Lecture 18

Another max flow application: baseball

Suppose there are n baseball teams, and team 1 is our favorite. It is the middle of baseball season, and some games have been played but not all have. We would like to know: is it possible for the outcome of the remaining games to be such that team 1 ends up winning the season? Or is team 1 basically “drawing dead”, i.e. there is mathematically no way possible for team 1 to win the season? For the purposes of this problem, winning the season means having the most number of wins in the league once all games are complete (if there’s a tie, all tied teams are winners!).

Clearly to make team 1 win the season, we should make it win all its remaining games. But then, what about the other games in which team 1 is not involved? For each of those games, *one* of the two teams involved in that game will be the winner, so who should we make the winner be in each of these games to make sure that no team accumulates so many wins that it surpasses team 1? It turns out that we can determine the feasibility of distributing these wins to make team 1 the season winner by solving the maximum flow in a particular graph and checking whether it’s big enough! For details, see the excellent notes on the subject in Section 24.5 of `http://www.cs.uiuc.edu/˜jeffe/teaching/algorithms/notes/24-maxflowapps.pdf` (if you copy and paste this link, you may find that the tilde before jeffe doesn’t copy properly and you have to fix the tilde manually).

Linear programming: an introductory example

Suppose that a company that produces three products wishes to decide the level of production of each so as to maximize profits. Let x_1 be the amount of Product 1 produced in a month, x_2 that of Product 2, and x_3 that of Product 3. Each unit of Product 1 yields a profit of 100, each unit of Product 2 a profit of 600, and each unit of Product 3 a profit of 1400. There are limitations on x_1, x_2, and x_3 (besides the obvious one, that $x_1, x_2, x_3 \geq 0$). First, x_1 cannot be more than 200, and x_2 cannot be more than 300, presumably because of supply limitations. Also, the sum of the three must be, because of labor constraints, at most 400. Finally, it turns out that Products 2 and 3 use the same piece of equipment, with Product 3 using three times as much, and hence we have another constraint $x_2 + 3x_3 \leq 600$. What are the best levels of production?

We represent the situation by a *linear program*, as follows:

$$\begin{aligned} \max 100x_1 + 600x_2 + 1400x_3 & \\ x_1 &\leq 200 \\ x_2 &\leq 300 \\ x_1 + x_2 + x_3 &\leq 400 \\ x_2 + 3x_3 &\leq 600 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

The set of all *feasible* solutions of this linear program (that is, all vectors in 3-d space that satisfy all constraints) is precisely the polyhedron shown in Figure 18.1.

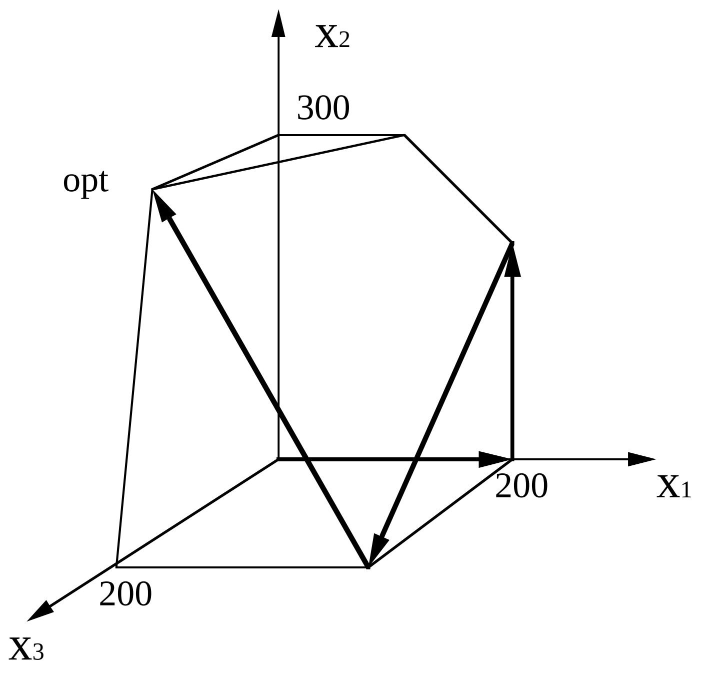

Figure 18.1: The feasible region

We wish to maximize the linear function $100x_1 + 600x_2 + 1400x_3$ over all points of this polyhedron. Geometrically, the linear equation $100x_1 + 600x_2 + 1400x_3 = c$ can be represented by a plane parallel to the one determined by the equation $100x_1 + 600x_2 + 1400x_3 = 0$. This means that we want to find the plane of this type that touches the polyhedron and is as far towards the positive orthant as possible. Obviously, the optimum solution will be a vertex (or the optimum solution will not be unique, but a vertex will do). Of course, two other possibilities with linear programming are that (a) the optimum solution may be infinity, or (b) that there may be no feasible solution at all. For this problem, an optimal solution exists, and moreover we shall show that it is easy to find.

Linear programs

Linear programs, in general, have the following form: there is an *objective function* that one seeks to optimize, along with *constraints* on the variables. The objective function and the constraints are all *linear* in the variables; that is, all equations have no powers of the variables, nor are the variables multiplied together. As we shall see, many problems can be represented by linear programs, and for many problems it is an extremely convenient representation. So once we explain how to solve linear programs, the question then becomes how to reduce other problems to linear programming (LP).

There are polynomial time algorithms for solving linear programs (such as Khachiyan's "ellipsoid algorithm", or an "interior point" algorithm of Karmarkar). In practice, linear programming problems are also often solved by the *simplex method* devised by George Dantzig in 1947. The simplex method actually encompasses a family of algorithms, but we will not discuss them here. We do point out though that for every known implementation of the simplex method, researchers have constructed linear programs which cause them to run for exponential time. Nevertheless, simplex is often used in practice since the bad inputs tend to be contrived, and the algorithm tends to be fast in practice. For the purposes of this course, let's just treat linear programming as a black box which has a polynomial-time solution (polynomial time in the number of variables, number of constraints, and number of bits needed to represent all the coefficients).

Reductions between versions of linear programming

A general linear programming problem may involve constraints that are equalities or inequalities in either direction. Its variables may be nonnegative, or could be unrestricted in sign. And we may be either minimizing or maximizing a linear function. It turns out that we can easily translate any such version to any other. One such canonical translation worth being aware of is the following: *minimization, nonnegative variables, and equality constraints.*

To turn an inequality $\sum a_i x_i \leq b$ into an equality constraint, we introduce a new variable s (the *slack variable* for this inequality), and rewrite this inequality as $\sum a_i x_i + s = b, s \geq 0$. Similarly, any inequality $\sum a_i x_i \geq b$ is rewritten as $\sum a_i x_i - s = b, s \geq 0$; s is now called a *surplus* variable.

We handle an unrestricted variable x as follows: we introduce two nonnegative variables, x^+ and x^-, and replace x by $x^+ - x^-$ everywhere. The idea is that we let $x = x^+ - x^-$, where we may restrict both x^+ and x^- to be nonnegative. This way, x can take on any value, but there are only nonnegative variables.

Finally, to turn a maximization problem into a minimization one, we just multiply the objective function by -1.

A production scheduling example

We have the demand estimates for our product for all months of 1997, $d_i : i = 1, \ldots, 12$, and they are very uneven, ranging from 440 to 920. We currently have 30 employees, each of which produce 20 units of the product each month at a salary of 2,000; we have no stock of the product. How can we handle such fluctuations in demand? Three ways:

- overtime —but this is expensive since it costs 80% more than regular production, and has limitations, as workers can only work 30% overtime.
- hire and fire workers —but hiring costs 320, and firing costs 400.
- store the surplus production —but this costs 8 per item per month

This rather involved problem can be formulated and solved as a linear program. As in all such reductions, the crucial first step is defining the variables:

- Let w_0 be the number of workers we have the ith month —we have $w_0 = 30$.
- Let x_i be the production for month i.
- o_i is the number of items produced by overtime in month i.
- h_i and f_i are the number of workers hired/fired in the beginning of month i.
- s_i is the amount of product stored after the end of month i.

We now must write the constraints:

- $x_i = 20w_i + o_i$ —the amount produced is the one produced by regular production, plus overtime.
- $w_i = w_{i-1} + h_i - f_i, w_i \geq 0$ —the changing number of workers.

- $s_i = s_{i-1} + x_i - d_i \geq 0$ —the amount stored in the end of this month is what we started with, plus the production, minus the demand.
- $o_i \leq 6w_i$ —only 30% overtime.

Finally, what is the objective function? It is

$$\min 2000\sum w_i + 400\sum f_i + 320\sum h_i + 8\sum s_i + 180\sum o_i,$$

where the summations are from $i = 1$ to 12.

A Communication Network Problem

We have a network whose lines have the bandwidth shown in Figure 18.2. We wish to establish three calls: one between A and B (call 1), one between B and C (call 2), and one between A and C (call 3). We must give each call at least 2 units of bandwidth, but possibly more. The link from A to B pays 3 per unit of bandwidth, from B to C pays 2, and from A to C pays 4. Notice that each call can be routed in two ways (the long and the short path), or by a combination (for example, two units of bandwidth via the short route, and three via the long route). Suppose we are a shady network administrator, and our goals is to maximize the network's income (rather than minimize the overall cost). How do we route these calls to maximize the network's income?

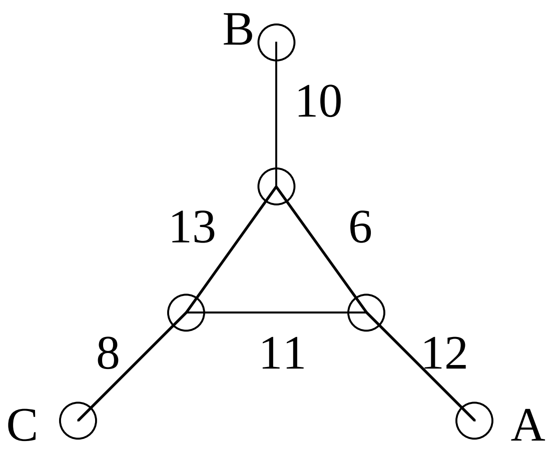

Figure 18.2: A communication network

This is also a linear program. We have variables for each call and each path (long or short); for example x_1 is the short path for call 1, and x'_2 the long path for call 2. We demand that (1) no edge bandwidth is exceeded, and (2) each call gets a bandwidth of 2.

$$\begin{aligned}
\max\ 3x_1+3x_1'+2x_2+2x_2'+4x_3+4x_3' \\
x_1+x_1'+x_2+x_2' &\leq 10 \\
x_1+x_1'+x_3+x_3' &\leq 12 \\
x_2+x_2'+x_3+x_3' &\leq 8 \\
x_1+x_2'+x_3' &\leq 6 \\
x_1'+x_2+x_3' &\leq 13 \\
x_1'+x_2'+x_3 &\leq 11 \\
x_1+x_1' &\geq 2 \\
x_2+x_2' &\geq 2 \\
x_3+x_3' &\geq 2 \\
x_1,x_1'\ldots,x_3' &\geq 0
\end{aligned}$$

The solution, obtained via simplex in a few milliseconds, is the following: $x_1 = 0, x_1' = 7, x_2 = x_2' = 1.5, x_3 = .5, x_3' = 4.5$.

Question: Suppose that we removed the constraints stating that each call should receive at least two units. Would the optimum change?

Approximate Separation

An interesting last application: Suppose that we have two sets of points in the plane, the *black points* $(x_i, y_i) : i = 1,\ldots,m$ and the *white points* $(x_i, y_i) : i = m+1,\ldots,m+n$. We wish to separate them by a straight line $ax+by=c$, so that for all black points $ax+by \leq c$, and for all white points $ax+by \geq c$. In general, this would be impossible. Still, we may want to separate them by a line that minimizes the sum of the "displacement errors" (distance from the boundary) over all misclassified points. Here is the LP that achieves this:

$$\begin{array}{rc}
\min e_1 & +e_2+\ldots+e_m+e_{m+1}+\ldots+e_{m+n} \\
e_1 \geq & ax_1+by_1-c \\
e_2 \geq & ax_2+by_2-c \\
\vdots & \\
e_m \geq & ax_m+by_m-c \\
e_{m+1} \geq & c-ax_{m+1}-by_{m+1} \\
\vdots & \\
e_{m+n} \geq & c-ax_{m+n}-by_{m+n} \\
 & e_i \geq 0
\end{array}$$

Duality

As it turns out, the max-flow min-cut theorem is a special case of a more general phenomenon called *duality*. Basically, duality means that for each maximization problem there is a corresponding minimizations problem with the property that any feasible solution of the original maximization problem is less than or equal any feasible solution of the "dual" minimization problem (known as *weak duality*). In fact, they actually the same optimum (this is known as *strong duality*, which we won't prove in this course, though you should simply know this fact).

So, what exactly is duality? Let's first start with the following question: given a linear programming problem to maximize $c^T x$ subject to $Ax \leq b$, what is a systematic way of obtaining an upper bound on the highest possible value (the "optimum") achievable in this maximization problem? Let us label the rows of A as $a_1, \ldots, a_m$, and suppose x is n-dimensional. Then $Ax \leq b$ is the same as saying $\forall i\ a_i^T x \leq b_i$. One way to obtain an upper bound on the optimum is then following simple observation: if $y_i \geq 0$, then $a_i^T x \leq b_i$ implies that $y_i a_i^T x \leq y_i b_i$ (we just multiplied both sides of the inequality by y_i). Now suppose we choose $y_1, \ldots, y_m \geq 0$ such that $\sum_{i=1}^m y_i a_i = c$. Then taking a linear combination of the inequalities, we find that

$$c^T x \leq \sum_{i=1}^m y_i b_i = b^T y.$$

That is, we've found an upper bound on the maximum possible value achievable by the given linear program! Now, how do we get the best upper bound possible via this approach? Well, we just minimize $b^T y$! This then gives us the *dual linear program*: minimize $b^T y$ subject to the constraints $A^T y = c$ and for all $i = 1, \ldots, m$, $y_i \geq 0$.

By construction, it is obvious that the optimal solution to the primal linear program (i.e. the given one) has value *at most* the optimum for the dual. This is known as the *weak duality* theorem. A harder theorem, which we won't prove in class, is known as *strong duality* and asserts that if the primal has a finite optimum, then the primal and dual optimal values are equal!

As an example to illustrate duality, let us return to network flow. Consider the network shown in Figure 18.3, and the corresponding max-flow problem. We know that it can be written as a linear program as follows:

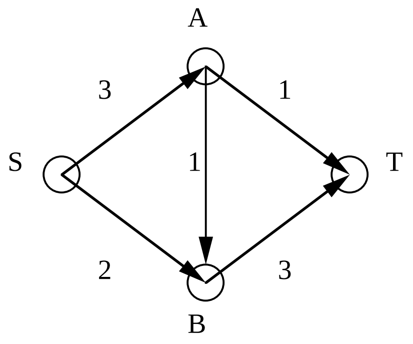

Figure 18.3: A simple max-flow problem

$$\begin{array}{llllllll}
\max & f_{SA} & +f_{SB} & & & & & \\
 & f_{SA} & & & & & \leq 3 & \\
 & & f_{SB} & & & & \leq 2 & \\
 & & & f_{AB} & & & \leq 1 & \\
 & & & & f_{AT} & & \leq 1 & \qquad P \\
 & & & & & f_{BT} & \leq 3 & \\
 & f_{SA} & & -f_{AB} & -f_{AT} & & = 0 & \\
 & & f_{SB} & +f_{AB} & & -f_{BT} & = 0 & \\
 & & & & & & f \geq 0 &
\end{array}$$

Consider now the following linear program:

$$\begin{array}{lllllllll}
\min 3y_{SA} & +2y_{SB} & +y_{AB} & +y_{AT} & +3y_{BT} & & & & \\
y_{SA} & & & & & +u_A & & \geq 1 & \\
 & y_{SB} & & & & & +u_B & \geq 1 & \\
 & & y_{AB} & & & -u_A & +u_B & \geq 0 & \qquad D \\
 & & & y_{AT} & & -u_A & & \geq 0 & \\
 & & & & y_{BT} & & -u_B & \geq 0 & \\
 & & & & & & & y \geq 0 &
\end{array}$$

This LP describes the min-cut problem! To see why, suppose that the u_A variable is meant to be 1 if A is in the cut with S, and 0 otherwise, and similarly for B (naturally, by the definition of a cut, S will always be with S in the cut, and T will never be with S). Each of the y variables is to be 1 if the corresponding edge contributes to the cut capacity, and 0 otherwise. Then the constraints make sure that these variables behave exactly as they should. For example, the second constraint states that *if A is not with S, then SA must be added to the cut.* The third one states that *if A is with S and B is not* (this is the only case in which the sum $-u_A + u_B$ becomes -1), *then AB must contribute* *to the cut.* And so on. Although the y and u's are free to take values larger than one, they will be "slammed" by the minimization down to 1 or 0.

Let us now make a remarkable observation: these two programs have strikingly symmetric, *dual*, structure. This structure is most easily seen by putting the linear programs in matrix form. The first program, which we call the primal (P), we write as:

max	1	1	0	0	0		
	1	0	0	0	0	$\leq$	3
	0	1	0	0	0	$\leq$	2
	0	0	1	0	0	$\leq$	1
	0	0	0	1	0	$\leq$	1
	0	0	0	0	1	$\leq$	3
	1	0	-1	-1	1	$=$	0
	0	1	1	0	-1	$=$	0
	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$		

Here we have removed the actual variable names, and we have included an additional row at the bottom denoting that all the variables are non-negative. (An unrestricted variable will be denoted by unr.

The second program, which we call the dual (D), we write as:

min	3	2	1	1	3	0	0		
	1	0	0	0	0	1	0	$\geq$	1
	0	1	0	0	0	0	1	$\geq$	1
	0	0	1	0	0	-1	1	$\geq$	0
	0	0	0	1	0	-1	0	$\geq$	0
	0	0	0	0	1	0	-1	$\geq$	0
	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	unr	unr		

Each variable of P corresponds to a constraint of D, and vice-versa. Equality constraints correspond to unrestricted variables (the u's), and inequality constraints to restricted variables. Minimization becomes maximization. The matrices are transpose of one another, and the roles of right-hand side and objective function are interchanged.

Such LP's are called *dual* to each other. It is mechanical, given an LP, to form its dual. Suppose we start with a maximization problem. Change all inequality constraints into $\leq$ constraints, negating both sides of an equation if necessary. Then

- transpose the coefficient matrix
- invert maximization to minimization
- interchange the roles of the right-hand side and the objective function
- introduce a nonnegative variable for each inequality, and an unrestricted one for each equality
- for each nonnegative variable introduce a $\geq$ constraint, and for each unrestricted variable introduce an equality constraint.

If we start with a minimization problem, we instead begin by turning all inequality constraints into $\geq$ constraints, we make the dual a maximization, and we change the last step so that each nonnegative variable corresponds to a $\leq$ constraint. Note that it is easy to show from this description that the dual of the dual is the original primal problem!

By the max-flow min-cut theorem, the two LP's P and D above have the same optimum. However as we discussed earlier, this is true for all linear programs by strong duality.

Circuit Evaluation

We have seen many interesting and diverse applications of linear programming. In some sense, the next one is the *ultimate* application. Suppose that we are given a *Boolean circuit*, that is, a DAG of gates, each of which is either an input gate (indegree zero, and has a value T or F), or an OR gate (indegree two), or an AND gate (indegree two), or a NOT gate (indegree one). One of them is designated as the output gate. We wish to tell if this circuit evaluates (following the laws of Boolean values bottom-up) to T. This is known as *the circuit value problem.*

There is a very simple and automatic way of translating the circuit value problem into an LP: for each gate g we have a variable x_g. For all gates we have $0 \leq x_g \leq 1$. If g is a T input gate, we have the equation $x_g = 1$; if it is F, $x_g = 0$. If it is an OR gate, say of the gates h and h', then we have the inequality $x_g \leq x_h + x_{h'}$. If it is an AND gate of h and h', we have the inequalities $x_g \leq x_h$, $x_g \leq x_{h'}$ (notice the difference). For a NOT gate we say $x_g = 1 - x_h$. Finally, we want to $\max x_o$, where o is the output gate. It is easy to see that the optimum value of x_o will be 1 if the circuit value if T, and 0 if it is F.

This is a rather straight-forward reduction to LP, from a problem that may not seem very interesting or hard at first. However, the circuit value problem is in some sense the most general problem solvable in polynomial time! Here is a justification of this statement: after all, a polynomial time algorithm runs on a computer, and the computer

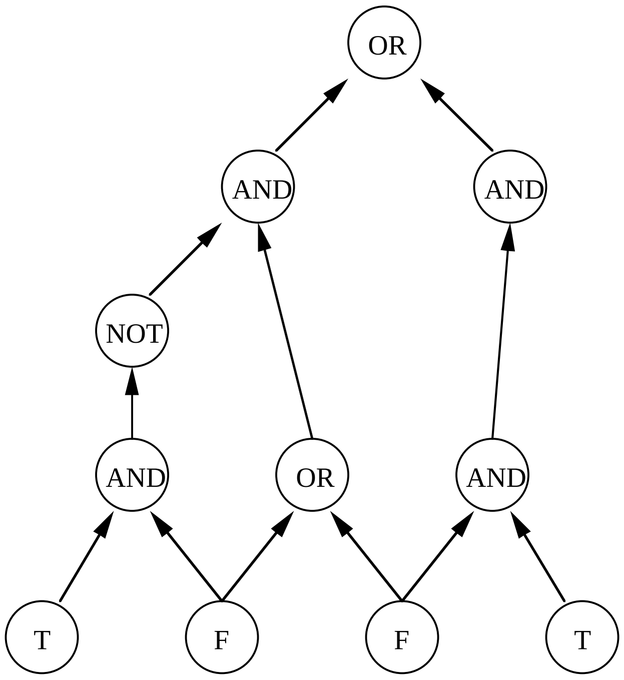

Figure 18.4: A Boolean circuit

is ultimately a Boolean circuit implemented on a chip. Since the algorithm runs in polynomial time, it can be rendered as a circuit consisting of polynomially many superpositions of the computer's circuit. Hence, the fact that circuit value problem reduces to LP means that *all polynomially solvable problems do!*

In our next topic, *Complexity and NP-completeness*, we shall see that a class that contains many hard problems reduces, much the same way, to *integer programming*.