6.S978 Graphs, Linear Algebra, and Optimization – Fall 2015 November 5, 2015

Lecture 14

Lecturer: Aleksander Mądry *Scribe: Jack Murtagh*

1 Overview

We introduce the notion of oblivious routing and describe its connection to the $(1-\varepsilon)$-approximate maximum flow algorithm we developed in the previous lecture. Then, we present a number of constructions of such oblivious routing schemes.

2 Recap of the Last Lecture

In the last lecture, we generalized the gradient descent method to make it work with an arbitrary geometry, instead of only ℓ_2-geometry. We then used an ℓ_∞-based variant of that method to design an $(1-\varepsilon)$-approximate maximum flow algorithm. The key component in that algorithm was an *α-approximate affine ℓ_∞-projection* $P(\cdot)$ onto the space $\mathcal{F}_{st}$ unit s-t-flows. In other words, $P(\cdot)$ should be:

(a) A projection onto $\mathcal{F}_{st}$, that is, $B(P(h)) = \chi_{st}$, for any h, and $P(h) = h$, whenever $h \in \mathcal{F}_{st}$.

(b) α-approximate in the ℓ_∞-norm, that is, for any h,

$$\|P(h) - h\|_\infty \leq \alpha \|\Pi(h) - h\|_\infty, \tag{1}$$

where Π is the exact ℓ_∞-projection onto $\mathcal{F}_{st}$, i.e.,

$$\Pi(h) := \arg\min_{g \in \mathcal{F}_{st}} \|g - h\|_\infty. \tag{2}$$

(c) Affine, that is, for any h, $P(h)$ should be of the form

$$P(h) = \widehat{P}h + \hat{h}, \tag{3}$$

where $\widehat{P}$ is a linear projection onto the space of circulations, i.e., of all the flows f with $Bf = 0$, and $\hat{h}$ is some fixed unit s-t flow.

As we showed last time, once we construct a projection $P(\cdot)$ then we obtain an $(1-\varepsilon)$-approximate maximum flow algorithm whose performance is described in the following theorem.

Theorem 1 *If $P(\cdot)$ is an α-approximate affine ℓ_∞-projection $P(\cdot)$ onto the space $\mathcal{F}_{st}$ of unit s-t-flows, then one can compute an $(1-\epsilon)$-approximate maximum flow in time*

$$O\left(\frac{\alpha^4}{\epsilon^2} \cdot \ln(m) \cdot (\tau(P) + m)\right)$$

where $\tau(P)$ is the time needed to apply the projection $P(\cdot)$.

In other words, the above theorem allows us to reduce fast $(1-\varepsilon)$-approximate maximum flow computations to the task of constructing a projection $P(\cdot)$ with a not too large approximation guarantee α and computation time $\tau(P)$. It is not hard to see that these two parameters are somewhat conflicted – the better approximation guarantee we need the more computationally expensive applying P should be, and vice versa. As a result, the key remaining question we need to investigate in this context is: how good of a trade-off between α and $\tau(P)$ can we achieve?

3 Oblivious Routing Schemes

The central object of our interest today will be the notion of an oblivious routing. Let us start by introducing it. Consider a graph G with m edges and n vertices. We say that an m-by-n matrix M is an *oblivious routing scheme* if, for any valid demand χ, we have that

$$BM\chi = \chi. \tag{4}$$

(Recall that a demand χ is valid if $\vec{1}^T\chi = 0$, i.e., the vertex deficits and surpluses of χ balance out.) In other words, oblivious routing scheme is any matrix M that maps demand vectors χ to flows $M\chi$ in G with that demand pattern χ. We will sometime refer to the flow $M\chi$ as the *routing of χ (wrt M).*

The crucial aspect of this definition is that M is a matrix, that is, M is a *linear* operator. Consequently, if we use M to route in G a collection of demands $\{\chi_i\}_i$ simultaneously, then each of the resulting routings $M\chi_i$ is determined solely by its individual demand pattern χ_i and thus it is completely *oblivious* to the routings of all the other demands.

3.1 Competitiveness of Oblivious Routing Schemes

Now, as in most routing tasks, our goal is to design oblivious routing schemes that minimize congestion. That is, for every collection of demands $\{\chi_i\}_i$, we want the maximum edge congestion induced by routing each one of these demands via our oblivious routing scheme M be as small as possible.

It is not hard to see that the linearity of oblivious routing schemes prevents them from delivering optimal congestion minimization. In general, a routing scheme that simply outputs some minimum-congestion routing for each input demand *cannot* be linear. Also, the knowledge of what all the demands to be routed are enables one to make better choices on how each of these demands should be routed so as to minimize the final congestion.

Still, even if we can't achieve the optimal congestion minimization, what is that we *can* achieve? To answer this question, let us define, for a given oblivious routing scheme M, its *competitiveness*, or *competitive ratio* $\beta(M)$, as

$$\beta(M) := \max_{\{\chi_i\}_i} \frac{\|\sum_i \mathrm{abs}(M\chi_i)\|_\infty}{\mathrm{OPT}(\{\chi_i\}_i)}, \tag{5}$$

where $\mathrm{abs}(v)$, for a given m-dimensional vector v, is an m-dimensional vector with $\mathrm{abs}(v)_i := |v_i|$, for each i, and $\mathrm{OPT}(\{\chi_i\}_i)$ is the maximum edge congestion induced by the (non-oblivious) optimal routing of all the demands χ_i. Observe that, for an edge e,

$$\left(\sum_i \mathrm{abs}(M\chi_i)\right)_e = \sum_i |(M\chi_i)_e|$$

is simply the total congestion on edge e induced by all the routings $M\chi_i$. (We take absolute value of each edge flow to avoid "canceling out" of routings flowing in opposite directions.)

So, $\beta(M)$ tells us how much worse the maximum edge congestion of the routing provided by the oblivious routing scheme M be compared to the optimal (non-oblivious) routing. One should view $\beta(M)$ as a measure of the "quality" of the oblivious routing scheme M.

The above definition is capturing exactly what we wanted to measure. However, computing this quantity for a given oblivious routing scheme M might be problematic. In principle, it would require us to enumerate M's performance on infinitely many collections of demands $\{\chi_i\}_i$.

Fortunately, there is an alternative characterization of competitiveness that makes computing it much more tractable.

Lemma 2 *For any oblivious routing scheme M, we have that*

$$\beta(M) = \|MB\|_\infty$$

Note that the matrix MB is an m-by-m matrix that maps a flow f to an oblivious routing MBf of the demand Bf of that flow. That is, $BMBf = Bf$. Also, the $\|\cdot\|_\infty$ norm in the statement of the lemma is an ℓ_∞-*operator norm* (or, more precisely, an ℓ_∞-ℓ_∞-*operator norm*) defined, for a matrix A, as

$$\|A\|_\infty := \max_{v\neq\vec{0}} \frac{\|Av\|_\infty}{\|v\|_\infty}. \tag{6}$$

Computing such ℓ_∞-operator norm of a matrix is fairly straightforward – it corresponds to computing the maximum ℓ_1-norm of a row. As a result, for any oblivious routing scheme M, we have that

$$\beta(M) = \|MB\|_\infty = \max_e |(MB)_e|_1 = \max_e \mathrm{load}_M(e), \tag{7}$$

where

$$\mathrm{load}_M(e) := \sum_{e'} |(MB)_{ee'}| = \sum_{e'} |(M\chi_{e'})_e| \tag{8}$$

is the sum of all the values of flows that pass through edge e in the routings $M\chi_{e'}$ of a unit flow between the endpoints of each edge e'. (Note that as we care only about values here, it does not matter which one of the endpoints of edge e' is the source and which one is the sink.)

Proof For a given collection of demands $\Xi = \{\chi_i\}_i$, let us define $\widehat{\Xi}$ to be a (larger) collection of demands constructed as follows. Let h^i, for each i, be the routing of demand χ_i in the optimal (non-oblivious) routing of the collection of demands Ξ. For each h^i and each edge e, let η^i_e be the demand vector that encodes exactly the value and direction of the flow h^i_e that passes through e in the routing h^i. We take $\widehat{\Xi}$ to be the union of all such demands η^i_e for all e and i. (Note that if $h^i_e = 0$ for some edge e and commodity i then the corresponding demand is a trivial zero demand.)

Observe that $\mathrm{OPT}(\Xi) = \mathrm{OPT}(\widehat{\Xi})$. This is so as, by construction, the optimal (non-oblivious) routings $\{h^i\}_i$ of Ξ is also a valid routing of the collection of demands in $\widehat{\Xi}$. Additionally, any routing of all the demands in $\widehat{\Xi}$ is also a valid routing of demands in Ξ.

Furthermore, routing the demands $\widehat{\Xi}$ via an oblivious routing scheme M cannot lead to a lower congestion than routing the demands Ξ. That is, we must have that

$$\left\|\sum_i \mathrm{abs}(M\chi_i)\right\|_\infty \leq \left\|\sum_i\sum_e \mathrm{abs}(M\eta^i_e)\right\|_\infty,$$

since M cannot take advantage of any flow cancellations when routing $\widehat{\Xi}$.

It is not hard to see that the above observation tells us that the worst-case competitiveness of an oblivious routing scheme M is achieved for collections of demands that: correspond to routings flows directly between edge endpoints; and for which simply routing each of these edge demands across the corresponding routing is an optimum (non-oblivious) routing[1]. In other words, we have that

$$\beta(M) \quad = \quad \max_{\{\chi_i\}_i} \frac{\|\sum_i \mathrm{abs}(M\chi_i)\|_\infty}{\mathrm{OPT}(\{\chi_i\}_i)} = \max_{x\in\mathbb{R}^m} \frac{\|\sum_e \mathrm{abs}(M(x_e\chi_e))\|_\infty}{\|x\|_\infty}$$

where $x_e\chi_e$ is a demand vector that requires routing $|x_e|$ units of flow between the endpoints of the edge e. (The sign of x_e encodes whether the direction of routing is aligned with or opposite to the orientation of edge e.)

We thus have that

$$\begin{aligned}\beta(M) \quad &= \quad \max_{x\in\mathbb{R}^m} \frac{\|\sum_e \mathrm{abs}(M(x_e\chi_e))\|_\infty}{\|x\|_\infty} = \max_{x\in\mathbb{R}^m} \frac{\|\sum_e |x_e|\,\mathrm{abs}(M\chi_e)\|_\infty}{\|x\|_\infty} \\ &= \quad \max_{x\in\mathbb{R}^m} \frac{\|\sum_e |x_e|\,\mathrm{abs}((MB)^e)\|_\infty}{\|x\|_\infty} \overset{(*)}{=} \max_{x\in\mathbb{R}^m} \frac{\|\sum_e (MB)^e x_e\|_\infty}{\|x\|_\infty} \\ &= \quad \max_{x\in\mathbb{R}^m} \frac{\|MBx\|_\infty}{\|x\|_\infty} = \|MB\|_\infty,\end{aligned}$$

[1]Note that even though such a routing strategy is very simple, in general graphs, this strategy *cannot* be implemented by oblivious routing schemes due to their linearity in the demand vectors.

where the equality $(*)$ follows as for the worst-case x the signs of its coordinate will be such that they prevent any cancellations on the coordinate of the vector $\sum_e (MB)^e x_e$ that achieves the value of $\|\sum_e |x_e| \operatorname{abs}((MB)^e)\|_\infty$. ■

3.2 Oblivious Routings and Affine ℓ_∞-projections

As we already mentioned above, the obliviousness of an oblivious routing scheme M comes at a price of its worse performance, as captured by its competitive ratio $\beta(M)$ – see (5). Given this inherent "price of obliviousness", why would we even want to use oblivious routing schemes? Why not just always use the optimal non-oblivious routings instead? After all, computing such an optimal routing corresponds to solving a multi-commodity flow problem, a task that can be solved fairly quickly.

There is a couple of reasons. First of all, even though multi-commodity flow algorithms are pretty fast, they are not fast enough to handle very large graphs. Deploying a compact and efficient oblivious routing scheme might be preferable in such settings, even if it leads to sub-optimal answers. More importantly, obliviousness is an enormous advantage in scenarios when the demands are changing frequently (which is the case, for example, in most networking applications). Oblivious routing scheme allows us to avoid recomputing all the routings from a scratch each time a new demand arrives or an existing one changes. We only need to add/update a single routing per each such event. Also, the structure of oblivious routing schemes lends itself very well to distributed nature of many routing tasks.

It turns out that there is also one more reason we should be interested in oblivious routing schemes: they allow us to construct α-approximate affine ℓ_∞-projections such as the projection $P(\cdot)$ that we need for our $(1-\varepsilon)$-approximate maximum flow algorithms (see Section 2). Specifically, given an oblivious routing scheme M, let us define

$$\widehat{P}_M := I - MB \tag{9}$$

and

$$\hat{h}_M := M\chi_{st}. \tag{10}$$

Then, for a flow h, let us define

$$P_M(h) := \widehat{P}_M(h) + \hat{h}_M. \tag{11}$$

We have the following lemma.

Lemma 3 *For any oblivious routing scheme M, the operator $P_M(\cdot)$ is a $\beta(M)$-approximate affine ℓ_∞-projection onto the space $\mathcal{F}_{st}$ of unit s-t-flows. Also, the time $\tau(P_M)$ needed to compute this projection is at most $O(\tau(M)+m)$, where $\tau(M)$ is the maximum time needed to compute the routing $M\chi$, for any demand χ.*

In the light of the above lemma, from now on, instead of trying to construct a projection $P(\cdot)$ for our $(1-\varepsilon)$-approximate maximum flow algorithm (see Theorem 1) explicitly, we can focus on constructing an oblivious routing scheme M with sufficiently small competitiveness $\beta(M)$ and computation time $\tau(M)$.

Proof Note that, for any flow h, we have that

$$B\widehat{P}_M h = B(h - MBh) = Bh - BMBh = 0.$$

So, as trivially $\widehat{P}_M 0 = 0$, $\widehat{P}_M$ is a (linear) projection on th space of all circulations. Also, by definition of M, $\hat{h}_M$ is a unit s-t-flow.

As a result, for any h, $P_M(h)$ is a unit s-t flow and, for any h that is a unit s-t flow already, we have that

$$P_M(h) = \widehat{P}_M(h) + \hat{h}_M = h - MBh + M\chi_{st}) = h - M(Bh - \chi_{st}) = h.$$

Thus, $P_M(\cdot)$ is an affine projection onto the space $\mathcal{F}_{st}$ of all the unit s-t-flows. (See Section 2.)

Next, we should observe that, for any h, it is the case that

$$\begin{aligned}\|P_M(h) - h\|_\infty &= \|h - M(Bh - \chi_{st}) - h\|_\infty = \|M(Bh - \chi_{st})\|_\infty \\ &\leq \beta(M) \cdot \mathrm{OPT}(\{Bh - \chi_{st}\}) = \beta(M)\|\Pi(h) - h\|_\infty,\end{aligned}$$

where we used the definition (5) of competitiveness $\beta(M)$ and the fact that, by definition of OPT and Π (see (2)),

$$\mathrm{OPT}(\{Bh - \chi_{st}\}) = \min_{h' : Bh' = Bh - \chi_{st}} \|h'\|_\infty = \min_{h'' \in \mathcal{F}_{st}} \|h'' - h\|_\infty = \|\Pi(h) - h\|_\infty.$$

In other words, $P_M(\cdot)$ is indeed a $\beta(M)$-approximate ℓ_∞-projection (see (1)).

Finally, the time $\tau(P_M)$ needed to compute $P_M(h)$, for any flow h, is at most

$$O(\tau(MB) + m) = O(\tau(M) + \tau(B) + m) = O(\tau(M) + m),$$

as desired. ■

4 Constructions of Oblivious Routing Schemes

Once we demonstrated usefulness and properties of oblivious routing schemes, it is time to explore the next natural question: how to construct them?

4.1 Oblivious Routing with Electrical Flows

Although we were not aware of that then, we already studied one oblivious routing scheme construction. It turns out that routing with electrical flows is oblivious! More precisely, let us define

$$M_{\mathcal{E}} := B^T L^+, \tag{12}$$

where L^+ is a pseudo-inverse of the Laplacian matrix of the underlying graph wrt all resistances being one. (See notes from Lecture 4 for all the relevant definitions and background.) Note that $M_{\mathcal{E}}$ corresponds to a routing in which each demand χ is routed by computing an electrical flow with this demand wrt unit resistances.

Clearly, for such a oblivious routing scheme we have that

$$\tau(M_{\mathcal{E}}) = O(m + \tau(L^+)) = \widetilde{O}(m),$$

where we used the fact that applying the pseudo-inverse L^+ corresponds to solving a Laplacian linear system and the latter task can be performed in nearly-linear time. (See notes from Lecture 8.)

The routing time $\tau(M_{\mathcal{E}})$ is very much acceptable, but how about the competitiveness $\beta(M_{\mathcal{E}})$ of that scheme? By simple use of the machinery we developed in Lecture 11 for solving the maximum flow problem via electrical flows, we can establish the following bound.

Lemma 4 *For any graph G with m edges, we have that $\beta(M_{\mathcal{E}}) \leq \sqrt{m}$.*

Proof By Lemma 2 and (7), we know that

$$\beta(M_{\mathcal{E}}) = \|M_{\mathcal{E}} B\|_\infty = \max_e \mathrm{load}_{M_{\mathcal{E}}}(e).$$

Note that, for a fixed edge e, we then have that, by (8),

$$\mathrm{load}_{M_{\mathcal{E}}}(e) = \sum_{e'} |(M_{\mathcal{E}} B)_{ee'}| = \sum_{e'} |(B^T L^+ B)_{ee'}| = \sum_{e'} |\chi_e L^+ \chi_{e'}|.$$

Observe now that, since all resistances are uniform, $|\chi_e^T L^+ \chi_{e'}|$ is the amount of flow passing through the edge e in an electrical flow that sends one unit of flow from one to the other endpoint of e'. However, as the pseudo-inverse L^+ is symmetric, $|\chi_e^T L^+ \chi_{e'}| = |\chi_{e'}^T L^+ \chi_e|$, i.e., $|\chi_e^T L^+ \chi_{e'}|$ is also the amount of flow that flows through edge e' in an electrical flow that sends one unit of flow from one endpoint of e to the other one.

Consequently, we can use Cauchy-Schwarz inequality to conclude that

$$\text{load}_{M_{\mathcal{E}}}(e) = \sum_{e'} |\chi_e^T L^+ \chi_{e'}| = \sum_{e'} |\chi_{e'}^T L^+ \chi_e| \leq \sqrt{m \sum_{e'} (\chi_{e'}^T L^+ \chi_e)^2} = \sqrt{m\mathcal{E}_e},$$

where $\mathcal{E}_e$ denotes the energy of a unit electrical flow between the endpoints of the edge e. Since routing this one unit of flow directly over e would result in a flow of energy 1, we need to have that $\mathcal{E}_e \leq 1$ and, as a result of all the above observations, we obtain that

$$\beta(M_{\mathcal{E}}) = \max_e \text{load}_{M_{\mathcal{E}}}(e) \leq \max_e \sqrt{m\mathcal{E}_e} \leq \sqrt{m},$$

which concludes our proof. ■

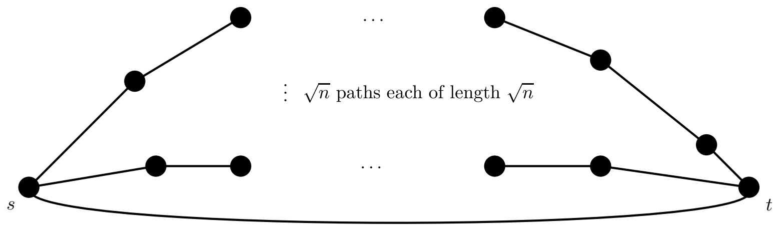

Figure 1: An example that shows that our bound on $\beta(M_{\mathcal{E}})$ is essentially tight in sparse graphs.

It turns out that the above upper bound is fairly tight, especially in sparse graphs. In particular, let us consider the graph presented in Figure 1. (An attentive reader might recall that this is exactly the graph we used in Lecture 11 to show how different electrical flows and maximum flow can be.) If we consider a demand vector χ_{st} that sends a unit of flow from the leftmost vertex to the rightmost one then the congestion of the routing $M_{\mathcal{E}}\chi_{st}$ will be roughly constant, while the optimum routing achieves a congestion of roughly $\frac{1}{\sqrt{n}}$. So, $\beta(M_{\mathcal{E}}) = \Omega(\sqrt{n})$, which is also $\Omega(m)$ in this graph.

Still, despite seeing the above example, one should not discard electrical flow–based oblivious routing scheme too easily. After all, the competitiveness bound $\beta(M)$ is a worst-case bound and, more importantly, this worst-case bound is not only over all the possible collections of demands we want to route (which is reasonable) but also – implicitly – over all the underlying graphs. As a result, even though there exist graphs in which the oblivious routing scheme $M_{\mathcal{E}}$ performs poorly, there are families of graphs – most notably, expanders – where $M_{\mathcal{E}}$ is *guaranteed* to perform very well. (This fact will be important later on.)

4.2 Oblivious Routing with Spanning Trees

Another way to construct an oblivious routing scheme is to base it on spanning trees. Specifically, let us fix a spanning tree T of our graph G and consider an oblivious routing scheme M_T that simply routes all the demands in that tree. (Note that if we insist on routing only over the edges of T then there is a unique mapping of demand χ to its routing $M_T\chi$, and this mapping is linear.)

Such a tree-based scheme is conceptually very simple and we have that $\tau(M_T) = O(n)$, which is the best bound possible. But, what is it competitiveness $\beta(M_T)$?

It is not hard to see that, for any edge e, we have that

$$\text{load}_{M_T}(e) = |\{e' \in E(G) \mid e \in \text{path}_T(e')\}|, \tag{13}$$

where $\text{path}_T(e')$ is the unique path in T between the endpoints of e'. (Note that if e is not in the tree then $\text{load}_T(e) = 0$.) Another way to view the quantity $\text{load}_{M_T}(e)$ is to realize that for a given tree edge e and edge e', $e \in \text{path}_T(e')$ iff e' is in the cut definite by the two connected components of T that removal of e from T creates. We will sometime refer to this cut as the *canonical cut* of e.

Consequently, by Lemma 2 and (7), we immediately obtain the following bound on $\beta(M_T)$.

Fact 5 *For any spanning tree T, we have that $\beta(M_T) = \max_e \text{load}_T(e) \leq m$.*

It is not hard to show that the overall upper bound of m is essentially tight in the worst case. In particular, one can prove that for any expander graph and *any* tree T, there always exists a tree edge e^* such that its canonical cut contains $\Omega(n)$ edges. This implies that

$$\max_e \text{load}_T(e) \geq \text{load}_T(e^*) = \Omega(n) = \Omega(m),$$

as expanders are sparse. Also, by starting from a complete graph and then sub-sampling its edges at an appropriate rate, one can extend this lower bound to all density regimes.

So, oblivious routing schemes that are based on a single spanning tree, despite their simplicity, have very unsatisfying worst-case performance.

4.3 Oblivious Routing with a Convex Combination of Spanning Trees

The poor quality of the tree-based oblivious routing schemes we analyzed in the previous section should not come as a surprise. A single spanning tree–based routing is just too simplistic to accurately capture the complex flow structure of general graphs.

However, there is a way to make these simplistic routing strategies dramatically more powerful and useful to us. We just need to avoid sticking to only one such tree-based routing scheme. Instead we should use a *convex combination* of them, with each scheme in this convex combination corresponding to a *different* tree. (Such surprising expressive power of convex combinations of simple objects is a quite widespread phenomena in optimization, machine learning, and theoretical computer science. Probably the most well-known example here is the boosting technique[2] in machine learning.)

Formally, let us consider a convex combination $\Gamma = \{(\lambda_i, T_i)\}_i$, where $\lambda \in \Delta_{|\mathfrak{T}|}$ are the coefficients, each λ_i corresponds to a different spanning tree $T_i \in \mathfrak{T}$. (Here, $\mathfrak{T}$ is the set of all spanning trees of the graph G.) Let us then define an oblivious routing scheme M_Γ as

$$M_\Gamma := \sum_i \lambda_i M_{T_i}, \tag{14}$$

where each M_{T_i} is an oblivious routing scheme based on a single spanning tree T_i, as described in the previous section. In other words, M_Γ constructs the routing $M_\Gamma \chi$ of a given demand vector χ by outputting an *average* (weighted by the vector λ) of the routings $M_{T_i}\chi$ corresponding to all the trees T_i in the support of Γ.

Now, using Lemma 2 and (7) as well as a straightforward adjustment of the analysis we performed in the previous section for the single spanning tree case, we can immediately conclude that

$$\beta(M_\Gamma) = \max_e \sum_i \lambda_i \text{load}_{T_i}(e), \tag{15}$$

where $\text{load}_{T_i}(e)$ is a shorthand notation for $\text{load}_{M_{T_i}}(e)$, as defined in (13). That is, the competitiveness $\beta(M_\Gamma)$ of the oblivious routing scheme M_Γ is equal to the maximum *average* load of any edge.

4.3.1 Bounding $\beta(M_\Gamma)$

How large can $\beta(M_\Gamma)$ be? Clearly, we know that if we consider only convex combinations Γ that are supported on a single tree then $\beta(M_\Gamma) = \Omega(m)$ in the worst-case. But the whole point here is to make Γ be supported on many trees. How much can this help us in driving $\beta(M_\Gamma)$ down?

To answer this question, let us phrase it first as a feasibility problem. Specifically, consider the following set of constraints:

$$\mathcal{M}(\beta) : \sum_i \lambda_i \text{load}_{T_i}(e) \leq \beta, \ \forall_e$$
$$\lambda \in \Delta_{|\mathfrak{T}|},$$

[2]Interestingly, the general principle behind boosting is very similar to what we will be doing here. In particular, both these techniques can be explained via multiplicative weights update framework. Unfortunately, we will not have time to flesh out this connection further.

where λ is the vector of variables and β is a free parameter.

Observe that if we find some λ^* that satisfies this set of constraints then, by (15), the corresponding convex combination $\Gamma^* = \{(\lambda_i^*, T_i)\}$ gives rise to an oblivious routing scheme M_{Γ^*} with

$$\beta(M_{\Gamma^*}) \leq \beta.$$

In other words, we can reduce the task of bounding the competitiveness of oblivious routing schemes that are based on convex combinations of trees to analyzing feasibility of the constraint set $\mathcal{M}(\beta)$ for different values of β.

To tackle the latter task, we will use the multiplicative weight update method–based framework for solving feasibility problems that we developed in Lecture 10. (See the notes from that lecture for necessary definitions and background.)

To apply this framework, let us first cast the constraint set $\mathcal{M}(\beta)$ in the following equivalent form

$$\begin{aligned} \mathcal{M}(\beta):\ & A\lambda \leq \beta \cdot \vec{1} \\ & \lambda \in \Delta_{|\mathfrak{T}|}, \end{aligned}$$

where A is an m-by-$|\mathfrak{T}|$ matrix defined as

$$A_{eT} := \mathrm{load}_T(e). \tag{16}$$

Next, we want to design an (θ, ρ)-oracle for that feasibility problem $\mathcal{M}(\beta)$. Our plan here will be to analyze what lower bound β^* on the value of β would ensure that, for any $\varepsilon > 0$, this oracle *never* fails, when used to find an ε-approximately feasible solution to the set of constraints $\mathcal{M}(\beta)$. Clearly, once we are able to find such β^*, we will know that $\mathcal{M}(\beta)$ is always feasible whenever $\beta \geq \beta^*$. This, in turn, will mean that there always exists a convex combination Γ^* for which $\beta(M_{\Gamma^*}) \leq \beta^*$.

To realize the above plan, let us recall that the key requirement for such a (θ, ρ)-oracle for $\mathcal{M}(\beta)$ is that, as long as $\mathcal{M}(\beta)$ is feasible, this oracle is able to output "feasible on average" solutions. More precisely, given a convex combination $p \in \Delta_m$ of linear constraints corresponding to the rows of the matrix A, the oracle should output a vector $\lambda \in \Delta_{|\mathfrak{T}|}$ such that

$$p^T A\lambda \leq \beta \cdot p^T \vec{1} = \beta, \tag{17}$$

unless $\mathcal{M}(\beta)$ is not feasible, in which case outputting either $\perp$ (to indicate "failure") or λ as above is allowed. (As already mentioned, we will actually make sure that $\perp$ is never returned in our setting.)

Now, to construct our oracle it will be crucial to understand the "feasible on average" condition (17) better. To this end, suppose that our oracle, given an input $p \in \Delta_m$, has to output a vector $\lambda \in \Delta_{|\mathfrak{T}|}$ that is supported on only a *single* tree T. What should this tree T be to satisfy condition (17)?

Observe that, by the definition (16) of A and the definition (13) of $\mathrm{load}_T(e)$, we have that

$$p^T A\lambda = \sum_e p_e \mathrm{load}_T(e) = \sum_e \sum_{e' : e \in \mathrm{path}_T(e')} p_e.$$

By changing the order of sums, we can rewrite the last expression as

$$\sum_e \sum_{e' : e \in \mathrm{path}_T(e')} p_e = \sum_{e'} \sum_{e \in \mathrm{path}_T(e')} p_e = \sum_{e'} p(\mathrm{path}_T(e')) = \sum_{e'} \mathrm{stretch}_T(e') p_{e'},$$

where $p(\mathrm{path}_T(e'))$ is the length of the unique path joining the endpoints of e' in the tree T wrt edge lengths given by p_es, and $\mathrm{stretch}_T(e') := \frac{p(\mathrm{path}_T(e'))}{p_{e'}}$ is the corresponding stretch of edge e' in tree T (see notes from Lecture 7).

So, in the light of the above, the "feasibility on average" condition (17) can be expressed as

$$p^T A\lambda = \sum_{e'} \mathrm{stretch}_T(e') p_{e'} \leq \beta.$$

We want now to choose the tree T that will make the left-hand side of this inequality as small as possible (and thus satisfy the condition (17) with as small β as possible). What should our choice of T be?

Note that the expression

$$\sum_{e'} \mathrm{stretch}_T(e') p_{e'}$$

can be viewed as the expected stretch of an edge e when sampling e to be e' with probability $p_{e'}$. Consequently, it is tempting to make T be low-stretch, as described in the theorem below. (Also, see the notes from Lecture 7.)

Theorem 6 (Low-stretch Spanning Trees) *For any graph G with m edges, one can construct in $\widetilde{O}(m)$ time a spanning tree T of G such that*

$$\frac{\sum_{e'} \mathrm{stretch}_T(e')}{m} = \widetilde{O}(\log n).$$

A low-stretch spanning tree as above is not exactly what we need though. Observe that the low-stretch property bounds the *average* stretch, i.e., expected stretch for the case of all $p_{e'} = \frac{1}{m}$, which might be very different to the expected stretch when $p_{e'}$s are very non-uniform. Fortunately, there is a simple way to alleviate this shortcomings and compute the tree that we need.

Lemma 7 *For any graph G with m edges and any $p \in \Delta_m$, one can construct in $\widetilde{O}(m)$ time a spanning tree T of G such that*

$$\sum_{e'} \mathrm{stretch}_T(e') p_{e'} = \widetilde{O}(\log n).$$

Clearly, in the light of the above lemma, setting $\beta \geq \beta^* := \widetilde{O}(\log n)$ suffices to ensure that the "feasibility on average" condition (17) is satisfied. By our discussion above, we can conclude that there always exists a convex combination of spanning trees Γ^* such that

$$\beta(M_{\Gamma^*}) \leq \beta^* = \widetilde{O}(\log n).$$

So, basing our oblivious routing scheme on a convex combination of spanning trees, instead of a single spanning tree, improved competitiveness exponentially, from $\Theta(m)$ to $\widetilde{O}(\log n)$!

Proof Given the graph G and the convex combination $p \in \Delta_m$, consider a multigraph $\widehat{G}_p$ that is over the same vertex set as the graph G and, for each edge e in G, it has $\lceil p_e m \rceil$ copies of that edge. (Intuitively, graph $\widehat{G}_p$ emphasizes edges e with unusually large value of p_e by including many copies of that edge.) Note that the number $\widehat{m}$ of edges of $\widehat{G}_p$ is at most

$$\widehat{m} = \sum_{e \in E(G)} \lceil p_e m \rceil \leq \sum_{e \in E(G)} (p_e m + 1) = 2m.$$

Now, let us use Theorem 6 to find a low-stretch spanning tree T for $\widehat{G}_p$. By our construction, T is also a spanning tree of the original graph G and the stretch of an edge e in G wrt T is exactly the same as it is in $\widehat{G}_p$. We thus have that

$$\sum_{e' \in E(G)} \mathrm{stretch}_T(e') p_{e'} \leq \sum_{e' \in E(G)} \mathrm{stretch}_T(e') \cdot \frac{2\lceil p_{e'} m \rceil}{\widehat{m}} = 2 \cdot \frac{\sum_{e \in E(\widehat{G}_p)} \mathrm{stretch}_T(e)}{\widehat{m}} \leq \widetilde{O}(\log n),$$

where the last inequality follows as T was a low-stretch spanning tree for $\widehat{G}_p$. Also, as $\widehat{m} \leq 2m$, finding T can be performed in $\widetilde{O}(\widehat{m}) = \widetilde{O}(m)$ time. ■

4.3.2 Bounding $\tau(M_\Gamma)$

We just established that for every graph G there exists a convex combination $\Gamma^* = \{(\lambda_i^*, T_i)\}_i$ of spanning trees such that $\beta(M_{\Gamma^*}) \leq \beta^* = \widetilde{O}(\log n)$. This is a dramatic improvement over the $\sqrt{m}$ and $O(m)$ bounds we obtained for electrical flow–based and single spanning tree–based oblivious routing schemes, respectively. What is the construction and application time $\tau(M_{\Gamma^*})$ of such scheme though? Can we still match the, essentially best possible, time bounds for the previous two schemes?

In principle, in the previous section we argued only that the convex combination Γ^* exists, without saying anything about computing it. Still, the nature of our argument was algorithmic. Specifically, if we set our error parameter ε to be $\frac{1}{2}$ then our multiplicative weights update method–based framework will use our (θ, ρ)-oracle to find a solution $\lambda' \in \Delta_{|\mathfrak{T}|}$ with

$$A\lambda' \leq \beta^* \cdot \vec{1} + \varepsilon \cdot \vec{1} = (\beta^* + \frac{1}{2}) \cdot \vec{1}.$$

In other words, it will find a convex combination $\Gamma' := \{(\lambda_i', T_i)\}_i$ such that the resulting oblivious routing scheme $M_{\Gamma'}$ will have competitiveness $\beta(M_{\Gamma'})$ of at most

$$\beta(M_{\Gamma'}) \leq \beta^* + \varepsilon \leq \beta^* + \frac{1}{2} = \widetilde{O}(\log n).$$

Furthermore, from the analysis we performed in Lecture 10 (see Theorem 3 in the notes from that lecture), the number of oracle calls needed to compute such λ' will be at most

$$O(\theta\rho\varepsilon^{-2}\log m) = \widetilde{O}(\theta\rho), \tag{18}$$

where θ and ρ are such that, for any $p \in \Delta_m$, if $\lambda(p) \in \Delta_{|\mathfrak{T}|}$ is the convex combination returned by our oracle in response to input p then, for each edge e,

$$[-\theta, \rho] \ni (A\lambda(p))_e - \beta^* = \left(\sum_i \lambda(p)_i \mathrm{load}_{T_i}(e)\right) - \beta^* = \mathrm{load}_{T(p)}(e) - \beta^*,$$

where we used the fact that our oracle always returns a convex combination $\lambda(p)$ that is supported on a single tree $T(p)$.

As $\mathrm{load}_{T(p)}(e)$ has to be always non-zero, we have that $\theta \leq \beta^*$. On the other hand, we can bound ρ as follows

$$\rho \leq \max_{p\in\Delta_m} \max_e \mathrm{load}_{T(p)}(e) - \beta^* \leq \max_{T\in\mathfrak{T}} \max_e \mathrm{load}_T(e) \leq m.$$

So, our bound (18) becomes

$$\widetilde{O}(\theta\rho) = \widetilde{O}(\beta^* m) = \widetilde{O}(m),$$

and we can conclude that

$$\tau(M_{\Gamma'}) \leq \widetilde{O}(m^2),$$

where we used the fact that, by Lemma 7, each oracle call takes $\widetilde{O}(m)$ time. Furthermore, recall that each of our oracle calls provided a convex combination $\lambda(p)$ that is supported only on a singe tree, and the final convex combination λ' is the average of all these returned single tree–supported $\lambda(p)$. Thus, we can conclude that there is at most $\widetilde{O}(m)$ different spanning trees in the support of the convex combination Γ' that we compute.

Finally, let us remark that having $\tau(M_{\Gamma'})$ be quadratic in the graph size is hardly satisfying. Especially, in the context of applying this oblivious routing scheme to our maximum flow algorithm – see Lemma 3. We thus will still need to address this problem in the future.