 Introduction

We have seen worst-case constructions where the simplex method visits $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ bases ([2] and in Section 5.1), asymptotically matching the maximum number of vertices, $M(d, n)$, that a $d-$polytope with n facets can have. However, Pfeifle and Ziegler [79] have proved that for nondegenerate linear programs the maximum number of bases along a simplex method pivot path does not achieve $M(d, n)$ in general. In the presence of degeneracy, less is known about the worst-case length of simplex method paths (see Chapter 3 and 5). These results, or lack of, lead to the following questions: Given a system of inequalities defining a polyhedron, what objective function and starting basis would return the longest simplex method pivot path following some finite pivot rule? Can we even compute this? The answer is yes: an algorithm and an implementation for combinatorial pivot rules[1] are the subject of the present chapter.

[1] We recall that combinatorial pivot rules select a cobasic variable with a positive coefficient in the $z-$row based on its index, and coefficient-based pivot rules select a variable based on the value of the its positive coefficient.

Given an $H-$representation of a polyhedron P and a feasible basis B, we show that there is a finite set of objective functions optimal at B which is sufficient for the enumeration of all possible simplex pivot paths, for some finite combinatorial pivot rule, from all other feasible bases to B. Such an algorithm is a useful tool to have for the analysis of pivot paths, both degenerate and nondegenerate. We provide an implementation and use it to present a complete path length analysis for the simplex method with Bland's rule on selected instances of low-dimensional polytopes.

Throughout this chapter, we consider a polyhedron P defined by n inequalities in d dimensions, and linear objective functions $z = \max cx$ for $x \in P$ which together define a linear program,

$$\begin{aligned} &\text{maximize } z = \sum_{j=1}^{d} c_j x_j \\ &\text{subject to } \sum_{j=1}^{d} a_{ij} x_j \leq b_i \qquad \text{for } i = 1, ..., n, \end{aligned} \tag{6.1}$$

and a dictionary,

$$\begin{aligned} x_i &= b'_i - \sum_{j \in N} a'_{ij} x_j \qquad \text{for } i \in B \\ z &= z' + \sum_{j \in N} c'_j x_j, \end{aligned} \tag{6.2}$$

for basis B and cobasis N. We will only consider linear programs where the objective function is in general position and bounded with respect to P, ensuring that no two vertices of P share the same objective value and that there is a unique optimal vertex.

In the following sections we show that only a finite number of objective functions need to be considered to enumerate all possible pivot paths, and we detail reverse search for basis enumeration and ray-shooting reverse search for cell enumeration - the primary ingredients used for our pivot path enumeration

algorithm. We then present our algorithm and an implementation in Section 6.4 and 6.5 respectively, displaying some computational results in Section 6.6.

6.2 Enumerating Pivot Paths in a Linear Program

Problem 6.1. *Given a linear program and a finite pivot rule R and optimal basis B^*, enumerate all possible pivot paths, when the simplex method follows R, that terminate at B^*.*

The reverse search algorithm of Avis and Fukuda [6] solves this problem efficiently. By the finiteness of pivot rule R, there exists a simplex method pivot path following R from every basis to an optimal basis. The set of paths terminating at B^* form a tree with root B^*. Reverse search traverses this tree in a depth-first-search manner by simply reversing the pivot rule R to go down the tree, and following R to go up the tree. The algorithm was originally designed to enumerate the vertices of a polytope, but the reverse search tree also represents the collection of all possible pivot paths to the optimal basis B^* taken by the simplex method following rule R for a given objective function. The height of the tree is the length of the longest pivot path to B^*. It is proved in [6] that $O(nd^2)$ time per basis and total space $O(nd)$ is required to enumerate the entire reverse search tree.

Algorithm 6.1 (Reverse Search for Pivot Path Enumeration).

Input: *n inequalities in d variables, a linear function cx.*

$B = B^*; j = 1;$ *height = 0;*

print(B, height)

Repeat

while $j \leq d$ **{**

$v = N_j;$ // *N_j is the j^{th} cobasic variable*//

if *reverse*(B, u, v)

then **{** *pivot*(B, u, v), $j = 1$ //*reverse traverse*//

height++;

print(B, height) }

else $j = j + 1$; }

selectpivot(B, r, j)

pivot(B, r, N_j) *//forward traverse//*

$j = j + 1$;

height$--$;

Until $j > d$ *and* $B = B^*$.

Output: *All pivot paths to the optimal basis* B^*.

- *pivot*(B, r, s): pivot from basis B on basic variable r and cobasic variable s.
- *selectpivot*(B, r, j): return basic variable r and index j with $s = N_j$ where r and s are as selected by pivot rule R given basis B.
- *reverse*(B, u, v): given basis B and cobasic index $v \in N$, return TRUE and u if there exists basic index $u \in B$ such that pivot rule R applied to dictionary with basis $\widetilde{B} = B - u + v$ generates a pivot back to B. Return FALSE otherwise.

6.3 Enumerating Objective Functions for an Optimal Basis

Problem 6.2. *Given a polyhedron P, basis $\overline{B}$, and a finite combinatorial pivot rule R, consider all distinct pivot paths that can be taken by the simplex method following R to optimal basis $B^* = \overline{B}$. List one objective function for each such path.*

We are interested in listing all distinct pivot paths that can be taken by the simplex method following R to optimal basis B^* over all objective functions yielding $B^* = \overline{B}$. The set of paths on a given polytope is finite, and for each such path there may or may not be an objective function that follows it. How to fit an objective function to a path (or prove no such fit exists) is not so trivial: for coefficient-based pivot rules we can enumerate the set of paths on the polytope and then for each path employ a dual version of the Perturbation Algorithm 3.1 of Chapter 3 to test whether we can scale the input inequalities so that the path abides by the specific

rule. (A similar approach was taken to compute scaling factors in simplex method cycles in Section 2.3 of Chapter 2.) For combinatorial pivot rules we present a more intelligent approach, which we will now outline.

Every pivot rule for the simplex method depends on the sign of the dictionary coefficients. Namely, in a dictionary (6.2) the simplex method requires $c'_s > 0$ before any pivot rule can consider a pivot on cobasic variable s. We have seen that the dictionary coefficients can be interpreted as a ratio of determinants, recall Theorem 1.6 of Section 1.2.5 which states that

$$c'_s = \frac{\begin{vmatrix} c \\ G_{N\setminus s} \end{vmatrix}}{\begin{vmatrix} a_{s-d} \\ G_{N\setminus s} \end{vmatrix}},$$

where $G_N = \begin{bmatrix} a_{j1} & \dots & a_{jd} & : & \{j+d\} \in N \end{bmatrix}$, $a_s = (a_{s1}, a_{s2}, ..., a_{sd})$, $c = (c_1, c_2, ..., c_d)$ and $s \in N$.

The sign of the determinant is the orientation of the arrangement of the vectors. In particular, the position of vector c with respect to the vectors of $G_{N\setminus s}$ determines the sign of c'_s. The sign of c'_s is fixed for N and s. c is a $d-$dimensional vector, and the vectors of $G_{N\setminus s}$ lie in a $d-1$ dimensional flat: the hyperplane $h_{N\setminus s}$ in dimension d through the origin. If we assign an orientation to this hyperplane then the sign of c'_s will be positive when c lies on one side of $h_{N\setminus s}$ and negative if c lies on the opposite side.

Consider the sign of $c_{N^i_j}$ (the j^{th} cobasic variable for cobasis N^i) for $j = 1, ..., d$ and for all possible feasible bases B^i (cobasis N^i). For each $c'_s = c'_{N^i_j}$ there is a hyperplane $h_{N^i\setminus s}$ containing the vectors of $G_{N^i\setminus s}$. There are up to $\binom{n}{d-1}$ of these hyperplanes, and together they form a central arrangement $A(H)$ decomposing $\Re^d$ into $\Theta\left(\binom{n}{d-1}^d\right)$ cells (for properties of an arrangement see [21]). Each cell represents a unique sign association of the cobasic variables (over all feasible bases) and one point from each cell adds to a sufficient list of objective vectors that yield

all possible cobasic variable sign structures over all feasible bases along simplex pivot paths to the optimal basis. So we just need to enumerate the cells of $A(H)$, and compute an interior point for each one to get the desired objective functions.

Of course we are only interested in the collection of vectors c which make $\overline{B}$ the optimal basis. Let a^i be the gradient of inequality i, then the dual of (6.1), the set of constraints

$$y_1 a^1 + y_2 a^2 + ... + y_n a^n = c \qquad (6.3)$$
$$y_1, y_2, ..., y_n \geq 0,$$

specifies that the objective vector c be a nonnegative combination of the gradients a^i of the primal inequalities. The duality theorem for linear programming states that the feasible basis $\overline{B}$ (cobasis $\overline{N}$) for the linear program (6.1) is optimal if and only if $\sum_{i \in \overline{N}} y_i a^i = c$. In other words c is contained in the $d-$dimensional cone $C = C(a^i)$ for $i \in \overline{N}$. Hence we only need to consider the cells of $A(H)$ that intersect cone C as only then do all cobasic variables of the dictionary with basis $\overline{B}$ have nonpositive sign (optimality). Note that the bounding hyperplanes of the inequalities that define C are also hyperplanes in $A(H)$. An interior point of each cell of $A(H)$ that intersects C corresponds to a vector c which is optimal at basis $\overline{B}$.

We now describe an algorithm for computing this set of objective functions. First we list the hyperplanes of $A(H)$ by considering each set of $d-1$ gradient vectors of the n inequalities. If the set of vectors is affinely independent then we add the hyperplane containing these vectors to $A(H)$. We then use a modified version of the cell enumeration algorithm of Ferrez et al. [23], described below, to compute an interior point for each cell of $A(H)$ that intersects with cone C.

Let $m = |H|$ and let $S = S(H)$ be the set of sign vectors of the cells of the arrangement $A(H) \in \Re^d$. Since $A(H)$ is central, only half of the cells need to be considered. In fact we can just look at a cut section of $A(H)$ with a fixed

hyperplane h not containing the origin. If we choose h to be the hyperplane containing a cross section of C, then we obtain a cut section of $A(H)$ which is an arrangement of m hyperplanes in $\Re^{d-1}$. We assign orientations to the hyperplanes so that a cell within the cross section of C has the sign vector of all positives. Figure 6–1 shows a cut section of an arrangement of 6 hyperplanes in $\Re^3$ and the sign vectors of the cells of length 6. The shaded region represents cross section of the optimal cone.

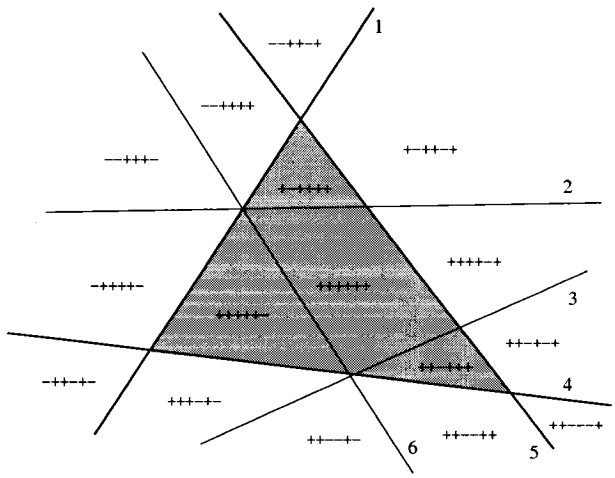


Figure 6–1: Cut section of an arrangement and associated cell sign vectors

We employ the reverse search algorithm of Ferrez et al [23] for enumerating the cells of $A(H)$ with modification to enumerate only those cells that intersect the optimal cone. The algorithm computes an interior point for each cell of $A(H)$ in C which we output as they are the desired objective vectors. The adjacency oracle and local search function using ray shooting of the reverse search are defined as follows:

- **Adjacency Oracle**: let s be any cell in S. An index $j \in \{1, ..., m\}$ is *flippable* in s if $s_j \neq 0$ and the vector obtained from s by reversing the j^{th} sign is again a cell. The adjacency oracle $Adj(s, j)$ returns this new cell if j is flippable and NULL otherwise, for $j = 1, ..., m$.

- **Ray Shooting Local Search Function**: let s^* be any cell in S that is known to be in C (we can find this cell using linear programming). Without loss of generality we can assume that s^* is the vector of all +'s. For any cell $s \in S \setminus s^*$ the local search function f must return a cell s' which is adjacent to s and is *closer* to s^* : $f(s) = s'$. Any cell s diffferent from s^* has flippable index j such that $s_j = -$. Flipping this index leads us closer to s^*. We define f as follows: let p^* be an interior point of goal cell s^* and p the interior point of cell s. Shoot a ray from p to p^*. The ray will hit all hyperplanes separating s and s^*. Select the first hyperplane hit by the ray (breaking ties by symbolic perturbation) and cross it to obtain the new cell s'. Ferrez et al. show that an interior point for a cell, defined by $\widetilde{A}y \leq \widetilde{b}$ for $(m-1) \times (d-1)$ matrix $\widetilde{A}$, can be computed by solving the linear program

$$\begin{aligned} \max\ & y_0 \\ \widetilde{A}y + e \cdot y_0 &\leq \widetilde{b} \\ y_0 &\leq K, \end{aligned} \tag{6.4}$$

where e is the vector of all 1's and K is a large number that bounds the LP.

Algorithm 6.2 (Objective Vector Enumeration).

Input: *m hyperplanes in $d-1$ dimensions and cone C.*

$s = s^*; j = 1$

print(interior point of s)

Repeat

while $j \leq m$ **{**

$j := j + 1;$

$next = Adj(s, j)$

if $next \neq NULL$ *and* $next \in C$ ***then***

if $f(next) = s$ ***then*** **{** *//reverse traverse//*

$s := next; j := 1$

print(interior point of s) } }

if $s \neq s^*$ ***then*** { *//forward traverse//*

$s' := s; s := f(s); j := 0$

repeat $j := j + 1;$

until $Adj(s, j) = s'$ }

Until $j = m$ *and* $s = s^*$.

Output: *Set of sufficient objective vectors inside cone C (optimal at* B^**).*

Figures 6–2a and 6–2b depict the reverse search for cell enumeration using ray shooting. The solid lines represent the reverse search tree generated by the Objective Vector Enumeration Algorithm. The entire tree, including the dotted lines, represents the reverse search tree which would be generated by the original algorithm of Ferrez et al. [23].

6.4 Enumerating Pivot Paths to an Optimal Basis

Problem 6.3. *Given a polyhedron P, a feasible basis* B^**, and a finite combinatorial pivot rule R, enumerate all possible simplex method pivot paths following R in linear programs defined over P with objective functions yielding optimal basis* B^*.

Algorithm 6.3 (Pivot Path Enumeration).

Input: *n inequalities in d dimensions, pivot rule R, and basis* B^*.

(0) *Let H be set of hyperplanes initially empty.*

(1) *Consider each set of* $d - 1$ *gradient vectors of the n inequalities. If the set of vectors is affinely independent and lie in a hyperplane h, then add h to H.*

(2) *Run the Objective Vector Enumeration Algorithm on* $A(H)$ *with cone C of basis* B^**. Let O be the set of sufficient objective vectors.*

(3) *For each objective vector of O, run the Reverse Search for Pivot Path Enumeration Algorithm.*

Output: *All possible simplex pivot paths following rule R to basis* B^*.

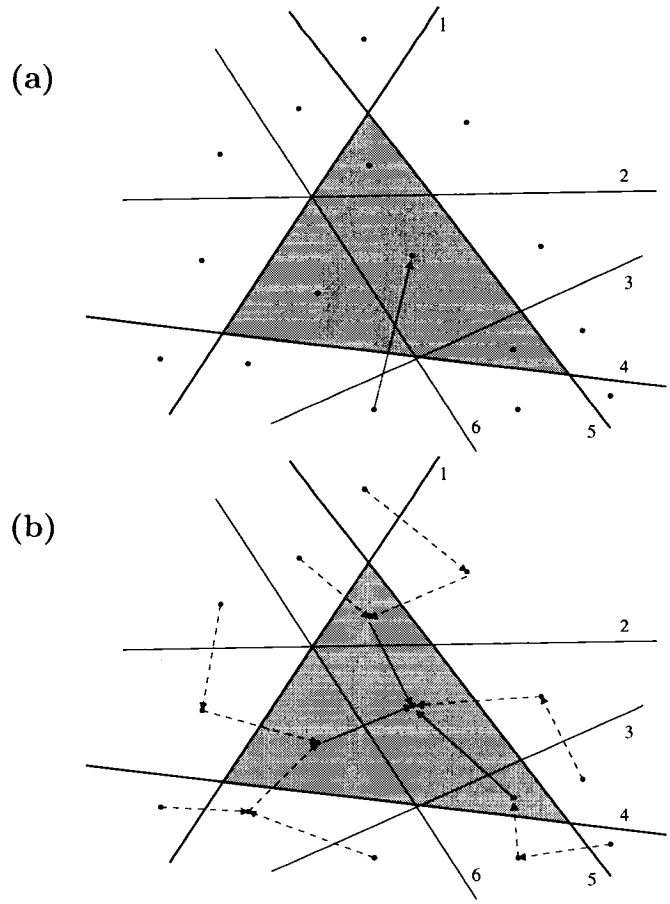


Figure 6–2: (a) Interior points and ray shooting local search, and (b) reverse search tree for cell enumeration using ray shooting

The complexity of the algorithm depends on the number $|O|$ of objective functions generated of which there can be $O\left(\binom{n}{d-1}^d\right)$. The modified reverse search algorithm of Ferrez et al. has time complexity $O(m\ LP(m,d)|O|)$ and space complexity $O(md)$ where m is $O\binom{n}{d-1}$ and $LP(m,d)$ is the time to solve an LP with m inequalities in d variables.

6.5 Implementation Details

We have implemented a version of this algorithm in C for three finite simplex pivot rules: Bland's least-index rule [13], and both Dantzig's largest coefficient pivot rule [19] and the least-index positive coefficient rule (as used in [3]) with lexicographic pivoting. (Dantzig's rule is not combinatorial, but implemented

nonetheless for comparison sake.) A single executable program and a user guide are available [107]. The package comprises of

- modified versions of Avis' *lrs* and *qrs* [106] reverse search implementations: functionality is added to start reverse search from a specific root basis and objective function, reverse search with Dantzig's rule implemented and set as an option in *lrs*,
- the *rs_tope* program of Ferrez and Fukuda [31] for enumeration cells of an arrangement using ray shooting (which relies on Fukuda's *cddlib*[30] to solve *lp*'s and Marzetta's *zram* package [69] for reverse search).

All computations are done exactly using extended precision arithmetic. The input files are in standard *polyhedra format* [3] listing the n inequalities in the form $(i:)\ b_i\ -a_{i1}\ -a_{i2}\ ...\ -a_{id}$ for $i = 1, ..., n$. Execution proceeds as follows:

- Data structures initialized and input is read
- All $\binom{n}{d-1}$ combinations of $d-1$ gradient vectors are computed. For each combination, if the lowest-dimensional flat f containing the vectors has dimension $d-1$ then we store hyperplane h containing f.
- The set of hyperplanes stored are sent as input to *rs_tope* which returns a set of points.
- For each point the program sends the original inequalities, optimal basis, and objective function (point) to *qrs* and *lrs*:
  (1) *qrs* returns the reverse search tree of Bland's rule.
  (2) *lrs* (default) returns the reverse search tree of the least-index lexicographic rule.
  (3) *lrs* (Dantzig rule option) returns the reverse search tree of Dantzig's lexicographic rule.

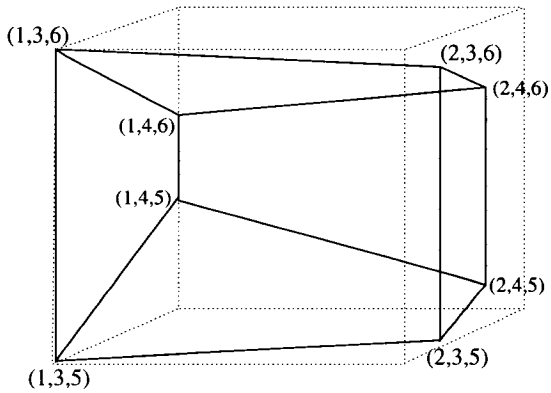
The program outputs the set of sufficient objective vectors and their associated reverse search trees, and the longest path from leaf to root (number of nodes). For

example, the simplex method following Bland's rule on the Klee-Minty cube in 3D (Bland rule variant) with optimal basis $\{1, 3, 6\}$ has the following input and output:

Input:

(1:) 0 1 0 0
(2:) 1 -1 0 0
(3:) 0 -1 3 0
(4:) 3 -1 -3 0
(5:) 0 0 -1 3
(6:) 3 0 -1 -3

basis: 1 3 6



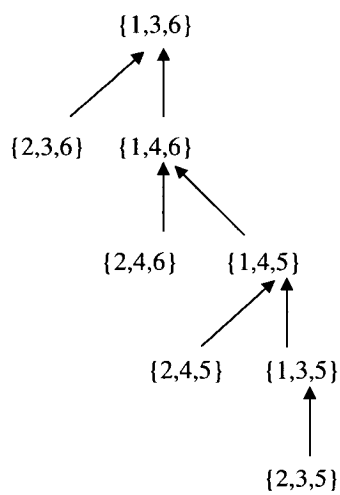
Output:

Objective vectors optimal at $\{1, 3, 6\}$:

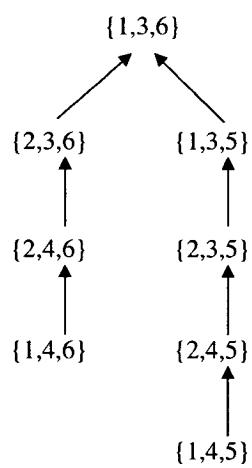
(1) -14617 2886 12777
(2) -2 -13/3 3
(3) -2 -19/3 3
(4) -2 -23/3 3
(5) 2 -23/3 3
(6) -2 2/3 18
(7) -2 0 20
(8) 2 0 20
(9) 2 -2/3 18
(10) 2 -19/3 3
(11) -2 -2/3 18

All Bland's rule pivot paths to optimal basis $\{1, 3, 6\}$:

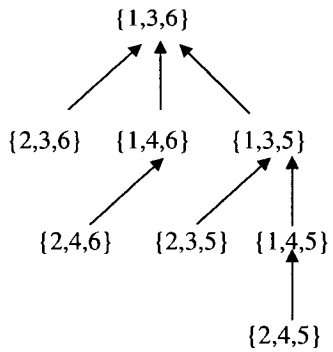
(1):



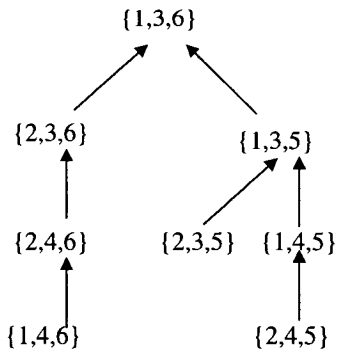
(4), (5):



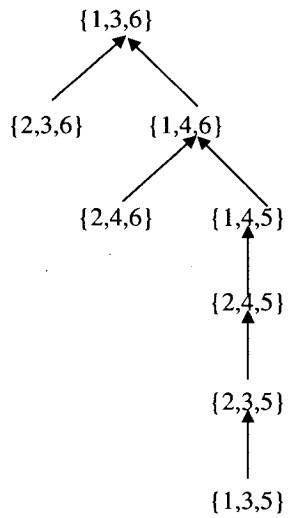
(2):



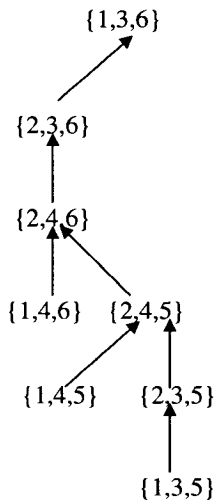
(3):



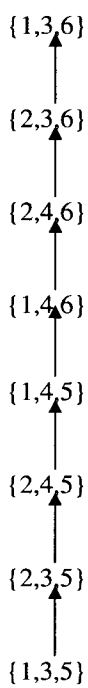
(6):



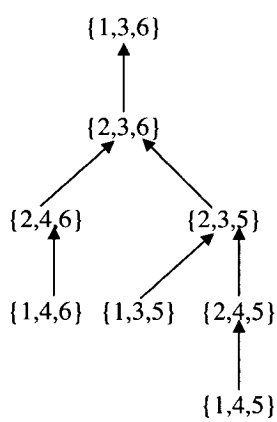
(9):



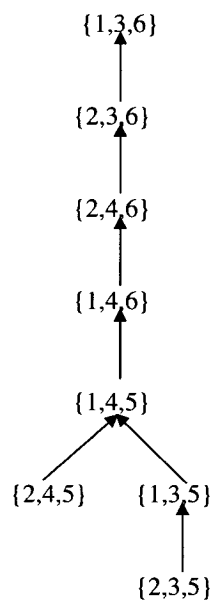
(7), (8):



(10):



(11):



Statistics:

```
Longest *Longest* path length = 8
Shortest *Longest* length = 4
Median *Longest* path length = 5
```

The sign structure of the bases along the paths generated is different for each objective function, however the difference may not affect the decision taken by the pivot rule. This explains why some objective functions may return the same pivot path trees, as is the case with the above example.

6.6 Computational Results

If we consider every feasible basis B of P and run the Pivot Path Enumeration Algorithm for each basis, then we can compute the longest possible path that the simplex method can take following a finite combinatorial pivot rule for a linear program defined over P. This approach allows us to analyze pivot paths for completely degenerate polyhedra, compare path lengths for polytopes and their *collapsed* (completely degenerate) versions. Computing these paths is impractical when n and d get too large as the algorithm executes a reverse search up to $O\left(\binom{n}{d-1}^d\right)$ times for each basis of P. The enumeration is an employable tool when the parameters are small; we ran experiments on instances of some low dimensional polytopes.

Table 6–1: Longest paths on Klee-Minty cube in $\Re^3$

Optimal Basis	# Obj.	Bland	Dantzig	Least-Index
$\{4,5,6\}$	11	4	6	4
$\{1,5,6\}$	3	5	7	4
$\{1,3,6\}$	2	4	6	4
$\{2,4,6\}$	4	5	5	4
$\{2,3,4\}$	4	5	5	4
$\{1,2,3\}$	2	6	6	4
$\{1,3,5\}$	3	7	7	4
$\{3,4,5\}$	11	6	8	4

In Table 6–1 we present the longest path lengths found by the Pivot Path Enumeration Algorithm for every feasible basis of the Klee-Minty cube [61]. The number of objective functions optimal at each basis, and the longest path length

for Bland's rule, Dantzig's rule, and the least-index positive coefficient rule are shown. As expected, there is an LP with a simplex pivot path going through all 8 vertices. When we collapse the Klee-Minty cube to a completely degenerate polyhedron more feasible bases are created. In Table 6–2 we present path length results returned by our algorithm for each basis of this degenerate polyhedron. Preliminary observations show that the longest path lengths become shorter, none achieve the maximal length of 8 of the nondegenerate counterpart. Similar results (see Table 6–3) were observed for other low-dimensional examples.

Table 6–2: Longest paths on collapsed Klee-Minty cube in $\Re^3$

Optimal Basis	**# Obj.**	**Bland**	**Dantzig**	**Least-Index**
$\{4,5,6\}$	11	4	3	2
$\{1,5,6\}$	3	5	3	3
$\{1,3,6\}$	2	4	6	4
$\{2,4,6\}$	4	4	3	3
$\{2,3,4\}$	4	4	3	3
$\{1,2,3\}$	2	5	4	4
$\{1,3,5\}$	3	6	3	3
$\{3,4,5\}$	11	7	3	2
$\{2,3,6\}$	2	5	2	3
$\{1,3,6\}$	6	2	3	3
$\{2,5,6\}$	5	2	4	3
$\{2,3,5\}$	3	2	4	3
$\{3,5,6\}$	1	1	1	1
$\{3,4,6\}$	6	6	3	2

Table 6–3: Longest paths on selected polytopes and their collapsed versions.

Polyhedron	**n**	**d**	**# Obj.**	**# Bases**	**Bland**	**Dantzig**	**Least-Index**
Klee-Minty	8	4	696	16	11	16	5
(collapsed)	8	4	696	41	11	7	5
Polar Cyclic	7	3	37	10	7	7	5
(collapsed)	7	3	37	26	6	7	5
Polar Cyclic	6	4	444	9	8	6	3
(collapsed)	6	4	444	15	6	5	3
Polar Cyclic	7	4	2908	14	9	8	5
(collapsed)	7	4	2908	31	8	8	5

A true exhaustive analysis of the possible pivot paths over a polyhedron for a combinatorial pivot rule would require executing a reverse search $O\left(\binom{n}{d-1}^d\right)$ times for each basis of P and for every permutation of the inequality indices (dictionary variable indices). Even in low dimensions the number of iterations explodes, demonstrating the vast number of possibilities that contributes to the difficulty of analyzing pivot paths.

Open Problem 6.1. *Find a general nondegenerate (linear program) construction on which the longest pivot path taken by the simplex method with Bland's rule is shorter than the longest pivot path taken on the collapsed version.*

CHAPTER 7
Computing Disjoint Paths on Polytopes

7.1 Introduction

The vertices and edges of a polytope P form an undirected graph $G(P)$. A linear function cx in R^d in *general position* gives an orientation on the edges of $G(P)$ allowing us to define an acyclic digraph $D_c(P)$. This digraph has a unique *source*, or vertex minimizing cx, and *sink*, or vertex maximizing cx. The primal simplex method follows a path in $D_c(P)$ from any given starting vertex to the sink.

Conjecture 3 (Ziegler's Strict Monotone Hirsch Conjecture, [102] p.86). *Let P be a $d-$polytope with n facets and cx a linear function in general position, then there exists a path in $D_c(P)$ from source to sink of length at most $n-d$.*

Holt and Klee proved that the Strict Monotone Hirsch Conjecture is true when $d \leq 4$. In doing so they unearthed the following interesting fact.

Theorem 7.1 (Holt-Klee Condition [50]). *Let P be a $d-$polytope with n facets and cx a linear function in general position with respect to P, then there exist d vertex-disjoint strict monotone paths from source to sink.*

Studying these disjoint paths has the potential to provide new insight into designing a polynomial-time simplex method, or proving none exists. To study these paths it would be useful to have a tool to compute them. In this chapter we give an algorithm for computing disjoint paths on a polytope P given by a

system of n linear inequalities in d variables. Given a directed graph, a maximum cardinality set of disjoint paths from source to sink can be found efficiently using network flow techniques, see Section 7.2 for details. However, in our case we are not given explicitly the digraph $D_c(P)$. Computing $D_c(P)$ would require the enumeration of all vertices of P, a computationally difficult task, see Avis, Bremner and Seidel [4]. Worse, the storage requirement for $D_c(P)$ can be exponential in the input size. For example, the polar of a cyclic $d-$polytope with n facets has $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ vertices. Complete vertex enumeration and storage would be required even if just a few disjoint paths are required. Our algorithm avoids computing $D_c(P)$ explicitly, using an oracle based on the simplex method to provide edges.

A further complication is caused by degeneracy. The set of all bases of P defines an undirected graph, called the basis graph $B(P)$ (defined in Chapter 5): the vertices of $B(P)$ are bases of P, and the edges of $B(P)$ are defined by the pivot operation, see Section 7.3 for details. We will avoid using the term "vertex of $B(P)$" to avoid confusion with the the term "vertex of P". Instead we will refer to the "bases" of $B(P)$. For a simple polytope P, the graphs $B(P)$ and $G(P)$ are identical. However, the graph $B(P)$ can be much larger than the graph $G(P)$, since a highly degenerate vertex may be representable by an exponential number of bases. The simplex method follows paths in $B(P)$.

Let cx be in general position with respect to P. A path in $B(P)$ is said to be *monotone* with respect to c, if cx is nondecreasing when applied to the vertices of P corresponding to consecutive bases in the path. In the case of nondegeneracy, cx increases strictly along the path. In the presence of degeneracy, a degenerate vertex appears as one or more consecutive bases on the path. The vertices of P as visited by a monotone path in $B(P)$ induce a strictly increasing path in $D_c(P)$. Note however, that to find vertex-disjoint monotone paths in $D_c(P)$ it is not sufficient

to find basis disjoint monotone paths in $B(P)$: in the presence of degeneracy basis disjoint paths are not neccessarily vertex-disjoint.

In this chapter we present an algorithm and an implementation for finding d vertex independent monotone paths in $D_c(P)$ from a basis of the source to a basis of the sink of P. The ingredients of the algorithm are:

- the simplex method to determine edges of the graph $B(P)$;
- a network flow algorithm to determine disjoint paths in $B(P)$;
- reverse search to handle degenerate vertices in a space efficient manner.

In Section 7.2 and 7.3 we give the details of these ingredients. Combining these we then present our algorithm in Section 7.4. In Section 7.5 we discuss complexity, implementation, and some experimental results. In the implementation, we will not in fact require that cx is in general position with respect to P, although for simplicity this will be assumed in describing the algorithm in Section 7.4. Experimental results show that the algorithm is particularly advantageous when only a few disjoint paths are required, but also excels when the input has little or no degeneracy, and is especially memory-efficient when the polytope has many vertices. For example, we computed 10 disjoint paths on the polar of the cyclic polytope of dimension 10 with 50 facets storing only $199,000$ vertices while the polytope has $1,357,510$ vertices. The median path length was 32 vertices. Further preliminary results show that the lengths of the disjoint paths are typically short.

7.2 Disjoint Paths in a Digraph

Let D be a directed graph with two specified vertices s and t. An algorithm for finding the maximum number of edge-disjoint $s-t$ paths in D is quite simple and can be found in most graph theory textbooks such as [85]. It is a specialization of the maximum flow algorithm of Ford and Fulkerson [25]: find a directed path from s to t in D, reverse the direction of the edges along this path, and repeat this process until no further path is found.

For a directed path $\overrightarrow{P}$ in a digraph D, let $D \leftarrow \overrightarrow{P}$ be the digraph arising from D by reversing the orientation of each arc in $\overrightarrow{P}$.

Algorithm 7.1 (Edge-Disjoint Paths). *Determine $D_0, D_1, \ldots$ as follows:*

Set $D_0 := D$.

***Find_Path**(D_i): Find an $s-t$ path $\overrightarrow{P}$ in D_i*

If $\overrightarrow{P}$ is found

*then **Reverse_Path**(D_i, $\overrightarrow{P}$): set $D_{i+1} := D_i \leftarrow \overrightarrow{P}$.*

Otherwise stop.

Proposition 7.1. *The set R_i of arcs of D that are reversed in D_i form i edge-disjoint $s-t$ paths.*

Proposition 7.2. *The edge-disjoint paths algorithm finds a maximum collection of edge-disjoint $s-t$ paths.*

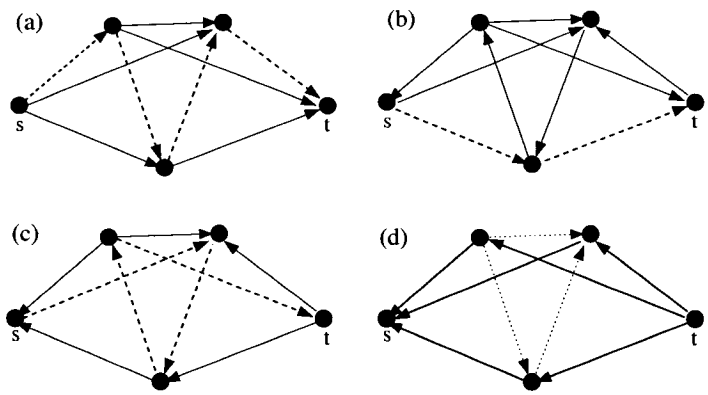
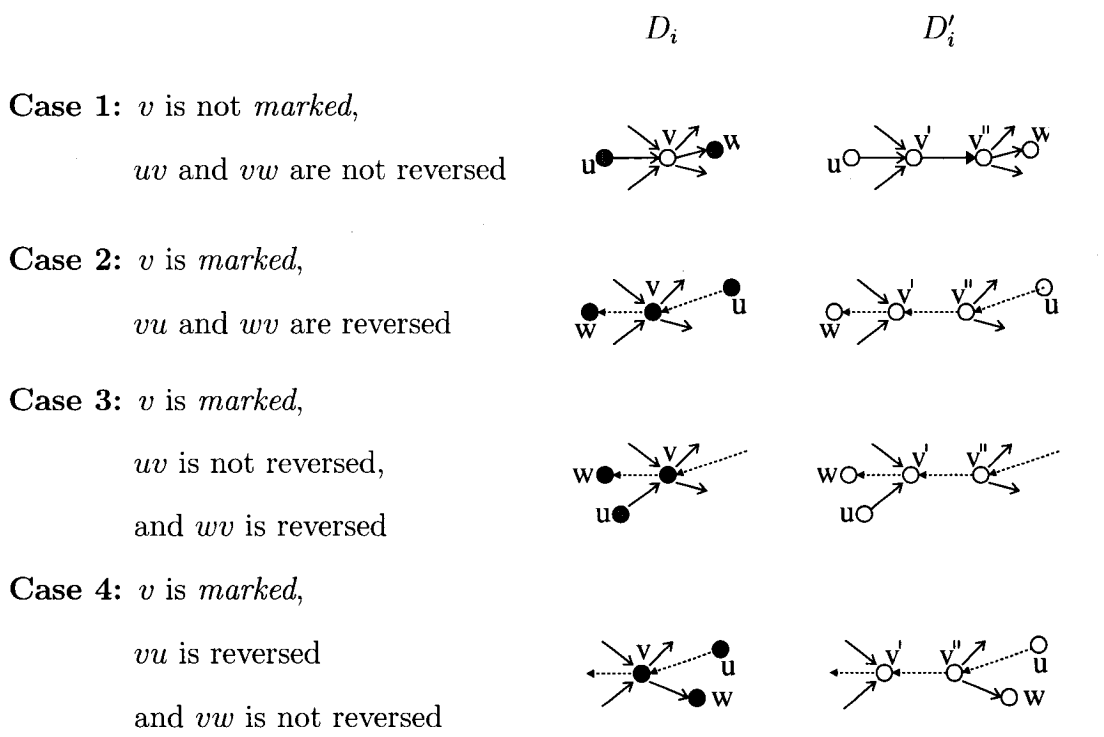


Figure 7–1: Computing disjoint paths: an example

See [85, p.135] for proofs. Figure 7–1 shows the algorithm at work. The dashed lines in Figures 7–1 (a)-(c) show the new path found at each iteration. The solid edges in Figure 7–1(d) indicate three edge-disjoint paths from source to sink, with edges reversed.

A simple modification of the algorithm allows us to find *vertex-disjoint* paths. Given a digraph D, create D' by replacing every vertex v in D by two vertices v', v'' and an arc $v'v''$. Replace each arc $uv \in D$ by uv', and $vw \in D$ by $v''w$. Now

edge-disjoint paths in D' are easily mapped to vertex-disjoint paths in D. Instead of creating new vertices and edges, we can modify the algorithm to simply *mark* a vertex v in D if it is used in a path. In **Find_Path**(D_i) we check whether a vertex v is marked, signifying that the artificial edge $v'v'' \in D'$ is reversed. At a vertex v, let u be the predecessor of v, then only in the following cases is vw an *admissible* edge in D_i. (In the corresponding figures, reversed edges are dotted, a black-filled vertex is marked, white-filled vertex unmarked, and grey-filled vertex could be either marked or unmarked.)



Theorem 7.2. *A collection of k internally vertex-disjoint $s-t$ paths can be found in $O(k|E|)$ time.*

Proof. A path in a graph can be found in $O(|E|)$ time with a graph search algorithm like *depth-first-search (dfs).* □

Of course Theorem 7.2 assumes that we have easy access to the vertices and edges of the graph. Unfortunately the polytopal digraph $D_c(P)$ is not readily available from our input. Instead we define the function **Find_Path**(D_i) of

Algorithm 7.1 by using an *adjacency oracle* based on the memory-less local search operation of the simplex method, as described in the next section. Then the idea is simple. We use the simplex method to compute the first path $\overrightarrow{P}_1$. We store each vertex of $D_c(P)$ that we encounter on the path $\overrightarrow{P}_1$ in a data structure named R. Thus we have found our first path and *reversed its edges*, abstractly setting $D_1 := D_c(P)$ and $D_2 := D_1 \leftarrow \overrightarrow{P}_1$. For successive iterations we initialize a data structure S to be used for marking nodes in a *dfs* of the vertices. We order the edges incident to each vertex so that given a vertex v and the last edge followed from v, we can define an adjacency oracle to return the next vertex adjacent to v. To find the next path, starting from the source vertex, we repeat the following *dfs* process until the sink is found: if $v \notin S$ add the current vertex v to S, query the *adjacency oracle* and either find a vertex $w \notin S$ where vw is the next *admissible* edge from v in D_i, or backtrack to the *dfs* predecessor of v. The next section describes the adjacency oracle.

7.3 Finding Edges: Simplex Method and Reverse Search

Let v be any vertex of a $d-$polytope P, let Δ be an upper bound on the maximum degree of any vertex in $G(P)$, and let cx be in general position with respect to P. In this section we describe an oracle $Adj(v, j), j = 1, ..., \Delta$, with the following properties:

- $Adj(v, j)$ is either empty or a vertex w of P such that vw is an edge in $G(P)$.

- as j ranges over all possible values, each vertex w adjacent to v appears exactly once.

The polytope P is given as a system of inequalities:

$$\sum_{j=1}^{d} a_{ij}x_j \leq b_i \qquad \text{for } i = 1, ..., n. \tag{7.1}$$

A linear program is formed by maximizing the linear objective function

$$z = cx = \sum_{j=1}^{d} c_j x_j \tag{7.2}$$

over P. By adding slack variables $x_{d+1}, ..., x_{n+d}$, we can write this linear program in *dictionary* form:

$$x_i = b'_i - \sum_{j \in N} a'_{ij} x_j \qquad \text{for } i \in B \tag{7.3}$$
$$z = z' + \sum_{j \in N} c'_j x_j,$$

where initially $N = \{1, 2, ..., d\}, B = \{d+1, d+2, ..., n+d\}, z' = 0, a'_{ij} = a_{ij}, b'_i = b_i, c'_j = c_j$ for all i, j.

The simplex method performs pivots that preserve basic feasibility, and this is achieved by a ratio test to choose the variable to leave the basis. We will only consider feasible pivots, and they are used to define the oracle $Adj(v, j)$. We first discuss the case where P is simple.

If P is simple then each basic feasible solution corresponds to a vertex v of P with a unique basis B. The ratio test is defined as follows:

$$ratio(j) = argmin\{b'_i/a'_{ij} : i \in B, a'_{ij} > 0\}, \tag{7.4}$$

where $argmin$ returns the index i minimizing the given ratio. The corresponding pivot replaces the cobasic variable x_j by the basic variable $x_{ratio(j)}$, and vice versa. Since P is simple, we must have $b'_i > 0$ and the pivot yields a dictionary representing a neighbour w of v. If $c'_j > 0$ then vw is an arc in $D_c(P)$, otherwise if $c'_j < 0$ then wv is an arc in $D_c(P)$. We may set $\Delta = n$ and define the oracle by:

$$Adj(v, j) = w \qquad \text{if} \qquad j \in N \tag{7.5}$$
$$= \emptyset \qquad\qquad \text{otherwise.}$$

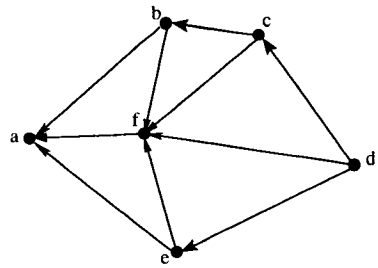


Figure 7–2: Polytopal digraph example

The degenerate case is more challenging. We begin with an example to illustrate the difficulties encountered. Consider the 3-polytope P defined by

$$\begin{aligned} -2x_1 + 2x_2 + x_3 &\leq 2 \\ x_2 + x_3 &\leq 3 \\ 2x_1 + x_2 + 3x_3 &\leq 9 \\ x_1 - 2x_2 + \frac{3}{2}x_3 &\leq 2 \\ -2x_1 - 4x_2 + x_3 &\leq -4 \\ -x_3 &\leq 0. \end{aligned} \tag{7.6}$$

P is a pyramid over a 5-gon with six vertices: $a : (0,1,0), b : (2,3,0), c : (3,3,0), d : (4,1,0), e : (2,0,0), f : (1,1,2)$. Vertex f is degenerate, since it is contained in five facets: the first five inequalities in (7.6) are satisfied as equations. If we choose $c = (-1,0,0)$ the linear function $z = cx = -x_1$ defines the polytopal digraph $D_c(G)$ shown in Figure 7–2.

Vertex f can be represented by the dictionary with cobasis $\{6, 7, 8\}$:

$$\begin{aligned}
x_1 &= 1 + \frac{1}{5}x_6 - \frac{13}{20}x_7 + \frac{3}{8}x_8 \\
x_2 &= 1 - \frac{1}{5}x_6 + \frac{2}{5}x_7 \\
x_3 &= 2 - \frac{2}{5}x_6 + \frac{3}{10}x_7 - \frac{1}{4}x_8 \\
&- - - - - - - - - - - - - - - \\
x_4 &= 0 + \frac{6}{5}x_6 - \frac{12}{5}x_7 + x_8 \\
x_5 &= 0 + \frac{3}{5}x_6 - \frac{7}{10}x_7 + \frac{1}{4}x_8 \\
x_9 &= 2 - \frac{2}{5}x_6 + \frac{3}{10}x_7 - \frac{1}{4}x_8 \\
z &= -1 - \frac{1}{5}x_6 + \frac{13}{20}x_7 - \frac{3}{8}x_8.
\end{aligned} \tag{7.7}$$

Consider for example a pivot on the cobasic variable x_7. The minimum ratio computed by (7.4) is zero and achieved by the basic variables x_4 and x_5. The corresponding pivots are degenerate pivots since they lead to other bases representing f. Observe that no pivot from (7.7) yields the edge af to vertex a whose cobasis is $\{4, 8, 9\}$. The basis graph $B(P)$ is shown in Figure 7–3(a). (In all figures we label vertices by the corresponding cobasic indices, since there are fewer of these than there are basic indices.) Note that vertex f, which is contained on 5 facets, is represented by $\binom{5}{2} = 10$ bases.

The adjacency oracle must return all the edges of $G(P)$ for each vertex v. This is known as the neighbourhood problem. When v is degenerate, we could achieve this by enumerating all the bases of v. The edges of v could then be extracted by considering nondegenerate pivots from each basis of v. Enumerating the bases of a degenerate vertex v can be a daunting task: if v is contained in $k \geq d$ facets, then v can be represented by up to $\binom{k}{d}$ bases. Fortunately a subset of these bases, known as *lex-positive* bases, are sufficient. The lex-positive bases are in one to one correspondence with the vertices of a simple polytope obtained by a perturbation of

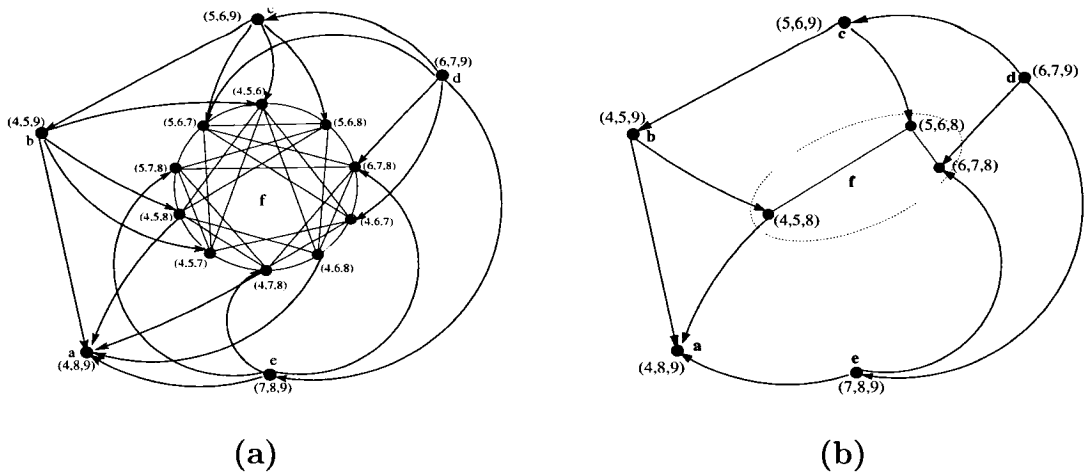


Figure 7–3: (a) Basis graph $B(P)$, and (b) lex-positive subgraph

the inequalities defining P. By McMullen's Upper Bound Theorem [72] this limits the number of lex-positive bases to at most $\Theta(k^{\lfloor \frac{d}{2} \rfloor})$. Whilst a big improvement, this bound is still exponential. The subgraph of $B(P)$ containing just the lex-positive bases is given in Figure 7–3(b). Fortunately, determining all the lex-positive bases can be achieved in a space efficient manner by making use of the *reverse search* method developed by Avis and Fukuda [6]. Its application to finding all the lex-positive bases of a polyhedron is described in [3], and is the basis of the *lrs* code [106] for generating all vertices of a polyhedron. We give just a brief summary here, and then explain how to specialize it for our purposes.

Every vertex of a $d-$polytope P can be represented by a lex-positive basis. In fact the *lex-min basis* of a vertex, defined as the lexicographically smallest subset of indices that represent that vertex, is lex-positive ([3], Proposition 5.2.). A *lexicographic simplex pivot* preserves the lex-positive property of the bases by employing a *lexicographic ratio test* which replaces the test described in (7.4). Note that in case of degeneracy, the test (7.4) may not give a unique index $i \in B$, but this is always guaranteed by the lexicographic ratio test. The *lexicographic pivot rule* applied to a feasible dictionary selects the smallest index j such that $c'_j > 0$ to determine the entering index j, and uses the lexicographic ratio test to

find the leaving index i. If there is no such index j, the dictionary is optimum. There is unique optimum dictionary with lex-positive basis, and this is the lex-min basis for the optimum basis ([3], Proposition 4.3). The lexicographic pivot rule therefore defines a spanning tree on the set of lex-positive bases of P, rooted at the lex-min basis of the optimal vertex. This tree, which contains at least one basis for each vertex of P, can be traversed without additional storage using the reverse search method. We adapt this method to determine all lex-positive bases of any degenerate vertex v. (Filippi [24] details this solution to the neighbourhood problem.)

Algorithm 7.2 (Lexicographic Reverse Search on a Degenerate Vertex). *Designate the lex-min basis of v as the root of a spanning tree T on the lex-positive bases for v, by creating an objective function that is optimal at this basis. There exists a lexicographic simplex pivot path from every lex-positive basis of v to the (optimal) lex-min basis. The collection of these paths define the enumeration tree T of v. Reverse search traces out T in a depth first manner without using additional storage:*

- *To find children of a node $t \in T$ reverse the lexicographic pivot rule: t is a parent of r if the* ***degenerate*** *pivot from r to t satisfies the lexicographic pivot rule.*
- *The parent of a node t in T is found by a single pivot following the lexicographic pivot rule.*

For our example, we show in Figure 7–4 a reverse search tree for f, as a subtree of the full reverse search tree for P, rooted at f. The dotted edges show the degenerate pivots performed by Algorithm 7.2. Note the tree is rooted at the lex-max cobasis for f, $\{6, 7, 8\}$, for which the corresponding basis is lex-min.

We can now fully define the adjacency oracle $Adj(v, j)$. We are given a dictionary with basis B representing a vertex v, and index i of the last cobasic

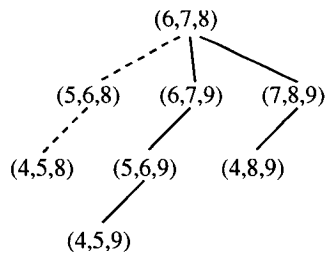


Figure 7–4: Reverse search tree of lex-positive bases rooted at $(6, 7, 8)$

variable pivoted on in B. The adjacency oracle attempts to find the next admissible edge from B by incrementing i until either $i > d$, or the pivot on this column of the dictionary is nondegenerate. In the latter case, this nondegenerate pivot defines a vertex w such that vw is an edge of $G(P)$. The oracle returns $Adj(v, j) = w$ and j is incremented. If $i > d$, then the oracle pivots to the next node in the reverse search tree of v, resets $i = 0$, and repeats this process until the next edge is found or all lex-positive bases for v have been considered.

7.4 Algorithm and Implementation

In this section we give a more complete description of the algorithm and its implementation. To *store* and *identify* a vertex we use the cobasis of its unique lex-min basis as its representative. The top level of the algorithm uses depth first search, *dfs*, to find paths in $D_c(P)$ from source to sink, as described in Section 7.2. We require two basic data structures.

- S: An AVL tree storing the vertices of $D_c(P)$ that have been visited by *dfs*. A node of the *dfs* tree, representing a vertex v of P, will consist of the cobasis of the lex-min basis of the vertex as the search key. Additionally we store information for backtracking, namely a pointer to the to its *dfs* predecessor, and in the case of a degenerate vertex, an encoding of the cobasis, pivot indices, artificial objective function, and depth of the last node considered in the reverse search tree for v. We also store pointers to the left son, right son, and balance factor of the AVL tree.

- R: An AVL tree storing the vertices and edges of $D_c(P)$ that have been reversed. A node of the tree, representing a vertex of P, will consist of the cobasis of the lex-min basis of the vertex as the search key. Additionally we will have two pointers to the two adjacent vertices stored in R (reversed edges associated with this vertex), as well as to the left son, right son, and balance factor of the AVL tree.

By Proposition 7.1, the k disjoint paths found at the k^{th} iteration are represented by vertices stored in R.

Algorithm 7.3 (k Vertex-Disjoint Monotone Paths).

Input: *n inequalities in d variables representing a full dimensional $d-$polytope P, a linear function cx in general position with respect to P, and an integer k.*

Output: *k vertex-disjoint monotone paths in $D_c(P)$ from the source to sink.*

(1) *Initialize data structure R. Find an initial path $\overrightarrow{P}$ using the lexicographic simplex method and store the vertices of $\overrightarrow{P}$ in R ($D_1 := D_0 \leftarrow \overrightarrow{P}$). Iteration $:= 1$.*

(2) *Initialize $S := \emptyset$.*

(3) *Find a new path $\overrightarrow{P}$:* `/*Find_Path(D_i)*/`

(3a) *Set $w :=$ lex-min basis of source, $b := w$ and $i := 0, pred := null$.* ***mark_vertex****$(w, b, i, pred)$.*

(3b) *Set $v := w$. If v is the sink then go to (4), otherwise set $b := v$ and $i := 0$.*

(3c) *$w :=$* ***next_edge_oracle****(v, b, i). If $w == \emptyset$, set $w :=$* ***backtrack****(v) and goto (3b). Else if $w \notin S$, and vw is admissible then* ***update****(v, b, i), set $b := w$, $i := 0$,* ***mark_vertex****(w, b, i, v), and goto (3b).*

(4) *Reverse each vertex p of $\overrightarrow{P}$:* `/*Reverse_Path(D_i, P)*/`

(4a) *If $p \notin R$, add p and the edges adjacent to p in $\overrightarrow{P}$ to $\overrightarrow{P}$ incident to p to R, otherwise remove p from R.*

(4b) *Iteration++. If Iteration == k, then go to (5), otherwise go to (2).*

(5) *Output the vertices and edges in R (up to k paths).* `/*End*/`

next_edge_oracle(v, b, i): `/*The Adjacency Oracle*/`

(a) *Increment i. If* $i > d$ *goto (c).*

(b) *Perform a lexicographic pivot on the* i^{th} *cobasic variable. If the pivot is nondegenerate, return the lex-min basis w of the new vertex. Otherwise goto (a).*

(c) *If v is degenerate, pivot to the next basis* b' *in reverse search tree (Algorithm 7.2). If the reverse search tree is exhausted, or v is nondegenerate then return* $w := \emptyset$. *Otherwise set* $b := b', i := 0$ *and goto (a).*

mark_vertex(v, b, i, w): *If* $v \notin S$ *then add node* $s := \{v, b, i, w\}$ *to* S.

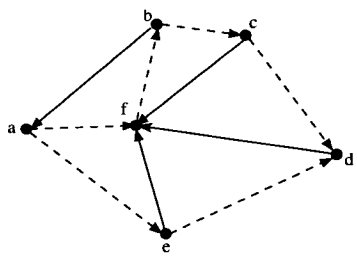
update(v, b, i): *Update b and i and of node storing v in S.*

backtrack(v): *If* $v ==$ *source the goto (5). Otherwise backtrack to predecessor w in dfs search, load the basis b of w and index i last used.*

The output of the algorithm gives a set of strict monotone vertex-disjoint paths from source to sink in the polytopal digraph $D_c(P)$. If P is simple, this is also a set of disjoint strict monotone paths in the basis graph $B(P)$, each edge of which is a pivot. If P is degenerate, we can also efficiently produce a set of disjoint monotone paths in $B(P)$. For each degenerate vertex v contained in a path, we may have two bases: B_{in} representing the dictionary for v when it is entered, and B_{out} when it is left. These two bases can be easily joined by a path in $B(P)$. We make pivots interchanging x_i and x_j, where $i \in B_{out} - B_{in}$ and $j \in B_{in} - B_{out}$. Since these pivots are degenerate, the set of disjoint paths formed in $B(P)$ are monotone.

We illustrate the algorithm on our previous example (7.6). Let the letters associated with each vertex represent the *lex-min* basis at that vertex. Assume that we have already found two source to sink paths, and that the data structures are set as $R = \{e : (a, d), b : (c, f), c : (b, d), f : (a, b)\}$ and $S = \emptyset$. Starting from the

source vertex d represented by cobasis $\{6,7,9\}$ we mark vertex d, $S = \{d : (pred = \emptyset, leave = \emptyset, depth = \emptyset, obj = \emptyset)\}$.



$$
\begin{aligned}
x_1 &= 4 - \tfrac{2}{5}x_6 - \tfrac{1}{5}x_7 - \tfrac{3}{2}x_9 \\
x_2 &= 1 - \tfrac{1}{5}x_6 + \tfrac{2}{5}x_7 \\
x_3 &= x_9 \\
&- - - - - - - - - - - - - \\
x_4 &= 8 - \tfrac{2}{5}x_6 - \tfrac{6}{5}x_7 - 4x_9 \\
x_5 &= 2 + \tfrac{1}{5}x_6 - \tfrac{2}{5}x_7 - x_9 \\
x_8 &= 8 - \tfrac{8}{5}x_6 + \tfrac{6}{5}x_7 - 4x_9 \\
z &= -4 + \tfrac{2}{5}x_6 + \tfrac{1}{5}x_7 + \tfrac{3}{2}x_9
\end{aligned}
$$

The adjacency oracle considers the first nondegenerate pivot from d, pivoting on x_6 and x_8, to vertex $e : \{7,8,9\}$. (d,e) is an arc in $D_c(P)$, as $c'_6 = \frac{2}{5}$ is positive in the $z-$row, but $(d,e) \in R$ and hence not admissible. The oracle examines the next edge, the pivot on x_7 and x_5, to vertex c which is also inadmissible as (d,c) is reversed. The oracle's last option, pivot on x_9 and x_8, is a nondegenerate pivot to vertex f. f is a degenerate vertex and cobasis $\{6,7,8\}$ represents the lex-min basis for f. (d,f) is an arc in $D_c(P)$ that is not reversed. We mark f, $S = \{d, f : (pred = d, leave = \{6,7,8\}, depth = \emptyset, obj = \emptyset)\}$.

$$
\begin{array}{l}
x_1 = 1 + \frac{1}{5}x_6 - \frac{13}{20}x_7 + \frac{3}{8}x_8 \\
x_2 = 1 - \frac{1}{5}x_6 + \frac{2}{5}x_7 \\
x_3 = 2 - \frac{2}{5}x_6 + \frac{3}{10}x_7 - \frac{1}{4}x_8 \\
- - - - - - - - - - - - \\
x_4 = 0 + \frac{6}{5}x_6 - \frac{12}{5}x_7 + x_8 \\
x_5 = 0 + \frac{3}{5}x_6 - \frac{7}{10}x_7 + \frac{1}{4}x_8 \\
x_9 = 2 - \frac{2}{5}x_6 + \frac{3}{10}x_7 - \frac{1}{4}x_8 \\
z = -1 - \frac{1}{5}x_6 + \frac{13}{20}x_7 - \frac{3}{8}x_8 \\
- - - - - - - - - - - - \\
z_{r.s.} = 0 - x_6 - x_7 - x_8,
\end{array}
\qquad
\begin{array}{l}
x_1 = 1 + \frac{13}{14}x_5 - \frac{5}{14}x_6 + \frac{1}{7}x_8 \\
x_2 = 1 - \frac{4}{7}x_5 + \frac{1}{7}x_6 + \frac{1}{7}x_8 \\
x_3 = 2 - \frac{3}{7}x_5 - \frac{1}{7}x_6 - \frac{1}{7}x_8 \\
- - - - - - - - - - - - \\
x_4 = 0 + \frac{24}{7}x_5 - \frac{6}{7}x_6 + \frac{1}{7}x_8 \\
x_7 = 0 - \frac{10}{7}x_5 + \frac{6}{7}x_6 + \frac{5}{14}x_8 \\
x_9 = 2 - \frac{3}{7}x_5 - \frac{1}{7}x_6 - \frac{1}{7}x_8 \\
z = -1 - \frac{13}{14}x_5 + \frac{5}{14}x_6 - \frac{1}{7}x_8 \\
- - - - - - - - - - - - \\
z_{r.s.} = \frac{10}{7}x_5 - \frac{13}{7}x_6 - \frac{19}{14}x_8
\end{array}
$$

$$
\begin{array}{l}
x_1 = 1 + \frac{5}{12}x_4 - \frac{1}{2}x_5 + \frac{1}{12}x_8 \\
x_2 = 1 - \frac{1}{6}x_4 + \frac{1}{6}x_8 \\
x_3 = 2 + \frac{1}{6}x_4 - x_5 - \frac{1}{6}x_8 \\
- - - - - - - - - - - - \\
x_6 = 0 - \frac{7}{6}x_4 + 4x_5 + \frac{1}{6}x_8 \\
x_7 = 0 - x_4 + 2x_5 + \frac{1}{5}x_8 \\
x_9 = 2 + \frac{1}{6}x_4 - x_5 - \frac{1}{6}x_8 \\
z = -1 - \frac{5}{12}x_4 + \frac{1}{2}x_5 - \frac{1}{12}x_8 \\
- - - - - - - - - - - - \\
z_{r.s.} = \frac{13}{6}x_4 - 6x_5 - \frac{5}{3}x_8
\end{array}
$$

We now query the oracle for an outgoing edge from f. The first nondegenerate pivot from $\{6, 7, 8\}$ is x_6 for x_9 to vertex e. Since $c'_6 = -\frac{1}{5}$ is negative and arc (e, f) is not reversed, this pivot is inadmissible. The nondegenerate pivot x_6 for x_9, to vertex d is inadmissible as $d \in S$. As no more nondegenerate pivots are possible from $\{6, 7, 8\}$ the oracle initiates a reverse search on the lex-positive bases of f, setting the lex-max cobasis $\{6, 7, 8\}$ as the root of a reverse search tree (refer to Figure 5). We construct an artificial objective function $z_{r.s.}$, and also carry along the original objective function, although this is not used in pivot selection during this phase. Following Algorithm 7.2, the next cobasis attained is $\{5, 6, 8\}$. We re-instate the LP's objective function.

From this cobasis, the only nondegenerate pivot finds vertex c which is inadmissible since arc (c, f) of $D_c(P)$ is not reversed. The oracle restarts the reverse search from $\{5, 6, 8\}$ and finds the next cobasis $\{4, 5, 8\}$ of degenerate vertex f. The first nondegenerate pivot from $\{4, 5, 8\}$ finds sink vertex a, however we check R and

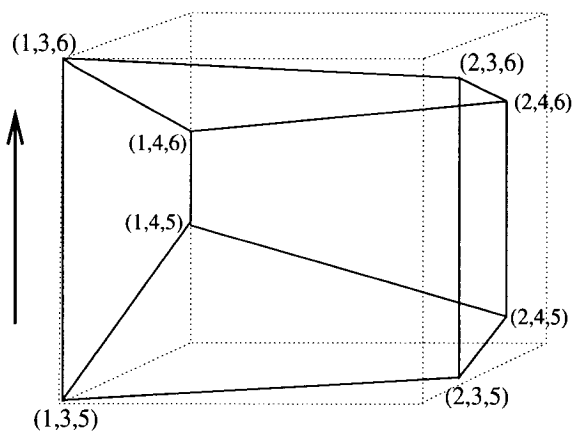
find that the arc (f, a) in $D_c(P)$ is reversed. The other nondegenerate pivot from $\{4, 5, 8\}$ yields an admissible edge to vertex b. We update $S = \{d, f : (pred = d, leave = \{4, 5, 8\}, depth = 2, obj = z_{r.s.})\}$. The algorithm continues and finds an edge to vertex a. $S = \{d, f : (pred = d), b : (pred = f), a : (pred = b)\}$. Following the predecessor links in S we reverse the edges along the path taken, storing only the arcs not in $D_c(P)$. $R = \{e : (a, d), b : (a, c), c : (b, d), f : (a, d)\}$.

7.5 Computational Results & Complexity

We have implemented the k vertex-disjoint monotone paths algorithm as the program *disjointlp*, using library functions of *lrs* [106] for lexicographic pivoting and rational arithmetic. All computations are done exactly using extended precision arithmetic. The program and a user guide are available [108]. The input files are in standard *polyhedra format* ([106]), and the program outputs k vertex-disjoint monotone paths, each as a sequence of vertices (represented by the cobasis of the lex-min basis) from source to sink. For example, the Klee-Minty cube LP in 3D has the following input and output:

Input:

```
H-representation
begin
6 4 rational
0 1 0 0
1 -1 0 0
0 -1 3 0
3 -1 -3 0
0 0 -1 3
3 0 -1 -3
end
disjoint 3
maximize 0 0 0 1
```



Output:

```
Disjoint Paths Computed:
Path 1:  [1 3 5 ]-->[2 3 5 ]-->[2 3 6 ]-->[1 3 6 ]
Path 2:  [1 3 5 ]-->[1 4 5 ]-->[1 4 6 ]-->[1 3 6 ]
Path 3:  [1 3 5 ]-->[1 3 6 ]

Disjoint path statistics:
      Longest path length = 4          Max # reversed nodes:  6
      Shortest path length = 2         Avg # reversed nodes:  4.67
      # of path vertices = 6           Max # marked nodes:  7
      Median path length = 4           Avg # marked nodes:  2.67
                                       Total # of pivots:  39
```

The only assumption is that input inequalities define a full dimensional polytope P. The objective function does not need to be in general position with respect to P, as dual degeneracy is broken by lexicographic ordering of the cobasic indices. For a dual degenerate edge, vw is an arc if lex-min basis of v is lexicographically smaller than the lex-min basis of w.

7.5.1 Computational Results

The implementation was used to compute a set of disjoint paths of maximum cardinality for various LP's both nondegenerate and degenerate. For each problem we list the number of vertices and lex-positive bases that the polytope has, the amount of additional memory used on top of storing the vertices of the disjoint paths, the path lengths (# vertices), the total number of pivots operations from the time the input is loaded, and the number k of disjoint paths computed. Apart from the *Klee-Minty* LP's, we generated random objective functions by picking coefficients from the set $\{-1000, ..., 1000\}$. Table 7–1 presents results from nondegenerate inputs.

The worst case for our algorithm is illustrated by the extremely degenerate examples in Table 7–2. The polytopal graphs for all are highly connected (e.g. the cut polytope's graph is a complete graph), so the problem of finding vertex-disjoint paths is in fact trivial. There are relatively few vertices, but these are extremely degenerate, making basis enumeration impractical.

Table 7–1: Nondegenerate examples

name	**n**	**d**	**k**	**|V|**	**memory** max, avg	**path lengths** max, min, med	**# pivots**
Hypercube	10	5	5	32	16, 8.2	6, 6, 6	182
Hypercube	16	8	8	256	36, 20.0	9, 9, 9	899
Klee-Minty LP	16	8	8	256	254, 161.1	72, 2, 23	6078
Klee-Minty LP	18	9	9	512	510, 352.7	114, 2, 48	16225
Klee-Minty LP	20	10	10	1024	1022, 730.4	282, 2, 67	37736
Klee-Minty LP	30	15	15	32768	32766, 23953.5	4310, 2, 1284	1988459
Polar Cyclic	20	5	5	272	121, 67.7	17, 10, 12	1691
Polar Cyclic	50	10	10	1357510	199000, 21479.9	70, 28, 32	1962605
Polar Cyclic	60	10	10	3795012	709380, 123209.0	171, 29, 150.5	10949702
Polar Cyclic	65	10	10	5916638	1775373, 283883.9	330, 35, 125	26767383

Three random LP models were used (Table 7–3). The *Kuhn & Quandt* polytopes [65] were constructed by generating inequalities of the form $\sum_{j=1,...,d} a_{ij}x_j \leq 10,000$ with a_{ij} chosen randomly from the set $\{0, ..., 1000\}$, and then adding nonnegativity constraints $x_j \geq 0, j = 1, ..., d$. Examples named *Random* were constructed by generating inequalities of the form $\sum_{j=1,...,d} a_{ij}x_j \leq 1$ with a_{ij} chosen randomly from the set $\{-1000, ..., 1000\}$.

The observed time taken by *disjointlp* to compute k disjoint paths does not increase linearly with every iteration $k = 1, ..., d$, as illustrated in Table 7–4. Computing the first few paths is comparable to the time taken to solve the LP with the simplex method, while computing the last few paths is more comparable to the time taken to enumerate all the vertices via known pivot techniques, for example using *lrs* [3]. Consider the *Kuhn-Quandt*, $d = 10, n = 60, |V| = 21044$, example on which *lrs* takes 21427 pivots to enumerate all the vertices (Table 7–4).

7.5.2 Complexity

Let k be the number of disjoint paths, $|V|$ the number of vertices in $G(P)$, $|B|$ the number of lex-positive bases of the LP, d the dimension, and n the number of

Table 7–2: Degenerate examples

name	**n**	**d**	**k**	**|V|**	**|B|**	**memory** max, avg	**path lengths** max, min, med	**# pivots**
Cut4	16	6	7	8	80	4, 2.4	3, 2, 3	1874
Cut5	56	10	15	16	57498	7, 3.8	3, 2, 3	5102007
Metric4	16	6	7	8	80	5, 2.5	3, 2, 3	1374
Metric5	40	10	21	32	9184	15, 6.8	3, 2, 3	3599164
Cross5	32	5	8	10	240	2, 1.8	3, 3, 3	948
Cross6	64	6	10	12	1440	2, 1.8	3, 3, 3	7255
Cross7	128	7	12	14	10080	2, 1.8	3, 3, 3	77678
Cyclic	20	3	11	12	36	3, 2.4	3, 2, 3	343
Cyclic	40	7	9	10	1206	2, 1.6	3, 2, 3	8566
Cyclic	240	7	13	14	95686	2, 1.7	3, 2, 3	962780
Cyclic	378	10	14	15	2795469	2, 1.9	3, 2, 3	287897132

Table 7–3: Random examples

name	**n**	**d**	**k**	**|V|**	**memory** max, avg	**path lengths** max, min, med	**# pivots**
Kuhn-Quandt	87	7	7	2326	72, 24.9	19, 8, 14	628
Kuhn-Quandt	60	10	10	21044	2237, 618.8	42, 16, 23.5	48141
Kuhn-Quandt	100	15	15	5029250	90071, 7563.87	66, 9, 15	1306273
Kuhn-Quandt	115	15	15	14699038	35138, 3316.8	82, 25, 47	531817
Random	50	5	5	538	204, 82.6	27, 12, 19	1776
Random	60	7	7	7530	323, 108.9	52, 31, 35	3362
Random	70	10	10	291566	527, 200.7	65, 51, 54.5	12032
Random	80	15	15	96936060	890, 320.3	114, 75, 90	36904

inequalities in the input. We assume the bit-vector computational model allows single-operation arithmetic. When the LP is nondegenerate, $|B| = |V|$.

Lemma 7.1. *The k vertex-disjoint monotone paths algorithm takes $O(kd^2|B|)$ pivots, and only $O(k|V|)$ pivots when the LP is nondegenerate*

Proof. Pivots are performed during *dfs* of $D_c(P)$, to extend the search path and also to backtrack. For every iteration, the *dfs* may build a spanning tree across all V vertices which has $|V| - 1$ edges. Each edge along this tree is traversed at most once in each direction. Similarly for every degenerate vertex, we may have to

Table 7–4: Growth in number of pivots

# of paths	memory max, avg	path lengths max, min, med	# pivots
$k = 1$	$0, 0.0$	$13, 13, 13$	26
$k = 2$	$22, 11.0$	$18, 13, 15.5$	164
$k = 3$	$53, 25.0$	$19, 12, 15$	367
$k = 4$	$66, 35.3$	$26, 8, 21$	697
$k = 5$	$147, 57.6$	$34, 7, 14$	1730
$k = 6$	$391, 113.2$	$36, 8, 21$	4333
$k = 7$	$537, 173.7$	$29, 10, 22$	8023
$k = 8$	$637, 231.62$	$27, 14, 21.5$	12608
$k = 9$	$2098, 439.9$	$45, 7, 22$	29784
$k = 10$	$2237, 618.8$	$42, 16, 23.5$	48141

enumerate all the lex-positive bases via reverse search. If a vertex v has $|B^v|$ bases, then the reverse search tree at v has $|B^v| - 1$ edges and each edge is traversed a maximum of two times. The number of pivots required to find the lex-min basis of a degenerate vertex is $O(d)$ and we may have to do this $O(d|B|)$ times. □

The number of operations required for a single pivot is $\theta(nd)$, and so $O(knd^3|B|)$ time is spent pivoting, $O(knd|V|)$ when $|V| = |B|$. Carefully constructed worst-case examples for simplex methods, such as [2] and [61], illustrate that $\Omega(|V|)$ pivots may have to be taken to find a single path from source to sink. However [79] proves that the number of vertices along a strict monotone path is strictly less than $|V|$.

Lemma 7.2. *The time spent on AVL tree search is* $O(kd^2|V| \log |V|)$.

Proof. For each iteration, the algorithm may need to perform an AVL search for every edge of $D_c(P)$ of which there can be $\frac{1}{2}|V|d$. The maximum number of nodes in each AVL tree is $O(|V|)$ and so the search time is $O(\log |V|)$. The time taken for comparison of keys in the AVL tree is $O(d)$ as we compare the indices of cobases. □

Lemma 7.3. *The k vertex-disjoint monotone paths algorithm requires $O(d|V|)$ space.*

Proof. In the worst-case all the vertices in $D_c(P)$ will be either marked during *dfs*, or elements of the disjoint paths computed. The storage cost of a vertex is $O(d)$. □

Theorem 7.3. *A collection of k internally vertex-disjoint monotone $s-t$ paths in a $D_c(P)$ can be found in $O(knd^3|B|)$ time ($O(knd|V|)$ when the P is simple) and $O(d|\widetilde{V}_{i-1}| + d|\overline{V}_i|)$ space per iteration i where $\widetilde{V}_0$ is the empty set, $\widetilde{V}_{i-1}$ is the set of vertices along the $i-1$ disjoint paths found so far, and $\overline{V}_i$ is the current set of vertices marked by the search path in iteration i.*

When the polytope is simple, or near simple, the theoretical time for searching the AVL trees slightly dominates the time spent pivoting in the theoretical analysis. However experimental observations show that the k vertex-disjoint monotone paths algorithm is memory efficient and little time is spent searching AVL trees. Time is spent pivoting and so the this approach to computing disjoint paths works favourably when the LP is nondegenerate or contains little degeneracy. Pivoting to compute all disjoint paths is much less effective when the input is highly degenerate, as expected, since pivot methods for vertex enumeration perform similarly in the presence of degeneracy [4]. The complexities of our algorithm are slightly worse than that of enumerating all the vertices and edges, explicitly storing $D_c(P)$ as a edge adjacency list, and employing a network flow algorithm on the stored digraph. However in practice *disjointlp* frequently computes disjoint paths faster than state-of-the-art vertex enumeration codes can enumerate all the vertices, especially if just a few disjoint paths are required (see Table 7–5).

Open Problem 7.1. *Do there exist d vertex-disjoint simplex method pivot paths from source to sink of $D_c(P)$?*

Table 7–5: Computing just a few paths

name	n	d	k	|V|	pivots	max. mem.
Cyclic	1782	10	7	18	2829	6
Cyclic	4004	10	7	20	6227	8
Polar Cyclic	65	10	7	5,916,638	30121	3276
Kuhn-Quandt	115	15	8	14,699,038	1439	129

(By the Holt-Klee Condition, the answer is yes when the linear program is nondegenerate.)

Open Problem 7.2. *Given a vertex v, basis B_u of a vertex u, and B_w of a vertex w such that (u, v) and (v, w) are directed edges of $D_c(P)$, is there a simplex method pivot path from B_u through the basis graph of vertex v to basis B_w?*
(This would help answer the previous question.)

Open Problem 7.3. *Does there exist a polytope and objective function such that d vertex-disjoint paths from source to sink of $D_c(P)$ each have length exponential in n and d?*

CHAPTER 8
Conclusion

The main contributions of this thesis are the following:

- In Chapter 2 we define products of arrangements and construct linear programs on which the simplex method cycles, achieving new bounds on cycle lengths. We show that $S(n,2)$ is $\Theta(n^2)$ and more generally that $S(n,m)$ is $\Theta(n^m)$ for $n \geq 3m$ with $m \geq 2$ fixed and even. ($S(n,m)$ is the maximal length of a simplex method cycle for a linear program with n nonnegative variables and m inequalities.)
- In Chapter 3 we show that Bland's rule is not a perturbation scheme leading to the possibility that the number of degenerate bases visited by the simplex method with Bland's rule, $D_{Bl}(d,n)$, can exceed the maximal number of vertices that a $d-$polytope with n facets can have. However we prove that $D_{Bl}(2,n) = n$ and $D_{Bl}(3,n) = O(n^2)$. Our proofs are based on a new geometric interpretation of (simplex method) dictionary coefficients using determinants (Section 1.2.5).
- In Chapter 4 we construct the longest admissible pivot path possible: we prove that the least-index criss-cross method can take $\Theta(n^d)$ pivots for $n \geq 2d$. We extend the notion of deformed products of polytopes to oriented hyperplane arrangements by defining *deformed products of arrangements.*

- In Chapter 5 we construct a family of examples proving that $D_{Bl}(d,n) = \Omega(n^{\lfloor \frac{d}{2} \rfloor})$ for $n \geq 2d$, and $D_{Bl}(d,n) = \Theta(n^{n-d})$ for $d \leq n \leq \frac{4d}{3}$. We define *collapsing deformed product* programs.
- In Chapter 6 we present software for pivot path enumeration. We compute the finite number of objective functions needed to enumerate all possible pivot paths taken by the simplex method following a combinatorial pivot rule to an optimal basis. We present a pivot path enumeration algorithm and implementation that allows us to compute the objective function and starting basis yielding the longest pivot path on user inputed polyhedra.
- In Chapter 7 we present a new algorithm and useful implementation for computing disjoint paths in polytopes. We present empirical results displaying its behaviour on nondegenerate and degenerate input.

Our research has lead to new open problems, which have been listed at the end of each chapter.

Appendix

To prove Theorem 1.6 we venture into the matrix representation of a linear program,

$$\begin{aligned} &\text{maximize } \overline{c}x \\ &\text{subject to } \overline{A}x = b, \text{ with } x_{d+1}, ..., x_{d+n} \geq 0 \end{aligned} \tag{8.1}$$

for matrix $\overline{A} \in \Re^{n \times (d+n)}$, column vector $b \in \Re^n$, and row vector $\overline{c} \in \Re^d$, and a dictionary,

$$\begin{aligned} x_B &= \overline{A}_B^{-1} \cdot b - \overline{A}_B^{-1} \cdot \overline{A}_N x_N \\ z &= \overline{c}_B \cdot \overline{A}_B^{-1} + (\overline{c}_N - \overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N) x_N \end{aligned} \tag{8.2}$$

for basis B and cobasis N. If $\overline{A} = \begin{bmatrix} A & I \end{bmatrix}$ with $A \in \Re^{n \times d}$, $\overline{c} = \begin{bmatrix} c & 0 \end{bmatrix}$ with $c \in \Re^d$, and $x = \begin{bmatrix} x & x^s \end{bmatrix}$ with $x = \begin{bmatrix} x_1 & x_2 & ... & x_d \end{bmatrix}$, $x^s = \begin{bmatrix} x_{d+1} & x_{d+2} & ... & x_{d+n} \end{bmatrix}$ then (8.1) is equivalent to

$$\begin{aligned} &\text{maximize } cx \\ &\text{subject to } Ax \leq b, \end{aligned} \tag{8.3}$$

which is the matrix form of (1.32). We need some elementary definitions and observations:

- The cobasic gradient matrix G_N is the $d \times d$ matrix where row $i' = i - d$ for $i \in N$ corresponds to row i' of matrix A.
- Let D^S be the submatrix of matrix D consisting of the rows $s \in S$ of D.
- Let D be a $m \times m$ matrix. Define $D^{i,j}$ to be the $m-1 \times m-1$ matrix obtained by deleting the i^{th} row and j^{th} column of D.

Remark 8.1. *Let D be a nonsingular matrix, then $D^{-1} = \frac{adj(D)}{|D|}$ where $adj(D) = [(-1)^{i+j}\alpha_{ij}]^T$ and $\alpha_{ij} = |D^{i,j}|$ (see [89]).*

- If we restrict indices $1, ..., d$ to the basis B then $\overline{A}_N$ will consists of d linearly independent columns e^T_{j-d} for $j \in N$, and the product $\overline{A}_B^{-1} \cdot \overline{A}_N = \frac{adj(\overline{A}_B)}{|\overline{A}_B|}\overline{A}_N$ appearing in dictionary (8.2) will consist of d columns of $\frac{adj(\overline{A}_B)}{|\overline{A}_B|}$, namely the columns with indices $j' = j - d$ for $j \in N$.
- Let $B' = B \setminus \{1, ..., d\}$.
- Let E be the $n \times (n-d)$ matrix whose $n-d$ columns are e^T_{j-d} for $j \in B'$, then $\overline{A}_B = [A : E]$.
- Order the sets B' and N, and let $p(k)$ be the position of k in B' if $k \in B$, otherwise the position of k in N if $k \in N$.

By definition $-a'_{rs} = -\left[\overline{A}_B^{-1} \cdot \overline{A}_N\right]_{p(r)+d,s-d}$ and $c'_s = (\overline{c}_N - \overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N)_{s-d}$. The following three lemmas dissect $-a_{rs}$ and c'_s.

Lemma 8.1. $\left[\overline{A}_B^{-1} \cdot \overline{A}_N\right]_{p(r)+d,s-d} = \frac{(-1)^{\xi}\left|{}^{a_{r-d}}_{G_{N\setminus s}}\right|}{|\overline{A}_B|}$ *for some $r \in B', s \in N$ and some positive constant ξ.*

(Note that the row with basic variable $x_r, r \in B'$, corresponds to row $p(r)+d$ of (1.33), and that the column with variable $x_s, s \in N$, corresponds to column $s-d$ of (1.33)).

Proof. $|\overline{A}_B| \cdot \left[\overline{A}_B^{-1} \cdot \overline{A}_N\right]_{p(r)+d,s-d} = \left[adj(\overline{A}_B) \cdot \overline{A}_N\right]_{p(r)+d,s-d}$ (by Remark 8.1). The latter is equal to row $p(r)+d$ of $adj(\overline{A}_B)$ multiplied by column $s-d$ of $\overline{A}_N$ which is equal to $adj(\overline{A}_B)_{p(r)+d,s-d}$.

$$\begin{aligned} adj(\overline{A}_B)_{p(r)+d,s-d} &= (-1)^{p(r)+d+s-d}\left|[\overline{A}_B]^{s-d,p(r)+d}\right| \\ &= (-1)^{p(r)+s}\left|[A:E]^{s-d,p(r)+d}\right|. \end{aligned} \tag{8.4}$$

$[A : E]^{s-d,p(r)+d}$ is matrix $[A : E]$ but with cobasic row s removed (it had index $s-d$), and column with index $p(r)+d$ removed (column e^T_{r-d}). Now rearrange the

rows of $[A:E]^{s-d,p(r)+d}$:

$$\text{to} \begin{bmatrix} A^{N\setminus s} & 0 \\ A^{B'} & I \end{bmatrix}, \text{ requiring } \sum_{\substack{k\in N,\\ k\neq s}} (k-d-p(k)) \text{ shifts,} \tag{8.5}$$

$$\text{then to} \begin{bmatrix} A^{N\setminus s} & 0 \\ a_{r-d} & 0 \\ A^{B'} & I \end{bmatrix}, \text{ requiring } p(r)+1 \text{ shifts,} \tag{8.6}$$

$$\text{then to} \begin{bmatrix} a_{r-d} & 0 \\ A^{N\setminus s} & 0 \\ A^{B'} & I \end{bmatrix}, \text{ requiring } d-1 \text{ shifts.} \tag{8.7}$$

Hence $\left|\overline{A}_B\right| \cdot \left[\overline{A}_B^{-1} \cdot \overline{A}_N\right]_{p(r)+d,s-d} = (-1)^{\xi} \left| \begin{smallmatrix} a_{r-d} \\ G_{N\setminus s} \end{smallmatrix} \right|$ with

$$\begin{aligned} \xi &= p(r) + s + \sum_{\substack{k\in N,\\ k\neq s}} (k-d-p(k)) + (p(r)+1) + (d-1) \\ &= \sum_{\substack{k\in N,\\ k\neq s}} (k-d-p(k)) + s + d + 2p(r). \end{aligned} \tag{8.8}$$

□

Lemma 8.2. $\left[\overline{c}_N - \overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N\right]_{s-d} = \frac{(-1)^{\zeta} \left| \begin{smallmatrix} c \\ G_{N\setminus s} \end{smallmatrix} \right|}{\left|\overline{A}_B\right|}$ *for some* $s \in N$ *and some positive constant* ζ.

Proof. First note that $\left[\overline{c}_N - \overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N\right]_{s-d} = -\left[\overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N\right]_{s-d}$ since $\overline{c}_N$ is a $n-$dimensional row vector of zero's as $\{1,...,d\} \subset B$. Furthermore $\overline{c}_B = \begin{bmatrix} c & 0 \end{bmatrix}$ so the last $n-d$ elements of $\overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N$ are zero. By Remark 8.1, $\left|\overline{A}_B\right| \cdot \left[\overline{A}_B^{-1} \cdot \overline{A}_N\right]_{s-d} = \left[adj(\overline{A}_B) \cdot \overline{A}_N\right]_{s-d}$ and we are specifically interested in the element $j = s-d$ of $\overline{c}_B \cdot adj(\overline{A}_B) \cdot \overline{A}_N$ which is row vector c multiplied by the column

j of $adj(\overline{A}_B)$. The column $j = s - d$ of $adj(\overline{A}_B)$ has form

$$\begin{bmatrix} (-1)^{s-d+1}\left|\left[\overline{A}_B\right]^{s-d,1}\right| \\ (-1)^{s-d+2}\left|\left[\overline{A}_B\right]^{s-d,2}\right| \\ \vdots \\ (-1)^{s-d+d}\left|\left[\overline{A}_B\right]^{s-d,d}\right| \\ \vdots \\ (-1)^{s-d+n-d}\left|\left[\overline{A}_B\right]^{s-d,n}\right| \end{bmatrix}. \tag{8.9}$$

$$\begin{aligned} \left[\overline{c}_B \cdot adj(\overline{A}_B) \cdot \overline{A}_N\right]_{s-d} &= c_1(-1)^{s-d+1}\left|\left[\overline{A}_B\right]^{s-d,1}\right| \\ &+ c_2(-1)^{s-d+2}\left|\left[\overline{A}_B\right]^{s-d,2}\right| \\ &+ ... \\ &+ c_d(-1)^{s-d+d}\left|\left[\overline{A}_B\right]^{s-d,d}\right| \end{aligned} \tag{8.10}$$

$$= (-1)^{s-d+1}\left(c_1\left|[A:E]^{s-d,1}\right| - c_2\left|[A:E]^{s-d,2}\right| + ... - ... + c_d(-1)^{d-1}\left|[A:E]^{s-d,d}\right|\right)$$

$$= (-1)^{s-d+1}\left|\begin{bmatrix} c_1 & c_2 & \cdots & c_d \\ [A:E]^{s-d,1} & [A:E]^{s-d,2} & \cdots & [A:E]^{s-d,d} \end{bmatrix}\right|$$

$$= (-1)^{s-d+1}(-1)^{\delta}\begin{vmatrix} c & 0 \\ A^{N\setminus s} & 0 \\ A^{B'} & I \end{vmatrix}, \text{ with } \delta = \sum_{\substack{k\in N,\\ k\neq s}}(k-d-p(k)) \text{ (see (8.5))},$$

showing that

$$\begin{aligned} \left|\overline{A}_B\right| \cdot \left[\overline{c}_N - \overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N\right]_{s-d} &= -\left|\overline{A}_B\right| \cdot \left[\overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N\right]_{s-d} \\ &= (-1)^{\zeta}\begin{vmatrix} c \\ G_{N\setminus s} \end{vmatrix} \text{ for } \zeta = s - d + \sum_{\substack{k\in N,\\ k\neq s}}(k-d-p(k)). \end{aligned}$$

□

Lemma 8.3. $\left|\overline{A}_B\right| = (-1)^\gamma \left|\begin{smallmatrix} a_{s-d} \\ G_{N\setminus s}\end{smallmatrix}\right|$ *for* $s \in N$ *and some positive constant* γ.

Proof. $\overline{A}_B = [A : E]$ consists of n rows: basic rows with index $i' = i - d$ for $i \in B'$, and cobasic rows with index $j' = j - d$ for $j \in N$. If we rearrange these rows, shifting the cobasic rows to the top we get $|[A : E]| = (-1)^\alpha \begin{vmatrix} A^N & 0 \\ A^{B'} & I \end{vmatrix}$, where α represents the number of row exchanges required:

$$\alpha = \sum_{k\in N}(k - d - p(k)). \tag{8.11}$$

We can then isolate cobasic row s at the top by shifting it $p(s) + 1$ times to get

$$|[A : E]| = (-1)^{\alpha+p(s)+1} \begin{vmatrix} A^{\{s-d\}} & 0 \\ A^{N\setminus s} & 0 \\ A^{B'} & I \end{vmatrix} = (-1)^\gamma \left|\begin{smallmatrix} a_{s-d} \\ G_{N\setminus s}\end{smallmatrix}\right| \text{ where } \gamma = \alpha + p(s) + 1. \qquad \square$$

Proof of Theorem 1.6.

- By definition $-a'_{rs} = -\frac{\left[adj(\overline{A}_B)\cdot\overline{A}_N\right]_{p(r)+d,s-d}}{\left|\overline{A}_B\right|}$, and by Lemma 8.1 and Lemma 8.3

$$-a'_{rs} = -\frac{\left[adj(\overline{A}_B)\cdot\overline{A}_N\right]_{p(r)+d,s-d}}{\left|\overline{A}_B\right|} = -\frac{(-1)^\xi \left|\begin{smallmatrix} a_{r-d} \\ G_{N\setminus s}\end{smallmatrix}\right|}{(-1)^\gamma \left|\begin{smallmatrix} a_{s-d} \\ G_{N\setminus s}\end{smallmatrix}\right|} \tag{8.12}$$

where

$$\gamma = \sum_{k\in N}(k - d - p(k)) + p(s) + 1 \tag{8.13}$$

$$= \sum_{\substack{k\in N,\\ k\neq s}}(k - d - p(k)) + (s - d - p(s)) + p(s) + 1,$$

$$\xi = \sum_{\substack{k\in N,\\ k\neq s}}(k - d - p(k)) + s + d + 2p(r). \tag{8.14}$$

Adding (8.13) and (8.14) we get that $(\gamma + \xi) \bmod 2 = 1$, proving that $-a'_{rs} = \frac{\left|\begin{smallmatrix} a_{r-d} \\ G_{N\setminus s}\end{smallmatrix}\right|}{\left|\begin{smallmatrix} a_{s-d} \\ G_{N\setminus s}\end{smallmatrix}\right|}$.

- Similarly $c'_s = (\overline{c}_N - \overline{c}_B \cdot \overline{A}_B^{-1} \cdot \overline{A}_N)_{s-d}$, and by Lemma 8.2 and Lemma 8.3

$$c'_s = \frac{\left[\overline{c}_B \cdot adj(\overline{A}_B) \cdot \overline{A}_N\right]_{s-d}}{\left|\overline{A}_B\right|} = \frac{(-1)^{\zeta} \left|{}^{c}_{G_{N\setminus s}}\right|}{(-1)^{\gamma} \left|{}^{a_{s-d}}_{G_{N\setminus s}}\right|} \tag{8.15}$$

where

$$\gamma = \sum_{k\in N}(k-d-p(k)) + p(s) + 1 \tag{8.16}$$

$$= \sum_{\substack{k\in N,\\ k\neq s}}(k-d-p(k)) + (s-d-p(s)) + p(s) + 1,$$

$$\zeta = s - d + \sum_{\substack{k\in N,\\ k\neq s}}(k-d-p(k)). \tag{8.17}$$

Adding (8.16) and (8.17) we get that $(\gamma + \zeta) \bmod 2 = 1$, proving that $c'_s = -\frac{\left|{}^{c}_{G_{N\setminus s}}\right|}{\left|{}^{a_{s-d}}_{G_{N\setminus s}}\right|}$.

□

Example 8.1.

$$If\ \overline{A} = \begin{bmatrix} a_{11} & a_{12} & 1 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 1 & 0 & 0 & 0 \\ a_{31} & a_{32} & 0 & 0 & 1 & 0 & 0 \\ a_{41} & a_{42} & 0 & 0 & 0 & 1 & 0 \\ a_{51} & a_{52} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix},$$

$$\overline{c} = [\ c_1 \quad c_2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]$$

with $B = \{1, 2, 4, 5, 7\}$, *and* $N = \{3, 6\}$, *then*

$$adj(\overline{A}_B) \cdot \overline{A}_N = \begin{bmatrix} a_{42} & -a_{12} \\ -a_{41} & a_{11} \\ -\begin{vmatrix} a_{21} & a_{22} \\ a_{41} & a_{42} \end{vmatrix} & \begin{vmatrix} a_{21} & a_{22} \\ a_{11} & a_{12} \end{vmatrix} \\ -\begin{vmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \end{vmatrix} & \begin{vmatrix} a_{31} & a_{32} \\ a_{11} & a_{12} \end{vmatrix} \\ -\begin{vmatrix} a_{51} & a_{52} \\ a_{41} & a_{42} \end{vmatrix} & \begin{vmatrix} a_{51} & a_{52} \\ a_{11} & a_{12} \end{vmatrix} \end{bmatrix},$$

and, for example,

$$-a'_{53} = \frac{\begin{vmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{41} & a_{42} \end{vmatrix}}, c'_3 = -\frac{\left[\begin{bmatrix} c_1 & c_2 & 0 & 0 & 0 \end{bmatrix} adj(\overline{A}_B)\overline{A}_N\right]_1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{41} & a_{42} \end{vmatrix}} = -\frac{\begin{vmatrix} c_1 & c_2 \\ a_{41} & a_{42} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{41} & a_{42} \end{vmatrix}}.$$

References

[1] I. Adler and N. Megiddo, A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension, *Journal of the Association of Computing Machinery* **32** (1985) 891–895.

[2] N. Amenta and G. Ziegler, Deformed products and maximal shadows of polytopes, *Contemporary Mathematics* **223** (1999) 57–90.

[3] D. Avis, lrs: a revised implementation of the reverse search vertex enumeration problem, In: G. Kalai & G. Ziegler eds., *Polytopes - Combinatorics and Computation*, Birkhauser-Verlag, DMV Seminar Band **29**, (2000) 177–198.

[4] D. Avis, D. Bremner, and R. Seidel, How good are convex hull algorithms, *ACM Symposium on Computational Geometry* (1995) 20–28.

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

[6] D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Applied Math* **6**, (1996) 21–46.

[7] D. Avis, B. Kaluzny, Solving inequalities and proving Farkas' lemma made easy, *Amer. Math. Monthly* **111** (2004), no. 2, 152–157.

[8] D. Avis, B. Kaluzny, Computing disjoint paths on polytopes, *GERAD Technical Report* **G-2005-26**. Submitted to Journal of Combinatorial Optimization (special issue on Franco-Canadian Workshop on Combinatorial Algorithms, McMaster, August 18-20, 2005).

[9] D. Avis, B. Kaluzny, and D. Titley-Péloquin, Visualizing and constructing cycles in the simplex method, *GERAD Technical Report* **G-2005-33**. Submitted to Journal of Operations Research.

[10] M. Balinski, A. Tucker, Duality theory of linear programs - a constructive approach with applications. *SIAM Review* **11** (3) (1969) 347–77.

[11] E. Beale, Cycling in the dual simplex method, *Naval Research Logistics Quarterly* **2** (4) (1955) 269–75.

[12] R. Bixby, Solving real-world linear programs: a decade and more of progress, *Operations Research 50th Anniversary Issue* **50** (1) (2002) 3–15.

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

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

[15] H. Borgwardt, *The Simplex Method: A Probabilistic Analysis.* Algorithms and Combinatorics, Vol. 1 Springer-Verlag 1987.

[16] E. Boyd, Resolving degeneracy on combinatorial linear programs, *Mathematical Programming* **68** (1995) 155–168.

[17] A. Charnes, Optimality and degeneracy in linear programming, *Econometrica* **20** (2) (1952) 160–170.

[18] V. Chvátal, *Linear Programming*, Freeman, 1980.

[19] G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton 1963.

[20] G. Dantzig, A. Orden, and P. Wolfe, Notes on linear programming: part I - the generalized simplex method for minimizing a linear form under linear inequality restrains, *Pacific Journal Mathematics* **5** (2) (1955) 183–195.

[21] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin Heidelberg 1987.

[22] J. Farkas, Theorie der einfachen Ungleichungen, *Journal für die reine and angewandte Mathematik* **124** (1902) 1–27.

[23] J. Ferrez, K. Fukuda, T. Liebling, Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm, *European Journal of Operations Research,* **166** (1) (2005), 35–50.

[24] C. Filippi, A reverse search algorithm for the neighbourhood problem, *Oper. Res. Letters* **25** (1999) 33–37.

[25] L. Ford and D. Fulkerson, Maximal flow through a network, *Canad. J. Math.* **8** (1956) 1142–1146.

[26] J. Fourier, Solution d'une question particulière du calcul des inégalités, 1826, and extracts from "Histoire de l'Académie," 1823, 1824, *Oeuvres II*, pp 317-328. G. Darboux, ed. Paris: Gauthiers-Villars.

[27] J. Fourier, Second extrait, in *Oeuvres*, G. Darboux, edited by Gautheirs-Villars, Paris (1890) 317–328.

[28] S. Fujishige, A simple proof of the validity of Bland's anticycling rule for the simplex method, *Discussion Paper Series No. 268* (1985), Institute of Policy and Planning Sciences, University of Tsukuba, Japan.

[29] K. Fukuda, *Oriented Matriod Programming*, Ph.D Thesis (1982), Waterloo University, Waterloo, Canada.

[30] K. Fukuda, cddlib reference manual, cddlib Version 092b, Swiss Federal Institute of Technology, Lausanne and Zurich, Switzerland, 2002.

[31] K. Fukuda, J. Ferrez, Implementations of LP-based reverse search algorithms for the zonotope construction and the fixed-rank convex quadratic maximization in binary variables using the ZRAM and cddlib libraries (2002). Available from http://www.cs.mcgill.ca/~fukuda/download/mink/RS_TOPE020713.tar.gz.

[32] K. Fukuda, H-J Lüthi, *Optimization Techniques* (course notes), 1999.

[33] K. Fukuda and B. Kaluzny, The criss–cross method can take $\Omega(n^d)$ pivots, *Proc. of the 20th Annual Symposium on Computational Geometry* (2004), Brooklyn, NY, 401–408.

[34] K. Fukuda and T. Matsui, On the finiteness of the criss–cross method, *European Journal of Operations Research* **52** (1991) 119–124.

[35] K. Fukuda and M. Namiki, Two extremal behaviour of the criss-cross method for linear complimentary problems, *Research Report B-241* (1991), Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan.

[36] K. Fukuda and T.Terlaky, Criss-cross methods: A fresh view on pivot algorithms, *Math. Program.* **79** (1997) 369-395.

[37] K. Fukuda and T. Terlaky, On the existence of a short admissible pivot sequence for feasibility and linear optimization problems, *Pure Mathematics and Applications, Mathematics of Optimization*, **10** (4) (2000) 431–447.

[38] B. Gärtner, J. Solymosi, F. Tschirschnitz, P. Valtr, and E. Welzl, One line and n points, in: *Proc. 33rd ACM Symposium on the Theory of Computing (STOC),* ACM Press (2001) 306–315.

[39] S. Gass, *Linear Programming: Methods and Applications*, 5th edition, McGraw-Hill Book Company, New York, 1985.

[40] K. Gauss, Theoria combinationis observationum erroribus minimis obnoxiae, *Werke Vol. 4 Supplementum*, Göttingen (1826) 55–93.

[41] T. Gal, Degeneracy graphs - theory and applications: a state-of-the-art survey, *Report No. 142* (1989), Fern Universitat Hagen, Germany.

[42] T. Gal, On the structure of the set bases of a degenerate point, *Journal of Optimization Theory and Application* **45** (1985) 577–589.

[43] T. Gal and F. Geue, A new pivoting rule for solving various degeneracy problems, *Operations Research Letters* **11** (1992) 23–32.

[44] T. Gal, H.-F. Kruse, P. Zörnig, Survey of solved and open problems in the degeneracy phenomenon, *Mathematical Programming B* **42** (1988) 125.

[45] D. Goldfarb, Worst case complexity of the shadow vertex simplex algorithm, *Report* (1983) Columbia University, Department of Industrial Engineering and Operations Research.

[46] D. Goldfarb, On the complexity of the simplex algorithm, in: *Advances in Optimization and Numerical Analysis*, Proc. 6th Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico, January 1992; Kluwer, Dordrecht 1994, 25–38.

[47] B. Grünbaum, *Convex Polytopes*, 2nd edition prepared by V. Kaibel, V. Klee, G. Ziegler, Springer-Verlag, New York, 2003.

[48] M. Haimovitz, The simplex method is very good! - on the expected number of pivot steps and related properties of random linear programs, *Report* (1983) Columbia University, New York.

[49] A. Hoffman, Cycling in the simplex algorithm, *National Bureau of Standards*, Washington 1953.

[50] F. Holt and V. Klee, A proof of the strict monotone 4–step conjecture, *Contemporary Mathematics* **223** (1999) 201–216.

[51] D. Jensen, *Colouring and Duality: Combinatorial Augmentation Methods*, Ph.D Thesis (1985), School of OR and IE, Cornell University, Ithaca, NY.

[52] W. Jordan, Handbuch der Vermessungskunde, Vol. 1, J. Metclersche Buchhandlung, Stuttgard, 5th ed. (1904) 81–83, 100–105.

[53] M. Joswig and E. Gawrilow, an approach to modular software design incomputational geometry, *In Proceedings of the 17th Annual Symposium on Computational Geometry*, (2001) 222–33, Medford, MA. (http://www.math.tu-berlin.de/polymake)

[54] G. Kalai, A subexponential randomized simplex algorithm, in: *Proc. 24th ACM Symposium on the Theory of Computing (STOC)*, ACM Press 1992, 475–482.

[55] G. Kalai, Linear programming, the simplex algorithm and simple polytopes, *Mathematical Programming* **79** (1997) 217–233.

[56] G. Kalai, Polytope skeletons and paths, in: *Handbook of Discrete and Computational Geometry*, CRC Press, 1997.

[57] L. Kantorovich and M Gavurin, The application of mathematical methods to problems of freight flow analysis (translation), *Akademii Nauk SSSR* (1949).

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

[59] L. Khachian, Polynomial algorithms in linear programming, *USSR Computational Mathematics and Mathematical Physics* **20** (1980) 53–72 (English translation).

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

[61] V. Klee and G. J. Minty, How good is the simplex algorithm? In O. Shisha, editor, *Inequalities–III*, 159–175. Academic Press, New York, 1972.

[62] V. Klee and P. Kleinschmidt, The $d-$step conjecture and its relatives, *Mathematics of Operations Research* **12** (4) (1987) 718–755.

[63] M. Konstantinov, D. Wei Gu, V. Mehrmann, and P. Petkov, *Perturbation Theory For Matrix Equations*, Studies in Computational Mathematics 9, Elsevier 2003.

[64] H.-J. Kruse, *Degeneracy Graphs and the Neighbourhood Problem*, Lecture Notes in Economics and Mathematical Systems No. 260, Springer-Verlag, Berlin, 1986.

[65] H. Kuhn and R. Quandt, An expiremental study of the simplex method, *Proc. of Symposium in Applied Mathematics* **15** (1963) 107–124.

[66] D. Larman, Paths on polytopes, *Proc. Lon. Math. Soc.* **20** (1970) 161–178.

[67] J. Lee, Hoffman's circle untangled, *SIAM Review* **39** (1997) 98–105.

[68] K. Marshall and J. Suurballe, A note on cycling in the simplex method, *Naval Research Logistics Quarterly* **16** (1) (1969) 121–37.

[69] A. Marzetta, ZRAM homepage, Available from http://www.cs.unb.ca/profs/bremner/zram/.

[70] J. Matoušek, M. Sharir and E. Welzl, A subexponential bound for linear programming, in: *Proc. 8th Annual ACM Symposium Computational Geometry* (Berlin 1992), ACM Press, 1–8.

[71] J. Matoušek, *Lectures on Discrete Geometry*, Springer-Verlag New York, 2002.

[72] P. McMullen, The maximum number of faces of a convex polytope, *Mathematika* **17** (1970) 179–184.

[73] N. Megiddo, A note on degeneracy in linear programming, *Mathematical Programming* **35** (1986) 365–367.

[74] K. Murty, Computational complexity of parametric linear programming, *Mathematical Programming* **19** (1980) 213–219.

[75] D. Naddef, The Hirsch conjecture is true for $\{0,1\}-$polytopes, *Math. Prog.* **45** (1989) 109–111.

[76] M. Namiki and T. Matsui, Some modifications of the criss-cross method, *Research Report* (1990), Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan.

[77] E. Nering, A. Tucker, *Linear Programs and Related Problems*, Academic Press, Boston, 1993.

[78] K. Paparrizos, Pivoting rules directing the simplex method through all feasible vertices of Klee-Minty examples, *Opsearch* **26** 2 (1989) 77–95.

[79] J. Pfeifle and G. Ziegler, On the monotone upper bound problem, *Experimental Mathematics* **13** 1 (2004) 1–12.

[80] M. de la Vallée Poussin, Sur la méthode d'approximation minimum, *Ann. Soc. Sci. de Bruxelles* **35** (1911) 1–16.

[81] J. Richter-Gebert and G. Ziegler. Oriented matroids. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 111–132. CRC Press, New York, 1997.

[82] C. Roos. An exponential example for Terlaky's pivoting rule for the criss-cross simplex method, *Mathematical Programming* **46** (1990) 78–94.

[83] D. Ryan and M. Osborne, On the solution of highly degenerate linear programs, *Mathematical Programming* **41** 1 (1988) 385–392.

[84] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley & Sons, Amsterdam, 1987.

[85] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency*, Springer Verlag, Berlin, 2003.

[86] G. Sierksma, *Linear and Integer Programming*, 2nd ed. Marcel Dekker Inc., New York, 1996.

[87] D. Solow, *Linear Programming: An Introduction to Finite Improvement Algorithms.* North-Holland Press, Amsterdam, 1984.

[88] D. Speilman and S.-H. Teng, Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing* (2001) 296–305.

[89] G. Strang, *Linear Algebra and its Applications*, 3rd ed. Philadelphia USA, Saunders, 1988.

[90] T. Terlaky, A convergent criss-cross method, *Math. Oper. und Stat. ser. Optimization* **16**(5) (1985) 683–690.

[91] T. Terlaky and S. Zhang, A survey on pivot rules for linear programming, Report 91-99, Faculty of Technical Mathematics and Informatics, Delft University of Technology, The Netherlands, 1991.

[92] M. Todd, The monotone bounded Hirsch conjecture is false for dimension at least four, *Math. Operations Research* **5** (1980) 599–601.

[93] M. Todd, Linear and quadratic programming in oriented matroids, *Journal of Combinatorial Theory Series B* **39** (1985) 105–133.

[94] M. Todd, Polynomial expected behaviour of a pivoting algorithm for linear complimentarity and linear programming problems, *Mathematical Programming* **35** (1986) 173–192.

[95] Z. Wang, A conformal elimination free algorithm for oriented matroid programming, *Chinese Annals of Mathematics*, (1987) **8**(B1).

[96] P. Wolfe, A technique for resolving degeneracy in linear programming, *Journal of SIAM* **11** (1963) 205–211.

[97] C. Yap, Symbolic treatment of geometric degeneracies, *Journal of Symbolic Computation* **10** (1990) 349–370.

[98] D. Yudin, E. Gol'shtein, *Linear Programming*, Israel Program of Scientific Translations, Jerusalem, 1965.

[99] N. Zadeh, What is the worst-case behaviour of the simplex algorithm? *Technical Report No. 27*, Department of Operations Research, Stanford University, Stanford, California, 1979.

[100] T. Zaslavsky. Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Mem. Amer. Math. Soc.*, 1(No 154 MR 50), 1975.

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

[102] G. Ziegler, *Lectures on Polytopes.* Springer-Verlag, New York, 1995.

[103] S. Zionts, The criss-cross method for solving linear programming problems, *Management Science* **15**(7) (1969) 426–445.

[104] P. Zörnig, *Degeneracy Graphs and Simplex Cycling*, Lecture Notes in Economics and Mathematical Systems No. 357, Springer-Verlag, Berlin, 1991.

[105] P. Zörnig, A theory of degeneracy graphs, *Annals of Operations Research* **47** (1993) 541–556.

[106] http://cgm.cs.mcgill.ca/~avis/C/lrs.html

[107] http://cgm.cs.mcgill.ca/~beezer/DisjointLP/

[108] http://cgm.cs.mcgill.ca/~beezer/PivotPathEnumeration/