Linear Programming

Jacob Denson

October 14, 2016

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 functional $r \in V^*$ contraint functionals $c_1, \dots, c_n \in V^*$, and a contraint vector $b \in \mathbf{R}^n$. The aim of the linear program is to maximize $r(v)$, subject to the contraints that $c_i(v) \leqslant b_i$. For computational purposes, we can choose a m dimensional basis for V, and specify the problem with a contraint matrix $M \in M_{n,m}(\mathbf{R})$ 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 $Mv \leqslant b$. We also assume the constraints $v_i \geqslant 0$ for purposes of calculation.

Our problem is much more general than upper bounding constraints, for there are ways of formulating more general problems into standard form, in a way in which solutions to the original problem can be recovered. First, if we want a contraint

$$\sum a_i v_j \geqslant b_i$$

Then we just add the contraint

$$\sum (-a_i) v_j \leqslant -b_i$$

If we want an equality contraint

$$\sum a_i v_j = b_i$$

Then we just introduce two constraints

$$\sum a_i v_j \leqslant b_i \qquad \sum a_i v_j \geqslant b_i$$

If we don't have the contraint $v_i \geqslant 0$ for some variable v_i, then we replace v_i with two new variables w_i and w_i', and replace any instance of v_i in the constraints via the equality $v_i = w_i - w_i'$. We can then assume $w_i, w_i' \geqslant 0$.

Without loss of generality, we may consider the problem of linear program as a maximization of $\langle r, x \rangle$. If our real objective is to minimize $\langle r, x \rangle$, this is equivalent to maximizing $\langle -r, x \rangle$. Furthermore, we may also assume each constraint is a less than or equal to relationship, as if we want the constraint $\langle a_i, v \rangle \geqslant b_i$, this is equivalent to saying $\langle -a_i, v \rangle \leqslant -b_i$.

Definition. The set of **feasible solutions** to a linear program is the subset of ^{n} such that, if v is a feasible solution, for each $a_i \in \mathcal{A}$, $\langle a_i, v \rangle \leqslant b_i$. A problem is infeasible if there are no feasible solutions. The optimal solution is the solution v such that $\langle v, r \rangle$ is maximized.

It is notationally simple to introduce an ordering on ^{n}. Say a vector a is less than or equal to a vector b if this is true for all corresponding components. Then a linear program is really just (M, b), where M is an $m \times n$ matrix, and b is a vector in ^{m}. Then the feasible solutions are just the vectors $v \in^n$ such that $Mv \leqslant b$.

It is not clear that an optimal solution to a problem will always exists. We could have infinitely many vectors and thus there is no guarenteed optimum.

Definition. A linear program is unbounded, if, for any $M \in$, there exists a feasible solution v such that $\langle v, r\rangle > M$. It then follows that there is no optimal solution to the program.

The main goal of this report is to find ways of optimizing linear programs in an algorithmically simple way. We introduce another way of formulating a linear program.

Definition. A linear program is in slack form if, instead of the feasible solutions being solutions to inequalities, the feasible solutions are equalities. More specifically, a linear program is in slack form if it is of the form.

$$\begin{array}{ll} \text{maximize} & \zeta(v) \\ \text{subject to} & f_1(v) = b_1 \\ & f_2(v) = b_2 \\ & \vdots \\ & f_n(v) = b_n \\ & v_1, v_2, \dots, v_n \geqslant 0 \end{array}$$

Every linear program can be put into slack form. Given an inequality, $f_i(v) \leqslant b_i$, introduce a slack variable w_i, and replace the inequality with the equality $f_i(v) + w_i = b_i$, and append the equality $w_i \geqslant 0$. Then every solution of the original linear program is a solution to the new linear program for some values of w_i, and has the same reward. If $v_i \geqslant 0$ is not already a constraint, introduce two new variables $v_i' v_i''$, enforce that $v_i', v_i'' \geqslant 0$, and replace every occurence of v_i with $v_i' - v_i''$.

Chapter 2

Polyhedra

Ideas developed in this chapter result from the idea that geometric properties of ^{n} can be used to explore solutions to linear programs. Note that every linear programming can be specified as a shape in ^{n}.

Definition. A polyhedra is a set in ^{n} defined by a matrix M and a real number b by

$$\{v \in^n : Mv \leqslant b\}$$

The constraints of every linear program form a polyhedra, and every polyhedra can be seen as the constraints of a linear program.

Theorem 2.1. *Every polyhedra is convex.*

Proof. Let v and w be vectors such that $Mv \leqslant b$ and $Mw \leqslant b$. Then, for any $\lambda \in [0,1]$, $M(\lambda v + (1-\lambda)w)) = \lambda Mv + (1-\lambda)Mw \leqslant \lambda b + (1-\lambda)b = b$. Thus $\lambda v + (1-\lambda)w$ is a feasible solution, and thus the set is convex. □

Definition. If a vector v in a polyhedra satisfies, for some row M_i in M with corresponding b_i,

$$M_i v = b_i$$

We say the constraint (M_i, b_i) is binding or active at v.

Definition. A vector v is a basic solution of a polyhedra in ^{n} if the binding rows of the constraint matrix M spans ^{n}. A vector is a basic feasible solution if the vector is also a vector such that $Mv \leqslant b$.

Theorem 2.2. *For a vector v, the following are equivalent:*

1. *There exists some vector w such that for any other point u in the polyhedra, $\langle v, w\rangle > \langle v, u\rangle$.*
2. *v is an extreme point of the polyhedra.*
3. *v is a basic feasible solution.*

Proof. We prove the implications in the order given.

- (1) $\implies$ (2): We prove by contraposition. Suppose v is not an extreme point, so that there exists two points w and u in the polyhedra such that for some number λ, $\lambda w + (1-\lambda)u = v$. Let z be an arbitrary vector. Without loss of generality, suppose $\langle w, z\rangle \leqslant \langle u, z\rangle$. Then

$$\begin{aligned}\langle z, v\rangle &= \langle z, \lambda w + (1-\lambda)u\rangle \\ &= \lambda\langle w, z\rangle + (1-\lambda)\langle u, z\rangle \\ &\leqslant \langle w, z\rangle\end{aligned}$$

  Hence the point is not a vertex.

- (2) $\implies$ (3): Suppose v is not a basic feasible solution. Then the set of rows I that are binding span a proper subspace in ^{n}, so we can pick some vector d such that $\langle a_i, d\rangle = 0$ for all a_i. Consider equations of the form

$$v + \varepsilon d$$

  Note for all $a_i \in I$, $\langle v + \varepsilon d, a_i\rangle = \langle v, a_i\rangle = b_i$. For $a_i \notin I$, we know that $\langle a_i, v\rangle < b_i$, and thus for a small enough ε, $\langle a_i, v + \varepsilon d\rangle < b_i$ for all a_i. The $v + \varepsilon d$ is in the set, and by the same argument, $v - \varepsilon d$ is also. We note that v is between these two points, and hence is not an extreme point.

- (3) $\implies$ (1): Let v be a basic feasible solution. Consider the vectors a_i that span ^{n}. We then know $\langle \sum_{i\in I} a_i, v\rangle = \sum_{i\in I} b_i$. For any other value w in the polyhedra, $\langle a_i, w\rangle \leqslant b_i$. Hence $\langle \sum_{i\in I} a_i, w\rangle \leqslant \langle \sum_{i\in I} a_i, v\rangle$, and equality only holds if $\langle a_i, w\rangle = b_i$ for all b_i needed. Since v is basic, and we have n linearly independant a_i, v is the only solution to the system of linear equations above, hence equality only holds for v.

□

Corollary 2.3. *Given a finite number of linear inequalities, there are only a finite number of basic solutions.*

Proof. Consider a system of m linear inequality contraints. At any basic solution, there are n linearly independant action constraints. These system determines a unique point in ^{n}; there can only be one solution to system of equalities $\langle a_i, x\rangle = b_i$. Hence there is at maximum $\binom{m}{n}$ basic solutions. □

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 2.4. *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...

$\square$

Theorem 2.5. *Suppose a linear programming problem has an optimal solution and an extreme point. Then there is an optimal solution that is an extreme point.*

Theorem 2.6. *If a linear programming problem has an extreme point, there is an optimal solution or the problem is unbounded.*

Chapter 3

Duality

Consider the linear programming problem below:

$$\begin{aligned} \text{maximize} \;\; & cx \\ \text{subject to} \;\; & Ax \leqslant b \\ & x \geqslant 0 \end{aligned}$$

Let us call this the primal problem. The dual problem is

$$\begin{aligned} \text{minimize} \;\; & \langle x, b \rangle \\ \text{subject to} \;\; & xA \geqslant c \\ & x \geqslant 0 \end{aligned}$$

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.

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, subject to the contraints that $0 \leqslant \lambda_v(w) \leqslant 1$ and $0 \leqslant \mu_e(w) \leqslant 1$. 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. 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.

Theorem 4.1. *Every network matrix is totally unimodular.*

Proof. We prove by induction. Let $e = uv$ be an edge, where u is a leaf node. We may assume e is directed this way, because swapping the orientation of the edge preserves the unimodularity of the network matrix (it either maintains the determinant of each submatrix, or negates it). Any submatrix of A which does not contain the column corresponding to e is unimodular by induction, since it corresponds to a submatrix in $(V - u, E - e)$.

Thus we may asume the submatrices of B we are analyzing contains the column corresponding to e. If no column of B is for a pari with $s_i = u$, then $\det(B) = 0$. □