COMPSCI 638: Graph Algorithms November 6, 2019

Lecture 21

Lecturer: Debmalya Panigrahi *Scribe: Kevin Sun*

1 Overview

In the last lecture, we gave an introduction to the multiplicative weight update (MWU) method. In this lecture, we first show how MWU can be used to solve a class of linear programs. We then apply this to the maximum flow problem to obtain an approximate algorithm.

2 MWU for Packing Linear Programs

In this section, we illustrate an algorithm that applies the multiplicative weight update (MWU) method to a class of linear programs known as *packing* linear programs. More specifically, we want to find a vector $x \in \mathbb{R}^n$ such that $Ax \leq b$, where the entries of $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are all non-negative. The objective of a packing LP seeks is to maximize $c^\top x$ over this space, where $c \in \mathbb{R}^n$ also has non-negative entries.

Oracle specifications: Suppose we can the m constraints into m_1 "easy" and m_2 "hard" constraints. (The exact details of this procedure, as well as the implementation of the oracle, depend on the actual problem we want to solve.) Let P denote the polytope that encodes the easy constraints. Our framework requires an oracle that, given a weight $w_i \geq 0$ for each hard constraint i, returns a solution x that satisfies the following:

1. All easy constraints are satisfied, that is, $x \in P$.
2. The "average" hard constraint is satisfied, that is, $\sum_i w_i A_i x / b_i \leq \sum_i w_i$. (Note that this is weaker than the condition we actually want, i.e., $A_i x \leq b_i$ for every i.)
3. For every hard constraint i, we have $A_i x / b_i \leq \rho$ for some ρ. The value of ρ is known as the *width* of the linear program.

Using this oracle and MWU, we can solve the feasibility problem, as shown in Algorithm 1.

Algorithm 1 Packing feasibility via MWU

Input: Constraints specified by $A \in \mathbb{R}^{m \times n}_{\geq 0}, b \in \mathbb{R}^m_{\geq 0}$, an oracle satisfiying the above conditions with width ρ, and an accuracy parameter $\epsilon \in (0, 1)$.

Initialize $w_i^0 = 1$ for each of the m_2 hard constraints indexed by i.
for $t = 1, \ldots, T$ **do**
    Input the weights w^{t-1} into the oracle to obtain a solution x^t.
    for each hard constraint i **do**
        Set $w_i^t = (1 + \epsilon)^{v_i x^t} w_i^{t-1}$ where $v_i = A_i / \rho b_i$.
return $\overline{x} = \sum_{t=1}^T x^t / T$

Intuitively, if x^t badly violates the i-th hard constraint, then $v_i x^t$ is very large, so Algorithm 1 increases the weight of w_i by a lot. Once w_i is large, then in order for the oracle must satisfy the "average" constraint, the solution must satisfy the i-th constraint. The final output is the average of all over the oracle's solutions obtained over T steps.

Theorem 1. *In Algorithm 1, if $T \geq \rho \ln m_2/\epsilon^2$, then the output $\overline{x}$ satisfies $A\overline{x} \leq (1+\epsilon)b$.*

Proof. For ease of presentation, we will use the approximation $(1+\epsilon)^\delta \approx 1+\epsilon\delta$ for any $\delta \in [0,1]$. Consider the final value of w_i, at the end of T iterations:

$$w_i^T = \prod_{t=1}^{T}(1+\epsilon)^{v_i x^t} = (1+\epsilon)^{\sum_{t=1}^{T} v_i x^t} = (1+\epsilon)^{T v_i \overline{x}}. \tag{1}$$

Now let us analyze the change in the sum of weights over time:

$$\sum_i w_i^t = \sum_i w_i^{t-1}(1+\epsilon)^{v_i x^t} \approx \sum_i w_i^{t-1}(1+\epsilon v_i x^t) \leq \sum_i w_i^{t-1} + \frac{\epsilon w_i^{t-1}}{\rho} = \left(1+\frac{\epsilon}{\rho}\right)\sum_i w_i^{t-1},$$

where the inequality holds because the oracle solution x^t must satisfy the average constraint with respect to w^{t-1}, i.e., $\sum_i w_i^{t-1}\rho v_i x^t \leq \sum_i w_i^{t-1}$. Applying this over T steps yields

$$\sum_i w_i^T \leq \left(1+\frac{\epsilon}{\rho}\right)^T \cdot \sum_i w_i^0 = \left(1+\frac{\epsilon}{\rho}\right)^T \cdot m_2 \approx (1+\epsilon)^{T/\rho} \cdot m_2. \tag{2}$$

We now apply the inequality $w_i^T \leq \sum_i w_i^T$ on (1) and (2) to obtain

$$(1+\epsilon)^{T v_i \overline{x}} \leq (1+\epsilon)^{T/\rho} \cdot m_2$$

Taking the logarithm (with base $1+\epsilon$) of both sides implies

$$T v_i \overline{x} \leq \frac{T}{\rho} + \log_{1+\epsilon} m_2 = \frac{T}{\rho} + \frac{\ln m_2}{\ln(1+\epsilon)}.$$

For sufficiently small values of ϵ, we have $\ln(1+\epsilon) \approx \epsilon$, so substituting $v_i = A_i/\rho b_i$ yields

$$\frac{A_i \overline{x}}{b_i} \leq 1 + \frac{\rho \ln m_2}{\epsilon T}$$

If $T \geq \rho \ln m_2/\epsilon^2$, then $A_i\overline{x}/b_i \leq 1+\epsilon$, as desired. □

Remark: Notice the dependence of T on ρ: if the oracle has small width, then Algorithm 1 requires a fewer number of iterations we need to solve the feasibility problem. Furthermore, it is worth noting that this framework has had widespread success in multiple problems, but as we saw, the analysis is short and fairly straightforward. In fact, the only inequality we really needed is the fact that total weight is at least any individual weight. Finally, replacing $(1+\epsilon)$ with $(1-\epsilon)$ essentially allows us to solve the covering LPs, in which we seek a vector x such that $Ax \geq b$.

3 Approximate Maximum Flow

In this section, we apply the framework described in Section 2 to the maximum flow problem. Let $G = (V, E)$ be a directed graph on n vertices and m edges, where each edge e has capacity $u(e) \geq 0$, and let $s, t \in V$ be the source and sink vertices. For any vertex $u \in V$, let $\delta^{in}(u)$ and $\delta^{out}(u)$ denote the set of incoming and outgoing edges incident to u. Our LP is the following:

$$\begin{aligned} \max |f| = \sum_{e \in \delta^{out}(s)} f_e - \sum_{e \in \delta^{in}(s)} f_e \\ f_e \leq u(e) \quad \forall e \in E \\ \sum_{e \in \delta^{in}(v)} f_e = \sum_{e \in \delta^{out}(v)} f_e \quad \forall v \in V \setminus \{s, t\} \\ f_e \geq 0 \quad \forall e \in E. \end{aligned}$$

The constraints $f_e \leq u(e)$ are a packing constraint, so these are our hard constraints (see Section 2). To simplify our presentation, we assume $u(e) = 1$ for every $e \in E$. The remaining constraints, i.e., non-negativity on each edge and conservation at each vertex, are the easy constraints.

Instead of finding an exact maximum flow, we are also given a parameter $\epsilon \in (0, 1)$, and the goal is to find a flow with value at least $(1 - \epsilon)F^*$, where F^* is the maximum flow value. With this goal in mind, we can reduce our optimization problem to a feasibility problem by encoding $|f| \geq F$ as an additional easy constraint, and binary searching over values of F to approximate F^*.

Oracle specifications: Now that we have the hard and easy constraints, we can restate the oracle specifications from Section 2 in the context of our maximum flow problem. Given a weight $w_e \geq 0$ for each $e \in E$ and a value F, the oracle must return an s-t flow f in G that satisfies the following:

1. The flow value of f must be F, $f_e \geq 0$ for every $e \in E$, and flow is conserved at every vertex.
2. The average capacity constraint is approximately satisfied, i.e., $\sum_{e \in E} w_e f_e \leq (1 + \epsilon) \sum_{e \in E} w_e$
3. The width at most ρ, that is, $f_e \leq \rho$ for every $e \in E$ and some ρ.

Electrical flow oracle: To construct such an oracle, we use the method of electrical flows (see Lecture 18). In particular, the electrical flow oracle does the following: set the resistance of each edge e to $r_e = w_e + \epsilon W/m$, where $W = \sum_{e \in E} w_e$ is the sum of all weights. Then, inject F units of current into s and remove F units from t. Finally, return the resulting electrical flow.

Theorem 2. *The electrical flow oracle satisfies the specifications above for $\rho = O(\sqrt{m/\epsilon})$.*

Proof. Let f be the electrical flow returned by the oracle given weights w_e. The easy constraints, i.e., the first specification, are automatically satisfied by f, being a feasible flow.

We now prove that f satisfies the average capacity constraint. Let F_e denote the flow value on e of the maximum (not necessarily electrical) s-t flow. Recall (from Lecture 18) that among all flows of value F, the electrical flow, i.e., f, has the least energy. In other words,

$$\sum_{e \in E} r_e f_e^2 = \sum_{e \in E} \left(w_e + \frac{\epsilon W}{m} \right) f_e^2 \leq \sum_{e \in E} \left(w_e + \frac{\epsilon W}{m} \right) F_e^2 \leq (1 + \epsilon) \sum_{e \in E} w_e, \tag{3}$$

where the final inequality holds because $F_e \leq u(e) = 1$ for every $e \in E$. We also have $w_e \leq r_e$ for every $e \in E$, so combining this with (3) yields

$$\sum_{e \in E} w_e f_e^2 \leq (1+\epsilon) \sum_{e \in E} w_e \tag{4}$$

Recall that we want to show $\sum_e w_e f_e \leq \sum_e w_e$. Observe the following:

$$\left(\sum_{e \in E} w_e f_e \right)^2 \leq \sum_{e \in E} w_e f_e^2 \sum_{e \in E} w_e \leq (1+\epsilon) \left(\sum_{e \in E} w_e \right)^2,$$

where the first inequality follows from Cauchy-Schwarz and the second follows from (4). Taking the square root of both sides gives us the desired result.

We now show that the width of the oracle is $O(\sqrt{m/\epsilon})$ by showing $f_e^2 = O(m/\epsilon)$ for every $e \in E$. From (3), for any edge $e \in E$, we have

$$f_e^2 \leq \frac{(1+\epsilon)W}{r_e} \leq \frac{(1+\epsilon)m}{\epsilon} = O(m/\epsilon),$$

where the second inequality follows from $r_e \geq \epsilon W/m$. □

We can now use the electrical flow oracle, whose width is bounded by Theorem 2, as part of Algorithm 1 to prove the following result for approximate maximum flow, due to Christiano et al. [CKM+11]. Note that our result is a simplification of their main algorithm, which runs in time $\tilde{O}(m^{4/3}/\epsilon^3)$, where $\tilde{O}(f(m))$ denotes $O(f(m) \log^c f(m))$ for some constant c.

Corollary 3. *There exists an algorithm that computes a $(1-\epsilon)$-approximate maximum flow in time $\tilde{O}(m^{3/2}\epsilon^{-5/2})$.*

Proof. As described above, we can reduce our problem to feasibility without affecting the final result. To apply Theorem 1, notice that each hard constraint corresponds to an edge, so the number of hard constraints is m. Thus, according to Theorem 1, the number of iterations we need to compute an approximate maximum flow is $T = \rho \ln m/\epsilon^2 = \tilde{O}(\rho\epsilon^{-2})$, where ρ is the width of the electrical flow oracle. Theorem 2 states $\rho = O(m^{1/2}\epsilon^{-1/2})$, which implies $T = \tilde{O}(m^{1/2}\epsilon^{-5/2})$.

In each iteration, the oracle computes an electrical flow, and it is known that this can be done in $\tilde{O}(m)$ time. Thus, the overall running time is $T \cdot \tilde{O}(m) = \tilde{O}(m^{3/2}\epsilon^{-5/2})$. □

4 Summary

In this lecture, we gave an algorithm that finds an approximately feasible solution to any packing linear program by using an oracle that satisfies certain properties and the multiplicative weight update method. We then applied this algorithm, using an oracle that finds an electrical flow, to approximately solve the maximum flow problem.

References

[CKM+11] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282. ACM, 2011.