1 Linear Programming with Randomized Rounding

Linear programming and randomization turn out to be a very powerful when used in combination. We will illustrate this by presenting an algorithm of Raghavan and Thompson [1] for a problem of routing paths in a network to minimize congestion. The analysis of the algorithm depends on the *Chernoff bound*, a fact from probability theory that is one of the most useful tools for analyzing randomized algorithms.

1.1 The Chernoff Bound

The Chernoff bound is a very useful theorem concerning the sum of a large number of independent random variables. It asserts that the probability that the sum deviates very far from its mean decays exponentially fast as the distance from the mean increases and as the number of trials increases.

Theorem 1. *Let $X_1, \ldots, X_n$ be independent random variables taking values in $[0,1]$, let X denote their sum, and let $\mu = \mathcal{E}(X)$. Then*

$$\Pr(X \geq k) < e^{-\mu}(e\mu/k)^k. \tag{1}$$

Corollary 2. *Let $X_1, \ldots, X_k$ be independent random variables taking values in $[0,1]$, let X denote their sum, and let $\mu = \mathcal{E}(X)$ such that $\mu \leq 1$. Then for any $N > 2$ and any $b \geq e \log N / \log\log N$,*

$$\Pr(X \geq b) < \frac{1}{N}. \tag{2}$$

Proof. Applying Theorem 1,

$$\Pr(X \geq b) \leq e^{-\mu}(e\mu/b)^b \leq (e/b)^b \leq \left(\frac{\log\log N}{\log N}\right)^{e \log N/\log\log N}, \tag{3}$$

so it suffices to show that

$$\left(\frac{\log\log N}{\log N}\right)^{e \log N/\log\log N} < \frac{1}{N}.$$

Both sides are positive, so we can take logs. Doing so reduces the problem to showing

$$\frac{\log N}{\log\log N} \log\left(\frac{\log\log N}{\log N}\right)^{e} < -\log N.$$

By a sequence of elementary manipulations, this simplifies to

$$\frac{(\log N)^{e-1}}{(\log \log N)^{e}} > 1. \tag{4}$$

The derivative of the function on the left-hand side vanishes when

$$\log \log N = \frac{e}{(e-1)\ln 2},$$

and plugging this into (4) gives a minimum value of $((e-1)\ln 2)^e \sim 1.61$. $\square$

1.2 An Approximation Algorithm for Congestion Minimization

We will design an approximation algorithm for the following optimization problem. The input consists of a directed graph $G = (V, E)$ with positive integer edge capacities c_e and a set of source-sink pairs (s_i, t_i), $1 \le i \le k$, where $(s_i, t_i) \in V^2$ and G contains at least one path from s_i to t_i. The algorithm must output a list of paths $P_1, \ldots, P_k$ such that P_i is a path from s_i to t_i. The *load* on edge e, denoted by ℓ_e, is defined to be the number of paths P_i that traverse edge e. The *congestion* of edge e is the ratio ℓ_e/c_e, and the algorithm's objective is to minimize congestion, that is, minimize the value of $\max_{e \in E}(\ell_e/c_e)$. This problem turns out to be NP-hard, although we will not prove that here.

The first step in designing our approximation algorithm is to formulate the problem as an integer programming problem. We define a decision variable x_{ie} for each $1 \le i \le k$ and $e \in E$ that can take values 1 or 0 to denote whether e belongs to P_i or not. The resulting integer program can be written as follows, using $\delta^+(v)$ to denote the set of edges leaving v and $\delta^-(v)$ to denote the set of edges entering v.

$$\begin{array}{lll}
\text{minimize} & r & \\
\text{subject to} & \sum_{e \in \delta^+(v)} x_{ie} - \sum_{e \in \delta^-(v)} x_{ie} = \begin{cases} 1, & v = s_i \\ -1, & v = t_i \\ 0, & v \notin \{s_i, t_i\} \end{cases} & \text{for } 1 \le i \le k,\ v \in V \\
& \sum_{i=1}^{k} x_{ie} \le c_e r,\ x_{ie} \ge 0 & \text{for } 1 \le i \le k,\ e \in E
\end{array}$$

When (x_{ie}) is a $\{0,1\}$-valued vector obtained from a collection of paths $P_1, \ldots, P_k$ by setting $x_{ie} = 1$ for all $e \in P_i$, the first constraint ensures that P_i is a path from s_i to t_i while the second ensures that the congestion of each edge is bounded above by r.

Now we take the LP relaxation of this integer program and allow the variables x_{ie} to take fractional values. Our approximation algorithm solves the LP, does some postprocessing of the solution to obtain a probability distribution over paths for each terminal pair (s_i, t_i), and then outputs an independent random sample from each of these distributions.

To describe the postprocessing step, it helps to observe that the first LP constraint says that for every $i \in \{1, \ldots, k\}$, the values x_{ie} define a network flow of value 1 from s_i to t_i.

Call a flow *acyclic* if there is no directed cycle C with positive flow on each edge of C. The first step of the postprocessing is to make the flow (x_{ie}) acyclic for each i. If there is an index $i \in \{1, \dots, k\}$ and a directed cycle C such that $x_{ie} > 0$ for every edge $e \in C$, then we can let $\delta = \min\{x_{ie} \mid e \in C\}$ and we can modify x_{ie} to $x_{ie} - \delta$ for every $e \in C$. This modified solution still satisfies all of the LP constraints and has strictly fewer variables with nonzero values. After finitely many such modifications, we must arrive at a solution in which each flow (x_{ie}), $1 \leq i \leq k$, is acyclic. This modified solution is also an optimal solution of the linear program, because we did not increase the value of any variable.

Next, for each $i \in \{1, \dots, k\}$ we take the acyclic flow (x_{ie}) and represent it as a probability distribution over paths from s_i to t_i, that is, as a set of ordered pairs (P, π_P) such that P is a path from s_i to t_i, π_P is a positive number interpreted as the probability of P, and the sum of the probabilities π_P over all paths P is equal to 1. The distribution can be constructed using the following algorithm.

Algorithm 1 Postprocessing algorithm to construct path distribution

```
1: **Given:** Source $s_i$, sink $t_i$, acyclic flow $x_{ie}$ of value 1 from $s_i$ to $t_i$.
2: Initialize $\mathcal{D}_i = \varnothing$.
3: while there is a path $P$ from $s_i$ to $t_i$ such that $x_{ie} > 0$ for all $e \in P$ do
4:     $\pi_P = \min\{x_{ie} \mid e \in P\}$
5:     $\mathcal{D}_i = \mathcal{D}_i \cup \{(P, \pi_P)\}$.
6:     for all $e \in P$ do
7:         $x_{ie} = x_{ie} - \pi_P$
8:     end for
9: end while
10: return $\mathcal{D}_i$
```

Each iteration of the **while** loop strictly reduces the number of edges with $x_{ie} > 0$, hence the algorithm must terminate after selecting at most m paths. When it terminates, the flow (x_{ie}) has value zero (as otherwise there would be a path from s_i to t_i with positive flow on each edge) and it is acyclic because (x_{ie}) was initially acyclic and we never put a nonzero amount of flow on an edge whose flow was initially zero. The only acyclic flow of value zero is the zero flow, so when the algorithm terminates we must have $x_{ie} = 0$ for all e.

Each time we selected a path P, we decreased the value of the flow by exactly π_P. The value was initially 1 and finally 0, so the sum of π_P over all paths P is exactly 1 as required. For every edge e, the value x_{ie} decreased by exactly π_P each time we selected a path P containing e, hence the combined probability of all paths containing e is exactly x_{ie}.

Performing the postprocessing Algorithm 1 for each i, we obtain probability distributions $\mathcal{D}_1, \dots, \mathcal{D}_k$ over paths from s_i to t_i, with the property that the probability of a random sample from $\mathcal{D}_i$ traversing edge e is equal to x_{ie}. Now we choose one path for each i by sampling the distributions $\mathcal{D}_i$ independently and output the resulting k-tuple of paths $P_1, \dots, P_k$.

We claim that with probability at least 1/2, the parameter $\max_{e \in E} \ell_e / c_e$ is at most αr, where $m = |E|$ and

$$\alpha = \frac{e \log(2m)}{\log \log(2m)}.$$

This follows from Corollary 2. For any edge e, define the independent random variables

$$X_i = \begin{cases} 1 & \text{if } e \in P_i \\ 0 & \text{otherwise.} \end{cases}$$

Let $X = X_1 + \cdots + X_k$. The X_i are independent and $\mathcal{E}(X) = \sum_{i=1}^k x_{ie}$, which is at most $c_e r$ because of the second LP constraint, so $\mathcal{E}(X/(c_e r)) \leq 1$. Applying Corollary 2 with $N = 2m$,

$$\Pr(X \geq \alpha c_e r) \;\leq\; \Pr(X/(c_e r) \geq \alpha) \;\leq\; 1/(2m).$$

Since $X = \ell_e$, this means that the probability that ℓ_e/c_e exceeds αr is at most $1/(2m)$. Summing the probabilities of these failure events for each of the m edges of the graph, we find that with probability at least 1/2, none of the failure events occur and $\max_{e \in E} \ell_e/c_e$ is bounded above by αr.

Now r is a lower bound on the parameter $\max_{e \in E} \ell_e/c_e$ for *any* k-tuple of paths with the specified source-sink pairs, including the optimal solution OPT of the integer program, since any such k-tuple defines a valid LP solution and r is the optimum value of the LP. Consequently, with probability at least 1/2,

$$\max_{e \in E} \ell_e/c_e \;\leq\; \alpha r \;\leq\; \alpha\,\mathrm{OPT},$$

so our randomized algorithm achieves approximation factor α with probability at least 1/2.

Using the amplification technique from homework 6, this probability can be improved to $1 - \varepsilon$ for any desired $\varepsilon > 0$.

References

[1] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.