CS 224: Advanced Algorithms Spring 2017

Lecture 24 — April 14, 2017

Prof. Jelani Nelson *Scribe: Evan Yao*

1 Overview

In the last lecture we looked at the multiplicative weights (MW) algorithm as well as a variation of the n experts problem where we are asked to give a probability distribution $p^{(t)}$ at each iteration. We started using MW to give a solution to LP's in general.

In this lecture we will finish the analysis of LP's with the multiplicative weights algorithm and then start talking about network flows.

2 Recap

Recall the γ bounded oracle from last lecture:

Definition 1. *In a* γ-Bounded Oracle*, we are given* p*, a probability distribution over the rows of* A*. If*

$$\{x \in P,\ p^T Ax \geq p^T b\}$$

is feasible, then it returns such a feasible x *with the following property:*

$$\forall i \, |\langle A_i, x\rangle - b_i| \leq \gamma$$

Note that this is an easier feasibility problem than satisfying $\{x \in P, Ax \geq b\}$. This oracle says that if the former is feasible, then we can find a x satisfying the former that **almost** satisfies the latter (if γ is small). Each dimension can be off by a factor of γ.

In the last lecture, we proved the following bound for the variation of the experts problem where we are asked to give a probability vector $p^{(t)}$ at each time step. For all $\eta \in (0, \frac{1}{2}]$, this algorithm achieves the following bound:

$$\begin{aligned}\sum_{t=1}^{T} \langle m^{(t)}, p^{(t)}\rangle &\leq \min_{1\leq i\leq n}\left(\sum_i m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}|\right) + \frac{\ln n}{\eta} \\ &\leq \min_i \sum_{t=1}^{T} m_i t + \eta T + \frac{\ln n}{\eta}\end{aligned}$$

The second inequality follows since we assume that $m_i^{(t)} \in [-1, 1]$ for all i, t. Today, we will combine this algorithm and this bound with a γ bounded oracle in order to solve LP's using MW.

3 Using Oracle together with MW

Recall the setup for the problem. There is a convex region P as well as $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$. We would like to decide feasibility of $\{x \in P, Ax \geq b\}$. If this problem is feasible, we would like to find $x \in P$ satisfying $Ax \geq b - \epsilon \cdot J$, where J is the all ones vector.

The MW-based algorithm runs as follows:

- Initialize $p^{(1)}$ to be the uniform distribution over the m rows of A
- Each row of A is an expert, so there are m experts.
- Use $p^{(t)}$ to get $x^{(t)}$ from the oracle. Output "infeasible" if the oracle says infeasible.
- Once we get back this $x^{(t)}$, we define $m_i^{(t)} = \frac{1}{\gamma} \cdot \left(\langle A_i, x^{(t)} \rangle - b_i\right)$. We know that the absolute value of $m_i^{(t)}$ is less than 1 because of the gamma bounded oracle gives you an $x^{(t)}$ such that $\left|\langle A_i, x^{(t)} \rangle - b_i\right| \leq \gamma$.

  Notice that if a constraint i is well-satisfied, $m_i^{(t)}$ will be close to 1, and if it is very unsatisfied, then $m_i^{(t)}$ will be close to -1.
- Based on $m^{(t)}$ and $p^{(t)}$, we then obtain $p^{(t+1)}$ via the MW update rule from last lecture.

Theorem 2. *$\forall \epsilon \in (0,1)$, if we have a γ bounded oracle for $\gamma \geq \epsilon/2$, then we can find x such that $x \in P$, $Ax \geq b - \epsilon \cdot J$ in $O(\frac{\gamma^2 \ln m}{\epsilon^2})$ iterations (i.e. days) of multiplicative weights.*

Note if $\gamma < \epsilon/2$, we can just redefine γ to be $\epsilon/2$.

Proof. For any time period t, we have:

$$\begin{aligned}
\langle m^{(t)}, p^{(t)} \rangle &= (p^{(t)})^T m^{(t)} \\
&= \frac{1}{\gamma} (p^{(t)})^T \left(Ax^{(t)} - b\right) \\
&= \frac{1}{\gamma} \left((p^{(t)})^T Ax^{(t)} - (p^{(t)})^T b\right) \\
&\geq 0.
\end{aligned}$$

Note that $(p^{(t)})^T$ refers to the *transpose* of the vector $p^{(t)}$. To get the last step, we used the fact that $x^{(t)}$ was returned by the oracle and thus satisfies $\{x \in P, p^T A \geq p^T b\}$. Once we have the above inequality, we sum over all t so that for all (constraints) i:

$$\begin{aligned}
0 &\leq \sum_{i=1}^{T} \langle m^{(t)}, p^{(t)} \rangle \\
&\leq \sum_{t=1}^{T} m_i^{(t)} + \eta T + \frac{\ln m}{\eta} \\
&\leq \frac{1}{\gamma} \sum_{t=1}^{T} \left(\langle A_i, x^t \rangle - b_i\right) + \eta T + \frac{\ln n}{\eta}
\end{aligned}$$

Now, let $x = \frac{1}{T}\sum_{t=1}^{T} x^{(t)}$. By the linearity of the inner product, we have:

$$0 \leq \frac{T}{\gamma} \cdot \langle A_i, x \rangle - \frac{T}{\gamma} \cdot b_i + \eta T + \frac{\ln n}{\eta}$$

Dividing through by $\frac{T}{\gamma}$ gives us:

$$0 \leq \langle A_i, x \rangle - b_i + \gamma\eta + \frac{\ln n \cdot \gamma}{\eta \cdot T}$$

Now, it is just a matter of choosing η and T so that $\gamma\nu + \frac{\ln n \cdot \gamma}{\eta \cdot T} \leq \epsilon$. Choose $\eta \leq \frac{\epsilon}{2\gamma}$ and then choose $T \geq \frac{2\gamma \ln m}{\eta}$. This will make both terms less than $\frac{\epsilon}{2}$.

This implies:

$$\langle A_i, x \rangle \geq b_i - \epsilon$$

for each row i of A. Hence, we know that

$$Ax \geq b - \epsilon \cdot J$$

Therefore, it suffices to run multiplicative weights for $T \geq \frac{2\gamma \ln m}{\eta\epsilon} \geq \frac{4\gamma^2 \ln m}{\epsilon^2}$ iterations. □

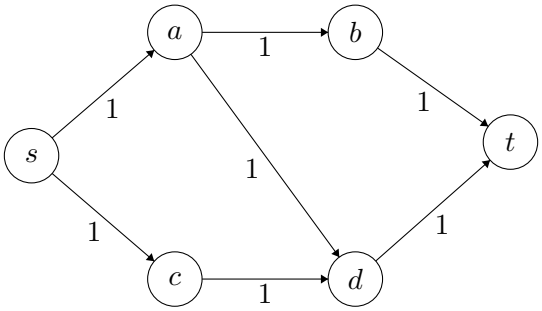
4 Flows

4.1 Setup

- We are given a directed graph $G = (V, E)$ with a source $s \in V$ and sink $t \in V$.
- Each edge e of the graph has a capacity $u_e \in \{1, 2, \ldots, U\}$.
- Our goal in max flow is "route" as much flow as possible from s to t.

Intuition: The source s is an infinite source of water while the sink t can accept an infinite pool of water. The capacity of each pipe is the amount of water that can go through it per unit of time. We are trying to find the maximum amount of water that can be transported, per unit of time, from s to t.

Example: In the graph below, it is possible to send 2 units of flow from s to t. However, the greedy algorithm of choosing paths from s to t and filling them completely does not work. If we choose the path $s \to a \to d \to t$ and put 1 unit of flow on that edge, then there is no way to find another path from s to t to send the other unit of flow.



Definition 3. *A flow f is a vector in $\mathbb{R}^m$ such that the following properties hold:*

- *Nonegativitiy:*
$$f_i \geq 0$$
for each coordinate of f. The flow through an edge cannot be negative.

- *Conservation of Flow: $\forall u \in V \setminus \{s, t\}$*
$$\sum_{e=(u,\cdot)} f_e = \sum_{e=(\cdot,u)} f_e$$
The sum of the water going in and coming out of a vertex (not the source or sink) must be equal.

  It follows from conservation of flow that the amount of flow exiting the source is the same as the amount entering the sink.

- *Capacity: $\forall e$*
$$f_e \leq u_e$$
The flow through an edge must obey that edge's capacity constraint.

The objective is to maximize the flow coming out of s or equivalently the flow flowing into t:

$$\max \sum_{e=(s,\cdot)} f_e = \max \sum_{e=(\cdot,t)} f_e$$

4.2 Current Literature

In this section, we will look at some of the existing algorithms for solving max flow. Sometimes, we will use $\tilde{O}$ instead of O to hide polylogarithmic factors. We will also use the standard notation of m being the number of edges and n being the number of vertices. Without loss of generality, we assume that $m \geq n - 1$ (in a pre-processing step we can throw away all vertices which are not reachable from s).

Interior Point Methods:

Using interior point methods, we can solve the LP for max flow in $\tilde{O}(\sqrt{m'}L)$ iterations of interior point, where m' is the number of constraints in the max flow LP. Each iteration is $O(1)$ Newton steps, where we have

$$x^{k+1} \leftarrow x^k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

Another way of thinking about this is

$$\nabla^2 f(x_k)\,(x_k - x_{k+1}) = \nabla f(x_k)$$

and so finding x_{k+1} can be done by solving a system of linear equations.

Let A be the constraint matrix in the LP formulation of max flow. What are the dimensions of A? The max flow LP has m variables and $m' = \Theta(m+n)$ constraints, one for edge's capacity and non-negativity and one for each vertex's conservation of flow. But we assumed that $m \geq n-1$ and thus both dimensions of A are $\Theta(m)$. We can solve a system of linear equations in the same time as matrix multiplication, which naively is $O(m^3)$. Therefore, we can solve this LP via the interior point method with run-time

$$\tilde{O}(\sqrt{m}m^3 \cdot L) = \tilde{O}(m^{7/2} \lg U)$$

Using our definiton of L from the interor point method, it can be shown that $L = O(\lg U)$ for max flow.

Daitch and Spielman proved in [1] that the matrices A we get from the max flow problem are special so that you can solve linear systems in $\tilde{O}(m)$ time. This means that max flow can be solved in time $\tilde{O}(m^{3/2} \lg U)$.

Recall the objective function from the interior points method: $f_\lambda(x) = \lambda c^T x - \sum_{i=1} \ln(s(x_i))$. There's something interesting about optimizing this function, which is that if you take a linear problem and you repeat a constraint a million times, you've incremented m, but you really haven't fundamentally changed the problem at all. Lee and Sidford found an algorithm in [2] such that the number of iterations of interior point it runs would not increase even if constraints were duplicates or near duplicates of each other. At a (very) high level, they did this by weighing each of the constraints in the barrier function depending on how important they were based on the polytope itself. This led to $\tilde{O}(m\sqrt{n} \lg U)$ max flow solution (and fast linear programming overall).

Summary Table

The table below summarizes the above as well as adds some other max flow algorithm landmarks:

Authors	Run-time	Remark
Daitch and Spielman [1]	$\tilde{O}(m^{3/2} \lg U)$	A is a special matrix in max flow
Lee and Sidford [2]	$\tilde{O}(m\sqrt{n} \lg U)$	Interior-point based approach
Madry [3]	$\tilde{O}((mU)^{10/7})$	Pseudopolynomial
Goldberg and Rao [4]	$O(m \cdot \min(\sqrt{m}, n^{2/3}) \cdot \lg(n^2/m) \lg U)$	Weakly Polynomial
Orlin [5]	$O(mn + m^{31/16} \lg^2 n)$	Strongly Polynomial
King, Rao, and Tarjan [6]	$O(mn \cdot \log_{m/(n \log n)} n)$	Min of [4] and [6] is $O(mn)$

Minimum Congestion Flow

In undirected graphs, Kelner et al. [7] and Sherman [8] showed that we can get running time $O(m^{1+o(1)}/\epsilon^2)$ to find at least $(1-\epsilon)$ of the max flow. This was improved to $\tilde{O}(m/\epsilon^2)$ by Peng in [9]. The idea for these algorithms is to use the idea that maximum flow is equivalent to minimum congestion flow.

Definition 4. *For the **Minimum Congestion Flow** problem, we want to send 1 unit of flow from s to t. The cost of the flow is the maximum of f_e/u_e. We want to minimize this cost.*

These two problem are equivalent because we can find simply divide all the f's by the value of the max flow. This will result in 1 unit of flow being transmitted.

Define a matrix $B \in \mathbb{R}^{m \times n}$ such that the rows are indexed by the edges of G and the columns are indexed by the vertices of G. For each row (edge), put 1, -1 at the (u, v). 1 for u, -1 for v.

We have a demand vector $\chi \in \mathbb{R}^n$. That vector describes each vertex's need to ship or absorb flow. Set χ to be 1 in s and -1 in t. $B^T f$ gives us an n dimensional vector such that the vth entry is how much flow is being pushed out of v. If this quantity is negative, then this vertex is absorbing some flow, and if it is positive then some extra flow is spilling out of the vertex. Conservation is achieved when this entry is 0.

Let U be diagonal $m \times m$ matrix of all the edge capacities. Thus for the minimum congestion flow problem (equivalent to max flow), we wish to compute an m-dimensional vector f^* such that $B^T f^* = \chi$ and $||U^{-1} f^*||_\infty$ is as small as possible. Here, we have a constrained optimization problem, so we can't use gradient descent. These papers try to remove those constraints and do gradient descent. You turn this into an unconstrained optimization problem by: Find some flow f_0 from s to t using DFS that satisfies the entire demand χ. Look at $f^* - f_0$: this is a circulation, i.e. it routes the demand $\chi = 0$.

Then we wish to compute

$$c^* = \text{argmin}_{B^T c = 0} ||U^{-1}(f_0 + c)||_\infty$$

Let $\alpha_0 = U^{-1} f_0$. To make this an unconstrained optimization problem, we project a vector $z \in \mathbb{R}^m$ onto "circulation space", which is just the linear subspace of all z satisfying $B^T f = z$.

$$z^* = \text{argmin}_{z \in \mathbb{R}^m} ||\alpha_0 + U^{-1} P z||_\infty$$

where P is a projection matrix. This is now an unconstrained optimization problem where z is allowed to be anything in $\mathbb{R}^m$. Now one can use continuous optimization methods such as gradient descent, and the heart of these works is in designing a good projection matrix P with certain nice properties that we won't get into here.

5 Algorithms

Today, we will look at the **Ford Fulkerson** algorithm [10]. Its run-time is $O(m \cdot f^*)$ where f^* is the value of the max flow. Ford Fulkerson works by initializing $f = 0$ and while there exists an $s - t$ path p in the residual graph G_f, let λ be the smallest capacity on edges in p in the residual graph. Let $f'_e = \lambda$ for $e \in p$, 0 otherwise. Update f with $f + f'$. When there is no path left, return f.

But what is the residual graph G_f?

For notational convenience, let $f_e = -f_{r(e)}$ where $r(e)$ is the reverse edge of e. If $e \in E$, but $r(e)$ is not, then we'll pretend that $r(e) \in E$ with capacity 0. The residual graph G_f has the same vertex and edge sets as G, but with changed capacities. Its capacities u' satisfy:

$$u'_e = u_e - f_e + f_{r(e)}$$

Intuition: When flow is being sent in one direction, we allow flow to travel in the opposite direction by "pushing back" flow.

We will state the following theorems, but will not prove them. You can refer to notes from CS 124 or CLRS for complete proofs.

Theorem 5. *f^* is a max flow iff there is no s-t path in G_f with positive minimum capacity.*

Theorem 6. *The maximum s-t flow is equal to the minimum $s-t$ cut.*

Recall that an s-t cut is a partitioning of the vertices into 2 groups such that s and t are in different groups. The value of the cut is the sum of the capacities of all the edges leaving the s side and towards t side.

Next time, we will look at scaling algorithms for max s-t flow, which will give us a running time like $O(m^2 \ln U)$ time. We will also look at blocking flow algorithms, which can combine with scaling to give a runtime of $O(mn \ln U)$. It is also possible to get $O(m\sqrt{n})$ in unit graphs, where every vertex besides s and t either have indegree 1 or outdegree 1, e.g. the kinds of graphs that arise when reducing maximum bipartite matching to maximum s-t flow.

References

[1] Samuel I. Daitch, Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. *STOC 2008*: 451-460

[2] Yin Tat Lee, Aaron Sidford. Path Finding Methods for Linear Programming: Solving Linear Programs in (vrank) Iterations and Faster Algorithms for Maximum Flow. *FOCS 2014*: 424-433

[3] Aleksander Madry. Computing Maximum Flow with Augmenting Electrical Flows. *FOCS 2016*: 593-602

[4] Andrew V. Goldberg, Satish Rao: Beyond the Flow Decomposition Barrier. *J. ACM 45(5)*: 783-797 (1998)

[5] James B. Orlin: Max flows in O(nm) time, or better. *STOC 2013*: 765-774

[6] Valerie King, S. Rao, Robert Endre Tarjan: A Faster Deterministic Maximum Flow Algorithm. *J. Algorithms 17(3)*: 447-474 (1994)

[7] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, Aaron Sidford An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations. *SODA 2014*: 217-226

[8] Jonah Sherman Nearly Maximum Flows in Nearly Linear Time. *FOCS 2013*: 263-269

[9] Richard Peng: Approximate Undirected Maximum Flows in O(mpolylog(n)) Time. *SODA 2016*: 1862-1867

[10] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics* 8(3):399404, 1956