Algorithms (XIII)

Yu Yu

Shanghai Jiaotong University

Review of the Previous Lecture

Chapter 7. Linear programming and reductions

Variants of linear programming

A general linear program has many degrees of freedom:

1. It can be either a maximization or a minimization problem.
2. Its constraints can be equations and/or inequalities.
3. The variables are often restricted to be nonnegative, but they can also be unrestricted in sign.

We will now show that these various LP options can all be *reduced* to one another via simple transformations.

Variants of linear programming (cont'd)

1. To turn a maximization problem into a minimization (or vice versa), just multiply the coefficients of the objective function by -1.

2a. To turn an inequality constraint like $\sum_{i=1}^{n} a_i x_i \leq b$ into an equation, introduce a new variable s and use

$$\sum_{i=1}^{n} a_i x_i + s = b$$

$$s \geq 0.$$

This s is called the **slack variable** for the inequality.

2b. To change an equality constraint into inequalities is easy: rewrite $ax = b$ as the equivalent pair of constraints $ax \leq b$ and $ax \geq b$.

3. Finally, to deal with a variable x that is unrestricted in sign, do the following:
   - Introduce two nonnegative variables, $x^+, x^- \geq 0$.
   - Replace x, wherever it occurs in the constraints or the objective function, by $x^+ - x^-$.

Standard form

Therefore, we can reduce any LP (maximization or minimization, with both inequalities and equations, and with both nonnegative and unrestricted variables) into an LP of a much more constrained kind that we call the **standard form**:

- the variables are all nonnegative,
- the constraints are all equations,
- and the objective function is to be minimized.

$$\begin{array}{c} \max\ x_1 + 6x_2 \\ x_1 \leq 200 \\ x_2 \leq 300 \\ x_1 + x_2 \leq 400 \\ x_1, x_2, x_3 \geq 0 \end{array} \quad \Longrightarrow \quad \begin{array}{c} \min\ -x_1 - 6x_2 \\ x_1 + s_1 = 200 \\ x_2 + s_2 = 300 \\ x_1 + x_2 + s_3 = 400 \\ x_1, x_2, s_1, s_2, s_3 \geq 0 \end{array}$$

Flows in networks

Shipping oil

We have a network of pipelines along which oil can be sent. The *goal* is to ship as much oil as possible from the **source** s and the **sink** t.

Each pipeline has a *maximum capacity* it can handle, and there are no opportunities for storing oil en route.

Maximizing flow

The networks consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, which are, respectively, a **source** and **sink** of G; and **capacities** $c_e > 0$ on the edges.

We would like to send as much oil as possible from s to t without exceeding the capacities of any of the edges.
A particular shipping scheme is called a **flow** and consists of a variable f_e for each edge e of the network, satisfying the following two properties:

1. It doesn't violate edge capacities: $0 \leq f_e \leq c_e$ for all $e \in E$.
2. For all nodes u except s and t, the amount of flow entering u equals the amount leaving

$$\sum_{(w,v)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$$

In other words, flow is *conserved*.

Maximizing flow (cont'd)

The **size** of a flow is the total quantity sent from s to t and, by the *conservation principle*, is equal to the quantity leaving s:

$$\texttt{size}(f) = \sum_{(s,u)\in E} f_{su}.$$

In short, our goal is to assign values to $\{f_e \mid e \in E\}$ that will satisfy a set of linear constraints and maximize a linear objective function.
But this is a linear program! The maximum-flow problem reduces to linear programming.

A closer look at the algorithm

All we know so far of the simplex algorithm is the vague *geometric intuition* that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function until it finally reaches the optimal solution.

The behavior of simplex has an *elementary interpretation:*

Start with zero flow.

Repeat: choose an appropriate path from s to t, and increase flow along the edges of this path as much as possible.

A closer look at the algorithm (cont'd)

There is just one complication.

What if we choose a path that blocks all other paths?

Simplex gets around this problem by also allowing paths to *cancel existing flow*.

To summarize, in each iteration simplex looks for an s-t path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet *at full capacity*.
2. The reverse edge (v, u) is in the original network, and there is some flow along it.

If the current flow is f, then in the first case, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow, and in the second case, up to f_{vu} additional units (canceling all or part of the existing flow on (v, u)).

A closer look at the algorithm (cont'd)

These flow-increasing opportunities can be captured in a **residual network** $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0. \end{cases}$$

Thus we can equivalently think of simplex as choosing an s-t path in the residual network.

By simulating the behavior of simplex, we get a *direct algorithm* for solving max-flow.

It proceeds in iterations, each time explicitly constructing G^f, finding a suitable s-t path in G^f by using, say, a linear-time breadth-first search, and halting if there is no longer any such path along which flow can be increased.

Cuts

A truly remarkable fact:

> *not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!*

An (s, t)-**cut** partitions the vertices into two *disjoint* groups L and R such that $s \in L$ and $t \in R$.
Its **capacity** is the total capacity of the edges from L to R,
and as argued previously, is an *upper bound* on *any* flow:

Pick any flow f and any (s, t)-cut (L, R). Then $\texttt{size}(f) \leq \texttt{capacity}(L, R)$.

A certificate of optimality

Theorem (Max-flow min-cut)

The size of the maximum flow in a network equals the capacity of the smallest (s, t)-cut.

Proof.

Suppose f is the final flow when the algorithm terminates.
We know that node t is no longer reachable from s in the residual network G^f.
Let L be the nodes that are reachable from s in G^f, and let $R = V \setminus L$ be the rest of the nodes.
We claim that

$$\texttt{size}(f) = \texttt{capacity}(L, R)$$

To see this, observe that by the way L is defined, any edge going from L to R must be at full capacity (in the current flow f), and any edge from R to L must have zero flow.
Therefore the net flow across (L, R) is exactly the capacity of the cut. □

Efficiency

Each iteration of our maximum-flow algorithm is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an s-t path.

But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$.
Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most $C|E|$, the number of iterations is at most this much.

If paths are chosen in a sensible manner – in particular, by using a BFS, which finds the path with the fewest edges – then the number of iterations is at most $O(|V| \cdot |E|)$, no matter what the capacities are.

This latter bound gives an overall running time of $O(|V| \cdot |E|^2)$ for maximum flow.

Duality

Recall:

$$\begin{aligned} \max \ & x_1 + 6x_2 \\ & x_1 \leq 200 \\ & x_2 \leq 300 \\ & x_1 + x_2 \leq 400 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Simplex declares the optimum solution to be $(x_1, x_2) = (100, 300)$, with objective value 1900.

Can this answer be checked somehow?

We take the first inequality and add it to six times the second inequality:

$$x_1 + 6x_2 \leq 2000.$$

Multiplying the three inequalities by 0, 5, and 1, respectively, and adding them up yields

$$x_1 + 6x_2 \leq 1900.$$

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

Multiplier						
	Inequality					
y_1		x_1			$\leq$	200
y_2				x_2	$\leq$	300
y_3		x_1	$+$	x_2	$\leq$	400

These y_i's must be nonnegative, for otherwise they are unqualified to multiply inequalities.

After the multiplication and addition steps, we get the bound:

$$(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \leq 200y_1 + 300y_2 + 400y_3.$$

We want the left-hand side to look like our objective function $x_1 + 6x_2$ so that the right-hand side is an upper bound on the optimum solution.

$$x_1 + 6x_2 \leq 200y_1 + 300y_2 + 400y_3 \quad \text{if} \quad \left\{ \begin{array}{c} y_1, y_2, y_3 \geq 0 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \end{array} \right\}$$

We can easily find y's that satisfy the inequalities on the right by simply making them large enough, for example $(y_1, y_2, y_3) = (5, 3, 6)$.
But these particular multipliers would tell us that the optimum solution of the LP is at most

$$200 \cdot 5 + 300 \cdot 3 + 400 \cdot 6 = 4300,$$

a bound that is far too loose to be of interest.

What we want is a bound that is as tight as possible, so we should minimize $200y_1 + 300y_2 + 400y_3$ subject to the preceding inequalities. And this is a *new linear program*!

The dual program

$$\begin{aligned} \min \ & 200y_1 + 300y_2 + 400y_3 \\ & y_1 + y_3 \geq 1 \\ & y_2 + y_3 \geq 6 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

By design, any feasible value of this *dual* LP is an upper bound on the original *primal* LP.

So if we somehow find a pair of primal and dual feasible values that are equal, then they must both be optimal. Here is just such a pair:

Primal: $(x_1, x_2) = (100, 300)$; **Dual:** $(y_1, y_2, y_3) = (0, 5, 1)$.

They both have value 1900, and therefore they certify each other's optimality.

A generic primal LP in matrix-vector form and its dual

Primal LP:

$$\begin{aligned} &\max \mathbf{c}^T\mathbf{x} \\ &\mathbf{A}\mathbf{x} \leq \mathbf{b} \\ &\mathbf{x} \geq 0 \end{aligned}$$

Dual LP:

$$\begin{aligned} &\min \mathbf{y}^T\mathbf{b} \\ &\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T \\ &\mathbf{y} \geq 0 \end{aligned}$$

Theorem (**Duality**)

If a linear program has a bounded optimum, then so does its dual, and the two optimum values coincide.

Zero-sum games

Rock-paper-scissors game

Payoff matrix G**:** the Row player vs. the Column player

	r	p	s
r	0	-1	1
p	1	0	-1
s	-1	1	0

Mixed strategy

We play this game *repeatedly*.

If Row always makes the same move, Column will quickly catch on and will always play the countermove, winning every time.

Therefore Row should *mix things up*: we can model this by allowing Row to have a mixed strategy, in which on each turn she plays r with probability x_1, p with probability x_2, and s with probability x_3.
This strategy is specified by the vector $\mathbf{x} = (x_1, x_2, x_3)$, positive numbers that add up to 1.

Similarly, Column's mixed strategy is some $\mathbf{y} = (y_1, y_2, y_3)$.

On any given round of the game, there is an $x_i y_j$ chance that Row and Column will play the ith and jth moves, respectively. Therefore the *expected (average) payoff* is

$$\sum_{i,j} G_{ij} \cdot \text{Prob}[\text{Row plays } i, \text{ Column plays } j] = \sum_{i,j} G_{ij} x_i y_j.$$

Row wants to *maximize* this, while Column wants to *minimize* it.

Completely random strategy

Suppose Row plays the "*completely random*" strategy $x = (1/3, 1/3, 1/3)$.

If Column plays r, then the average payoff will be

$$\frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot -1 = 0.$$

This is also true if Column plays p, or s.

And since the payoff of any mixed strategy (y_1, y_2, y_3) is just a *weighted average* of the individual payoffs for playing r, p, and s, *it must also be zero*.

$$\sum_{i,j} G_{ij} x_i y_j = \sum_{i,j} G_{ij} \cdot \frac{1}{3} y_j = \sum_j y_j \left(\sum_i \frac{1}{3} G_{ij} \right) = \sum_j y_j \cdot 0 = 0.$$

Completely random strategy (cont'd)

Thus by playing the "completely random" strategy, Row *forces an expected payoff of zero*, no matter what Column does.
This means that Column cannot hope for a negative expected payoff (remember that he wants the payoff to be as small as possible).

Symmetrically, if Column plays the completely random strategy, he also forces an expected payoff of zero, and thus Row cannot hope for a positive expected payoff.

The best each player can do is to play completely randomly, with an expected payoff of zero.

Two scenarios

1. First Row *announces* her strategy, and then Column picks his.

2. First Column announces his strategy, and then Row chooses hers.

We've seen that the average payoff is the same (zero) in either case if both parties play optimally.
But this might well be due to the high level of symmetry in rock-paper-scissors.

In general games, we'd expect the first option to *favor Column*, since he knows Row's strategy and can fully exploit it while choosing his own.

Likewise, we'd expect the second option to *favor Row*.

Amazingly, this is not the case: if both play optimally, then it doesn't hurt a player to announce his or her strategy in advance!

What's more, this remarkable property is a consequence of and in fact – equivalent to – *linear programming duality*.

Presidential election

There are two candidates for office, and the moves they make correspond to campaign issues on which they can focus (the initials stand for **economy**, **society**, **morality**, and **tax cut**).

The payoff entries of G are millions of votes lost by Column.

	m	t
e	3	-1
s	-2	1

Pure strategy

Suppose Row announces that she will play the mixed strategy $x = (1/2, 1/2)$. What should Column do?

Move m will incur an expected loss of $1/2$, while t will incur an expected loss of 0. The best response of Column is therefore the *pure strategy* $\mathbf{y} = (0, 1)$.

More generally, once Row's strategy $\mathbf{x} = (x_1, x_2)$ is fixed, there is always a pure strategy that is optimal for Column: either move m, with payoff $3x_1 - 2x_2$, or t, with payoff $-x_1 + x_2$, whichever is smaller.

Therefore, if Row is forced to announce $\mathbf{x}$ before Column plays, she knows that his best response will achieve an expected payoff of $\min\{3x_1 - 2x_2, -x_1 + x_2\}$. She should choose $\mathbf{x}$ defensively to *maximize her payoff against this best response*:

$$\text{Pick } (x_1, x_2) \text{ that maximizes } \min\{3x_1 - 2x_2, -x_1 + x_2\}.$$

LP formulation

$$z = \min\{3x_1 - 2x_2, -x_1 + x_2\} \quad \text{is equivalent to} \quad \begin{array}{c} \max\ z \\ z \le 3x_1 - 2x_2 \\ z \le -x_1 + x_2 \end{array}$$

Row needs to chooses x_1 and x_2 to maximize this z:

$$\begin{array}{c} \max\ z \\ -3x_1 + 2x_2 + z \le 0 \\ x_1 - x_2 + z \le 0 \\ x_1 + x_2 = 1 \\ x_1, x_2 \ge 0 \end{array}$$

LP formulation (cont'd)

Symmetrically, if Column has to announce his strategy first, his best bet is to choose the mixed strategy **y** that minimizes his loss under Row's best response:

$$\text{Pick } (y_1, y_2) \text{ that minimizes } \max\{3y_1 - y_2, -2y_1 + y_2\}.$$

In LP form:

$$\begin{gathered} \min \ w \\ -3y_1 + y_2 + w \geq 0 \\ 2y_1 - y_2 + w \geq 0 \\ y_1 + y_2 = 1 \\ y_1, y_2 \geq 0 \end{gathered}$$

These two LPs are dual to each other (see Figure 7.11)! Hence, they have the same optimum, call it V.

Value of the game

By solving an LP, Row (the maximizer) can determine a strategy for herself that guarantees an expected outcome of at least V no matter what Column does.

And by solving the *dual* LP, Column (the minimizer) can guarantee an expected outcome of at most V , no matter what Row does.

It follows that this is the *uniquely defined optimal play*: a priori it wasn't even certain that such a play existed.

V is known as the value of the game.

In our example, it is $1/7$ and is realized when Row plays her optimum mixed strategy $(3/7, 4/7)$ and Column plays his optimum mixed strategy $(2/7, 5/7)$.

Min-max theorem of games

Theorem

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \sum_{i,j} G_{ij} x_i y_j \quad = \quad \min_{\mathbf{y}} \max_{\mathbf{x}} \sum_{i,j} G_{ij} x_i y_j$$

This is surprising, because the left-hand side, in which Row has to announce her strategy first, should presumably be better for Column than the right-hand side, in which he has to go first.

Duality equalizes the two.

The simplex algorithm

General description

```
let v be any vertex of the feasible region
while there is a neighbor v' of v with better objective value:
    set v = v'
```

Say there are n variables, $x_1, \ldots, x_n$.

Any setting of the x_i's can be represented by an n-tuple of real numbers and plotted in *n-dimensional space*.

A linear equation involving the x_i's defines a *hyperplane* in this same space $\mathbb{R}^n$, and the corresponding linear inequality defines a *half-space*, all points that are either precisely on the hyperplane or lie on one particular side of it.

Finally, the *feasible region* of the linear program is specified by a set of inequalities and is therefore the intersection of the corresponding half-spaces, a *convex polyhedron*.

Vertices and neighbors in n-dimensional space

Definition

Each vertex is the unique point at which some subset of hyperplanes meet.

Pick a subset of the inequalities. If there is a unique point that satisfies them with equality, and this point happens to be feasible, then it is a vertex.

Each vertex is specified by a set of n inequalities.

Definition

Two vertices are neighbors if they have $n - 1$ defining inequalities in common.

The algorithm

On each iteration, simplex has two tasks:

1. Check whether the current vertex is optimal (and if so, halt).
2. Determine where to move next.

As we will see, both tasks are easy if the vertex happens to be at the origin. And if the vertex is elsewhere, we will transform the coordinate system to move it to the origin!

The convenience for the origin

Suppose we have some generic LP:

$$\begin{aligned} &\max \mathbf{c}^T \mathbf{x} \\ &\mathbf{A}\mathbf{x} \leq \mathbf{b} \\ &\mathbf{x} \geq 0 \end{aligned}$$

where $\mathbf{x}$ is the vector of variables, $\mathbf{x} = (x_1, \ldots, x_n)$.

Suppose the origin is feasible. Then it is certainly a vertex, since it is the unique point at which the n inequalities

$$\{x_1 \geq 0, \ldots, x_n \geq 0\}$$

are *tight*.

Task 1 in the origin

Lemma

The origin is optimal if and only if all $c_i \leq 0$. (We need $\mathbf{b} > 0$)

Proof.

If all $c_i \leq 0$, then considering the constraints $\mathbf{x} \geq 0$, we can't hope for a better objective value.

Conversely, if some $c_i > 0$, then the origin is not optimal, since we can increase the objective function *by raising* x_i. □

Task 2 in the origin

We can move by increasing some x_i for which $c_i > 0$.

How much can we increase it?

Until we hit some other constraint. That is, we release the tight constraint $x_i \geq 0$ and increase x_i until some other inequality, previously loose, now becomes tight.

At that point, we again have exactly n tight inequalities, so we are at a *new* vertex.

What if our current vertex $\mathbf{u}$ is elsewhere?

The trick is to transform $\mathbf{u}$ into the origin, by shifting the coordinate system from the usual $(x_1, \ldots, x_n)$ to the "local view" from $\mathbf{u}$.

These local coordinates consist of (appropriately scaled) distances $y_1, \ldots, y_n$ to the n hyperplanes (inequalities) that define and enclose u.

Specifically, if one of these enclosing inequalities is $\mathbf{a}_i \cdot \mathbf{x} \leq b_i$, then the distance from a point x to that particular "wall" is

$$y_i = b_i - \mathbf{a}_i \cdot \mathbf{x}.$$

The n equations of this type, one per wall, define the y_i's as linear functions of the x_i's, and this relationship can be inverted to express the x_i's as a linear function of the y_i's.

Rewriting the LP

Thus we can rewrite the entire LP in terms of the y's.

This doesn't fundamentally change it (for instance, the optimal value stays the same), but expresses it in a different coordinate frame.

The revised "local" LP has the following three properties:

1. It includes the inequalities $\mathbf{y} \geq 0$, which are simply the transformed versions of the inequalities defining $\mathbf{u}$.
2. $\mathbf{u}$ itself is the origin in $\mathbf{y}$-space.
3. The cost function becomes $\max c_{\mathbf{u}} + \tilde{\mathbf{c}}^T \mathbf{y}$, where $c_{\mathbf{u}}$ is the value of the objective function at $\mathbf{u}$ and $\tilde{\mathbf{c}}$ is a transformed cost vector.