Electrical Flows for Polylogarithmic Competitive Oblivious Routing

Gramoz Goranci
Faculty of Computer Science, University of Vienna, Austria

Monika Henzinger
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Harald Räcke
Technical University Munich, Germany

Sushant Sachdeva
University of Toronto, Canada

A. R. Sricharan
Faculty of Computer Science, UniVie Doctoral School Computer Science DoCS,
University of Vienna, Austria

Abstract

Oblivious routing is a well-studied paradigm that uses static precomputed routing tables for selecting routing paths within a network. Existing oblivious routing schemes with polylogarithmic competitive ratio for general networks are tree-based, in the sense that routing is performed according to a convex combination of trees. However, this restriction to trees leads to a construction that has time quadratic in the size of the network and does not parallelize well.

In this paper we study oblivious routing schemes based on electrical routing. In particular, we show that general networks with n vertices and m edges admit a routing scheme that has competitive ratio $O(\log^2 n)$ and consists of a convex combination of only $O(\sqrt{m})$ electrical routings. This immediately leads to an improved construction algorithm with time $\tilde{O}(m^{3/2})$ that can also be implemented in parallel with $\tilde{O}(\sqrt{m})$ depth.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases oblivious routing, electrical flows

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.55

Related Version *Full Version*: arXiv:2303.02491

Funding *Monika Henzinger and A. R. Sricharan*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564) and the Austrian Science Fund (FWF) project Z 422-N, project I 5982-N, and project P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.
Harald Räcke: Research supported by German Research Foundation (DFG), grant 470029389 (FlexNets), 2021-2024.
Sushant Sachdeva: SS's work is supported by an Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2018-06398 and a Sloan Research Fellowship.

1 Introduction

Oblivious routing schemes use static-precomputed routing tables for selecting routing paths instead of routing paths that adapt dynamically to the observed traffic pattern of a parallel system. While at first glance this restriction seems like a serious barrier to obtaining good

performance, it has been shown that in undirected networks oblivious routing does not only provide good theoretical guarantees [26, 13, 27], but is also an excellent choice for practical implementations [2, 18] due to its simple structure.

Formally an oblivious routing scheme provides a unit flow $f_{s,t}$ between every source-target pair (s,t) in the network. When a demand $d_{s,t}$ between s and t appears, this unit flow is scaled by the demand to provide the required flow between source and target. When using an oblivious routing scheme for packet routing the path of a packet is chosen according to the flow so that the probability that the packet takes a certain edge is equal to the flow value along that edge. The main strength of any oblivious routing algorithm stems from the fact that determining the next hop for a packet can be done via a simple table lookup after precomputing the necessary routing tables.

In this paper we consider oblivious routing algorithms that aim to minimize network *congestion*, i.e., the maximum flow on any edge of the network. Most existing oblivious routing algorithms for this congestion cost-measure are *tree based* [26, 27, 13, 28, 7]. This means a convex combination of trees is embedded into the network. A routing path between two vertices s and t is then in principle chosen by first sampling a random tree and then following the path between s and t in this tree.

Another approach for oblivious routing is to use *electrical flows*. In an electrical flow routing one assigns a *resistance* (or its inverse, which is called *conductance*) to each edge of the graph. The flow between two vertices s and t is then defined by the current that would result when adding a voltage source between s and t. Lawler and Narayanan [20] studied the performance of electrical flows as an oblivious routing scheme. They show that if every edge is assigned unit resistance, then electrical routing has a *competitive ratio* of $\min\{\sqrt{m}, O(T_{\text{mix}})\}$ against any ℓ_p-norm *simultaneously*, where the competitive ratio is the ratio of the cost of routing any demand using oblivious routing to the cost of the optimal routing for that demand, and T_{mix} is the mixing time of a random walk on the graph. Kelner and Maymounkov [17] consider electrical routing on expander graphs (again with uniform resistances) and show that this scheme achieves small competitive ratio and is quite robust under edge deletions. Schild, Rao, and Srivastava [31] study the average length of electrical flow paths, and as a consequence obtain $O(\log^2 n)$ competitive ratio for the special family of *edge-transitive* graphs. However, no non-trivial bound on the competitive ratio is known for electrical routing on general networks. This leads to the following important question:

Does there exist an electrical routing (with non-uniform *conductances) that obtains a polylogarithmic competitive ratio with respect to the congestion cost-measure?*

In this paper we give a partial answer to this question by showing how to obtain a competitive ratio of $O(\log^2 n)$ w.r.t. congestion with a *convex combination* of only $O(\sqrt{m})$ electrical flows[1].

▶ **Theorem 1.** *Given a capacitated graph $G = (V, E)$ with n vertices and m edges, there is an algorithm that finds an oblivious routing scheme composed of a convex combination of $O(\sqrt{m})$ electrical flows. The algorithm has competitive ratio $O(\log^2 n)$, runs in time $\widetilde{O}(m^{3/2})$, and can be implemented in parallel with $\widetilde{O}(\sqrt{m})$ depth.*

[1] In personal communication, Sidford and Lee [32] claimed that they had also observed that there exists an oblivious routing scheme based on a convex combination of $O(\sqrt{m})$ electrical flows that achieves $O(\log^2 n)$ competitive ratio.

More specifically, we show that our competitive ratio is proportional to the best bound on the *localization of electrical flows* (Lemma 2), which is currently shown to be $O(\log^2 n)$. Hence, any improvement on the localization bound would lead to an improvement in our competitive ratio.

In contrast to this, all existing tree based oblivious routing schemes that guarantee a polylogarithmic competitive ratio require at least $\Omega(m)$ trees[2]. The best achievable competitive ratio with tree based schemes is $O(\log n)$. Note that electrical flows in particular generalize oblivious routings that are based on a convex combination of *spanning trees*[3] (as one could simply give all edges that are not part of the spanning tree a resistance of infinity). Therefore, our work shows that going from trees to electrical flows allows us to reduce the support of the convex combination from $O(m)$ to $O(\sqrt{m})$ with a slight increase in competitive ratio.

This reduction in the support of the convex combination also has an important implication for the construction time, and in particular for the parallel depth of the construction. The state of the art for constructing tree based oblivious routing schemes (with a polylogarithmic guarantee on the competitive ratio) is the multiplicative weights update method [3]. One iteration computes a low stretch spanning tree in time $\widetilde{O}(m)$ and updates weights/distances on edges. This results in depth $\widetilde{O}(m)$ and work $\widetilde{O}(m^2)$ for computing $O(m)$ trees. Räcke, Shah and Täubig [29] obtained a near-linear time algorithm for computing a single tree flow sparsifiers with quality $O(\log^4 n)$ using the near-linear time approximate maximum flow result of Peng [24]. While their construction can be adapted to an $O(\log^4 n)$ competitive oblivious routing scheme, the best-known efficient implementations of such a scheme require at least quadratic time in the size of the network.

We use the multiplicative weights update method with electrical flows. One iteration computes a set of resistances/conductances that is good on average for the current edge-weights and updates weights for the next iteration. We show that by using matrix sketching we can implement one iteration in time $\widetilde{O}(m)$ as in the tree case. Because electrical flows minimizes the ℓ_2-squared norm of the flow values, we can show that we only require $O(\sqrt{m})$ iterations to obtain a good oblivious routing scheme. This results in depth $\widetilde{O}(\sqrt{m})$ and work $\widetilde{O}(m^{3/2})$ for computing the convex combination of electrical flows[4].

We show that our techniques also apply to oblivious routing measured with respect to the ℓ_1-norm of loads on the edges, where we obtain a competitive ratio of $O(\log^2 n)$ with $O(\sqrt{m})$ electrical flows. See Section 6 for details.

Related work

The study of oblivious routing was initiated by Valiant and Brebner [35], who developed an oblivious routing protocol for routing in the hypercube that routes any permutation in time that is only a logarithmic factor away from optimal. For the cost measure of minimizing congestion, Räcke [26] proved the existence of an oblivious routing scheme with

[2] The routing schemes in [26] and [28] are based on a single tree with different embeddings into the network; for this discussion, this is viewed as several trees.

[3] In general tree based routings may embed trees that contain Steiner nodes, i.e., vertices that are not part of the network, and they may also use arbitrary paths to connect embedded vertices.

[4] Note that computing the routing tables for quickly determining the outgoing edge from the header of a packet adds additional work for tree-based and electrical-flow-based routing. For tree based routing this can be done with work $\widetilde{O}(n^2)$ per tree and depth $\widetilde{O}(1)$. For electrical flow based routing the work is $\widetilde{O}(mn)$ per flow with depth $\widetilde{O}(1)$. See Section 7.

polylogarithmic competitive ratio for any undirected network. This result was subsequently improved to a competitive ratio of $O(\log^2 n \log\log n)$ [13], and then to a competitive ratio of $O(\log n)$ [27], which is optimal. Englert and Räcke [7] extend these results to oblivious routing when the cost-measure is the ℓ_p-norm of the edge congestions.

Oblivious routing by electrical flows was first considered by Harsha et al. [14] for the goal of designing oblivious routing schemes that minimize the cost-measure $\|\cdot\|_2^2$, also known as *average latency*, for the case of a single target. In this scenario, using electrical flows is a very intuitive approach as it minimizes the $\|\cdot\|_2^2$ of the flow. Specifically, in their algorithm, every source in the oblivious routing scheme optimizes its flow as if no other source was active. They show that this gives a competitive ratio of $O(\log n)$ for the ℓ_2-norm squared of the flow.

Electrical flows also play a major role as a tool for speeding up flow computations (see e.g. [23, 36, 10, 6, 4, 21, 16, 5]). This is due to the fact that electrical flow computations reduce to solving Laplacian systems, which is a task that can be performed in nearly linear time. In addition, flow algorithms usually aim to minimize the ℓ_∞-norm in some way. The fact that the ℓ_2-norm is closer to the ℓ_∞ than e.g. the ℓ_1 norm helps in these optimizations. Electrical flows have also been used to generate alternative routes in road networks [33].

Ghaffari, Haeupler, and Zuzic [11] introduced the concept of *hop-constrained* oblivious routing. This is an oblivious routing scheme that is given an additional parameter h that can be viewed as an upper bound on the dilation used by an optimum routing. They show that one can obtain an oblivious routing scheme that guarantees a congestion of $\widetilde{O}(C_h)$ and a dilation of $\widetilde{O}(h)$, where C_h is the optimum congestion that can be obtained with routing paths of length at most h. However, their construction still suffers from the same drawback as tree-based routing schemes in that it requires $\Omega(m)$ iterations of multiplicative weights, which results in a depth of $\Omega(m)$. There has also been recent interest in computing competitive routing schemes with small support, as shown in the work of Zuzic, Haeupler, and Roeyskoe [37] in the context of *semi-oblivious* routing.

In the sub-polynomial competitive ratio regime, Kelner et al. [16] give an algorithm to compute an $n^{o(1)}$ competitive oblivious routing in $m^{1+o(1)}$ time, and use this as a subroutine to compute a $(1+\epsilon)$-approximate maximum s-t flow and concurrent multicommodity flow in almost-linear time. Haeupler et al. [12] present a construction of routing tables for h-hop routing that runs in $O(D+\text{poly}(h))$ rounds of the CONGEST-model of distributed computing, and guarantee a competitive ratio of $n^{o(1)}$. This essentially means that the resulting packet routing algorithm can schedule any permutation in time $n^{o(1)}$ whenever this is possible, and it's extremely fast. However, the scheme crucially uses the fact that routing requests are initiated from both communication partners; it requires name-dependent routing (the node ids are changed during initialization to allow for compact routing tables), and it does not achieve a polylogarithmic competitive ratio.

Technical Overview

For the so-called *linear oblivious routing schemes* (such as tree-based routing schemes or electrical flows) it is well known that the worst demand is a demand of 1 along every edge in the unweighted network. An optimal algorithm can trivially route this demand by sending 1 unit of flow along every edge in the network, and, hence, has maximum congestion 1. Let $\text{load}_w(e)$ denote the load on edge e generated by an electrical flow routing (with conductances w) when routing this worst case demand. The competitive ratio of the routing scheme is then $\max_e \text{load}_w(e)$. We can write the search for a good convex combination of electrical flow routings w_i as a linear program:

$$\begin{array}{lll} \min & \alpha & \\ \text{s.t.} & \forall e \quad \sum_i \lambda_i \cdot \text{load}_{w_i}(e) & \leq \alpha \\ & \qquad\qquad \sum_i \lambda_i & = 1 \\ & \forall i \qquad\qquad \lambda_i & \geq 0 \ . \end{array}$$

We want to find a good solution to this problem, say with objective value β. For this the multiplicative weights update method maintains a set of weights $p_e \geq 0$ on the edges with $\sum_e p_e = 1$. In each iteration it computes *one* routing R_i such that R_i fulfills $\sum_e p_e \text{load}_{R_i}(e) \leq \beta$, i.e., the weighted total load over all edges for R_i is at most β. Here $\text{load}_{R_i}(e)$ denotes the load induced on edge e when routing a demand of 1 along every edge using the routing R_i. This means *on average* the edges have load at most β when using routing R_i. For the next iteration the weight p_e of edges with $\text{load}_{R_i}(e) > \beta$ is increased while edges with $\text{load}_{R_i}(e) < \beta$ decrease their weight. In the end the convex combination of all the routings will have load at most β for every edge.

To apply this scheme we first need a bound β such that we always can (efficiently) find a routing with $\sum_e p_e \text{load}_{R_i}(e) \leq \beta$. For tree-based routing one can use small stretch trees [8, 1] to solve this problem with $\beta = O(\log n)$. For electrical routing we show that we can bound β using a specific choice of parameters in the so-called *localization lemma* due to Schild et al. [31]. Informally, the localization lemma states that the average length of flow paths in an electrical routing is small. In particular, we show that we can find parameters for localization which guarantee the existence of conductances w such that the load when routing via the electrical flow with conductances w (say load_w) fulfills $\sum_e p_e \text{load}_w(e) \leq \beta$, with $\beta = O(\log^2 n)$.

Secondly we need to be able to efficiently compute the load on every edge after computing the routing R_i in an iteration, in order to be able to adjust the weights for the next iteration. Naïvely routing across each edge using R_i would require time $O(m^2)$ for solving m Laplacian systems to compute the electrical flow between the end points of each edge. We use a sketching result due to Indyk [15] for approximating the ℓ_1-norm of vectors. This allows us to approximate the loads due to all m electrical flows in time $\widetilde{O}(m)$. We show that this approximation is still sufficient for the multiplicative weights method to work correctly.

The advantage of using electrical flows within the multiplicative weights update framework is that it allows to derive a better bound on the convergence time. The number of iterations required for the method is proportional to how far each inequality, namely $\text{load}_w(e) \leq \beta$ for each e, is violated by the routing found in each iteration. This is the so-called *width* of the algorithm, and we bound the width by $O(\sqrt{m})$ by regularizing the weights to be nearly uniform. This in turn implies that we require only $O(\sqrt{m})$ iterations.

2 Preliminaries

For simplicity of presentation, we present the uncapacitated case in the following sections, and explain the changes required for the capacitated case in Section 8.

2.1 Oblivious Routing

Graph. We will route on unweighted, undirected graphs. Choose an arbitrary orientation of the edges and let B be the $m \times n$ incidence matrix of G. Denote by b_e the row of B corresponding to edge e.

Demand. A demand $\chi \in \mathbb{R}^n$ encodes the flow deficit/surplus requirements for each vertex in the graph. A demand is *valid* if $\sum_i \chi_i = 0$, that is, the requirements at all the vertices of the graph cancel out. Let $X \subseteq \mathbb{R}^n$ denote the space of all valid demands. Note that the vector b_e^T has unit demand across the endpoints of e (with the direction of demand decided by the orientation of the edge). If the demands are only between pairs of vertices (s,t), then we encode the demands between all pairs of vertices in a *demand vector* $d \in \mathbb{R}^{\binom{n}{2}}$.

Flow. A flow corresponding to a demand χ is a vector $f \in \mathbb{R}^m$ that ensures that the flow out of a vertex satisfies the demand, namely, $\chi_v + \sum_{u \sim v} f_{uv} = 0$ for all $v \in V$. For example, an (s,t)-flow is one where the demands of every vertex except s and t are zero.

Oblivious routing. An $m \times n$ matrix M that maps vertex demands to edge flows is called an *oblivious routing scheme*[5] if for any valid demand χ the vertex demands created by the flow generated by M are equal to the input vertex demands. Formally, we require that $B^\mathsf{T} M$ is identity on X, i.e., $B^\mathsf{T} M \chi = \chi$ for all valid demands $\chi \in X$.

Competitive ratio. For an oblivious routing M and any collection of demands $D = \{\chi_i\}_i$, the flow across edge e for routing demand χ_i is given by $|b_e M \chi_i|$. Thus the total flow (without cancellations) across edge e when routing all the demands in D *independently* using M is given by $f_e = \sum_{\chi_i \in D} |b_e M \chi_i|$. We would like to minimize the ℓ_p norm (in this paper, $p \in \{1, \infty\}$) of the flow resulting from routing these demands independently with an oblivious routing M, as compared with the optimal routing that can adaptively choose the routing based on the demands. If $OPT_p(D)$ is the cost resulting from the optimal routing, and f is the flow described above, then the competitive ratio of the oblivious routing scheme M is given by

$$\beta_p(M) = \max_D \frac{\|f\|_p}{OPT_p(D)}$$

Kelner and Maymounkov [17, Theorem 3.1] show that for a linear oblivious routing on an uncapacitated graph, the worst case demand set is routing one unit of flow across every edge of the graph, i.e., the worst case requests are the columns of B^T. Since the optimal routing is to route each demand across the same edge, $OPT(B^\mathsf{T}) = 1$. For $p \in \{1, \infty\}$, the competitive ratio is then given by $\beta_p(M) = \|MB^\mathsf{T}\|_p$.

Load. For oblivious routing M, the flow across edge e for the demand b_f is given by $\text{load}_M(f \to e)$. This can be obtained from the matrix M as $\text{load}_M(f \to e) = |(Mb_f^\mathsf{T})_e|$. The sum of all flows across an edge e for every possible edge demand vector b_f is then the load across the edge e.

$$\text{load}_M(e) = \textstyle\sum_f \text{load}_M(f \to e) = \sum_f |(Mb_f^\mathsf{T})_e|.$$

Note that the congestion, or the competitive ratio for the ℓ_∞ norm, is then just the maximum load on any edge in the graph by definition of $\|MB^\mathsf{T}\|_\infty$.

[5] While an oblivious routing scheme in general could be non-linear, we only consider linear routing schemes in this paper.

Convexity of load. Load is convex on oblivious routings. For two oblivious routings M_0 and M_1, let $M_\lambda = \lambda M_1 + (1-\lambda)M_0$ for $\lambda \in (0,1)$. Then

$$\begin{aligned}\text{load}_{M_\lambda}(f \to e) &= |((\lambda M_1 + (1-\lambda)M_0)b_f^\mathsf{T})_e| \le \lambda|(M_1 b_f^\mathsf{T})_e| + (1-\lambda)|(M_0 b_f^\mathsf{T})_e| \\ &= \lambda \text{load}_{M_1}(f \to e) + (1-\lambda)\text{load}_{M_0}(f \to e)\end{aligned}$$

The simplex. We use $\Delta^m = \{x \in \mathbb{R}^m : \sum x_i = 1, x_i \ge 0\}$ to denote the m-simplex. Any $p \in \Delta^m$ then represents a probability distribution over the edges.

2.2 Electrical Flow

Graph. Let (G, w) be a weighted, undirected graph with edge weights $\{w_e\}$[6]. Choose an arbitrary orientation of the edges. B is the $m \times n$ incidence matrix of G. If $W = \text{diag}(w)$ is the $m \times m$ diagonal matrix of edge weights, then the $n \times n$ *Laplacian matrix* is given by $L = B^\mathsf{T}WB$, and $L^\dagger$ is its pseudoinverse. Notationally, when we use $L^\dagger$, we always mean the pseudoinverse of the Laplacian with respect to the current weighted graph and *not* the underlying unit-weighted graph.

Electrical flow. Given a weighted graph (G, w) and a demand χ, the electrical flow corresponding to χ is given by $WBL^\dagger\chi$.

Load. By the above characterization of electrical flow, the load across an edge e for unit current demand across f is given by $\text{load}_w(f \to e) = w_e|b_e L^\dagger b_f^\mathsf{T}|$. The load on an edge is the sum of all loads on the edge for demands b_f for all edges f, given by $\text{load}_w(e) = w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|$.

Weights determine electrical flow. An electrical flow is completely determined by the edge weights. That is, given an edge weighting w of an unweighted graph G, there is a unique routing $WBL^\dagger$ corresponding to this network. If the unweighted graph G is fixed, we denote the load due to weights w on an edge e as $\text{load}_w(e)$.

Electrical flows are oblivious. Let G be an unweighted graph for which we wish to find an oblivious routing. Then for any weighting w of the graph, the corresponding electrical flow $M_w = WBL^\dagger$ is an oblivious routing. This is easy to check, since $B^\mathsf{T}WBL^\dagger = (B^\mathsf{T}WB)(B^\mathsf{T}WB)^\dagger$. We will refer to M_w as the *electrical routing with respect to the weights* $\{w_e\}$.

Localization. We will repeatedly use the following so-called *localization lemma*. A concrete instantiation with $\alpha_{\text{LOCAL}} = c\log^2(n)$ for some constant c was given by Schild et al. [31].

▶ **Lemma 2** (Localization [31]). *Let G be a graph with weights $\{w_e\}$. Then for any vector $\ell \in \mathbb{R}^m_{\ge 0}$,*

$$\sum_{e,f \in E} \ell_e \ell_f \sqrt{w_e w_f} |b_e L_G^\dagger b_f^\mathsf{T}| \le \alpha_{LOCAL} \cdot \|\ell\|_2^2$$

[6] Think of them as the conductances of the edges in an electrical network.

Algorithm 1 ComputeRouting, to achieve $O(\alpha_{\text{LOCAL}})$-competitive oblivious routing.

Input: An unweighted graph G.
Output: An oblivious routing scheme on G.

Set $\rho \leftarrow \sqrt{2m}$ and $T \leftarrow \frac{8\rho \ln m}{\alpha_{\text{LOCAL}}}$
Initialize $x_e^{(0)} \leftarrow 1$ for all $e \in E$, and $X^{(0)} \leftarrow m$
for $t = 1, 2, \ldots, T$ **do**
  Set $p_e^{(t)} \leftarrow {x_e^{(t-1)}}/{X^{(t-1)}}$ for all $e \in E$
  Set $w_e^{(t)} \leftarrow (p_e + {1}/{m})^{-1}$ for all $e \in E$
  Set $W^{(t)} \leftarrow \text{diag}(w)$
  Set $M^{(t)} \leftarrow W^{(t)} B \left(B^{\mathsf{T}} W^{(t)} B\right)^{\dagger}$
  Set $\text{aload}_{w^{(t)}} \leftarrow \text{GetApproxLoad}(G, w, 1/2)$
  Set $x_e^{(t)} \leftarrow x_e^{(t-1)} \cdot \left[1 + \frac{1}{2\rho} \cdot \text{aload}_{w^{(t)}}(e)\right]$
  Set $X^{(t)} \leftarrow \sum_e x_e^{(t)}$
end
return $M^* = \frac{1}{T} \cdot \sum_{t=1}^{T} M^{(t)}$

2.3 Matrix sketches

To speed up our algorithms we use a sketching result by Indyk [15, Theorem 3], adapted from a formulation by Schild [30, Theorem 9.13].

▶ **Theorem 3** ([15]). *Given $m \in \mathbb{Z}_{\geq 1}$, $\delta \in (0,1)$, and $\epsilon \in (0,1)$, there is a sketch matrix $C = \textsc{SketchMatrix}(m, \delta, \epsilon) \in \mathbb{R}^{\ell \times m}$ and an algorithm $\textsc{RecoverNorm}(s)$ for $s \in \mathbb{R}^{\ell}$ such that the following properties hold:*

- *(Approximation) For any $v \in \mathbb{R}^m$, with probability at least $1 - \delta$ over the randomness of SketchMatrix, the value of $r = \textsc{RecoverNorm}(Cv)$ is*

$$(1-\epsilon)\|v\|_1 \leq r \leq (1+\epsilon)\|v\|_1$$

- *$\ell = c/\epsilon^2 \cdot \log(1/\delta)$ for some constant $c > 1$*
- *(running time) SketchMatrix and RecoverNorm take time $O(\ell m)$ and $\text{poly}(\ell)$ respectively.*

We will require that our theorems hold with high probability, and thus we will set δ to be $1/\text{poly}(n)$. We will also set the approximation constant ϵ to be $1/2$, which gives $\ell = O(\log n)$.

3 Algorithm for routing

In this section, we give an algorithm that returns a routing which achieves competitive ratio $O(\alpha_{\text{LOCAL}})$ by taking a convex combination of $\sqrt{m}$ electrical flows, where α_{LOCAL} is the best localization result for electrical flows in Lemma 2. We use the *multiplicative weights update* method (MWU) to obtain this guarantee. The algorithm is presented in Algorithm 1 (ComputeRouting). We are essentially solving the following linear program with MWU

$$\begin{aligned} \min \quad & \alpha \\ \text{s.t.} \quad \forall e \quad & \textstyle\sum_i \lambda_i \cdot \text{load}_{w_i}(e) \leq \alpha \\ & \textstyle\sum_i \lambda_i = 1 \\ \forall i \quad & \lambda_i \geq 0. \end{aligned}$$

Algorithm 2 GetApproxLoad, to compute approximate loads for electrical routing.

Input: A graph G, weights $\{w_e\}_{e\in E}$ on the edges, approximation factor ϵ.
Output: Approximation $\{\text{aload}_w(e)\}_{e\in E}$ to the load on the edges.

Let B be the edge-vertex incidence matrix of G
Let $L := B^\mathsf{T}\text{diag}(w)B$ be the Laplacian matrix
Set $C \leftarrow \text{SketchMatrix}(m, n^{-10}, \epsilon)$
Set $X \leftarrow B^\mathsf{T}C^\mathsf{T}$
Let $X^{(i)}$ be the i^{th} column of X for all $i \in [\ell]$
Set $U^{(i)} \leftarrow \text{LapSolve}(L, X^{(i)})$ for all $i \in [\ell]$
Set $U \leftarrow (U^{(1)}, U^{(2)}, \ldots, U^{(\ell)})$ ▷ $U = (CBL^\dagger)^\mathsf{T}$
Set $\text{aload}_w(e) \leftarrow w_e \cdot \text{RecoverNorm}(U^\mathsf{T}b_e)$ for all $e \in E$
return aload_w

The primal asks for a convex combination of electrical routings that gives low competitive ratio, where the coefficients of the convex combination are collected in λ. The dual of the above linear program is equivalent to solving

$$\max_{p\in\Delta^m} \min_i \sum_e p_e \text{load}_{w_i}(e), \tag{1}$$

Denote the optimal values of the primal and dual by α^* and β^* respectively. Since $\alpha^* = \beta^*$ by strong duality, we can obtain an existential bound on α^* by showing that $\beta^* \le \alpha_{\text{LOCAL}}$. This bound on β^* is obtained by bounding (1) using the weighting returned by Lemma 4 which returns, for any $p \in \Delta^m$, a weighting w for which the p_e weighted average load is smaller than $2\alpha_{\text{LOCAL}}$. The proof of the bound in Lemma 4 uses localization, and we wish to convert this existential result into an algorithmic one, which we do using MWU in Algorithm 1.

Our MWU algorithm can be described as the following primal-dual algorithm running for T iterations: We initially start with the primal vector $\lambda = 0$ and dual vector $p_e^{(1)} = 1/m$. We then build a primal solution incrementally as follows. At iteration t, we look at the dual variables $p_e^{(t)}$ of the edges, and find a particular set of weights $w_e^{(t)}$ such that routing with respect to these weights gives low average load with respect to the dual variables, i.e., $\sum_e p_e^{(t)} \text{load}_{w^{(t)}}(e)$ is low. As in the existential case, we show a bound of $O(\alpha_{\text{LOCAL}})$ on this average load using localization.

Recall that the primal vector λ gives the coefficients for a convex combination of electrical routings. We now increase the primal coefficient corresponding to the routing that was found at this iteration, $\lambda_{M^{(t)}}$, by $1/T$. We essentially "add" this routing to the final convex combination that we output at the end of the algorithm. Now we need to update the dual variables for the next iteration. For the routing returned in the next iteration, we want to reduce the load on edges that had really high load in the current iteration. To this end, we compute the loads induced on each edge (using GetApproxLoad), and adjust the dual variables based on how high the loads for the current routing $M^{(t)}$ are.

At the end of T iterations, λ is the uniform combination of all T electrical routings computed during each iteration of the algorithm. The analysis of MWU shows that our dual variable updates ensure low load on all the edges for the returned routing. The number of iterations T is proportional to how far each primal inequality, namely $\text{load}_{w^{(t)}}(e) \le \alpha_{\text{LOCAL}}$ for each e, is violated by the routing found in each iteration. This is the *width* of the algorithm, and we bound the width by $O(\sqrt{m})$.

For the running time, note that evaluating loads for all the edges naïvely would take time $O(m^2)$ in each iteration, since it would involve calculating $|b_e L^\dagger b_f^\mathsf{T}|$ for every pair of edges $(e, f) \in E^2$. However, the load on each edge can also be expressed as the ℓ_1 norm of the

vector $BL^\dagger b_e$. We use a sketching result by Indyk [15] stated in Theorem 3, which gives a sketch matrix C that preserves ℓ_1 norms up to small multiplicative errors. Similar ideas have been implicitly used by Schild [30], and later in other works [22, 9] as well. Our sketching improves the running time from $O(m^2)$ to $\widetilde{O}(m)$ per iteration. Since there are $T = O(\sqrt{m})$ iterations, we get a bound of $\widetilde{O}(m\sqrt{m})$ for constructing our routing.

For simplicity, we first assume that in GetApproxLoad we have an exact Laplacian solver that runs in time $\widetilde{O}(m)$, and present below the analysis in Section 4. The analysis when using an approximate Laplacian solver is more technical, and we defer it to the full version. The complication to overcome when using an approximate Laplacian solver in line 6 of GetApproxLoad is the following: The input to RecoverNorm might not necessarily be of the form Cv for some $v \in \mathbb{R}^m$, since we use the approximate Laplacian solver on $B^\mathsf{T}C^\mathsf{T}$, and thus would have obtained $\textsc{approx}(CBL^\dagger)b_e$ at the end. Note that if we instead had $C\textsc{approx}(BL^\dagger b_e)$ this would be fine, since we would get that $\textsc{RecoverNorm}(CBL^\dagger b_e) \approx \|BL^\dagger b_e\|_1$ and $\|BL^\dagger b_e\|_1 \approx \|\textsc{approx}(BL^\dagger b_e)\|_1$, and the resulting guarantee would be the product of these two approximation guarantees.

At a high level, we have an approximation to the vector *after* sketching ($\textsc{approx}(Cv)$) instead of an approximation to the vector *before* sketching ($C\textsc{approx}(v)$). Since the function RecoverNorm is not a norm, we do not have a guarantee that it behaves well on inputs that are close to each other. We need to argue that RecoverNorm still returns a useful answer when given an input vector that is not in the column space of C, but is nevertheless close in the L norm to the required vector Cv. This requires some technical analysis.

4 Proof of correctness

The proof is an adaptation of the multiplicative weight update proof to our setting. The proof uses the following three lemmas. Due to their length we defer their proofs to Sections 4.1, 4.2, and 4.3 respectively. The first lemma shows that the *(p_e weighted) average edge load* is $O(\alpha_{\text{LOCAL}})$.

▶ **Lemma 4.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\sum_e p_e \text{load}_w(e) \le 2\alpha_{\text{LOCAL}}$.*

We use the second lemma to show that in each iteration of ComputeRouting the true loads on each edge are $O(\sqrt{m})$ away from α_{LOCAL}.

▶ **Lemma 5.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\text{load}_w(e) \le \sqrt{2m}$ for every edge e.*

The third lemma shows that the error introduced by using matrix sketching while computing GetApproxLoad is not too large, i.e., that GetApproxLoad approximates the true loads well. Since existing Laplacian solvers work in the high-accuracy regime (with error proportional to $\log(1/\epsilon)$), we assume here for simplicity that the Laplacian solver used in GetApproxLoad is exact and runs in time $\widetilde{O}(m)$. The analysis with an approximate solver is deferred to the full version.

▶ **Lemma 6.** *For any approximation factor $0 < \epsilon < 1$, and any weighted graph (G, w), let $\text{load}_w = \left(w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|\right)_{e \in E}$ be the true loads, and $\text{aload}_w = \textsc{GetApproxLoad}(G, w, \epsilon)$ be the approximate loads computed by the algorithm. Then with probability $\ge 1 - 1/\text{poly}(n)$,*

$$(1-\epsilon) \cdot \text{load}_w(e) \le \text{aload}_w(e) \le (1+\epsilon) \cdot \text{load}_w(e) \qquad \text{for all } e \in E$$

We first track the potential function $\|x^{(t)}\|_1$, and upper bound the increase in $\|x^{(t)}\|_1$ throughout the algorithm.

▶ **Lemma 7.** *For any $t \geq 1$, $\|x^{(t)}\|_1 \leq \|x^{(t-1)}\|_1 \cdot \exp(2\alpha_{LOCAL}/\rho)$. At the end of the algorithm, $\|x^{(T)}\|_1 \leq m \cdot \exp(2\alpha_{LOCAL}T/\rho)$.*

Proof. Once we prove the first statement, the implication for $\|x^{(T)}\|_1$ follows from noting that $\|x^{(0)}\|_1 = m$. For any $t \geq 1$, we have

$$\begin{aligned}
\|x^{(t)}\|_1 &= \textstyle\sum_e x_e^{(t)} = \sum_e x_e^{(t-1)} \cdot \left(1 + \frac{1}{2\rho} \cdot \mathrm{aload}_{w^{(t)}}(e)\right) \\
&= \textstyle\sum_e x_e^{(t-1)} + \frac{1}{2\rho} \cdot \sum_e x_e^{(t-1)} \cdot \mathrm{aload}_{w^{(t)}}(e) \\
&\leq \textstyle\sum_e x_e^{(t-1)} + \frac{\|x^{(t-1)}\|_1}{\rho} \cdot \sum_e \frac{x_e^{(t-1)}}{\|x^{(t-1)}\|_1} \cdot \mathrm{load}_{w^{(t)}}(e) && \text{(by Lemma 6)} \\
&\leq \|x^{(t-1)}\|_1 + \tfrac{\|x^{(t-1)}\|_1}{\rho} \cdot 2\alpha_{\mathrm{LOCAL}} && \text{(by Lemma 4)} \\
&\leq \|x^{(t-1)}\|_1 \cdot \exp\left(\tfrac{2\alpha_{\mathrm{LOCAL}}}{\rho}\right) && \blacktriangleleft
\end{aligned}$$

Next, we lower bound the weight $x_e^{(t)}$ in terms of the load on each edge.

▶ **Lemma 8.** *Let M^* be the routing returned by the algorithm. For any edge e,*

$$x_e^{(T)} \geq \exp\left(\tfrac{T}{8\rho} \cdot \mathrm{load}_{M^*}(e)\right).$$

Proof. For any edge e and $t \geq 1$, we have

$$\begin{aligned}
x_e^{(t)} &= \textstyle\prod_{t'=1}^{t} \left(1 + \frac{1}{2\rho} \cdot \mathrm{aload}_{w^{(t')}}(e)\right) \\
&\geq \textstyle\prod_{t'=1}^{t} \left(1 + \frac{1}{4\rho} \cdot \mathrm{load}_{w^{(t')}}(e)\right) && \text{(by Lemma 6)} \\
&\geq \textstyle\prod_{t'=1}^{t} \exp\left(\frac{1}{8\rho} \cdot \mathrm{load}_{w^{(t')}}(e)\right) && \text{(since } e^x \leq 1 + 2x \text{ for } 0 < x < 1\text{, and Lemma 5)} \\
&= \exp\left(\tfrac{1}{8\rho} \cdot \textstyle\sum_{t'=1}^{t} \mathrm{load}_{w^{(t')}}(e)\right)
\end{aligned}$$

In particular, for $t = T$, we have

$$\begin{aligned}
x_e^{(T)} &\geq \exp\left(\tfrac{1}{8\rho} \cdot \textstyle\sum_{t'=1}^{T} \mathrm{load}_{w^{(t')}}(e)\right) \\
&\geq \exp\left(\tfrac{T}{8\rho} \cdot \mathrm{load}_{M^*}(e)\right) && \text{(by convexity of load)} \quad \blacktriangleleft
\end{aligned}$$

▶ **Theorem 9.** *The routing returned by Algorithm 1 has competitive ratio $O(\alpha_{LOCAL})$.*

Proof. Combining Lemmas 7 and 8, for any edge e,

$$m \exp\left(\tfrac{2\alpha_{\mathrm{LOCAL}}T}{\rho}\right) \geq \|x^{(T)}\|_1 \geq x_e^{(T)} \geq \exp\left(\tfrac{T}{8\rho} \cdot \mathrm{load}_{M^*}(e)\right)$$

which in particular gives the required upper bound on $\mathrm{load}_{M^*}(e)$ for any edge e, since

$$\begin{aligned}
\mathrm{load}_{M^*}(e) &\leq 16\alpha_{\mathrm{LOCAL}} + \tfrac{8\rho \log m}{T} \\
&= O(\alpha_{\mathrm{LOCAL}}) \qquad \blacktriangleleft
\end{aligned}$$

4.1 Bound on Average Loads

▶ **Lemma 4.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\sum_e p_e \text{load}_w(e) \leq 2\alpha_{\text{LOCAL}}$.*

Proof. Setting ℓ_e to $1/\sqrt{w_e}$ and applying Lemma 2, we get

$$\begin{aligned}
\textstyle\sum_e p_e \text{load}_w(e) &= \textstyle\sum_e p_e \sum_f \text{load}_w(f \to e) \\
&= \textstyle\sum_e p_e \cdot w_e \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of load)} \\
&= \textstyle\sum_e p_e \cdot (p_e + 1/m)^{-1} \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of } w_e\text{)} \\
&\leq \textstyle\sum_e \sum_f |b_e L^\dagger b_f^\mathsf{T}| \\
&= \textstyle\sum_{e,f} 1/\sqrt{w_e w_f} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \textstyle\sum_{e,f} \ell_e \ell_f \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by choice of } \ell_e\text{)} \\
&\leq \alpha_{\text{LOCAL}} \cdot \|\ell\|_2^2 && \text{(by Lemma 2)} \\
&= \alpha_{\text{LOCAL}} \cdot \textstyle\sum_e 1/w_e \\
&= \alpha_{\text{LOCAL}} \cdot \textstyle\sum_e (p_e + 1/m) \\
&= 2\alpha_{\text{LOCAL}},
\end{aligned}$$

as required. ◀

4.2 Bounding the Width

Next we show that the width is bounded above by $O(\sqrt{m})$. We first prove a couple of properties of the Π matrix that we will use in our analysis.

▶ **Lemma 10.** *The matrix Π is a projection matrix.*

Proof. We show that $\Pi^2 = \Pi$.

$$\begin{aligned}
\Pi^2 &= \left(W^{1/2} B L^\dagger B^\mathsf{T} W^{1/2}\right) \cdot \left(W^{1/2} B L^\dagger B^\mathsf{T} W^{1/2}\right) \\
&= \left(W^{1/2} B\right) \cdot \left(L^\dagger B^\mathsf{T} W B L^\dagger\right) \cdot \left(B^\mathsf{T} W^{1/2}\right) \\
&= \left(W^{1/2} B\right) \cdot L^\dagger \cdot \left(B^\mathsf{T} W^{1/2}\right) = \Pi.
\end{aligned}$$

as required. ◀

We will use the next lemma to bound the width for both the ℓ_∞ and the ℓ_1 case.

▶ **Lemma 11.** *Let G be a graph with weights $\{w_e\}$ and let L be the Laplacian matrix associated with G. For any edge e, we have that*

$$\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \leq 1.$$

Proof. By Lemma 10 we know that Π is a projection matrix. This in particular implies that the diagonal entries of Π (and hence Π^2) are less then 1. Thus,

$$\begin{aligned}
\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 &= \textstyle\sum_f \left(\sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}|\right)^2 \\
&= \textstyle\sum_f \Pi(e,f)^2 = \Pi^2(e,e) \leq 1,
\end{aligned}$$

which was what we were after. ◀

▶ **Lemma 5.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\text{load}_w(e) \le \sqrt{2m}$ for every edge e.*

Proof. Fixing an edge e,

$$\begin{aligned}\text{load}_w(e) = w_e \textstyle\sum_f |b_e L^\dagger b_f^\mathsf{T}| &\le \textstyle\sum_f \sqrt{w_e/w_f} \cdot \sqrt{w_e w_f}|b_e L^\dagger b_f^\mathsf{T}| \\ &\le \sqrt{\textstyle\sum_f w_e/w_f} \cdot \sqrt{\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2} \qquad \text{(by Cauchy-Schwarz)}\end{aligned}$$

We now bound each of the two terms separately. For the first term, note that

$$w_e \cdot \textstyle\sum_f 1/w_f = (p_e + 1/m)^{-1} \cdot \sum_f (p_f + 1/m) \le 2 \cdot (p_e + 1/m)^{-1} \le 2 \cdot 1/(1/m) = 2m.$$

For the second term, by Lemma 11, we know that

$$\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \le 1.$$

Putting these two inequalities together, we get that $\text{load}_w(e) \le \sqrt{2m}$ for any edge e, which gives the desired bound of $\sqrt{2m}$ on the width. ◀

4.3 Proof of GetApproxLoad correctness assuming exact Laplacian solver

We use the guarantees of SketchMatrix and RecoverNorm provided by Theorem 3 to prove the following lemma.

▶ **Lemma 6.** *For any approximation factor $0 < \epsilon < 1$, and any weighted graph (G, w), let $\text{load}_w = \left(w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|\right)_{e \in E}$ be the true loads, and $\text{aload}_w = \text{GetApproxLoad}(G, w, \epsilon)$ be the approximate loads computed by the algorithm. Then with probability $\ge 1 - 1/\text{poly}(n)$,*

$$(1-\epsilon) \cdot \text{load}_w(e) \le \text{aload}_w(e) \le (1+\epsilon) \cdot \text{load}_w(e) \qquad \text{for all } e \in E$$

Proof. Note that GetApproxLoad sends $(L^\dagger B^\mathsf{T} C^\mathsf{T})^\mathsf{T} b_e$ to RecoverNorm. This simplifies to $CBL^\dagger b_e$. We get from the approximation guarantee of Theorem 3 that

$$(1-\epsilon) \cdot \|BL^\dagger b_e\|_1 \le \text{RecoverNorm}(BL^\dagger b_e) \le (1+\epsilon) \cdot \|BL^\dagger b_e\|_1$$

Since $\text{load}_w(e) = w_e \cdot \|BL^\dagger b_e\|_1$, multiplying the above inequality by w_e, we get

$$(1-\epsilon) \cdot \text{load}_w(e) \le \text{GetApproxLoad}(G, w) \le (1+\epsilon) \cdot \text{load}_w(e)$$

as required. ◀

5 Running time analysis

In this section, we show that Algorithm 1 (ComputeRouting) runs in time $\widetilde{O}(m^{3/2})$. If we show that each iteration of the for loop in ComputeRouting takes time $\widetilde{O}(m)$, then since $T = O(\sqrt{m})$, the claimed running time then follows. We first show that Algorithm 2 (GetApproxLoad) runs in time $\widetilde{O}(m)$.

▶ **Lemma 12.** *Algorithm 2 runs in time $\widetilde{O}(m)$.*

Proof. Calculating B and L takes time $O(m)$. By Theorem 3, building the sketch matrix C takes time $O(\ell m)$. Since we set $\delta = n^{-10}$ and $\epsilon = 1/2$ for the sketch matrix, we get $\ell = O(\log n)$, which gives $O(\ell m) = \widetilde{O}(m)$.

Each row of $B^\mathsf{T}C^\mathsf{T}$ can be obtained as follows: Since the row of B^T corresponding to vertex u has $\deg(u)$ non-zero entries, the row of $B^\mathsf{T}C^\mathsf{T}$ corresponding to vertex u can be obtained by taking the sum of every row of C^T corresponding to an edge that is incident to u. Since each row of C^T has ℓ entries, this involves $O(\deg(u) \cdot \ell)$ calculations for computing row u of $B^\mathsf{T}C^\mathsf{T}$. Since $\sum_u \deg(u) = 2m$, this gives an overall bound of $\widetilde{O}(m)$ for calculating $B^\mathsf{T}C^\mathsf{T}$. Each Laplacian solver takes time $\widetilde{O}(m)$, and since we solve for ℓ vectors, all Laplacian solves together take time $\widetilde{O}(m)$ as well.

Finally, we perform RecoverNorm for m edges. Each invocation of RecoverNorm takes time $O(\ell)$ by Theorem 3, and thus all calls to RecoverNorm together run in time $\widetilde{O}(m)$. This proves the lemma. ◀

We can use this to show that each iteration of the for loop in ComputeRouting runs in time $\widetilde{O}(m)$, and thus the entire algorithm runs in time $\widetilde{O}(m^{3/2})$.

▶ **Lemma 13.** *Algorithm 1 runs in time $\widetilde{O}(m^{3/2})$.*

Proof. For each iteration of the for loop, normalizing the $x_e^{(t-1)}$s and calculating $w_e^{(t)}$ (and thus $W^{(t)}$) takes time $O(m)$. Calculating approximate loads is $\widetilde{O}(m)$ by Lemma 12. Since computing $x_e^{(t)}$ takes time $O(m)$, each iteration runs in time $\widetilde{O}(m)$. Thus the algorithm runs in time $\widetilde{O}(Tm) = \widetilde{O}(m^{3/2})$ by choice of $T = O(\sqrt{m})$. ◀

6 Extension to the ℓ_1 norm Oblivious Routing

In this section we prove that a convex combination of $O(\sqrt{m})$ electrical routings gives an oblivious routing scheme with respect to the ℓ_1 norm that achieves a $O(\log^2 n)$ competitive ratio. By the characterization of the competitive ratio as $\beta_1 = \|MB^\mathsf{T}\|_1$, our goal is now to bound the maximum *stretch* of an edge.

Consider the oblivious routing operator $M_w = WBL^\dagger$. The *stretch* of an edge $e \in E$ with respect to the routing operator M_w is given by:

$$\text{stretch}_w(e) := \textstyle\sum_f w_f |b_e L^\dagger b_f^\mathsf{T}|.$$

Similar to earlier, our goal is to solve the following linear program.

$$\begin{aligned} \min \quad & \alpha \\ \text{s.t.} \quad \forall e \quad & \textstyle\sum_i \lambda_i \cdot \text{stretch}_{w_i}(e) \leq \alpha \\ & \lambda \in \Delta^m \end{aligned}$$

Doing the same dual construction as for the load gives us that the dual is equivalent to solving

$$\max_{p \in \Delta^m} \min_i \textstyle\sum_e p_e \text{stretch}_{w_i}(e), \tag{2}$$

Then the only difference from earlier is having to show that we can bound the average stretch and the width for stretch as well. In particular, we first show that for every $p \in \Delta^m$ we can produce a weighting w such that electrical routing on G with weights w gives low average stretch, i.e., $\sum_e p_e \text{stretch}_w \leq \alpha_{\text{LOCAL}}$.

▶ **Lemma 14.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)$ satisfies $\sum_e p_e \text{stretch}_w(e) \leq 2\alpha_{\text{LOCAL}}$.*

Proof. Setting ℓ_e to $\sqrt{w_e}$ and applying Lemma 2, we get

$$\begin{aligned}
\textstyle\sum_e p_e \text{stretch}_w(e) &= \textstyle\sum_e p_e \cdot \sum_f w_f \cdot \text{stretch}_w(e \to f) \\
&= \textstyle\sum_e p_e \cdot \sum_f w_f \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of stretch)} \\
&\leq \textstyle\sum_e (p_e + 1/m) \cdot \sum_f w_f \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&\leq \textstyle\sum_e w_e \cdot \sum_f w_f \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of } w_e\text{)} \\
&= \textstyle\sum_e \sum_f w_e \cdot w_f \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \textstyle\sum_{e,f} \sqrt{w_e w_f} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \textstyle\sum_{e,f} \ell_e \ell_f \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by choice of } \ell_e\text{)} \\
&\leq \alpha_{\text{LOCAL}} \cdot \|\ell\|_2^2 && \text{(by Lemma 2)} \\
&= \alpha_{\text{LOCAL}} \cdot \textstyle\sum_e w_e \\
&= \alpha_{\text{LOCAL}} \cdot \textstyle\sum_e (p_e + 1/m) \\
&= 2\alpha_{\text{LOCAL}},
\end{aligned}$$

as required. ◀

We next bound the width by showing an upper bound on the stretch of every single edge.

▶ **Lemma 15.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)$ satisfies $\text{stretch}_w(e) \leq \sqrt{2m}$ for every edge e.*

Proof. Fixing an edge e,

$$\begin{aligned}
\text{stretch}_w(e) &= \textstyle\sum_f w_f |b_e L^\dagger b_f^\mathsf{T}| \\
&\leq \textstyle\sum_f \sqrt{w_f / w_e} \cdot \sqrt{w_e w_f} |b_e L^\dagger b_f^\mathsf{T}| \\
&\leq \sqrt{\textstyle\sum_f w_f / w_e} \cdot \sqrt{\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2} && \text{(by Cauchy-Schwarz)}
\end{aligned}$$

We now bound each of the two terms separately. For the first term,

$$\begin{aligned}
1/w_e \cdot \textstyle\sum_f w_f &= (p_e + 1/m)^{-1} \cdot \textstyle\sum_f (p_f + 1/m) \\
&\leq 2 \cdot (p_e + 1/m)^{-1} \\
&\leq 2 \cdot 1/(1/m) \\
&= 2m.
\end{aligned}$$

For the second term, by Lemma 11, we know that

$$\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \leq 1.$$

Putting these two inequalities together, we get that $\text{stretch}_w(e) \leq \sqrt{2m}$ for any edge e, which gives the desired bound of $\sqrt{2m}$ on the width. ◀

With these two lemmas, the rest of the proofs of the bound on the competitive ratio are exactly the same as in the ℓ_∞ case. While most of the running time analysis holds, note that we need to sketch slightly different matrices now. Earlier, we wanted to approximate

$\text{load}_w(e) = w_e \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}|$. Thus we approximated $\|BL^\dagger b_e\|_1$ with our sketch matrix, and then multiplied it with w_e to obtain approximate loads. Since we now need to approximate $\text{stretch}_w(e) = \sum_f w_f |b_e L^\dagger b_f^\mathsf{T}|$, we instead sketch $WBL^\dagger b_e$ to obtain the approximate stretch. Note that load and stretch are simply the row and column 1-norms respectively of $WBL^\dagger B^\mathsf{T}$.

7 The Representation of Oblivious Routing and the Parallel Complexity

We discuss the representation of our oblivious routing scheme and the parallel complexity of our algorithms.

Representation of Oblivious Routing

Any *linear* oblivious routing scheme can be implemented using the following representation: every edge $e \in E$ stores the flow sent across edge e by an oblivious routing that sends one unit of flow from u to x, for every vertex $u \in V$ and some arbitrary but fixed target vertex $x \in V$. We let $f_{u,x}(e)$ denote the value of such an oblivious flow. Thus every edge stores n values and the total space is $O(nm)$. Upon receiving a query for routing demand pairs $d_{s,t}$ of demand vector $d \in \mathbb{R}^{\binom{n}{2}}$, we can compute the (s,t)-flow along any edge $e \in E$ by computing

$$d_{s,t} \cdot (f_{s,x}(e) - f_{t,x}(e)),$$

where the correctness follows from the fact that the oblivious routing operator is linear.

We need to show that our oblivious routing operator based on a convex combination of electrical routings is linear. Note that $M_w := WBL^\dagger$ is a linear routing operator for any weighting on the edges $\{w_e\}$. Therefore, any convex combination $\sum_i \lambda_i M_{w_i}$ with $\sum_i \lambda_i = 1$ is also a linear routing operator.

It remains to study the running time of constructing such a representation, which we refer to as the *preprocessing time*. We start by considering the cost of constructing the representation for a single electrical routing $M_w := WBL^\dagger$. Since $f_{u,x}(e) = (WBL^\dagger \chi_{u,x})_e$, our goal is to compute $WBL^\dagger \chi_{u,x}$, for every $u \in V$ and the fixed vertex x, which can be achieved by solving n Laplacian systems. Each such system can be solved in $\tilde{O}(m)$ time using, say, the solver of Spielman and Teng [34], which in turn leads to a processing time of $\tilde{O}(mn)$ for a single electrical routing. Since our oblivious routing scheme consists of $O(\sqrt{m})$ electrical routings, it follows that the total preprocessing time for constructing the representation for these electrical routings is $\tilde{O}(m^{3/2}n)$.

Parallel Complexity

We next bound the parallel complexity of (i) multiplicative weights updates for computing the weights of our oblivious routing scheme (Algorithm 1) and (ii) the algorithm for computing the representations of our scheme. Both results rely on the fact that a Laplacian system can be solved to high accuracy with nearly-linear work and polylogarithmic depth.

▶ **Theorem 16** ([25, 19]). *Given a n-vertex m-edge graph G, a Laplacian matrix L, a demand vector $y \in \mathbb{R}^n$ and an error bound ϵ_L, there is a parallel algorithm that achieves $\tilde{O}(m)$ work and $\tilde{O}(1)$ depth and returns a vector $x \in \mathbb{R}^n$ such that*

$$\|x - L^\dagger y\|_L \leq \epsilon_L \cdot \|L^\dagger y\|_L.$$

Our first result shows that our MWU-based oblivious routing can be implemented in parallel with $\tilde{O}(m^{3/2})$ work and $\tilde{O}(\sqrt{m})$ depth.

▶ **Lemma 17.** *There is a parallel implementation of Algorithm 1 that achieves $\tilde{O}(m^{3/2})$ work and $\tilde{O}(\sqrt{m})$ depth.*

Proof. We start by analyzing the parallel complexity of Algorithm 1. Consider one iteration of the **for** loop, and observe that the parallel complexity is dominated by the parallel cost of computing approximate loads in Line 9, i.e., Algorithm 2. Hence, it suffices to bound the parallel complexity of the latter. For generating the sketch matrix C, note that each edge $e \in E$ draws $\ell = O(\log n)$ independent random variables from the Cauchy distribution [15], and since these operations can be performed locally, it follows that constructing C takes $\tilde{O}(m\ell) = \tilde{O}(m)$ work and $O(1)$ depth.

Next, we compute the $(n \times \ell)$-dimensional matrix $X = B^{\mathsf{T}}C^{\mathsf{T}}$ in a row-wise fashion (Algorithm 2, Line 4). The u-th row of B^{T} contains $\deg(u)$ non-zero entries, and thus an entry $(X)_{u,i} = (B^{\mathsf{T}}C^{\mathsf{T}})_{u,i} = \sum_{e|e\sim u} b^{\mathsf{T}}_{u,e}c^{\mathsf{T}}_{e,i}$ can be evaluated with $O(\deg(u))$ work and $O(\log n)$ depth. As X has only $\ell = O(\log n)$ columns, it follows that u-th row can be computed with $\tilde{O}(\deg(u))$ work and $O(\log n)$ depth. Summing the costs over all rows of X, we conclude that X can be computed with $\tilde{O}\left(\sum_u \deg(u)\right) = \tilde{O}(m)$ work and $O(\log n)$ depth. Finally, Lines 5 and 6 of Algorithm 2 involve solving $\ell = O(\log n)$ Laplacian systems. By Theorem 16, we can solve all these systems in parallel with $\tilde{O}(m)$ work and $\tilde{O}(1)$ depth.

Bringing all the above bounds together shows that an iteration of the **for** loop in Algorithm 1 can be implemented in parallel with $\tilde{O}(m)$ work and $\tilde{O}(1)$ depth. Since in total there are $\tilde{O}(\sqrt{m})$ iterations, we get that a parallel implementation of Algorithm 1 has $\tilde{O}(m^{3/2})$ work and $\tilde{O}(\sqrt{m})$ depth. ◀

Our second result show that the representation of our oblivious routing can be implemented in parallel with $\tilde{O}(m^{3/2}n)$ work and $\tilde{O}(1)$ depth.

▶ **Lemma 18.** *The representation of the oblivious routing based on $O(\sqrt{m})$ electrical routings can be implemented in parallel with $\tilde{O}(m^{3/2}n)$ work and $\tilde{O}(1)$ depth.*

Proof. We first analyze the cost of a single electrical routing given a weighting of the edges. Recall that our representation requires that every vertex solves one Laplacian system. These systems can be solved independently of each other (and thus in parallel), and by Theorem 16, each of them can be implemented in parallel with $\tilde{O}(m)$ work and $\tilde{O}(1)$ depth. Thus, the parallel complexity of a single electrical routing is $\tilde{O}(mn)$ work and $\tilde{O}(1)$ depth.

Now, note that our MWU algorithm has already computed $\tilde{O}(\sqrt{m})$ weightings of the graph, each corresponding to a single electrical routing. Once computed, these weightings are independent of each other and, thus, the electrical routings (one per vertex) for all of these weightings can be computed in parallel. Therefore, the total complexity for computing the representation of our routing scheme is $\tilde{O}(m^{3/2}n)$ work and $\tilde{O}(1)$ depth.

To evaluate the average (oblivious) flow from the convex combination of $\tilde{O}(\sqrt{m})$ electrical routings, each edge can locally compute the convex combination of the oblivious flows sent along that edge with $\tilde{O}(\sqrt{m})$ work and $O(\log(\sqrt{m})) = \tilde{O}(1)$ depth. Thus, it follows that the total parallel complexity of this step is $\tilde{O}(m^{3/2})$ work and $\tilde{O}(1)$ depth.

Bringing the above bounds together proves the lemma. ◀

8 Routing on capacitated graphs

We detail the changes to obtain an oblivious routing on capacitated graphs.

Capacitated Graph. A capacitated graph $G = (V, E, u)$ is a undirected graph along with a function $u : E \to \mathbb{R}^+$ that represents the capacity of each edge.

Congestion. Given a flow $f \in \mathbb{R}^m$, the congestion of an edge is the amount of flow on that edge relative to its capacity, given by $|f_e|/u_e$.

Let U denote the $m \times m$ diagonal matrix with u_e on the diagonals. Kelner and Maymounkov [17, Theorem 3.1] show that the worst-case demands for a capacitated graph is u_e along each edge, i.e., the columns of $B^\mathsf{T}U$. Note that these can be routed optimally with congestion 1, by simply routing each demand of u_e across the same edge. In their presentation, they use $\{w_e\}$ for both the capacities and the conductances. We, on the other hand, need to use conductances that are different from the capacities. While their proof works for our case, for the sake of clarity, we present their proof of worst-case demands in Section 8.4 using our notation. This then leads to the following definition of load.

Load. For a linear oblivious routing M, the congestion of edge e for routing $u_f \cdot b_f^\mathsf{T}$ is given by $\text{load}_M(f \to e) = u_e^{-1} \cdot u_f \cdot |(Mb_f^\mathsf{T})_e|$. The load on edge e is then given by summing this congestion up for each $f \in E$, giving

$$\text{load}_M(e) = \textstyle\sum_f \text{load}_M(f \to e) = u_e^{-1} \cdot \sum_f u_f \cdot |(Mb_f^\mathsf{T})_e|$$

While $\text{cong}_M(e)$ would be better than $\text{load}_M(e)$ in the capacitated case, we continue using load_M for continuity with the main body of the paper. For an electrical flow with weights (i.e. conductances) $\{w_e\}$, the corresponding oblivious routing is then $WBL^\dagger$ (where the Laplacian is with respect to W, given by $B^\mathsf{T}WB$), and the load is

$$\text{load}_w(e) = \tfrac{w_e}{u_e} \cdot \textstyle\sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}|$$

To show the bound on competitive ratio for capacitated graphs, we then need to give a set of weights $\{w_e\}$ for each $p_e \in \Delta^m$ such that the MWU algorithm can be performed, and we need to show that we can still use sketching to get approximate loads in $\widetilde{O}(m)$ time.

Concretely, we use the weights $w_e = u_e^2 \cdot (p_e + 1/m)^{-1}$ in the algorithm. We need to prove analogues of Lemmas 4 and 5, and we need to provide a version of GetApproxLoad that works in the capacitated case. We do so in the next three sections, which gives Theorem 1.

8.1 Bound on Average Loads

▶ **Lemma 19.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = u_e^2 \cdot (p_e + 1/m)^{-1}$ satisfies $\sum_e p_e \text{load}_w(e) \leq 2\alpha_{\text{LOCAL}}$.*

Proof. Setting ℓ_e to $u_e/\sqrt{w_e}$ and applying Lemma 2, we get

$$\begin{aligned}
\textstyle\sum_e p_e \text{load}_w(e) &= \textstyle\sum_e p_e \cdot \sum_f \text{load}_w(f \to e) \\
&= \textstyle\sum_e p_e \cdot w_e/u_e \cdot \sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of load)} \\
&= \textstyle\sum_e p_e \cdot u_e \cdot (p_e + 1/m)^{-1} \cdot \sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of } w_e\text{)} \\
&\leq \textstyle\sum_e \sum_f u_e u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \textstyle\sum_{e,f} u_e u_f/\sqrt{w_e w_f} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \textstyle\sum_{e,f} \ell_e \ell_f \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by choice of } \ell_e\text{)} \\
&\leq \alpha_{\text{LOCAL}} \cdot \|\ell\|_2^2 && \text{(by Lemma 2)} \\
&= \alpha_{\text{LOCAL}} \cdot \textstyle\sum_e u_e^2/w_e \\
&= \alpha_{\text{LOCAL}} \cdot \textstyle\sum_e (p_e + 1/m) \\
&= 2\alpha_{\text{LOCAL}}
\end{aligned}$$

◀

8.2 Bound on Width

We will use Lemma 11 again in our proof of the following lemma.

▶ **Lemma 20.** *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = u_e^2 \cdot (p_e + 1/m)^{-1}$ satisfies $\text{load}_w(e) \leq \sqrt{2m}$ for every edge e.*

Proof. Note that by definition of w_e, we have $u_e = \sqrt{w_e \cdot (p_e + 1/m)}$. Fixing an edge e,

$$\begin{aligned}
\text{load}_w(e) &= {}^{w_e}/_{u_e} \cdot \textstyle\sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&\leq \sqrt{{}^{w_e}/_{(p_e+1/m)}} \cdot \textstyle\sum_f \sqrt{w_f \cdot (p_f + 1/m)} \cdot |b_e L^\dagger b_f^\mathsf{T}| \qquad \text{(by definition of } u_e \text{ and } u_f\text{)} \\
&\leq \textstyle\sum_f \sqrt{{}^{(p_f+1/m)}/_{(p_e+1/m)}} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&\leq \sqrt{\textstyle\sum_f {}^{(p_f+1/m)}/_{(p_e+1/m)}} \cdot \sqrt{\textstyle\sum_f w_e w_f \cdot |b_e L^\dagger b_f^\mathsf{T}|^2} \qquad \text{(by Cauchy-Schwarz)}
\end{aligned}$$

We now bound each of the two terms separately. For the first term, note that

$$(p_e + 1/m)^{-1} \cdot \textstyle\sum_f (p_f + 1/m) \leq 2 \cdot (p_e + 1/m)^{-1} \leq 2 \cdot {}^1/_{(1/m)} = 2m.$$

For the second term, by Lemma 11, we know that

$$\textstyle\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \leq 1.$$

Putting these two inequalities together, we get that $\text{load}_w(e) \leq \sqrt{2m}$ for any edge e, which gives the desired bound of $\sqrt{2m}$ on the width. ◀

8.3 Capacitated GetApproxLoad

Algorithm 3 GetApproxLoad, to compute approximate loads for electrical routing.

Input: A graph G, weights $\{w_e\}_{e \in E}$ and capacities $\{u_e\}_{e \in E}$ on the edges, approximation factor ϵ.
Output: Approximation $\{\text{aload}_w(e)\}_{e \in E}$ to the load on the edges.

1 Let B be the edge-vertex incidence matrix of G
2 Let $L := B^\mathsf{T} \text{diag}(w) B$ be the Laplacian matrix
3 Set $C \leftarrow \text{SketchMatrix}(m, n^{-10}, \epsilon)$
4 Set $X \leftarrow B^\mathsf{T} U C^\mathsf{T}$
5 Let $X^{(i)}$ be the i^{th} column of X for all $i \in [\ell]$
6 Set $V^{(i)} \leftarrow \text{LapSolve}(L, X^{(i)})$ for all $i \in [\ell]$
7 Set $V \leftarrow (V^{(1)}, V^{(2)}, \ldots, V^{(\ell)})$ ▷ $V = (CUBL^\dagger)^\mathsf{T}$
8 Set $\text{aload}_w(e) \leftarrow w_e u_e^{-1} \cdot \text{RecoverNorm}(U^\mathsf{T} b_e)$ for all $e \in E$
9 **return** aload_w

Algorithm 3 contains the changes to GetApproxLoad for the capacitated case. Correctness follows from noting that $\text{load}_w(e)$ is the ℓ_1 norm of the e^{th} row of $U^{-1} W B L^\dagger B^\mathsf{T} U$.

8.4 Worst-case demands

We present the proof of the worst-case demands for oblivious routing on a capacitated graph $G = (V, E, u)$ being $B^\mathsf{T} U$ from Kelner and Maymounkov [17, Theorem 3.1], using $\{u_e\}$ for the edge capacities, and M as any linear oblivious routing.

Since congestion of a linear oblivious routing scales linearly when the demands are multiplicatively increased, it suffices to consider demands that can be routed optimally with congestion 1. Let $\{\chi_i\}_i$ be some set of demands that can be optimally routed with congestion 1, and let $\{f_i\}_i$ be such an optimal routing. The claim follows from noting that demands $\{\sum_i |f_{i,e}|\}_e$, i.e., demands of $\sum_i |f_{i,e}|$ across each edge e, can still be (non-linearly) routed with congestion 1.

$$
\begin{aligned}
\beta_\infty(M) &\leq \max_{e'} u_{e'}^{-1} \cdot \sum_i |b_{e'} M \chi_i| && \text{for any } \{\chi_i\}_i \text{ with } \mathrm{OPT}(\{\chi_i\}_i) = 1 \\
&= \max_{e'} u_{e'}^{-1} \cdot \sum_i \left| b_{e'} M \left(\sum_e f_{i,e} b_e^\mathsf{T} \right) \right| && \text{(since } f \text{ routes } \{\chi_i\}_i\text{)} \\
&= \max_{e'} u_{e'}^{-1} \cdot \sum_i \left| \sum_e f_{i,e} \cdot b_{e'} M b_e^\mathsf{T} \right| && \text{(by linearity of } M\text{)} \\
&\leq \max_{e'} u_{e'}^{-1} \cdot \sum_{i,e} \left| f_{i,e} \cdot b_{e'} M b_e^\mathsf{T} \right| && \text{(since } |\textstyle\sum \cdot| \leq \sum |\cdot|\text{)} \\
&= \max_{e'} u_{e'}^{-1} \cdot \sum_e \left| \sum_i |f_{i,e}| \cdot b_{e'} M b_e^\mathsf{T} \right| \\
&\leq \max_{e'} u_{e'}^{-1} \cdot \sum_e \left| u_e \cdot b_{e'} M b_e^\mathsf{T} \right| && \text{(since } \{f_i\}_i \text{ has congestion 1)} \\
&= \max_{e'} u_{e'}^{-1} \cdot \sum_e \left| \cdot b_{e'} M (u_e b_e^\mathsf{T}) \right| \\
&= \| U^{-1} M B^\mathsf{T} U \|_\infty
\end{aligned}
$$

as required.

References

1. Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *Symposium on Foundations of Computer Science (FOCS)*, pages 781–790, 2008. doi:10.1109/FOCS.2008.62.
2. David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Symposium on Communications Architectures & Protocols (SIGCOMM)*, pages 313–324, 2003. doi:10.1145/863955.863991.
3. Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. doi:10.4086/toc.2012.v008a006.
4. Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 93–104, 2020. doi:10.1109/FOCS46700.2020.00018.
5. 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 *Symposium on Theory of Computing (STOC)*, pages 273–282, 2011. doi:10.1145/1993636.1993674.
6. Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In *Symposium on Discrete Algorithms (SODA)*, pages 124–153, 2022. doi:10.1137/1.9781611977073.7.
7. Matthias Englert and Harald Räcke. Oblivious routing for the lp-norm. In *Symposium on Foundations of Computer Science (FOCS)*, pages 32–40, 2009. doi:10.1109/FOCS.2009.52.
8. Jittat Fakcharoenphol, Satish B. Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Symposium on Theory of Computing (STOC)*, pages 448–455, 2003. doi:10.1145/780542.780608.

9 Sebastian Forster, Gramoz Goranci, Yang P. Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. Minor sparsifiers and the distributed laplacian paradigm. In *Symposium on Foundations of Computer Science (FOCS)*, pages 989–999, 2021. doi:10.1109/FOCS52979.2021.00099.

10 Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *Symposium on Foundations of Computer Science (FOCS)*, pages 516–527, 2021. doi:10.1109/FOCS52979.2021.00058.

11 Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Symposium on Theory of Computing (STOC)*, 2021. doi:10.1145/3406325.3451098.

12 Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Symposium on Theory of Computing (STOC)*, pages 1325–1338, 2022. doi:10.1145/3519935.3520026.

13 Chris Harrelson, Kirsten Hildrum, and Satish B. Rao. A polynomial-time tree decomposition to minimize congestion. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 34–43, 2003. doi:10.1145/777412.777419.

14 Prahladh Harsha, Thomas P. Hayes, Hariharan Narayanan, Harald Räcke, and Jaikumar Radhakrishnan. Minimizing average latency in oblivious routing. In *Symposium on Discrete Algorithms (SODA)*, pages 200–207, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347105.

15 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006. doi:10.1145/1147954.1147955.

16 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Symposium on Discrete Algorithms (SODA)*, pages 217–226, 2014. doi:10.1137/1.9781611973402.16.

17 Jonathan A. Kelner and Petar Maymounkov. Electric routing and concurrent flow cutting. *Theor. Comput. Sci.*, 412(32):4123–4135, 2011. doi:10.1016/j.tcs.2010.06.013.

18 Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *USENIX Conference on Networked Systems Design and Implementation*, pages 157–170, 2018. URL: https://www.usenix.org/conference/nsdi18/presentation/kumar.

19 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Symposium on Theory of Computing (STOC)*, pages 842–850, 2016. doi:10.1145/2897518.2897640.

20 Gregory Lawler and Hariharan Narayanan. Mixing times and lp bounds for oblivious routing. In *Meeting on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 66–74, 2009. doi:10.1137/1.9781611972993.10.

21 Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Symposium on Theory of Computing (STOC)*, pages 755–764, 2013. doi:10.1145/2488608.2488704.

22 Huan Li and Aaron Schild. Spectral subspace sparsification. In *Symposium on Foundations of Computer Science (FOCS)*, pages 385–396, 2018. doi:10.1109/FOCS.2018.00044.

23 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *Symposium on Foundations of Computer Science (FOCS)*, pages 593–602, 2016. doi:10.1109/FOCS.2016.70.

24 Richard Peng. Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time. In *Symposium on Discrete Algorithms (SODA)*, pages 1862–1867, 2016. doi:10.1137/1.9781611974331.CH130.

25 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Symposium on Theory of Computing (STOC)*, pages 333–342, 2014. doi:10.1145/2591796.2591832.

26 Harald Räcke. Minimizing congestion in general networks. In *Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002. doi:10.1109/SFCS.2002.1181881.

27 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Symposium on Theory of Computing (STOC)*, pages 255–264, 2008. doi:10.1145/1374376.1374415.

28 Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Symposium on Discrete Algorithms (SODA)*, pages 227–238, 2014. doi:10.1137/1.9781611973402.17.

29 Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 227–238. SIAM, 2014.

30 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Symposium on Theory of Computing (STOC)*, pages 214–227, 2018. doi:10.1145/3188745.3188852.

31 Aaron Schild, Satish Rao, and Nikhil Srivastava. Localization of electrical flows. In *Symposium on Discrete Algorithms (SODA)*, pages 1577–1584, 2018. doi:10.1137/1.9781611975031.103.

32 Aaron Sidford and Yin Tat Lee. Personal communication, 2022.

33 Ali Sinop, Lisa Fawcett, Sreenivas Gollapudi, and Kostas Kollias. Robust routing using electrical flows. In *Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, 2021. doi:10.1145/3474717.3483961.

34 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Symposium on Theory of Computing (STOC)*, pages 81–90, 2004. doi:10.1145/1007352.1007372.

35 Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *Symposium on Theory of Computing (STOC)*, pages 263–277, 1981. doi:10.1145/800076.802479.

36 Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Symposium on Foundations of Computer Science (FOCS)*, pages 543–556, 2022. doi:10.1145/3519935.3520068.

37 Goran Zuzic, Bernhard Haeupler, and Antti Roeyskoe. Sparse semi-oblivious routing: Few random paths suffice, 2023. arXiv:2301.06647.