Algorithms (XIV)

Yu Yu

Shanghai Jiaotong University

Review of the Previous Lecture

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

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

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.

The starting vertex

In a general LP, the origin might not be feasible and thus not a vertex at all.

However, it turns out that finding a starting vertex can be reduced to an LP and solved by simplex!

Start with any linear program in *standard form*:

$$\min \mathbf{c}^T\mathbf{x} \text{ such that } \mathbf{Ax} = \mathbf{b} \text{ and } \mathbf{x} \geq 0.$$

We first make sure that the right-hand sides of the equations are all nonnegative: if $b_i < 0$, just multiply both sides of the ith equation by -1.

Then we create a new LP as follows:

- Create m new artificial variables $z_1, \ldots, z_m \geq 0$, where m is the number of equations.
- Add z_i to the left-hand side of the ith equation.
- Let the objective, to be *minimized*, be $z_1 + z_2 + \cdots + z_m$.

The starting vertex (cont'd)

For this new LP, it's easy to come up with a starting vertex, namely, the one with $z_i = b_i$ for all i and all other variables zero.

Therefore we can solve it by simplex, to obtain the optimum solution.

There are two cases:

1. If the optimum value of $z_1 + \cdots + z_m$ is zero, then all z_i's obtained by simplex are zero, and hence from the optimum vertex of the new LP we get a starting feasible vertex of the original LP, *just by ignoring the z_i's*.
2. If the optimum objective turns out to be positive: We tried to minimize the sum of the z_i's, but simplex decided that it cannot be zero. But this means that the original linear program is *infeasible*: it needs some nonzero z_i's to become feasible.

Degeneracy

A vertex is degenerate if it is the intersection of more than n faces of the polyhedron, say $n + 1$.
Algebraically, it means that if we choose any one of n sets of $n + 1$ inequalities and solve the corresponding system of three linear equations in n unknowns, we'll get the *same* solution in all $n + 1$ cases.
This is a *serious problem*: simplex may return a *suboptimal* degenerate vertex simply because all its neighbors are identical to it and thus have no better objective.

And if we modify simplex so that it detects degeneracy and continues to hop from vertex to vertex despite lack of any improvement in the cost, it may end up looping forever.

Degeneracy (cont'd)

One way to fix this is by a *perturbation*:

change each b_i by a tiny random amount to $b_i \pm \varepsilon_i$.

This doesn't change the essence of the LP since the ε_i's are tiny, but it has the effect of differentiating between the solutions of the linear systems.

Unboundedness

In some cases an LP is *unbounded*, in that its objective function can be made arbitrarily large (or small, if it's a minimization problem).

If this is the case, simplex will discover it: in exploring the neighborhood of a vertex, it will notice that taking out an inequality and adding another leads to an underdetermined system of equations that has an infinity of solutions.

And in fact (this is an easy test) the space of solutions contains a whole line across which the objective can become larger and larger, all the way to ∞.

In this case simplex halts and complains.

The running time of simplex

What is the running time of simplex, for a generic linear program:

$$\max \mathbf{c}^T \mathbf{x} \text{ such that } \mathbb{A}\mathbf{x} \leq 0 \text{ and } \mathbf{x} \geq 0.$$

where there are n variables and $\mathbf{A}$ contains m inequality constraints?

It is an **iterative** algorithm that proceeds from vertex to vertex.

Let $\mathbf{u}$ be the current vertex, i.e., *unique point at which n inequality constraints are satisfied with equality*. Each of its neighbors shares $n-1$ of these inequalities, so $\mathbf{u}$ can have at most $n \cdot m$ neighbors.

A *naive* way for an iteration: (1) check each potential neighbor to see whether it really is a vertex of the polyhedron, (2) determine its cost.

(2) is easy. (1) involves: *solve a system of n equations* and check whether the result is feasible.

By Gaussian elimination this takes $O(n^3)$ time, giving total $O(mn^4)$ per iteration.

The running time of simplex (cont'd)

A much better way: the mn^4 can be improved to mn.

Recall the *local view* from vertex $\mathbf{u}$. The per-iteration overhead of rewriting the LP in terms of the current local coordinates is just $O((m+n)n)$; since the local view changes only slightly between iterations, in just *one of its defining inequalities*.

Next, to select the *best* neighbor, we recall that the (local view of) the objective function is of the form

$$\max c_{\mathbf{u}} + \tilde{\mathbf{c}} \cdot \mathbf{y}$$

where $c_{\mathbf{u}}$ is the value of the objective function at $\mathbf{u}$.

This immediately identifies a promising direction to move: we pick any $\tilde{c}_i > 0$ (if there is none, then the current vertex is optimal and simplex halts).

Since the rest of the LP has now been rewritten in terms of the $\mathbf{y}$-coordinates, it is easy to determine how much y_i can be increased before some other inequality is violated. (And if we can increase y_i indefinitely, we know the LP is unbounded.)

The running time of simplex (cont'd)

How many iterations could there be?

At most

$$\binom{m+n}{n},$$

i.e., the number of vertices.

It is **exponential** in n. And in fact, there are examples of LPs for which simplex does indeed take an exponential number of iterations.

Simplex is an exponential-time algorithm.

However, such exponential examples do not occur in practice, and it is this fact that makes simplex so valuable and so widely used.

Postscript: circuit evaluation

An ultimate application

We are given a Boolean circuit, that is, a dag of gates of the following types.

- **Input** gates have indegree zero, with value `true` or `false`.
- **AND** gates and OR gates have indegree 2.
- **NOT** gates have indegree 1.

In addition, one of the gates is designated as the **output**.

The CIRCUIT VALUE problem is the following: when the laws of Boolean logic are applied to the gates in topological order, does the output evaluate to `true`?

LP formulation

Create a variable x_g for each gate g, with constraints $0 \leq x_g \leq 1$.

Add additional constraints for each type of gate:

- `true` gate: $x_g = 1$.
- `false` gate: $x_g = 0$.
- OR gate with inputs h and h': $x_g \geq x_h$, $x_g \geq x_{h'}$, $x_g \leq x_h + x_{h'}$.
- AND gate with inputs h and h': $x_g \leq x_h$, $x_g \leq x_{h'}$, $x_g \geq x_h + x_{h'} - 1$.
- NOT gate with input h: $x_g = 1 - x_h$.

These constraints force all the gates to take on exactly the right values – 0 for `false`, and 1 for `true`. We don't need to maximize or minimize anything, and we can read the answer off from the variable x_o corresponding to the **output** gate.

The generality

The CIRCUIT VALUE problem is in a sense the most general problem solvable in polynomial time!
After all, any algorithm will eventually run on a computer, and the computer is ultimately a Boolean combinational circuit implemented on a chip.

If the algorithm runs in polynomial time, it can be rendered as a Boolean circuit consisting of polynomially many copies of the computer's circuit, one per unit of time, with the values of the gates in one layer used to compute the values for the next.
Hence, the fact that CIRCUIT VALUE reduces to linear programming means that *all problems that can be solved in polynomial time do!*