Linear Programming

Albert Ye

March 30, 2023

1 Linear Programs

A linear program is a

2 Simplex

The idea behind simplex is simple. Our linear program is bounded by a polygon, and our goal is to find the optimal point on that polygon. We can see that optimal point must be on a vertex, so we just keep going to the next better vertex until our vertex is better than all neighboring vertices. Once it's better than all neighboring vertices, we see that it must be the best element in the graph or something.

The fact that optimality is gotten at a vertex can be seen through level curves. If a level curve intersects a line, then there is a better level curve that must also intersect the line.

There are two implementation issues:

1. Finding the starting "feasible vertex"
2. How to find neighboring vertices?

3 Duality

We want to find $\max 5x_1 + 4x_2$ such that

$$\begin{aligned} 2x_1 + x_2 &\leq 100(\cdot y_1) \\ x_1 \quad\quad &\leq 30(\cdot y_2) \\ x_2 &\leq 60(\cdot y_1) \end{aligned}$$

Then, we have that $(2y_1 + y_2) \cdot x_1 + (y_1 + y_3) \cdot x_2 \leq 100y_1 + 30y_2 + 60y_3$.

Now, we want to find $\min 100y_1 + 30y_2 + 60y_3$ such that

$$\begin{aligned} y_1, y_2, y_3 &\geq 0 \\ 5 &\leq 2y_1 + y_2 \\ 4 &\leq y_1 + y_2 \end{aligned}$$

This results in another linear program: the **dual** linear program. We then have $5x_1+4x_2 \leq 100y_1+30y_2+60y_3$.

In matrix form, we have:

Primal LP: $\max C^T \vec{x}$ s.t. $A\vec{x} \leq \vec{b}$ and $\vec{x} \geq 0$.

Dual LP: $\min \vec{b}^T \vec{y}$ s.t. $A^t \vec{y} \geq c$ and $\vec{y} \geq 0$.

3.1 Weak and Strong Duality

Theorem 3.1 (Weak Duality). The value of the feasible solution $\vec{x}$ to primal linear program **must be** $\leq$ the value of the feasible solution $\vec{y}$ to dual linear program.

Proof. FILL IN LATER ■

Theorem 3.2 (Strong Duality). If the primal opt is bounded, then primal opt = dual opt.

For example, min cut = max flow.

3.2 Zero-sum Games

Input: a payoff matrix M, with the row and column values determining different actions. The row player picks row r and column player picks row c.

The row player wins $+M_{rc}$ and the column player wins $-M_{rc}$. It's called a **zero-sum game** because the scores sum to 0.

Types of Strategies:

1. **Pure Strategy**: a single row / column, e.g. the row player always picks rock (beaten by column player always playing paper)
2. **Mixed Strategy**: probability distribution over pure strategies. For example, $\Pr[\text{Rock}] = \frac{1}{3}, \Pr[\text{Paper}] = \frac{1}{3}, \Pr[\text{Scissors}] = \frac{1}{3}$. The average score is 0 regardless of what the column player does.

Example 3.1 (Game 1).

$$M = \begin{bmatrix} 3 & -1 \\ -2 & 1 \end{bmatrix}.$$

Row player goes first, column player goes second.

First, the row player announces a mixed strat $p = (p_1, p_2)$, then the column player announces a mixed strat $q = (q_1, q_2)$.

Definition 3.3 (Average Score). Row player's **average score** is $\text{Score}(p, q) = 3p_1q_1 - 1p_1q_2 - 2p_2q_1 + 1p_2q_2$.

The column player's best strat is to minimize $\text{Score}(p, q)$ over all mixed strategies q, which is equivalent to minimizing $\text{Score}(p, q)$ over all pure strategies q. This is equal to $\min(3p_1 - 2p_2, -p_1 + p_2)$, which is the minimum of the column 1 and column 2 score.

The row player will have to pick (p_1, p_2) that gives them the maximum $\text{Score}(p, q)$. We can compute the $\max_p(\min(3p_1 - 2p_2, -p_1 + p_2))$ with linear programming.

Proof. Maximize z, subject to

$$\begin{aligned} z &\leq 3p_1 - 2p_2 \\ z &\leq -p_1 + p_2 \\ p_1 + p_2 &= 1 \\ p_1 \geq 0, p_2 &\geq 0. \end{aligned}$$

This is a linear program. Note that $z = \min(3p_1 - 2p_2, -p_1 + p - 2)$. ■

Example 3.1 (Game 2)**.** Same as game 1, except the column player goes first and the row player goes second.

Now, the row player does a pure strategy and the col player does a mixed strategy.

Note that the payoff of row 1 is $3q_1 - q_2$ and the payoff of row 2 is $-2q_1 + q_2$. So the **row** player's best strat is $\max(3q_1 - q_2, -2q_1 + q_2)$. As a result, the column player's best strat is $\min_q max(3q_1 - q_2, -2q_1 + q_2)$, where $\min_q(x)$ means the minimum value of x over all strategies q.

Comparing the two games, we have $\max_p(\min_q(score(p,q))) \leq \min_q(\max_p(score(p,q)))$. We can go further with strong duality, though:

Theorem 3.4 (Min-Max Theorem)**.** $\max_p(\min_q(score(p,q))) = \min_q(\max_p(score(p,q)))$

3.3 Experts Problem

There are n experts $E_1, \ldots, E_n$. Every day, they make a prediction. On the tth day,

1. Pick and expert E_i
2. Each expert E_j incurs a loss $l_j^{(t)} \in \{0, 1\}$.
3. You incur the loss $l_i^{(t)}$

Allowed to pick a random expert on the tth day. We have a probability distribution $p^{(t)} = (p_1^{(t)}, p_2^{(t)}, \ldots, p_n^{(t)})$ such that we pick expert E_i with probability $p_i^{(t)}$.

The **expected loss** on the tth day is

$$a^{(t)} = \sum_{i=1}^{n} p_i^{(t)} \cdot l_i^{(t)}.$$

The expected loss after T days is then

$$A^T = \sum_{t=1}^{T} a^{(t)}.$$

The loss of expected E_i after T days is

$$L_i^T = \sum_{i=1}^{T} l_i^{(t)},$$

and our **regret** $R^T =$ (expected loss after T days) $-$ (loss of best expert) $= A^T - \min_{1 \leq i \leq n} L_i^t$.

Remark 3.5. Surprising features of the model:

1. **Not** assuming that past performance predicts future performance
   $\therefore$ all algorithms are useless.
2. Expert losses can be adversarial, that is, losses can depend on the distribution you specify.

Theorem 3.6. There exists an algorithm with regret $R^T \leq 2\sqrt{T \ln n}$.

$\therefore$ after time T, **average regret** $\frac{R^T}{T} \leq \frac{2\sqrt{\ln n}}{\sqrt{T}}$, so as $T \to \infty$, $\frac{R^T}{T} \to 0$.

For each timestep, we update the weight $p_i^{(t+1)} = p_i^{(t)}(1-\epsilon)^{l(t)}$, where $l(t) = 0$ if it's correct and $l(t) = 1$ if it's wrong.

Regret is bounded to $R_T \leq \epsilon T + \frac{\ln n}{\epsilon}$. This becomes $R_T \leq 2\sqrt{T \ln n}$ if $\epsilon = \sqrt{\frac{\ln n}{T}}$.

4 Reductions

Reducing problem A to problem B means we have a black-box subroutine to solve problem B, we can then solve A.

The good news is if B is fast, then A is fast. But the bad news is, if A is hard then we know that B must be hard as well.

Example 4.1. The maximum bipartite matching can be reduced to max flow.

More formally, the input of the matching problem is a bipartite graph with two sets L and R with some edges between $E \subseteq L \times R$.

A matching is a subset of edges such that no pair of edges touch the same vertex. A **matching** is the matching that includes the most number of vertices.

An even more general case of the perfect bipartite matching in the flows notes, and the solution is pretty much similar. Just hook s to all vertices in L with capacity 1, and hook all vertices in R to t with capacity 1. Also, set the capacities of each edge in the bipartite graph to 1 and going from L to R.

Example 4.2. Any polynomial-time algorithm can be reduced to a linear program.

Example 4.3. Matrix inversion can be reduced to matrix multiplication, and vice versa.

THe bad news is related to NP-completeness.

5 Multiplicative Weights