LINEAR PROGRAMMING[1]

by

V. Chandru[2] & M. R. Rao[3]

March 1998

Please address all correspondence to :

Dr. M. R. Rao
Indian Institute of Management Bangalore
Bannerghatta Road
Bangalore 560 076
India

Fax: (080) 6644050

---

[1] *To appear as a chapter of the Handbook of Algorithms edited by M.J. Atallah, CRC Press (1998)*

[2] *CS & Automation, Indian Institute of Science, Bangalore-560 012, India. chandru@csa.iisc.ernet.in*

[3] *Indian Institute of Management - Bangalore, Bangalore 560 076, India. mrao@iimb.ernet.in*

Copies of the Working Papers may be obtained from the FPM & Research Office

LINEAR PROGRAMMING

VIJAY CHANDRU, Indian Institute of Science, Bangalore 560 012, India.
M.R. RAO, Indian Institute of Management, Bangalore 560 076, India.

NOVEMBER, 1997

Dedicated to George Dantzig on this the 50^{th} Anniversary of the Simplex Algorithm

Abstract

Linear programming has been a fundamental topic in the development of the computational sciences. The subject has its origins in the early work of L.B.J. Fourier on solving systems of linear inequalities, dating back to the 1820's. More recently, a healthy competition between the simplex and interior point methods has led to rapid improvements in the technologies of linear programming. This combined with remarkable advances in computing hardware and software have brought linear programming tools to the desktop, in a variety of application software for decision support. Linear programming has provided a fertile ground for the development of various algorithmic paradigms. Diverse topics such as symbolic computation, numerical analysis, computational complexity, computational geometry, combinatorial optimization, and randomized algorithms all have some linear programming connection. This chapter reviews this universal role played by linear programming in the science of algorithms.

1 Introduction

Linear programming has been a fundamental topic in the development of the computational sciences [49]. The subject has its origins in the early work of L.B.J. Fourier [29] on solving systems of linear inequalities, dating back to the 1820's. The revival of interest in the subject in the 1940's was spearheaded by G.B.Dantzig [18] in USA and L.V.Kantorovich [44] in the erstwhile USSR. They were both motivated by the use of linear optimization for optimal resource utilization and economic planning. Linear programming, along with classical methods in the calculus of variations, provided the foundations of the field of mathematical programming, which is largely concerned with the theory and computional methods of mathematical optimization. The 1950's and 1960's marked the period when linear programming fundamentals (duality, decomposition theorems, network flow theory, matrix factorizations) were worked out in conjunction with the advancing capabilities of computing machinery [19].

The 1970's saw the realization of the commercial benefits of this huge investment of intellectual effort. Many large-scale linear programs were formulated and solved on mainframe computers to support applications in industry (for example: Oil, Airlines) and for the state (for example: Energy Planning, Military Logistics). The 1980's were an exciting period for linear programmers. The polynomial time-complexity of linear programming was established. A healthy competition between the simplex and interior point methods ensued which finally led to rapid improvements in the technologies of linear programming. This combined with remarkable advances in computing hardware and software have brought linear programming tools to the desktop, in a variety of application software (including spreadsheets) for decision support.

The fundamental nature of linear programming in the context of algorithmics is borne out by a few examples.

- Linear programming is at the starting point for variable elimination techniques on algebraic constraints [11] which in turn forms the core of algebraic and symbolic computation.
- Numerical linear algebra and particularly sparse matrix technology was largely driven in its early development by the need to solve large-scale linear programs [36,61].
- The complexity of linear programming played an important role in the 1970's in the early stages of the development of the polynomial hierarchy and particularly in the $\mathcal{NP}$-completeness and $\mathcal{P}$-completeness in the theory of computation [67,68].
- Linear-time algorithms based on "prune and search" techniques for low-dimensional linear programs have been used extensively in the development of computational geometry [24].
- Linear programming has been the testing ground for very exciting new devlopments in randomized algorithms [60].
- Relaxation strategies based on linear programming have played a unifying role in the construction of approximation algorithms for a wide variety of combinatorial optimization problems [16, 32,74].

In this chapter we will encounter the basic algorithmic paradigms that have been invoked in the solution of linear programs. An attempt has been made to provide intuition about some fairly deep and technical ideas without getting bogged down in details. However, the details are important and the interested reader is invited to explore further through the references cited. Fortunately, there are many excellent texts, monographs and expository papers on linear programming [8,5,15,19,62, 65,67,69,71], that the reader can choose from, to dig deeper into the fascinating world of linear programming.

2 Geometry of Linear Inequalities

Two of the many ways in which linear inequalities can be understood are through algorithms or through non-constructive geometric arguments. Each approach has its own merits (æsthetic and otherwise). Since the rest of this chapter will emphasize the algorithmic approaches, in this section we have chosen the geometric version . Also, by starting with the geometric version, we hope to hone the reader's intuition about a convex polyhedron, the set of solutions to a finite system of linear inequalities[4]. We begin with the study of linear, homogeneous inequalities. This involves the geometry of (convex) polyhedral cones.

2.1 Polyhedral Cones

A homogeneous linear equation in n variables defines a *hyperplane* of dimension (n-1) which contains the origin and is therefore a linear subspace. A homogeneous linear inequality defines a *halfspace* on one "side" of the hyperplane, defined by converting the inequality into an equation. A system of linear homogeneous inequalities therefore, defines an object which is the intersection of finitely many halfspaces, each of which contains the origin in its boundary. A simple example of such an object is the non-negative orthant. Clearly the objects in this class resemble cones with the apex defined at the origin and with a prismatic boundary surface. We call them *convex polyhedral cones.*

A convex polyhedral cone is the set of the form

$$\mathcal{K} = \{x | Ax \leq 0\}$$

Here A is assumed to be an $m \times n$ matrix of real numbers. A set is convex if it contains the line segment connecting any pair of points in the set. A convex set is called a convex cone if it contains all non-negative, scalar multiples of points in it. A convex set is polyhedral if it is represented by a finite system of linear inequalities. As we shall deal exclusively with cones that are both convex and polyhedral, we shall refer to them as cones.

The representation of a cone as the solutions to a finite system of homogeneous linear inequalities is sometimes, referred to as the "constraint" or "implicit" description. It is implicit because it takes an algorithm to generate points in the cone. An "explicit" or "edge" description can also be derived for any cone.

Theorem 2.1 *Every cone* $\mathcal{K} = \{x : Ax \leq 0\}$ *has an "edge" representation of the following form.* $\mathcal{K} = \{x : x = \sum_{j=1}^{L} e^j \mu_j,\ \mu_j \geq 0\ \forall j\}$ *where each distinct edge of* $\mathcal{K}$ *is represented by a point* e^j.

[4]For the study of infinite systems of linear inequalities see Chapter (NOTE TO EDITOR: CROSS-REFEERENCE CHAPTER BY VAVASIS ON CONVEX PROGRAMMING HERE) of this handbook

Thus, for any cone we have two representations:

- CONSTRAINT REPRESENTATION: $\mathcal{K} = \{x : Ax \leq 0\}$
- EDGE REPRESENTATION: $\mathcal{K} = \{x : x = E\mu,\ \mu \geq 0\}$

The matrix E is a representation of the edges of $\mathcal{K}$. Each column $E_{i.}$ of E contains the coordinates of a point on a distinct edge. Since positive scaling of the columns is permitted, we fix the representation by scaling each column so that the last non-zero entry is either 1 or -1. This scaled matrix E is called the Canonical Matrix of Generators of the cone $\mathcal{K}$.

Every point in a cone can be expressed as a positive combination of the columns of E. Since the number of columns of E can be huge, the edge representation does not seem very useful. Fortunately, the following tidy result helps us out.

Theorem 2.2 (Caratheodory) *[10] For any cone $\mathcal{K}$, every $\bar{x} \in \mathcal{K}$ can be expressed as the positive combination of at most d edge points, where d is the dimension of $\mathcal{K}$.*

CONIC DUALITY

The representation theory for convex polyhedral cones exhibits a remarkable duality relation. This duality relation is a central concept in the theory of linear inequalities and linear programming as we shall see later.

Let $\mathcal{K}$ be an arbitrary cone. The *dual* of $\mathcal{K}$ is given by

$$\mathcal{K}^* = \{u : x^T u \leq 0,\ \forall x \in \mathcal{K}\}$$

Theorem 2.3 *The representations of a cone and its dual are related by*
$\mathcal{K} = \{x : Ax \leq 0\} = \{x : x = E\mu,\ \mu \geq 0\}$ *and*
$\mathcal{K}^* = \{u : E^T u \leq 0\} = \{u : u = A^T\lambda,\ \lambda \geq 0\}$

Corollary 2.4 *$\mathcal{K}^*$ is a convex polyhedral cone and duality is involutory (that is $(\mathcal{K}^*)^*). = \mathcal{K}$.*

As we shall see, there is not much to linear inequalities or linear programming, once we have understood convex polyhedral cones.

2.2 Convex Polyhedra

The transition from cones to polyhedra may be conceived of, algebraically, as a process of dehomogenization. This is to be expected, of course, since polyhedra are represented by systems of (possibly inhomogeneous) linear inequalities and cones by systems of *homogeneous* linear inequalities. Geometrically, this process of dehomogenization corresponds to realizing that a polyhedron is the Minkowski

or set sum of a cone and a polytope (bounded polyhedron). But before we establish this identity, we need an algebraic characterization of polytopes. Just as cones in $\Re^n$ are generated by taking *positive* linear combinations of a finite set of points in $\Re^n$, polytopes are generated by taking *convex* linear combinations of a finite set of (generator) points.

Definition: Given K points $\{x^1, x^2, \cdots, x^K\}$ in $\Re^n$ the *Convex Hull* of these points is given by

$$C.H.(\{x^i\}) = \{\bar{x} : \bar{x} = \sum_{i=1}^{K} \alpha_i x^i, \ \sum_{i=1}^{K} \alpha_i = 1, \ \alpha \geq 0\}$$

i.e. the convex hull of a set of points in $\Re^n$ is the object generated in $\Re^n$ by taking all convex linear combinations of the given set of points. Clearly, the convex hull of a finite list of points, is always bounded.

Theorem 2.5 *[80] P is a polytope, if and only if, it is the convex hull of a finite set of points.*

DEFINITION: An *extreme point* of a convex set S is a point $x \in S$ satisfying

$$x = \alpha\bar{x} + (1-\alpha)\tilde{x}, \ \bar{x}, \tilde{x} \in S, \ \alpha \in (0,1) \ \rightarrow \ x = \bar{x} = \tilde{x}$$

Equivalently, an extreme point of a convex set S is one that cannot be expressed as a convex linear combination of some other points in S. When S is a polyhedron, extreme points of S correspond to the geometric notion of corner points. This correspondence is formalized in the corollary below.

Corollary 2.6 *A polytope P is the convex hull of its extreme points.*

Now we go on to discuss the representation of (possibly unbounded) convex polyhedra.

Theorem 2.7 *Any convex polyhedron P represented by a linear inequality system $\{y : yA \leq c\}$ can be also represented as the set addition of a convex cone R and a convex polytope Q.*

$$P = Q + R = \{x : x = \bar{x} + \bar{r}, \ \bar{x} \in Q, \ \bar{r} \in R\}$$

$$Q = \{\bar{x} : \bar{x} = \sum_{i=1}^{K} \alpha_i x^i, \ \sum_{i=1}^{K} \alpha_i = 1, \ \alpha \geq 0\}$$

$$R = \{\bar{r} : \bar{r} = \sum_{j=1}^{L} \mu_j r^j, \ \mu \geq 0\}$$

It follows from the statement of the theorem that P is non-empty if and only if the polytope Q is non-empty. We proceed now to discuss the representations of R and Q, respectively.

The cone R associated with the polyhedron P is called the *recession or characteristic cone* of P. A hyperplane representation of R is also readily available. It is easy to show that

$$R = \{r : Ar \leq 0\}$$

An obvious implication of the theorem and lemma above is that P equals the polytope Q if and only if $R = \{0\}$. In this form, the vectors $\{r^j\}$ are called the *extreme rays* of P.

The polytope Q associated with the polyhedron P is the convex hull of a finite collection $\{x^i\}$ of points in P. It is not difficult to see that the minimal set $\{x^i\}$ is precisely the set of extreme points of P. A non-empty pointed polyhedron P, it follows, must have atleast one extreme point.

The **affine hull** of P is given by

$$A.H.\{P\} = \{x : x = \sum \alpha_i x^i\}$$

$$x^i \in P \ \ \forall i, \ \text{ and } sum\alpha_i = 1$$

Clearly, the x^i can be restricted to the set of extreme points of P in the definition above. Furthermore, $A.H.\{P\}$ is the smallest affine set that contains P. A hyperplane representation of $A.H.\{P\}$ is also possible. First let us define the implicit linear equality system of P to be

$$\{A^= x = b^=\} = \{A_{i.} x = b_i \ \ \forall x \in P\}$$

Let the remaining inequalities of P be defined as

$$A^+ x \leq b^+$$

It follows that P must contain at least one point $\bar{x}$ satisfying

$$A^= \bar{x} = b^= \ and \ A^+ \bar{x} < b^+$$

Lemma 2.8 $A.H.\{P\} = \{x : A^= x = b^=\}$

The **dimension** of a polyhedron P in $\Re^n$ is defined to be the dimension of the affine hull of P, which equals the maximum number of affinely independent points, in $A.H.\{P\}$, minus one. P is said to be full-dimensional if its dimension equals m or, equivalently, if the affine hull of P is all of $\Re^n$.

A **supporting hyperplane** of the polyhedron P is a hyperplane H

$$H = \{x : b^T x = z^*\}$$

satisfying

$$b^T x \leq z^* \ \ \forall x \in P$$

$$b^T \hat{x} = z^* \ for \ some \ \hat{x} \in P$$

A supporting hyperplane H of P is one that touches P such that all of P is contained in a halfspace of H. Note that a supporting plane can touch P at more than one point.

A face of a non-empty polyhedron P is a subset of P that is either P itself or is the intersection of P with a supporting hyperplane of P. It follows that a face of P is itself a non-empty polyhedron.

A face of dimension, one less than the dimension of P, is called a **facet**. A face of dimension one is called an **edge** (note that extreme rays of P are also edges of P). A face of dimension zero is called a **vertex** of P (the vertices of P are precisely the extreme points of P). Two vertices of P are said to be adjacent if they are both contained in an edge of P. Two facets are said to be adjacent if they both contain a common face of dimension one less than that of a facet. Many interesting aspects of the facial structure of polyhedra can be derived from the following representation lemma.

Lemma 2.9 *F is a face of $P = \{x : Ax \leq b\}$ if and only if F is non-empty and $F = P \cap \{x : \tilde{A}x = \tilde{b}\}$, where $\tilde{A}x \leq \tilde{b}$ is a subsystem of $Ax \leq b$.*

As a consequence of the lemma, we have an algebraic characterization of extreme points of polyhedra.

Theorem 2.10 *Given a polyhedron P, defined by $\{x : Ax \leq b\}$, x^i is an extreme point of P if and only if it is a face of P satisfying $A^i x^i = b^i$ where $((A^i), (b^i))$ is a submatrix of (A, b) and the rank of A^i equals n.*

Now we come to Farkas Lemma which says that a linear inequality system has a solution if and only if a related (polynomial size) linear inequality system has no solution. This lemma is representative of a large body of theorems in mathematical programming known as *theorems of the alternative*

Lemma 2.11 (Farkas) *[26] Exactly one of the alternatives*

$$I.\ \exists\, x : Ax \leq b \qquad II.\ \exists\, y \geq 0 : A^T y = 0, b^T y < 0$$

is true for any given real matrices A, b.

2.3 Optimization and Dual Linear Programs

The two fundamental problems of linear programming (which are polynomially equivalent) are:

- **Solvability:** This is the problem of checking if a system of linear constraints on real (rational) variables is solvable or not. Geometrically, we have to check if a polyhedron, defined by such constraints, is nonempty.
- **Optimization:** This is the problem (LP) of optimizing a linear objective function over a polyhedron described by a system of linear constraints.

Building on polarity in cones and polyhedra, duality in linear programming is a fundamental concept which is related to both the complexity of linear programming and to the design of algorithms

for solvability and optimization. We will encounter the solvability version of duality (called Farkas' Lemma) while discussing the Fourier elimination technique below. Here we will state the main duality results for optimization. If we take the *primal* linear program to be

$$(P) \quad \min_{c^T x \in \Re^n} \{cx : Ax \geq b\}$$

there is an associated *dual* linear program

$$(D) \quad \max_{y \in \Re^m} \{b^T y : A^T y = c^T, y \geq 0\}$$

and the two problems satisfy

1. For any $\hat{x}$ and $\hat{y}$ feasible in (P) and (D) (i.e. they satisfy the respective constraints), we have $c\hat{x} \geq b^T \hat{y}$ (`weak duality`).

2. (P) has a finite optimal solution if and only if (D) does.

3. x^* and y^* are a pair of optimal solutions for (P) and (D) respectively, if and only if x^* and y^* are feasible in (P) and (D) (i.e. they satisfy the respective constraints) and $cx^* = b^T y^*$ (`strong duality`).

4. x^* and y^* are a pair of optimal solutions for (P) and (D) respectively, if and only if x^* and y^* are feasible in (P) and (D) (i.e. they satisfy the respective constraints) and $(Ax^* - b)^T y^* = 0$ (`complementary slackness`).

The strong duality condition above gives us a good stopping criterion for optimization algorithms. The complementary slackness condition, on the other hand gives us a constructive tool for moving from dual to primal solutions and vice-versa. The weak duality condition gives us a technique for obtaining lower bounds for minimization problems and upper bounds for maximization problems.

Note that the properties above have been stated for linear programs in a particular form. The reader should be able to check, that if for example the primal is of the form

$$(P') \quad \min_{x \in \Re^n} \{cx : Ax = b, x \geq 0\}$$

then the corresponding dual will have the form

$$(D') \quad \max_{y \in \Re^m} \{b^T y : A^T y \leq c^T\}$$

The tricks needed for seeing this is that any equation can be written as two inequalities, an unrestricted variable can be substituted by the difference of two non-negatively constrained variables and an inequality can be treated as an equality by adding a non-negatively constrained variable to the lesser side. Using these tricks, the reader could also check that dual construction in linear programming is involutory (i.e. the dual of the dual is the primal).

2.4 Complexity of Linear Equations and Inequalities

COMPLEXITY OF LINEAR ALGEBRA

Let us restate the fundamental problem of linear algebra as a decision problem.

$$\mathcal{CLS} = \{(A,b) : \exists\, x \in \mathcal{Q}^n, Ax = b\} \tag{1}$$

In order to solve the decision problem on $\mathcal{CLS}$ it is useful to recall *homogeneous* linear equations. A basic result in linear algebra is that any linear subspace of $\mathcal{Q}^n$ has two representations, one from hyperplanes and the other from a vector basis.

$\mathcal{L} = \{x \in \mathcal{Q}^n : Ax = 0\}$

$\mathcal{L} = \{x \in \mathcal{Q}^n : x = Cy,\ y \in \mathcal{Q}^k\}$

Corresponding to a linear subspace $\mathcal{L}$ there exists a dual (orthogonal complementary) subspace $\mathcal{L}^*$ with the roles of the hyperplanes and basis vectors of $\mathcal{L}$ exchanged.

$\mathcal{L}^* = \{z : Cz = 0\}$

$\mathcal{L}^* = \{z : z = Ax\}$

$dimension\mathcal{L} + dimension\mathcal{L}^* = n$

Using these representation results it is quite easy to establish the *Fundamental Theorem of Linear Algebra.*

Theorem 2.12 *Either Ax=b for some x* or *yA=0, yb≠0 for some y.*

Along with the basic theoretical constructs outlined above let us also assume knowledge of the *Gaussian Elimination Method* for solving a system of linear equations. It is easily verified that on a system of size m by n, this method uses $O(m^2n)$ elementary arithmetic operations. However we also need some bound on the size of numbers handled by this method. By the size of a rational number we mean the length of binary string encoding the number. And similarly for a matrix of numbers.

Lemma 2.13 *For any square matrix S of rational numbers, the size of the determinant of S is polynomially related to the size of S itself.*

Since all the numbers in a basic solution (ie. basis generated) of Ax=b are bounded in size by sub-determinants of the input matrix (A,b) we can conclude that $\mathcal{CLS}$ is a member of $\mathcal{NP}$. The Fundamental Theorem of Linear Algebra further establishes that $\mathcal{CLS}$ is in $\mathcal{NP} \bigcap co\mathcal{NP}$. And finally the polynomiality of Gaussian Elimination establishes that $\mathcal{CLS}$ is in $\mathcal{P}$.

COMPLEXITY OF LINEAR INEQUALITIES:

From our earlier discussion of polyhedra, we have the following algebraic characterization of extreme points of polyhedra.

Theorem 2.14 *Given a polyhedron P, defined by* $\{x : Ax \leq b\}$, x^i *is an extreme point of P if and only if it is a face of P satisfying* $A^i x^i = b^i$ *where* $((A^i), (b^i))$ *is a submatrix of* (A, b) *and the rank of* A^i *equals m.*

Corollary: The decision problem of verifying the membership of an input string (A,b) in the language $\mathcal{L}_I = \{(A, b) : \exists\, x$ *such that* $Ax \leq b\}$ belongs to $\mathcal{NP}$.

Proof: It follows from the theorem that every extreme point of the polyhedron $P = \{x : Ax \leq b\}$ is the solution of an (nxn) linear system whose coefficients come from (A, b). Therefore we can guess a polynomial length string representing an extreme point and check its membership in P in polynomial time. □

A consequence of Farkas Lemma is that the decision problem of testing membership of input (A,b) in the language

$$\mathcal{L}_I = \{(A, b) : \exists\, x \text{ such that } Ax \leq b\}$$

is in $\mathcal{NP} \bigcap co\mathcal{NP}$. That $\mathcal{L}_I$ can be recognized in polynomial time, follows from algorithms for linear programming that we now discuss.

We are now ready for a tour of some algorithms for linear programming. We start with the classical technique of Fourier which is interesting because of its simple syntactic specification. It leads to simple proofs of the duality principle of linear programming that was alluded to above. We will then review the Simplex method of linear programming [18], a method that uses the vertex-edge structure of a convex polyhedron to execute an optimization march. The simplex method has been finely honed over almost five decades now. We will spend some time with the Ellipsoid method and in particular with the polynomial equivalence of solvability (optimization) and separation problems. This aspect of the Ellipsoid method [34] has had a major impact on the identification of many tractable classes of combinatorial optimization problems. We conclude the tour of the basic methods with a description of Karmarkar's [46] breakthrough in 1984 which was an important landmark in the brief history of linear programming. A noteworthy role of interior point methods has been to make practical, the theoretical demonstations of tractability of various aspects of linear programming, including solvability and optimization, that were provided via the Ellipsoid method.

In later sections we will review the more sophisticated (and naturally esoteric) aspects of linear programming algorithms. This will include strongly polynomial algorithms for special cases, randomized algorithms and specialized methods for large-scale linear programming. Some readers may notice that we do not have a special section devoted to the discussion of parallel computation in the context of linear programming. This is partly because we are not aware of a well developed framework for such a discussion. We have instead introduced discussion and remarks about the effects of parallelism in the appropriate sections of this chapter.

3 Fourier's Projection Method

Linear programming is at the starting point for variable elimination techniques on algebraic constraints [11] which in turn forms the core of algebraic and symbolic computation. Constraint systems of linear *inequalities* of the form $Ax \leq b$, where A is an $m \times n$ matrix of real numbers are widely used in mathematical models. Testing the solvability of such a system is equivalent to linear programming. We now describe the elegant syntactic variable elimination technique due to Fourier [29].

Suppose we wish to eliminate the first variable x_1 from the system $Ax \leq b$. Let us denote

$$I^+ = \{i : A_{i1} > 0\} \quad I^- = \{i : A_{i1} < 0\} \quad I^0 = \{i : A_{i1} = 0\}$$

Our goal is to create an equivalent system of linear inequalities $\tilde{A}\tilde{x} \leq \tilde{b}$ defined on the variables $\tilde{x} = (x_2, x_3, \cdots, x_n)$.

- If I^+ is empty then we can simply delete all the inequalities with indices in I^- since they can be trivially satisfied by choosing a large enough value for x_1. Similarly, if I^- is empty we can discard all inequalities in I^+.
- For each $k \in I^+, l \in I^-$ we add $-A_{l1}$ times the inequality $A_k x \leq b_k$ to A_{k1} times $A_l x \leq b_l$. In these new inequalities the coefficient of x_1 is wiped out, i.e. x_1 is eliminated. Add these new inequalities to those already in I^0.
- The inequalities $\{\tilde{A}_{i1}\tilde{x} \leq \tilde{b}_i\}$ for all $i \in I^0$ represent the equivalent system on the variables $\tilde{x} = (x_2, x_3, \cdots, x_n)$.

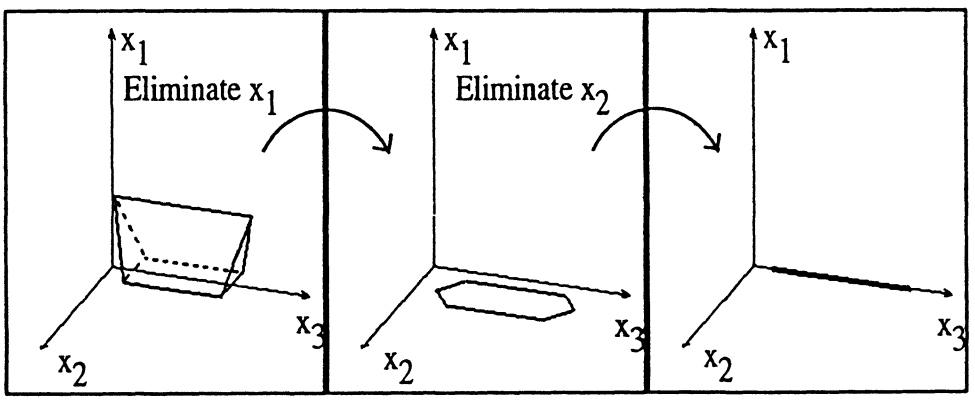

Figure 1 Variable Elimination and Projection

Repeat this construction with $\tilde{A}\tilde{x} \leq \tilde{b}$ to eliminate x_2 and so on until all variables are eliminated. If the resulting $\tilde{b}$ (after eliminating x_n) is non-negative we declare the original (and intermediate) inequality systems as being consistent. Otherwise[5] $\tilde{b} \not\geq 0$ and we declare the system inconsistent.

As an illustration of the power of elimination as a tool for theorem proving, we show now that Farkas Lemma is a simple consequence of the correctness of Fourier elimination. The lemma gives a direct proof that solvability of linear inequalities is in $\mathcal{NP} \bigcap co\mathcal{NP}$.

Farkas Lemma Exactly one of the alternatives

$$I.\ \exists\, x \in \Re^n : Ax \leq b \qquad II.\ \exists\, y \in \Re^m_+ : y^tA = 0, y^tb < 0$$

is true for any given real matrices A, b.

Proof: Let us analyze the case when Fourier Elimination provides a proof of the inconsistency of a given linear inequality system $Ax \leq b$. The method clearly converts the given system into $RAx \leq Rb$ where RA is zero and Rb has atleast one negative component. Therefore there is some row of R, say r, such that $rA = 0$ and $rb < 0$. Thus $\neg I$ implies II. It is easy to see that I and II cannot both be true for fixed A, b. □

In general, the Fourier elimination method is quite inefficient. Let k be any positive integer and n the number of variables be $2^k + k + 2$. If the input inequalities have lefthand sides of the form $\pm x_r \pm x_s \pm x_t$ for all possible $1 \leq r < s < t \leq n$ it is easy to prove by induction that after k variables are eliminated, by Fourier's method, we would have at least $2^{\frac{n}{2}}$ inequalities. The method is therefore exponential in the worst case and the explosion in the number of inequalities has been noted, in practice as well, on a wide variety of problems. We will discuss the central idea of minimal generators of the projection cone that results in a much improved elimination method [40].

First let us identify the set of variables to be eliminated. Let the input system be of the form

$$P = \{ (x, u) \in \Re^{n_1+n_2} \mid Ax + Bu \leq b \}$$

where u is the set to be eliminated. The projection of P onto x or equivalently the effect of eliminating the u variables is

$$P_x = \{ x \in \Re^{n_1} \mid \exists u \in \Re^{n_2} \text{ such that } Ax + Bu \leq b \}$$

Now W, the *projection cone* of P, is given by

$$W = \{ w \in \Re^m \mid wB = 0, w \geq 0 \}.$$

[5]Note that the final $\tilde{b}$ may not be defined if all the inequalities are deleted by the monotone sign condition of the first step of the construction described above. In such a situation we declare the system $Ax \leq b$ *strongly consistent* since it is consistent for any choice of b in $\Re^m$. In order to avoid making repeated references to this exceptional situation, let us simply assume that it does not occur. The reader is urged to verify that this assumption is indeed benign.

A simple application of Farkas Lemma yields a description of P_x in terms of W.

Projection Lemma Let G be any set of generators (eg. the set of extreme rays) of the cone W. Then $P_x = \{ x \in \Re^{n_1} \mid (gA)x \leq gb \quad \forall g \in G \}$.

The lemma, sometimes attributed to Černikov [9], reduces the computation of P_x to enumerating the extreme rays of the cone W or equivalently the extreme points of the polytope $W \cap \{ w \in \Re^m \mid \sum_{i=1}^m w_i = 1 \}$.

4 The Simplex Method

Consider a polyhedron $\mathcal{K} = \{x \in \Re^n : Ax = b, x \geq 0\}$. Now $\mathcal{K}$ cannot contain an infinite (in both directions) line since it is lying within the non-negative orthant of $\Re^n$. Such a polyhedron is called a *pointed* polyhedron. Given a pointed polyhedron $\mathcal{K}$ we observe that

- If $\mathcal{K} \neq \emptyset$ then $\mathcal{K}$ has at least one extreme point.
- If $\min\{cx : Ax = b, x \geq 0\}$ has an optimal solution then it has an optimal extreme point solution.

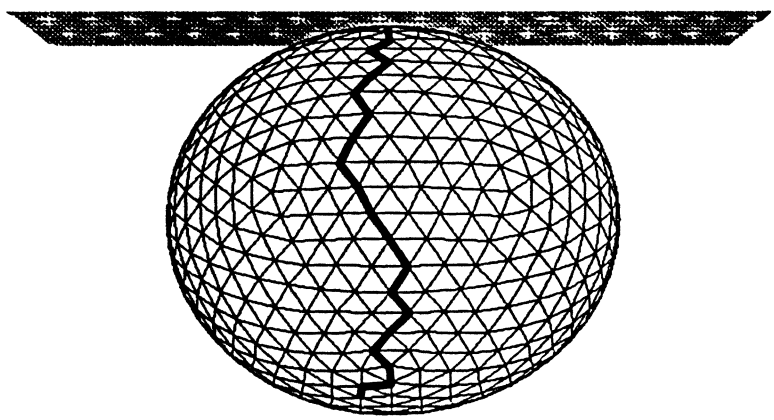

Figure 2 The Simplex Path

These observations together are sometimes called the fundamental theorem of linear programming since they suggest simple finite tests for both solvability and optimization. To generate all extreme points of $\mathcal{K}$, in order to find an optimal solution, is an impractical idea. However, we may try to run a partial search of the space of extreme points for an optimal solution. A simple local improvement

search strategy of moving from extreme point to adjacent extreme point until we get to a local optimum is nothing but the simplex method of linear programming [18,19]. The local optimum also turns out to be a global optimum because of the convexity of the polyhedron $\mathcal{K}$ and the objective function cx.

Procedure: **Primal Simplex**($\mathcal{K}$,c)

0. **Initialize:**

   - x_0 := an extreme point of $\mathcal{K}$
   - $k := 0$

1. **Iterative Step:**

   do

   If for all edge directions $\mathcal{D}_k$ at x_k, the objective function is non-decreasing, i.e.

   $$cd \geq 0 \quad \forall\, d \in \mathcal{D}_k$$

   then exit and return optimal x_k.

   Else pick some d_k in $\mathcal{D}_k$ such that $cd_k < 0$.

   If $d_k \geq 0$ then declare the linear program unbounded in objective value and exit.

   Else $x_{k+1} := x_k + \theta_k * d_k$, where

   $$\theta_k = \max\{\theta : x_k + \theta * d_k \geq 0\}$$

   $k := k+1$

   od

2. **End**

Remarks:

1. In the initialization step we assumed that an extreme point x_0 of the polyhedron $\mathcal{K}$ is available. This also assumes that the solvability of the constraints defining $\mathcal{K}$ has been established. These assumptions are reasonable since we can formulate the solvability problem as an optimization problem, with a self-evident extreme point, whose optimal solution either establishes unsolvability of $Ax = b$, $x \geq 0$, or provides an extreme point of $\mathcal{K}$. Such an optimization problem is usually called a Phase I model.The point being, of course, that the simplex method, as

described above, can be invoked on the Phase I model and if successful, can be invoked once again to carry out the intended minimization of cx. There are several different formulations of the Phase I model that have been advocated. Here is one.

$$\min\{v_0 : Ax + bv_0 = b,\ x \geq 0,\ v_0 \geq 0\}$$

The solution $(x, v_0)^T = (0, \cdots, 0, 1)$ is a self-evident extreme point and $v_0 = 0$ at an optimal solution of this model is a necessary and sufficient condition for the solvability of $Ax = b$, $x \geq 0$.

2. The scheme for generating improving edge directions uses an algebraic representation of the extreme points as certain bases, called feasible bases, of the vector space generated by the columns of the matrix A. It is possible to have linear programs for which an extreme point is geometrically over-determined (degenerate) i.e., there are more than d facets of $\mathcal{K}$ that contain the extreme point, where d is the dimension of $\mathcal{K}$. In such a situation, there would be several feasible bases corresponding to the same extreme point. When this happens, the linear program is said to be *primal degenerate.*

3. There are two sources of non-determinism in the primal simplex procedure. The first involves the choice of edge direction d_k made in step 1. At a typical iteration there may be many edge directions that are improving in the sense that $cd_k < 0$. Dantzig's Rule, Maximum Improvement Rule, and Steepest Descent Rule are some of the many rules that have been used to make the choice of edge direction in the simplex method. There is, unfortunately, no clearly dominant rule and successful codes exploit the empirical and analytic insights that have been gained over the years to resolve the edge selection nondeterminism in the simplex method.

   The second source of non-determinism arises from degeneracy. When there are multiple feasible bases corresponding to an extreme point, the simplex method has to pivot from basis to adjacent basis by picking an entering basic variable (a psuedo edge direction) and by dropping one of the old ones. A wrong choice of the leaving variables may lead to cycling in the sequence of feasible bases generated at this extreme point. Cycling is a serious problem when linear programs are highly degenerate as in the case of linear relaxations of many combinatorial optimization problems. The Lexicographic Rule (Perturbation Rule) for choice of leaving variables in the simplex method is a provably finite method (i.e., all cycles are broken).

   A clever method proposed by Bland (cf. [71]) preorders the rows and columns of the matrix A. In case of non-determinism in either entering or leaving variable choices, Bland's Rule just picks the lowest index candidate. All cycles are avoided by this rule also.

IMPLEMENTATION ISSUES: BASIS REPRESENTATIONS

The enormous success of the simplex method has been primarily due to its ability to solve large size problems that arise in practice. A distinguishing feature of many of the linear problems that are solved routinely in practice, is the sparsity of the constraint matrix. So from a computational point of view, it is desirable to take advantage of the sparseness of the constraint matrix. Another important consideration in the implementation of the simplex method is to control the accumulation of round off errors that arise because the arithmetic operations are performed with only a fixed number of digits and the simplex method is an iterative procedure.

An algebraic representation of the simplex method in matrix notation is as follows :

0: Find an initial feasible extreme point x^0, and the corresponding feasible basis B (of the vector space generated by the columns of the constraint matrix A). If no such x_0 exists, stop, there is no feasible solution. Otherwise, let $t = 0$, and go to step 1.

1: Partition the matrix A as $A = (B, N)$, the solution vector x as $x = (x_B, x_N)$ and the objective function vector c as $c = (c_B, c_N)$, corresponding to the columns in B.

2: The extreme point x^t is given by $x^t = (x_B^*, 0)$, where $Bx_B^* = b$

3: Solve the system $\pi_B\ B = c_B$ and calculate $r = c_N - \pi_B\ N$. If $r \geq 0$, stop, the current solution $x^t = (x_B^*, 0)$, is optimal. Otherwise, let $r_k = \min_j\{r_j\}$, where r_j is the j^{th} component of r (actually one may pick any $r_j < 0$ as r_k).

4: Let a_k denote the k^{th} column of N corresponding to r_k. Find y_k such that $B\ y_k = a_k$

5: Find $x_B(p)/y_{pk} = \min_i\{x_B(i)/y_{ik} : y_{ik} > 0\}$ where $x_B(i)$ and y_{ik} denote the i^{th} component of x_B and y_k respectively.

6: The new basis $\hat{B}$ is obtained from B by replacing the p^{th} column of B by the k^{th} column of N. Let the new feasible basis $\hat{B}$ be denoted as B. Return to step 1.

LU Factorization

At each iteration, the simplex method requires the solution of the following systems:
$Bx_B = b$; $\pi_B B = c_B$ and $By_k = a_k$.

After row interchanges, if necessary, any basis B can be factorized as $B = LU$ where L is a lower triangular matrix and U is an upper triangular matrix. So solving $LUx_B = b$ is equivalent to

Solving the triangular systems $Lv = b$ and $Ux_B = v$. Similarly, for $By_k = a_k$, we solve $Lw = a_k$ and $Uy_k = w$. Finally, for $\pi_B B = c_B$, we solve $\pi_B L = \lambda$ and $\lambda U = c_B$.

Let the current basis B and the updated basis $\hat{B}$ be represented as

$B = (a_1, a_2, ..., a_{p-1}, a_p, a_{p+1}, ..., a_m)$ and $\hat{B} = (a_1, a_2, ..., a_{p-1}, a_{p+1}, a_{p+2}, ..., a_m, a_k)$

An efficient implementation of the simplex method requires the updating of the triangular matrices L and U as triangular matrices $\hat{L}$ and $\hat{U}$ where $B = LU$ and $\hat{B} = \hat{L}\hat{U}$. This is done by first obtaining $H = (u_1, u_2, ..., u_{p-1}, u_{p+1}, ..., u_m, w)$ where u_i is the i^{th} column of U and $w = L^{-1}a_k$. The matrix H has zeros below the main diagonal in the first $p-1$ columns and zeros below the element immediately under the diagonal in the remaining columns. The matrix H can be reduced to an upper triangular matrix by Gaussian elimination which is equivalent to multiplying H on the left by matrices M_i, $i = p, p+1, ..., m-1$, where M_j differs from an identity matrix in column j which is given by $(0, ..., 0, 1, m_j, 0...0)^T$, where m_j is in position $j+1$. Now $\hat{U}$ is given by $\hat{U} = M_{m-1}, M_{m-2}, ...M_p H$ and $\hat{L}$ is given by $\hat{L} = LM_p^{-1}, ..., M_{m-1}^{-1}$. Note that M_j^{-1} is M_j with the sign of the off-diagonal term m_j reversed.

The LU factorization preserves the sparsity of the basis B, in that the number of non-zero entries in L and U is typically not much larger than the number of non-zero entries in B. Furthermore, this approach effectively controls the accumulation of round off errors and maintains good numerical accuracy. In practice, the LU factorization is periodically recomputed for the matrix $\hat{B}$ instead of updating the factorization available at the previous iteration. This computation of $\hat{B} = \hat{L}\hat{U}$ is achieved by Gaussian elimination to reduce $\hat{B}$ to an upper triangular matrix (for details, see for instance [36,61,62]). There are several variations of the basic idea of factorization of the basis matrix B, as described here, to preserve sparsity and control round off errors.

REMARK: The simplex method is not easily amenable to parallelization. However, some steps such as identification of the entering variable and periodic refactorization can be efficiently parallelized.

GEOMETRY AND COMPLEXITY OF THE SIMPLEX METHOD

An elegant geometric interpretation of the simplex method can be obtained by using a *column space representation* [19], i.e. $\Re^{m+1}$ coordinatized by the rows of the $(m+1) \times n$ matrix $\begin{pmatrix} c \\ A \end{pmatrix}$. In fact it is this interpretation that explains why it is called the simplex method. The bases of A correspond to an arrangement of simplicial cones in this geometry and the pivoting operation corresponds to a physical pivot from one cone to an adjacent one in the arrangement. An interesting insight that can be gained from the column space perspective is that Karmarkar's interior point method can be seen as a natural generalization of the simplex method [77,13].

However, the geometry of linear programming, and of the simplex method, has been largely developed in the space of the x variables, i.e. in $\Re^n$. The simplex method walks along edge paths on the combinatorial graph structure defined by the boundary of convex polyhedra. These graphs are quite dense (Balinski's theorem [83] states that the graph of d-dimensional polyhedron must be d-connected). A polyhedral graph can also have a huge number of vertices since the Upper Bound Theorem of McMullen, see [83], states that the number of vertices can be as large as $O(k^{\lfloor d/2 \rfloor})$ for a polytope in d dimensions defined by k constraints. Even a polynomial bound on the diameter of polyhedral graphs is not known. The best bound obtained to date is $O(k^{1+\log d})$ of a polytope in d dimensions defined by k constraints. Hence it is no surprise that there is no known variant of the simplex method with a worst-case polynomial guarantee on the number of iterations.

Klee and Minty [48] exploited the sensitivity of the original simplex method of Dantzig, to projective scaling of the data, and constructed exponential examples for it. These example were simple projective distortions of the hypercube to embed long isotonic (improving objective value) paths in the graph. Scaling is used in the Klee-Minty construction, to trick the choice of entering variable (based on most negative reduced cost) in the simplex method and thus keep it on an exponential path. Later, several variants of the entering variable choice (best improvement, steepest descent, etc.) were all shown to be susceptible to similar constructions of exponential examples (cf. [71]).

Despite its worst-case behaviour, the simplex method has been the veritable workhorse of linear programming for five decades now. This is because both empirical [19,6] and probabilistic [8,38] analyses indicate that the number of iterations of the simplex method is just slightly more than linear in the dimension of the primal polyhedron.

The ellipsoid method of Shor [75] was devised to overcome poor scaling in convex programming problems and therefore turned out to be the natural choice of an algorithm to first establish polynomial-time solvability of linear programming. Later Karmarkar [46] took care of both projection and scaling simultaneously and arrived at a superior algorithm.

5 The Ellipsoid Method

The Ellipsoid Algorithm of Shor [75] gained prominence in the late 1970's when Haĉijan (pronounced Khachiyan) [37] showed that this convex programming method specializes to a polynomial-time algorithm for linear programming problems. This theoretical breakthrough naturally led to intense study of this method and its properties. The survey paper by Bland et al. [7] and the monograph by Akgül [2] attest to this fact. The direct theoretical consequences for combinatorial optimization problems was independently documented by Padberg and Rao [66], Karp and Papadimitriou [47] and

Grötschel, Lovász and Schrijver [33]. The ability of this method to implicitly handle linear programs with an exponential list of constraints and maintain polynomial-time convergence is a characteristic that is the key to its applications in combinatorial optimization. For an elegant treatment of the many deep theoretical consequences of the Ellipsoid Algorithm, the reader is directed to the monograph by Lovász [50] and the book by Grötschel, Lovász and Schrijver [34].

Computational experience with the Ellipsoid Algorithm, however, showed a disappointing gap between the theoretical promise and practical efficiency of this method in the solution of linear programming problems. Dense matrix computations as well as the slow average-case convergence properties are the reasons most often cited for this behaviour of the Ellipsoid Algorithm. On the positive side though, it has been noted (cf. Ecker and Kupferschmid [23]) that the Ellipsoid method is competitive with the best known algorithms for (non-linear) convex programming problems.

Let us consider the problem of testing if a polyhedron $\mathcal{Q} \in \Re^d$, defined by linear inequalities, is non-empty. For technical reasons let us assume that $\mathcal{Q}$ is rational, i.e. all extreme points and rays of $\mathcal{Q}$ are rational vectors or equivalently that all inequalities in some description of $\mathcal{Q}$ involve only rational coefficients. The Ellipsoid method does not require the linear inequalities describing $\mathcal{Q}$ to be explicitly specified. It suffices to have an oracle representation of $\mathcal{Q}$. Several different types of oracles can be used in conjunction with the ellipsoid method [34,47,66]. We will use the *strong separation oracle* described below.

Oracle: **Strong Separation**($\mathcal{Q}$,y)
Given a vector $y \in \Re^d$, decide whether $y \in \mathcal{Q}$, and if not find a hyperplane that separates y from $\mathcal{Q}$; more precisely, find a vector $c \in \Re^d$ such that $c^T y < \min\{c^T x : x \in \mathcal{Q}\}$.

The ellipsoid algorithm initially chooses an ellipsoid large enough to contain a part of the polyhedron $\mathcal{Q}$ if it is non-empty. This is easily accomplished because we know that if $\mathcal{Q}$ is non-empty then it has a rational solution whose (binary encoding) length is bounded by a polynomial function of the length of the largest coefficient in the linear program and the dimension of the space.

The centre of the ellipsoid is a feasible point if the separation oracle tells us so. In this case, the algorithm terminates with the co-ordinates of the centre as a solution. Otherwise, the separation oracle outputs an inequality that separates the centre point of the ellipsoid from the polyhedron $\mathcal{Q}$. We translate the hyperplane defined by this inequality to the centre point. The hyperplane slices the ellipsoid into two halves, one of which can be discarded. The algorithm now creates a new ellipsoid that is the minimum volume ellipsoid containing the remaining half of the old one. The algorithm questions if the new centre is feasible and so on. The key is that the new ellipsoid has substantially smaller volume than the previous one. When the volume of the current ellipsoid shrinks

to a sufficiently small value, we are able to conclude that Q is empty. This fact is used to show the polynomial time convergence of the algorithm.

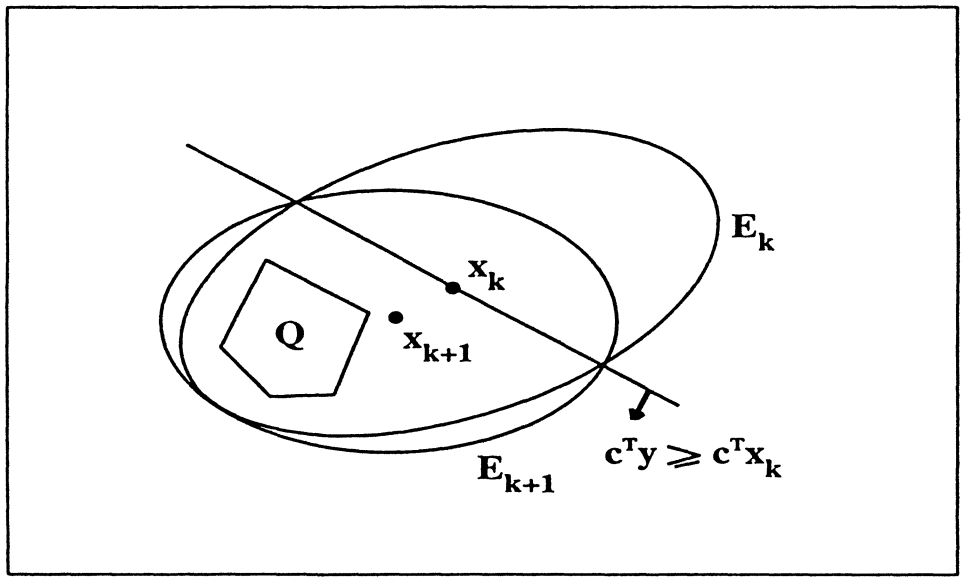

Figure 3 Shrinking Ellipsoids

Ellipsoids in $\Re^d$ are denoted as $E(A, y)$ where A is an $d \times d$ positive definite matrix and $y \in \Re^d$ is the centre of the ellipsoid $E(A, y)$.

$$E(A, y) = \{x \in \Re^d \mid (x - y)^T A^{-1}(x - y) \leq 1\}$$

The ellipsoid algorithm is described on the iterated values, A_k and x^k which specify the underlying ellipsoids $E_k(A_k, x^k)$.

Procedure: **Ellipsoid**($\mathcal{Q}$)

0. **Initialize:**

   - $N := N(\mathcal{Q})$ (comment: iteration bound)
   - $R := R(\mathcal{Q})$ (comment: radius of the initial ellipsoid/sphere E_0)
   - $A_0 := R^2 I$
   - $x_0 := 0$ (comment: centre of E_0)
   - $k := 0$

1. **Iterative Step:**

   while $k < N$

   call Strong Separation $(\mathcal{Q}, x^k)$

   if $x^k \in \mathcal{Q}$ **halt**

   else hyperplane $\{x \in \Re^d \mid c^T x = c_0\}$ separates x^k from $\mathcal{Q}$

   Update

$$b := \frac{1}{\sqrt{c^T A_k c}} A_k c$$

$$x^{k+1} := x^k - \frac{1}{d+1} b \text{ v}$$

$$A_{k+1} := \frac{d^2}{d^2-1}(A_k - \frac{2}{d+1} b b^T)$$

$$k := k + 1$$

   endwhile

2. **Empty Polyhedron:**

   - **halt and declare** ``$\mathcal{Q}$ **is empty**''

3. **End**

The crux of the complexity analysis of the algorithm is on the apriori determination of the iteration bound. This in turn depends on three factors. The volume of the initial ellipsoid E_0, the rate of volume shrinkage ($\frac{vol(E_{k+1})}{vol(E_k)} < e^{-\frac{1}{(2d)}}$) and the volume threshold at which we can safely conclude that $\mathcal{Q}$ must be empty. The assumption of $\mathcal{Q}$ being a rational polyhedron is used to argue that $\mathcal{Q}$ can be modified into a full-dimensional polytope without affecting the decision question ("Is $\mathcal{Q}$ non-empty ?"). After careful accounting for all these technical details and some others (eg. compensating for the round-off errors caused by the square root computation in the algorithm) it is

possible to establish the following fundamental result.

Theorem 5.1 *There exists a polynomial $g(d,\phi)$ such that the* `ellipsoid method` *runs in time bounded by $T\,g(d,\phi)$ where ϕ is an upper bound on the size of linear inequalities in some description of $\mathcal{Q}$ and T is the maximum time required by the oracle* `Strong Separation`$(\mathcal{Q},y)$ *on inputs y of size at most $g(d,\phi)$.*

The size of a linear inequality is just the length of the encoding of all the coefficients needed to describe the inequality. A direct implication of the theorem is that solvability of linear inequalities can be checked in polynomial time if strong separation can be solved in polynomial time. This implies that the standard linear programming solvability question has a polynomial-time algorithm (since separation can be effected by simply checking all the constraints). Happily, this approach provides polynomial-time algorithms for much more than just the standard case of linear programming solvability. The theorem can be extended to show that the optimization of a linear objective function over $\mathcal{Q}$ also reduces to a polynomial number of calls to the strong separation oracle on $\mathcal{Q}$. A converse to this theorem also holds, namely separation can be solved by a polynomial number of calls to a solvability/optimization oracle [34]. Thus, optimization and separation are polynomially equivalent. This provides a very powerful technique for identifying tractable classes of optimization problems. Semi-definite programming and submodular function minimization are two important classes of optimization problems that can be solved in polynomial time using this property of the Ellipsoid method.

SEMI-DEFINITE PROGRAMMING

The following optimization problem defined on symmetric $(n \times n)$ real matrices

$$(SDP) \quad \min_{X \in \Re^{n \times n}} \{\sum_{ij} C \bullet X : A \bullet X = B,\ X \succeq 0\}$$

is called a semi-definite program. Note that $X \succeq 0$ denotes the requirement that X is a positive semi-definite matrix, and $F \bullet G$ for $n \times n$ matrices F and G denotes the product matrix $(F_{ij} * G_{ij})$. From the definition of positive semi-definite matrices, $X \succeq 0$ is equivalent to

$$q^T X q \ \geq\ 0 \quad \text{for every} \quad q \in \Re^n$$

Thus (SDP) is really a linear program on $O(n^2)$ variables with an (uncountably) infinite number of linear inequality constraints. Fortunately, the strong separation oracle is easily realized for these constraints. For a given symmetric X we use Cholesky factorization to identify the minimum eigenvalue λ_{min}. If λ_{min} is non-negative then $X \succeq 0$ and if, on the other hand, λ_{min} is negative we have a separating inequality

$$\gamma_{min}^T X \gamma_{min} \ \geq\ 0$$

where γ_{min} is the eigenvector corresponding to λ_{min}. Since the Cholesky factorization can be computed by an $O(n^3)$ algorithm, we have a polynomial-time separation oracle and an efficient algorithm for (SDP) via the Ellipsoid method. Alizadeh [3] has shown that interior point methods can also be adapted to solving (SDP) to within an additive error ϵ in time polynomial in the size of the input and $\log \frac{1}{\epsilon}$.

This result has been used to construct efficient approximation algorithms for Maximum Stable Sets and Cuts of Graphs [32], Shannon Capacity of Graphs, Minimum Colorings of Graphs. It has been used to define hierarchies of relaxations for integer linear programs that strictly improve on known exponential-size linear programming relaxations [51].

Minimizing Submodular Set Functions

The minimization of submodular set functions is a generic optimization problem which contains a large class of important optimization problems as special cases [25]. Here we will see why the ellipsoid algorithm provides a polynomial-time solution method for submodular minimization.

Definition 5.2 *Let N be a finite set. A real valued set function f defined on the subsets of N is*

- *submodular if $f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$ for $X, Y \subseteq N$.*

Example 5.3 *Let $G = (V, E)$ be an undirected graph with V as the node set and E as the edge set. Let $c_{ij} \geq 0$ be the weight or capacity associated with edge $(ij) \in E$. For $S \subseteq V$, define the cut function $c(S) = \sum_{i \in S, j \in V \setminus S} c_{ij}$. The cut function defined on the subsets of V is submodular since $c(X) + c(Y) - c(X \cup Y) - c(X \cap Y) = \sum_{i \in X \setminus Y, j \in Y \setminus X} 2c_{ij} \geq 0$.*

The optimization problem of interest is

$$\min\{f(X) : X \subseteq N\}$$

The following remarkable construction that connects submodular function minimization with convex function minimization is due to Lovász (cf. [34]).

Definition 5.4 The Lovász extension $\hat{f}(.)$ *of a submodular function $f(.)$ satisfies*

- $\hat{f} : [0,1]^N \rightarrow \Re$.
- $\hat{f}(x) = \sum_{I \in \mathcal{I}} \lambda_I f(x_I)$ *where $x = \sum_{I \in \mathcal{I}} \lambda_I x_I$, $x \in [0,1]^N$, x_I is the incidence vector of I for each $I \in \mathcal{I}$, $\lambda_I > 0$ for each I in $\mathcal{I}$, and $\mathcal{I} = \{I_1, I_2, \cdots, I_k\}$ with $\emptyset \neq I_1 \subset I_2 \subset \cdots \subset I_k \subseteq N\}$. Note that the representation $x = \sum_{I \in \mathcal{I}} \lambda_I x_I$ is unique given that the $\lambda_I > 0$ and that the sets in $\mathcal{I}$ are nested.*

It is easy to check that $\hat{f}(.)$ is a convex function. Lovász also showed that the minimization of the submodular function $f(.)$ is a special case of convex programming by proving

$$\min\{f(X) : X \subseteq N\} = \min\{\hat{f}(x) : x \in [0,1]^N\}$$

Further, if x^* is an optimal solution to the convex program and

$$x^* = \sum_{I \in \mathcal{I}} \lambda_I x_I$$

then for each $\lambda_I > 0$, it can be shown that $I \in \mathcal{I}$ minimizes f. The Ellipsoid method can be used to solve this convex program (and hence submodular minimization) using a polynomial number of calls to an oracle for f (this oracle returns the value of $f(X)$ when input X).

6 Interior Point Methods

The announcement of the polynomial solvability of linear programming followed by the probabilistic analyses of the simplex method in the early 1980's left researchers in linear programming with a dilemma. We had one method that was good in a theoretical sense but poor in practice and another that was good in practice (and on average) but poor in a theoretical worst-case sense. This left the door wide open for a method that was good in both senses. Narendra Karmarkar closed this gap with a breathtaking new projective scaling algorithm. In retrospect, the new algorithm has been identified with a class of nonlinear programming methods known as logarithmic barrier methods. Implementations of a primal-dual variant of the logarithmic barrier method have proven to be the best approach at present. The recent monogragh by S.J. Wright [81] is dedicated to primal-dual interior point methods. It is a variant of this method that we descibe below.

It is well known that moving through the interior of the feasible region of a linear program using the negative of the gradient of the objective function, as the movement direction, runs into trouble because of getting "jammed" into corners (in high dimensions, corners make up most of the interior of a polyhedron). This jamming can be overcome if the negative gradient is balanced with a "centering" direction. The centering direction in Karmarkar's algorithm is based on the *analytic center* y_c of a full dimensional polyhedron $\mathcal{D} = \{x : A^T y \leq c\}$ which is the unique optimal solution to

$$\max\{\sum_{j=1}^{n} \ln(z_j) : A^T y + z = c\}$$

Recall the primal and dual forms of a linear program may be taken as

$$(P) \quad \min\{cx : Ax = b, x \geq 0\}$$

$$(D) \quad \max\{b^T y : A^T y \leq c\}$$

The logarithmic barrier formulation of the dual (D) is

$$(D_\mu) \quad \max\{b^T y + \mu \sum_{j=1}^{n} \ln(z_j) : \; A^T y + z = c\}$$

Notice that as $(D_\mu$ is equivalent to (D) as $\mu \to 0^+$. The optimality (Karush-Kuhn-Tucker) conditions for (D_μ) are given by

$$D_x D_z e = \mu e$$

$$Ax = b$$

$$A^T y + z = c$$

where D_x and D_z denote $n \times n$ diagonal matrices whose diagonals are x and z respectively. Notice that if we set μ to 0, the above conditions are precisely the primal-dual optimality conditions; complementary slackness, primal and dual feasibility of a pair of optimal (P) and (D) solutions. The problem has been reduced to solving the above equations in x, y, z. The classical technique for solving equations is Newton's method which prescribes the directions

$$\begin{aligned} \Delta y &= -(AD_x D_z^{-1} A^T)^{-1} A D_z^{-1}(\mu e - D_x D_z e) \\ \Delta z &= -A^T \Delta y \\ \Delta x &= D_z^{-1}(\mu e - D_x D_z e) - D_x D_z^{-1} \Delta z \end{aligned} \tag{2}$$

The strategy is to take one Newton step, reduce μ and iterate until the optimization is complete. The criterion for stopping can be determined by checking for feasibility $(x, z \geq 0)$ and if the duality gap $(x^t z)$ is close enough to 0. We are now ready to describe the algorithm.

Procedure: **Primal-Dual Interior**

0. **Initialize:**

   - $x_0 > 0,\ y_0 \in \Re^m,\ z_0 > 0, \mu_0 > 0, \epsilon > 0, \rho > 0$
   - $k := 0$

1. **Iterative Step:**

   do

   Stop if $Ax_k = b,\ A^T y_k + z_k = c$ and $x_k^T z_k \leq \epsilon$.

   $x_{k+1} \leftarrow x_k + \alpha_k^P \Delta x_k$

   $y_{k+1} \leftarrow y_k + \alpha_k^D \Delta y_k$

   $z_{k+1} \leftarrow z_k + \alpha_k^D \Delta z_k$

   /* Δx_k, Δy_k, Δz_k are the Newton directions from (2) */

   $\mu_{k+1} \leftarrow \rho \mu_k$

   $k := k + 1$

   od

2. **End**

Remarks:

1. The primal-dual algorithm has been used in several large-scale implementations. For appropriate choice of parameters, it can be shown that the number of iterations in the worst-case is $O(\sqrt{n} \log(\epsilon_0/\epsilon))$ to reduce the duality gap from ϵ_0 to ϵ [69,81]. While this is sufficient to show that the algorithm is polynomial time, it has been observed that the "average" number of iterations is more like $O(\log n \log(\epsilon_0/\epsilon))$. However, unlike the simplex method we do not have a satisfactory theoretical analysis to explain this observed behaviour.

2. The stepsizes α_k^P and α_k^D are chosen to keep x_{k+1} and z_{k+1} strictly positive. The ability in the primal-dual scheme to choose separate stepsizes for the primal and dual variables is a major computational advantage that this method has over the pure primal or dual methods. Empirically this advantage translates to a significant reduction in the number of iterations.

3. The stopping condition essentially checks for primal and dual feasibility and near complementary slackness. Exact complementary slackness is not possible with interior solutions. It is possible to maintain primal and dual feasibility through the algorithm, but this would require

a Phase I construction via artificial variables. Empirically, this feasible variant has not been found to be worthwhile. In any case, when the algorithm terminates with an interior solution, a post-processing step is usually invoked to obtain optimal extreme point solutions for the primal and dual. This is usually called the *purification* of solutions and is based on a clever scheme described by Megiddo [56].

4. Instead of using Newton steps to drive the solutions to satisfy the optimality conditions of (D_μ), Mehrotra [59] suggested a predictor-corrector approach based on power series approximations. This approach has the added advantage of providing a rational scheme for reducing the value of μ. It is the predictor-corrector based primal-dual interior method that is considered the current winner in interior point methods. The OB1 code of Lustig, Marsten and Shanno [52] is based on this scheme. CPLEX 4.0 [17], a general purpose linear (and integer) programming solver, also contains implementations of interior point methods.

   Saltzman [70] describes a parallelization of the OB1 method to run on shared-memory vector multiprocessor architectures. Recent computational studies of parallel implementations of simplex and interior point methods on the SGI Power Challenge (SGI R8000) platform indicate that on all but a few small linear programs in the NETLIB linear programming benchmark problem set, interior point methods dominate the simplex method in run times. New advances in handling Cholesky factorizations in parallel are apparently the reason for this exceptional performance of interior point methods.

   As in the case of the simplex method, there are a number of special structures in the matrix A which can be exploited by interior point methods to obtain improved efficiencies. Network flow constraints, generalized upper bounds (GUB) and variable upper bounds (VUB) are structures that often come up in practice and which can be useful in this context [14,79].

5. Interior point methods, like ellipsoid methods, do not directly exploit the linearity in the problem description. Hence they generalize quite naturally to algorithms for semidefinite and convex programming problems. More details of these generalizations are given in chapter (NOTE TO EDITOR: CROSS-REFEERENCE CHAPTER BY VAVASIS ON CONVEX PROGRAMMING HERE) of this handbook. Karmarkar [45] has proposed an interior-point approach for integer programming problems. The main idea is to reformulate an integer program as the minimization of a quadratic energy function over linear constraints on continuous variables. Interior-point methods are applied to this formulation to find local optima.

7 Strongly Polynomial Methods

The number of iterations and hence the number of elementary arithmetic operations required for the Ellipsoid Method as well as the Interior Point Method is bounded by a polynomial function of the number of bits required for the binary representation of the input data. Recall that the size of a rational number a/b is defined as the total number of bits required in the binary representation of a and b. The dimension of the input is the number of data items in the input. An algorithm is said to be *strongly polynomial* if it consists of only elementary arithmetic operations (performed on rationals of size bounded by a polynomial in the size of the input) and the number of such elementary arithmetic operations is bounded by a polynomial in the dimension of the input.

It is an open question as to whether there exists a strongly polynomial algorithm for the general linear programming problem. However, there are some interesting partial results:

- Tardos [78] has devised an algorithm for which the number of elementary arithmetic operations is bounded by a polynomial function of n, m and the number of bits required for the binary representation of the elements of the constraint matrix A which is $m \times n$. The number of elementary operations does not depend upon the right hand side or the cost coefficients.

- Megiddo [57] described a strongly polynomial algorithm for checking the solvability of linear constraints with at most two non-zero coefficients per inequality. Later, Hochbaum and Naor [39] showed that Fourier Elimination can be specialized to provide a strongly polynomial algorithm for this class.

- Megiddo [58] and Dyer [21] have independently designed strongly polynomial (linear-time) algorithms for linear programming in fixed dimensions. The number of operations for these algorithms is linear in the number of constraints and independent of the coefficients but doubly exponential in the number of variables.

The rest of this section details these three results and some of their consequences.

7.1 Combinatorial Linear Programming

Consider the linear program, (LP) $Max\{cx : Ax = b,\ x \geq 0\}$, where A is a $m \times n$ integer matrix. The associated dual linear program is $Min\ \{yb\ :\ y\ A \geq c\}$. Let L be the maximum absolute value in the matrix and let $\Delta = (nL)^n$. We now describe Tardos' algorithm for solving (LP) which permits the number of elementary operations to be free of the magnitudes of the "rim" coefficients b and c.

The algorithm uses Procedure 1 to solve a system of linear inequalities. Procedure 1, in turn, calls Procedure 2 with any polynomial-time linear programming algorithm as the required subroutine.

Procedure 2 finds the optimal objective function value of a linear program and a set of variables which are zero in some optimal solution, if the optimum is finite. Note that Procedure 2 only finds the optimal objective value and not an optimal solution. The main algorithm also calls Procedure 2 directly with Subroutine 1 as the required subroutine. For a given linear program, Subroutine 1 finds the optimal objective function value and a dual solution, if one exists. Subroutine 1, in turn, calls Procedure 2 along with any polynomial-time linear programming algorithm as the required subroutine. We omit the detailed descriptions of the Procedures 1 & 2 and Subroutine 1 and instead only give their input/output specifications.

Algorithm: Tardos

INPUT: A linear programming problem $\max\{cx : Ax = b,\ x \geq 0\}$

OUTPUT: An optimal solution, if it exists and the optimal objective function value.

1. Call
   Procedure 1 to test whether $\{Ax = b,\ x \geq 0\}$ is feasible. If the system is not feasible, the optimal objective function value $= -\infty$, stop.
2. Call Procedure 1, to test whether $\{yA \geq c\}$ is feasible. If the system is not feasible, the optimal objective function value $= +\infty$, stop.
3. Call Procedure 2 with the inputs as the linear program $Max\{cx : Ax = b,\ x \geq 0\}$ and Subroutine 1 as the required subroutine. Let $x_i = 0,\ i \in K$ be the set of equalities identified.
4. Call Procedure 1 to find a feasible solution x^* to $\{Ax = b,\ x \geq 0,\ x_i = 0,\ i \in K\}$. The solution x^* is optimal and the optimal objective function value is cx^*
5. End

Specification of Procedure 1:

INPUT: A linear system $Ax \leq b$, where A is a $m \times n$ matrix .

OUTPUT: Either $Ax \leq b$ is infeasible or $\hat{x}$ is a feasible solution.

Specification of Procedure 2:

INPUT: Linear program $Max\{cx \ : \ Ax = b, \ x \geq 0\}$, which has a feasible solution and
a subroutine which for a given integer vector $\bar{c}$ with $\| \bar{c}\|_\infty \ \leq n^2 \ \Delta$ and a set K of indices, determines $\max\{\bar{c}x \ : \ Ax = b, \ x \geq 0, \ x_i = 0, \ i \in \ K\}$ and if the maximum is finite, finds an optimal dual solution.

OUTPUT: The maximum objective function value z^* of the input linear program $\max\{c \ x \ : \ A \ x = b, \ x \geq 0,\}$ and the set K of indices such $x_i = 0, \ i \ \in \ K$ for some optimum solution to the input linear program.

Specification of Subroutine 1:

INPUT: A Linear program $\max\{\bar{c}x \ : \ Ax = b, \ x \geq 0, \ x_i = 0, \ i \in K\}$, which is feasible and $\| \bar{c}\|_\infty \ \leq n^2 \ \Delta$.

OUTPUT: The Optimal objective function value z^* and an optimal dual solution y^*, if it exists.

The validity of the algorithm and the analysis of the number of elementary arithmetic operations required are in the paper by Tardos [78]. This result may be viewed as an application of techniques from *diophantine approximation* to linear programming. A scholarly account of these connections is given in the book by Schrijver [71].

REMARK: Linear programs with $\{0, \ \pm \ 1\}$ elements in the constraint matrix A arise in many applications of polyhedral methods in combinatorial optimization. Network flow problems (shortest path, maximum flow and transshipment) [1] are examples of problems in this class. Such linear programs, and more generally linear programs with the matrix A made up of integer coefficients of bounded magnitude, are known as *combinatorial linear programs.* The algorithm described shows that combinatorial linear programs can be solved by strongly polynomial methods.

7.2 Fourier Elimination and $LI(2)$:

We now describe a special case of the linear programming solvability problem for which Fourier elimination yields a very efficient (strongly polynomial) algorithm. This is the case $LI(2)$ of linear inequalities with at most two variables per inequality. Nelson [63] observed that Fourier elimination is subexponential in this case. He showed that the number of inequalities generated never exceeds

$O(mn^{\lceil \log n \rceil} \log n)$. Later Aspvall & Shiloach [4] obtained the first polynomial-time algorithm for solving $LI(2)$ using a graph representation of the inequalities. We give a high-level description of the technique of Hochbaum & Naor [39] that combines Fourier elimination and a graph reasoning technique to obtain the best known sequential complexity bounds for $LI(2)$.

An interesting property of $LI(2)$ systems is that they are closed under Fourier Elimination. Therefore the projection of an $LI(2)$ system on to a subspace whose coordinates are a subset of the variables is also an $LI(2)$ system. Note that $LI(3)$ does not have this closure property. Indeed $LI(3)$ is unlikely to have any special property since any system of linear inequalities can be reduced to an instance of $LI(3)$ with $0, \pm 1$ coefficients [42].

Given an instance of $LI(2)$ of the form $Ax \leq b$ with each row of A containing at most two nonzero coefficients we construct a graph $G(V,E)$ as follows. The vertices V are $x_0, x_1, \cdots, x_n$ corresponding to the variables of the constraints (x_0 is an extra dummy variable). The edges E of G are composed of pairs (x_i, x_j) if x_i and x_j are two variables with nonzero coefficients of at least one inequality in the system. There are also edges of the form (x_0, x_k) if x_k is the only variable with a nonzero coefficient in some constraint. Let us also assume that each edge is labelled with all the inequalities associated with its existence.

Aspvall & Shiloach [4] describe a "grapevine algorithm" that takes as input a "rumour" $x_j = \alpha$ and checks its authenticity i.e. checks if α is too small, too large or within the range of feasible values of x_j. The idea is simply to start at node x_j and set $x_j = \alpha$. Obviously, each variable x_k that is a neighbour of x_j in G gets an implied lower bound or upper bound (or both) depending on the sign of the coefficient of x_k in the inequality shared with x_j. These bounds get propogated further to neighbours of the x_k and so on. If this propogation is carried out in a breadth-first fashion, it is not hard to argue that the implications of setting $x_j = \alpha$ are completely revealed in $3n - 2$ stages. Proofs of inconsistency can be traced back to delineate if α was either too high or too low a value for x_j.

The grapevine algorithm is similar to Bellman & Ford's classical shortest path algorithm for graphs which also takes $O(mn)$ effort. This subroutine provides the classification test for being able to execute binary search in choosing values for variables. The specialization of Fourier's algorithm for $LI(2)$ can be described now.

Algorithm Fourier $LI(2)$:

For $j = 1, 2, \cdots n$

1. The inequalities of each edge (x_j, x_k) define a convex polygon Q_{jk} in x_j, x_k-space. Compute J_k the sorted collection of x_j coordinates of the corner (extreme) points of Q_{jk}. Let J denote the sorted union (merge) of the J_k (x_k a neighbour of x_j in G).

2. Perform a binary search on the sequence J for a feasible value of x_j. If we succeed in finding a feasible value for x_j among the values in J we fix x_j at that value and contract vertex x_j with x_0. Move to the next variable $j \leftarrow j + 1$ and repeat.

3. Else we know that the sequence is too coarse and that all feasible values lie in the strict interior of some interval $[x_j^1, x_j^2]$ defined by consecutive values in J. In this latter case we prune all but the two essential inequalities, defining the edges of the polygon Q_{jk}, for each of the endpoints x_j^1 and x_j^2.

4. Eliminate x_j using standard Fourier elimination.

End

Notice that at most four new inequalities are created for each variable elimination. Also note that the size of J is always $O(m)$. The complexity is dominated by the search over J. Each search step requires a call to the "grapevine" procedure and there are at most $n \log m$ calls. Therefore the overall time-complexity is $O(mn^2 \log m)$ which is strongly polynomial in that it is polynomial and independent of the size of the input coefficients.

An open problem related to $LI(2)$ is the design of a strongly polynomial algorithm for optimization of an arbitrary linear function over $LI(2)$ constraints. This would imply, via duality, a strongly polynomial algorithm for generalized network flows (flows with gains and losses).

7.3 Fixed Dimensional LP's: Prune and Search

Consider the linear program $\max\{cx : Ax \leq b\}$ where A is a $m \times n$ matrix that includes the non-negativity constraints. Clearly, for fixed dimension n, there is a polynomial-time algorithm because

there are at most $\binom{m}{n}$ system of linear equations to be solved, to generate all extreme points of the feasible region. However, Megiddo [56] and Dyer [21] have shown that for the above linear program with fixed n, there is a linear-time algorithm that requires $O(m)$ elementary arithmetic operations on numbers of size bounded by a polynomial in the input data. The algorithm is highly recursive. Before we give an outline of the algorithm, some definitions are required.

DEFINITION 1: Given a linear program $\max\{cx : Ax \leq b\}$ and a linear equality $fx = q$,

(i) the inequality $fx < q$ said to hold for the optimum if either

(a) we know that $Ax \leq b$ is feasible and
$\max\{cx : Ax \leq b, fx \leq q\} > \max\{cx : Ax \leq b, fx = q\}$

or

(b) we know a row vector $y \geq 0$ such that $yA = f$ and $yb < q$.

(ii) the inequality $fx > q$ is said to hold for the optimum if either

(a) we know that $Ax \leq b$ is feasible and
$\max\{cx : Ax \leq b, fx \geq q\} > \max\{cx : Ax \leq b, fx = q\}$

or

(b) we know a vector $y \geq 0$ such that $yA = -f$ and $yb < -q$.

DEFINITION 2: For a given linear program $\max\{cx : Ax \leq b\}$ and a given linear equation $fx = q$, the position of the optimum of the linear program relative to the linear equation is said to be known if either we know that $fx < q$ holds for an optimum or $fx > q$ holds for an optimum.

An outline of the algorithm is presented below. The algorithm requires an oracle, denoted as Procedure 1, with inputs as the linear program $\max\{cx : Ax \leq b\}$ where A is a $m \times n$ matrix and a system of p linear equations $Fx = d$ with the rank of F being r. The output of Procedure 1 is either a solution to the linear program (possibly unbounded or infeasible) or a set of $\lceil p/2^{2^{r-1}} \rceil$ equations in $Fx = d$ relative to each of which we know the position of the optimum of the linear program.

Algorithm Sketch: Prune & Search

Call Procedure 1 with inputs as the linear program $\max\{cx : Ax \leq b\}$ and the system of m equations $Ax = b$. Procedure 1 either solves the linear program or identifies $k = \lceil m/2^{2^{n-1}} \rceil$ equations in $Ax = b$ relative to each of which we know the position of the optimum of the linear program. The identified equations are then omitted from the system $Ax = b$. The resulting subsystem, $A_1x = b_1$ has $m_1 = m - k$ equations. Procedure 1 is applied again with the original given linear program and the system of equations $A_1x = b_1$ as the inputs. This process is repeated until either the linear program is solved or we know the position of the optimum with respect to each of the equations in $Ax = b$. In the latter case the system $Ax \leq b$ is infeasible.

We next describe the input/output specification of Procedure 1. The procedure is highly recursive and splits into a lot of cases. This procedure requires a linear-time algorithm for the identification of the median of a given set of rationals in linear time.

Specification of Procedure 1:

INPUT: Linear program $\max\{cx : Ax \leq b\}$ where $m \times n$ matrix and a system of p equations $Fx = d$ where rank of F is r.

OUTPUT: A solution to the linear program or a set of $\lceil p/2^{2^{r-1}} \rceil$ equations in $Fx = d$ relative to each of which we know the position of the optimum as in Definition 2.

For fixed n, Procedure 1 requires $O(p+m)$ elementary arithmetic operations on numbers of size bounded by a polynomial in the size of the input data. Since at the outset $p = m$, algorithm Prune & Search solves linear programs with fixed n in linear time. Details of the validity of the algorithm as well as analysis of its linear time complexity for fixed n are given by Megiddo [56], Dyer [21] and in the book by Schrijver [71]. As might be expected, the linear-time solvability of linear programs in fixed dimension has important implications in the field of computational geometry which deals largely with two and three dimensional geometry. The book by Edelsbrunner [24] documents these connections.

The linear programming problem is known to be $\mathcal{P}$-complete and therefore we do not expect to find a parallel algorithm that achieves polylog run time. However, for fixed n, there are simple polylog algorithms [22]. In a recent paper, Sen [72] shows that linear programming in fixed dimension n can be solved in $O(\log\log^{n+1} m)$ steps using m processors in a *CRCW PRAM*.

8 Randomized Methods for Linear Programming

The advertising slogan for randomized algorithms has been "simplicity and speed" [60]. In the case of fixed-dimensional linear programming there appears to be some truth in the advertisement. In stark contrast with the very technical deterministic algorithm outlined in the last section, we will see that an almost trivial randomized algorithm will achieve comparable performance (but of course at the cost of determinism).

Consider a linear programming problem of the form

$$\min\{cx : Ax \leq b\}$$

with the following properties:

- The feasible region $\{x : Ax \leq b\}$ is non-empty and bounded.
- The objective function c has the form $(1, 0, \ldots, 0)$.
- The minimum to the linear program is unique and occurs at an extreme point of the feasible region.
- Each vertex of $\{x : Ax \leq b\}$ is defined by exactly n constraints where A is $m \times n$.

Note that none of these assumptions compromise the generality of the linear programming problem that we are considering.

Let $\mathcal{S}$ denote the constraints $Ax \leq b$. A feasible $B \subseteq \mathcal{S}$ is called optimal if it defines the uniquely optimal extreme point of the feasible region. The following randomized algorithm due to Sharir and Welzl [76] uses an incremental technique to obtain the optimal basis of the input linear program.

Algorithm: **ShW**

INPUT: The constraint set $\mathcal{S}$ and a feasible basis T.

OUTPUT: The optimal basis for the linear program.

1. If $\mathcal{S}$ equals T, return T;
2. Pick a random constraint $s \in \mathcal{S}$. Now define $\bar{T} = \mathrm{ShW}(\mathcal{S} \setminus \{s\}, T)$;
3. If the point defined by $\bar{T}$ satisfies s, output $\bar{T}$; Else output $\mathrm{ShW}(\mathcal{S}, \mathrm{opt}(\{s\} \cup \bar{T}))$
4. End

The subroutine **opt** when given an input of $n+1$ or less constraints $H \subseteq S$ returns an optimal basis for the linear program with constraints defined by H (and objective function cx). It has been shown [54] that algorithm ShW has an expected running time of $O(\min\{mn \exp\sqrt[4]{n \ln (m+1)},\ n^4 2^n m\})$. Thus algorithm **ShW** is certainly linear expected time for fixed n but has a lower complexity than `Prune & Search` if n is allowed to vary.

9 Large Scale Linear Programming

Linear programming problems with thousands of rows and columns are routinely solved either by variants of simplex method or by interior point methods. However, for several linear programs that arise in combinatorial optimization, the number of columns (or rows in the dual) are too numerous to be enumerated explicitly. The columns, however, often have a structure which is exploited to generate the columns as and when required in the simplex method. Such an approach which is refered to as column generation is illustrated next on the *cutting stock problem* (Gilmore and Gomory [31]) which is also known as the *bin packing problem* in the computer science literature.

9.1 Cutting Stock Problem

Rolls of sheet metal of standard length L are used to cut required lengths $l_i, i = 1, 2, .., m$. The j^{th} cutting pattern should be such that a_{ij}, the number of sheets length l_i cut from one roll of standard length L must satisfy $\sum_{i=1}^{m} a_{ij} l_i \leq L$. Suppose $n_i, i = 1, 2, .., m$ sheets of length l_i are required. The problem is to find cutting patterns so as to minimize the number of rolls of standard length L that are used to meet the requirements. A linear programming formulation of the problem is as follows:

Let $x_j,\ j = 1, 2, .., n$ denote the number of times the j^{th} cutting pattern is used. In general, $x_j,\ j = 1, 2, .., n$ should be an integer but in the formulation below the variables are permitted to be fractional.

$$
\begin{array}{lll}
(P1) & \min \sum_{j=1}^{n} x_j & \\
\text{Subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq n_i & i = 1, 2, .., m \\
& x_j \geq 0 & j = 1, 2, .., n \\
\text{where} & \sum_{i=1}^{m} l_i a_{ij} \leq L & j = 1, 2, .., n
\end{array}
$$

The formulation can easily be extended to allow for the possibility of p standard lengths, L_k, $k = 1, 2, \ldots, p$ from which the n_i units of length l_i, $i = 1, 2, .., m$ are to be cut.

The cutting stock problem can also be viewed as a bin packing problem. Several bins, each of standard capacity L are to be packed with n_i units of item i, each of which uses up capacity of l_i in

a bin. The problem is to minimize the number of bins used.

9.1.1 Column generation

In general, the number of columns in (P1) is too large to enumerate all the columns explicitly. The simplex method, however, does not require all the columns to be explicitly written down. Given a basic feasible solution and the corresponding simplex multipliers $w_i, i = 1, 2, .., m$, the column to enter the basis is determined by applying dynamic programming to solve the following knapsack problem:

$$\begin{aligned} (P2) \quad z = \max \quad & \sum_{i=1}^{m} w_i a_i \\ \text{Subject to} \quad & \sum_{i=1}^{m} l_i a_i \leq L \\ & a_i \geq 0 \text{and integer} \;\; i = 1, 2, .., m \end{aligned}$$

Let $a_i^*, i = 1, 2, .., m$ denote an optimal solution to (P2). If $z > 1$, the k^{th} column to enter the basis has coefficients $a_{ik} = a_i^*, i = 1, 2, .., m$.

Using the identified columns, a new improved (in terms of the objective function value) basis is obtained and the column generation procedure is repeated. A major iteration is one in which (P2) is solved to identify, if there is one, a column to enter the basis. Between two major iterations, several minor iterations may be performed to optimize the linear program using only the available (generated) columns.

If $z \leq 1$ the current basic feasible solution is optimal to (P1). From a computational point of view, alternative strategies are possible. For instance, instead of solving (P2) to optimality, a column to enter the basis can be indentified as soon as a feasible solution to (P2) with an objective function value greater than 1 has been found. Such an approach would reduce the time required to solve (P2) but may increase the number of iterations required to solve (P1).

A column, once generated may be retained, even if it comes out of the basis at a subsequent iteration, so as to avoid generating the same column again later on. However, at a particular iteration some columns which appear unattractive in terms of their reduced costs, may be discarded in order to avoid having to store a large number of columns. Such columns can always be generated again subsequently, if necessary. The rationale for this approach is that such unattractive columns will rarely be required subsequently.

The dual of (P1) has a large number of rows. Hence column generation may be viewed as row generation in the dual. In other words, in the dual we start with only a few constraints explicitly written down. Given an optimal solution w to the current dual problem (i.e. with only a few constraints which have been explicitly written down) find a constraint that is violated by w or

conclude that no such constraint exists. The problem to be solved for identifying a violated constraint, if any, is exactly the separation problem that we encountered in Section 5.

9.2 Decomposition

Large scale linear programming problems sometimes have a block diagonal structure. Consider for instance, the following linear program:

$$(P3) \qquad \max \textstyle\sum_{j=1}^{p} c^j x^j \tag{1}$$

$$\text{Subject to} \qquad \textstyle\sum_{j=1}^{p} A^j x^j = b \tag{2}$$

$$D^j x^j = d^j \quad j = 2,3,...,p \tag{3}$$

$$x^j \geq 0 \quad j = 1,2,..,p \tag{4}$$

where A^j is a $m \times n_j$ matrix; D^j is a $m_j \times n_j$ matrix; x_j is a $n_j \times 1$ column vector; c^j is a $1 \times n_j$ row vector; b is a $m \times 1$ column vector; d^j is a $m_j \times 1$ column vector.

The constraints (2) are referred to as the linking master constraints. The p sets of constraints (3) and (4) are referred to as sub-problem constraints. Without the constraints (2), the problem decomposes into p separate problems which can be handled in parallel. The Dantzig-Wolfe [20] decomposition approach to solving (P3) is described next.

Clearly, any feasible solution to (P3) must satisfy constraints (3) and (4). Now consider the polyhedron P_j, $j = 2,3,..,p$ defined by the constraints (3) and (4). By the representation theorem of polyhedra (see Section 2) a point $x^j \in P_j$ can be written as

$$\begin{aligned} x^j &= \sum_{k=1}^{h_j} x^{jk}\rho_{jk} + \sum_{k=1}^{g_j} y^{jk}\mu_{jk} \\ \sum_{k=1}^{h_j} \rho_{jk} &= 1 \\ \rho_{jk} &\geq 0 \quad k = 1,2,...,h_j \\ \mu_{jk} &\geq 0 \quad k = 1,2,...,g_j \end{aligned}$$

where x^{jk} $(k = 1,2,...,h_j)$ are the extreme points and y^{jk} $(k = 1,2,...,g_j)$ are the extreme rays of P_j.

Now substituting for x^j, $j = 2,3,...,p$ in (1) and (2), (P3) is written as

$$(P4) \quad \max\{c^1x^1 + \textstyle\sum_{j=2}^{p}\left[\sum_{k=1}^{h_j}(c^jx^{jk})\rho_{jk} + \sum_{k=1}^{g_j}(c^jy^{jk})\mu_{jk}\right]\}$$

$$\text{Subject to} \quad A^1x^1 + \textstyle\sum_{j=2}^{p}\left[\sum_{k=1}^{h_j}(A^jx^{jk})\rho_{jk} + \sum_{k=1}^{g_j}(A^jy^{jk})\mu_{jk}\right] = b \tag{5}$$

$$\textstyle\sum_{k=1}^{h_j}\rho_{jk} = 1 \quad j = 2,3,..,p \tag{6}$$

$$\rho_{jk} \geq 0 \quad j = 2,3,..,p; \quad k = 1,2,..,h_j$$

$$\mu_{jk} \geq 0 \quad j = 2,3,..,p; \quad k = 1,2,..,g_j$$

In general, the number of variables in (P4) is an exponential function of the number of variables $n_j, j = 1, 2, .., p$ in (P3). However, if the simplex method is adapted to solve (P4), the extreme points or the extreme rays of $P_j, j = 2, 3, .., p$ and consequently the columns in (P4) can be generated, as and when required, by solving the linear programs associated with the p subproblems. This column generation is described next.

Given a basic feasible solution to (P4), let w and u be row vectors denoting the simplex multipliers associated with constraints (5) and (6) respectively. For $j = 2, ..., p$ solve the following linear programming sub-problems:

$$(S_j) \quad z_j = \min(wA^j - c^j)x^j$$
$$D^j x^j = d^j$$
$$x^j \geq 0$$

Suppose z_j is finite. An extreme point solution x^{jt} is then identified. If $z_j + u_j < 0$, a candidate column to enter the basis is given by $\begin{pmatrix} A^j x^{jt} \\ 1 \end{pmatrix}$. On the other hand if $z_j + u_j \geq 0$, there is no extreme point of P_j that gives a column to enter the basis at this iteration. Suppose the optimal solution to S_j is unbounded. An extreme ray y^{jt} of P_j is then identified and a candidate column to enter the basis is given by $\begin{pmatrix} A^j y^{jt} \\ 0 \end{pmatrix}$. If the simplex method is used to solve S_j, the extreme point or the extreme ray is identified automatically. If a column to enter the basis is identified from any of the sub-problems, a new improved basis is obtained and the column generation procedure is repeated. If none of the sub-problems identify a column to enter the basis, the current basic solution to (P4) is optimal.

As in Section 9.1, a major iteration is when the sub-problems are solved. Instead of solving all the p sub-problems at each major iteration, one option is to update the basis as soon as a column to enter the basis has been identified in any of the sub-problems. If this option is used at each major iteration,the sub-problems that are solved first are typically the ones that were not solved at the previous major iteration.

The decomposition approach is particularly appealing if the sub-problems have a special structure that can be exploited. Note that only the objective functions for the sub-problems change from one major iteration to another. Given the current state of the art, ($P4$) can be solved in polynomial time (polynomial in the problem parameters of ($P3$)) by the ellipsoid method but not by the simplex method or interior point methods. However ($P3$) can be solved in polynomial time by interior point methods.

It is interesting to note that the reverse of decomposition is also possible. In other words, suppose we start with a statement of a problem and an associated linear programming formulation with a large number columns (or rows in the dual). If the column generation (or row generation in the dual) can be accomplished by solving a "compact" linear program, then a "compact" formulation of the original problem can be obtained. Here "compact" refers to the number of rows and columns being bounded by a polynomial function of the parameters (not the number of the columns in the original linear programming formulation) in the statement of the original problem. This result due to Martin [53] enables one to solve the problem in the polynomial time by solving the compact formulation directly using interior point methods.

9.3 Compact Representation

Consider the following linear program:

$$\begin{aligned} (P5) \quad & \min cx \\ \text{Subject to} \quad & Ax \geq b \\ & x \geq 0 \\ & x \in \textstyle\bigcap_{j=1}^{p} P_j \end{aligned}$$

where A is a $m \times n$ matrix; x is a $n \times 1$ vector; c is a $1 \times n$ vector and b is a $m \times 1$ vector; P_j, $j = 1, 2, ..., p$ is a polyhedron and p is bounded by a polynomial in m and n.

Without loss of generality, it is assumed that $P_j, j = 1, 2, ..., p$ is non-empty and $(P5)$ is feasible. Given $\bar{x}$ such that $A\bar{x} \geq b$, the constraint identification or separation problem $S_j(\bar{x})$ with respect to P_j is to *either* (a) conclude that $\bar{x} \in P_j$, *or* (b)find a valid inequality $D_i^j x \leq d_i^j$ that is satisfied by $x \in P_j$ but $D_i^j \bar{x} > d_i^j$.

Suppose the separation problem $S_j(\bar{x})$ can be solved by the following linear program

$$\begin{aligned} S_j(\bar{x}): \quad z_j = \quad & \max (\bar{x}^T G^j + g^j) y \\ & F^j y \leq f^j \\ & y \geq 0 \end{aligned}$$

where

G^j is an $n \times k$ matrix; F^j is a $r \times k$ matrix;

g^j is a $1 \times k$ vector; f^j is a $r \times 1$ vector;

y is a $k \times 1$ vector; r and k are bounded by a polynomial in m and n; and

$\bar{x} \in P_j \bigcap \{x : Ax \geq b\}$ if and only if $z_j \leq h^j \bar{x} + k_j$ where h^j is a $1 \times n$ vector and k_j is a scalar.

Now, if w^j denotes the dual variables associated with the $F^j y \leq f^j$ constraints in S_j, it follows from the duality theorem of linear programming that a compact representation of $(P5)$ is given by

$$\begin{aligned} & \min cx \\ \text{Subject to} \quad & Ax \geq b \\ & (F^j)^T w^j - (G^j)^T x \geq (g^j)^T \quad j = 1, 2, ...p \\ & (f^j)^T w^j - h^j x \leq k_j \quad j = 1, 2, ...p \\ & x \geq 0 \\ & w^j \geq 0 \quad j = 1, 2, ..., p \end{aligned}$$

Note that this approach to obtaining a compact formulation is predicated on being able to formulate the separation problem as a compact linear program. This may not always be possible. In fact, Yannakakis [82] shows that for a b-matching problem under a symmetry assumption, no compact formulation is possible. This despite the fact that b-matching can be solved in polynomial time using a polynomial-time separation oracle.

An Application: Neural Net Loading

The decision version of the Hopfield neural net loading problem (cf. [64]) is:

Given y^i $(i = 1, 2, ..., p)$ where each y^i is a n-dimensional training vector whose components are $(+1, -1)$, construct a symmetric $n \times n$ synaptic weight matrix W, such that for every n-dimensional vector v whose components are $(+1, -1)$, the following holds:

$$\boxed{\text{If } d(y^i, v) \leq k \text{ then } y^i = sgn(Wv) \text{ for } i = 1, 2, \ldots, p}$$

A feasible W would represent a Hopfield net with a radius of direct attraction of at least k around each training vector, i.e. a robust network of associative memory. Here k is specified and $d(y^i, v)$ is the Hamming distance between y^i and v. For $t = 1, 2, ..., p$, let $v^{tq}, q = 1, 2, ..., m_t$ be all the vectors whose components are $(+1, -1)$ and $d(y^t, v^{tq}) \leq k$. The Hopfield neural net loading problem is to find a matrix W such that

$$(P6) \quad \sum_{j=1}^{n} y_i^t w_{ij} v_j^{tq} \geq 1 \text{ for } i = 1, 2, \ldots, n;\ t = 1, 2, \ldots, p; \text{ and } q = 1, 2, \ldots, m_t$$

This is a linear program with the number of inequalities equal to $pn \begin{pmatrix} n \\ k \end{pmatrix}$ which is huge. The synaptic weights have to be symmetric, so in addition to the inequalities given above, $(P6)$ would include the constraints $w_{ij} = w_{ji}$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

Given the weights $\bar{w_{uv}}, u = 1,2,...,n; v = 1,2,...,n$, the separation problem, $S_{it}(\bar{w})$ for specified i and t, where $1 \leq i \leq n$ and $1 \leq t \leq p$, can be formulated as follows ([12]):

Let

$$u_{ij}^{tq} = \begin{cases} 1 & \text{if } v_j^{tq} = -y_j^t \\ 0 & \text{if } v_j^{tq} = y_j^t \end{cases}$$

Since $d(y^t, v^{tq}) \leq k$ it follows that

$$\sum_{j=1}^{n} u_{ij}^{tq} \leq k \quad q = 1,2,..,m_t$$

Note that for specified i, t, and q, the inequality in $(P6)$ is equivalent to

$$\sum_{j=1}^{n} y_i^t \bar{w}_{ij} [y_j^t (-u_{ij}^{tq}) + y_j^t (1 - u_{ij}^{tq})] \geq 1$$

which reduces to

$$-\sum_{j=1}^{n} 2 y_i^t y_j^t w_{ij} u_{ij}^{tq} + \sum_{j=1}^{n} y_i^t y_j^t w_{ij} \geq 1$$

Consequently, the separation problem after dropping the superscript q is

$$S_{it}(\bar{w}) : z_{it} = \max \sum_{j=1}^{n} 2 y_i^t y_j^t w_{ij} u_{ij}^t$$

$$\text{Subject to} \quad \sum_{j=1}^{n} u_{ij}^t \leq k \tag{7}$$

$$0 \leq u_{ij}^t \leq 1 \quad j = 1,2,...,n \tag{8}$$

Note that $S_{it}(\bar{w})$ is trivial to solve and always has a solution such that $u_{ij}^t = 0$ or 1. It follows that for given i and t, $\bar{w}$ satisfies the inequalities in $(P6)$ corresponding to $q = 1,2,..,m_t$ if and only if

$$z_{it} \leq \sum_{j=1}^{n} y_i^t y_j^t \bar{w}_{ij} - 1$$

Let θ_{it} and $\beta_{ij}^t, j = 1,2,...,n$ denote the dual variables associated with constraints (7) (8) respectively. Now applying the compact representation result stated above, a compact formulation of the neural net loading problem (P6) is as follows:

$$\begin{aligned}
\theta_{it} + \beta ij^t - 2 y_i^t y_j^t w_{ij} &\geq 0 \quad i = 1,2,..,n \;\; t = 1,2,..,p \;\; j = 1,2,..,n \\
k\,\theta_{it} + \sum_{j=1}^{n} \beta ij^t - \sum_{j=1}^{n} y_i^t y_j^t w_{ij} &\leq -1, \quad i = 1,2,..,n \;\; t = 1,2,..,p \\
\theta_{it} &\geq 0 \quad i = 1,2,..,n; \;\; t = 1,2,..,p \\
\beta_{ij}^t &\geq 0 \quad i = 1,2,..,n; \;\; t = 1,2,..,p; \;\; j = 1,2,..,n
\end{aligned}$$

With the symmetry condition $w_{ij} = w_{ji}$ for $i = 1,2,\ldots,n$ and $j = 1,2,\ldots,n$ added in, we now have a linear programming formulation of the Hopfield net loading problem which is compact in the size of the network $n \times n$ and the size of the training set $n \times p$.

10 Linear Programming: A User's Perspective

This chapter has been written for readers interested in learning about the algorithmics of linear programming. However, for someone who is primarily a user of linear programming software there are a few important concerns that we address briefly here.

1. THE EXPRESSION OF LINEAR PROGRAMMING MODELS. The data sources from which the coefficients of a linear programming model are generated, may be organized in a format that is incompatible with the linear programming software in use. Tools to facilitate this translation have come to be known as "matrix generators" [29]. Over the years such tools have evolved into more complete modeling languages (for example, AMPL [28], GAMS [55]).

2. THE VIEWING, REPORTING AND ANALYSIS OF RESULTS. This issue is similar to that of model expression. The results of a linear programming problem when presented as raw numerical output are often difficult for a user to digest. Report writers and modeling languages like the ones mentioned above usually provide useful features for processing the output into a user friendly form. Because of the widespread use of linear programming in decision support, user interfaces based on spreadsheets have become popular with software vendors [73].

3. TOOLS FOR ERROR DIAGNOSIS AND MODEL CORRECTION. Many modeling exercises using linear programming involve a large amount of data and are prone to numerical as well as logical errors. Some sophisticated tools [35] are now available for helping users in this regard.

4. SOFTWARE SUPPORT FOR LINEAR PROGRAMMING MODEL MANAGEMENT. Proliferation of linear programming models can occur in practice because of several reasons. The first is that when a model is being developed, it is likely that several versions are iterated on before converging to a suitable model. Scenario analysis is the second type of source for model proliferation. Finally, iterative schemes such as those based on column generation or stochastic linear programming, may require the user to develop a large number of models. The software support in optimization systems for helping users in all such situations have come to be known as tools for "model management" [27,30].

5. THE CHOICE OF A LINEAR PROGRAMMING SOLUTION PROCEDURE. For linear programs with special structure (for example, network flows [1]) it pays to use specialized versions of linear programming algorithms. In the case of general linear optimization software, the user may be provided a choice of a solution method from a suite of algorithms. While the right choice of an algorithm is a difficult decision, we hope that insights gained from reading this chapter would help the user.

11 Defining Terms

- `Analytic Center`: The interior point of a polytope at which the product of the distances to the facets is uniquely maximized.
- `Column Generation`: A scheme for solving linear programs with a huge number of columns.
- `Compact Representation`: A polynomial size linear programming representation of an optimization problem.
- `Decomposition`: A strategy of divide and conquer applied to large scale linear programs.
- `Duality` The relationship between a linear program and its dual.
- `Extreme Point`: A corner point of a polyhedron.
- `Linear Program`: Optimization of a linear function subject to linear equality and inequality constraints.
- `Matrix Factorization`: Representation of a matrix as a product of matrices of simple form.
- `Polyhedral Cone`:The set of solutions to a finite system of homogeneous linear inequalities on real-valued variables.
- `Polyhedron`: The set of solutions to a finite system of linear inequalities on real-valued variables. Equivalently, the intersection of a finite number of linear half-spaces in $\Re^n$.
- `Polytope`: A bounded polyhedron.
- `Relaxation`: An enlargement of the feasible region of an optimization problem. Typically, the relaxation is considerably easier to solve than the original optimization problem.
- `Separation`: Test if a given point belongs to a convex set and if it does not, identify a separating hyperplane.
- `Strongly Polynomial`: Polynomial algorithms with the number of elementary arithmetic operations bounded by a polynomial function of the dimensions of the linear program.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows*, Prentice Hall, (1993).

[2] Akgul, M., Topics in Relaxation and Ellipsoidal Methods, Research notes in Mathematics, *Pitman Publishing Ltd.*, (1984).

[3] F.Alizadeh, Interior point methods in semidefinite programming with applications to combinatorial optimization, in *SIAM Journal on Optimization*, Vol. 5, No. 1, pp. 13-51, February 1995.

[4] B.I. Aspvall and Y. Shiloach, A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality, *FOCS*, (1979) pp. 205-217.

[5] D. Bertsimas and J.N. Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, Massachusetts (1997).

[6] R.E.Bixby, Progress in Linear Programming, *ORSA Journal on Computing*, Vol. 6, No. 1, (1994) 15-22.

[7] R.Bland, D.Goldfarb, and M.J.Todd, The ellipsoid method: a survey, in *Operations Research* **29** (1981), 1039-1091.

[8] K.H. Borgwardt, *The Simplex Method: A Probabilistic Analysis*, Springer-Verlag, Berlin Heidelberg (1987).

[9] R.N. Černikov, The Solution of Linear Programming Problems by Elimination of Unknowns, *Doklady Akademii Nauk* **139** (1961) 1314-1317. [Translation in: *Soviet Mathematics Doklady* **2** (1961) 1099-1103.]

[10] C. Caratheodory, Über den Variabiltätsbereich der Fourierschen Konstanten von positiven harmonischen Funktionen, *Rendiconto del Circolo Matematico di Palermo* **32** (1911) 193-217.

[11] V. Chandru, Variable Elimination in Linear Constraints, *The Computer Journal*, **36**, No. 5, August 1993, 463-472.

[12] V. Chandru, A. Dattasharma, S.S. Keerthi, N.K. Sancheti and V. Vinay, Algorithms for the Design of Optimal Discrete Neural Networks, in *Proceedings of the Sixth ACM/SIAM Symposium on Discrete Algorithms*, SIAM Press, January 1995.

[13] V. Chandru and B.S.Kochar, A Class of Algorithms for Linear Programming, *Research Memorandum 85-14*, School of Industrial Engineering, Purdue University, November 1985.

[14] V. Chandru and B.S.Kochar, Exploiting Special Structures Using a Variant of Karmarkar's Algorithm, *Research Memorandum 86-10*, School of Industrial Engineering, Purdue University, June 1986.

[15] V. Chvatal, *Linear Programming*, Freeman Press, New York (1983).

[16] W.Cook, L.Lovász, and P.Seymour (Editors), *Combinatorial Optimization: Papers from the DIMACS Special Year*, Series in Discrete Mathematics and Theoretical Computer Science, Volume 20, AMS, 1995.

[17] CPLEX *Using the CPLEX callable Library and CPLEX mixed integer library*, CPLEX Optimization, Inc., 1993.

[18] G.B.Dantzig, Maximization of a linear function of variables subject to linear inequalities, in *Activity Analysis of Production and Allocation*, edited by C.Koopmans, Wiley, New York (1951), 339-347.

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

[20] G.B.Dantzig and P.Wolfe, The decomposition algorithm for linear programming, *Econometrica*, bf 29, (1961), 767-778.

[21] M.E. Dyer, Linear time algorithms for two- and three-variable linear programs, *SIAM Journal on Computing* **13** (1984) 31-45.

[22] M.E. Dyer, A parallel algorithm for linear programming in fixed dimension, *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, ACM Press (1995) pp. 345-349.

[23] Ecker, J.G. and M.Kupferschmid, " An Ellipsoid Algorithm for Nonlinear Programming", *Mathematical programming, 27* (1983).

[24] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag, 1987.

[25] J. Edmonds, Submodular functions, matroids and certain polyhedra, in *Combinatorial Structures and their Applications*, edited by R. Guy et al., Gordon Breach, (1970) 69-87.

[26] Gy. Farkas, A Fourier-féle mechanikai elv alkalmazásai, (in Hungarian), *Mathematikai és Természettudományi Értesitõ* **12** (1894) 457-472.

[27] R.Fourer, Software for Optimization: A Buyer's Guide (Parts I and II), in *INFORMS Computer Science Technical Section Newsletter*, Volume 17, Number 1/2, (1996).

[28] R.Fourer, D.M.Gay, and B.W.Kernighian, *AMPL: A Modeling Language for Mathematical Programming*, Scientific Press, 1993.

[29] L.B.J. Fourier, reported in : Analyse des travaux de l'Academie Royale des Sciences, pendant l'annee 1824, Partie mathematique, *Histoire de l'Academie Royale des Sciences de l'Institut de*

France 7 (1827) xlvii-lv. (Partial English Translation in: D.A. Kohler, Translation of a Report by Fourier on his Work on Linear Inequalities, *Opsearch* **10** (1973) 38-42).

[30] A.M. Geoffrion, An Introduction to Structured Modeling, *Management Science* **33** (1987) 547-588.

[31] P.Gilmore and R.E.Gomory, A linear programming approach to the cutting stock problem, Part I, *Operations Research*, **9**, 849-854; Part II, *Operations Research*, **11**, 1963, 863-887.

[32] M.X.Goemans and D.P.Williamson, .878 approximation algorithms MAX CUT and MAX 2SAT, in *Proceedings of ACM STOC*, 1994, pp. 422-431..

[33] M.Grötschel, L.Lovasz and A.Schrijver,"The ellipsoid method and its consequences in Combinatorial optimization", Combinatorica, **1**, (1982) 169-197.

[34] M.Grötschel, L.Lovasz, and A.Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

[35] H.J. Greenberg, *A computer-assisted analysis system for mathematical programming models and solutions: A user's guide for ANALYZE*, Kluwer Academic Publishers, Boston (1993).

[36] G.H. Golub and C.F. van Loan, *Matrix Computations*, The Johns Hopkins University Press (1983).

[37] Hacijan, L.G., "A Polynomial Algorithm in Linear Programming", *Soviet Math. Dokl.*, 20, (1979) 191-194.

[38] M. Haimovich, The simplex method is very good! On the expected number of pivot steps and related properties of random linear programs, *Unpublished Manuscript*, (1983).

[39] D. Hochbaum and Joseph Naor, Simple and fast algorithms for linear and integer programs with two variables per inequality, *Proceedings of the Symposium on Discrete Algorithms (SODA)* (1992) SIAM Press (also in the Proceedings of the Second Conference on Integer Programming and Combinatorial Optimization *IPCO*, Pittsburgh, June 1992)

[40] T. Huynh, C. Lassez and J-L. Lassez, Practical Issues on the Projection of Polyhedral Sets, *Annals of Mathematics and Artificial Intelligence* **6** (1992) 295-316.

[41] IBM, *Optimization Subroutine Library - Guide and Reference (Release 2)*, Third Edition, 1991.

[42] A. Itai, Two-Commodity Flow, *Journal of the ACM* **25** (1978) 596-611.

[43] G. Kalai and D.J. Kleitman, A quasi-polynomial bound for the diameter of graphs of polyhedra, *Bulletin of the American Mathematical Society*, **26** (1992) 315-316.

[44] L.V. Kantorovich, Mathematical methods of organizing and planning production, (in Russian), Publication House of the Leningrad State University, Leningrad (1939); English translation in *Management Science* **6** (1959) 366-422.

[45] A. Kamath and N.K. Karmarkar, A continuous method for computing bounds in integer quadratic optimization problems, *Journal of Global Optimization*, **2**, 229-241 (1992).

[46] Karmarkar, N. K., A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica*, *4*, (1984) 373-395.

[47] R.M. Karp and C.H. Papadimitriou, On Linear Characterizations of Combinatorial Optimization Problems, *SIAM Journal on Computing*, **11** (1982), 620-632.

[48] V. Klee and G.J. Minty, How good is the simplex algorithm?, in *Inequalities III*, edited by O. Shisha, Academic Press (1972).

[49] J.K. Lenstra, A.H.G. Rinooy Kan and A. Schrijver (editors), *History of Mathematical Programming: A Collection of Personal Reminiscences*, North Holland (1991).

[50] L.Lovász, *An Algorithmic Theory of Numbers, Graphs and Convexity*, SIAM Press, 1986.

[51] L.Lovasz and A.Schrijver, Cones of matrices and setfunctions, *SIAM Journal on Optimization* 1, (1991), pp. 166-190.

[52] I.J.Lustig, R.E.Marsten, and D.F.Shanno, Interior Point Methods for Linear Programming: Computational State of the Art, *ORSA J. on Computing*, Vol. 6, No. 1, (1994) 1-14.

[53] R.K.Martin, Using separation algorithms to generate mixed integer model reformulations, *Operations Research Letters*, 10, (1991) 119-128.

[54] J. Matousek, M. Sharir and E. Welzl, A subexponential bound for linear programming, in *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, ACM Press, (1992) pp. 1-8.

[55] J. Bisschop and A. Meerhaus, On the development of a General Algebraic Modeling System (GAMS) in a Strategic Planning Environment, *Mathematical Programming Study* **20** (1982) 1-29.

[56] N.Megiddo, On finding primal- and dual-optimal bases, in *ORSA Journal on Computing* 3, pp. 63-65.

[57] N.Megiddo, Towards a genuinely polynomial algorithm for linear programming, *SIAM Journal on Computing*, **12** (1983) 347-353.

[58] N.Megiddo, Linear Programming in linear time when the dimension is fixed, *JACM* **31** (1984) 114-127.

[59] S.Mehrotra, On the implementation of a primal-dual interior point method, *SIAM Journal on Optimization*, 2:4, 1992, pp. 575-601.

[60] R. Motwani and P.Raghavan, *Randomized Algorithms*, Cambridge University Press (1996).

[61] B.A. Murtagh, *Advanced Linear Programming: Computation and Practice* McGraw Hill, New York, 1981.

[62] K.G. Murty, *Linear Programming*, Wiley, New York, 1983.

[63] C.G. Nelson, An $O(n^{\log n})$ algorithm for the two-variable-per-constraint linear programming satisfiablity problem, *Technical Report AIM-319*, Dept. of Computer Science, Stanford University (1978).

[64] Orponen, P., "Neural Networks and Complexity Theory", in *Proceedings of the* 17^{th} *International Symposium on Mathematical Foundations of Computer Science* (ed. I.M. Havel and V. Koubek) Lecture Notes in Computer Science 629, Springer-Verlag 1992, pp. 50-61.

[65] M.W.Padberg, *Linear Optimization and Extensions*, Springer-Verlag, 1995.

[66] M.W.Padberg and M.R.Rao, The Russian method for linear inequalities, Part III, Bounded integer programming, Preprint, New York University, (1981).

[67] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall (1982).

[68] C.H.Papadimitriou and M.Yannakakis, Optimization, approximation, and complexity classes, in *Journal of Computer and Systems Sciences* 43, 1991, pp. 425-440.

[69] R. Saigal, *Linear Programming: A Modern Integrated Analysis*, Kluwer Press, 1995.

[70] M.J. Saltzman, Implementation of an interior point LP algorithm on a shared-memory vector multiprocessor, in *Computer Science and Operations Research*, edited by O.Balci, R.Sharda and S.A.Zenios, Pergamon Press, 1992, pp. 87-104.

[71] A.Schrijver, *Theory of Linear and Integer Programming*, John Wiley, 1986.

[72] S. Sen, Parallel multidimensional search using approximation algorithms: with applications to linear-programming and related problems, *Proceedings of SPAA*, (1996).

[73] R. Sharda, Linear Programming Solver Software for Personal Computers: 1995 Report, *OR/MS Today* **22:5** (1995) 49-57.

[74] D.B.Shmoys, Computing near-optimal solutions to combinatorial optimization problems, in [16] cited above, (1995) 355-398.

[75] Shor, N.Z., "Convergence Rate of the Gradient Descent Method with Dilation of the Space", *Cybernetics, 6*, (1970).

[76] M. Sharir and E. Welzl, A combinatorial bound for linear programming and related problems, in *Proceedings of the 9th Symposum on Theoretical Aspects of Computer Science*, LNCS 577, Springer Verlag (1992) 569-579.

[77] R.E. Stone and C.A. Tovey, The simplex and projective scaling as iterated reweighting least squares, *SIAM Review* **33** (1991) 220-237.

[78] E. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, *Operations Research*, **34**, 250-256 (1986).

[79] M.J. Todd, Exploiting special structure in Karmarkar's Algorithm for Linear Programming, *Technical Report 707*, School of Operations Research and Industrial Engineering, Cornell University, July 1986.

[80] H.Weyl, Elemetere Theorie der konvexen polyerer, Comm. Math. Helv., Vol. I, (1935) 3-18, (English translation in *Annals of Mathematics Studies*, **24**, Princeton, 1950).

[81] S.J.Wright, Primal-Dual Interior-Point Methods, SIAM Press, 1997.

[82] M.Yannakakis, Expressing Combinatorial optimization problems by linear programs, in *Proceedings of ACM Symposium of Theory of Computing*, (1988) 223-228.

[83] G.M.Ziegler, *Lectures on Polytopes*, Springer Verlag, 1995.

12 Other Sources

JOURNALS:

Research publications in linear programming are dispersed over a large range of journals. The following is a partial list which emphasize the algorithmic aspects.

Mathematical Programming

Mathematics of Operations Research

Operations Research

INFORMS Journal on Computing

Operations Research Letters

SIAM Journal on Optimization

SIAM Journal on Computing

SIAM Journal on Discrete Mathematics

Algorithmica

Combinatorica

Linear programming professionals frequently use the following newsletters:

- INFORMS Today (earlier OR/MS Today) published by The Institute for Operations Research and Management Science.
- INFORMS CSTS Newsletter published by the INFORMS computer science technical section.
- Optima published by the Mathematical Programming Society.

WWW:

The linear programming FAQ (frequently asked questions) facility is maintained at

http://www.mcs.anl.gov/home/otc/Guide/faq/

To have a linear program solved over the internet check the following locations.

http://www.mcs.anl.gov/home/otc/Server/

http://www.mcs.anl.gov/home/otc/Server/lp/

The universal input standard for linear programs is the MPS format [61].