Linear Programming

Jacob Denson

November 13, 2016

Table Of Contents

Chapter 1

Introduction

Most problems in computing science can be described in a specific format. The problem, given an input, has a certain set of feasible outputs. On this output we can place a partial ordering. The problem is then phrased as an optimization – find the maximum or minimum feasible solution according to the ordering placed on the solutions. Examples of these problems are finding the 'shortest' path, the 'longest' common subsequence. Since we only have a finite set of solutions, the maximum can always be obtained in these problems.

Linear Programming attempts to provide a general solution to optimization problems with a 'linear' structure. We attempt to optimize a linear reward function, subject to linear constraints, where the parameters we attempt to optimize take values in a linear space. We give an example to show this idea in action.

Example. *Suppose we are stocking items at a store. Each item needs to be created from a specific number of ingredients we need to buy, each with a specific stock. For simplicity, suppose we can only stock two items, apple pie, and chocolate bars. Let a be the number of apple pies we make, and c the number of chocolate bars.*

For each apple pie we make, we must buy 3 *apples, and use* $1/2$ *bags of sugar, and for every three chocolate bars,* 1 *bag of sugar must be used. Each apple is* \$1*, and each bag of sugar is* \$3*. Then we can model the cost in dollars of the process by the equation*

$$9a/2 + 1/3c$$

However, we are not only buying items, but selling them as well. Each apple pie is sold for \$6*, and each chocolate bar for* \$1*. The problem is to find the optimal number of applie pies and chocolate bars to make, maximizing the equation*

$$(6 - 9/2)a + (1 - 1/3)c$$

The problem with the current description is that theoretically we could make as many apple pies and chocolate bars as we wanted, making theoretically an unbounded amount of money. Thus we add additional constraints. There is only a limited supply of sugar: we can only buy 50 bags. This means that, if we let $s = 1/2a + 1/3c$ *be the number of bags of sugar we buy*

$$1.2a + 1.3c \leqslant 50$$

We can additionally only buy 100 apples

$$a/3 \leqslant 100$$

What's more, it takes time to produce these items. In the time alloted, we can only produce 40 items.

$$a + c \leqslant 40$$

With these constraints, the problem of maximizing revenue becomes much more involved. This is an example of a linear program. If we require the number of apple pies and chocolate bars we make to be integers, then this is an integer linear programming, which is computationally much harder.

A linear program is specified by a reward (objective) functional $r \in V^*$, contraint functionals $c_1, \ldots, c_n \in V^*$, and a contraint vector $b \in \mathbf{F}^n$ (where $\mathbf{F}$ is either the rational or real numbers). The aim of the linear program is to maximize $r(v)$, subject to the contraints that $c_i(v) \leqslant b_i$ (the set of all v such that $c_i(v) \leqslant b_i$ is known as the solution space, and forms a polyhedron, a finite intersection of half planes). For computational purposes, we can choose a m dimensional basis for V, and specify the problem with a contraint matrix $M \in M_{n,m}(\mathbf{F})$ and reward functional $r \in \mathbf{R}^m$. Our job is then to maximize $\langle r, v \rangle$, subject to the constraints

$$\sum M_{ij} v_j \leqslant b_i$$

which we write in condensed form as $Mv \leqslant b$ (which makes sense if we make $\mathbf{R}^n$ into a Riesz space, viewing it as functions over a discrete set

of n points). We say a linear program is **infeasible** if the solution space is empty, and unbounded if there is x_i in the solution space such that $r(x_i) \to \infty$.

In a more general form, we allow more fluid constraints for the problem, which can be simplified to the existing problem with little asymptotic cost. For instance, we may allow constraints of the form $c_i(v) \geqslant b_i$. This can be reduced to the form $-c_i(v) \leqslant -b_i$, which fits into our current assumptions nicely. We can also allow equalities of the form $c_i(v) = b_i$, which corresponds to two inequalities of the form $c_i(v) \leqslant b_i$ and $c_i(v) \geqslant b_i$. Finally, we shall assume there is a basis $x_1, \ldots, x_n$ such that, relative to this basis, the constraints $x_i^*(v) \geqslant 0$ are satisfied for any v in the solution space. In general, if we do not have these constraints on a certain basis $x_1, \ldots, x_n$, we may introduce a new linear program on $V \times V$, with constraints (and objective function) induced from V by the map $T : (y, x) \mapsto y - x$. Any maximum solution on this new linear program induces a solution on the original programming by considering the image under T, and since each element of V can be written as $y - x$, where $y_i, x_i \geqslant 0$ for all i, we can introduce these new constraints at no fault.

Linear programming can be considered a specialization of convex optimization, since the solution space is always the intersection of finitely many closed hyperplanes, and is therefore convex. We recall that a point in a convex set is an **extreme point** if it does not lie between any two points in the convex set. An alternate criteria for x on a set X is that for any y, either $x + y$ or $x - y$ is not in X. Furthermore, a point on a polyhedron is an extreme point if it is the maximum of some linear functional over the polyhedron, which indicates their importance in linear ptimization. The extreme points of a linear constraint problem are particularly easily to quantify. If we consider a particular constraint matrix $Ax \leqslant b$, we shall let A^x denote the rows the matrix obtained from A for which the inequality is tight, i.e. $(Ax)_i = b_i$. Similarily, we let b^x denote denote the subvector of b such that $A^x x = b^x$.

Theorem 1.1. *The extremities to $Ax \leqslant b$ are those where A^x has full rank.*

Proof. Let A^x have full rank, and suppose $Ax \leqslant b - Ay$, $Ax \leqslant b + Ay$. Then $A^x y = 0$, so $y = 0$ since A^x has full rank. Conversely, suppose that A^x does not have full rank. Then there is $y \neq 0$ for which $A^x y = 0$. If $\varepsilon > 0$ is made small enough, we have $Ax \leqslant b - Ay$ and $Ax \leqslant b + Ay$. □

Lemma 1.2. *If a linear program has an optimal solution, it has an optimal solution which is an extreme point.*

Proof. Let X be the solution space of the linear program. We take some optimal solution x, and slide it along $\ker(r)$. If y is an extreme point of $Y = (x + \ker(r)) \cap X$ (which must exist because of the $x_i \geqslant 0$ condition), then we claim $x + y$ is also an extreme point of X. If $x + y + z \in X$ and $x + y - z \in X$, then we cannot have $r(z) = 0$, since y is an extreme point of Y. But then since $r(x+y) = r(x)$ maximizes r, we cannot have both $x+y+z$ and $x + y - z$ be elements of X, because either $r(x + y + z)$ and $r(x + y - z)$ is greater than $r(x + y) = r(x)$. □

Note that the last theorem is not true if we do not assume $x_i \geqslant 0$ for all i, which is why we prefer discussing linear programs in standard form. Given m constraints on an n-dimensional linear program, there can only be $\binom{m}{n}$ extreme points on the solution space, because a point is uniquely specified by its particular value on an invertible matrix, and every extreme point is determined by its values on some invertible submatrix of the constraint matrix. This gives us a particular algorithm for finding the solution to a linear program – just find the value of the function at the extreme points. Unfortunately, this method can be exponential in the description of the problem. However, we can represent these points in polynomial time.

Theorem 1.3. *If a polyhedron is defined by an inequality of the form $Ax \leqslant b$, where $A \in M_{n,m}(\mathbf{Q})$, and all coefficients involved can be expressed in K bits, then the extreme points of the polyhedron can be expressed in coefficients using only $2m\log(m)K$ bits.*

Proof. Each extreme point x can be defined as the unique solution to $Bx = c$, where B is an invertible $m \times m$ submatrix of A. Then, using Cramer's rule, we may write

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i is obtained from A by replacing the i'th column of A with c. Now the determinant can be expressed as

$$\det(X) = \sum_{\sigma \in S_m} \operatorname{sgn}(\sigma) \prod_{i=1}^{m} X_{i\sigma(i)}$$

The product of m numbers representable in K bits can be written in mK bits, and the sum of these numbers can be expressed in $m\log(m)K$ bits. Similarily, the number of bits to express a/b is bounded by the sum of the number of bits to express a and b. Thus each coefficient of x can be expressed in $2m\log(m)K$ bits. □

Since A and c take $(n+1)mK$ bits to describe, this gives us a polynomial bound in the size of solutions. Thus, if we can choose extreme points wisely, then we may well achieve a polynomial solution.

Theorem 1.4. *A feasible, bounded linear program has an optimal solution.*

Proof. If X is the feasible solution space of a linear program, then it is closed and convex. Thus $r(X)$ is also closed and convex. If we are working over $\mathbf{R}$, we may apply completeness and conclude that $r(X) = (-\infty, b]$ or $r(X) = [a, b]$. In either case, we conclude there is $v \in X$ for which $r(x) = b$ maximizes the functional. To obtain results in $\mathbf{Q}$, we first maximize over $\mathbf{R}$, and then justify why we can obtain estimates over $\mathbf{Q}$. The last lemma implies that if $x \in \mathbf{R}^n$ maximizes $r(x)$, and we take x to be an extreme point, then A^x has full rank. Since $A^x x = b^x$, we may throw away rows from A^x to make an invertible matrix, and then express x as the product of b^x with an inverse, which has rational entries, and thus $x \in \mathbf{Q}^n$. □

The main goal of this report is to find ways of optimizing linear programs in an algorithmically simple way. In the combinatorial method (where we maximize over the finite number of extreme points) the linear optimization problem is often easier to solve if almost all constraints are equalities. That is, a linear program is in **slack form** if we attempt to maximize $c(v)$ such that $Av = b$, and $v \geqslant 0$. We can make an inequality of the form $\lambda(v) \leqslant b$ into an equality by introducing a new variable x, defining $\lambda(v, x) = \lambda(v) + x$, and enforcing that $\lambda(v, x) = b$, and $x \geqslant 0$. This only increases the dimensionality of the problem by a linear amount in the constraints of the problem.

We finish this chapter by discussing some applications of linear programming, mainly to combinatorial problems in computing science (since they are the most interesting).

Example. *The classical computing science problem to be explained in linear programming is the max flow problem. Given a directed graph, where every edge e has a certain capacity c, we want to maximize the amount of 'flow'*

between two vertices s and t, such that flow conservation is preserved at every vertex other than s and t. That is, we wish to find a function $f : E \to \mathbf{R}^+$, such that

$$\sum_{sv \in E} f(sv)$$

is maximized, and for any vertex $v \neq s, t$ in the graph,

$$\sum_{wv \in E} f(wv) = \sum_{vw \in E} f(wv)$$

which is the flow conservation law. After the formalization of linear programming, we see this is just a linear constraint problem over $\mathbf{R}^E$.

Example. *The shortest path problem on an undirected graph with edge costs $c : E \to \mathbf{R}^+$ asks us to find the path between two fixed vertices s and t which minimizes the sum of the costs of the edges in the path. To obtain a linear program describing the program, we would like to associate each subset of edges as an element $f \in \{0,1\}^E$. The only problem with this is that no system of linear constraints can enforce that $f(e) \in \{0,1\}$ for all e, because the set formed is not convex. The problem is we are discussing an* **integer linear program**, *where we attempt to optimize over a $\mathbf{Z}$-module rather than an $\mathbf{R}$-vector space (a mathematically well posed problem, though normally exponentially more complex than a normal linear program, though this problem, an exception of the rule, can be computed using linear programming in subexponential time). In addition to the 0/1 constraint, we desire that for each vertex v,*

$$\sum_{wv \in E} f(wv) \leqslant 1 \qquad \sum_{vw \in E} f(vw) - \sum_{wv \in E} f(wv) = 0 \qquad \sum_{vw \in E} f(vw) \leqslant 1$$

and

$$\sum_{ws \in E} f(ws) = \sum_{tw \in E} f(tw) = 1$$

Once the constraints are specified, we just need to minimize $\sum c(e) f(e)$.

Example. *Suppose that we are classifying objects into two categories, based on information contained in a feature vectors $v_1, \ldots, v_n$. To prevent overfitting, we attempt to find a linear classifier – an linear functional λ together with a scalar b such that the object corresponding to v_i is in category A if $\lambda(v_i) \geqslant b$, and in category B if $\lambda(v_i) < b$. These constraints describe exactly a convex region corresponding to a linear problem, so this problem can be reduced to a finding a* **feasible solution** *with respect to the linear constraints.*

Example. *Consider a set X described by constraints of the form $\langle y_i, x\rangle \leqslant b_i$ for some $y_i, x \in \mathbf{R}^n$ and $b_i \in \mathbf{R}$. The* **Chebyshev center** *is the largest circle contained within X. A circle can be specified by a center x and a radius r. We of course require that $r > 0$. To verify the circle is contained within X, we require $\langle y_i, x + tv\rangle \leqslant b_i$ is true for all $t \leqslant r$ and $\|v\|^2 = 1$. This can be reexpressed as $t\langle y_i, v\rangle \leqslant b_i - \langle y_i, x\rangle$. Since $t\langle y_i, v\rangle \leqslant t\|y_i\|$. The Cauchy Schwarz gives us bounds of the form $\langle y_i, tv\rangle \leqslant t\|y_i\|$, and this bound is tight since we can choose v to be a scalar multiple of y_i. Thus $B_r(x) \subset X$ if and only if $\langle y_i, x\rangle \leqslant b - \|y_i\|$ for each i. This gives us a linear constraint problem to find the Chebyshev center.*

1.1 Piecewise Linear Convex Functions

A function f is **convex** if $f(\lambda a + (1-\lambda)b) \leqslant \lambda f(a) + (1-\lambda)f(b)$ for every a, b in the domain of f. All linear functions are convex (as well as linear functions plus a constant, known as affine functions), but not all convex functions are linear (for instance, the exponential function is convex, and any differentiable function which satisfies $f'' \geqslant 0$). If $f_1, \ldots, f_n$ are convex functions, then $\max(f_1, \ldots, f_n)$ is also a convex function. Thus the maximum of affine functions is convex. This tells us that the absolute value function $|x| = \max(x, -x)$ is convex. A more general optimization problem, given affine functionals $\lambda_i(\cdot) + a_i$, would be to maximize $\max(\lambda_i(v) + a_i)$ subject to some constraint function $Av \leqslant b$. This turns out to be no more general than linear programming. Note that $\max(\lambda_i(v) + a_i)$ is equal to the minimum value of z such that $z \geqslant \lambda_i(v) + a_i$ for all i. Thus we just need to add a linear number of constraints of this form, in addition to the new variable z, and then mimimize the value of z over the constraint region.

What about if we are just discussing absolute value. For example, consider the problem of maximizing $\sum c_i|x_i|$, such that $Ax \geqslant b$, and all c_i are non-negative. We could perform the trick in the last problem, but it is quite involved. Instead, we note that $|x_i|$ is the smallest value of z_i such that $x_i \leqslant z_i$ and $-x_i \leqslant z_i$. Thus we are minimizing $\sum c_i z_i$, subject to these new constraints. Alternatively, we replace each x_i with non-negative values x_i^+ and x_i^- for each x_i, such that $x_i = x_i^+ - x_i^-$.

Example. *Consider the problem of minimizing $2|x| + |y|$, such that $x + y \geqslant 4$. We introduce new variables a and b, and minimize $2a + b$, such that $-x, x \leqslant a$, and $-y, y \leqslant b$. The solution is seen to be unbounded, because we can set $x = 4 - y$, and then as we let $x \to \infty$, $y \to -\infty$ and so $2|x| + |y| \to \infty$. An alternate*

and equivalent linear program to optimize is minimize $2x_+ + 2x_- + y_- + y_-$, *such that* $x_+ - x_- + y_+ + y_- \geqslant 4$, $x_+, x_-, y_+, y_- \geqslant 0$.

Example. *Returning to linear optimization, we may have to perform a linear regression on a certain regression sample* $(x_1, y_1), \ldots, (x_n, y_n)$, *with* $x_i \in \mathbf{R}^n$ *and* $y_i \in \mathbf{R}^m$. *We wish to find a linear relation of the form* $y = Mx + b$, *which fits the data best. For each data point* (x_i, y_i), *we consider the prediction error* $|y_i - (Mx_i + b)|$. *The total error is the sum of these errors*

$$\sum_i |y_i - (Mx_i + b)|$$

We have discussed how to formulate this problem as a linear program. The quadratic error $\sum_i |y_i - (Mx_i + b)|^2$ *can also be discussed, which can be solved by methods of elementary calculus.*

Example. *Consider a linear discrete dynamical system defined by*

$$x_{n+1} = Ax_n + Bz_n \qquad y_n = c_n x_n$$

Where $x_n \in \mathbf{R}^n$ *is the state of the system,* $y_n \in \mathbf{R}^n$ *is the output, and* x' *is a sequence of values we get to choose to control how the system evolves. We may enforce that* $Cx_n \leqslant a$ *for some constraint matrix* C. *Given initial conditions* $x_0^i = a_i$, *we may want to drive the state of the system to a state* $b \in \mathbf{R}^n$ *at time* N. *If this is possible, we might want to minimize the total pertubation of the system. This turns into an optimization problem, trying to minimize* $\sum |c_n x_n|$ *subject to the constraints described above.*

Chapter 2

Polyhedra

We shall now describe a more optimal way to check the extreme points of a polyhedron against a constraint matrix, to find optimal solutions. Given two extreme points on a polyhedron, we say they are adjacent

Definition. Let v and w be two basic solutions in a polyhedra. We say the two vertices are adjacent if we can find $n-1$ linearly independent constraints that are active at both of them. The line segment joining the two vertices is called an edge.

We now focus on polyhedra in slack form, that is, vectors such that $Mv = b$. How do we find basic solutions in this polyhedra? Choose n linearly independant rows in M. Find solutions to the vector $Mv = b$, where $v_i = 0$ for a row i not in the n linearly independant columns. The elements forced to be zero are non-basic, whereas the others are basic. Any vector solution is a basic solution.

Definition. A polyhedron contains a line if there is a vector v and a non-zero vector w such that the polyhedra contains $v + \lambda w$ for all scalar values λ.

Theorem 2.1. *The following are equivalent:*

1. *The polyhedron does not contain a line*
2. *The polyhedron has at least one extreme point.*
3. *There are n rows in the matrix defining the polyhedron that are linearly independant.*

Proof. We prove the implications in the order they appear:

- $(1) \implies (2)$: We prove that if a polyhedron does not contain a line, it has a basic feasible solution, and therefore must have an extreme point. Let x be a point in the polyhedron...

□

Chapter 3

Duality

The method of Lagrangian multipliers allows us to find the optima of a function f over the level set of some function $g : \mathbf{R}^n \to \mathbf{R}$ – i.e. $g(x) = c$. For instance, we may wish to find the maximum x^2+y^2, such that $x+y = 1$. If we could find a maximum over all of $\mathbf{R}^n$, and it happened to lie on the level set of g, then we'd be done, but this almost never happens. However, suppose we can modify f to a new function f_1, such that it takes on the same values on the level set of g. Then, if the maximum of f_1 lies on the level set of g, we'd be done, since this would also be the maximum of f on the level set of g. The easiest method is to define $f_1(x) = f(x)+p(g(x)-c)$, for some $p \in \mathbf{R}$. We then express the values of $\partial_i f_1(x) = 0$ as functions of p, and then solve for p to guarantee these solutions lie on $g(x) = c$. For instance, we take $f_1(x,y) = x^2 + y^2 + p(x + y - 1)$. Then

$$\partial_x f_1(x,y) = 2x + p \qquad \partial_y f_1(x,y) = 2y + p$$

so the optima here occur at $x = y = -p/2$. If we want $x + y = 1$, then we must have $p = -1$, and then we find the maximum occurs at $(1/2, 1/2)$, with maximum value $1/2$.

Let us apply a similar method to linear programming. Consider a problem of maximizing $c^t x$, such that $Ax \leqslant b$, and $x \geqslant 0$. We call this the **primal** linear program. We switch to the dual space, and attempt to find y which minimizes $b^t y$, such that $A^t y \geqslant c$, and $y \geqslant 0$. This is the **dual** linear program. Note that this really reflects a duality, because the double dual of a linear program is the original linear program.

Theorem 3.1 (Weak Duality)**.** *Any solution satisfying the dual of a linear program bounds the solutions of the original program.*

Proof. Note that if x and y are as in previous notation,

$$c^t x \leqslant y^t A x \leqslant y^t b$$

The first part follows because $x \geqslant 0$, and the last because $y \geqslant 0$. □

We can actually prove that the optimum value of the dual is the same as the optimum value of the original program. The proof gives us much more about the relationship between the two programs. First, a fundamental theorem from the theory of convex bodies, which emerges from the Hilbert space theory of finite dimensional vector spaces.

Theorem 3.2. *If $X \subset \mathbf{R}^n$ is closed and convex, and $y \notin X$, then there is an affine functional λ such that $\lambda(X) < 0$, and $\lambda(y) > 0$.*

Proof. Let x be the point in X that is closest to y. Then if $x' \in X$,

$$\begin{aligned}\langle y - x, x' \rangle &= \langle y - x, x \rangle + \|y - x\|^2 - \langle y - x, y - x' \rangle \\ &\leqslant \langle y - x, x \rangle + \|y - x\| \left(\|y - x\| - \|y - x'\| \right) \\ &\leqslant \langle y - x, x \rangle\end{aligned}$$

and

$$\langle y - x, y \rangle = \|y - x\|^2 + \langle y - x, x \rangle > \langle y - x, x \rangle$$

Hence $a \mapsto \langle y - x, a \rangle - \langle y - x, x \rangle - \varepsilon$ is the affine functional we need, for small enough ε. □

Geometrically, this theorem says there is a hyperplane separating y and X, not containing y or any points in X, since a hyperplane is just a level set of a linear functional.

A **cone** is a convex subset of $\mathbf{R}^n$ closed under multiplication by non-negative scalars. In the two dimensional case, we obtain the points between some angle, hence the geometric name.

Theorem 3.3. *If X is a non-empty closed cone, an $y \notin C$ there is linear functional λ such that $\lambda(X) \geqslant 0$, and $\lambda(y) < 0$.*

Proof. Using the last theorem, find an affine function μ such that $\mu(X) > 0$ and $\mu(y) < 0$. If we define $\lambda = \mu - \mu(0)$, then λ is linear, and if $\lambda(y) > 0$, then we find $\lambda(y) > \mu(0) > 0$, which is impossible. □

Theorem 3.4 (Farkas)**.** *If $A \in M_{n,m}(\mathbf{R})$, $b \in \mathbf{R}^m$, and $c \in \mathbf{R}^n$. Then either there is $x \geqslant 0$ such that $Ax \leqslant b$, $y \geqslant 0$ such that $A^t y \geqslant 0$ and $b^t y < 0$, but not both.*

Proof. Suppose that both x and y existed. Then we find

$$0 \leqslant y^t(Ax) \leqslant y^t b < 0$$

We now show that one of the two conditions must hold. □

Farkas' lemma gives us strong duality in linear programs, for it relates the slack in the dual program to slack in the original program.

Theorem 3.5 (Strong duality)**.** *The optimal value of the dual of a linear program is equal to the optimal value of the original program.*

Proof. Let y^* be the optimal value of the dual of the linear program which maximizes $\lambda(x)$, such that $Ax \leqslant b$, and let $v^* = b^t y^*$. We apply Farka's lemma to conclude that there is either $x \geqslant 0$ such that $-\lambda(x) \leqslant -v^*$ and $Ax \leqslant b$, which implies that $\lambda(x) \geqslant v^*$ (which would imply equality since we have already proven that $\lambda(x) \leqslant v^*$), or there is $(y_0, y) \geqslant 0$ such that $\langle b, y\rangle < \langle v^*, y_0\rangle$, and if $M = \begin{pmatrix} -c^t \\ A \end{pmatrix}$, then $M^t(y_0, y) \geqslant 0$. If $y_0 = 0$, then $A^t y \geqslant 0$ and $b^t y < 0$, so $A^t(y^* + y) \geqslant c$ and $b^t(y^* + y) < v^*$, contradicting optimality. Similarily, if $y_0 > 0$, then we obtain the same conclusion by some tricky manipulation. This completes the proof. □

We finalize our discussion of duality by talking about complementary slackness. Let x be a solution of a linear program, and y a solution of its dual. Then x and y are both optimal if for all $1 \leqslant j \leqslant n$, $x_j(A^t)_j y - c_j = 0$, and for all $1 \leqslant i \leqslant m$, $y_i(A_i x - b_i) = 0$. To see this, we recall the proof of weak duality, and calculate

$$\sum_j c_j x_j \leqslant \sum_j \sum_i (A^t)_{j,i} y_i x_j = \sum_i \left(\sum_j A_{i,j} x_j \right) y_i \leqslant \sum_i b_i y_i$$

and the last theorem shows the inequalities are equalities if and only if the condition holds.

Chapter 4

Applications to Combinatorial Optimization

Unfortunately, we normally have to perform combinatorial problems over $\mathbf{Z}$-modules, a much more intractable problem than over real vector spaces. However, there is a trick, known as unimodularity, which enables us to optimize over the real vector space corresponding to the $\mathbf{Z}$-module, and then recover results in the original space. Call a matrix unimodular if the determinant of every square submatrix is 0, 1, or -1.

Theorem 4.1. *If a polyhedron is expressed by $Ax \leqslant b$ where A is totally unimodular, and $b \in \mathbf{Z}^n$, then every extreme point of the polyhedron is integral.*

Proof. We apply Cramer's rule. If we take a submatrix equality $Bx = c$ to find an extreme point x, then

$$x_i = \frac{\det B_i}{\det B}$$

Then $\det B \in \{-1, 0, 1\}$, and $\det B_i \in \mathbf{Z}$, so that $x \in \mathbf{Z}^n$. □

Thus we may apply linear programming techniques to integer linear programs, provided they are specified by a unimodular constraint.

Example. *Let $G = (L \cup R, E)$ be a Bipartite graph. The problem of finding a maximal matching in this graph can be reduced to integer linear programming. Let $V = \mathbf{Z} \cdot E$ be the free module generated by the edges. Consider the linear function λ_v which assigns a value of one to edges containing v as an endpoint,*

and assigning zero to other edges. We can also consider the functionals μ_e which maps e to one, and other edges to zero. The bipartite matching problem can then be expressed as maximizing $\sum_e \mu_e(w)$ over V, such that $0 \leqslant \mu_e(w) \leqslant 1$ for all e, and $\lambda_v(w) \leqslant 1$. It turns out that the matrix constraints related to this problem are unimodular, so we may solve the problem using non-integer linear programming in $\mathbf{R} \cdot E$, and then project results back into $\mathbf{Z} \cdot E$, which makes the problem tractable.

Example. *The maximum $s-t$ flow problem for a graph (V,E) with capacities c can be represented as a linear program over $\mathbf{R} \cdot E$. We attempt to maximize the operator $\mu_{out} - \mu_{in}$, where $\mu_{out} = \lambda_t$ and $\mu_{in} = \lambda_s$, using the notation of the last example. The constraints are that $0 \leqslant \mu_e \leqslant c(e)$. The constraint matrix is not necessarily unimodular, but here we allow real values, so we can solve the problem in polynomial time using linear programming.*

A **network matrix** is a matrix corresponding to a tree (V,E), with edges oriented arbitrarily (and non-empty). Given a pair of vectors (v,w), and an edge e, we define $A_{(v,w),e} = 1$ if e appears on the $v-w$ path positively, -1 if it appears negatively, and zero if it doesn't occur on the path. More generally, we can consider any submatrix of a matrix of this form to be a network matrix, for we only need to discuss these matrices to prove unimodularity.

Theorem 4.2. *Every network matrix is totally unimodular.*

Proof. We prove by induction. If we only have one edge, then our network matrix is a column of elements in $\{0,1,-1\}$, and is therefore unimodular. Inductively, suppose a network matrix A is presented by (V,E), and a network matrix corresponding to fewer nodes is unimodular. Let B be a submatrix. We may assume E contains more than one edge, because otherwise the matrix is just a columns of elements of $\{0,1,-1\}$, and is therefore unimodular. Fix some edge $vw \in E$. If the row corresponding to vw and wv does not lie in B, then B is a submatrix of the network matrix corresponding to $(V,E-e)$, and is unimodular by induction. We may without loss of generality assume vw lies in B. If there is no ur such that vw is on the path from u to r, then the row corresponding to vw is equal to zero, and $\det(B) = 0$. Thus we may assume ur exists. By negating columns, we may assume that for any such ur vw occurs positively. If we subtract the column corresponding to ur with every other column where vw occurs, we obtain a new matrix B' with the same determinant as B. What's

more, the row corresponding to vw has value one only on the column corresponding to ur, so we may consider the submatrix C of B' obtained from removing the row vw and column ur, and C has the same determinant of B', thought perhaps multiplied by -1. Now we may apply induction to C, since it doesn't contain vw or wv, to conclude that the determinant of B is in $\{0,-1,1\}$. □

It is very useful in the intersection of linear programming and combinatorial optimization to be able to link intuitions about the specific problems at hand (in this case graph theory) with the intuitions about linearity from in the programming itself.

Example. *The Bipartite problem has unimodular constraints. To see this, we need only note that each row of the constraint matrix corresponds to a row of the network matrix corresponding to some tree in $G = (L \cup R, E)$. To prove this, we may assume none of the rows corresponding to the constraints $0 \leqslant \mu_e(w) \leqslant 1$ are in the matrix, since the row is just a single value of 1. Thus we take n vertices $v_1, \ldots, v_n$ and consider certain columns corresponding to edges $e_1, \ldots, e_n$. If the edges have a cycle $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$, then by summing the corresponding columns by alternately adding and subtracting the columns, we end up with a zero vector,b and so the determinant of the submatrix is zero. Thus the*

The Gouila-Houri theorem is also useful for verifying unimodularity.

Theorem 4.3. *A matrix $A \in M_{m,n}(\mathbf{R})$ is unimodular if and only if for every subset of rows can be partitioned into two subset $v_1, \ldots, v_l$ and $w_1, \ldots, w_k$ such that $\sum v_i - \sum w_i \in \{-1, 0, 1\}^n$.*

Chapter 5

The Ellipsoid Method

Recall that every Ellipsoid in $\mathbf{R}^n$ at the origin is the unit ball relative to some positive definite inner product on $\mathbf{R}^n$. The classification theorem on inner product tells us that we can identify every ellipsoid in $\mathbf{R}^n$ with a positive definite matrix $M > 0$ and $x \in \mathbf{R}^n$. To find a polynomial time algorithm for linear programming, we shall shrink ellipsoids around polyhedra to find optimal points.

Thus, given $M > 0$, $x, a \in \mathbf{R}^n$, we wish to find a new ellipsoid $E(N, y)$, which shrinks around a, such that

$$E(N, y) \supset E(M, x) \cap \{z \in \mathbf{R}^n : a_z \geqslant a_x\}$$

and the volume of $E(N, y)$ is less than or equal to $\sqrt{\alpha}$ times the volume of $E(M, x)$, fo rsmall enough α (we shall find $\alpha = e^{-1/2n}$). Since the determinant measures the volume change of the respective linear transformation, the volume condition algebraically means $\det(N) \leqslant \alpha \det(M)$.

Define

$$b = \frac{Ma}{\|a\|_M} \qquad N = \frac{n^2}{n^2 - 1}(M - \frac{2}{n+1} b \cdot b^t)$$

First, we verify that N is positive definite. Note

$$M - \frac{2}{n+1} bb^t = M - \frac{2}{n+1} \frac{Maa^t M^t}{\|a\|_M^2}$$