Optimization

Jacob Denson

November 7, 2019

Table Of Contents

Chapter 1

Introduction

Many problems in computing science can be phrased as optimization problems, given some specified set of inputs X and some function $f : X \to Y$, where Y is given a linear ordering. The elements of X are known as *feasible solutions*. The goal of optimization is to try and find a feasible solution $x \in X$ which *maximizes* or *minimizes* the value $f(x)$, i.e. we wish to compute

$$\operatorname{argmin}_{x \in X} f(x) \quad \text{or} \quad \operatorname{argmax}_{x \in X} f(x).$$

Examples of these problems are finding the 'shortest' path, or the 'longest' common subsequence. If this minimum or maximum does not exist, we say the problem is *unbounded*. In this generality, finding such solutions in a time efficient manner is simply impossible. But in optimization problems, we often assume X, Y, and f have some particular structure which makes the problem possible to solve efficiently.

Part I

Linear Programming

Chapter 2

Linear Programs

Linear Programming attempts to provide a general solution to optimization problems with a 'linear' structure. In particular,

- X is a finite dimensional convex polytope, in the sense that there exists n such that $X \subset \mathbf{R}^n$, and there are finitely many vectors $a_1, \ldots, a_m \in \mathbf{R}^n$ and constants $b_1, \ldots, b_m$ such that

  $$X = \{x \in \mathbf{R}^n : A_i \cdot x \leqslant b_i \text{ for each } i\}.$$

  If we consider the linear map $Ax = (A_1 \cdot x, \ldots, A_m \cdot x)$, then we write the constraints defining X as $Ax \leqslant b$. The vectors A_i are then the rows of A.

- The objective function f is a linear functional, i.e. there exists $c \in \mathbf{R}^n$ such that $f(x) = c \cdot x$.

Here is an example.

Example. Suppose we are stocking items at a cafe. Each item needs to be created from a specific number of ingredients from our supply room. 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. Each apple pie is made from 3 apples, and $1/2$ a bag of sugar. For every 3 chocolate bars, 1 bag of sugar must be used. If we sell an apple pie for 6 dollars, and a chocolate bar for 1 dollar, then our total profit is $6a + b$ dollars. However, we only have 100 apples, and 50 bags of sugar. Thus our choice of a and b must satisfy $3a \leqslant 100$ and $a/2 + c/2 \leqslant 50$.

To summarize, our goal is then to maximize the linear function $f(a,c) = 6a + b$, over the region

$$X = \left\{(a,c) \in \mathbf{R}^2 : \begin{pmatrix} 3 & 0 \\ 1/2 & 1/3 \end{pmatrix} \begin{pmatrix} a \\ c \end{pmatrix} \leqslant \begin{pmatrix} 100 \\ 50 \end{pmatrix}\right\}.$$

If we require the number of apple pies and chocolate bars we make to be integers, then we obtain an *integer linear program*, which is computationally much harder to solve. Often, a good strategy is to solve the linear program, and then round the solution to a set of integers. In this case, we need to round down the values of a and b we get.

Example. Here is a more sophisticated problem. Consider forming an assignment of shifts for a job. On each day i, there is a need for a certain number of people p_i. Each person is assigned a consecutive set of 5 day periods. The goal is then to hire as few people as possible. For each day i, we let x_i denote the number of people starting their shift on day i. For each day i, we then have a constraint that

$$x_i + x_{i-1} + \cdots + x_{i-4} \geqslant p_i,$$

where the indices i are treated modulo 7. Our goal is then to minimize $x_1 + \cdots + x_7$. This, again, is most correctly treated as an integer linear program. But we can find a solution, and then round up.

Example. Suppose we have a directed graph G, with two specified nodes v_1 and v_2. Each edge e has a certain capacity c_e. We image the graph describes a sequence of water pipes, and our goal is to try and push as much water from v_1 to v_2 as possible. We can solve this using linear programming. We have a variable x_e for each edge e. We add the constraint that $0 \leqslant x_e \leqslant c_e$, i.e. we can only push at most the capacity through the pipe. For each vertex $v \notin \{v_1, v_2\}$, we consider a *conservation of flow condition*, i.e. if I_v is the set of edges ending at v, and O_v is the set of edges beginning at v, then $\sum_{e \in I_v} x_e = \sum_{e \in O_v} x_e$. Our goal is then to maximize $\sum_{e \in O_{v_2}} x_e$. This is the *max flow problem*.

2.1 Standard Form

We often choose to work with linear programs in *standard form*. This is a linear program given by a pair (A, b), such that

$$X = \{x \in \mathbf{R}^n : Ax = b, x \geqslant 0\}.$$

Any linear program can be reduced to a linear program of this form by introducing auxilary variables. Given a polytope

$$X = \{x \in \mathbf{R}^n : Ax \leqslant b\},$$

where $Ax = (a_1 \cdot x, \ldots, a_m \cdot x)$, we introduce a sequence of *slack variables* $s_1, \ldots, s_m$, and consider the polytope

$$X' = \left\{(x,s) \in \mathbf{R}^{n+m} : Ax + s = b, s \geqslant 0\right\}.$$

Clearly the problem of optimizing over X is equivalent to optimizing over X'. Given

$$X = \{x \in \mathbf{R}^n : Ax = b\},$$

for each x_i, we introduce two new variables x_i^+, x_i^-, and consider the polytope

$$X' = \left\{(x^+, x^-) \in \mathbf{R}^{2n} : A(x^+ - x^-) = b, x^+, x^- \geqslant 0\right\}.$$

Thus every linear program is *equivalent* to a linear program in standard form, in the sense that for each triple (A, b, c), where A is an $n \times m$ matrix, $b \in \mathbf{R}^m$, and $c \in \mathbf{R}^n$, there exists a triple (A', b', c'), where A is an $n' \times m'$ matrix, $b' \in \mathbf{R}^{m'}$, and $c' \in \mathbf{R}^{n'}$, and there exists a linear transformation $T : \mathbf{R}^n \to \mathbf{R}^{n'}$ such that the image of the polytope $\{x \in \mathbf{R}^n : Ax \leqslant b\}$ under T is $\{x \in \mathbf{R}^{n'} : A'x = b', x \geqslant 0\}$, and moreover, $c \cdot x = c' \cdot T(x)$. Moreover, coming up with the linear transformation T and computing the reduction can all be done in linear time. We also often assume that in standard form, the matrix A has full rank m. We can always assume that the rows of A are linearly independant, since either certain inequalities $A_i \cdot x = b_i$ are redundant, or the problem has no feasible solutions.

Chapter 3

Geometry of Polyhedra

Given an $m \times n$ matrix A and $b \in \mathbf{R}^m$, we define a corresponding *polyhedron*

$$X = \{x \in \mathbf{R}^n : Ax \leqslant b\}$$

We now aim to understand the structure of the polyhedron X. First, note that this region is *convex*, since A is linear. Thus we can now use the terminology of convex geometry to understand X. Recall that $x \in X$ is an *extreme point* if there are $x_1, x_2 \in X$ and $\lambda \in [0,1]$ such that $x = \lambda x_1 + (1-\lambda)x_2$, then $x_1 = x_2 = x$. Our first goal is to show that X has finitely many extreme points. Given $x \in X$, we say the ith constraint is *binding*, or *tight*, if $A_i \cdot x = b_i$, where A_i is the ith row of A.

Given a particular ordered collection of indices $I = (i_1, \ldots, i_k) \subset \{1, \ldots, m\}^k$, we let A_I be the $k \times n$ matrix whose jth row is A_{i_j}. Similarily, if $x \in \mathbf{R}^n$, and $I = (i_1, \ldots, i_k) \subset \{1, \ldots, n\}^k$ then we let $x_I = (x_{i_1}, \ldots, x_{i_k})$.

Lemma 3.1. *Fix $x \in X$. Let $I = \{i \in \{1, \ldots, m\} : A_i x = b_i\}$. Then x is an extreme point if and only if $A_I : \mathbf{R}^I \to \mathbf{R}^m$ is injective.*

Proof. Let $x_1, x_2 \in X$ and $\lambda \in [0,1]$ be such that $x = \lambda x_1 + (1-\lambda)x_2$. If $A_i \cdot x = b_i$, then this forces $A_i \cdot x_1 = A_i \cdot x_2 = b_i$. But this means that

$$A_I x = A_I x_1 = A_I x_2$$

so $x = x_1 = x_2$.

Conversely, if A_I is not injective, there is a vector v such that $A_I v = 0$. But this means that $A_I(x \pm \varepsilon v) = b_I$. If $i \notin I$, then $A_i x < b_i$, so if ε is small enough, $A_i(x \pm \varepsilon v) < b_i$ as well. Thus $x \pm \varepsilon v \in X$, which shows x is not an extreme point of V. □

Corollary 3.2. *Every polyhedron has finitely many extreme points.*

Proof. If x is an extreme point, there are a collection of indices I such that $A_I x = b_I$. But since A_I is injective, this uniquely specifies x. Since there are only finitely many choices of index sets I, there can only be finitely many extreme points. □

If $x = \lambda x_1 + (1-\lambda)x_2$, then $c \cdot x \leqslant \max(c \cdot x_1, c \cdot x_2)$. In particular, if X has an extreme point, then a maximum solution to a linear program can *always* be chosen to be an extreme point. This gives a method of solving linear programs. Just check the values at all extreme points and take the maximum value. However, this is horribly inefficient, since in n dimensional space, there can be at least $\Theta(m^n)$ extreme points (for instance, the unit cube in $\mathbf{R}^n$ has 2^n extreme points).

Given two extreme points x_1 and x_2, we say they are *adjacent* if there are $n-1$ linearly independant constraints A_I such that $A_I x_1 = A_I x_2 = b_I$. The line segment which connects x_1 and x_2 is then known as an *edge* of X. An intuitive way to search through the class of extreme points is to move along adjacent extreme points, in such a way that the solution is eventually found.

3.1 Polyhedra in Standard Form

Our analysis is often simplified by considering a polyhedron in standard form, i.e. by the equation

$$X = \{Ax = b, x \geqslant 0\},$$

where A is a $m \times n$ matrix with rank m. Let the columns of A be denoted by $A^1, \ldots, A^n \in \mathbf{R}^m$. Given $I = (i_1, \ldots, i_k) \subset \{1, \ldots, m\}^k$, we let A^I be the $m \times k$ matrix $A^I = (A^{i_1}, \ldots, A^{i_k})$.

Lemma 3.3. *A point $x \in X$ is an extreme point if and only if, there exists a collection of m indices I such that if $i \notin I$, $x_i = 0$, and A^I is surjective.*

Proof. Assume A^I is surjective. Then let $x = \lambda x_1 + (1-\lambda)x_2$ with $x_1, x_2 \in X$ and $\lambda \in [0,1]$. If $x_i = 0$, then $(x_1)_i = (x_2)_i = 0$. But then x, x_1, and x_2 all lie in a common m dimensional subspace

$$V = \{v \in \mathbf{R}^n : v_i = 0 \text{ if } i \notin I\}.$$

Now $Av = A^I v$ for each $v \in V$. Since A^I is surjective, then since V and $\mathbf{R}^m$ have the same dimension, this means A is actually a bijection restricted to V. Since $Ax = Ax_1 = Ax_2 = b$, this means $x = x_1 = x_2$. Thus x is an extreme point of X.

Conversely, suppose x is an extreme point of X, and let $I = (i_1, \ldots, i_k)$ be the set of all indices such that $x_{i_k} = 0$. Then Lemma 3.1 implies that if K is the kernel of A, and

$$V = \{v \in \mathbf{R}^n : v_I = 0\},$$

then $V \cap K = (0)$. Thus A has rank m when restricted to V. Lemma 3.1 implies that there must be at least $n - m$ indices i such that $x_i = 0$, so V has dimension at most m. But since A has rank m, this implies that A is invertible when restricted to V, which gives the column condition. □

Thus every extreme point in X corresponds to a basis of $\mathbf{R}^m$ generated by columns of A. Given a particular index set $I = \{i_1, \ldots, i_m\}$ such that A^I is invertible, and the unique point $x \in X$ corresponds to this basis, given by setting $x_i = 0$ if $i \notin I$, and $x_I = (A_I)^{-1} b_I$. Note also that adjacent extreme points in X correspond to index sets which differ by only a single index. But we need not have $x \geqslant 0$. We refer to x as a *basic solution*, and if $x \geqslant 0$, a *basic feasible solution*.

3.2 Degeneracy

An extreme point x to the set $X = \{x \in \mathbf{R}^n : Ax \leqslant b\}$ is called *degenerate* if more than n indices are active at x. For instance, in two dimensions, an extreme point is degenerate if it lies at the intersection of three or more edges of the polyhedron. If we instead deal with a polyhedron in standard form, i.e. $X = \{x \in \mathbf{R}^n : Ax = b, x \geqslant 0\}$, then an extreme point is degenerate if more than $n - m$ components of x are equal to zero.

It is possible for an extreme point to be degenerate with respect to one pair (A, b) such that $X = \{x \in \mathbf{R}^n : Ax \leqslant b\}$, but not degenerate with respect to another pair. However, if we work in standard form, then degeneracy *is* a property of the geometric set, and does not really depend on A and b.

3.3 Existence of Extreme Points

We now show extreme points exist for most polyhedra.

Lemma 3.4. *If*

$$X = \{x \in \mathbf{R}^n : Ax \leqslant b\},$$

then extreme points exist if and only if A is injective.

Proof. Consider the polyhedron $X = \{x \in \mathbf{R}^n : Ax \leqslant b\}$. If A is not injective, then there is v such that $Av = 0$. Then for any point x, $x + v, x - v \in X$, so X does not have any extreme points. However, if A is injective, and X is non-empty, then given $x \in X$, we consider the index set $I = \{i : A_i x = b_i\}$. If $\{A_i : i \in I\}$ spans $\mathbf{R}^n$, then we already know x is an extreme point. If not, then there exists $v \in \mathbf{R}^n$ such that $A_i v = 0$ for each $i \in I$. Since A is injective, there must be some $\lambda \in \mathbf{R}^n$ such that $x + \lambda v$ violates at least one more constraint j. Thus we can keep finding values which violate more constraints until we find an extreme point. □

Remark. If we are given $x \in X$, then the lemma above shows we can find an extreme point of X in polynomial time.

Note that the above theorem applies to all polyhedra in standard form, since no such polyhedron contains a line.

3.4 Convex Hulls

For a collection of points $X \subset \mathbf{R}^n$, it's *convex hull* $\mathrm{Conv}(X)$ is the smallest convex set containing X. If $X = \{x_1, \ldots, x_N\}$ is finite, then we can write

$$\mathrm{Conv}(X) = \{\lambda_1 x_1 + \cdots + \lambda_N x_N : \lambda_i \in [0,1], \lambda_1 + \cdots + \lambda_N = 1\}.$$

Note that the extreme points in $\mathrm{Conv}(X)$ form a subset of $\{x_1, \ldots, x_N\}$. In fact, one can reduce this process for polyhedra.

Lemma 3.5. *Every bounded polyhedron is the convex hull of it's extreme points.*

TODO: USE DUALITY THEORY PROOF.

3.5 Fourier Motzkin Elimination

The oldest method for solving linear programs in not efficient, but has interesting theoretical consequences. Consider a polyhedron

$$X = \{x \in \mathbf{R}^n : Ax \leqslant b\}.$$

The idea is to eliminate x_n, so we can deal with a lower dimensional polyhedron. The idea here is simple. For each constraint $A_i \cdot x \geqslant b_i$, rewrite the constraints as

$$A_{in}x_n \geqslant b_i - A_i^0 \cdot x^0$$

where $A = (A^0, A^n)$, and $x = (x^0, x_n)$. If we let $d_i = b_i/A_{in}$ and $f_i = A_i^0/A_{in}$, then these equations are equivalent to

$$b_i - a_i^0 \cdot x^0 \leqslant 0 \text{ if } a_{in} = 0$$

$$d_i - f_i \cdot x^0 \leqslant x_n \text{ if } a_{in} > 0$$

$$d_i - f_i \cdot x^0 \geqslant x_n \text{ if } a_{in} < 0.$$

We now consider the polyhedron Y given by the constraints

$$b_i - a_i^0 \cdot x^0 \leqslant 0 \text{ if } a_{in} = 0,$$

$$d_i - f_i \cdot x^0 \leqslant d_j - f_j \text{ if } a_{in} > 0,\ a_{jn} < 0.$$

If $\pi(x_1, \ldots, x_n) = (x_1, \ldots, x_{n-1})$, then $\pi(X) = Y$. A result of this argument is that the projection of a polyhedron is always a polyhedron. Furthermore, for any finite set of points $x_1, \ldots, x_N$, the convex hull of this collection is a polyhedron, since it is the image of

$$\{(\lambda_1, \ldots, \lambda_N) : 0 \leqslant \lambda_i \leqslant 1, \lambda_1 + \cdots + \lambda_N = 1\},$$

under the map $\lambda \mapsto \lambda_1 x_1 + \cdots + \lambda_N x_N$.

Chapter 4

The Simplex Algorithm

We now describe the simplex algorithm, which gives an efficient method to maximize $c \cdot x$ over a polyhedra given in standard form by

$$\{x : Ax = b, x \geqslant 0\},$$

where the rows of A are linearly independant. Our goal is to move between adjacent extreme points in an optimal manner.

Given an index set $I = (i_1, \ldots, i_m)$ which corresponds to a basic solution x with $x_I = (A^I)^{-1}b$, and $x_i = 0$ if $i \notin I$. For each i, we wish to find a direction d such that $d_i = 1$, $A(x + \theta d) = b$, and $d_j = 0$ for $j \notin \{i\} \cup I$. Since $Ax = b$, we must have $Ad = 0$, and so $0 = Ad = A^I d_I + A_i$. Thus $d_I = -(A^I)^{-1}A_i$. We refer to the vector d as the ith basic direction.

For small values of $\theta > 0$, $x + \theta d \in X$, since the only indices j other than i for which the values x_j are changed are those with $x_j > 0$. This shift changes the objective function by $\theta(c \cdot d)$, and therefore increases the objective function if $c \cdot d > 0$. We write $c \cdot d = c_i - c_I \cdot (A^I)^{-1}A_i$. We write this as the ith *reduced objective value*, denoted $\overline{c_i}$.

Lemma 4.1. *If $\overline{c} \leqslant 0$, then x is optimal, and if x is optimal and non-degenerate, $\overline{c} \leqslant 0$.*

Proof. Suppose $\overline{c} \leqslant 0$. Let $y \in X$ be another feasible solution, and let $d = y - x$. Then $Ad = 0$. We can rewrite this as

$$A^I d_I + \sum_{i \notin I} A_i d_i = 0.$$

Thus

$$d_I = -\sum_{i \notin I} (A^I)^{-1} A_i d_i.$$

Thus

$$c \cdot d = c_I \cdot d_I + \sum_{i \notin I} c_i d_i = \sum_{i \notin I} (c_i - (A_I)^{-1} A_i) d_i = \overline{c} \cdot d.$$

Since $x_i = 0$ for any $i \notin I$, and $y_i \geqslant 0$, $d_i \geqslant 0$ for any $i \notin I$. Since $\overline{c} \leqslant 0$, this implies $c \cdot d \leqslant 0$. Thus $c \cdot y \leqslant c \cdot x$.

On the other hand, suppose x is optimal and non-degenerate. Note that $\overline{c}_i = 0$ if $i \in I$. If $\overline{c}_j > 0$ for some j, then by nondegeneracy, $x_j \notin I$. If we let d be the jth direction vector, then $c \cdot d = \overline{c}_j > 0$. If θ is small enough, then $x + \theta d \in X$, and $c \cdot (x + \theta d) = c \cdot x + \theta c \cdot d > c \cdot x$, so x is not optimal. □

An index set I is optimal if $A_I^{-1} b \geqslant 0$, and $\overline{c} \geqslant 0$. The vector $x = (A_I)^{-1} b$ is then an optimal solution to the linear program.

The idea of the simplex algorithm is to *pivot* between basic feasible solutions. To begin with, we assume that all extreme values are non degenerate. Let I be an index set corresponding to a basic feasible solution x. If $\overline{c}_i \leqslant 0$ for all i, then x is optimal. Otherwise, there is some i with $\overline{c}_i > 0$. Consider the ith direction vector d. We then consider

$$x + \theta^* d \in X,$$

where $\theta^* = \max\{\theta : x + \theta d \in X\}$. We note that this is easily calculated, since it is equal to ∞ if $d \geqslant 0$, and otherwise equal to

$$\min(-x_i/d_i : d_i < 0).$$

If $\theta^* = \infty$, then the problem is unbounded. Otherwise, we consider $x + \theta^* d$. There is at least one index $j \in I$ such that $x_j + \theta^* d_j = 0$. If we swap this index j with the index i, then the resulting index set I' still satisfies the assumption that $A^{I'}$ is invertible. This is because if $A^i = \sum_{k=1}^m \lambda_k A^{i_k}$ for some constants λ_k, then $(A^I)^{-1} A^k = \sum_{k=1}^m \lambda_k (A^I)^{-1} A^{i_k}$. But $(A^I)^{-1} A^{i_k} = e_k$, for each $i_k \in I$,so $-d_I = (A^I)^{-1} A^k = \sum_{k=1}^m \lambda_k e_k$. But then $(-d_I)_j = \lambda_j \neq 0$. Thus $A^{I'}$ is still invertible. Thus $x + \theta^* d$ is a basic feasible solution. This is a pivot, a single iteration of the simplex algorithm.

Provided there are no degenerate basic feasible solutions, since a pivot always increases the value of the objective function, we never return to an extreme point, and since there are only finitely many, the program is guaranteed to terminate.

4.1 Simplex for Degenerate Polyhedra

We now consider the case of degenerate polyhedra. A modification of the simplex algorithm that works for this degenerate polyhedra must then take into account the following situation.

- If we choose some direction vector d, and there is a basic variable $i \in I$ such that $x_i = 0$, but $d_i < 0$, then $\theta^* = 0$, and we do not change our solution. Nonetheless, our new basis is obtained by swapping i with the new direction index.
- Even if $\theta^* > 0$, it is possible that more than one basic variable becomes zero at a new point, so that we have a choice of which index we swap out of the basis.

The choices of which index to add into a basis, as well as which value i with $c_i > 0$ to add into the basis, are known as *pivoting rules*. Such a rule must be chosen to avoid *cycling*, which can occur if one runs into the same basis more than once. A simple choice is to always choose the value i with the smallest index.

4.2 Finding an initial feasible solution

Given an initial solution $x \in X$, the last chapter gives ways of finding an extreme point, i.e. a basic feasible solution in X with which to start the simplex algorithm. In general, it is not easy to even find a feasible solution. One technique to finding the solution is to consider another linear program. Consider the region of x values such that $Ax = b$ and $x \geqslant 0$. Without loss of generality, we may assume $b \geqslant 0$. We consider the problem of minimizing $y_1 + \cdots + y_m$, such that $Ax + y = b$, $x \geqslant 0$, and $y \geqslant 0$. For this problem, a feasible solution is given by letting $x = 0$, and $y = b$, which gives a basic feasible solution. Thus we can solve the problem using the simplex method. If the original problem has a feasible solution x, then setting $y = 0$ gives a solution to this problem. Conversely, if the original problem is unfeasible, then the optimal values of this problem will have $y = 0$, and the resulting values of x can be plugged back into the original problem. This is called the two phase method.

Chapter 5

Duality

Consider minimizing $M(x,y) = x^2 + y^2$ subject to the constraint $x + y = 1$. One way to solve this problem is to introduce a Lagrangian multiplier λ, and try and optimize the unrestrained problem of maximizing

$$L(x,y) = x^2 + y^2 + \lambda(1 - x - y)$$

over all values of $x, y \in \mathbf{R}$, which now penalizes having $x + y > 1$ if $\lambda > 0$, and penalizes $x + y < 1$ if $\lambda < 0$. Notice that for each λ,

$$\min_{x+y=1} M(x,y) = \min_{x+y=1} L(x,y) \geqslant \min_{x,y \in \mathbf{R}} L(x,y).$$

The latter quantity is easy to calculate for any particular λ. If the optimum solution to the latter problem actually occured on the line for some particular λ, we'd actually the optimum on the line. And this should occur for some λ, since if λ is large enough, it becomes too expensive to have $x + y > 1$, and if λ is too small, it becomes too expensive to have $x + y < 1$. For each λ, the optimal solution to the unconstrained problem is obtained by taking $x = y = \lambda/2$. If $\lambda = 1$, then the optimal solution is $(1/2, 1/2)$, which is the optimal solution to the original problem.

So now consider the linear programming problem of minimizing $c \cdot x$, such that $Ax = b$, and $x \geqslant 0$ is a column vector in $\mathbf{R}^n$. We now consider the *relaxed* problem where the constraint $Ax = b$ is replaced with a penalty $p(b - Ax)$, where p is a row vector in $\mathbf{R}^n$. We then consider the problem of minimizing $cx + p(b - Ax) = (c - pA)x + pb$, such that $x \geqslant 0$. The optimum value of x here gives an *lower bound* for the original problem, and if p is

chosen correctly, we find it gives the *exact solution* to the original problem. If $c - pA \geqslant 0$, then this minimum is obtained by setting $x = 0$, which gives the minimum as pb. If $c - pA < 0$, then the minimum $-\infty$, so this bound is useless. Thus we are left with the *dual* optimization problem of maximizing pb, such that $pA \leqslant c$.

Notice the dual of a problem in standard form is a problem in the inequality form. Conversely, we can reverse this approach, so that the dual of a problem in inequality form is in standard form. Given the problem of maximizing cx such that $Ax \leqslant b$, we introduce a penalty $p(c - Ax)$, where $p \geqslant 0$ since we don't need to penalize having $Ax \leqslant c$, only $Ax > c$. Thus we want to maximize $cx + p(b - Ax) = (c - pA)x + pb$. If $c - pA \neq 0$, then this problem is unrestrained, and the maximum is equal to ∞. If $c - pA = 0$, the optimal value is pb. Thus the dual problem is now to minimize pb, such that $pA = c$, and $p \geqslant 0$.

Remark. Note that the dual of the dual of a linear program is equal to the original program.

The fact that the values of feasible solutions to the dual problem bound the values of optimal solutions to the original problem is known as *weak duality*. This is not a deep result, but it does imply that if the optimal cost in the original program is $-\infty$, then the dual program must be infeasible, and if the optimal cost in the dual is ∞, then the original problem is infeasible. Furthermore, if x and p are feasible solutions in the original and dual problem, and $cx = pb$, then x and p are both *optimal* solutions. A deeper result is *strong duality*.

Theorem 5.1. *If a linear program has an optimal solution, so does it's dual, and the values of these solutions are equal.*

Proof. Consider a program in standard form, minimizing cx such that $Ax = b$, and $x \geqslant 0$. Consider the simplex program applies to this problem, which by assumption produces an optimal solution x at an extreme point. Let B be the basis at this point. Then the reduced costs in each direction must be non-negative, so we conclude that $c - c_B B^{-1} A \geqslant 0$. Let $p = c_B B^{-1}$. Then $pA \leqslant c$ by definition, so p is a feasible solution to the dual problem. Furthermore, $pb = c_B B^{-1} b = c_B x_B = cx$. Thus p and x are both optimal solutions. □

An important relation between primal and dual programs is *complimentary slackness*.

Theorem 5.2. *Let x and p be feasible solutions to a program and it's dual. These vectors are optimal if and only if $p_i(a_i x + b_i) = 0$ for all i, and $(c_j - pa_j) = 0$.*

Chapter 6

Blah

There are three classes of linear programs:

- It is possible that $X = \varnothing$, in which we call the linear program *unfeasible*.
- It is possible that $\sup_{x \in X} f(x) = \infty$, in which case we call the program *unbounded*.
- If neither of these conditions hold, then $\sup_{x \in X} f(x) < \infty$, and $\sup_{x \in X} f(x)$ is actually attained since X is closed and f is a linear map, hence $f(X)$ is closed.

Most importantly for us, linear programs are *effectively solvable*. In 1947, Dantzig developed the simplex method for solving linear programs, and many more effective techniques have been viewed in the meantime.

It will also often be convenient to work with a linear program in *slack form*. Given a system of inequalities

$$\sum a_{ij} x_j \leqslant b_i \text{ for each } i,$$

we can introduce additional variables $y_1, \ldots, y_m$, assert that $y_i \geqslant 0$ for each i, and enforce $b_i = \sum a_{ij} x_j + y_i$. The constraints $\sum a_{ij} x_j \leqslant b_i$ are then guaranteed by the two equations introduced, so we can remove them. The variables $y_1, \ldots, y_m$ are known as *slack variables*, because they give the amount of 'slack' in the equation $\sum a_{ij} x_j \leqslant b_i$.

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 6.1. *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 6.2. *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 6.3. *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 writted 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 6.4. *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.

6.1 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 7

Simplex Method

We begin with a system of equations in slack form, i.e. we are given an $m \times n$ matrix A and $b \in \mathbf{R}^m$ such that

$$X = \{x \in \mathbf{R}^n : x_1, \ldots, x_n \geqslant 0, Ax = b\},$$

and we wish to maximize $c_1 x_1 + \cdots + c_n x_n$. We solve this using the simplex method.

We apply the *dictionary method* to more conveniently describe the simplex method, as introduced by J.E. Strum. With each $x_0 \in X$, we associate a *dictionary*, which is a pair (A', b') such that $A'x = b'$ if and only if $A'x = b'$ for each $x \in \mathbf{R}^n$.

a system of linear equations

Chapter 8

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 8.1. *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 9

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 9.1 (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 9.2. *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 9.3. *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 9.4 (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 9.5 (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 10

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 10.1. *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 $\mu_{\text{out}} - \mu_{\text{in}}$, where $\mu_{\text{out}} = \lambda_t$ and $\mu_{\text{in}} = \lambda_s$, 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 10.2. *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 10.3. *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 11

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

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

Part II

Convex Optimization

It is often the case that the functions we attempt to maximize are non-linear, or the set we are optimizing over is not a polyhedron. Convex optimization is the safest step to generalizing the linear programming optimization problem, for it is still a semi-tractable problem.

Chapter 12

Convex Functions

Let $D \subset \mathbf{R}^n$ be a convex subset of euclidean space. A function $f : D \to \mathbf{R}$ is **convex** if the graph of f lies below the lines between any two points on the graph. Analytically, this can be expressed by the inequality

$$f(\lambda x + (1 - \lambda y)) \leqslant \lambda f(x) + (1 - \lambda) f(y)$$

Or equivalently, the epigraph $\{(x, y) : f(x) \leqslant y\}$ of the function is convex.

The inner product on $\mathbf{R}^n$ gives us lots of information about the convex subsets of $\mathbf{R}^n$. Indeed, the Banach and Hilbert space structure of $\mathbf{R}^n$ tells us nice separation theorems about convex sets.

Theorem 12.1. *If $D \subset \mathbf{R}^n$ is convex and closed, and $x_0 \in \mathbf{R}^n - D$, then there is $y \in \mathbf{R}^n$ such that for any $x \in D$,*

$$\langle y, x \rangle < \langle y, x_0 \rangle$$

Geometrically, there is a hyperplane separating D and x_0. If $x_0 \in \partial D$, then there is a nonzero $y \in \mathbf{R}^n$ such that for $x \in D$,

$$\langle y, x \rangle \leqslant \langle y, x_0 \rangle$$

So there is a hyperplane supporting the convex set in $\mathbf{R}^n$.

If f is a function defined on a convex set, then v is a **subgradient** of f at x if the hyperplane defined at x by v lies below the graph of f. Analytically, for any y,

$$f(x) - f(y) \geqslant \langle v, x - y \rangle$$

The set of subgradients of f at x is denoted $\partial f(x)$. It is a pleasant way to obtain a form of differentiation of a convex function that need not be convex.

Theorem 12.2. *If a convex f is differentiable at x, then $\partial f(x)$ contains a single element.*

Proof. Write $f(x+h) = f(x) + \langle \nabla f(x), h\rangle + O(\|h\|^2)$, so

$$f(x+h) - f(x) = \langle \nabla f(x), h\rangle + O(\|h\|^2)$$

Now if $v \in \partial f(x)$, this implies that

$$|\langle v - \nabla f(x), h\rangle| = O(\|h\|^2)$$

But this implies $v = \nabla f(x)$, before linear functions cannot approach the origin quadratically. Indeed, find $x_0 \in S^{n-1}$ with

$$|\langle \nabla f(x) - v, x_0\rangle| = \|\nabla f(x) - v\|$$

Then

$$\frac{\langle \nabla f(x) - v, \lambda x_0\rangle}{|\lambda|^2} = \frac{\|\nabla f(x) - v\|}{|\lambda|}$$

which must vanish as $\lambda \to 0$, impossible unless $\|\nabla f(x) - v\| = 0$.

Conversely, we verify that $\nabla f(x)$ is a subgradient for f. We use the convexity of f to write

$$f(y) \geqslant \frac{f((1-\lambda)x + \lambda y) - (1-\lambda)f(x)}{\lambda} = f(x) + \frac{f((1-\lambda)x + \lambda y) - f(x)}{\lambda}$$

By the convexity in one dimension, the values on the right converge monotonically to $\langle \nabla f(x), y\rangle$, and this completes the proof. □

Theorem 12.3. *If $\partial f(x)$ is non-empty for each x, then f is convex. Conversely, if f is convex, $\partial f(x)$ is non-empty on the interior of the domain of f.*

Proof. If $v \in \partial f((1-\lambda)x + \lambda y)$, then

$$f((1-\lambda)x + \lambda y) \leqslant f(x) - \lambda\langle v, y - x\rangle$$

$$f((1-\lambda)x + \lambda y) \leqslant f(y) + (1-\lambda)\langle v, y - x\rangle$$

If we multiply the first equation by λ, the second by $(1-\lambda)$, and then by adding the two equation togther, we obtain the convexity inequality.

Now let $x \in D$. Then $(x, f(x))$ is on the boundary of the epigraph of f, so using the supporting hyperplanes theorem, there is $(a,b) \in \mathbf{R}^n \times \mathbf{R}$ such that for any (y,t), with $t \geqslant f(y)$, with

$$\langle a, x\rangle + bf(x) \leqslant \langle a, y\rangle + bt$$

Letting $t \to \infty$, we conclude $b \leqslant 0$. If $b = 0$, and $x \in D^\circ$, then we can let $y = x - \varepsilon a$, for which we conclude that

$$\langle a, x\rangle \leqslant \langle a, x - \varepsilon a\rangle$$

so that $-\varepsilon\|a\|^2 \geqslant 0$, an impossibility unless $a = 0$, which is impossible due to the separating hyperplanes theorem. Thus $b \neq 0$, and we may divide out, choosing $t = f(y)$, we find

$$f(y) - f(x) \geqslant \langle v/b, y - x\rangle$$

So $v/b \in \partial f(x)$. □

Why are convex functions so interesting in optimization theory? Optimizing an arbitrary general convex function is *not* easy in general, but if we have a pleasant convex domain, the problem becomes much easier. First, note that if we have an arbitrary function $f : D \to \mathbf{R}$, then we can write a minimization problem $\inf_{x\in D} f(x)$ as minimizing the linear function $\inf_{(x,t)\in \mathrm{epi}(f)} t$ over the epigraph of f, so regardless of the qualities of f, we can view a minimization of any function as a minimization of a linear functional on your set. Next, note that minimizing a function over D is the same as minimizing the function over the convex hull of D. However, obtaining a solution on D from a solution over the convex hull of D is normally very difficult, for it involves, given a set $y \in D' \supset D$, finding $x_1, \ldots, x_n \in D$ for which $y = \sum a_i x_i$, with $a_i \geqslant 0$ and $\sum a_i = 1$. This is a very difficult problem algorithmically.

Chapter 13

Mirror Descent

Mirror descent is a way to apply Gradient descent in space with arbitrary norms, that is, on a Banach space rather than a Hilbert space. We cannot apply standard gradient descent to a function f, because for Banach spaces the gradients (Fréchet derivatives) $\nabla f(x)$ lie in the dual space of the Banach space. On a Hilbert space, there is a canonical way of identifying $\nabla f(x)$ with a vector in the original space, so we can consider the iterative procedure $x \mapsto x - \eta \nabla f(x)$. The main idea of mirror descent is to generalize this procedure to Banach spaces is by mapping the Banach space into its dual, finding some way of performing gradient descent in the dual space of the Banach space, and then mapping the space back into the original space, in such a way that we truly obtain a minimizing procedure.

Suppose we are optimizing over a compact, convex subset $D \subset \mathbf{R}^n$ contained in the closure of an open convex subset U, for which $U \cap D$ is nonempty. Now consider a convex function Φ defined on U, which will enable us to map D into the dual space by the map $x \mapsto \nabla\Phi(x)$, the 'mirror map'. Provided that $\nabla\Phi(U) = \mathbf{R}^n$, we can always reverse the mirror rap. If we consider the iteration

$$x \mapsto (\nabla\Phi)^{-1}(\Phi(x) - \eta\nabla f(x))$$

where $(\nabla\Phi)^{-1}(\Phi(x) - \eta\nabla f(x))$ is a point $y \in U$ for which

$$\nabla\Phi(y) = \Phi(x) - \eta\nabla f(x)$$

Now this point may not lie in D, so we define the Bregman divergence

$$D_\Phi(x, y) = \Phi(x) - \Phi(y) - \langle \nabla\Phi(y), x - y \rangle$$

and project y onto a point x' in D which minimizes the divergence $D_\Phi(x', y)$, for $x' \in D \cap U$. This projection exists provided that

$$\lim_{x \to \partial U} \|\nabla \Phi(x)\| = \infty$$

which implies $x \mapsto D_\Phi(x, y)$ is locally increasing on the boundary of U, and if $D_\Phi(x, y)$ is strongly convex, then the existence and uniqueness of a projection is guaranteed (I don't have a proof right now, so let's just assume this is true).

Now we describe mirror descent. Let x_1 minimize $\Phi(x)$ on $U \cap D$. Then, once x_n is specified, let y_n be defined by $\nabla \Phi(y_n) = \nabla \Phi(x_n) - \eta v$, where $v \in \partial f(x_n)$ and $x_{n+1} \in \operatorname{argmin} D_\Phi(x, y_n)$.

Theorem 13.1. *If Φ is strongly convex on $D \cap U$ with respect to the norm $\|\cdot\|$, that is*

$$\Phi(x) - \Phi(y) \leqslant \langle \nabla \Phi(x), x - y \rangle - \frac{m}{2} \|y - x\|^2$$

Define $\lambda^2 = \sup_{x \in U \cap D} \Phi(x) - \Phi(x_1)$, let f be convex, and let L be Lipschitz with respect to $\|\cdot\|$. If we let $\eta = (\lambda / L)\sqrt{2a/K}$, then

$$f\left(\frac{1}{K} \sum_{k=1}^{K} x_k\right) - \min_{x \in D} f(x) \leqslant \lambda L \sqrt{\frac{2}{mK}}$$

Proof. To Be Continued... □