THE LEAST-USED DIRECTION PIVOT RULE ON ACYCLIC UNIQUE SINK ORIENTATIONS

Theresa Kiyoko Deering

Master of Science

School of Computer Science

McGill University

Montreal, Quebec

July 2010

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Science.

ACKNOWLEDGEMENTS

I would like to thank the faculty, staff, and students at McGill. They have made my time at McGill an enjoyable experience as well as an academic one. In particular, I would like to thank my supervisor, Professor David Avis, for the course that introduced me to linear programming, for his guidance throughout my research, and for his invaluable feedback during the writing of this thesis.

I would like to thank the faculty, staff, and students at Malaspina University-College (now Vancouver Island University), without whom I would not be where I am today.

I am grateful for the two unforgettable months spent in Kyoto, Japan writing my thesis. This was made possible by Professor Kazuo Iwama of Kyoto University for allowing me to work in his laboratory, Yoshikazu Aoshima of Tokyo University for stimulating discussion, the Ministère du Développement économique, de l'innovation et de l'exportation (MDEIE) for financial support, and Yoko Kurauchi, my home-stay mother.

Financial support of my research and education is courtesy of the Natural Science and Engineering Research Council (NSERC) and the McGill School of Computer Science.

Most of all, I wish to thank my family and friends for all their unconditional love and emotional support.

ABSTRACT

The least-used direction (LUD) rule is one of a class of largely unanalyzed pivot rules - the history-based rules. History-based pivot rules guide the progression of edge following algorithms like the Simplex method. This thesis investigates the problem of finding an exponential length LUD path on a particular kind of digraph known as an acyclic unique sink orientation of a hypercube (AUSO). In addition, a survey of six well-known history-based pivot rule and examples to illustrate their independence is given. The Fibonacci construction is introduced as a potential way of creating families of AUSOs that allows for exponential LUD paths. The most straight-forward application of this technique is unsuccessful, but there is room for more exploration. An exponential lower bound is given for the number of times the least-used direction is used by a Hamiltonian path following the related history-based Zadeh's rule. This result shows that the number of times each direction is used grows at a similar rate and is thus relatively balanced.

ABRÉGÉ

La règle de least-used direction (LUD) fait parti de la grande classe des règles pivots history-based. Ceux-ci guident la progression des algorithmes de détection des contours, tel que la méthode Simplex. Ce mémoire examine le problème de la recherche de chemin LUD de longueur exponentielle dans un graphe acyclique dirigé à bloc récepteur unique (acyclic unique sink orientation - AUSO) dans les hypercubes. De plus, une vue densemble de six règles pivots history-based ainsi que des exemples sont fournis pour illustrer leurs indépendances. La structure Fibonacci est présentée comme une possibilité de créer des familles de graphe acyclique dirigé à bloc récepteur unique permettant des chemins LUD de longueurs exponentielles. Exécuter cette technique de faon simple savère infructueuse, par contre cela nous laisse la place à plus damples explorations. Une borne inférieure exponentielle est fournie pour le nombre de fois que LUD est présent dans un chemin Hamiltonian selon la règle history-based Zadeh. Les résultats obtenus démontrent que le nombre de fois utilisé par chaque direction augmente à une fréquence semblable et est donc relativement équilibré.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

Computational complexity theory categorizes algorithms as either "good" or "bad" based on how resource-usage increases as the size of the input increases. One important resource is time. Algorithms that take a constant amount of time on each unit of input plus any constant overhead are known as *linear*; these algorithms are considered very good. Linear algorithms are a subset of the class of algorithms known as polynomial. *Polynomial* algorithms take time proportional to some polynomial function of the units of input; these algorithms are considered good. Algorithms that take time proportional to some exponential function of the units of input are known as *exponential*; these algorithms are considered bad. Algorithms that involve processing combinatorial input, such as searching for a *Hamiltonian path* in a graph (a path that visits every vertex exactly once), are often exponential.

The categories linear, polynomial, exponential, and others not mentioned here are theoretical distinctions based solely on worst-case behaviour. In many instances, the distinction provides a simple way of determining the practicality of using a particular algorithm. However, this is not always the case. There are some "bad" algorithms that perform well in practice. They are able to execute quickly on the vast majority of typical input. Various versions of the Simplex method for solving linear programs have this property.

A *linear program* (LP) is an optimization problem of the following form:

$$\max c^T x$$

$$\text{subject to } Ax \leq b$$

$$x \geq 0$$

where c, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. The region of allowable solutions, known as the *feasible region*, is given by a system of linear inequalities ($Ax \leq b$, $x \geq 0$), and the goal is to find a point x in the feasible region where the linear function $c^T x$ is maximized. Geometrically, this feasible region is called a *polyhedron* and when it is bounded, a *polytope*. The maximum value will always occur at a vertex of the polytope. The *Upper Bound Theorem* states that a polytope defined by $m+n$ inequalities can have as many as $\binom{n+m-\lfloor\frac{n+1}{2}\rfloor}{m} + \binom{n+m-\lfloor\frac{n+2}{2}\rfloor}{m}$ vertices; this is superexponential [25]. Therefore, simply checking the value of every vertex to find the maximum is a superexponential algorithm.

The *Simplex method* for solving LPs is a framework for a family of algorithms. It solves an LP by following a path along the directed edges of the feasible region. The edges are directed from a vertex x to a vertex y if and only if $c^T x < c^T y$. Once the path reaches a vertex with no outgoing edges, the process is complete and the value given by the last vertex is the maximum.

It is possible that there are more than one improving vertices; therefore, a rule for selecting the vertex must be selected in order for the Simplex method to be an algorithm. This rule is called a *pivot rule*. There are a number of pivot rules that cause the Simplex method to be exponential, but there are other rules that have not yet been analyzed. One of these rules may yield a polynomial algorithm.

There are two main reasons to suspect that there exists a pivot rule that will make the Simplex method polynomial. Firstly, as previously mentioned, the method performs well in practice even using exponential pivot rules. Secondly, there exist several non-Simplex-based algorithms that solve LPs in polynomial time.

Victor Klee and George Minty were the first to prove that the Simplex method is exponential when used in conjunction with the pivot rule known as Dantzig's rule. They did so by constructing an LP on which the pivot rule forces the Simplex method to complete a Hamiltonian path of the feasible region before finding the sink. The feasible region of Klee and Minty's LP is the perturbed hypercube now known as the *Klee-Minty cube.* An n-hypercube has 2^n vertices, so a Hamiltonian path is exponentially long.

On the graph of a hypercube, the *least-used direction* (LUD) rule is a pivot rule that keeps track of how many times the path selects each direction. When the Simplex method reaches a point where a decision must be made between multiple outgoing edges, the LUD rule selects the direction that has been used the least thus far. Presently, no time complexity results are known for the Simplex method using this pivot rule.

To prove that the LUD rule makes the Simplex method exponential using a construction similiar to the Klee-Minty cube involves two steps. For the algorithm to take exponential time, there must be an exponential length path through the vertices of some polytope that obeys the LUD rule at every move. The first step is to find such a path. If such a path can be found then the second step is to look for an LP whose feasible region corresponds to the polytope and whose objective function evaluates to the same relative values at each vertex.

This thesis investigates the LUD pivot rule only for polytopes combinatorially equivalent to a hypercube. In order for such a hypercube to be realized as an LP (the second step), it must satisfy two necessary conditions, acyclicity and unique sink orientation. *Unique sink orientation* (USO) is the property that every face of the hypercube, including the hypercube itself, has exactly one source. Therefore, this thesis analyzes the problem of finding an exponential path that obeys the LUD pivot rule on an acyclic USO (the first step).

Chapter 2 gives the necessary background knowledge and a review of the classical and current literature related to the problem. The chapter is further divided into the following sections: linear programming, the Simplex method, acyclic USOs, and pivot rules.

Chapter 3 details a construction for acyclic USOs that has the potential for giving exponential paths. The most straight-forward implementation of this Fibonacci construction is unsuccessful, but further research may meet with interesting results. The chapter is further divided into the following sections: general construction and implementation.

Chapter 4 gives an exponential lower bound on the number of times the least-used direction is used by the LUD-related pivot rule known as Zadeh's rule assuming that there exists a Hamiltonian path that follows Zadeh's rule. The relationship between Zadeh's rule and the LUD rule is very close, so this bound may give some insight into the LUD rule.

Chapter 5 concludes this thesis with a summary and a discussion of future work related to finding an exponential path that obeys the LUD pivot rule on an acyclic USO.

CHAPTER 2
Background

The aim of this chapter is to review the related literature and to provide all the background knowledge needed to understand the remainder of the thesis. The first section describes linear programming and will explain its history, its main algorithms, and some of its properties. The second section explains the Simplex method of solving linear programs. The third section examines the use and the usefulness of the acyclic unique sink orientation abstraction. The last section details select pivot rules that may be used in conjunction with the Simplex method with an emphasis on history-based rules.

2.1 Linear Programming

A *linear program* (LP) is a common type of optimization problem. It has applications in the areas of resource allocation [6], scheduling [8], network flows [12], game theory [9], approximation algorithms [17], and more. Much of the continued practical interest in the Simplex method of solving LPs is for their applications in cutting plane methods. Cutting plane methods are used for solving the NP-hard problem of solving integer linear programs. Every problem in NP can be formulated as an integer linear program, so LPs are especially important for this application.

In general, optimization problems are defined by a set of feasible solutions and the goal is to find some "best" or optimal value over all the solutions in the set.

In linear programming, the set of feasible solutions is given by a system of linear inequalities ($Ax \leq b$, $x \geq 0$) referred to as *linear constraints*, and the optimal value is given by the vector x that maximizes the *objective function* ($c^T x$). The standard form of an LP [6] is

$$\begin{aligned} \max\ & c^T x \\ \text{subject to}\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where c, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. Its corresponding *dictionary representation* is the system of linear equations

$$\begin{aligned} x_s &= b - Ax \\ z &= c^T x \end{aligned}$$

The newly introduced variables, $x_s \geq 0$, are called *slack variables*, and the original variables, $x \geq 0$, are *decision variables.* Variables on the left-hand side of the equations, initially the slack variables, are called *basic* and are said to be in the *basis.* The last row, $z = c^T x$, is called the *objective row.*

LPs have a natural *geometrical representation* in n-dimensional space, and it is often helpful to think of them in this way. Each linear inequality defines a halfspace. If the intersection of all the halfspaces is empty then there are no feasible solutions for the LP. If, on the other hand, the intersection is non-empty then it forms an n-polyhedron called the *feasible region.* More information about polyhedra can be found in Ziegler's book [38]. A point is in the feasible region if and only if it satisfies all the LP constraints. When the feasible region is unbounded then it extends to infinity in at least one direction. For unbounded polyhedra there may be no optimal solution. When the feasible region forms a bounded polyhedron, a *polytope*, then there is always at least one optimal solution. A polytope is *simple* if each vertex is the intersection of exactly n of the LP constraints (see Figure 2–1).

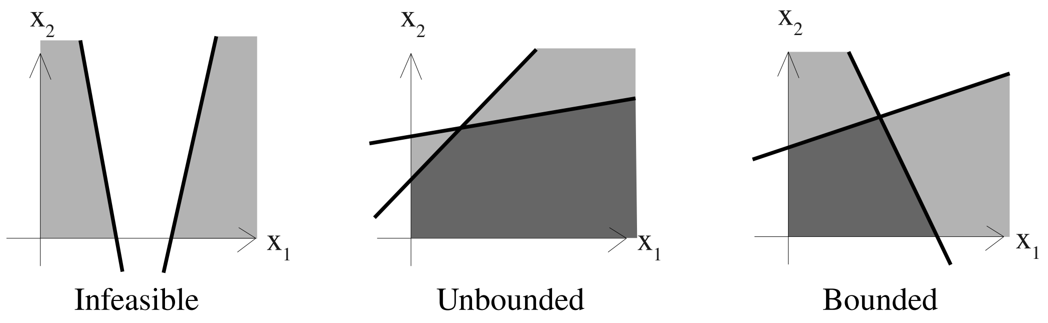


Figure 2–1: The three possible intersections of halfspaces in 2 dimensions.

A *hypercube* is an example of a simple polytope. An n-dimensional hypercube is a polytope defined by the halfspaces $0 \leq x_i \leq 1$ for $i = 1, ..., n$. Each vertex is a point where exactly n of these $2n$ inequalities are at equality. A related, but not necessarily simple, class of polytopes is the *zonotopes*. A zonotope is the affine projection of an n-dimensional hypercube onto $d < n$ dimensional space. For a non-simple example in three dimensions, see [38].

The objective function of an LP is geometrically represented by an n-dimensional hyperplane. To maximize the objective function, move the hyperplane through the feasible region to the furthest point such that they still intersect. At that time, the intersection points give the maximum value (see Figure 2–2).

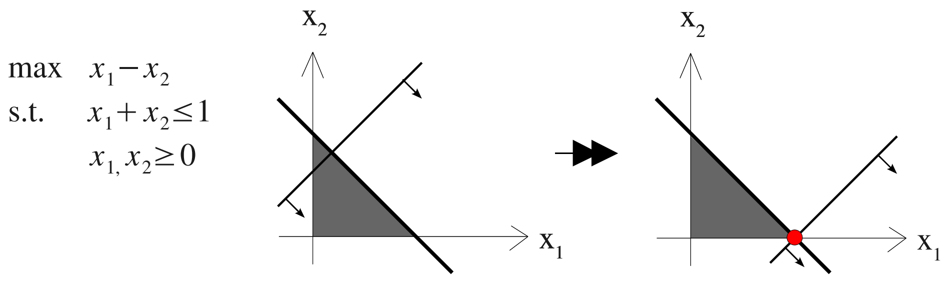


Figure 2–2: An example of the hyperplane $x_1 - x_2$ moving through the feasible region to the vertex with the maximum value.

There is no loss of generality to assume that the feasible region defines a simple polytope and that the objective function has a unique value at each vertex. This is the same as assuming that degeneracy [6] does not occur and that exactly one vertex gives the maximum value.

To work with the geometrical representation, take the graph of the polytope's edges and direct each edge $\{x, y\}$ from vertex x to vertex y if and only if $c^T x < c^T y$ (see Figure 2–3). This preserves the structure of the polytope, and any subgraph induced by a face of the polytope will be considered a *face* of the new digraph. Now the search for the polytope's maximizing vertex is equivalent to the search for the digraph's unique sink.

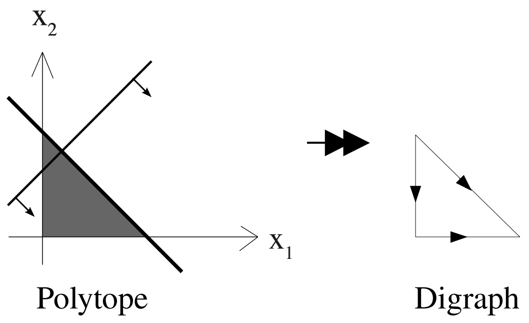


Figure 2–3: The conversion from a polytope to a digraph.

Every bounded LP has such a corresponding digraph, but not every digraph has a corresponding LP. Those digraphs that do are known as *LP digraphs*. It is difficult to tell whether an arbitrary digraph is an LP digraph. No properties are known to be both necessary and sufficient aside from actually finding a corresponding LP. However, Avis and Moriyama [3] analyze the relationships between four distinct necessary properties. These properties are *acyclicity*, *unique sink orientation*, *Holt-Klee*, and *shelling*. All digraphs that fail to possess even one of these properties are not LP digraphs. The acyclicity property means that there are no directed cycles. The USO property means that the directed subgraph defined by each face of the polytope has a unique sink (and consequently a unique source for simple polytopes [38]). The Holt-Klee property means that every k-dimensional face of the digraph has at least k vertex-disjoint paths from its source to its sink. The shelling property involves a topological sort of the digraph's associated polytope. Acyclicity and USO are discussed in Section 2.3. For more information about Holt-Klee and shelling, see [18] and [3] respectively. These properties can be useful in the complexity analysis of LP-solving algorithms.

There are three main methods for solving LPs. Two of these methods work well in practice, the Simplex method [8] and the *interior-point method* [22, 36], and two of them are polynomial algorithms, the interior-point method and the *ellipsoid method* [23, 16]. The Simplex method concentrates solely on the vertices of the feasible region; the other two algorithms utilize the interior region of the polytope.

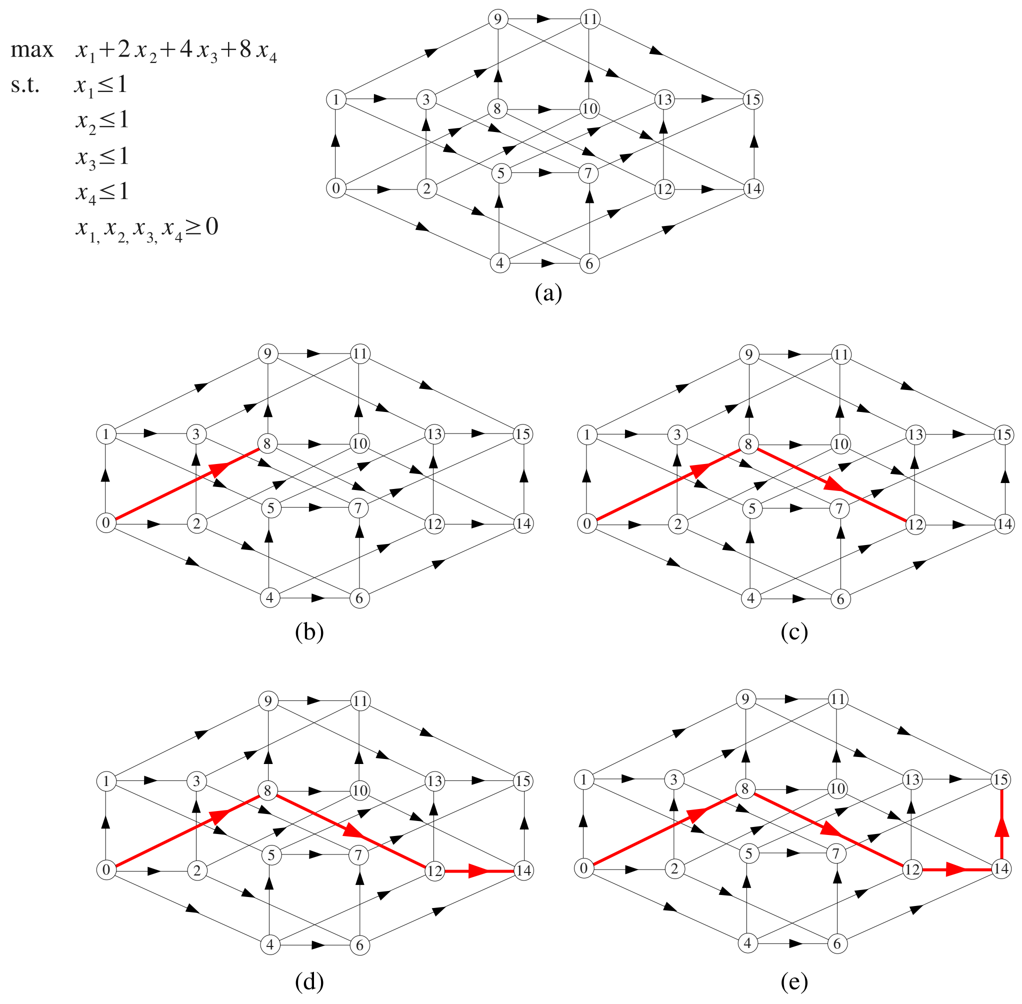
The existence of polynomial algorithms is very important for the theory of linear programming because it gives researchers confidence that any problem that can be formulated as an LP has a good solution. But this is not the end of the complexity battle between these methods because the two polynomial methods are what is known as *weakly polynomial.* This means that they are polynomial in terms of the bit-size of the input as opposed to *strongly polynomial* algorithms, which would be polynomial in terms of the number of variables and the number of constraints of the LP. It is due to the combinatorial nature of the Simplex method that if a polynomial variant is found it would most likely be strongly polynomial.

2.2 Simplex Method

More than 30 years prior to the introduction of the ellipsoid method and the interior-point method there was the Simplex method. Developed by George B. Dantzig during World War II but not published until 1947, it was intended to solve planning and management problems for the U.S. military [8].

The Simplex method takes advantage of two important properties of the feasible regions of LPs: the optimal value is given by a vertex and, since the polytope is convex, the local optimum is also the global optimum. The first property, that the optimal value is given by a vertex, allows the algorithm to check only the finite number of vertices rather than the infinite number of points that make up the feasible region. The second property, that the local optimum is the global optimum, allows the algorithm to use local improvement rules to find the global optimum. These two properties together ensure that the Simplex method will terminate at the maximum vertex in finite time.

The Simplex method starts at any vertex of the polytope and then moves to an adjacent vertex on the condition that the value of the objective function is greater at the new vertex than at the old one (local improvement). This move is called a *pivot*. Geometrically, successive pivots form a path on the edges of the associated LP digraph. When there is no such improving vertex, the method has found the maximum value (the local and global optimum) (see Figure 2–4).



$$\begin{aligned}\max\quad & x_1+2x_2+4x_3+8x_4\\ \text{s.t.}\quad & x_1\le 1\\ & x_2\le 1\\ & x_3\le 1\\ & x_4\le 1\\ & x_1,x_2,x_3,x_4\ge 0\end{aligned}$$

Figure 2–4: A geometrical example of the Simplex method on a 4-dimensional hypercube. (a) The associated LP-digraph labelled with the value of the objective function at each vertex. (b) Starting at vertex 0, pivot to vertex 8 ($0 < 8$). (c) From vertex 8, pivot to vertex 12 ($8 < 12$). (d) From vertex 12, pivot to vertex 14 ($12 < 14$). (e) From vertex 14, pivot to vertex 15 ($14 < 15$) and stop because there are no more improving vertices. 15 is the maximum value.

In the dictionary representation, a pivot is a change of basis that improves the value of the objective row. One variable *enters* the basis and is known as the *entering variable*, and one *leaves* the basis and is known as the *leaving variable*. When there is no change of basis that gives a better value in the objective row, the method has found the maximum value (see Figure 2–5).

$$\begin{array}{ll} \text{max} & x_1+2x_2+4x_3+8x_4 \\ \text{s.t.} & x_1\leq 1 \\ & x_2\leq 1 \\ & x_3\leq 1 \\ & x_4\leq 1 \\ & x_1,x_2,x_3,x_4\geq 0 \end{array}$$

$$\begin{array}{l} x_5=1-x_1 \\ x_6=1-x_2 \\ x_7=1-x_3 \\ x_8=1-x_4 \\ z=x_1+2x_2+4x_3+8x_4 \end{array}$$

(a)

$$\begin{array}{l} x_5=1-x_1 \\ x_6=1-x_2 \\ x_7=1-x_3 \\ x_4=1-x_8 \\ z=8+x_1+2x_2+4x_3-8x_8 \end{array}$$

(b)

$$\begin{array}{l} x_5=1-x_1 \\ x_6=1-x_2 \\ x_3=1-x_7 \\ x_4=1-x_8 \\ z=12+x_1+2x_2-4x_7-8x_8 \end{array}$$

(c)

$$\begin{array}{l} x_5=1-x_1 \\ x_2=1-x_6 \\ x_3=1-x_7 \\ x_4=1-x_8 \\ z=14+x_1-2x_6-4x_7-8x_8 \end{array}$$

(d)

$$\begin{array}{l} x_1=1-x_5 \\ x_2=1-x_6 \\ x_3=1-x_7 \\ x_4=1-x_8 \\ z=15-x_5-2x_6-4x_7-8x_8 \end{array}$$

(e)

Figure 2–5: A dictionary example of the Simplex method. (a) The associated dictionary with the basis $\{x_5, x_6, x_7, x_8\}$ where $z = 0$. (b) Pivot to the basis $\{x_5, x_6, x_7, x_4\}$ where $z = 8 > 0$. (c) Pivot to the basis $\{x_5, x_6, x_3, x_4\}$ where $z = 12 > 8$. (d) Pivot to the basis $\{x_5, x_2, x_3, x_4\}$ where $z = 14 > 12$. (e) Pivot to the basis $\{x_1, x_2, x_3, x_4\}$ where $z = 15 > 14$ and stop because there are no more improving bases. 15 is the maximum value.

In order to analyze the Simplex method as an algorithm, there must be a rule to select the next vertex when multiple adjacent vertices offer an improvement in the objective function. Such a rule is called a *pivot rule*. A number of pivot rules are described in Section 2.4. Some polytopes are known to have "long" paths to the sink; the number of pivots is exponential in the size of the input [27]. A desirable pivot rule should never follow an exponential path. It is not known whether such a rule exists.

The *diameter* of a polytope is the maximum length shortest-path between any two vertices. It gives a lower bound on the time complexity of the Simplex method regardless of the chosen pivot rule. If the diameter is not polynomial then neither is the Simplex method. The *Hirsch conjecture*, or equivalently the *d-step conjecture*, [8] surmises that the diameter of an undirected skeleton of an n-dimensional polytope is no greater than $m - n$ for all $m > n \geq 2$ [26]. Santos [30] recently claimed to have found a counterexample with 43 dimensions and 86 facets.

Settling the Hirsch conjecture does not give any definitive answers about the existence of a polynomial pivot rule. If the conjecture is true, a pivot rule that always finds a polynomial path must still be found. If the conjecture is false, it does not mean a polynomial path does not exist, just that it is longer than Hirsch believed. On the other hand, if a pivot rule is discovered that finds a polynomial path in all polytopes then the length of the path may confirm the conjecture. In any case, it would give a polynomial upper bound on the diameter of a polytope, which would be a major theoretical advance. As it stands, the randomized pivot rule of [20] and [28] shows that the diameter is subexponential. A comprehensive survey of the d-step conjecture and its relatives was done by Klee and Kleinschmidt in 1987 [26]. For a more recent survey, see the 2009 paper by Kim and Santos [24].

2.3 Acyclic Unique Sink Orientations

An *acyclic unique sink orientation* (AUSO) is a tool for analyzing the complexity of pivot rules for the Simplex method. An AUSO is an orientation of the graph of a polytope (normally a graph of a hypercube) such that each face of the graph, including the graph itself, has exactly one sink (see Figure 2–6).

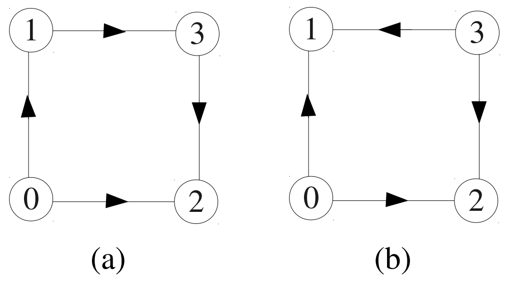


Figure 2–6: (a) An example of an AUSO. The 1-dimensional sinks are the end-points of each edge. The 2-dimensional sink is vertex 2. (b) Not an AUSO because on the 2-dimensional face there are two sinks, vertex 1 and vertex 2.

This is useful for analyzing pivot rules because it is an abstraction of an LP and it satisfies two of the four known properties necessary in LP digraphs. Although finding an exponential path through an AUSO that obeys a particular pivot rule does not guarantee that the Simplex method is exponential using that rule, it is a step in that direction.

The easiest AUSO to work with when looking for an exponential path is one on a digraph *combinatorially equivalent* to a hypercube. To be combinatorially equivalent is to have isomorphic face lattices (see [38]). In other words, there is a mapping between the vertices of the AUSO and the unit hypercube such that there is an edge between two vertices in the AUSO if and only if there is an edge between the two corresponding vertices in the hypercube, and there is a face in the AUSO if and only if there is a face in the unit hypercube. An AUSO on a hypercube is useful because it has exponentially many vertices in terms of the dimension and it can be defined iteratively. Therefore, exponential paths can exist and it is possible that an exponential path in one dimension can be used to define exponential paths in higher dimensions. AUSOs on hypercubes form the basis of the construction presented in Chapter 3 and the proof presented in Chapter 4.

AUSOs on hypercubes have structure that makes for convenient notation and terminology. Each vertex is labelled $0, ..., (2^n - 1)$ such that the binary representation of adjacent vertices' labels differ by exactly one bit. Each edge has a *direction* and *orientation*. The direction is given by a number $1, ..., n$ indicating which bit is different between the two endpoints (counted right-to-left). The orientation is given by a positive sign (+) if the differing bit is 0 at the edge's tail and 1 at its head, and it is given by a negative sign (-) otherwise. This is illustrated in Figure 2–7. For some pivot rules the direction is important but not the orientation and in others the (orientation, direction)-pair is important. Although AUSOs do not necessarily correspond to LP digraphs, the above vertex labelling can be used to model moving along a path on an AUSO as pivoting in the dictionary $x_{n+i} = 1 - x_i$ for $i = 1, ..., n$. A pivot $+i$ corresponds to a pivot where x_i enters the basis and x_{n+i} leaves, and a pivot $-i$ corresponds to a pivot where x_{n+i} enters the basis and x_i leaves. This allows the AUSO to inherit various pivoting strategies that are defined in terms of LPs. Unless otherwise stated, all further references to AUSOs are to AUSOs on hypercubes.

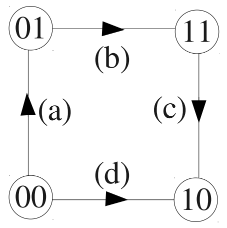


Figure 2–7: Each vertex is labelled with its binary representation. Edge (a) has direction 1 since the first bit changed and orientation + since it changed from 0 to 1. Edge (b) has direction 2 since the second bit changed and orientation + since it changed from 0 to 1. Edge (c) has direction 1 since the first bit changed and orientation - since it changed from 1 to 0. Edge (d) has direction 2 since the second bit changed and orientation + since it changed from 0 to 1.

AUSOs are not the first abstraction used to study LPs and the Simplex method. Two previous abstractions were developed specifically for this area of research. Adler and Saigal [1] introduced *abstract objective functions* (AOFs) in 1976, and Hoke [35] introduced *completely unimodal numberings* in 1988. Both of these abstractions assign a number to each of the vertices of a polytope. In an AOF, each vertex is assigned a real number, and in a completely unimodal numbering each vertex is assigned a different integer $0, ..., (k - 1)$ where k is the number of vertices. In addition, on every face of an AOF there is a strictly increasing valued path from every vertex to the maximum-valued vertex and a strictly decreasing path from every vertex to the minimum-valued vertex. On every face of a completely unimodal numbering there is exactly one vertex whose number is higher than all of its neighbours. In these two abstractions, finding the maximum-valued vertex is equivalent to finding the LP solution.

There is a strong relationship between these three abstractions. AUSOs (not restricted to hypercubes) and completely unimodal numberings are equivalent, all completely unimodal numberings are AOFs, and all AOFs define an AUSO (not restricted to hypercubes) and a completely unimodal numbering (see Figure 2–8).

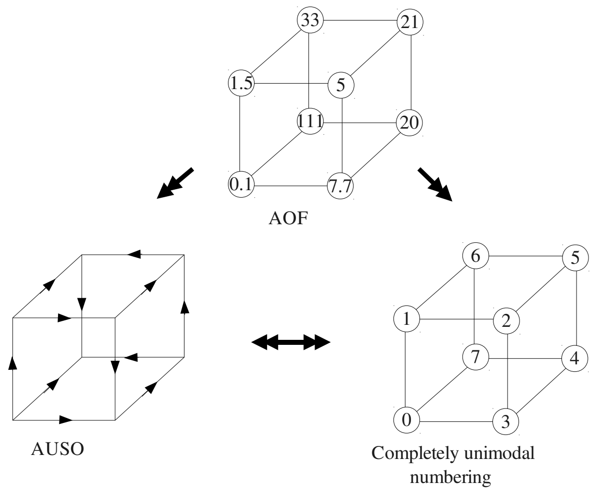


Figure 2–8: The relationship between AOF, AUSO, and completely unimodal numbering.

If AUSOs and completely unimodal numberings are equivalent then why were AUSO introduced? The fact is that USOs were not originally intended to be used for LPs. USOs on hypercubes were introduced by Szabó and Welzl [33] in 2001 for use on two other research problems. They noticed that the linear complementarity problem on a class of matrices known as P-matrices [32] and some quadratic optimization problems including finding the smallest enclosing ball of a set of points [33] could both be solved by finding the sink of an appropriate USO on a hypercube. Gärtner and Schurr [15] later found that the problem of solving arbitrary LPs can also be encoded as a USO where the sink gives the solution if there is one or a proof of infeasibility or unboundedness otherwise. Note that this encoding is not an LP digraph of the problem. In all three of these problems, cycles can arise in the USO.

2.4 Pivot Rules

The Simplex method is not an algorithm unless it has a pivot rule. There are many existing pivot rules and many more that have not yet been discovered. Pivot rules can be deterministic or randomized. They can be based on historical information or not. They can cause the Simplex method to be exponential, sub-exponential, or maybe someday polynomial.

The original pivot rule was introduced by Dantzig in 1947. It is more easily explained in the context of the dictionary representation of LPs than the geometrical representation.

Dantzig's rule (a.k.a. the *largest coefficient rule*) [8]: For the entering variable, select the variable with the largest coefficient in the objective row.

Figure 2–4 uses Dantzig's rule.

In practice, Dantzig's rule performs exceedingly well. In empirical tests, the Simplex method using Dantzig's rule finds the maximum in m to $3m$ iterations [8]; this is linear in the number of constraints in the LP. For years Dantzig's rule was used with no problems but without knowing whether it were a theoretically "good" algorithm. Then in 1970, Klee and Minty published a breakthrough paper answering the question of "how good is the Simplex algorithm?" [27]. They construct an LP digraph on a hypercube that forces Dantzig's rule to visit all 2^n vertices in its search for the sink, thus proving that Dantzig's rule is exponential.

The structure of the Klee-Minty cube is that of a deformed product. The Klee-Minty cube is an example of an AUSO. Each cube can be built recursively from the k-dimensional case to the $(k+1)$-dimensional case as follows (see Figure 2–9):

1. Given the k-dimensional cube C, create a duplicate C'. For each vertex $x \in C$, call its duplicate vertex in C' x'.
2. Add the edge (x, x') for all vertices $x \in C$.
3. Reverse all edges that have both endpoints in C'.

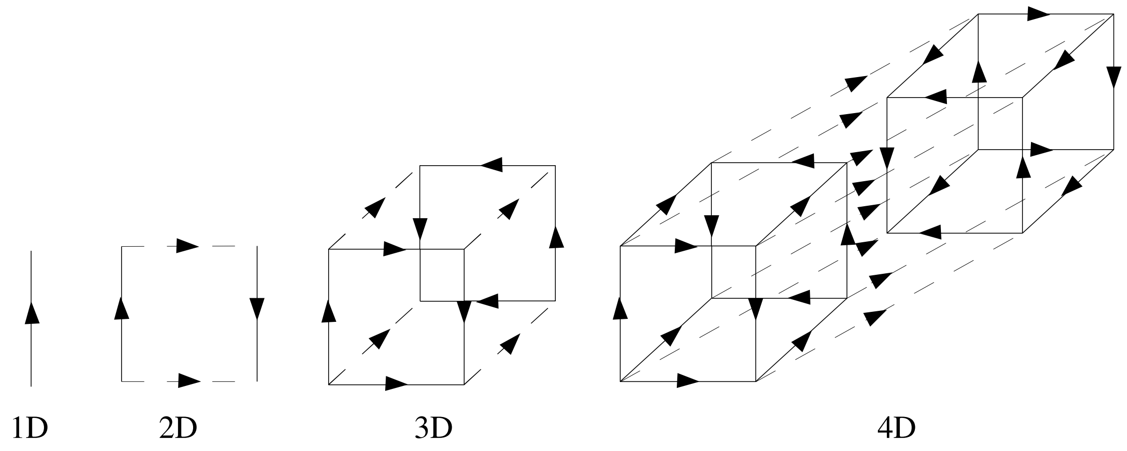


Figure 2–9: The Klee-Minty cube for dimensions 1, 2, 3, and 4.

A number of other pivot rules give exponential paths on the Klee-Minty cube. They include the *maximum improvement rule* (a.k.a. the *maximum increase rule*), which Jeroslow [19] proved exponential in 1973, and *Bland's rule* (a.k.a. the *smallest subscript rule*) [5], which Avis and Chvátal [2] proved exponential in 1978. All of these pivot rules are deterministic and not history-based.

In a 1980 technical report, Norman Zadeh [37] investigated the Klee-Minty cube construction hoping to determine why it is so effective against certain pivot rules. He found that the paths following these rules recurse in the same way as the cube's construction. First the k-dimensional Hamiltonian path is followed in C, then it takes an edge from C to C'. Once in C' it follows the same k-dimensional Hamiltonian path backwards (see Figure 2–10).

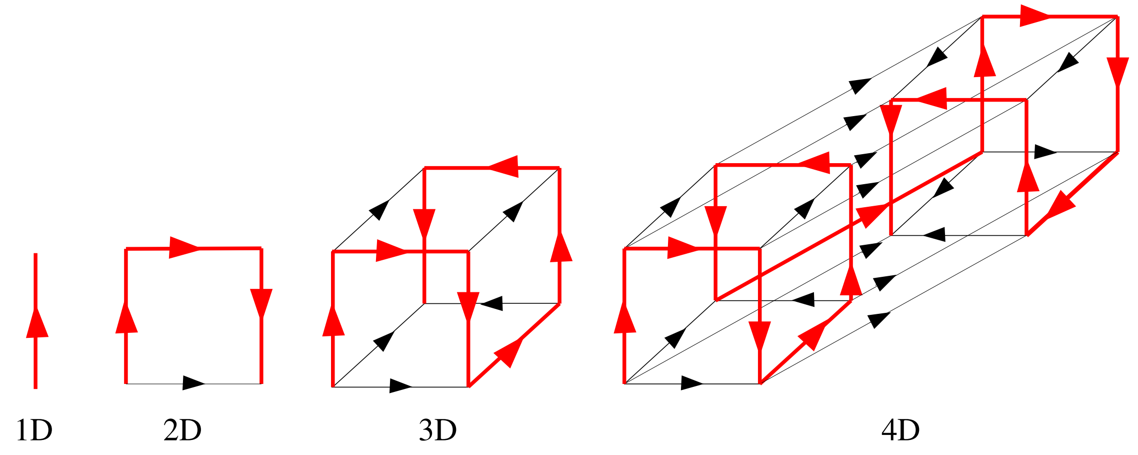


Figure 2–10: The Hamiltonian path on the Klee-Minty cube for dimensions 1, 2, 3, and 4.

Notice that these paths tend to favour some directions over others. This tendency is built into the pivot rule and is exploited by the Klee-Minty cube. Zadeh designed a new pivot rule to defeat the Klee-Minty construction.

Zadeh's rule (a.k.a. the *least entered rule*) [37]: For the entering variable, select the improving variable that has entered the basis least often thus far.

Zadeh's rule results in a path that favours the edges that the earlier pivot rules avoided. Klee and Kleinschmidt [26], Terlaky and Zhang [34], and Fathi and Tovey [11] are amongst those who think that Zadeh's rule is a good candidate for being polynomial.

Note that Zadeh's rule chooses between the $2n$ variables (n decision variables and n slack variables) whereas the next history-based rule chooses between the n pairs of decision and slack variables, (x_i, x_{n+i}), each of which defines a *direction*. Directions are not a very useful concept in arbitrary polytopes, as no two edges may be parallel, but they are a natural feature of hypercubes and are inherited by zonotopes. A zonotope that is a projection of the n-dimensional hypercube, will directly inherit the n directions of the hypercube, some of which may no longer appear.

Least-used direction rule (LUD) [4]: For the entering variable, select the improving variable whose direction has been used least often thus far.

Other history-based rules include

Least-recently considered rule [7]: Set a fixed ordering of the variables $v_1, v_2, ..., v_{2n}$ and let the previous entering variable be v_i. For the entering variable, select the improving variable that first appears in the sequence $v_{i+1}, v_{i+2}, ..., v_{2n}, v_1, ..., v_{i-1}$ (or $v_1, ..., v_{2n}$ if this is the first pivot).

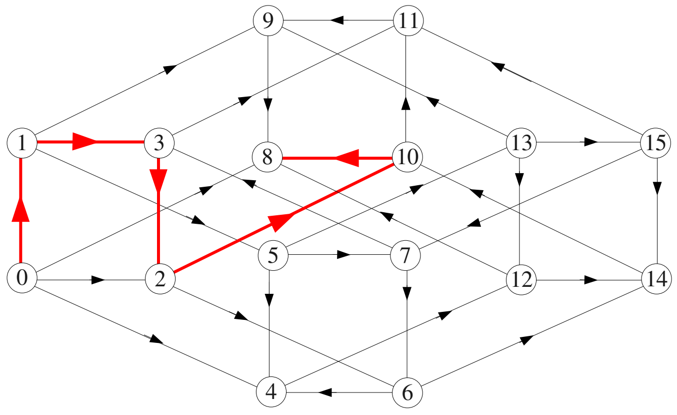
Least-recently basic rule [Johnson in [7]]: For the entering variable, select the improving variable that left the basis least recently.

Least-recently entered rule (a.k.a. *least-recently used*) [11]: For the entering variable, select the improving variable that entered the basis least recently thus far.

Least iteration in the basis rule [4]: For the entering variable, select the improving variable that has been in the basis for the least number of iterations.

The differences between these rules may be easier to understand when expressed geometrically in the special case when the feasible region and the objective function are modeled by an AUSO.

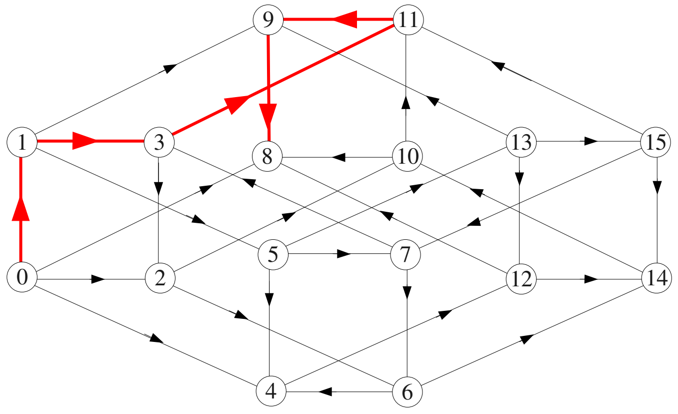
Zadeh's rule: Choose the outgoing edge whose (orientation, direction)-pair has been used least often thus far. See Figure 2–11.



Vertex		(orientation, direction)-pair								Options
		+ 4	-4	+ 3	-3	+ 2	-2	+ 1	-1	
0	0 0 0 0	0	0	0	0	0	0	0	0	**+1**, +2, +3, +4
1	0 0 0 1	0	0	0	0	0	0	1	0	**+2**, +3, +4
3	0 0 1 1	0	0	0	0	1	0	1	0	**-1**, +4
2	0 0 1 0	0	0	0	0	1	0	1	1	+3, **+4**
10	1 0 1 0	1	0	0	0	1	0	1	1	**-2**
8	1 0 0 0	1	0	0	0	1	1	1	1	

Figure 2–11: An example of Zadeh's rule. The chart shows at each step how many times each (orientation, direction)-pair has been selected, what options are available, and which option is selected. Notice at vertex 0011, -1 is an option because of the distinction between +1 and -1. Notice at vertex 1010, +1 is not an option because it has already been used once whereas -2 has never been used.

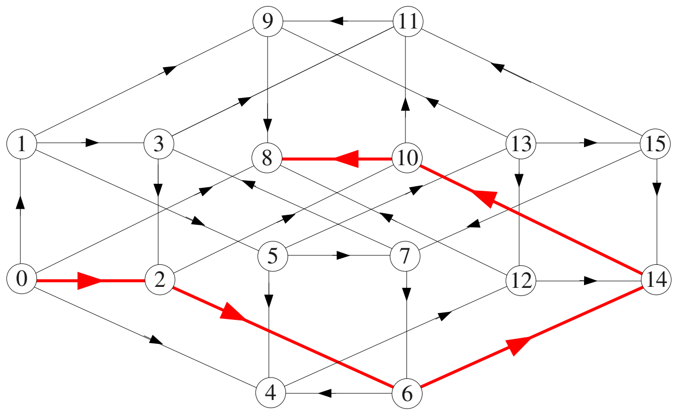
Least-used direction rule (LUD): Choose the outgoing edge whose direction (with either orientation) has been used least often thus far. See Figure 2–12.



Vertex		Direction				Options
		4	3	2	1	
0	0 0 0 0	0	0	0	0	**1**, 2, 3, 4
1	0 0 0 1	0	0	0	1	**2**, 3, 4
3	0 0 1 1	0	0	1	1	**4**
11	1 0 1 1	1	0	1	1	**2**
9	1 0 0 1	1	0	2	1	**1**
8	1 0 0 0	1	0	2	2	

Figure 2–12: An example of the LUD rule. The chart shows at each step how many times each direction has been selected, what options are available, and which option is selected. Notice at vertex 0011, 1 is not an option because it has already been used once whereas 4 has never been used.

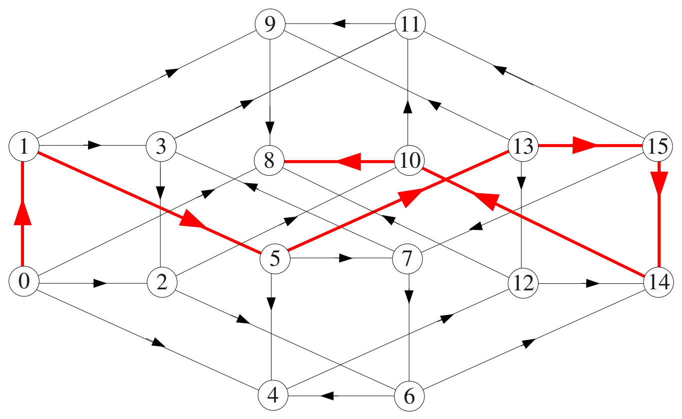
Least-recently considered rule: Set a fixed ordering of the (orientation, direction)-pairs $v_1, v_2, ..., v_{2n}$ and let the previously selected pair be v_i. Choose the outgoing edge whose pair first appears in the sequence $v_{i+1}, v_{i+2}, ..., v_{2n}, v_1, ..., v_{i-1}$ (or $v_1, ..., v_{2n}$ if this is the first pivot). See Figure 2–13.



Vertex		Sequence	Options
0	0 0 0 0	+ 2, - 2, + 1, - 1, + 3, - 3, + 4, - 4	**+ 2**
2	0 0 1 0	- 2, + 1, - 1, + 3, - 3, + 4, - 4	**+ 3**
6	0 1 1 0	- 3, + 4, - 4, + 2, - 2, + 1, - 1	**+ 4**
14	1 1 1 0	- 4, + 2, - 2, + 1, - 1, + 3, - 3	**- 3**
10	1 0 1 0	+ 4, - 4, + 2, - 2, + 1, - 1, + 3	**- 2**
8	1 0 0 0	+ 1, - 1, + 3, - 3, + 4, - 4, + 2	

Figure 2–13: An example of the least-recently considered rule. The chart shows at each step the considered sequence, what options are available, and which option is selected. Notice that there is never any choice.

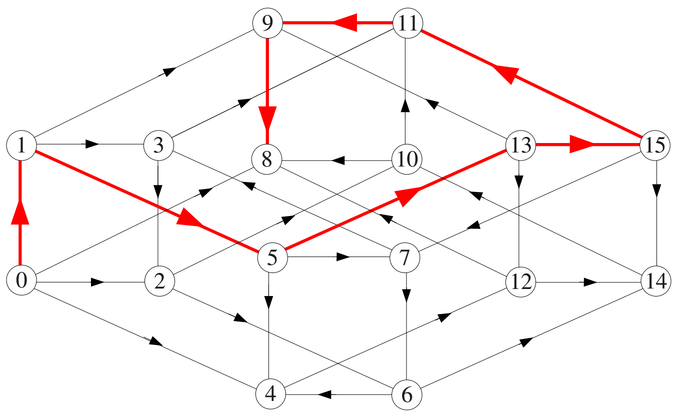
Least-recently basic rule: Choose the outgoing edge whose direction has been used least recently thus far. See Figure 2–14.



Vertex		(orientation, direction)-pair								Options
		+ 4	- 4	+ 3	- 3	+ 2	- 2	+ 1	- 1	
0	0 0 0 0		✓		✓		✓		✓	**+1**, +2, +3, +4
1	0 0 0 1		✓		✓		✓	✓		+2, **+3**, +4
5	0 1 0 1		✓	✓			✓	✓		+2, **+4**
13	1 1 0 1	✓		✓			✓	✓		**+2**
15	1 1 1 1	✓		✓		✓		✓		**-1**
14	1 1 1 0	✓		✓		✓			✓	**-3**
10	1 0 1 0	✓			✓	✓			✓	**-2**
8	1 0 0 0	✓			✓		✓		✓	

Figure 2–14: An example of the least-recently basic rule. The chart shows at each step which (orientation, direction)-pairs are in the basis, what options are available, and which option is selected. Notice at vertex 1111, -4 is not an option because it was last in the basis in iteration #3 whereas -1 was last in the basis less recently (iteration #1). Notice at vertex 1010, +1 is not an option because it was last in the basis in iteration #5 whereas -2 was in the basis less recently (iteration #4).

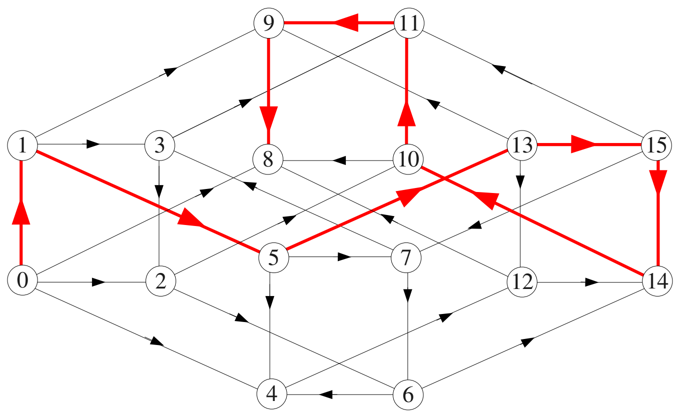
Least-recently entered rule: Choose the outgoing edge whose (orientation, direction)-pair was selected least recently thus far. See Figure 2–15.



Vertex		(orientation, direction)-pair								Options
		+ 4	- 4	+ 3	- 3	+ 2	- 2	+ 1	- 1	
0	0 0 0 0		✓		✓		✓		✓	**+1**, +2, +3, +4
1	0 0 0 1		✓		✓		✓	✓		+2, **+3**, +4
5	0 1 0 1		✓	✓			✓	✓		+2, **+4**
13	1 1 0 1	✓		✓			✓	✓		**+2**
15	1 1 1 1	✓		✓		✓		✓		-1, **-3**, -4
11	1 0 1 1	✓			✓	✓		✓		**-2**
9	1 0 0 1	✓			✓		✓	✓		**-1**
8	1 0 0 0	✓			✓		✓		✓	

Figure 2–15: An example of the least-recently entered rule. The chart shows at each step which (orientation, direction)-pairs are in the basis, what options are available, and which option is selected. Notice at vertex 1111, -3 and -4 are options because -1, -3, and -4 all last entered the basis at the same time (iteration #1).

Least iteration in the basis rule: Set a counter for each (orientation, direction)-pair to 0. At each vertex, for each direction d, increment the $+d$ counter if the vertex label has a 1 in the d^{th} counter bit and increment the $-d$ counter otherwise. Choose the outgoing edge whose (orientation, direction)-pair has the lowest count. See Figure 2–16.



Vertex		(orientation, direction)-pair								Options
		+ 4	- 4	+ 3	- 3	+ 2	- 2	+ 1	- 1	
0	0 0 0 0	0	1	0	1	0	1	0	1	**+1**, +2, +3, +4
1	0 0 0 1	0	2	0	2	0	2	1	1	+2, **+3**, +4
5	0 1 0 1	0	3	1	2	0	3	2	1	+2, **+4**
13	1 1 0 1	1	3	2	2	0	4	3	1	**+2**
15	1 1 1 1	2	3	3	2	1	4	4	1	**-1**
14	1 1 1 0	3	3	4	2	2	4	4	2	**-3**
10	1 0 1 0	4	3	4	3	3	4	4	3	**+1**, -2
11	1 0 1 1	5	3	4	4	4	4	5	3	**-2**
9	1 0 0 1	6	3	4	5	4	5	6	3	**-1**
8	1 0 0 0	7	3	4	6	4	6	6	4	

Figure 2–16: An example of the least iterations in the basis rule. The chart shows at each step how many iterations each (orientation, direction)-pair has been in the basis, what options are available, and which option is selected. Notice at vertex 1111, -3 is not an option because it has been in the basis for two iterations whereas -1 has been in the basis for less (one iteration). Notice at vertex 1010, +1 is an option because both +1 and -2 have been in the basis for the same number of iterations (four iterations).

All of the above history-based rules are deterministic. No complexity results are known for their use on LP digraphs.

The LUD pivot rule has certain qualities that make it interesting. Like Zadeh's rule [37], which it is based on, the LUD rule does not fall victim to the Klee-Minty cube. In some ways it self-balances its search for the sink and does not allow large portions of the digraph to remain unexplored. The LUD rule is a restricted form of Zadeh's rule in that all paths obeying the LUD rule also obey Zadeh's rule. The converse is not necessarily true.

In examining either of these rules, the history of the path is key. The history is given in terms of the n directions for the LUD rule and in terms of the $2n$ (orientation, direction)-pairs for Zadeh's rule. Both of these histories can be represented by an *nv vector*. Each vertex along the path has an nv vector (v_d, v_{d-1}, ..., v_1) where d is the number of directions or the number of (orientation, direction)-pairs and v_i is the number of times the path has selected direction i. In Figures 2–11 and 2–12 the nv vector at each vertex is given in the chart.

The distinction between the two rules is significant. For example, there are numerous Hamiltonian paths on AUSOs that obey Zadeh's rule, but on the 6-, 7-, and 8-dimension AUSOs there are no Hamiltonian paths that obey the LUD rule [4]. When studying the LUD rule rather than Zadeh's rule, there are fewer cases to consider. It is unclear whether having fewer cases will make a suitable construction more or less apparent.

For deterministic algorithms, all the known results are exponential. At this point, it is time to start investigating randomized pivot rules. Randomized algorithms involve an element of randomness that makes the behaviour of any given execution unpredictable but often gives good *expected* results. Two interesting randomized pivot rules are random edge and random facet.

> *Random edge*: For the entering variable, select an improving variable uniformly at random.

> *Random facet* [21]: Given a vertex v of the polytope, uniformly at random select a facet F containing v. Apply this algorithm on F until reaching $w = top(F)$. Repeat this algorithm from w.

Random edge is perhaps the most natural randomized pivot rule, but it is difficult to analyze. In a 1998 paper, Gärtner, Henk, and Ziegler [14] look at the special case of Klee-Minty cubes. They find that on these hypercubes, random edge gives a nearly quadratic upper bound on the number of pivots. In a 2006 paper, Matoušek and Szabó [29] look at the more general case of AUSOs. They present a construction of an AUSO on which random edge is expected to need a mildly exponential number of pivots. Neither of these results give definitive answers for LP digraphs. It is possible that the nearly quadratic behaviour on Klee-Minty cubes does not generalize to arbitrary LP digraphs. It is also possible that the mildly exponential behaviour on AUSOs does not occur on any LP digraphs.

The random facet rule is not intuitive like the random edge rule, but it has the best known complexity results. Kalai [20] and Matoušek, Sharir, and Welzl [28] independently found subexponential bounds for the random facet rule as stated above and its dual, respectively. This bound is for all LP digraphs. For the special case of Matoušek orientation of LP digraphs, Gärtner [13] used the Holt-Klee property of LP digraphs to show that the complexity of random facet is expected polynomial. This is a great improvement on the bound for general Matoušek orientations, and it shows how the properties of LP digraphs can be used to obtain better complexity results.

CHAPTER 3
Fibonacci Construction

One solution to the central problem of this thesis is to find a family of AUSOs, one for each dimension n, for which the length of a path obeying the LUD rule increases as an exponential function of n. A path through all 2^n vertices of each AUSO would satisfy this, but even if this is not possible, shorter exponential paths may exist. The aim of this chapter is to describe the Fibonacci construction and the families of AUSOs that it produces. It is a modification of the Fibonacci construction given by [4] for Zadeh's rule. The first section describes the general Fibonacci construction and explains how it can be used to produce families of AUSOs. The second section describes one implementation of the Fibonacci construction and the results for its family of AUSOs.

3.1 General Construction

The *Fibonacci sequence* is the sequence of numbers $f_1, f_2, ...$ such that $f_1 = 1$, $f_2 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n > 2$. Each term is simply the sum of its two preceding terms. The closed form representation $f_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$ by Binet (1843) is exponential in n.

In general, given integers $a < b < n$, a recurrence relation of the form $L_n = L_{n-a} + L_{n-b}$ with base cases $L_1, L_2, ..., L_b$ will result in a formula for L_n that is exponential in n. The Fibonacci construction hopes to exploit this fact by building a path in dimension n whose length is greater than or equal to the sum of the length of the path in dimension $(n-a)$ and the length of the path in dimension $(n-b)$.

A family of AUSOs must define an AUSO in every dimension. An AUSO can be identified by its *indegree sequence*. The indegree sequence of a digraph G is defined by $g_k(G) =$ the number of vertices with indegree equal to k, for $k = 0, ..., n$ [38].

Theorem 3.1. *If the indegree sequence of an acyclic graph is $g_k(G) = \binom{n}{k}$ for $k = 0, ..., n$ then G is an AUSO [35].*

Theorem 3.2. *Given two (possibly identical) $(n-2)$-dimensional AUSOs A and B, let A_1, A_2, and A_3 be exact copies of A. The n-dimensional hypercube given by directing all edges between A_1 and B towards B, all edges between A_2 and B towards B, all edges between A_1 and A_3 towards A_1, and all edges between A_2 and A_3 towards A_2 (see Figure 3–1) is an AUSO.*

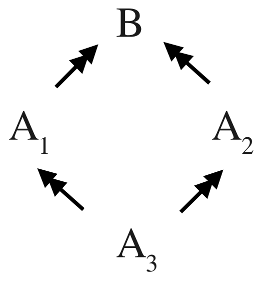


Figure 3–1: Constructing an n-dimensional AUSO from four $(n-2)$-dimensional AUSOs.

Proof. Since the original $(n-2)$ dimensional A and B are AUSOs, they each have the indegree sequence given by $g_k(A) = g_k(B) = \binom{n-2}{k}$ for $k = 0, ..., n-2$. In the new n-dimensional hypercube Z, the indegree of all the vertices in A_1 and A_2 increases by one and the indegree of all the vertices in B increases by two. Thus the indegree sequence of Z is given for $k = 0, ..., n$ by

$$\begin{aligned} g_k(Z) &= g_k(A_3) + g_{k-1}(A_1) + g_{k-1}(A_2) + g_{k-2}(B) \\ &= \left[\binom{n-2}{k} + \binom{n-2}{k-1}\right] + \left[\binom{n-2}{k-1} + \binom{n-2}{k-2}\right] \\ &= \binom{n-1}{k} + \binom{n-1}{k-1} \\ &= \binom{n}{k} \end{aligned}$$

Therefore, Z is an AUSO by Theorem 3.1. □

Schurr and Szabó [31] give a proof of Theorem 3.2 for a more general product construction.

In order to follow a path through the AUSO constructed by Theorem 3.2, it is sometimes necessary to reverse the directions of some edges. After the edge reversals the hypercube must still be an AUSO. This can be done using the following theorem.

Theorem 3.3. *Given an AUSO made up of A_1, A_2, A_3, and B as in Theorem 3.2, any number of edges between A_1 and A_3 and between A_2 and A_3 may be reversed and the resulting orientation is an AUSO.*

Proof. Let (u, v) be an edge from A_3 to A_1 (or from A_3 to A_2). Let $d(u)$ and $d(v)$ be the original indegrees of u and v. Let $D(u)$ and $D(v)$ be the indegrees of u and

v after reversing the edge (u, v). Since A_1 and A_3 are copies (and A_2 and A_3 are copies),

$$d(u) = d(v) - 1$$

Reversing (u, v) gives

$$\begin{aligned} D(u) &= d(u) + 1 \\ &= d(v) - 1 + 1 \\ &= d(v) \end{aligned}$$

and

$$\begin{aligned} D(v) &= d(v) - 1 \\ &= d(u) + 1 - 1 \\ &= d(u) \end{aligned}$$

The indegree sequence of the overall hypercube remains unchanged as the indegrees of u and v have simply been interchanged. Therefore, the hypercube after any number of reversals of this type is still an AUSO by Theorem 3.1. □

Note that this is not true for edges between B and A_1 or for edges between B and A_2 (see Figure 3–2).

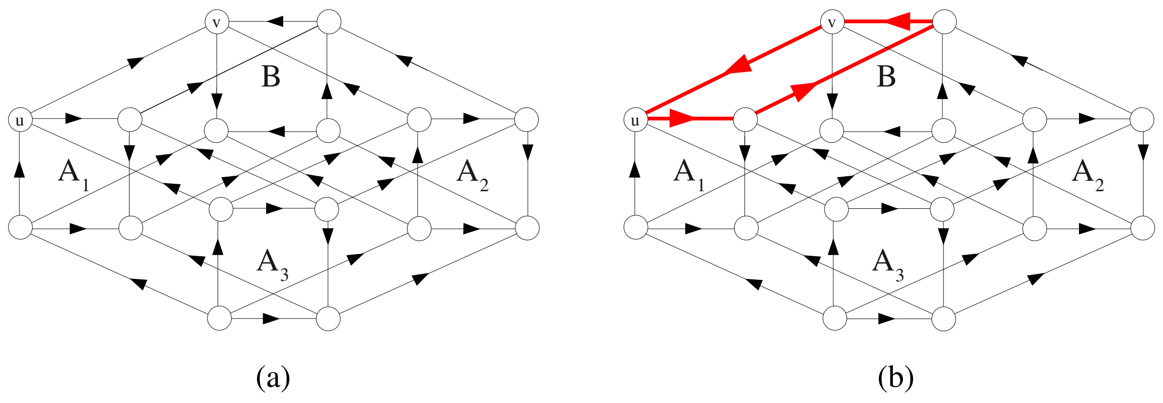


Figure 3–2: An example of an invalid edge reversal. (a) An AUSO as described in Theorem 3.2. (b) The digraph obtained by reversing the edge (u, v). This graph is no longer an AUSO because it contains the highlighted cycle.

The Fibonacci construction uses Theorems 3.2 and 3.3 to guarantee that the constructed hypercube is an AUSO in all dimensions. Let Z_n denote the n-dimensional AUSO constructed by the Fibonacci construction and let L_n denote the length of the LUD path on Z_n. The construction of Z_n is as follows for both the even case (see Figure 3–3) and the odd case (see Figure 3–4):

1. Make three copies of Z_{n-2}, A_1, A_2, and A_3, and one undefined $(n-2)$-dimensional hypercube, B.
2. Create a n-dimensional hypercube by directing all edges between A_1, A_2, A_3, and B as in Theorem 3.2.
3. Set the starting vertex in A_1.
4. In A_1, follow Z_{n-2}'s path until the LUD rule is left with no option but to select an edge from A_1 to B.
5. Instead of going to B, do the following:

- i Reverse the edge between the current vertex in A_1 and its neighbour in A_3 as in Theorem 3.3.
- ii Follow the newly reversed edge to A_3 and immediately follow the edge leading to A_2.
- iii Follow the next edge of the $(n-2)$-dimensional path.
- iv Reverse the edge between the current vertex in A_2 and its neighbour in A_3 as in Theorem 3.3.
- v Follow the newly reversed edge to A_3 and immediately follow the edge leading to A_1.

6. Repeat Steps 4 and 5 until Z_{n-2}'s path is complete.

This construction defines a large family of AUSOs.

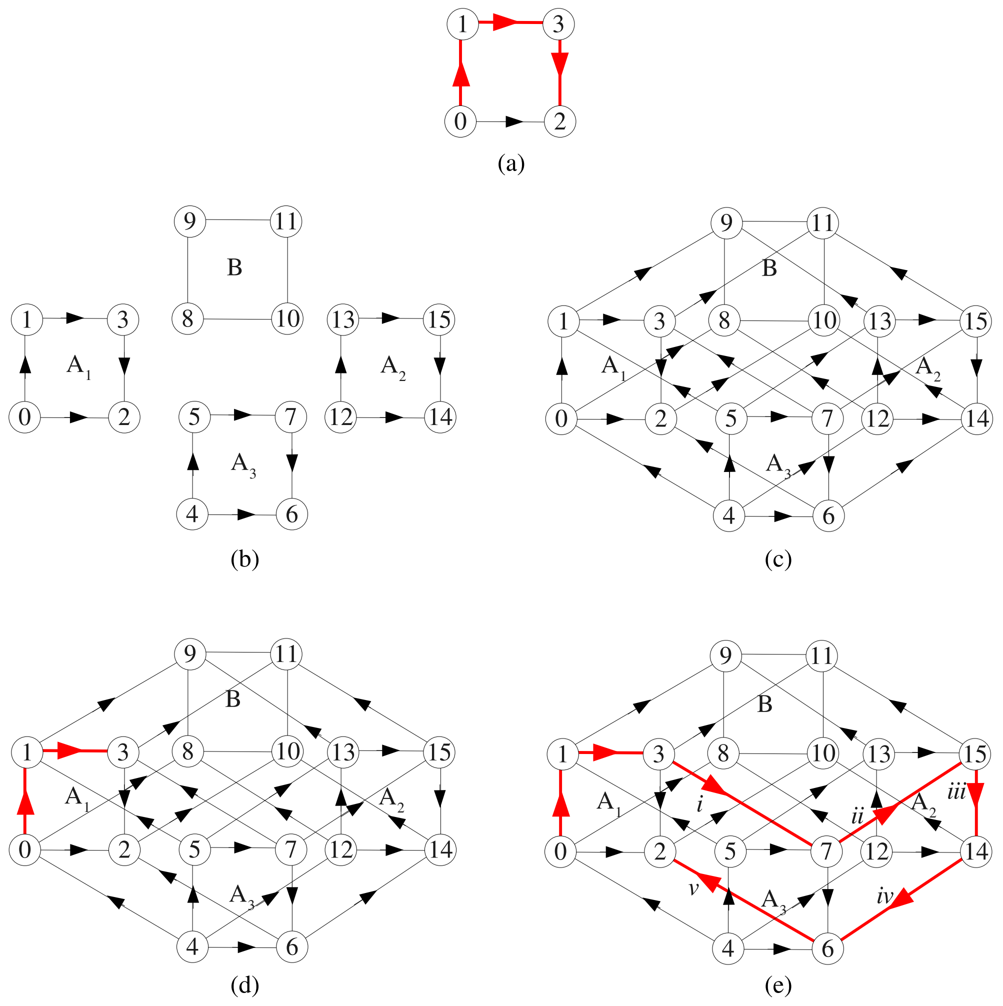
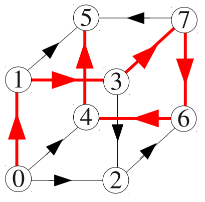
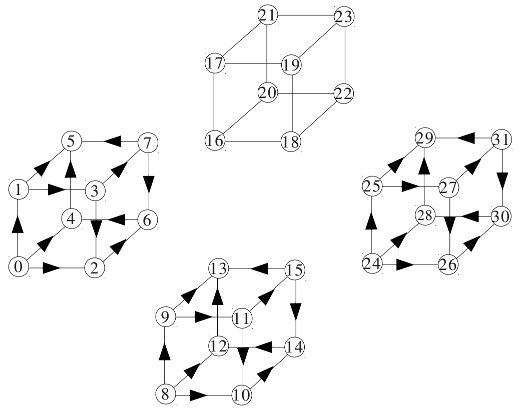
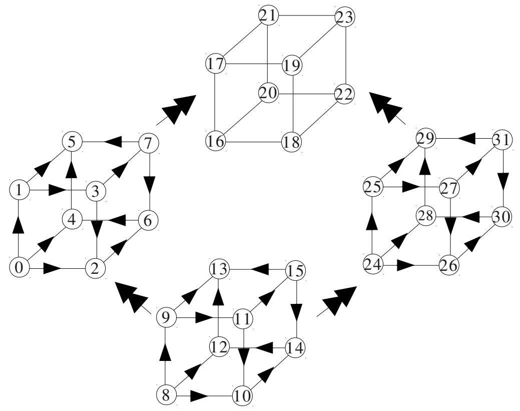
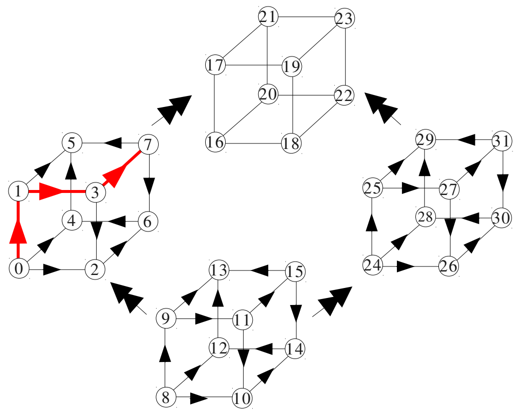


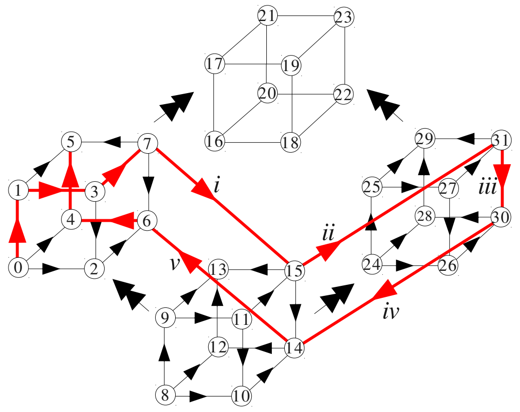
Figure 3–3: An example of the Fibonacci construction of Z_4 from Z_2 and Z_2's path. (a) Z_2 and its path. (b) Step 1. (c) Step 2. (d) Step 3. At vertex 3 the nv vector is (0, 0, 1, 1). As a result the LUD rule's only option is to select direction 3. (e) Steps 4 and 5.


(a)


(b)


(c)


(d)



(e)

Figure 3–4: An example of the Fibonacci construction of Z_5 from Z_3 and Z_3's path. (a) Z_3 and its path. (b) Step 1. (c) Step 2. (d) Step 3. At vertex 7 the nv vector is (0, 0, 1, 1, 1). As a result the LUD rule's only option is to select direction 4.(e) Steps 4 and 5.

The family of AUSOs defined by the Fibonacci construction has three properties that are desirable in the search for exponential paths. Firstly, it guarantees that the $(n-2)$-dimensional path can be completed on the n-dimensional AUSO; therefore, $L_n \geq L_{n-2}$. Secondly, in completing the $(n-2)$-dimensional path, the edges of B are never used. Lastly, the orientation of B may be that of any AUSO. The last two properties leave great freedom to define subfamilies of AUSOs that will perhaps extend the n-dimensional path to $L_n \geq L_{n-2} + L_{n-a}$ for some $a \geq 2$.

If the nv vector were completely balanced before entering B, then the path could be extended by L_{n-2} simply by ensuring that the orientation of B is that of Z_{n-2} rotated so that the path arrives on its starting vertex, but this is not possible. To be completely balanced either the path has returned to the starting vertex (each direction is used an even number of times) or the path has reached the vertex antipodal to the starting vertex (each direction is used an odd number of times). The former is not possible because the hypercube is an AUSO. The latter is not possible because directions $(n-1)$ and n are always used an even number of times. Therefore, the nv vector cannot be completely balanced going into B.

Examination of the nature of the unbalanced nv vector before entering B reveals some general properties. The directions $1, ..., (n-2)$ will be precisely Z_{n-2}'s final nv vector, and the directions $(n-1)$ and n are used the same number of times. The directions $(n-2)$ and $(n-3)$ will be under-used compared to the others because in the previous iteration, they are not used after the path moves into its B hypercube.

3.2 Implementation

The most straight-forward implementation of the Fibonacci construction that addresses the issue of two under-used directions is one that attempts to complete Z_{n-4}'s path in B. For this, the construction of B is as follows:

1. Divide B into four identical hypercubes, b_1, b_2, b_3, and b_4 where b_4 contains the vertex that the path arrives on.

   The edges between b_1 and b_2 are direction $n-2$ and are directed towards b_1.

   The edges between b_1 and b_3 are direction $n-3$ and are directed towards b_1.

   The edges between b_2 and b_4 are direction $n-3$ and are directed towards b_2.

   The edges between b_3 and b_4 are direction $n-2$ and are directed towards b_3.

2. Ignoring the two under-used directions, order the remaining $n-4$ directions from least-used to most-used and order directions that have been used the same number of times numerically from smallest to largest. This gives a mapping of the directions of Z_{n-4} to the directions of b_i. Direction 1 of Z_{n-4} is mapped to the the first direction of b_i in the ordering, direction 2 to the second, and so on.

3. Set the starting vertex of each b_i to the vertex corresponding to the arrival vertex in b_4. From the starting vertex, direct the edges of b_i according to Z_{n-4} using the mapping in Step 2.

4. Map Z_{n-4}'s path using the mapping in Step 2.

5. In the current b_i, follow the mapped path from Step 4 until the LUD rule is left with no option but to take different direction.

6. Follow the least-used direction with the smallest subscript; reverse any edges between b_2 and b_4 and between b_3 and b_4 as necessary using Theorem 3.3.
7. Repeat Steps 5 and 6 until the path of Step 4 is complete or upon reaching the sink.

This construction gives Z_n. If the stopping condition is the completion of the mapped path, then this is an exponential construction. Otherwise, the stopping condition is reaching the sink of Z_n before completing the mapped path; in this case, this may not be an exponential construction if the sink is reached too quickly.

Smaller dimensional cases can be done by hand, but for larger dimensions and for greater accuracy a computer program is more useful. Therefore, one was written that follows the above construction with various base cases [10].

A typical example starts with the base cases in Figure 3–5.

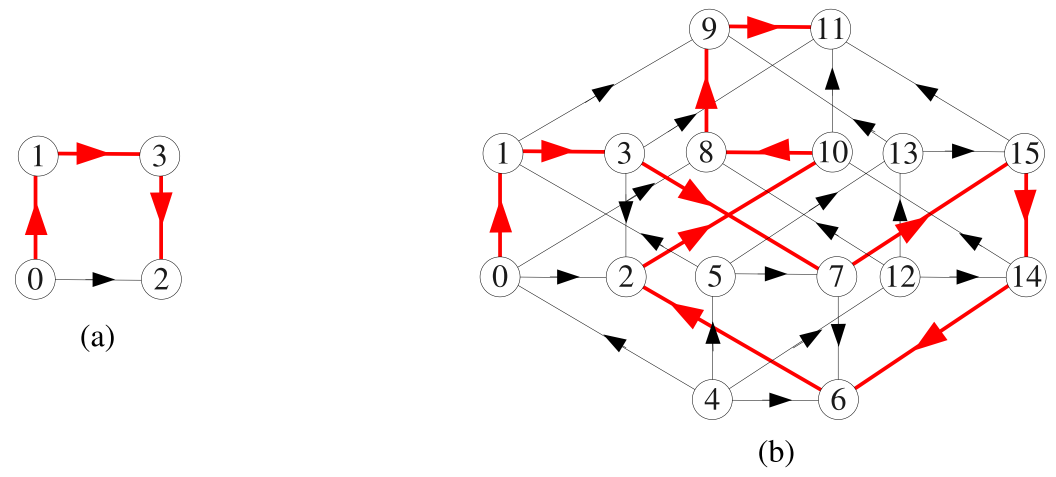
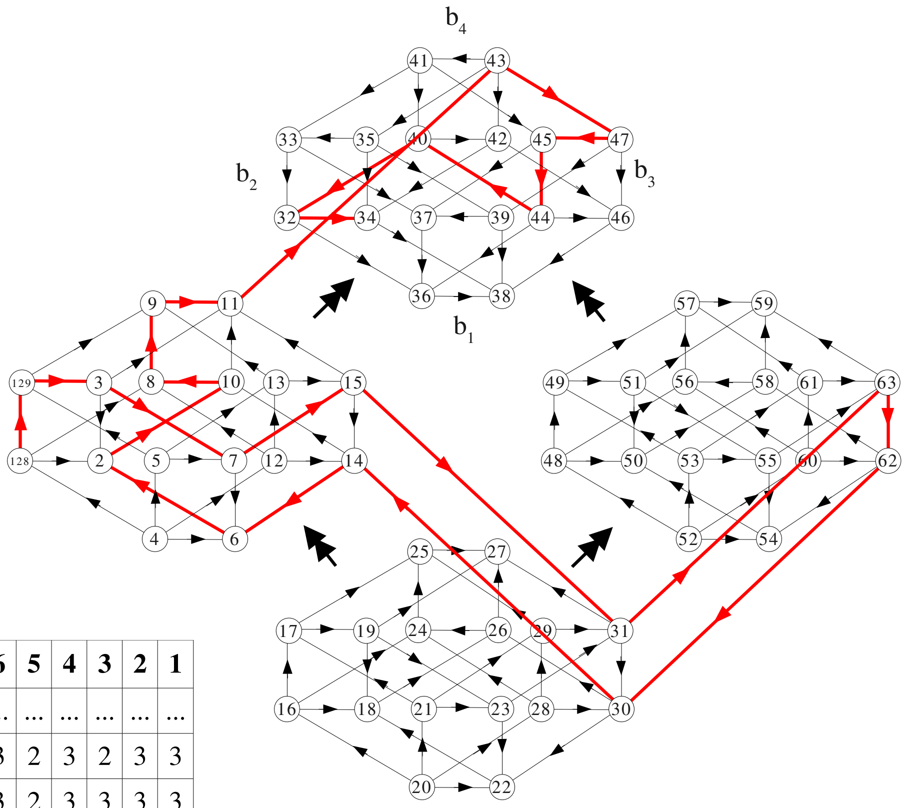


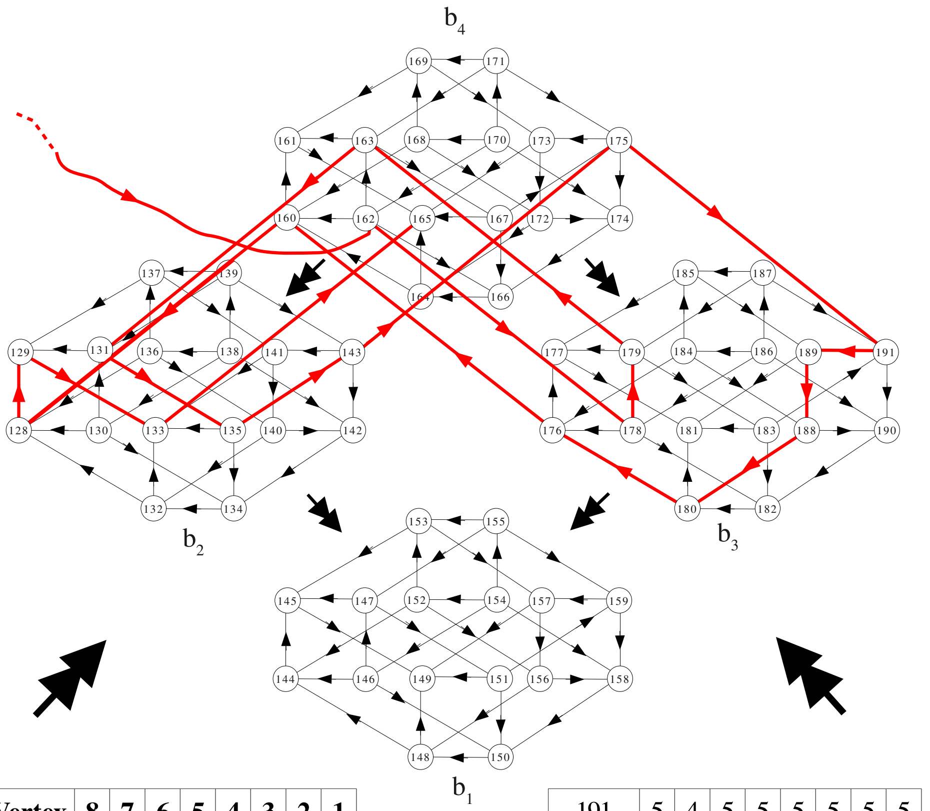
Figure 3–5: The base cases for the example of the implemented Fibonacci construction. (a) Z_2 and its path. (b) Z_4 and its path.

The example continues with the construction of Z_6 (see Figure 3–6) and Z_8 (see Figure 3–7).



Vertex	6	5	4	3	2	1
...	...	...	...	...	...	...
43	3	2	3	2	3	3
47	3	2	3	3	3	3
45	3	2	3	3	4	3
37	3	2	4	3	4	3
33	3	2	4	4	4	3
32	3	2	4	4	4	4
34	3	2	4	4	5	4

Figure 3–6: An example of the implemented Fibonacci construction. The construction of Z_6 and its path. Both the 4-dimensional path and the 2-dimensional mapped path are completed. The chart shows the nv vector at each step of the path after reaching B.



Vertex	8	7	6	5	4	3	2	1
...	...	...	...	...	...	...	...	...
162	5	4	3	2	4	4	5	4
178	5	4	3	3	4	4	5	4
179	5	4	3	3	4	4	5	5
163	5	4	3	4	4	4	5	5
131	5	4	4	4	4	4	5	5
135	5	4	4	4	4	5	5	5
143	5	4	4	4	5	5	5	5
175	5	4	5	4	5	5	5	5

191	5	4	5	5	5	5	5	5
189	5	4	5	5	5	5	6	5
188	5	4	5	5	5	5	6	6
180	5	4	5	5	6	5	6	6
176	5	4	5	5	6	6	6	6
160	5	4	5	6	6	6	6	6
128	5	4	6	6	6	6	6	6
129	5	4	6	6	6	6	6	7
133	5	4	6	6	6	7	6	7
165	5	4	7	6	6	5	6	7

Figure 3–7: An example of the implemented Fibonacci construction. The B hypercube of the construction of Z_8 and its path. The 4-dimensional mapped path is not completed in B. The chart shows the nv vector at each step of the path after reaching B.

The results in Figure 3–8 are produced by using the computer program [10] to continue the example in Figures 3–5, 3–6, and 3–7.

dim		**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Total path length**	**Finishes both paths**	**> length of both paths**
2																						1	2	3	N/A	N/A
4																				3	2	3	3	11	N/A	N/A
6																		3	2	4	4	5	4	22	Yes	Yes
8																5	4	7	6	6	7	6	7	48	No	Yes
10														7	6	9	8	8	8	9	9	9	9	82	No	No
12												9	8	11	12	10	11	10	11	11	12	11	11	127	No	No
14										13	12	13	14	12	13	13	13	13	13	13	12	14	14	182	No	No
16								15	14	17	15	15	15	16	16	15	16	16	17	16	18	14	17	252	No	No
18						19	18	18	18	17	18	19	19	18	19	19	19	19	18	19	19	20	17	333	No	No
20				21	20	21	22	23	22	23	22	22	22	22	22	22	23	23	23	22	23	21	25	444	No	No
22		25	24	23	26	24	25	23	26	25	25	25	25	25	26	25	23	26	27	25	26	27	26	552	No	No

Figure 3–8: A typical example of the final nv vectors of the 6- to 22-dimensional AUSOs produced from the implemented Fibonacci construction with cases from 3–6.

This subfamily does not appear to give an exponential LUD path.

CHAPTER 4
Lower Bound

The aim of this chapter is to show that the least-used (orientation, direction)-pair is used an exponential number of times on Hamiltonian paths that follow Zadeh's rule.

Hamiltonian paths are very special. Not only does one give an exponential path, but it also completely defines an AUSO. Suppose that the vertices of the Hamiltonian path were ordered from 0 to $(2^n - 1)$. This is a completely unimodal numbering of the n-dimensional hypercube. All edges of the AUSO will be directed from the lower-numbered vertices to the higher-numbered vertices. This is more memory efficient as an AUSO defined by a Hamiltonian path need only remember the order of the vertices rather than the direction of every edge.

If the least-used pair grows subexponentially while the most-used pair grows exponentially then the imbalance in the nv vector would prevent constructions from relying on a balanced vector. Therefore, the following exponential lower bound on the least-used (orientation, direction)-pair is important.

Theorem 4.1. *In any Hamiltonian path following Zadeh's rule on an n-dimensional AUSO, every (orientation, direction)-pair is used at least $\frac{2^n-2^{n-1}-1}{2n}$ times.*

Proof. Partition the n-dimensional AUSO into two $(n-1)$-dimensional hypercubes C_1 and C_2 where the direction d separates the two (see Figure 4–1).

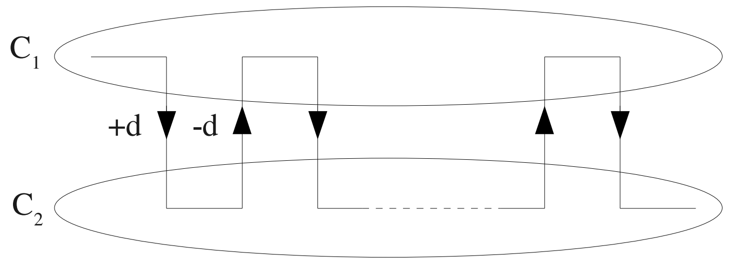


Figure 4–1: The two halfcubes C_1 and C_2 separated by the direction d.

Let k be the number of times direction $-d$ is used. Without loss of generality, assume the Hamiltonian path starts in C_1.

The Hamiltonian path must visit each of the 2^n vertices. It has length $2^n - 1$, so the sum of the values of the final nv vector is $2^n - 1$.

This sum can also be counted by looking at how high each individual (orientation, direction)-pair can be. Each of the $2n$ (orientation, direction)-pairs of the nv vector can be either k or $k+1$ while the vector remains balanced. This contributes at most $2n(k+1) - 1$ to the sum of the values of the nv vector. Each vertex in C_1 has an adjacent vertex in C_2. If the path reaches a vertex in C_1 whose adjacent vertex in C_2 is already in the path or a vertex in C_2 whose adjacent vertex is in C_1, then the path could potentially select a direction that has already been used $k+1$ times. This can occur only once for each pair of vertices, so contributes at most 2^{n-1} to the sum of the values of the nv vector.

Now count the sum of the nv vector in two different ways and solve for k:

$$2^n - 1 \leq 2n(k+1) - 1 + 2^{n-1}$$
$$k \geq \frac{2^{n-1} - 2^{n-2}}{n} - 1$$

which is exponential in n. $\square$

The proof of Theorem 4 only holds when the path is Hamiltonian. It is possible for a non-Hamiltonian exponential path to use an (orientation, direction)-pair a subexponential number of times. It is even possible that a pair is never used (see Figure 4–2). Since in some dimensions there are no Hamiltonian paths that follow the LUD rule, Theorem 4 cannot be applied directly to the LUD rule.

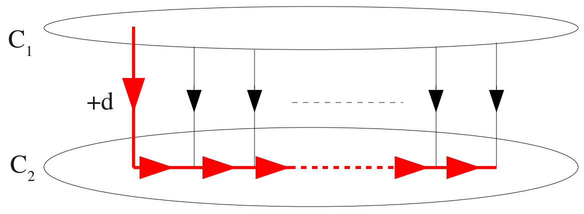


Figure 4–2: An example of a non-Hamiltonian exponential path where one (orientation, direction)-pair is never used. When all the directions from one halfcube is directed towards the other halfcube, then the exponential path consisting of all the vertices in the second halfcube ($\frac{n^2}{2}$ vertices) never uses the direction $-d$.

CHAPTER 5
Conclusion

The question of whether or not there exists an exponential path that obeys the LUD rule on an AUSO is still open. It is an interesting problem whose solution would be valuable for the field of linear programming as well as USOs. The aim of this chapter is to summarize this thesis and to outline some areas of promising future work.

The background material covered in Chapter 2 brings together linear programming, the Simplex method, and AUSOs in a structured way. The dictionary and geometrical representations of the feasible region and the concept of pivoting are presented in a manner that allows them to be used directly on LP digraphs and analogously on AUSOs. Definitions of the history-based pivot rules, namely Zadeh's rule, the least-used direction rule, the least-recently considered rule, the least-recently basic rule, the least-recently entered rule, and the least iterations in the basis rule, are given in the dictionary representation and, for AUSOs, in the geometrical representation. An example AUSO is used to illustrate the differences between these six history-based rules.

The Fibonacci construction presented in Chapter 3 can be used to produce many different families of AUSOs depending on the construction of the B hypercube. Following the $(n-2)$-dimensional path cannot work mainly because two directions will become vastly underused as n grows. Following the $(n-4)$-dimensional path in order does not succeed mainly because of the unbalanced nature of the nv vector upon arriving to the B hypercube.

In related future work, the goal remains to complete a path in the B cube. There are at least three ways in which this could be achieved. First, there are multiple paths of dimension lower than $(n-4)$ that will produce superpolynomial complexity results. One is to use some path of dimension $(n-a)$ for some $a > 4$, and another is to use some path of dimension $\frac{n}{d}$ where $d > 1$. Second, the path in B may be performed in a different sequence. This could preserve the length of the path, and might compensate for the unbalanced nv vector described in Chapter 3. Third, the path in B may be completed on, say, every second iteration of the Fibonacci construction. As above, this could also preserve the length of the path and might also compensate for the unbalanced nv vector. For example, when building the B cube of the n-dimensional AUSO, rather than trying to complete a path, try to re-balance the nv vector. If this can be done effectively, then upon entering the B cube of the $(n+2)$-dimensional AUSO there is a much more balanced nv vector and following the $(n-2)$-dimensional path in B might now be possible.

The lower bound proven in Chapter 4 shows that the nv vector does not get too far out of balance. However, this result is for Zadeh's rule rather than the LUD rule, and it assumes the existence of a Hamiltonian path that follows it. A lower bound assuming the existence of any exponential path, not just a

Hamiltonian one would be a major improvement. Under that assumption, the result would hold for both Zadeh's rule and the LUD rule.

References

[1] I. Adler and R. Saigal. Long monotone paths in abstract polytopes. *Mathematics of Operations Research*, 1(1):89–95, 1976.

[2] D. Avis and V. Chvátal. Notes on Blands pivoting rule. *Polyhedral Combinatorics*, 8:24–34, 1978.

[3] D. Avis and S. Moriyama. On Combinatorial Properties of Linear Program Digraphs. *Polyhedral Computation*, 48:1, 2009.

[4] D. Avis, S. Moriyama, and Y. Matsumoto. History based pivot rules and unique sink orientations, Japan-Canada Workshop, July 2009.

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

[6] V. Chvátal. *Linear programming.* WH Freeman, 1983.

[7] W. H. Cunningham. Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 4(2):196–208, 1979.

[8] G.B. Dantzig. *Linear programming and extensions.* Princeton Univ Pr, 1963.

[9] C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou. The complexity of computing a Nash equilibrium. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 71–78. ACM New York, NY, USA, 2006.

[10] T. Deering. Implementation of the fibonacci construction. `http://www.cs.mcgill.ca/~tdeeri/fib_program.zip`, 2010.

[11] Y. Fathi and C. Tovey. Affirmative action algorithms. *Mathematical Programming*, 34(3):292–301, 1986.

[12] L. Ford and D. Fulkerson. Flows in networks. 1962.

[13] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures and Algorithms*, 20(3):353–381, 2002.

[14] B. Gärtner, M. Henk, and G.M. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, 18(3):349–372, 1998.

[15] B. Gärtner and I. Schurr. Linear programming and unique sink orientations. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, page 757. ACM New York, NY, USA, 2006.

[16] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[17] D.S. Hochba. Approximation algorithms for NP-hard problems. *ACM SIGACT News*, 28(2):40–52, 1997.

[18] F. Holt and V. Klee. A proof of the strict monotone 4-step conjecture. In *Advances in discrete and computational geometry: proceedings of the 1996 AMS-IMS-SIAM joint summer research conference, Discrete and Computational Geometry–Ten Years Later, July 14-18, 1996, Mount Holyoke College*, volume 233, page 201. Amer Mathematical Society, 1998.

[19] R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973.

[20] G. Kalai. A subexponential randomized simplex algorithm (extended abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 475–482. ACM New York, NY, USA, 1992.

[21] G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79(1):217–233, 1997.

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

[23] L.G. Khachiian. Polynomial algorithms in linear programming. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 20:51–68, 1980.

[24] E.D. Kim and F. Santos. An update on the Hirsch conjecture: Fifty-two years later. *Available at arXiv*, 0907.1186v2.

[25] V. Klee. On the number of vertices of a convex polytope, 1963.

[26] V. Klee and P. Kleinschmidt. The d-step conjecture and its relatives. *Mathematics of Operations Research*, 12(4):718–755, 1987.

[27] V. Klee and G.J. Minty. How good is the simplex algorithm?, 1970.

[28] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4):498–516, 1996.

[29] J. Matoušek and T. Szabó. RANDOM EDGE can be exponential on abstract cubes. *Advances in Mathematics*, 204(1):262–277, 2006.

[30] F. Santos. A counterexample to the Hirsch conjecture. *Available at arXiv*, 1006.2814v1.

[31] I. Schurr and T. Szabó. Jumping doesn't help in abstract cubes. *Lecture Notes in Computer Science*, 3509:225–235, 2005.

[32] A. Stickney and L. Watson. Digraph models of Bard-type algorithms for the linear complementarity problem. *Mathematics of Operations Research*, 3(4):322–333, 1978.

[33] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 547–555, 2001.

[34] T. Terlaky and S. Zhang. Pivot rules for linear programming: a survey on recent theoretical developments. *Annals of Operations Research*, 46(1):203–233, 1993.

[35] K. Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20(1):81, 1988.

[36] S.J. Wright. *Primal-dual interior-point methods.* Society for Industrial Mathematics, 1997.

[37] N. Zadeh. What is the worst case behavior of the simplex algorithm? Technical Report 27, Stanford University, Standford, CA, 1980.

[38] G.M. Ziegler. *Lectures on polytopes.* Springer, 1995.