An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations

Citation	Kelner, Jonathan A., Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. “An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and Its Multicommodity Generalizations.” Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (December 18, 2013): 217–226. © 2014 Society for Industrial and Applied Mathematics
As Published	http://dx.doi.org/10.1137/1.9781611973402.16
Publisher	Society for Industrial and Applied Mathematics
Version	Final published version
Citable link	http://hdl.handle.net/1721.1/92917
Terms of Use	Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.

An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations [*]

Jonathan A. Kelner[†] Yin Tat Lee[‡] Lorenzo Orecchia[§] Aaron Sidford[¶]

Abstract

In this paper, we introduce a new framework for approximately solving flow problems in capacitated, undirected graphs and apply it to provide asymptotically faster algorithms for the maximum *s*-*t* flow and maximum concurrent multicommodity flow problems. For graphs with n vertices and m edges, it allows us to find an ε-approximate maximum *s*-*t* flow in time $O(m^{1+o(1)}\varepsilon^{-2})$, improving on the previous best bound of $\widetilde{O}(mn^{1/3}\mathsf{poly}(\varepsilon^{-1}))$. Applying the same framework in the multicommodity setting solves a maximum concurrent multicommodity flow problem with k commodities in $O(m^{1+o(1)}\varepsilon^{-2}k^2)$ time, improving on the existing bound of $\widetilde{O}(m^{4/3}\mathsf{poly}(k,\varepsilon^{-1}))$.

Our algorithms utilize several new technical tools that we believe may be of independent interest:

- We give a non-Euclidean generalization of gradient descent and provide bounds on its performance. Using this, we show how to reduce approximate maximum flow and maximum concurrent flow to oblivious routing.
- We define and provide an efficient construction of a new type of *flow sparsifier*. Previous sparsifier constructions approximately preserved the size of cuts and, by duality, the *value* of the maximum flows as well. However, they did not provide any direct way to route flows in the sparsifier G' back in the original graph G, leading to a longstanding gap between the efficacy of sparsification on flow and cut problems. We ameliorate this by constructing a sparsifier G' that can be embedded (very efficiently) into G with low congestion, allowing one to transfer flows from G' back to G.
- We give the first almost-linear-time construction of an $O(m^{o(1)})$-competitive oblivious routing scheme. No previous such algorithm ran in time better than $\widetilde{\Omega}(mn)$. By reducing the running time to almost-linear, our work provides a powerful new primitive for constructing very fast graph algorithms.

The interested reader is referred to the full version of the paper [8] for a more complete treatment of these results.

1 Introduction

In this paper, we introduce a new framework for approximately solving flow problems in capacitated, undirected graphs and apply it to provide asymptotically faster algorithms for the **maximum *s*-*t* flow** and **maximum concurrent multicommodity flow** problems. For graphs with n vertices and m edges, it allows us to find an ε-approximately maximum *s*-*t* flows in time $O(m^{1+o(1)}\varepsilon^{-2})$, improving on the previous best bound of $\widetilde{O}(mn^{1/3}\mathsf{poly}(\varepsilon^{-1}))$[4]. Applying the same framework in the multicommodity setting solves a maximum concurrent multicommodity flow problem with k commodities in $O(m^{1+o(1)}\varepsilon^{-2}k^2)$ time, improving on the existing bound of $\widetilde{O}(m^{4/3}\mathsf{poly}(k,\varepsilon^{-1}))$[7].

We believe that both our general framework and several of the pieces necessary for its present instantiation are of independent interest and will find other applications. These include:

- a non-Euclidean generalization of gradient descent, bounds on its performance, and a way to use this to reduce approximate maximum flow and maximum concurrent flow to oblivious routing;
- the definition and efficient construction of a new type of *flow sparsifier*;
- the first almost-linear-time construction of an $O(m^{o(1)})$-competitive oblivious routing scheme.

The interested reader is referred to the full version of the paper [8] for a more complete treatment of these results.

1.1 Related Work For the first several decades of its study, the fastest algorithms for the maximum flow problem were essentially all deterministic algorithms based on combinatorial techniques, culminating in the work of Goldberg and Rao [5], which computes exact maximum flows in time $O(\min(n^{2/3}, m^{1/2})\log(n^2/m)\log U)$ on graphs with edge weights in $\{0,\ldots,U\}$. We refer the reader to [5] for a survey of these results.

More recently, Christiano *et al.*[4] introduced a new linear algebraic approach to the problem in which one

[*] This work was partially supported by NSF awards 0843915 and 1111109, NSF award 1319460, a Sloan Fellowship, NSF Graduate Research Fellowship (grant no. 1122374) and Hong Kong RGC grant 2150701.

[†] Massachusetts Institute of Technology.
[‡] Massachusetts Institute of Technology
[§] Massachusetts Institute of Technology.
[¶] Massachusetts Institute of Technology.

treats the edges of a graph as electrical resistors and computes a sequence of *electrical flows* using fast algorithms for solving Laplacian linear systems [11, 12, 9, 15]. They used this to find ε-approximately maximum s-t flows in time $\tilde{O}(mn^{1/3}\mathrm{poly}(1/\varepsilon))$. Kelner, Miller, and Peng [7] showed how to generalize this approach to solve the maximum concurrent multicommodity flow problem in time $\tilde{O}(m^{4/3}\mathrm{poly}(k, 1/\varepsilon))$. In later work, Lee, Rao, and Srivastava [14] showed how to use electrical flows in a different iterative framework, to obtain a better dependence on ε in unweighted graphs and allowed them to solve the problem *exactly* in unweighted graphs with maximum flow F in time $\tilde{O}(m^{5/4}F^{1/4})$, which is the fastest in certain parameter regimes.

Our algorithm draws extensively on the intellectual heritage established by these works and several others, and many of our technical tools were motivated by barriers faced by these earlier techniques; we refer the reader to the full paper [8] for a more in-depth discussion.

In simultaneous, independent work [22], Jonah Sherman used somewhat different techniques to find another almost-linear-time algorithm for the (single-commodity) maximum flow problem. His approach is essentially dual to ours: Our algorithm maintains a flow that routes the given demands throughout its execution and iteratively works to improve its congestion. Our main technical tools thus consist of efficient methods for finding ways to route flow in the graph while maintaining flow conservation. Sherman, on the other hand, maintains a flow that does *not* route the given demands, along with a bound on the congestion required to route the excess flow at the vertices. He then uses this to iteratively work towards achieving flow conservation. (In a sense, our algorithm is more in the spirit of augmenting paths, whereas his is more like preflow-push.) As such, his main technical tools are efficient methods for producing dual objects that give congestion bounds. Objects meeting many of his requirements were given in the work of Madry [17] (whereas there were no previous constructions of flow-based analogues, requiring us to start from scratch); leveraging these allows him to avoid some of the technical complexity required by our approach. We believe that these paper nicely complement each other, and we enthusiastically refer the reader to Sherman's paper.

1.2 Our Approach In this section, we give a high-level description of our approach. To simplify the exposition, for the remainder of the introduction we focus on the maximum s-t flow problem, and suppose for now that all of the edges have capacity 1. Our problem is then to send as many units of flow as possible from s to t without sending more than one unit over any edge.

Our algorithm works with the equivalent congestion minimization problem, where we try to find the unit s-t flow $\vec{f}$ (i.e., a flow sending one unit from s to t) that minimizes $\|\vec{f}\|_\infty = \max_e |\vec{f}_e|$. Beginning with some initial unit s-t flow $\vec{f_0}$ we give an iterative algorithm to approximately find the circulation $\vec{c}$ to add to $\vec{f_0}$ that minimizes $\|\vec{f_0} + \vec{c}\|_\infty$. Our algorithm takes $2^{O(\sqrt{\log n \log\log n})}\varepsilon^{-2}$ iterations, each of which take $m \cdot 2^{O(\sqrt{\log n \log\log n})}$ time to add a new circulation to the present flow. Constructing this scheme consists of two main parts: an iterative scheme that reduces the problem to constructing a projection matrix with certain properties; and constructing such an matrix.

The iterative scheme: Non-Euclidean gradient descent The simplest way to improve the flow would be to just perform gradient descent on $\|\vec{f} + \vec{c}\|_\infty$. There are two problems with this:

First, gradient descent depends on having a smoothly varying gradient, but ℓ_∞ is very far from smooth. This is easily remedied by a standard technique: we replace the infinity norm with a smoother "soft max" function. Doing this would lead to an update that would be a linear projection onto the space of circulations. This could be computed using an electrical flow, and the resulting algorithm would be very similar to the unaccelerated gradient descent algorithm in [14].

The more serious problem is the difference between ℓ_2 and ℓ_∞. Gradient steps choose a direction by optimizing a local approximation of the objective function over a sphere, whereas the ℓ_∞ constraint asks us to optimize over a cube. The difference between the size of the largest sphere inside a cube and the smallest sphere containing it gives rise to an inherent $O(\sqrt{m})$ in the number of iterations, unless one can exploit additional structure.

To deal with this, we introduce and analyze a non-Euclidean variant of gradient descent that operates with respect to an arbitrary norm.[1] Rather than choosing the direction by optimizing a local linearization of the objective function over the sphere, it performs an optimization over the unit ball in the given norm. By taking this norm to be ℓ_∞ instead of ℓ_2, we obtain a much smaller bound on the number of iterations, albeit at the expense of having to solve a nonlinear minimization problem at every step. The number

[1] This idea and analysis seems to be implicit in other work, e.g., [20]. However, we could not find a clean statement like the one we need in the literature and we have not seen it previously applied in similar settings. We believe that it will find further applications, so we state it in general terms before specializing to what we need.

of iterations required by the gradient descent method depends on how quickly the gradient can change over balls in the norm we are using, which we express in terms of the Lipschitz constant of the gradient in the chosen norm.

To apply this to our problem, we write flows meeting our demands as $\vec{f_0}+\vec{c}$, as described above. We then need a parametrization of the space of circulations so that the objective function (after being smoothed using soft max) has a good bound on its Lipschitz constant. Similarly to what occurs in [9], this comes down to finding a good linear representation of the space of circulations, which we show amounts in the present setting to finding a matrix that projects into the space of circulations while meetings certain norm bounds.

Constructing a projection matrix This reduces our problem to the construction of such a projection matrix. We show how to construct such a projection matrix from any *linear oblivious routing scheme* A with a good competitive ratio.[2] This leads to an iterative algorithm that converges in a small number of iterations. Each of these iterations performs a matrix-vector multiplication with both A and A^T. Intuitively, this is letting us replace the electrical flows used in previous algorithms with the flows given by an oblivious routing scheme. Since the oblivious routing scheme was constructed to meet ℓ_∞ guarantees, while the electrical flow could only obtain such guarantees by relating ℓ_2 to ℓ_∞, it is quite reasonable that we should expect this to lead to a better iterative algorithm.

However, the computation involved in existing oblivious routing schemes is not fast enough to be used in this setting. Our task thus becomes constructing an oblivious routing scheme that we can compute and work with very efficiently. To this end, we show that if G can be embedded with low congestion into H (existentially), and H can be embedded with low congestion into G *efficiently*, one can use an oblivious routing on H to obtain an oblivious routing on G. The crucial difference between the simplification operations we perform here and those in previous papers (e.g., in the work of Benczur-Karger [2] and Madry [17]) is that ours are accompanied by such embeddings, which enables us to transfer flows from the simpler graphs to the more complicated ones.

We construct our routing scheme by recursively composing two types of reductions, each of which we show how to implement without incurring a large increase in the competitive ratio:

- **Vertex elimination** This shows how to efficiently reduce oblivious routing on a graph $G = (V, E)$ to routing on t graphs with roughly $\tilde{O}(|E|/t)$ vertices.

  To do this, we show how to efficiently embed G into t simpler graphs, each consisting of a tree plus a subgraph supported on roughly $\tilde{O}(|E|/t)$ vertices. This follows easily from a careful reading of Madry's paper [17]. We then show that routing on such a graph can be reduced to routing on a graph with at most $\tilde{O}(|E|/t)$ vertices by collapsing paths and eliminating leaves.

- **Flow sparsification** This allows us to efficiently reduce oblivious routing on an arbitrary graph to oblivious routing on a graph with $\tilde{O}(n)$ edges, which we call a flow sparsifier.

  To construct flow sparsifiers, we use local partitioning to decompose the graph into well-connected clusters that contain many of the original edges. We then sparsify these clusters using standard techniques and then show that we can embed the sparse graph back into the original graph using electrical flows. If the graph was originally dense, this results in a sparser graph, and we can recurse on the result. While the implementation of these steps is somewhat different, the outline of this construction parallels Spielman and Teng's approach to the construction of spectral sparsifiers [23, 25].

Furthermore, to solve the maximum concurrent multicommodity flow problem, we apply the same framework, modifying the norm and regularization appropriately.

2 Preliminaries

General Notation: For $\vec{x} \in \mathbb{R}^n$, we let $|\vec{x}| \in \mathbb{R}^n$ be such that $\forall i$, $|\vec{x}|_i \stackrel{\mathrm{def}}{=} |\vec{x}_i|$. For $\mathbf{A} \in \mathbb{R}^{n\times m}$, we let $|\mathbf{A}| \in \mathbb{R}^{n\times m}$ be such that $\forall i, j$, $|\mathbf{A}|_{ij} \stackrel{\mathrm{def}}{=} |\mathbf{A}_{ij}|$.

Graphs: Throughout this paper we let $G = (V, E, \vec{\mu})$ denote an undirected capacitated graph with $n = |V|$ vertices, $m = |E|$ edges, and non-negative capacities $\vec{\mu} \in \mathbb{R}^E$. We let $w_e \geq 0$ denote the weight of an edge and let $r_e \stackrel{\mathrm{def}}{=} 1/w_e$ denote the resistance of an edge. For $S \subseteq V$ we let $G(S)$ denote the subgraph of G consisting of vertices S and all the edges of E with both endpoints in S, i.e. $\{(a, b) \in E \mid a, b \in S\}$. We use subscripts to make the graph under consideration

[2] A linear oblivious routing scheme maps a collection of demands to a multicommodity flow meeting these demands by routing each demand vector using a pre-specified operator, independently of the others. The competitive ratio of such an operator is the worst possible ratio between the congestion incurred by a set of demands in this scheme and the congestion of the best multicommodity flow routing these demands. In [21], Räcke showed how to construct an oblivious routing scheme with a competitive ratio of $O(\log n)$.

clear, e.g. $\mathrm{vol}_{G(S)}(A)$ denotes the volume of vertex set A in the subgraph of G induced by S.

Fundamental Matrices: We let $\mathbf{U}, \mathbf{W}, \mathbf{R} \in \mathbb{R}^{E \times E}$ denote the diagonal matrices associated with the capacities, the weights, and the resistances respectively. While all graphs in this paper are undirected, we assume an arbitrary orientation for notational convenience and we let $\mathbf{B} \in \mathbb{R}^{E \times V}$ denote the graphs incidence matrix where for all $e = (a,b) \in E$ we have $\mathbf{B}^T \vec{\mathbb{1}}_e = \vec{\mathbb{1}}_a - \vec{\mathbb{1}}_b$.

Matrices: Let $\| \cdot \|$ be a family of norms applicable to $\mathbb{R}^n$ for any n. We define this norms' *induced norm* or *operator norm* on the set of of $m \times n$ matrices by $\|\mathbf{A}\| \stackrel{\text{def}}{=} \max_{\vec{x} \in \mathbb{R}^n} \|\mathbf{A}\vec{x}\| / \|\vec{x}\|$. For matrix $\mathbf{A}$, we let $\mathcal{T}(\mathbf{A})$ denote the maximum time needed to apply $\mathbf{A}$ or $\mathbf{A}^T$ to a vector.

Cuts: For $S \subseteq V$ we denote the cut induced by S by edge subset $\partial(S) \stackrel{\text{def}}{=} \{e \in E \mid e \not\subseteq S \text{ and } e \not\subseteq E \setminus S\}$ and we denote the cost of $F \subseteq E$ by $w(F) \stackrel{\text{def}}{=} \sum_{e \in F} w_e$. For $a \in V$ we let $d_a \stackrel{\text{def}}{=} \sum_{\{a,b\}} w_{a,b}$, for $S \subseteq V$ we define its *volume* by $\mathrm{vol}(S) \stackrel{\text{def}}{=} \sum_{a \in V} d_a$. Using these we denote the *conductance* of $S \subseteq V$ by $\Phi(S) \stackrel{\text{def}}{=} \frac{w(\partial(S))}{\min\{\mathrm{vol}(S), \mathrm{vol}(V-S)\}}$ and the conductance of G by $\Phi(G) \stackrel{\text{def}}{=} \min_{S \subseteq V\ : S \notin \{\emptyset, V\}} \phi(S)$.

Flows: Thinking of edge vectors, $\vec{f} \in \mathbb{R}^E$, as flows we let the *congestion of* $\vec{f}$ be given by $\mathrm{cong}(\vec{f}) \stackrel{\text{def}}{=} \|\mathbf{U}^{-1}\vec{f}\|_\infty$. For any collection of flows $\{\vec{f_i}\} = \{\vec{f}_1, \ldots, \vec{f}_k\}$ we overload notation and let their *total congestion* be given by $\mathrm{cong}(\{\vec{f_i}\}) \stackrel{\text{def}}{=} \|\mathbf{U}^{-1} \sum_i |\vec{f_i}|\|_\infty$. We call a vector $\vec{\chi} \in \mathbb{R}^V$ a *demand vector* if $\sum_{a \in V} \vec{\chi}(a) = 0$ and we say $\vec{f} \in \mathbb{R}^E$ *meets demands* $\vec{\chi}$ if $\mathbf{B}^T \vec{f} = \vec{\chi}$. Given a set of demands $D = \{\vec{\chi}_1, \ldots, \vec{\chi}_k\}$, i.e. $\forall i \in [k], \sum_{a \in V} \vec{\chi}_i(a) = 0$, we denote the optimal low congestion routing of these demands as follows $\mathrm{opt}(D) \stackrel{\text{def}}{=} \min_{\{\vec{f_i}\} \in \mathbb{R}^E\ :\ \{\mathbf{B}^T \vec{f_i}\} = \{\vec{\chi}_i\}} \mathrm{cong}(\{\vec{f_i}\})$ We call a set of flows $\{\vec{f_i}\}$ that meet demands $\{\vec{\chi}_i\}$, i.e. $\forall i, \mathbf{B}^T \vec{f_i} = \vec{\chi}_i$, a *multicommodity flow* meeting the demands.

3 Framework for Solving Max-Flow

Here we present the *gradient descent method for general norms* and show how to use this method to solve undirected maximum flow approximately given access to a *circulation projection matrix.*

3.1 Gradient Descent:

Let $\| \cdot \| : \mathbb{R}^n \to \mathbb{R}$ be an arbitrary norm on $\mathbb{R}^n$ and recall that the gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ at $\vec{x}$ is defined as the vector $\nabla f(\vec{x}) \in \mathbb{R}^n$ such that

$$f(\vec{y}) = f(\vec{x}) + \langle \nabla f(\vec{x}), \vec{y} - \vec{x} \rangle + o(\|\vec{y} - \vec{x}\|). \tag{3.1}$$

The gradient descent method is a greedy minimization method for unconstrained smooth minimization that updates a current vector, $\vec{x}$, using the direction which minimizes $\langle f(\vec{x}), \vec{y} - \vec{x} \rangle$.

To analyze this method's performance, we need to compare the improvement $\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \rangle$ with the step size, $\|\vec{y} - \vec{x}\|$, and the quantity, $\|\nabla f(\vec{x})\|$. For the Euclidian norm, this can be done by Cauchy Schwarz inequality and in general, we can define a new norm for $\bigtriangledown f(\vec{x})$ to make this happens. We call this the *dual norm* $\| \cdot \|^*$ defined by $\|\vec{x}\|^* \stackrel{\text{def}}{=} \max_{\vec{y} \in \mathbb{R}^n} \langle \vec{y}, \vec{x} \rangle$ such that $\|\vec{y}\| \leq 1$. In the full version we show that this definition indeed yields that $\langle \vec{y}, \vec{x} \rangle \leq \|\vec{y}\|^* \|\vec{x}\|$.

Letting $x^{\#} \in \mathbb{R}^n$ denote the fastest increasing direction for this norm, i.e., $x^{\#}$ is an arbitrary point satisfying $\vec{x}^{\#} \stackrel{\text{def}}{=} \arg\max_{\vec{s} \in \mathbb{R}} \langle \vec{x}, \vec{s} \rangle - \frac{1}{2}\|\vec{s}\|^2$, the *gradient descent method* simply produces a sequence of $\vec{x}_k$ such that $\vec{x}_{k+1} := \vec{x}_k - t_k(\bigtriangledown f(\vec{x}_k))^{\#}$ where $t_k \in \mathbb{R}$ is some chosen step size for iteration k. To determine what these step sizes should be we need some information about the smoothness of the function, in particular, the magnitude of the second order term in (3.1). The natural notion of smoothness for gradient descent is the Lipschitz constant of the gradient of f, that is the smallest constant L such that $\forall \vec{x}, \vec{y} \in \mathbb{R}^n\ \| \bigtriangledown f(\vec{x}) - \bigtriangledown f(\vec{y})\|^* \leq L \cdot \|\vec{x} - \vec{y}\|$. Picking the t_k to be the optimal value to guarantee progress for a single step we get the gradient descent method which has the following convergence rate.

Theorem 3.1. (Gradient Descent) *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex continuously differentiable function with a non-empty set of optimal solutions X^* and minimal value f^*. Let L be the Lipschitz constant of ∇f. Then for initial point $\vec{x}_0 \in \mathbb{R}^n$ we define a sequence of $\vec{x}_k$ by the update rule $\vec{x}_{k+1} := \vec{x}_k - \frac{1}{L}(\bigtriangledown f(\vec{x}_k))^{\#}$. For all $k \geq 0$, we have*

$$f(\vec{x}_k) - f^* \leq \frac{2 \cdot L \cdot R^2}{k+4}$$

where

$$R \stackrel{\text{def}}{=} \max_{\vec{x} \in \mathbb{R}^n : f(\vec{x}) \leq f(\vec{x}_0)} \min_{\vec{x}^* \in X^*} \|\vec{x} - \vec{x}^*\|.$$

Proof. A proof of this fact modeled after a proof in [20] appears in the Appendix.

3.2 Circulation Projection: In order to solve maximum flow using (non-Euclidian) gradient descent we need to convert the problem to a more manageable form. Recall that given demand vector $\vec{\chi} \in \mathbb{R}^V$ the *maximum flow* problem is as follows

$$\max_{\alpha \in R, \vec{f} \in \mathbb{R}^E} \alpha \qquad \text{s.t} \qquad \mathbf{B}^T \vec{f} = \alpha \vec{\chi} \text{ and } \|\mathbf{U}^{-1} \vec{f}\|_\infty \leq 1 \ .$$

By scaling the answer we see that we could instead solve the equivalent *minimum congestion flow* problem $\min_{\vec{f} \in \mathbb{R}^E \,:\, \mathbf{B}^T \vec{f} = \vec{\chi}} \|\mathbf{U}^{-1} \vec{f}\|_\infty$. Furthermore, if we have some initial *feasible flow* $\vec{f_0} \in \mathbb{R}^E$, i.e. $\mathbf{B}^T \vec{f_0} \stackrel{\text{def}}{=} \vec{\chi}$, then we can write this problem equivalently as $\min_{\vec{c} \in \mathbb{R}^E} \|\mathbf{U}^{-1}(\vec{f_0} + \vec{c})\|_\infty$ such to the constrain that $\mathbf{B}^T \vec{c} = \vec{0}$ where the output flow is $\vec{f} = \vec{f_0} + \vec{c}$.

Note that these are constrained minimization problems and while there are variants of gradient descent applicable to constrained optimization naive application of these techniques may be complicated for our purposes. Instead, we simply note that the $\mathbf{B}^T \vec{f} = \vec{0}$ is a linear subspace and we can avoid the constraints by fixing a matrix that projects the variables onto this subspace. We define such a *circulation projection matrix* as follows.

Definition 3.1. (Circulation Projection) *A matrix* $\tilde{\mathbf{P}} \in \mathbb{R}^{E \times E}$ *is a* circulation projection matrix *if it is a projection matrix onto the circulation space, i.e.* $\forall \vec{x} \in \mathbb{R}^E$ *we have* $\mathbf{B}^T \tilde{\mathbf{P}} \vec{x} = \vec{0}$ *and* $\mathbf{B}^T \vec{x} = \vec{0} \Rightarrow \mathbf{P} \vec{x} = \vec{x}$.

Given a circulation projection we can reformulate the problem as $\min_{\vec{c} \in \mathbb{R}^E} \|\mathbf{U}^{-1}(\vec{f_0} + \tilde{\mathbf{P}} \vec{c})\|_\infty$. By applying the change of basis $\vec{x} = \mathbf{U}^{-1} \vec{c}$ and letting $\vec{\alpha_0} = \mathbf{U}^{-1} \vec{f_0}$ and $\mathbf{P} = \mathbf{U}^{-1} \tilde{\mathbf{P}} \mathbf{U}$, we can get a simple formulation of maximum flow as $\min_{\vec{x} \in \mathbb{R}^E} \|\vec{\alpha_0} + \mathbf{P} \vec{x}\|_\infty$ where given an approximate solution $\vec{x}$ the output approximate maximum flow is $\vec{f}(\vec{x}) = \mathbf{U}(\vec{\alpha_0} + \mathbf{P}\vec{x}) / \|\mathbf{U}(\vec{\alpha_0} + \mathbf{P}\vec{x})\|_\infty$.

3.3 An Approximate Maximum Flow Algorithm Note that the gradient descent method requires the objective function to be differentiable. So to solve the previous optimization problem using gradient descent we introduce a smooth function smax_t that approximates $\|\cdot\|_\infty$.

Lemma 3.1. (Softmax Properties) *For any $t > 0$ the* softmax function *defined for all $\vec{x} \in \mathbb{R}^E$ by*

$$\text{smax}_t(\vec{x}) \stackrel{\text{def}}{=} t \ln \left(\frac{1}{2m} \sum_{e \in E} \exp\left(\frac{\vec{x}_e}{t}\right) + \exp\left(-\frac{\vec{x}_e}{t}\right) \right)$$

is convex and continuously differentiable. Its gradient is Lipschitz continuous with Lipschitz constant $\frac{1}{t}$ and $\|\vec{x}\|_\infty - t\ln(2m) \leq \text{smax}_t(\vec{x}) \leq \|\vec{x}\|_\infty$ for all $\vec{x} \in \mathbb{R}^E$. Furthermore, both smax_t and $\bigtriangledown \text{smax}_t(\vec{x})$ are computable in $O(m)$ time.

Proof. The softmax function and its properties can be derived from smoothing techniques using convex conjugates[19] [3, Sec 5.4]. However, for simplicity and completeness, in the full version [8] we provide a complete proof of this lemma.

By replacing $\|\cdot\|_\infty$ in the previous optimization problem with $\text{smax}_t(\cdot)$ for a particular value of t and applying gradient descent in the $\|\cdot\|_\infty$ norm to this problem, we get the following algorithm.

```
MaxFlow    Input: any feasible flow $\vec{f_0}$.
1. Let $\vec{\alpha_0} = (\mathbf{I} - \mathbf{P})\,\mathbf{U}^{-1} \vec{f_0}$ and $\vec{x_0} = 0$.
2. Let $t = \varepsilon \mathrm{OPT} / 2\ln(2m)$ and $g_t = \text{smax}_t(\vec{\alpha_0} + \mathbf{P}\vec{x})$.
3. For $i = 1, \cdots, 300\|\mathbf{P}\|_\infty^4 \ln(2m)/\varepsilon^2$
4.     $\vec{x}_{i+1} = \vec{x}_i - t\|\mathbf{P}\|_\infty^{-2} (\bigtriangledown g_t(\vec{x}_i))^\#$.
5. Output $\mathbf{U}\,(\vec{\alpha_0} + \mathbf{P}\vec{x}_{\text{last}}) / \|\vec{\alpha_0} + \mathbf{P}\vec{x}_{\text{last}}\|_\infty$.
```

Theorem 3.2. *Let $\widetilde{\mathbf{P}}$ be a circulation projection matrix, let $\mathbf{P} = \mathbf{U}^{-1} \widetilde{\mathbf{P}} \mathbf{U}$, and let $\varepsilon < 1$.* ***MaxFlow*** *outputs an $(1-\varepsilon)$-approximate maximum flow in time* $O\left(\|\mathbf{P}\|_\infty^4 \ln(m) \left(\mathcal{T}(\mathbf{P}) + m\right) \varepsilon^{-2}\right)$.

Proof. To bound the performance of gradient descent we we show that the gradient of g_t is Lipschitz continuous with Lipschitz constant $L = \|\widetilde{\mathbf{P}}\|_\infty^2 / t$. Then, to bound the running time of the algorithm we show that, in $\|\cdot\|_\infty$, each $\vec{x}_e^\# = \text{sign}(\vec{x}_e)\|\vec{x}\|_1$. Combining these facts along with the results in this section yields the result. See [8] for the details.

4 Oblivious Routing

In the next few sections we show how to construct a circulation projection matrix so that application of Theorem 3.2 immediately yields the following theorem.

Theorem 4.1. *We can compute a $1 - \varepsilon$ approximate maximum flow in an undirected capacitated graph $G = (V, E, \vec{\mu})$ with capacity ratio $U = \mathsf{poly}(|V|)$ in time*

$$O(|E| 2^{O\left(\sqrt{\log|V| \log\log|V|}\right)} \varepsilon^{-2}) \ .$$

Our construction focuses on the notion of *(linear) oblivious routings*[3], that is fixed linear mappings from

[3] While non-linear algorithms could be considered, we restrict our attention to the linear case and typically use the term "oblivious routing" to refer to these. Note that the oblivous routing algorithms in [6] [13] [21] are all linear.

demands to flows that meet the demands. Rather than constructing a circulation projection matrix directly, we show how the efficient construction of an oblivious routing algorithm with a good *competitive ratio* immediately allows us to produce a circulation projection matrix.

Definition 4.1. (Oblivious Routing) *An* oblivious routing *on graph $G = (V, E)$ is a linear operator $\mathbf{A} \in \mathbb{R}^{E\times V}$ such that for all demands $\vec{\chi}$ the* routing *of $\vec{\chi}$ by $\mathbf{A}$, $\mathbf{A}\vec{\chi}$, satisfies $\mathbf{B}^T\mathbf{A}\vec{\chi} = \vec{\chi}$.*

Oblivious routings get their name due to the fact that, given an oblivious routing strategy $\mathbf{A}$ and a set of demands $D = \{\vec{\chi}_1, \ldots, \vec{\chi}_k\}$, one can construct a multicommodity flow satisfying all the demands in D by using $\mathbf{A}$ to route each demand individually, obliviously to the existence of the other demands. We measure the *competitive ratio* of such an oblivious routing strategy to be the worst ratio of the congestion of such a routing to the minimal-congestion routing of the demands.

Definition 4.2. (Competitive Ratio) *The* competitive ratio *of oblivious routing $\mathbf{A}$ is given by $\rho(\mathbf{A}) \stackrel{\text{def}}{=} \max_{\{\vec{\chi}_i\}} \text{cong}(\{\mathbf{A}\vec{\chi}_i\})/\text{opt}(\{\vec{\chi}_i\})$ where $\{\vec{\chi}_i\}$ is a set of a demand vectors.*

The competitive ratio of an oblivious routing algorithm can be gleaned from the the operator norm of a related matrix ([13] and [6]). Below we state weighted generalization of this result.

Lemma 4.1. *For any oblivious routing $\mathbf{A}$, we have $\rho(\mathbf{A}) = \|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\|_\infty$*

This allows us to explicitly connect oblivious routings and circulation projection matrices.

Lemma 4.2. *For an oblivious routing $\mathbf{A} \in \mathbb{R}^{E\times V}$ the matrix $\widetilde{\mathbf{P}} \stackrel{\text{def}}{=} \mathbf{I} - \mathbf{A}\mathbf{B}^T$ is a circulation projection matrix such that*

$$\|\mathbf{U}^{-1}\widetilde{\mathbf{P}}\mathbf{U}\|_\infty \leq 1 + \rho(\mathbf{A}) \enspace .$$

Using this lemma and Theorem 3.2 we see that proving the following suffices to prove Theorem 4.1.

Theorem 4.2. (Routing Algorithm) *Given an undirected capacitated graph $G = (V, E, \vec{\mu})$ with capacity ratio $U = O\mathsf{poly}(|V|)$ we can construct an oblivious routing algorithm $\mathbf{A}$ on G in time $O(|E|2^{O(\sqrt{\log|V|\log\log|V|})})$ with $\mathcal{T}(\mathbf{A}) = |E|2^{O(\sqrt{\log|V|\log\log|V|})}$ and $\rho(\mathbf{A}) = 2^{O(\sqrt{\log|V|\log\log|V|})}$.*

We obtain such a routing from a recursive construction. Given a complicated graph we show how to reduce computing an oblivious routing on this graph to computing an oblivious routing on a simpler graph. Key to these constructions will be the notion of an embedding which will allow us to reason about the competitive ratio of oblivious routing algorithms on different graphs.

Definition 4.3. (Embedding and Congestion) *Let $G = (V, E, \vec{\mu})$ and $G' = (V, E', \vec{\mu}')$ denote two undirected capacitated graphs on the same vertex set with incidence matrices $\mathbf{B} \in \mathbb{R}^{E\times V}$ and $\mathbf{B}' \in \mathbb{R}^{E'\times V}$ and capacity matrices μ and μ' respectively. An* embedding *from G to G' is linear operator $\mathbf{M} \in \mathbb{R}^{E'\times E}$ such that $\mathbf{B}'^T\mathbf{M} = \mathbf{B}^T$. The* congestion *of the embedding $\mathbf{M}$ is given by* $\text{cong}(\mathbf{M}) \stackrel{\text{def}}{=} \max_{\vec{x}\in\mathbb{R}^E} \|\mathbf{U}'^{-1}\mathbf{M}\vec{x}\|_\infty / \|\mathbf{U}^{-1}\vec{x}\|_\infty = \|\mathbf{U}'^{-1}|\mathbf{M}|\mathbf{U}\vec{\mathbb{1}}\|_\infty$ *and we say G* embeds *into G' with congestion α if there exists an embedding $\mathbf{M}$ from G to G' such that* $\text{cong}(\mathbf{M}) \leq \alpha$.

Using this concept we prove Theorem 4.2 by recursive application of two techniques. First, in the full version [8] we show how to take an arbitrary graph $G = (V, E)$ and approximate it by a *sparse graph* $G' = (V, E')$ so that flows in G can be routed in G' with low congestion and such that there is an $\widetilde{O}(1)$ embedding from G' to G that can be applied in $\widetilde{O}(|E|)$ time. We prove the following theorem.

Theorem 4.3. (Edge Reduction) *Let $G = (V, E, \vec{\mu})$ be an undirected capacitated graph with capacity ratio $U = \mathsf{poly}(|V|)$. In $\widetilde{O}(|E|)$ time we can construct a graph G' on the same vertex set with at most $\widetilde{O}(|V|)$ edges and capacity ratio at most $U \cdot \mathsf{poly}(|V|)$ such that given an oblivious routing $\mathbf{A}'$ on G' in $\widetilde{O}(|E|)$ time we can construct an oblivious routing $\mathbf{A}$ on G such that $\mathcal{T}(\mathbf{A}) = \widetilde{O}(|E| + \mathcal{T}(\mathbf{A}'))$ and $\rho(\mathbf{A}) = \widetilde{O}(\rho(\mathbf{A}'))$*

Next, in the full version [8] we show how to embed a graph into a collection of graphs consisting of trees plus extra edges. Then we show how to embed these graphs into better structured graphs consisting of trees plus edges so that by simply removing degree 1 and degree 2 vertices we are left with graphs with fewer vertices. Formally, we prove the following.

Theorem 4.4. (Vertex Reduction) *Let $G = (V, E, \vec{\mu})$ be an undirected capacitated graph with capacity ratio U. For all $t > 0$ in $\widetilde{O}(t \cdot |E|)$ time we can compute graphs $G_1, \ldots, G_t$ each with at most $\widetilde{O}(|E|\log(U)/t)$ vertices, at most $|E|$ edges, and capacity ratio at most $|V| \cdot U$ such that given oblivious routings $\mathbf{A}_i$ for each G_i, in $\widetilde{O}(t \cdot |E|)$ time we can compute an oblivious routing $\mathbf{A}$ on G such that $\mathcal{T}(\mathbf{A}) = \widetilde{O}(t \cdot |E| + \sum_{i=1}^{t} \mathcal{T}(\mathbf{A}_i))$ and $\rho(\mathbf{A}) = \widetilde{O}(\max_i \rho(\mathbf{A}_i))$.*

By carefully recursively applying Theorem 4.3 and Theorem 4.4 with the right parameters using simple electric routing as a base case, we prove Theorem 4.2 in the full version [8].

5 Flow Sparsifiers

In order to prove Theorem 4.3, i.e. reduce the problem of efficiently computing a competitive oblivious routing on a dense graph to the same problem on a sparse graph, we introduce a new algorithmic tool called *flow sparsifiers.* [4] A flow sparsifier is an efficient cut-sparsification algorithm that also produces an efficiently-computable low-congestion embedding mapping the sparsified graph back to the original graph.

Definition 5.1. (Flow Sparsifier) *An algorithm is a* (h, ε, α)-flow sparsifier *if on input graph* $G = (V, E, \mu)$ *with capacity ratio* U *it outputs a graph* $G' = (V, E', \mu')$ *with capacity ratio* $U' \leq U \cdot \mathsf{poly}(|V|)$ *and an embedding* $\mathbf{M} : \mathbb{R}^{E'} \to \mathbb{R}^{E}$ *of* G' *into* G *with the following properties:*

- Sparsity: $|E'| \leq h$
- Cut Approximation: $\forall S \subseteq V \;:\; (1-\varepsilon)\mu(\partial_G(S)) \leq \mu'(\partial_{G'}(S)) \leq (1+\varepsilon)\mu(\partial_G(S))$
- Flow Approximation: $\text{cong}(\mathbf{M}) \leq \alpha$
- Efficiency: *The running times needed to compute* G' *as well as* $\mathcal{T}(\mathbf{M})$ *are all* $\tilde{O}(m)$.

In this section we provide the first proof of the following theorem.

Theorem 5.1. *For any constant* $\varepsilon \in (0, 1)$, *there is an* $(\tilde{O}(n), \varepsilon, \tilde{O}(1))$*-flow sparsifier.*

Flow sparsifiers allow us to solve a multi-commodity flow problem on a possibly dense graph G by converting G into a sparse graph G' and solving the flow problem on G', while suffering a loss of a factor of α in the congestion when mapping the solution back to G using $\mathbf{M}$. In the appendix, we show how Theorem 5.1 suffices to yield Theorem 4.3. This proof exploits the cut-approximation condition by applying the flow-cut-gap theorem [1] to compare the congestion of an optimal routing in G' to that of the optimal routing in G.

To prove Theorem 5.1, we follow as similar approach as the spectral sparsifcation algorithm of Spielman and Teng [25] and partition the input graph into vertex sets, such that each set induces a near-expander and most edges of the graph do not cross set boundaries. We then sparsify these induced subgraphs using standard sparsification techniques and iterate on the edges not in the subgraphs. As each iteration removes a constant fraction of the edges, using standard sparsification techniques we immediately obtain the sparsity and cut-approximation properties. To obtain the embedding $\mathbf{M}$ with $\text{cong}(\mathbf{M}) = \tilde{O}(1)$, we prove a generalization of results in [6, 13] showing that electrical-flow routing achieves a low competitive ratio on near-expanders and subsets thereof.

5.1 Routing Subsets of Near-Expanders

We begin by formalizing the notion of oblivious routing on a subgraph. Given an oblivious routing strategy $\mathbf{A}$ and a subset of the edges $F \subseteq E$, we let $\text{opt}^F(\{\vec{\chi}_i\})$ denote the minimal congestion achieved by any routing restricted to only sending flow on edges in F and we denote the F-*competitive* ratio of $\mathbf{A}$ by $\rho^F(\mathbf{A}) \overset{\text{def}}{=} \max_{\{\vec{\chi}_i\}} \frac{\text{cong}(\mathbf{A}\vec{\chi}_i)}{\text{opt}^F(\{\vec{\chi}_i\})}$ where $\{\vec{\chi}_i\}$ are demand vectors routable in F. As before, we can relate $\rho^F(\mathbf{A})$ to operator norms.

Lemma 5.1. *Let* $\vec{\mathbb{1}}_F \in \mathbb{R}^E$ *denote the indicator vector for set* F *(i.e.* $\vec{\mathbb{1}}_F(e) = 1$ *if* $e \in F$ *and* $\vec{\mathbb{1}}_F(e) = 0$*) and let* $\mathbf{I}_F \overset{\text{def}}{=} diag(\vec{\mathbb{1}}_F)$. *For any* $F \subseteq E$ *we have* $\rho^F(\mathbf{A}) = \|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\mathbf{I}_F\|_\infty$

To obtain good F-competitive oblivious routings for certain graphs we study *electrical-flow oblivious routings* defined as follows.

Definition 5.2. (Electric Oblivious Routing) *Consider a graph* $G = (V, E, \mu)$ *with capacity matrix* $\mathbf{U}$ *set edge resistances so that* $\mathbf{R} = \mathbf{U}^{-1}$. *Recalling that* $\mathcal{L} \overset{\text{def}}{=} \mathbf{B}^T\mathbf{R}^{-1}\mathbf{B}$ *the oblivious* electrical-flow routing *strategy is the linear operator* $\mathbf{A}_{\mathcal{E}} \overset{\text{def}}{=} \mathbf{R}^{-1}\mathbf{B}\mathcal{L}^\dagger$.

For electrical-flow routing strategy $\mathbf{A}_{\mathcal{E}}$, the upper bound on the competitive ratio $\rho(\mathbf{A}_{\mathcal{E}})$ in Lemma 4.1 can be rephrased in terms of the voltages induced on G by electrically routing an edge. We extend this interpretation appearing in [6, 13] to the weighted subgraph-routing case below.

Lemma 5.2. *For electrical-flow routing* $\mathbf{A}_{\mathcal{E}}$, *edge subset* $F \subseteq E$, *and resistances* $\mathbf{R} = \mathbf{U}^{-1}$, *we have* $\rho^F(\mathbf{A}_{\mathcal{E}}) = \max_{e\in E} \sum_{(a,b)\in F} r_{ab}^{-1} |v_e(a) - v_e(b)|$, *where* $\vec{v}_e \overset{\text{def}}{=} \mathcal{L}^\dagger \vec{\chi}_e$.

Next, we use Lemma 5.2 to show that $\mathbf{A}_{\mathcal{E}}$ has a good F-competitive ratio provided the edges F are contained within an induced near-expander $G(U) = (U, E(U))$ for some $U \subseteq V$.

[4] Note that our flow sparsifiers aim to reduce the number of edges, and are different from the flow sparsifiers of Leighton and Moitra [16], which work in a different setting and reduce the number of vertices.

Lemma 5.3. *For weighted graph $G = (V, E, w)$ with integer weights, vertex subset $U \subseteq V$, and edge $e \in E$, we have $\rho^F(\mathbf{A}_{\mathcal{E}}) \le 8\log(\mathrm{vol}(G(U)))\Phi(G(U))^{-2}$.*

From this lemma, the following is immediate:

Lemma 5.4. *Let $F \subseteq E$ be contained within some vertex induced subgraph $G(U)$, then for $\mathbf{R} = \mathbf{U}^{-1}$ we have $\rho^F(\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^\dagger) \le \rho^{E(U)}(\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^\dagger) \le 8\log(\mathrm{vol}(G(U)))\Phi(G(U))^{-2}$.*

5.2 Construction and Analysis of Flow Sparsifiers

To prove Theorem 5.1, we show how we can use the framework of Spielman and Teng [25] to partition G into edge subsets that we can route electrically with low competitive ratio using Lemma 5.4. By carefully creating this partitioning, we can bound the congestion of the resulting embedding. By using standard sparsification algorithms, we then obtain the remaining properties needed for Theorem 5.1.

First, we use techniques in [25] to reduce the problem to the unweighted case.

Lemma 5.5. *Given an (h, ε, α)-flow-sparsifier for unweighted graphs, it is possible to construct an $(h \cdot \log U, \varepsilon, \alpha)$-flow-sparsifier for weighted graphs with capacity ratio $U = \mathsf{poly}(|V|)$.*

Next, we show that we can use a decomposition lemma, which is implicit in Spielman and Teng's local clustering approach to spectral sparsification [25] (see [8]), and the above insight to construct a routine that flow-sparsifies a constant fraction of the edges of E.

Lemma 5.6. *Given an unweighted graph $G = (V, E)$ there is an algorithm that runs in $\tilde{O}(m)$ and computes a partition of E into $(F, \bar{F})$, an edge set $F' \subseteq F$ with weight vector $\vec{w_{F'}} \in \mathbb{R}^{F'}$, and an embedding $\mathbf{H} : \mathbb{R}^{F'} \to \mathbb{R}^E$ from $H' = (V, F', w_{F'})$ to G with the following properties:*

1. *F contains most of the volume of G: $|F| \ge \frac{|E|}{2}$;*
2. *F' contains only $\tilde{O}(n)$ edges: $|F'| \le \tilde{O}(n)$.*
3. *The weights $w_{F'}$ are bounded: $\forall e \in F'$, $1/\mathsf{poly}(n) \le w_{F'}(e) \le n$.*
4. *H' ε-cut approximates $H = (V, F, \vec{\mathbb{1}})$.*
5. *The embedding $\mathbf{H}$ has $\mathrm{cong}(\mathbf{H}) = \widetilde{O}(1)$ and runs in $\mathcal{T}(\mathbf{H}) = \tilde{O}(m)$.*

By applying this lemma iteratively we produce the flow-sparsifier with the desired properties.

6 Removing Vertices in Oblivious Routing Construction

Here we prove Theorem 4.4, i.e. reduce computing an efficient oblivious routing for a graph $G = (V, E)$ to computing an oblivious routing for t graphs with $\widetilde{O}(|E|/t)$ vertices and $O(|E|)$ edges. To do this we follow an approach extremely similar to [18] consisting of a series of embeddings to increasingly simple combinatorial objects.

Definition 6.1. (Tree Path and Cut) *For undirected graph $G = (V, E)$, spanning tree T, and $\forall a, b \in V$, let $P_{a,b}$ denote the unique a to b path using only edges in T, let $\partial_T(e) \stackrel{\mathrm{def}}{=} \{e' \in E \mid e' \in P_e\}$ denote the edges cut by $e \in E$, and let $\partial_T(F) \stackrel{\mathrm{def}}{=} \cup_{e \in F}\partial(e)$ denote the edges cut by $F \subseteq E$.*

Definition 6.2. (Partial Tree Embedding) *For undirected capacititated graph $G = (V, E, \vec{\mu})$ spanning T and spanning tree subset $F \subseteq T$ we define the* partial tree embedding graph *$H = H(G, T, F) = (V, E', \vec{\mu}')$ to a be a graph on the same vertex set where $E' = T \cup \partial_T(F)$ and $\vec{\mu}'(e) = \sum_{e' \in E | e \in P_{e'}} \vec{\mu}_{e'}$ if $e \in T \setminus F$ and $\vec{\mu}'(e) = \vec{\mu}(e)$ otherwise. Furthermore, we let $\mathbf{M}_H \in \mathbb{R}^{E' \times E}$ denote the embedding from G to $H(G, T, F)$ where edges not cut by F are routed over the tree and other edges are mapped to themselves and we let $\mathbf{M}'_H \in \mathbb{R}^{E \times E'}$ denote the embeding from H to G that simply maps edges in H to their corresponding edges in G.*

The first step in proving Theorem 4.4 is showing that we can reduce computing oblivious routings in an arbitrary graph to computing oblivious routings in partial tree embeddings. For this we use a lemma from [18] stating that we can find a distribution of probabilistic tree embeddings with the desired properties and show that these properties suffice.

Next we show how to reduce constructing an oblivious routing for a partial tree embedding to constructing an oblivious routing for what Madry [18] calls an "almost j-tree."

Definition 6.3. (Almost j-tree) *We call a graph $G = (V, E)$ an* almost j-tree *if there is a spanning tree $T \subseteq E$ such that the endpoints of $E \setminus T$ include at most j vertices.*

Lemma 6.1. *For undirected capacitated $G = (V, E, \vec{\mu})$ and partial tree embedding $H(G, T, F)$ in $\widetilde{O}(|E|)$ time we can construct almost $2|F|$-tree G' with at most $|E|$ edges and an embedding $\mathbf{M}'$ from G' to H, so H is embeds into G' with congestion 2, $\mathrm{cong}(\mathbf{M}') = 2$, and $\mathcal{T}(\mathbf{M}') = \widetilde{O}(|E|)$.*

Finally we use "greedy elimination" [24] [10] [12], i.e. removing all degree 1 and degree 2 vertices in $O(m)$ time, to reduce oblivious routing in almost-j-trees to oblivious routing in graphs with $O(j)$ vertices while only losing $O(1)$ in the competitive ratio. In the full version [8] we show that greedy elimination has the desired algorithmic properties and by applying the following lemma we complete the proof of Theorem 4.4. Further details can be found in [8] and [18].

7 Generalizations

7.1 Gradient Descent Method for Non-Linear Projection Problem

Here we strengthen and generalize the **MaxFlow** algorithm. We believe this algorithm may be of independent interest as it includes maximum concurrent flow problem, the compressive sensing problem, etc.

Given a norm $\|\cdot\|$, we wish to solve the what we call the *non-linear projection* problem $\min_{\vec{x}\in L}\|\vec{x}-\vec{y}\|$ where $\vec{y}$ is an given point and L is a linear subspace. We assume the following:

Assumption 7.1.

1. *There are a family of convex differentiable functions f_t such that for all $\vec{x}\in L$, we have $\|\vec{x}\| \le f_t(\vec{x}) \le \|\vec{x}\| + Kt$, and the Lipschitz constant of ∇f_t is $\frac{1}{t}$.*

2. *There is a projection matrix $\mathbf{P}$ onto the subspace L.*

The projection matrix $\mathbf{P}$ can be viewed as an approximation algorithm of this projection problem with approximate ratio $\|\mathbf{P}\|+1$. Hence, the following theorem says that given a problem instance we can iteratively improve the approximation quality given by $\mathbf{P}$.

Theorem 7.1. *Assume the conditions of Assumption 7.1 are satisfied. Let $\mathcal{T}$ be the time needed to compute $\mathbf{P}\vec{x}$ and $\mathbf{P}^T\vec{x}$ and $x^{\#}$. Then, there is an algorithm, **NonLinearProjection**, outputs a vector $\vec{x}'$ with $\|\vec{x}'-\vec{y}\| \le (1+\varepsilon)\min_{\vec{x}\in L}\|\vec{x}-\vec{y}\|$ in time $O\left(\|\mathbf{P}\|^2 K\left(\mathcal{T}+m\right)\left(\varepsilon^{-2}+\log\|\mathbf{P}\|\right)\right)$.*

Proof. Similar to Theorem 3.2. To get the running time $\propto \|\mathbf{P}\|^2$ instead of $\propto \|\mathbf{P}\|^4$, we first minimize f_t for small t to get a rough solution and use it to minimize f_t for larger t and repeat.

7.2 Maximum Concurrent Flow

For an arbitrary set of demand vectors $\vec{\chi}_i \in \mathbb{R}^V$ we wish to solve the following *maximum concurrent flow* problem

$$\max_{\alpha\in R,\vec{f}\in\mathbb{R}^{E\times k}} \alpha \text{ s.t } \mathbf{B}^T\vec{f}_i = \alpha\vec{\chi}_i \text{ and } \|\mathbf{U}^{-1}\sum_{i=1}^{k}|\vec{f}_i|\|_\infty \le 1$$

Similar to *maximum flow*, we can solve the *maximum concurrent flow* via the problem

$$\min_{\forall i\in[k]\ :\ \mathbf{B}^T\mathbf{U}\vec{x}_i=\vec{0}} \|\vec{x}-\vec{y}\|_{1;\infty}$$

where

$$\|\vec{x}\|_{1;\infty} \stackrel{\text{def}}{=} \max_{e\in E}\sum_{i=1}^{k}|x_i(e)|.$$

and $\vec{y}$ is some flow that meets the demands, i.e. $\forall i$, $\mathbf{B}^T\mathbf{U}\vec{y}_i = \vec{\chi}_i$.

To solve this problem using Theorem 7.1, we extend a circulation projection matrix $\mathbf{P}$ to a projection matrix for this problem using the formula $(\mathbf{Q}\vec{x})_i \stackrel{\text{def}}{=} \mathbf{P}\vec{x}_i$ and we use $\text{smax}_t(\sum_{i=1}^{k}(x_i(e))^2+t^2)^{1/2}$ as regularized $\|\cdot\|_{1;\infty}$. In the full version [8] prove the following.

Theorem 7.2. *For undirected graph $G=(V,E,\vec{\mu})$ with capacity ratio $U=\mathsf{poly}(|V|)$ we can compute a $(1-\varepsilon)$ approximate Maximum Concurrent Flow in $k^2|E|2^{O\left(\sqrt{\log|V|\log\log|V|}\right)}\varepsilon^{-2}$ time.*

8 Acknowledgements

We thank Jonah Sherman for agreeing to coordinate arXiv postings, and we thank Satish Rao, Daniel Spielman, Shang-Hua Teng for many helpful conversations.

References

[1] Y. Aumann and Y. Rabani. An o(log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[2] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *STOC'96: Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 47–55, New York, NY, USA, 1996. ACM.

[3] Dimitri P Bertsekas. Nonlinear programming. 1999.

[4] 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 *STOC '11*, pages 273–282, 2011.

[5] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[6] Jonathan A. Kelner and Petar Maymounkov. Electric routing and concurrent flow cutting. *CoRR*, abs/0909.2859, 2009.

[7] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1–18, New York, NY, USA, 2012. ACM.

[8] Jonathan A. Kelner, Lorenzo Orecchia, Yin Tat Lee, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. *CoRR*, abs/1304.2338, 2013.

[9] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. *CoRR*, abs/1301.6628, 2013.

[10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 235–244, Washington, DC, USA, 2010. IEEE Computer Society.

[11] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, 2010.

[12] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society.

[13] Gregory Lawler and Hariharan Narayanan. Mixing times and lp bounds for oblivious routing. In *WORKSHOP ON ANALYTIC ALGORITHMICS AND COMBINATORICS, (ANALCO 09) 4*, 2009.

[14] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A New Approach to Computing Maximum Flows using Electrical Flows. *Proceedings of the 45th symposium on Theory of Computing - STOC '13*, 2013.

[15] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Proceedings of the 2013 IEEE 54st Annual Symposium on Foundations of Computer Science*, FOCS '13. IEEE Computer Society, 2013.

[16] F. Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 47–56, New York, NY, USA, 2010. ACM.

[17] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, 2010.

[18] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. *CoRR*, abs/1008.1975, 2010.

[19] Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

[20] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *Core discussion papers*, 2:2010, 2010.

[21] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 255–264, New York, NY, USA, 2008. ACM.

[22] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science*, 2013.

[23] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, New York, NY, USA, 2004. ACM.

[24] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.

[25] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008.