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

Jonathan A. Kelner
kelner@mit.edu
MIT

Yin Tat Lee
yintat@mit.edu
MIT

Lorenzo Orecchia
orecchia@mit.edu
MIT

Aaron Sidford
sidford@mit.edu
MIT

Abstract

In this paper we present an almost linear time algorithm for solving approximate maximum flow in undirected graphs. In particular, given a graph with m edges we show how to produce a $1-\varepsilon$ approximate maximum flow in time $O(m^{1+o(1)} \cdot \varepsilon^{-2})$. Furthermore, we present this algorithm as part of a general framework that also allows us to achieve a running time of $O(m^{1+o(1)}\varepsilon^{-2}k^2)$ for the maximum concurrent k-commodity flow problem, the first such algorithm with an almost linear dependence on m. We also note that independently Jonah Sherman has produced an almost linear time algorithm for maximum flow and we thank him for coordinating submissions.

1 Introduction

Given a graph $G = (V, E)$ in which each edge $e \in E$ is assigned a nonnegative capacity u_e, the **maximum s-t flow problem** asks us to find a flow f that routes as much flow as possible from a source vertex s to a sink vertex t while sending at most u_e units of flow over each edge e. Its generalization, the **maximum concurrent multicommodity flow problem**, supplies k source-sink pairs (s_i, t_i) and asks for the maximum α such that we may simultaneously route α units of flow between each source-sink pair, That is, it asks us to find find find flows $f_1, \ldots, f_k$ (which we think of as corresponding to k different commodities) such that f_i sends α units of flow from s_i to t_i, and $\sum_i |f_i(e)| \leq u_e$ for all $e \in E$.

These problems lie at the core of of graph algorithms and combinatorial optimization and have been extensively studied for over the past 60 years [26, 1]. They have found a wide range of theoretical and practical applications [2], and they are widely used as key subroutines in other algorithms (see [3, 27]).

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 $\tilde{O}(m^{1+o(1)}\varepsilon^{-2}k^2)$, improving on the previous best bound of $\tilde{O}(mn^{1/3}\text{poly}(1/\varepsilon))$[7]. Applying the same framework in the multicommodity setting solves a maximum concurrent multicommodity flow problem with k commodities in $\tilde{O}(m^{1+o(1)}\varepsilon^{-2}k^2)$ time, improving on the existing bound of $\tilde{O}(m^{4/3}\text{poly}(\varepsilon))$[10].

We believe that both our general framework and several of the pieces necessary for its present instantiation are of independent interest, and we hope that they 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 *flow sparsifiers*; and
- the construction of a new oblivious routing scheme that can be implemented extremely efficiently.

We have aimed to make our algorithm fairly modular, and we have occasionally worked in slightly more generality than is strictly necessary for the problem at hand. This has slightly increased the length of the

exposition, but we believe that it clarifies the high-level structure of the argument, and it will hopefully facilitate the application of these tools in other settings.

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, such as augmenting paths, blocking flows, preflows, and the push-relabel method. These culminated in the work of Goldberg and Rao [8], which computes exact maximum flows in $O(\min(n^{2/3}, m^{1/2}) \log(n^2/m) \log U)$ on graphs with edge weights in $\{0, \ldots, U\}$. We refer the reader to [8] for a survey of these results.

More recently, a collection of new techniques based on randomization, spectral graph theory and numerical linear algebra, graph decompositions and embeddings, and iterative methods for convex optimization have emerged. These have allowed researchers to provide better provable algorithms for a wide range of flow and cut problems, particularly when one aims to obtain approximately optimal solutions on undirected graphs.

Our algorithm draws extensively on the intellectual heritage established by these works. In this section, we will briefly review some of the previous advances that inform our algorithm. We do not give a comprehensive review of the literature, but instead aim to provide a high-level view of the main that motivated the present work, along with the limitations of these tools that had to be overcome. For simplicity of exposition, we will primarily focus throughout the remainder of the introduction on the maximum s-t flow problem.

Sparsification In [5], Benczur and Karger showed how to efficiently approximate any graph G with a sparse graph G' on the same vertex set. To do this, they compute a carefully chosen probability p_e for each $e \in E$, sample each edge e with probability p_e, and include e in G' with its weight increased by a factor of $1/p_e$ if it is sampled. Using this, they obtain, in nearly linear time time, a graph G' with $O(n \log n/\varepsilon^2)$ edges such that the total weight of the edges crossing any cut in G' is within a multiplicative factor of $1 \pm \varepsilon$ of the weight crossing the corresponding cut in G. In particular, the Max-Flow Min-Cut Theorem implies that the value of the maximum flow on G' is within a factor of $1 \pm \varepsilon$ of that of G.

This is an extremely effective tool for approximately solving cut problems on a dense graph G, since one can simply solve the corresponding problem on the sparsified graph G'. However, while this means that one can approximately compute the *value* of the maximum s-t flow on G by solving the problem on G', it is not known how to use the maximum s-t flow on G' to obtain an actual approximately maximum flow on G. Intuitively, this is because the weights of edges included in G' are larger than they were in G, and the sampling argument does not provide any guidance about how to route flows over these edges in the original graph G.

Iterative algorithms based on linear systems and electrical flows In 2010, Christiano *et al.*[7] described a new linear algebraic approach to the problem that found ε-approximately maximum s-t flows in time $\tilde{O}(mn^{1/3}\text{poly}(1/\varepsilon))$. They treated the edges of G as electrical resistors and then computed the *electrical flow* that would result from sending electrical current from s to t in the corresponding circuit. They showed that these flows can be computed in nearly-linear time using fast Laplacian linear system solvers [13, 14, 11], which we further discuss below. The electrical flow obeys the flow conservation constraints, but it could violate the capacity constraints. They then adjusted the resistances of edges to penalize the edges that were flowing too much current and repeated the process. Kelner, Miller, and Peng [10] later showed how to use more general objects that they called *quadratically coupled flows* to use a similar approach to solve the maximum concurrent multicommodity flow problem in time $\tilde{O}(m^{4/3}\text{poly}(k, 1/\varepsilon))$.

Following this, Lee, Rao, and Srivastava [16] proposed another iterative algorithm that uses electrical flows, but in a way that was substantially different than in [7]. Instead of adjusting the resistances of the edges in each iteration to correct overflowing edges, they keep the resistances the same but compute a new electrical flow to reroute the excess current. They explain how to interpret this as gradient descent in a certain space, from which a standard analysis would give an algorithm that runs in time $\tilde{O}(m^{3/2}\text{poly}(1/\varepsilon))$. By replacing the standard gradient descent step with Nesterov's accelerated gradient descent method [22]

and using a regularizer to make the penalty function smoother, they obtain an algorithm that runs in time $\tilde{O}(mn^{1/3}\text{poly}(1/\varepsilon))$ in unweighted graphs.

In all of these algorithms, the superlinear running times arise from an intrinsic $\Theta(\sqrt{n})$ factor introduced by using electrical flows, which minimize an ℓ_2 objective function, to approximate the maximum congestion, which is an ℓ_∞ quantity.

Fast solvers for Laplacian linear systems In their breakthrough paper [30], Spielman and Teng showed how to solve Laplacian systems in nearly-linear time. (This was later sped up and simplified by Koutis, Miller, and Peng [13, 14] and Kelner, Orecchia, Sidford, and Zhu [11]) Their algorithm worked by showing how to approximate the Laplacian L_G of a graph G with the Laplacian L_H of a much simpler graph H such that one could use the ability to solve linear systems in L_H to accelerate the solution of a linear system in L_G. They then applied this recursively to solve the linear systems in L_H. In addition to providing the electrical flow primitive used by the algorithms described above, the structure of their recursive sequence of graph simplifications provides the motivating framework for much of the technical content of our oblivious routing construction.

Oblivious routing In an *oblivious routing scheme*, one specifies a linear operator taking any demand vector to a flow routing these demands over the edges of G. Given a collection of demand vectors, one can produce a multicommodity flow meeting these demands by routing each demand vector using this 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 [25], Räcke showed how to construct an oblivious routing scheme with a competitive ratio of $O(\log n)$. is construction worked by providing a probability distribution over trees T_i such that G embeds into each T_i with congestion at most 1, and such that the corresponding convex combination of trees embeds into G with congestion $O(\log n)$. In a sense, one can view this as showing how to approximate G by a probability distribution over trees. Using this, he was able to show how to obtain polylogarithmic approximations for a variety of cut and flow problems, given only the ability to solve these problems on trees.

We note that an oblivious routing scheme clearly yields a logarithmic approximation to the maximum flow and maximum concurrent multicommodity flow problems. However, Räcke's construction took time substantially superlinear time, making it too slow to be useful for computing approximately maximum flows. Furthermore, it only gives a logarithmic approximation, and it not clear how to use this a small number of times to reduce the error to a multiplicative ε.

In a later paper [18], Madry applied a recursive technique similar to the one employed by Spielman and Teng in their Laplacian solver to accelerate many of the applications of Räcke's construction at the cost of a worse approximation ratio. Using this, he obtained almost-linear-time polylogarithmic approximation algorithms for a wide variety of cut problems.

Unfortunately, his algorithm made extensive use of sparsification, which, for the previously mentioned reasons, made it unable to solve the corresponding flow problems. This meant that, while it could use flow-cut duality to find a polylogarithmic approximation of the value of a maximum flow, it could not construct a corresponding flow or repeatedly apply such a procedure a small number of times to decrease the error to a multiplicative ε.

1.2 Our Approach

In this section, we will give a high-level description of how we we overcome the obstacles described in the previous section. To avoid unnecessary notation, we will suppose for now that all of the edges have capacity 1.

The problem is thus to send as many units of flow as possible from s to t without sending more than one unit over any edge. It will be more convenient for us to work with an equivalent congestion minimization problem, where we try to find the unit s-t flow f (i.e., a flow sending one unit from s to t) that minimizes

$\|f\|_\infty = \max_e |f_e|$. If we begin with some initial unit s-t flow $\vec{f}_0$, the goal will be thus be to find the circulation $\vec{c}$ to $\vec{f}_0$ that minimizes $\|\vec{f}_0 + \vec{c}\|_\infty$.

We will give an iterative algorithm to approximately find such a $\vec{c}$. There will be $2^{O(\sqrt{\log n \log\log n})}/\varepsilon^2$ iterations, each of which will add a circulation to the present flow and run in $m \cdot 2^{O(\sqrt{\log n \log\log n})}$ time. Constructing this scheme will consist of two main parts: an iterative scheme that reduces the problem to the construction of a projection matrix with certain properties; and the construction of such an operator.

The iterative scheme: Non-Euclidean gradient descent

The simplest way to improve the flow would be to just perform gradient descent on the maximum congestion of an edge. There are two problems with this:

The first problem is that gradient descent depends on having a smoothly varying gradient, but the infinity norm 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 to 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 [16].

The more serious problem is the one discussed in the previous section: 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 $O(\sqrt{n})$ in the number of iterations, unless one can somehow exploit additional structure in the problem.

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 are able to 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 of iterations required by the gradient descent method depend 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, write flows meeting our demands as $\vec{f}_0 + \vec{c}$, as described above. We then 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 [11], this comes down to finding a good linear representation of the space of circulations, which we show here amounts 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. A simple calculation shows that any linear oblivious routing scheme A with a good competitive ratio gives rise to a projection matrix with the desired properties, and thus 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 flow routing used in previous algorithms with that 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. We do this with a recursive construction that reduces oblivious routing in a graph to oblivious routing in various successively simpler graphs.

[1] This idea and analysis seems to be implicit in other work, e.g., [24] 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 fairly general terms before specializing to what we need for flow problems.

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 [5] and Madry [18]) 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 such reductions, each of which we show how to implement without incurring a large increase in the competitive ratio:

Vertex reduction These show how to efficiently reduce oblivious routing on a graph to routing on t graphs with roughly $\tilde{O}|E|/t$ vertices.

To do this, we show how to efficiently embed a graph $G = (V, E)$ 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 [18]. 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 sparsifiers These allow us to efficiently reduce oblivious routing on an arbitrary graph to oblivious routing on a graph with $\tilde{O}(n)$ edges.

To construct flow sparsifiers, we use local partitioning to decompose the graph into well-connected clusters that contain many of the original edges. (These clusters are not quite expanders, but they are contained in graphs with good expansion in a manner that is sufficient for our purposes.) 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 graphs, and we can recurse on the result. While the implementation of these steps is somewhat different, the outline of this construction closely parallel's Spielman and Teng's approach to the construction of spectral sparsifiers.

Combining these recursively yields an efficient oblivious routing scheme, and thus an algorithm for the maximum flow problem.

We then show that the same framework can be applied to the maximum concurrent multicommodity flow problem. While the norm and regularization change, the structure of the argument and the construction of the oblivious routing scheme go through without requiring substantial modification.

2 Preliminaries

General notation: We typically use $\vec{x}$ to denote a vector and $\mathbf{A}$ to denote a matrix. For a vector $\vec{x} \in \mathbb{R}^n$, we let $|\vec{x}| \in \mathbb{R}^n$ denote the vector such that $\forall i$ we have $|\vec{x}|_i \stackrel{\text{def}}{=} |\vec{x}_i|$. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, we let $|\mathbf{A}|$ denote the matrix such that $\forall i, j$ we have $|\mathbf{A}|_{ij} \stackrel{\text{def}}{=} |\mathbf{A}_{ij}|$. We let $\mathbb{1}$ denote the all ones vector, we let $\mathbb{1}_i$ denote the vector that is one at position i and 0 elsewhere, we let $\mathbf{I}$ be the identity matrix and we overlaod notation let $\mathbf{I}_{a \to b} \in \mathbb{R}^{b \times a}$ denote the matrix such that for all $i \leq \min\{a, b\}$ we have $\mathbf{I}_{ii} = 1$ and $\mathbf{I} = 0$ elsewhere.

Graph Specification: Let $G = (V, E, \vec{\mu})$ be an undirected capacitated graph with $n = |V|$ vertices and $m = |E|$ edges where $\vec{\mu}_e$ is the capacity of edge e. Let $w_e \geq 0$ denote the weight of an edge and let $r_e \stackrel{\text{def}}{=} 1/w_e$ denote the resistance of an edge. For now we make no connection between μ_e and r_e; we fix their relationship later. Also, note that while all graphs in this paper will be undirected, we will always assume an arbitrary orientation assigned to the edges.

Fundamental Matrices: 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. Let $\mathbf{B} \in \mathbb{R}^{E \times V}$ denote the graphs incidence matrix where for all $e = (a, b) \in E$ we have $\mathbf{B}^T \mathbb{1}_e = \mathbb{1}_a - \mathbb{1}_b$. Let $\mathcal{L} \in \mathbb{R}^{V \times V}$ denote the combinatorial graph Laplacian and recall that $\mathcal{L} \stackrel{\text{def}}{=} \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B}$.

Sizes: For all $a \in V$ we let d_a denote the *(weighted) degree* of vertex a

$$\forall a \in V \quad : \quad d_a \stackrel{\text{def}}{=} \sum_{\{a,b\}} w_{a,b}$$

and let $\mathbf{D} \in \mathbb{R}^{V \times V}$ be the diagonal matrix where $\mathbf{D}_{a,a} = d_a$. Furthermore, for any vertex subset $S \subseteq V$ we define its *volume* by

$$\mathrm{vol}(S) \stackrel{\text{def}}{=} \sum_{a \in V} d_a$$

Furthermore, we let $\deg(a)$ denote the *(combinatorial) degree* of a, i.e. $\deg(a) = |\{e \in E \mid e = \{a, b\} \text{ for some } b \in V\}|$.

Cuts: For any set of vertex subset $S \subseteq V$ we denote the cut induced by S by the edge subset

$$\partial(S) \stackrel{\text{def}}{=} \{e \in E \mid e \notin S \text{ and } e \notin E \setminus S\}$$

and we define the cost of any cut $F \subseteq E$ by $w(F) \stackrel{\text{def}}{=} \sum_{e \in F} w_e$. Using this we define the *conductance* of a set $S \subseteq V$ by

$$\Phi(S) \stackrel{\text{def}}{=} \frac{w(\partial(S))}{\min\{\mathrm{vol}(S), \mathrm{vol}(V - S)\}}$$

and we denote the conductance of a graph by

$$\Phi(G) \stackrel{\text{def}}{=} \min_{S \subseteq \ : S \notin \{\emptyset, V\}} \phi(S)$$

Subgraphs: For a graph $G = (V, E)$ and a vertex subset $S \subseteq V$ let $G(V)$ denote the subgraph of G consisting of vertex set S and all the edges of E with both endpoints in S, i.e. $\{(a, b) \in E \mid a, b \in S\}$. When we we consider a term such as vol or Φ and we wish to make the graph such that this is respect to clear we use subscripts, so for example $\mathrm{vol}_{G(S)}(A)$ denotes the volume of vertex set A in the subgraph of G induced by S.

Congestion: For any vector $\vec{f} \in \mathbb{R}^E$ we let the *congestion*[2] *of* $\vec{f}$ be given by $\mathrm{cong}(\vec{f}) \stackrel{\text{def}}{=} \left\|\mathbf{U}^{-1}\vec{f}\right\|_\infty$. For any collection of flows $\{\vec{f}_i\} = \{\vec{f}_1, \ldots, \vec{f}_k\}$ we overload notation and let the *total congestion* of these flows be given by

$$\mathrm{cong}(\{\vec{f}_i\}) \stackrel{\text{def}}{=} \left\|\mathbf{U}^{-1} \sum_i |\vec{f}_i|\right\|_\infty$$

Demands and Multicommodity Flow Now we call a vector $\vec{\chi} \in \mathbb{R}^V$ a *demand vector* if $\sum_{a \in V} \vec{\chi}(a) = 0$ and 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 vectors $\{\vec{f}_i\}$ 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.

Operator Norm: 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

$$\forall \mathbf{A} \in \mathbb{R}^{n \times m} \quad : \quad \|\mathbf{A}\| \stackrel{\text{def}}{=} \max_{\vec{x} \in \mathbb{R}^n} \frac{\|\mathbf{A}\vec{x}\|}{\|\vec{x}\|}$$

Running Time: For any matrix $\mathbf{A}$ we let $\mathcal{T}(\mathbf{A})$ denote the time needed to apply both of $\mathbf{A}$ and $\mathbf{A}^\mathbf{T}$.

[2]Note that here and in the rest of the paper we will focus our analysis with congestion with respect to the norm $\|\cdot\|_\infty$ and we will look at oblivious routing strategies that are competitive with respect to this norm. However, many of the results present are easily generalizable to other norms but outside the scope of this paper.

3 Solving Max-Flow Using a Circulation Projection

3.1 Gradient Descent

In this section, we discuss the gradient descent method for general norms. Let $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ be an arbitrary norm on $\mathbb{R}^n$ and recall that the gradient of f at $\vec{x}$ is defined to be the vector $\nabla f(\vec{x}) \in \mathbb{R}^n$ such that

$$f(\vec{y}) = f(\vec{x}) + \langle \nabla f(\vec{x}), \vec{y} - \vec{x} \rangle + o(\|\vec{y} - \vec{x}\|). \tag{1}$$

The gradient descent method is a greedy minimization method that updates the current vector, $\vec{x}$, using the direction which minimizes $\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \rangle$. To analyze this method's performance, we need a tool 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 L^2 norm, this can be done by Cauchy Schwarz inequality and in general, we can define a new norm for $\nabla f(\vec{x})$ to make this happens. We call this the *dual norm* $\|\cdot\|^*$ defined as follows

$$\|\vec{x}\|^* \stackrel{\text{def}}{=} \max_{\vec{y} \in \mathbb{R}^n \,:\, \|\vec{y}\| \le 1} \langle \vec{y}, \vec{x} \rangle.$$

Fact 53 shows that this definition indeed yields that $\langle \vec{y}, \vec{x} \rangle \le \|\vec{y}\|^* \|\vec{x}\|$. Next, we define the fastest increasing direction $x^\#$, which is an arbitrary point satisfying the following

$$\vec{x}^\# \stackrel{\text{def}}{=} \arg\max_{\vec{s} \in \mathbb{R}} \langle \vec{x}, \vec{s} \rangle - \frac{1}{2}\|\vec{s}\|^2.$$

In the appendix, we provide some facts about $\|\cdot\|^*$ and $\vec{x}^\#$ that we will use in this section. Using the notations defined, 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 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 (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

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \;\; : \;\; \|\bigtriangledown f(\vec{x}) - \bigtriangledown f(\vec{y})\|^* \le L \cdot \|\vec{x} - \vec{y}\|.$$

In the appendix we provide an equivalent definition and a way to compute L, which is useful later.

Let $X^* \subseteq \mathbb{R}^n$ denote the set of optimal solutions to the unconstrained minimization problem $\min_{x \in \mathbb{R}^n} f$ and let f^* denote the optimal value of this minimization problem, i.e.

$$\forall \vec{x} \in X^* \;:\; f(\vec{x}) = f^* = \min_{\vec{y} \in \mathbb{R}^n} f(y) \quad \text{and} \quad \forall \vec{x} \notin X^* \;:\; f(\vec{x}) > f^*$$

We assume that X^* is non-empty. Now, we are ready to estimate the convergence rate of the gradient descent method.

Theorem 1 (Gradient Descent)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex continuously differentiable function and let L be the Lipschitz constant of ∇f. 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 $k \ge 0$, we have

$$f(\vec{x}_k) - f^* \le \frac{2 \cdot L \cdot R^2}{k+4} \quad \textit{where} \quad R \stackrel{\text{def}}{=} \max_{\vec{x} \in \mathbb{R}^n : f(\vec{x}) \le f(\vec{x}_0)} \min_{\vec{x}^* \in X^*} \|\vec{x} - \vec{x}^*\|.$$

Proof. [3] By the Lipschitz continuity of the gradient of f and Lemma 54 we have

$$f(\vec{x}_{k+1}) \leq f(\vec{x}_k) - \frac{1}{2L}\left(\left\| \bigtriangledown f(\vec{x}_k)\right\|^*\right)^2.$$

Furthermore, by the convexity of f, we know that

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \quad : \quad f(\vec{y}) \geq f(\vec{x}) + \left\langle \bigtriangledown f(\vec{x}), \vec{y} - \vec{x}\right\rangle.$$

Using this and the fact that $f(\vec{x}_k)$ decreases monotonically with k, we get

$$f(\vec{x}_k) - f^* \leq \min_{\vec{x}^* \in X^*} \left\langle \bigtriangledown f(\vec{x}_k), \vec{x}_k - \vec{x}^*\right\rangle \leq \min_{\vec{x}^* \in X^*} \left\| \bigtriangledown f(\vec{x}_k)\right\|^* \left\|\vec{x}_k - \vec{x}^*\right\| \leq R\left\| \bigtriangledown f(\vec{x}_k)\right\|^*.$$

Therefore, letting $\phi_k \stackrel{\text{def}}{=} f(\vec{x}_k) - f^*$, we have

$$\phi_k - \phi_{k+1} \geq \frac{1}{2L}(\left\| \bigtriangledown f(\vec{x}_k)\right\|^*)^2 \geq \frac{\phi_k^2}{2 \cdot L \cdot R^2}.$$

Furthermore, since $\phi_k \geq \phi_{k+1}$, we have

$$\frac{1}{\phi_{k+1}} - \frac{1}{\phi_k} = \frac{\phi_k - \phi_{k+1}}{\phi_k \phi_{k+1}} \geq \frac{\phi_k - \phi_{k+1}}{\phi_k^2} \geq \frac{1}{2 \cdot L \cdot R^2}.$$

So, by induction, we have that

$$\frac{1}{\phi_k} - \frac{1}{\phi_0} \geq \frac{k}{2 \cdot L \cdot R^2}.$$

Now, note that since $\bigtriangledown f(\vec{x}^*) = 0$, we have that

$$f(\vec{x}_0) \leq f(\vec{x}^*) + \left\langle \bigtriangledown f(\vec{x}^*), \vec{x}_0 - \vec{x}^*\right\rangle + \frac{L}{2}\left\|\vec{x}_0 - \vec{x}^*\right\|^2 \leq f(\vec{x}^*) + \frac{L}{2}R^2.$$

So, we have that $\phi_0 \leq \frac{L}{2}R^2$ and putting this all together yields that

$$\frac{1}{\phi_k} \geq \frac{1}{\phi_0} + \frac{k}{2 \cdot L \cdot R^2} \geq \frac{4}{2 \cdot L \cdot R^2} + \frac{k}{2 \cdot L \cdot R^2}.$$

□

3.2 Maximum Flow Formulation

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

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

Equivalently, we want to compute a *minimum congestion flow*

$$\min_{\vec{f} \in \mathbb{R}^E \,:\, \mathbf{B}^T \vec{f} = \vec{\chi}} \left\|\mathbf{U}^{-1}\vec{f}\right\|_\infty.$$

where we call $\left\|\mathbf{U}^{-1}\vec{f}\right\|_\infty$ *the congestion* of $\vec{f}$.

Letting $\vec{f}_0 \in \mathbb{R}^E$ be some initial *feasible flow*, i.e. $\mathbf{B}^T \vec{f}_0 \stackrel{\text{def}}{=} \vec{\chi}$, we write the problem equivalently as

$$\min_{\vec{c} \in \mathbb{R}^E \,:\, \mathbf{B}^T \vec{c} = 0} \left\|\mathbf{U}^{-1}(\vec{f}_0 + \vec{c})\right\|_\infty$$

where the output flow is $\vec{f} = \vec{f}_0 + \vec{c}$. Although the gradient descent method is applicable to constrained optimization problems and has a similar convergence guarantee, the sub-problem involved in each iteration is a constrained optimization problem, which is too complicated in this case. Since the domain is a linear subspace, the constraints can be avoided by projecting the variables onto this subspace.

Formally, we define a circulation projection matrix as follows.

[3] The structure of this specific proof was modeled after a proof in [24] for a slightly different problem.

Definition 2. *We call a matrix $\tilde{\mathbf{P}} \in \mathbb{R}^{E\times E}$ a circulation projection matrix if it is a projection matrix onto the circulation space, i.e. it satisfies the following*

- $\forall \vec{x} \in \mathbb{R}^E$ *we have* $\mathbf{B}^T\tilde{\mathbf{P}}\vec{x} = 0$.
- $\forall \vec{x} \in \mathbb{R}^E$ *with* $\mathbf{B}^T\vec{x} = 0$ *we have* $\mathbf{P}\vec{x} = \vec{x}$.

Then, the problem becomes

$$\min_{\vec{c}\in\mathbb{R}^E} \left\|\mathbf{U}^{-1}(\vec{f_0} + \tilde{\mathbf{P}}\vec{c})\right\|_\infty.$$

Applying gradient descent on this problem is similar to applying projected gradient method on the original problem. But, instead of using the orthogonal projection that is not suitable for $\|\cdot\|_\infty$, we try to pick a better projection matrix.

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 write the problem equivalently as

$$\min_{\vec{x}\in\mathbb{R}^E} \left\|\vec{\alpha_0} + \mathbf{P}\vec{x}\right\|_\infty$$

where the output maximum flow is

$$\vec{f}(\vec{x}) = \mathbf{U}(\vec{\alpha_0} + \mathbf{P}\vec{x})/\left\|\mathbf{U}(\vec{\alpha_0} + \mathbf{P}\vec{x})\right\|_\infty.$$

3.3 An Approximate Maximum Flow Algorithm

Since the gradient descent method requires the objective function to be differentiable, we introduce a smooth version of $\|\cdot\|_\infty$ which we call smax_t. In next section, we prove that there is a convex differentiable function smax_t such that $\bigtriangledown\text{smax}_t$ is Lipshitz continuous with Lipshitz constant $\frac{1}{t}$ and such that

$$\forall \vec{x} \in \mathbb{R}^E \quad : \quad \left\|\vec{x}\right\|_\infty - t\ln(2m) \le \text{smax}_t(\vec{x}) \le \left\|\vec{x}\right\|_\infty \quad .$$

Now we consider the following regularized optimization problem

$$\min_{\vec{x}\in\mathbb{R}^E} g_t(\vec{x}) \quad \text{where} \quad g_t(\vec{x}) = \text{smax}_t(\vec{\alpha_0} + \mathbf{P}\vec{x}).$$

For the rest of this section, we consider solving this optimization problem using gradient descent under $\|\cdot\|_\infty$.

First, we bound the Lipschitz constant of the gradient of g_t.

Lemma 3. *The gradient of g_t is Lipschitz continuous with Lipschitz constant $L = \frac{\|\mathbf{P}\|_\infty^2}{t}$*

Proof. By Lemma 54 and the Lipschitz continuity of $\bigtriangledown\text{smax}_t$, we have

$$\text{smax}_t(\vec{y}) \le \text{smax}_t(\vec{x}) + \left\langle\nabla\text{smax}_t(\vec{x}), \vec{y} - \vec{x}\right\rangle + \frac{1}{2t}\left\|\vec{y} - \vec{x}\right\|_\infty.$$

Seting $\vec{x} \leftarrow \vec{\alpha_0} + \mathbf{P}\vec{x}$ and $\vec{y} \leftarrow \vec{\alpha_0} + \mathbf{P}\vec{y}$, we have

$$\begin{aligned}
g_t(\vec{y}) &\le g_t(\vec{y}) + \left\langle\nabla\text{smax}_t(\vec{\alpha_0} + \mathbf{P}\vec{x}), \mathbf{P}\vec{y} - \mathbf{P}\vec{x}\right\rangle + \frac{1}{2t}\left\|\mathbf{P}\vec{y} - \mathbf{P}\vec{x}\right\|_\infty^2 \\
&\le g_t(\vec{y}) + \left\langle\mathbf{P}^T\nabla\text{smax}_t(\vec{\alpha_0} + \mathbf{P}\vec{x}), \vec{y} - \vec{x}\right\rangle + \frac{1}{2t}\left\|\mathbf{P}\right\|_\infty^2\left\|\vec{y} - \vec{x}\right\|_\infty^2 \\
&= g_t(\vec{y}) + \left\langle\nabla g_t(\vec{x}), \vec{y} - \vec{x}\right\rangle + \frac{1}{2t}\left\|\mathbf{P}\right\|_\infty^2\left\|\vec{y} - \vec{x}\right\|_\infty^2.
\end{aligned}$$

Hence, the result follows from Lemma 54. □

Now, we apply gradient descent to find an approximate max flow as follows.

MaxFlow
Input: any initial feasible flow $\vec{f}_0$ and $\mathrm{OPT} = \min_{\vec{x}} \left\|\mathbf{U}^{-1}\vec{f}_0 + \mathbf{P}\vec{x}\right\|_\infty$.
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 $k = 300\left\|\mathbf{P}\right\|_\infty^4 \ln(2m)/\varepsilon^2$.
3. Let $g_t = \mathrm{smax}_t(\vec{\alpha_0} + \mathbf{P}\vec{x})$.
4. For $i = 1, \cdots, k$
5. $\quad \vec{x}_{i+1} = \vec{x}_i - \frac{t}{\left\|\mathbf{P}\right\|_\infty^2}(\bigtriangledown g_t(\vec{x}_i))^{\#}$. (See Lemma 48)
6. Output $\mathbf{U}\left(\vec{\alpha_0} + \mathbf{P}\vec{x}_k\right)/\left\|\vec{\alpha_0} + \mathbf{P}\vec{x}_k\right\|_\infty$.

Theorem 4. *Let $\widetilde{\mathbf{P}}$ be a cycle 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*

$$O\left(\frac{\left\|\mathbf{P}\right\|_\infty^4 \ln(m)\left(\mathcal{T}(\mathbf{P}) + m\right)}{\varepsilon^2}\right).$$

Proof. First, we bound $\left\|\vec{\alpha_0}\right\|_\infty$. Let $\vec{x}^*$ be a minimizer of $\min_{\vec{x}} \left\|\mathbf{U}^{-1}\vec{f}_0 + \mathbf{P}\vec{x}\right\|_\infty$ such that $\mathbf{P}\vec{x}^* = \vec{x}^*$. Then, we have

$$\begin{aligned}
\left\|\vec{\alpha_0}\right\|_\infty &= \left\|\mathbf{U}^{-1}\vec{f}_0 - \mathbf{P}\mathbf{U}^{-1}\vec{f}_0\right\|_\infty \\
&\le \left\|\mathbf{U}^{-1}\vec{f}_0 + \vec{x}^*\right\|_\infty + \left\|\vec{x}^* + \mathbf{P}\mathbf{U}^{-1}\vec{f}_0\right\|_\infty \\
&= \left\|\mathbf{U}^{-1}\vec{f}_0 + \vec{x}^*\right\|_\infty + \left\|\mathbf{P}\vec{x}^* + \mathbf{P}\mathbf{U}^{-1}\vec{f}_0\right\|_\infty \\
&\le \left(1 + \left\|\mathbf{P}\right\|_\infty\right)\left\|\mathbf{U}^{-1}\vec{f}_0 + \vec{x}^*\right\|_\infty \\
&= \left(1 + \left\|\mathbf{P}\right\|_\infty\right)\mathrm{OPT}.
\end{aligned}$$

Second, we bound R in Theorem 1. Note that

$$g_t(\vec{x}_0) = \mathrm{smax}_t(\vec{\alpha_0}) \le \left\|\vec{\alpha_0}\right\|_\infty \le \left(1 + \left\|\mathbf{P}\right\|_\infty\right)\mathrm{OPT}.$$

Hence, the condition $g_t(\vec{x}) \le g_t(\vec{x}_0)$ implies that

$$\left\|\vec{\alpha_0} + \mathbf{P}\vec{x}\right\|_\infty \le \left(1 + \left\|\mathbf{P}\right\|_\infty\right)\mathrm{OPT} + t\ln(2m).$$

For any $\vec{y} \in X^*$ let $\vec{c} = \vec{x} - \mathbf{P}\vec{x} + \vec{y}$ and note that $\mathbf{P}\vec{c} = \mathbf{P}\vec{y}$ and therefore $\vec{c} \in X^*$. Using these facts, we can bound R as follows

$$\begin{aligned}
R &= \max_{\vec{x}\in\mathbb{R}^E \,:\, g_t(\vec{x}) \le g_t(\vec{x}_0)} \left\{\min_{\vec{x}^* \in X^*} \left\|\vec{x} - \vec{x}^*\right\|_\infty\right\} \\
&\le \max_{\vec{x}\in\mathbb{R}^E \,:\, g_t(\vec{x}) \le g_t(\vec{x}_0)} \left\|\vec{x} - \vec{c}\right\|_\infty \\
&\le \max_{\vec{x}\in\mathbb{R}^E \,:\, g_t(\vec{x}) \le g_t(\vec{x}_0)} \left\|\mathbf{P}\vec{x} - \mathbf{P}\vec{y}\right\|_\infty \\
&\le \max_{\vec{x}\in\mathbb{R}^E \,:\, g_t(\vec{x}) \le g_t(\vec{x}_0)} \left\|\mathbf{P}\vec{x}\right\|_\infty + \left\|\mathbf{P}\vec{y}\right\|_\infty \\
&\le 2\left\|\vec{\alpha_0}\right\|_\infty + \left\|\vec{\alpha_0} + \mathbf{P}\vec{x}\right\|_\infty + \left\|\vec{\alpha_0} + \mathbf{P}\vec{y}\right\|_\infty \\
&\le 2\left\|\vec{\alpha_0}\right\|_\infty + 2\left\|\vec{\alpha_0} + \mathbf{P}\vec{x}\right\|_\infty \\
&\le 4\left(1 + \left\|\mathbf{P}\right\|_\infty\right)\mathrm{OPT} + 2t\ln(2m).
\end{aligned}$$

From Lemma 3, we know that the Lipschitz constant of $\bigtriangledown g_t$ is $\left\|\mathbf{P}\right\|_\infty^2/t$. Hence, Theorem 1 shows that

$$\begin{aligned}
g_t(\vec{x}_k) &\le \min_{\vec{x}} g_t(\vec{x}) + \frac{2\cdot L\cdot R^2}{k+4} \\
&\le \mathrm{OPT} + \frac{2\cdot L\cdot R^2}{k+4}.
\end{aligned}$$

So, we have

$$\begin{aligned}\left\|\vec{\alpha_0} + \mathbf{P}\vec{x}_k\right\|_\infty &\leq g_t(\vec{x}_k) + t\ln(2m) \\ &\leq \mathrm{OPT} + t\ln(2m) + \frac{2\left\|\mathbf{P}\right\|_\infty^2}{t(k+4)}\left(4\left(1+\left\|\mathbf{P}\right\|_\infty\right)\mathrm{OPT} + 2t\ln(2m)\right)^2.\end{aligned}$$

Using $t = \varepsilon\mathrm{OPT}/2\ln(2m)$ and $k = 300\left\|\mathbf{P}\right\|_\infty^4 \ln(2m)/\varepsilon^2$, we have

$$\left\|\vec{\alpha_0} + \mathbf{P}\vec{x}_k\right\|_\infty \leq (1+\varepsilon)\mathrm{OPT}.$$

Therefore, $\vec{\alpha_0} + \mathbf{P}\vec{x}_k$ is an $(1-\varepsilon)$ approximate maximum flow.

Now, we estimate the running time. In each step 5, we are required to compute $(\bigtriangledown g(\vec{x}_k))^\#$. The gradient

$$\nabla g(\vec{x}) = \mathbf{P}^T \nabla \mathrm{smax}_t(\vec{\alpha_0} + \mathbf{P}\vec{x})$$

can be computed in $O(\mathcal{T}(\mathbf{P}) + m)$ using the formula of the gradient of smax_t, applications of $\mathbf{P}$ and $\mathbf{P}^T$. Lemma 48 shows that the $\#$ operator can be computed in $O(m)$. □

Lemma 5. *In $\left\|\cdot\right\|_\infty$, the $\#$ operator is given by the explicit formula*

$$\left(\vec{x}^\#\right)_e = \mathrm{sign}(x_e)\left\|\vec{x}\right\|_1 \quad \textit{for } e \in E.$$

Proof. Recall that

$$\vec{x}^\# = \arg\max_{\vec{s}\in\mathbb{R}} \left\langle \vec{x}, \vec{s}\right\rangle - \frac{1}{2}\left\|\vec{s}\right\|_\infty^2.$$

It is easy to see that for all $e \in E$, $||\vec{x}^\#||_\infty = \left|\left(\vec{x}^\#\right)_e\right|$. In particular, we have

$$\left(\vec{x}^\#\right)_e = \mathrm{sign}(x_e)||\vec{x}^\#||_\infty.$$

The Fact 52 shows that $||\vec{x}^\#||_\infty = \left\|\vec{x}\right\|_1$ and the result follows. □

3.4 Properties of soft max

In this section, we define smax_t and discuss its properties. Formally, the regularized convex function can be found by smoothing technique using convex conjugate [21] [6, Sec 5.4]. For simplicity and completeness, we define it explicitly and prove its properties directly. Formally, we define

$$\forall \vec{x} \in \mathbb{R}^E, \forall t \in \mathbb{R}^+ \quad : \quad \mathrm{smax}_t(\vec{x}) \stackrel{\mathrm{def}}{=} t\ln\left(\frac{\sum_{e\in E}\exp\left(\frac{x_e}{t}\right) + \exp\left(-\frac{x_e}{t}\right)}{2m}\right).$$

For notational simplicity, for all $\vec{x}$ where this vector is clear from context, we define $\vec{c}$ and $\vec{s}$ as follows

$$\forall e \in E \quad : \quad \vec{c}_e \stackrel{\mathrm{def}}{=} \exp\left(\frac{x_e}{t}\right) + \exp\left(-\frac{x_e}{t}\right) \quad \text{and} \quad \vec{s}_e \stackrel{\mathrm{def}}{=} \exp\left(\frac{x_e}{t}\right) - \exp\left(-\frac{x_e}{t}\right),$$

where the letters are chosen due to the very close resemblance to hyperbolic sine and hyperbolic cosine.

Lemma 6.

$$\forall \vec{x} \in \mathbb{R}^n \quad : \quad \bigtriangledown \mathrm{smax}_t(\vec{x}) = \frac{1}{\mathbb{1}^T\vec{c}}\vec{s}$$

$$\forall \vec{x} \in \mathbb{R}^n \quad : \quad \bigtriangledown^2 \mathrm{smax}_t(\vec{x}) = \frac{1}{t\left(\mathbb{1}^T\vec{c}\right)}\left[\mathrm{diag}(\vec{c}) - \frac{\vec{s}\vec{s}^T}{\mathbb{1}^T\vec{c}}\right]$$

Proof. For all $i \in E$ and $\vec{x} \in \mathbb{R}^E$, we have

$$\begin{aligned}\frac{\partial}{\partial x_i}\text{smax}_t(\vec{x}) &= \frac{\partial}{\partial x_i}\left(t\ln\left(\frac{\sum_{e\in E}\exp\left(\frac{x_e}{t}\right)+\exp\left(-\frac{x_e}{t}\right)}{2m}\right)\right)\\ &= \frac{\exp\left(\frac{x_i}{t}\right)-\exp\left(-\frac{x_i}{t}\right)}{\sum_{e\in E}\exp\left(\frac{x_e}{t}\right)+\exp\left(-\frac{x_e}{t}\right)}.\end{aligned}$$

For all $i, j \in E$ and $\vec{x} \in \mathbb{R}^E$, we have

$$\begin{aligned}\frac{\partial^2}{\partial x_i \partial x_j}\text{smax}_t(\vec{x}) &= \frac{\partial^2}{\partial x_i \partial x_j}\left(t\ln\left(\frac{\sum_{e\in E}\exp\left(\frac{x_e}{t}\right)+\exp\left(-\frac{x_e}{t}\right)}{2m}\right)\right)\\ &= \frac{\partial}{\partial j}\left[\frac{\exp\left(\frac{x_i}{t}\right)-\exp\left(-\frac{x_i}{t}\right)}{\sum_{e\in E}\exp\left(\frac{x_e}{t}\right)+\exp\left(-\frac{x_e}{t}\right)}\right]\\ &= \frac{1}{t}\frac{\left(\mathbb{1}^T\vec{c}\right)\mathbb{1}_{i=j}\left(\vec{c}_i\right)-\vec{s}_i\vec{s}_j}{\left(\mathbb{1}^T\vec{c}\right)^2}.\end{aligned}$$

□

Lemma 7. *The function* smax_t *is a convex continuously differentiable function and it has Lipschitz continuous gradient with Lipschitz constant* $1/t$ *and*

$$\left\|\vec{x}\right\|_\infty - t\ln(2m) \le \text{smax}_t(\vec{x}) \le \left\|\vec{x}\right\|_\infty$$

for $\vec{x} \in \mathbb{R}^E$.

Proof. By the formulation of the Hessian, for all $\vec{x}, \vec{y} \in \mathbb{R}^E$, we have

$$\vec{y}^T\left(\bigtriangledown^2\text{smax}_t(\vec{x})\right)\vec{y} \le \frac{\sum_i c_i\vec{y}_i^2}{t(\mathbb{1}^T\vec{c})} \le \frac{\sum_i c_i(\max_j \vec{y}_j^2)}{t(\mathbb{1}^T\vec{c})} \le \frac{1}{t}\|y\|_\infty^2.$$

On the other side, for all $\vec{x}, \vec{y} \in \mathbb{R}^E$, we have by $s_i \le c_i$ and Cauchy Schwarz shows that

$$\vec{y}^T\vec{s}\vec{s}^T\vec{y} \le \vec{y}^T\vec{c}\vec{c}^T\vec{y} \le (\mathbb{1}^T\vec{c})(\vec{y}^T\,\text{diag}(\vec{c})\vec{y}).$$

and hence

$$0 \le \vec{y}^T\left(\bigtriangledown^2\text{smax}_t(\vec{x})\right)\vec{y}.$$

Thus, the first part follows from Lemma 55. For the later part, we have

$$\left\|\vec{x}\right\|_\infty \ge t\ln\left(\frac{\sum_{e\in E}\exp\left(\frac{x_e}{t}\right)+\exp\left(-\frac{x_e}{t}\right)}{2m}\right) \ge t\ln\left(\frac{\exp\left(\frac{\left\|\vec{x}\right\|_\infty}{t}\right)}{2m}\right) = \left\|\vec{x}\right\|_\infty - \ln(2m).$$

□

4 Oblivious Routing

In the previous sections we saw how a circulation projection matrix can be used to solve max flow. In the next few sections we show how to construct a circulation projection matrix in order to obtain an almost linear time algorithm for solving max flow.

Our proof focuses on the notion of (linear) oblivious routings. Rather than constructing the 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.

In the remainder of this section we formally define oblivious routings and prove the relationship between oblivious routing and circulation projection matrices (Section 4.1), provide a high level overview of our recursive approach and state the main theorems we will prove in later sections (Section 4.2), and we prove the main theorem about our construction of circulation projection in almost linear time with norm $2^{O(\sqrt{log(n)loglog(n)})}$ assuming the proofs in the later sections (Section 4.3).

4.1 From Oblivious Routing to Circulation Projection

Here we provide definitions and prove basic properties of *oblivious routings*, that is fixed mapping from demands to flows that meet the demands. While non-linear algorithms could be considered, we restrict our attention to linear oblivious routing strategies and for notational convenience use oblivious routing to refer to the linear subclass for the remainder of the paper.[4]

Definition 8 (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 $\vec{\chi} \in \mathbb{R}^V$, $\mathbf{B}^T\mathbf{A}\vec{\chi} = \vec{\chi}$. We call $\mathbf{A}\vec{\chi}$ the* routing *of $\vec{\chi}$ by $\mathbf{A}$.*

Oblivious routings get there name due to the fact that given an oblivious routing $\mathbf{A}$ and a set of demands $D = \{\vec{\chi}_1, \ldots, \vec{\chi}_k\}$ one can route all the demands by *routing* each demand individually by $\mathbf{A}$ and obtain a multicommodity flow that was oblivious to the relationship between the demands. We measure the *competitive ratio*[5] of such an oblivious routing strategy to be the worst relative congestion of such a routing to the minimal congestion routing of the demands.

Definition 9 (Competitive Ratio)**.** *The* competitive ratio of oblivious routing $\mathbf{A} \in \mathbb{R}^{E \times V}$*, denoted $\rho(\mathbf{A})$, is given by*

$$\rho(\mathbf{A}) \stackrel{\text{def}}{=} \max_{\{\vec{\chi}_i\} \,:\, \forall i \,:\, \vec{\chi}_i \perp \mathbb{1}} \frac{\text{cong}(\{\mathbf{A}\vec{\chi}_i\})}{\text{opt}(\{\vec{\chi}_i\})}$$

At times it will be more convenient to analyze an oblivious routing as a linear algebraic object rather a combinatorial algorithm; towards this end we note that the competitive ratio of a linear oblivious routing strategy can be gleaned from the operator norm of a related matrix ([15] and [9]). Below we state and prove a generalization of this result to weighted graphs that will be vital to relating $\mathbf{A}$ to $\widetilde{\mathbf{P}}$.

Lemma 10. *For any oblivious routing $\mathbf{A}$ we have $\rho(\mathbf{A}) = \left\|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\right\|_\infty$*

Proof. For a set of demands D let D_∞ be the set of demands that result by taking the routing of every demand in D by $\text{opt}(D)$ and splitting it up into demands on every edge corresponding to the flow sent by $\text{opt}(D)$. Now clearly $\text{opt}(D) = \text{opt}(D_\infty)$ since routing D can be used to route D_∞ and vice versa and clearly $\text{cong}(\mathbf{A}D) \leq \text{cong}(\mathbf{A}D_\infty)$ by the linearity of $\mathbf{A}$ (routing D_∞ simply doesn't reward $\mathbf{A}$ routing for cancellations). Therefore

$$\begin{aligned}\rho_p(\mathbf{A}) &= \max_D \frac{\text{cong}(\{\mathbf{A}D\})}{\text{opt}(D)} = \max_{D_\infty} \frac{\text{cong}(\mathbf{A}D_\infty)}{\text{opt}(D_\infty)} = \max_{\vec{x} \in \mathbb{R}^E} \frac{\left\| \sum_{e \in E} \vec{x}_e \left|\mathbf{U}^{-1}\mathbf{A}\vec{\chi}_e\right| \right\|_\infty}{\left\|\mathbf{U}^{-1}\vec{x}\right\|_\infty} \\ &= \max_{\vec{x} \in \mathbb{R}^E} \frac{\left\| \left|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\right| \vec{x}\right\|_\infty}{\left\|\mathbf{U}^{-1}\vec{x}\right\|_\infty} = \max_{\vec{x} \in \mathbb{R}^E} \frac{\left\| \left|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\right| \vec{x}\right\|_\infty}{\left\|\vec{x}\right\|_\infty}\end{aligned}$$

□

[4] Note that the oblivous routing strategies considered in [9] [15] [25] are all linear oblivious routing strategies.

[5] Again note that here and in the rest of the paper we focus our analysis on competitive ratio with respect to norm $\|\cdot\|_\infty$. However, many of the results present are easily generalizable to other norms but outside the scope of this paper.

To make this lemma easily applicable in a variety of settings we make use of the following easy to prove lemma.

Lemma 11 (Operator Norm Bounds)**.** *For all* $\mathbf{A} \in \mathbb{R}^{n \times m}$ *we have that*

$$\left\|\mathbf{A}\right\|_\infty = \left\| \left|\mathbf{A}\right| \right\|_\infty = \left\| \left|\mathbf{A}\right| \mathbb{1} \right\|_\infty = \max_{i \in n} \left\| |\mathbf{A}|^T \mathbb{1}_i \right\|_1$$

The previous two lemmas make the connection between oblivious routings and circulation projection matrices clear and below we prove this formally.

Lemma 12 (From Oblivious Routing to Circulation Projection)**.** *For 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* $\left\|\mathbf{U}\widetilde{\mathbf{P}}\mathbf{U}^{-1}\right\|_\infty \leq 1 + \rho(\mathbf{A})$ *.*

Proof. First we verify that $\text{im}(\widetilde{\mathbf{P}})$ is contained in cycle space

$$\forall \vec{x} \in \mathbb{R}^E \quad : \quad \mathbf{B}^T \widetilde{\mathbf{P}} \vec{x} = \mathbf{B}^T \vec{x} - \mathbf{A}\mathbf{B}^T \vec{x} = \vec{0}$$

Next we check that $\widetilde{\mathbf{P}}$ is the identity on cycle space

$$\forall \vec{x} \in \mathbb{R}^E \text{ s.t. } \mathbf{B}^T \vec{x} = \vec{0} \quad : \quad \widetilde{\mathbf{P}} \vec{x} = \vec{x} - \mathbf{A}\mathbf{B}^T \vec{x} = \vec{x}$$

Finally we bound infinite norm of the scaled projection matrix

$$\left\|\mathbf{U}\widetilde{\mathbf{P}}\mathbf{U}^{-1}\right\|_\infty = \left\|\mathbf{I} + \mathbf{U}\mathbf{A}\mathbf{B}^T\mathbf{U}^{-1}\right\|_\infty \leq 1 + \rho(\mathbf{A})$$

□

4.2 A Recursive Approach by Embeddings

We construct an oblivious routing for a graph recursively. 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 graphs on the same vertex sets but different edge sets.

Definition 13 (Embedding)**.** *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* $\mathbf{B}' \in \mathbb{R}^{E' \times V}$ *respectively. An* embedding *from* G *to* G' *is a matrix* $\mathbf{M} \in \mathbb{R}^{E' \times E}$ *such that* $\mathbf{B'}^T \mathbf{M} = \mathbf{B}^T$*.*

In other words an embedding is a map from flows in one graph to flows in another graph that maintain the demands met by the flow. We can think of an embedding as a way of routing any flow in one graph in another graph without increasing the congestion of that too much or we can think of an embedding as a way of routing one graph in another graph with low congestion. The views are equivalent in the sense that they give equivalent definitions of quality of the embedding.

Definition 14 (Embedding Congestion)**.** *Let* $\mathbf{M} \in \mathbb{R}^{E' \times E}$ *be an embedding from* $G = (V, E, \vec{\mu})$ *to* $G' = (V, E', \vec{\mu}')$ *and let* $\mathbf{U} \in \mathbb{R}^{E \times E}$ *and* $\mathbf{U}' \in \mathbb{R}^{E' \times E'}$ *denote the capacity matrices of* G *and* G' *respectively. The* congestion *of embedding* $\mathbf{M}$ *is given by*

$$\text{cong}(\mathbf{M}) \stackrel{\text{def}}{=} \max_{\vec{x} \in \mathbb{R}^E} \frac{\left\|\mathbf{U'}^{-1}\mathbf{M}\vec{x}\right\|_\infty}{\left\|\mathbf{U}^{-1}\vec{x}\right\|_\infty} = \left\|\mathbf{U'}^{-1}|\mathbf{M}|\mathbf{U}\mathbb{1}\right\|_\infty \ .$$

We say G embeds into G' *with congestion* α *if there exists an embedding* $\mathbf{M}$ *from* G *to* G' *such that* $\text{cong}(\mathbf{M}) \leq \alpha$

Embeddings potentially allow us to reduce computing an oblivious routing in a complicated graph to computing an oblivious routing in a simpler graph. More to the point, if we can embed a complicated graph in a simpler graph and we can efficiently embed the simple graph in the original graph, both with low congestion then we can just focus on constructing oblivious routings in the simpler graph. We prove this formally as follows.

Lemma 15 (Embedding Lemma)**.** *Let $G = (V, E, \vec{\mu})$ and $G' = (V, E', \vec{\mu}')$ denote two undirected capacitated graphs on the same vertex sets, let $\mathbf{M} \in \mathbb{R}^{E' \times E}$ denote an embedding from G into G', let $\mathbf{M}' \in \mathbb{R}^{E \times E'}$ denote an embeding from G' into G, and let $\mathbf{A}' \in \mathbb{R}^{E' \times V}$ denote an oblivious routing algorithm on G'. Then $\mathbf{A} \stackrel{\text{def}}{=} \mathbf{M}'\mathbf{A}'$ is an oblivious routing algorithm on G and*

$$\rho(\mathbf{A}) \leq \text{cong}(\mathbf{M}) \cdot \text{cong}(\mathbf{M}') \cdot \rho(\mathbf{A}')$$

Proof. For all $\vec{x} \in \mathbb{R}^V$ we have by definition of embeddings and oblivious routings that

$$\mathbf{B}^T\mathbf{A}\vec{x} = \mathbf{B}^T\mathbf{M}'\mathbf{A}'\vec{x} = \mathbf{B}^T\vec{x}$$

To bound $\rho(A)$ we let $\mathbf{U}$ denote the capacity matrix of G, we let $\mathbf{U}'$ denote the capacity matrix of G' and using Lemma 10 we get

$$\rho(\mathbf{A}) = \left\|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\right\|_\infty = \left\|\mathbf{U}^{-1}\mathbf{M}'\mathbf{A}'\mathbf{B}^T\mathbf{U}\right\|_\infty$$

Using that $\mathbf{M}$ is an embedding and therefore $\mathbf{B'}^T\mathbf{M} = \mathbf{B}^T$ we get

$$\rho(\mathbf{A}) = \left\|\mathbf{U}^{-1}\mathbf{M}'\mathbf{A}'\mathbf{B'}^T\mathbf{M}\mathbf{U}\right\|_\infty \leq \left\|\mathbf{U}^{-1}\mathbf{M}'\mathbf{U}'\right\|_\infty \cdot \left\|\mathbf{U'}^{-1}\mathbf{A}'\mathbf{B'}^T\mathbf{U}\right\|_\infty \cdot \left\|\mathbf{U'}^{-1}\mathbf{M}\mathbf{U}\right\|_\infty$$

By the definition of competitive ratio and congestion we get the result. □

Note how in this lemma we only use the embedding from G to G' to certify the quality of flows in G', we do not actually need to apply this embedding in the reduction.

Using this concept we construct oblivious routings via recursive application of two techniques. First, in Section 5 we show how to take an arbitrary graph $G = (V, E)$ and approximate it by a *sparse graph* $G' = (V, E')$ (i.e. one in which $|E'| = \widetilde{O}(|E|)$) 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. As a result of proving how to efficiently create such *flow sparsifiers* we prove the following theorem.

Theorem 16 (Edge Reduction)**.** *Let $G = (V, E, \vec{\mu})$ be an undirected capacitated graph with capacity ratio $U \leq \mathsf{poly}(|V|)$. 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*

$$\mathcal{T}(\mathbf{A}) = \widetilde{O}(|E| + \mathcal{T}(\mathbf{A}')) \quad \text{and} \quad \rho(\mathbf{A}) = \widetilde{O}(\rho(\mathbf{A}'))$$

Next, in Section 6 we show how to embed a graph into a collection of graphs consisting of trees plus extra edges. Then we will 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 17 (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 $\widetilde{O}(\frac{|E| \log(U)}{t})$ 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*

$$\mathcal{T}(\mathbf{A}) = \widetilde{O}(t \cdot |E| + \sum_{i=1}^{t} \mathcal{T}(\mathbf{A}_i)) \quad \text{and} \quad \rho(\mathbf{A}) = \widetilde{O}(\max_i \rho(\mathbf{A}_i))$$

In the next section we show that the careful application of these two ideas along with a powerful primitive for routing on constant sized graphs suffices to produce an oblivious routing with the desired properties.

4.3 Efficient Oblivious Routing Construction Proof

Before, we prove that the previous theorems suffice to provide an efficient low congestion oblivious routing we provide one further lemma that will serve as the base case of our recursion. In particular we show that electric routing can be used to obtain a routing algorithm with constant comptetive ratio for constant sized graphs.

Lemma 18 (Base Case)**.** *Let $G = (V, E, \vec{\mu})$ be an undirected capacitated graph and let us assign weights to edges so that $\mathbf{W} = \mathbf{U}^2$. For $\mathcal{L} \stackrel{\text{def}}{=} \mathbf{B}^T\mathbf{W}\mathbf{B}$ we have that $\mathbf{A} \stackrel{\text{def}}{=} \mathbf{W}\mathbf{B}\mathcal{L}^\dagger$ is an oblivious routing on G with $\rho(\mathbf{A}) \leq \sqrt{|E|}$ and $\mathcal{T}\left(\mathcal{L}^\dagger\right) = \widetilde{O}(|E|)$*

Proof. To see that $\mathbf{A}$ is an oblivious routing strategy we note that for any demands $\vec{\chi} \in \mathbb{R}^V$ we have $\mathbf{B}^T\mathbf{A} = \mathcal{L}\mathcal{L}^\dagger\mathbf{A} = \mathbf{A}$. To see bound $\rho(\mathbf{A})$ we note that by Lemma 10 and standard norm inequalities we have

$$\rho(\mathbf{A}) = \max_{\vec{x}\in\mathbb{R}^E} \frac{\left\|\mathbf{U}^{-1}\mathbf{W}\mathbf{B}\mathcal{L}^\dagger\mathbf{B}^T\mathbf{U}\vec{x}\right\|_\infty}{\left\|\vec{x}\right\|_\infty} \leq \max_{\vec{x}\in\mathbb{R}^E} \frac{\left\|\mathbf{U}\mathbf{B}\mathcal{L}^\dagger\mathbf{B}^T\mathbf{U}\vec{x}\right\|_2}{\frac{1}{\sqrt{|E|}}\left\|\vec{x}\right\|_2} = \sqrt{|E|}\cdot\left\|\mathbf{U}\mathbf{B}\mathcal{L}^\dagger\mathbf{B}^T\mathbf{U}\right\|_2$$

The result follows from the fact in [29] that $\mathbf{\Pi} \stackrel{\text{def}}{=} \mathbf{U}\mathbf{B}\mathcal{L}^\dagger\mathbf{B}^T\mathbf{U}$ is an orthogonal projection, and therefore $\left\|\mathbf{\Pi}\right\|_2 \leq 1$ and the fact in [31, 12, 14, 11] that $\mathcal{T}\left(\mathcal{L}^\dagger\right) = \widetilde{O}(|E|)$. □

Assuming Theorem 16 and Theorem 17, which we prove in the next two sections, we prove that low-congestion oblivious routings can be constructed efficiently.

Theorem 19 (Recursive construction)**.** *Given an undirected capacitated graph $G = (V, E, \vec{\mu})$ with capacity ratio U. Assume $U = \mathsf{poly}(|V|)$. We can construct an oblivious routing algorithm $\mathbf{A}$ on G in time*

$$O(|E|2^{O(\sqrt{\log|V|\log\log(|V|)})})$$

such that

$$\mathcal{T}(\mathbf{A}) = |E|2^{O(\sqrt{\log|V|\log\log(|V|)})} \quad \text{and} \quad \rho(\mathbf{A}) = 2^{O(\sqrt{\log|V|\log\log(|V|)})}.$$

Proof. Let c be the constant hidden in the exponent terms, including $\tilde{O}(\cdot)$ and $\mathsf{poly}(\cdot)$ in Theorem 16 and Theorem 17. Apply Theorem 16 to construct a sparse graph $G^{(1)}$, then apply Theorem 17 with $t = \left\lceil 2^{\sqrt{\log|V|\log\log(|V|)}}\right\rceil$ to get t graphs $G_1^{(1)}, \cdots G_t^{(1)}$ such that each graphs have at most $O\left(\frac{1}{t}|E|\log^{2c}|V|\log(U)\right)$ vertices and at most $U\cdot|V|^{2c}$ capacity ratio.

Repeat this process on each $G_i^{(1)}$, it produces t^2 graphs $G_1^{(2)}, \cdots, G_{t^2}^{(2)}$. Keep doing this until all graphs G_i produced have $O(1)$ vertices. Let k be the highest level we go through in this process. Since at the k-th level the number of vertices of each graph is at most $O\left(\frac{1}{t^k}|E|\log^{2kc}|V|\log^{2k}(U|V|^{2ck})\right)$ vertices, we have $k = O\left(\sqrt{\frac{\log|V|}{\log\log(|V|)}}\right)$.

On each graph G_i, we use Theorem 18 to get an oblivious routing algorithm $\mathbf{A}_i$ for each G_i with

$$\mathcal{T}(\mathbf{A}_i) = O(1) \quad \text{and} \quad \rho(\mathbf{A}_i) = O(1).$$

Then, the Theorem 17 and 16 shows that we have an oblivious routing algorithm $\mathbf{A}$ for G with

$$\mathcal{T}(\mathbf{A}) = O(tk|E|\log^{ck}(|V|)\log^{2k}(U|V|^{2ck})) \quad \text{and} \quad \rho(\mathbf{A}) = O(\log^{2kc}|V|\log^k(U|V|^{2ck})).$$

The result follows from $k = O\left(\sqrt{\frac{\log|V|}{\log\log(|V|)}}\right)$ and $t = \left\lceil 2^{\sqrt{\log|V|\log\log(|V|)}}\right\rceil$. □

Using Theorem 19, Lemma 12 and Theorem 4, we have the following almost linear time max flow algorithm on undirected graph.

Theorem 20. *Given an undirected capacitated graph $G = (V, E, \vec{\mu})$ with capacity ratio U. Assume $U = poly(|V|)$. There is an algorithm finds an $(1-\varepsilon)$ approximate maximum flow in time*

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

5 Flow Sparsifiers

In this section, we define and construct efficient flow sparsifiers, that is, algorithms for sparsifying a graph and mapping flows in the sparse graph into the original with low congestion in time $\tilde{O}(m)$. Notice that our flow sparsifiers aim to reduce the number of edges, and are different from the flow sparsifiers of Leighton and Moitra [17], which work in a different setting and reduce the number of vertices.

Definition 21 (Flow Sparsifier)**.** *We call an algorithm an efficient (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:*** *G' is h-sparse, i.e.*
$$|E'| \leq h$$
- ***Cut Approximation:*** *G' is an ε-cut approximation of G, i.e.*
$$\forall S \subseteq V \quad : \quad (1-\varepsilon)\mu(\partial(S)) \leq \mu'(\partial(S)) \leq (1+\varepsilon)\mu(\partial(S))$$
- ***Flow Approximation:*** *$\mathbf{M}$ has congestion at most α, i.e.*
$$\mathrm{cong}(\mathbf{M}) \leq \alpha.$$
- ***Efficiency:*** *Both the algorithm and any matrix-vector product involving $\mathbf{M}$ can be run in $\tilde{O}(m)$ time.*

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 $\mathbf{M}$.

Theorem 22. *Consider a graph $G = (V, E, \mu)$ and let $G' = (V, E', \mu')$ be given by an (h, ε, α)-flow sparsifier of G. Then, for any set of k demands $D = \vec{\chi}_1, \vec{\chi}_2, \ldots, \vec{\chi}_k$ between vertex pairs of V, we have:*

$$\mathrm{opt}_{G'}(D) \leq \frac{O(\log k)}{1-\varepsilon} \cdot \mathrm{opt}_G(D). \tag{2}$$

Given the optimum flow $\{f_i^\star\}$ over G', we have

$$\mathrm{cong}_G(\{\mathbf{M}f_i^\star\}) \leq \alpha \cdot \mathrm{opt}_{G'}(D) \leq \frac{O(\alpha \log k)}{1-\varepsilon} \cdot \mathrm{opt}_G(D).$$

Proof. By the flow-cut gap theorem of Aumann and Rabani [4], we have that, for any set of k demands D on V we have:

$$\mathrm{opt}_{G'}(D) \geq O(\log k) \cdot \min_{S \subset V} \frac{\mu'(\partial(S))}{D(\partial(S))}.$$

where $D(\partial(S))$ denotes the total amount of demand separated by the cut between S and $\bar{S}$. As any cut $S \subseteq V$ in G' has capacity $\mu'(\partial(S)) \geq (1-\varepsilon)\mu'(S)$, we have:

$$\mathrm{opt}_{G'}(D) \geq \frac{O(\log k)}{1-\varepsilon} \cdot \min_{S \subset V} \frac{\mu(\partial(S))}{D(\partial(S))} \geq \frac{O(\log k)}{1-\varepsilon} \cdot \mathrm{opt}_G(D).$$

The second part of the theorem follows as a consequence of the definition of the congestion of the embedding $\mathbf{M}$. □

Our flow sparsifiers should be compared with the cut-based decompositions of Räcke[25]. Räcke constructs a probability distribution over trees and gives explicit embeddings from G to this distribution and backwards, achieving a congestion of $O(\log n)$. However, this distribution over tree can include up to $O(n \log n)$ trees and is not suitable for an almost linear time algorithm. Flow sparsifiers answer these problems by embedding G into a single graph G', which is larger than a tree, but still sparse. Moreover, they provide an explicit efficient embedding of G' into G. Interestingly, the embedding from G to G' is not necessary for our notion of flow sparsifier, and is replaced by the cut-approximation guarantee. This requirement, together with the application of the flow-cut gap [4], lets us argue that the optimal congestion of a k-commodity flow problem can change at most by a factor of $O(\log k)$ between G and G'.

5.0.1 Main Theorem on Flow Sparsifiers and Proof of Theorem 16

The main goal of this section will be to show the following theorem:

Theorem 23. *For any constant $\varepsilon \in (0,1)$, there is an efficient $(\tilde{O}(n), \varepsilon, \tilde{O}(1))$-flow sparsifier.*

Assuming Theorem 23, we can now prove Theorem 16, the main theorem necessary for edge reduction in our construction of low-congestion projections.

Proof of Theorem 16. We apply the flow sparsifier of Theorem 23 to $G = (V, E, \vec{\mu})$ and obtain output $G' = (V, E', \vec{\mu}')$ with embedding $\mathbf{M}$. By the definition of efficient flow sparsifier, we know that the capacity ratio U' of G' is at most $U \cdot \mathsf{poly}(|V|)$, as required. Moreover, again by Theorem 23, G' has at most $\tilde{O}(|V|)$ edges. Given an oblivious routing $\mathbf{A}'$ on G' consider the oblivious routing $\mathbf{A} \stackrel{\text{def}}{=} \mathbf{M}\mathbf{A}'$. By the definition of flow sparsifier, we have that $\mathcal{T}(\mathbf{M}) = \widetilde{O}(|E|)$. Hence $\mathcal{T}(\mathbf{A}) = \mathcal{T}(\mathbf{M}) + \mathcal{T}(\mathbf{A}') = \widetilde{O}(|E|) + \mathcal{T}(\mathbf{A}')$.

To complete the proof, we bound the competivity ratio $\rho(\mathbf{A})$. Using the same argument as in Lemma 10, we can write $\rho(\mathbf{A})$ as

$$\rho(\mathbf{A}) = \max_{D} \frac{\mathrm{cong}_G(\{\mathbf{A}D\})}{\mathrm{opt}_G(D)} \leq \max_{D_\infty} \frac{\mathrm{cong}_G(\mathbf{A}D_\infty)}{\mathrm{opt}_G(D_\infty)},$$

where D_∞ is the set of demands that result by taking the routing of every demand in D by $\mathrm{opt}(D)$ and splitting it up into demands on every edge corresponding to the flow sent by $\mathrm{opt}(D)$. Notice that D_∞ has at most $\|E\|$ demands that are routed between pairs of vertices in V. Then, because G' is an ε-cut approximation to G, the flow-cut gap of Aumann and Rabani [4] guarantees that

$$\mathrm{opt}_G(D_\infty) \geq \frac{1}{O(\log n)} \mathrm{opt}_{G'}(D_\infty).$$

As a result, we obtain:

$$\begin{aligned}\rho(\mathbf{A}) &\leq O(\log n) \cdot \max_{D_\infty} \frac{\mathrm{cong}_G(\mathbf{A}D_\infty)}{\mathrm{opt}_{G'}(D_\infty)} = O(\log n) \cdot \max_{D_\infty} \frac{\mathrm{cong}_G(\mathbf{M}\mathbf{A}'D_\infty)}{\mathrm{opt}_{G'}(D_\infty)} \\ &\leq O(\log n) \cdot \mathrm{cong}(\mathbf{M}) \cdot \max_{D_\infty} \frac{\mathrm{cong}_{G'}(\mathbf{A}'D_\infty)}{\mathrm{opt}_{G'}(D_\infty)} \leq \widetilde{O}(\rho(\mathbf{A}')).\end{aligned}$$

□

5.0.2 Techniques

We will construct flow sparsifiers by taking as a starting point the construction of spectral sparsifiers of Spielman and Teng [32]. Their construction achieves a sparsity of $\widetilde{O}\left(\frac{m}{\varepsilon^2}\right)$ edges, while guaranteeing an ε-spectral approximation.

As the spectral approximation implies the cut approximation, the construction in [32] suffices to meet the first two conditions in Definition 21. Moreover, their algorithm also runs in time $\tilde{O}(m)$, meeting the fourth condition. Hence, to complete the proof of Theorem 23, we will modify the construction of Spielman and Teng to endow their spectral sparsifier G' with an embedding $\mathbf{M}$ onto G of low congestion that can be both

computed and invoked efficiently. The main tool we use in constructing $\mathbf{M}$ is the notion of electrical-flow routing and the fact that electrical-flow routing schemes achieve a low competitive ratio on near-expanders and subsets thereof [9, 15].

To exploit this fact and construct a flow sparsifier, we follow Spielman and Teng [32] and partition the input graph into vertex sets, where each sets induces a near-expanders 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 and by using standard sparsification techniques we get the sparsity and cut approximation properties for free. The flow embedding follows from the fact that the electrical-flow routing is competitive within these near-expander subgraphs.

In the next two subsections, we introduce the necessary concept about electric flow routing and prove that electric flow routing achieves low competitive ratio over near-expanders (and subsets of near-expanders).

5.1 Subgraph Routing

Given an oblivious routing strategy $\mathbf{A}$ we may be interested only in routing demands coming from a subset of edge $F \subseteq E$. In this setting, given a set of demands D routable in F we let $\text{opt}_p^F(D)$ denote the minimal congestion achieved by any routing restricted to only sending flow on edges in F and we measure the F-competetive ratio of $\mathbf{A}$ by

$$\rho^F(\mathbf{A}) \stackrel{\text{def}}{=} \max_{D \text{ routable in } F} \frac{\text{cong}(\mathbf{A}D)}{\text{opt}^F(D)}$$

As before, we can upper bound the F-competitive ratio $\rho^F(\mathbf{A})$ by operator norms.

Lemma 24. *Let $\mathbb{1}_F \in \mathbb{R}^E$ denote the indicator vector for set F (i.e. $\mathbb{1}_F(e) = 1$ if $e \in F$ and $\mathbb{1}_F(e) = 0$) and let $\mathbf{I}_F \stackrel{\text{def}}{=} diag(\mathbb{1}_F)$. For any $F \subseteq E$ we have*

$$\rho^F(\mathbf{A}) = \left\| \left| \mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\mathbf{I}_F \right| \right\|_\infty$$

Proof. We use the same reasoning as the non-subgraph case. For a set of demands $D = \{\vec{d}_i\}$, we consider D_P^F, the demands on the edges in F used by $\text{opt}^F(D)$. Then, it is the case that $\text{opt}^F(D) = \text{opt}^F(D_P)$ and we know that cost of obliviously routing D_P is greater than the cost of obliviously routing D. Therefore we have

$$\begin{aligned}
\rho^F &= \max_{\vec{x}\in\mathbb{R}^E \,:\, \mathbf{I}_{E\setminus F}\vec{x}=0} \frac{\left\| \sum_{e\in E} |\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbb{1}_e\vec{x}_e| \right\|_\infty}{\left\|\mathbf{U}^{-1}\vec{x}\right\|_\infty} \\
&= \max_{\vec{y}\in\mathbb{R}^E \,:\, \mathbf{I}_{E\setminus F}\vec{y}=0} \frac{\left\| \sum_{e\in E} |\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\mathbb{1}_e\vec{y}_e| \right\|_\infty}{\left\|\vec{y}\right\|_\infty} \\
&\le \max_{\vec{y}\in\mathbb{R}^E} \frac{\left\| \sum_{e\in E} |\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\mathbf{I}_F\mathbb{1}_e\vec{y}_e| \right\|_\infty}{\left\|\vec{y}\right\|_\infty} \qquad \text{(Having } x_e \neq 0 \text{ for } e \in E\setminus F \text{ decreases the ratio.)}
\end{aligned}$$

□

5.2 Electrical-Flow Routings

In this section, we define the notion of electrical-flow routing and prove the results necessary to construct flow sparsifiers. Recall that $\mathbf{R}$ is the diagonal matrix of resistances and the Laplacian $\mathcal{L}$ is defined as $\mathbf{B}^T\mathbf{R}^{-1}\mathbf{B}$. For the rest of this section, we assume that resistances are set as $\mathbf{R} = \mathbf{U}^{-1}$.

Definition 25. *Consider a graph $G = (V, E, \mu)$ and set the edge resistances as $r_e = \frac{1}{\mu_e}$ for all $e \in E$. The oblivious electrical-flow routing strategy is the linear operator $\mathbf{A}_\mathcal{E}$ defined as*

$$\mathbf{A}_\mathcal{E} \stackrel{\text{def}}{=} \mathbf{R}^{-1}\mathbf{B}\mathcal{L}^\dagger,$$

In words, the electrical-flow routing strategy is the routing scheme that, for each demand $\vec{\chi}$ sends the electrical flow with boundary condition $\vec{\chi}$ on the graph G with resistances $\mathbf{R} = \mathbf{U}^{-1}$.

For the electrical-flow routing strategy $\mathbf{A}_{\mathcal{E}}$, the upper bound on the competitive ratio $\rho(\mathbf{A}_{\mathcal{E}})$ in Lemma 10 can be rephrased in terms of the voltages induced on G by electrically routing an edge $e \in E$. This interpretation appears in [9, 15].

Lemma 26. *Let $\mathbf{A}_{\mathcal{E}}$ be the electrical-flow routing strategy. For an edge $e \in E$, we let the voltage vector $\vec{v}_e \in \mathbb{R}^V$ be given by $\vec{v}_e \stackrel{\text{def}}{=} \mathcal{L}^{\dagger}\vec{\chi}_e$. For $\mathbf{R} = \mathbf{U}^{-1}$, we then have*

$$\rho(\mathbf{A}_{\mathcal{E}}) \leq \max_{e \in E} \sum_{(a,b) \in E} \frac{|v_e(a) - v_e(b)|}{r_{ab}}.$$

Proof. We have:

$$\rho(\mathbf{A}_{\mathcal{E}}) \leq \left\|\mathbf{B}\mathcal{L}^{\dagger}\mathbf{B}^{\mathbf{T}}\mathbf{R}^{-\mathbf{1}}\right\|_{\infty} \leq \max_{e \in E} \left\|\mathbf{R}^{-\mathbf{1}}\mathbf{B}\mathcal{L}^{\dagger}\mathbf{B}^{\mathbf{T}}\mathbb{1}_e\right\|_1 = \max_{e \in E} \sum_{(a,b) \in E} \frac{|v_e(a) - v_e(b)|}{r_{ab}}.$$

□

The same reasoning can be extended to the subgraph-routing case to obtain the following lemma.

Lemma 27. *For $F \subseteq E$ and $\mathbf{R} = \mathbf{U}^{-1}$ we have*

$$\rho^F(\mathbf{A}_{\mathcal{E}}) \leq \max_{e \in E} \sum_{(a,b) \in F} \frac{|v_e(a) - v_e(b)|}{r_{ab}}.$$

Proof. As before, we have:

$$\begin{aligned} \rho^F(\mathbf{A}_{\mathcal{E}}) &\leq \left\|\mathbf{B}\mathcal{L}^{\dagger}\mathbf{B}^{T}\mathbf{R}^{-1}\mathbf{I}_F\right\|_{\infty} && \text{(By Lemma 24)} \\ &= \max_{e \in E} \left\|\mathbf{I}_F\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^{\dagger}\mathbf{B}^{T}\mathbb{1}_e\right\|_1 = \max_{e \in E} \sum_{(a,b) \in F} \frac{|v_e(a) - v_e(b)|}{r_{ab}} \end{aligned}$$

□

5.2.1 Bounding the Congestion

In this section, we prove that we can bound the F-competitive ratio of the oblivious electrical-routing strategy as long as the edges F that the optimum flow is allowed to route over are contained within an induced expander $G(U) = (U, E(U))$ for some $U \subseteq V$. . Towards this we provide and prove the following lemma. This is a generalization of a similar lemma proved in [9].

Lemma 28. *For weighted graph $G = (V, E, w)$ with integer weights and vertex subset $U \subseteq V$ the following holds:*

$$\forall e \in E \;\; : \;\; \left\|\mathbf{I}_{E(U)}\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^{\dagger}\vec{\chi}_e\right\|_1 \leq \frac{8\log(\text{vol}(G(U)))}{\Phi(G(U))^2}$$

Proof. Let $v \stackrel{\text{def}}{=} \mathcal{L}^{\dagger}\vec{\chi}_e$ and recall that with this definition

$$\left\|\mathbf{I}_{E(U)}\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^{\dagger}\vec{\chi}_e\right\|_1 = \sum_{(a,b) \in G(U)} \frac{|v(a) - v(b)|}{r_{ab}} = \sum_{(a,b) \in G(U)} w_{ab} \cdot |v(a) - v(b)| \tag{3}$$

We define the following vertex subsets:

$$\forall x \in \mathbb{R} \;\; : \;\; S_x^{\leq} \stackrel{\text{def}}{=} \{a \in U \mid v(a) \leq x\} \quad \text{and} \quad S_x^{\geq} \stackrel{\text{def}}{=} \{a \in U \mid v(a) \geq x\}$$

Since adding a multiple of the all-ones vector to v does not change the quantity of interest in Equation 3, we can assume without loss of generality that

$$\mathrm{vol}_{G(U)}(S_0^{\geq}) \geq \frac{1}{2}\left(\mathrm{vol}(G(U))\right) \quad \text{and} \quad \mathrm{vol}_{G(U)}(S_0^{\leq}) \geq \frac{1}{2}\left(\mathrm{vol}(G(U))\right).$$

For any vertex subset $S \subseteq U$, we denote the flow out of S and the weight out of S by

$$f(S) \stackrel{\text{def}}{=} \sum_{e=(a,b)\in E(U)\bigcap \partial(S)} w_e|v(a) - v(b)|, \quad \text{and} \quad w(S) \stackrel{\text{def}}{=} \sum_{e\in E(U)\bigcap \partial(S)} w_e.$$

At this point, we define a collections of subsets $\{C_i \in S_0^{\geq}\}$. For an increasing sequence of real numbers $\{c_i\}$, we let

$$C_i \stackrel{\text{def}}{=} S_{c_i}^{\geq}$$

. We define the sequence $\{c_i\}$ inductively as follows:

$$c_0 \stackrel{\text{def}}{=} 0 \quad , \quad c_i \stackrel{\text{def}}{=} c_{i-1} + \Delta_{i-1} \quad \text{, and} \quad \Delta_i \stackrel{\text{def}}{=} 2\frac{f(C_i)}{w(C_i)}.$$

In words, the c_{i+1} equals the sum of c_i and an increase Δ_i which depends on how much the cut $\delta(C_i) \cap E(U)$ was congested by the electrical flow.

Now, $l_i \stackrel{\text{def}}{=} w(\partial_{E(U)}(C_{i-1}) - \partial_{E(U)}(C_i))$, i.e. the weight of the edges in $E(U)$ cut by C_{i-1} but not by C_i. We get

$$\begin{aligned}
\mathrm{vol}(C_{i+1}) &\leq \mathrm{vol}(C_i) - l_i \\
&\leq \mathrm{vol}(C_i) - \frac{w(C_i)}{2} && \text{(By choice of } l_i \text{ and } \Delta_i) \\
&\leq \mathrm{vol}(C_i) - \frac{1}{2}\mathrm{vol}(C_i)\Phi(G(U)) && \text{(Definition of conductance)}
\end{aligned}$$

Applying this inductively and using our assumption on $\mathrm{vol}(S_0^{\geq})$ we have that

$$\mathrm{vol}(C_i) \leq \left(1 - \frac{1}{2}\Phi_{G(U)}\right)^i \mathrm{vol}(C_0) \leq \frac{1}{2}\left(1 - \frac{1}{2}\Phi(G(U))\right)^i \mathrm{vol}(G(U))$$

Since $\phi(G(U)) \in (0,1)$, for $j+1 = \frac{2\log(\mathrm{vol}(G(U)))}{\Phi(G(U))}$ we have that $\mathrm{vol}(S_j) \leq \frac{1}{2}$. Since $\mathrm{vol}(S_i)$ decreases monotonically with i, if we let r be the smallest value such that $C_{r+1} = \emptyset$, we must have

$$r \leq \frac{2 \cdot \log(\mathrm{vol}(G(U)))}{\Phi(G(U))}$$

Since v corresponds to a unit flow, we know that $f(C_i) \leq 1$ for all i. Moreover, by the definition of conductance we know that $w(C_i) \geq \Phi(G(U)) \cdot \mathrm{vol}(C_i)$. Therefore,

$$\Delta_i \leq \frac{2}{\Phi(G(U)) \cdot \mathrm{vol}(C_i)}.$$

We can now bound the contribution of $C_0^{\geq}$ to the volume of the linear embedding v. In the following, for a

vertex $a \in V$, we let $d(a) \stackrel{\text{def}}{=} \sum_{e=\{a,b\} \in E(U)} w_e$ be the degree of a in $E(U)$.

$$\begin{aligned}
\sum_{a \in C_0^{\geq}} d(a)v(a) &= \sum_{i=0}^{r} \left[\sum_{a \in C_i - C_{i+1}} d(a)v(a) \right] \\
&\leq \sum_{i=0}^{r} \left[\sum_{a \in C_i - C_{i+1}} d(a) \left(\sum_{i=0}^{r} \Delta_i \right) \right] && \text{(By definition of } C_i\text{)} \\
&\leq \sum_{i=0}^{r} \left[(\text{vol}(C_i) - \text{vol}(C_{i+1})) \cdot \left(\sum_{i=0}^{r} \Delta_i \right) \right] \\
&= \sum_{i=0}^{r} \text{vol}(C_i)\Delta_i \leq \frac{2r}{\Phi(G(U))} && \text{(Rearrangement and fact that vol}(C_{r+1}) = 0\text{)}
\end{aligned}$$

By repeating the same argument on $S_0^{\leq}$, we get that $\sum_{a \in S_0^{\leq}} d(a)v(a) \leq \frac{2r}{\Phi(G(U))}$. Putting this all together yields

$$\left\| \mathbf{I}_{E(U)} \mathbf{R}^{-1} \mathbf{B} \mathcal{L}^{\dagger} \vec{\chi}_e \right\| = \sum_{(a,b) \in G(U)} w_{ab} \cdot |v(a) - v(b)| \leq \sum_{a \in G(U)} d(a)v(a) \leq \frac{4r}{\Phi(G(U))}$$

□

From this lemma and Lemma 27, the following is immediate:

Lemma 29. *Let $F \subseteq E$ be contained within some vertex induced subgraph $G(U)$, then for $\mathbf{R} = \mathbf{U}^{-1}$ we have*

$$\rho^{F}(\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^{\dagger}) \leq \rho^{E(U)}(\mathbf{R}^{-1}\mathbf{B}\mathcal{L}^{\dagger}) \leq \frac{8 \log(\text{vol}(G(U)))}{\Phi(G(U))^2}$$

5.3 Construction and Analysis of Flow Sparsifiers

In the remainder of this section we show how to produce an efficient $O(\log^c)$-flow sparsifier for some fixed constant c, proving Theorem 23. In this version of the paper, we make no attempt to optimize the value of c. For the rest of this section, we again assume that we choose the conductance of an edge to be the capacity an edge, i.e. $\mathbf{U} = \mathbf{W} = \mathbf{R}^{-1}$.

As discussed before, our approach follows closely that of Spielman and Teng [32] to the construction of spectral sparsifiers. The first step of this line of attack is to reduce the problem to the unweighted case.

Lemma 30. *Given an efficient (h, ε, α)-flow-sparsifier algorithm for unweighted graphs, it is possible to construct an efficient $(h \cdot \log U, \varepsilon, \alpha)$-flow-sparsifier algorithm for weighted graphs $G = (V, E, \mu)$ with capacity ratio U obeying*

$$U = \frac{\max_{e \in E} \mu_e}{\min_{e \in E} \mu_e} = \mathsf{poly}(|V|).$$

Proof. We write each edge in binary so that $G = \sum_{i=0}^{\log U} 2^i G_i$ for some unweighted graphs $\{G_i = (V, E_i\}_{i \in [\log U]}\}$ where $\|E_i\| \leq m$ for all i. We now apply the unweighted flow-sparsifier to each G_i in turn to obtain graphs $\{G_i'\}$. We let $G' \stackrel{\text{def}}{=} \sum_{i=0}^{\log U} 2^i G_i'$ be the weighted flow-sparsified graph. By the assumption on the unweighted flow-sparsifier, each G_i' is h-sparse, so that G' must have at most $h \cdot \log U$ edges. Similarly, G' is an ε-cut approximation of G, as each G_i' is an ε-cut approximation of the corresponding G_i. Letting $\mathbf{M}_i$ be the embedding of G_i' into G_i, we can consider the embedding $\mathbf{M} = \sum_{i=0}^{\log U} 2^i \mathbf{M}_i$ of G' into G. As each $\mathbf{M_i}$ has congestion bounded by α, it must be the case that $\mathbf{M}$ also has congestion bounded by α. The time to run the weighted flow sparsifier and to invoke $\mathbf{M}$ is now $\tilde{O}(m) \cdot \log U = \tilde{O}(m)$ by our assumption on U. □

The next step is to construct a routine which flow-sparsifies a constant fraction of the edges of E. This routine will then be applied iteratively to produce the final flow-sparsifier.

Lemma 31. *On input 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 $\vec{w_{F'}} \in \mathbb{R}^E, \text{support}(w_{F'}) = F'$, and an embedding $\mathbf{H} : \mathbb{R}^{F'} \to \mathbb{R}^E$ with the following properties:*

1. *F contains most of the volume of G, i.e.*
$$|F| \geq \frac{|E|}{2};$$
2. *F' contains only $\tilde{O}(n)$ edges, i.e. $|F'| \leq \tilde{O}(n)$.*
3. *The weights $w_{F'}$ are bounded*
$$\forall e \in F' \ , \ \frac{1}{\mathsf{poly}(n)} \leq w_{F'}(e) \leq n.$$
4. *The graph $H' = (V, F', w_{F'})$ is an ε-cut approximation to $H = (V, F)$, i.e.*
$$(1-\varepsilon)|\delta_H(S)| \leq w_{F'}(\delta(S)) \leq (1+\varepsilon)|\delta_H(S)|.$$
5. *The embedding $\mathbf{H}$ from $H = (V, F', w_{F'})$ to G has bounded congestion*
$$\text{cong}(\mathbf{H}) = \widetilde{O}(1).$$
and can be applied in time $\tilde{O}(m)$.

Given Lemma 30 and Lemma 31, it is straightforward to complete the proof of Theorem 23.

Proof. Using Lemma 30, we reduce the objective of Theorem 23 to running a $(\widetilde{O}(n), \varepsilon, \widetilde{O}(1))$-flow sparsifier on $\log U$ unweighted graphs, where we use the fact that $U \leq \mathsf{poly}(n)$. To construct this unweighted flow sparsifier, we apply Lemma 31 iteratively as follows. Starting with the instance unweighted graph $G_1 = (V, E_1)$, we run the algorithm of Lemma 31 on the current graph $G_t = (V, E_t)$ to produce the sets F_t and F'_t, the weight vector $w_{F'_t}$ and the embedding $\mathbf{H_t} : \mathbb{R}^{F'_t} \to R^E$. To proceed to the next iteration, we then define $E_{t+1} \stackrel{\text{def}}{=} E_t \setminus F_t$ and move on to G_{t+1}.

By Lemma 31, at every iteration t, $|F_t| \geq \frac{1}{2} \cdot |E_t|$, so that $|E_{t+1}| \leq \frac{1}{2} \cdot |E_t|$. This shows that there can be at most $T \leq \log(|E_1|) = O(\log n)$ iterations.

After the last iteration T, we have effectively partitioned E_1 into disjoint subsets $\{F_t\}_{t \in [T]}$, where each F_t is well-approximated but the weighted edgeset F'_t. We then output the weighted graph $G' = (V, E' \stackrel{\text{def}}{=} \cap_{t=1}^T F'_t, w' \stackrel{\text{def}}{=} \sum_{t=1}^T w_{F'_t})$, which is the sum of the disjoint weighted edges sets $\{F'_t\}_{t \in [T]}$. We also output the embedding $\mathbf{M} : \mathbb{R}^{E'} \to \mathbb{R}^E$ from G' to G, defined as the direct sum

$$\mathbf{M} = \bigoplus_{t=1}^{T} \mathbf{H_t}.$$

In words, $\mathbf{M}$ maps an edge $e' \in E'$ by finding t for which $e' \in F'_t$ and applying the corresponding $\mathbf{H_t}$.

We are now ready to prove that this algorithm with output G' and $\mathbf{M}$ is an efficient $(\widetilde{O}(n), \varepsilon, \widetilde{O}(n))$-flow sparsifier. To bound the capacity ratio U' of G', we notice that

$$U' \leq \max_t \frac{\max_{e \in F'_t} w_{F'_t}(e)}{\min_{e \in F'_t} w_{F'_t}(e)} \leq \mathsf{poly}(n),$$

where we used the fact that the sets F'_t are disjoint and the guarantee on the range of $w_{F'_t}$.

Next, we bound the sparsity of G'. By Lemma 31, F'_t contains at most $\widetilde{O}(n)$ edges. As a result, we get the required bound

$$|E'| = \sum_{t=1}^{T} |F'_t| \leq \widetilde{O}(Tn) = \widetilde{O}(n).$$

For the cut approximation, we consider any $S \subseteq V$. By the cut guarantee of Lemma 31, we have that, for all $t \in [T]$,

$$(1-\varepsilon)|\delta(S) \cap F_t| \leq w_{F'_t}(\delta(S) \cap F'_t) \leq (1+\varepsilon)|\delta(S) \cap F_t|.$$

Summing over all t, as $E' = \dot{\bigcup} F'_t$ and $E = \dot{\bigcup} F_t$, we obtain the required approximation

$$(1-\varepsilon)|\delta_G(S)| \leq w'(\delta(S)) \leq (1+\varepsilon)|\delta_G(S)|.$$

The congestion of $\mathbf{M}$ can be bounded as follows

$$\text{cong}(\mathbf{M}) \leq \sum_{t=1}^{T} \text{cong}(\mathbf{H_t}) = \widetilde{O}(T) = \widetilde{O}(1).$$

To conclude the proof, we address the efficiency of the flow sparsifier. The algorithm applies the routine of Lemma 31 for $T = \widetilde{O}(1)$ times and hence runs in time $\widetilde{O}(m)$, as required. Invoking the embedding $\mathbf{M}$ requires invoking each of the T embeddings $\mathbf{H_t}$. This takes time $\widetilde{O}(Tm) = \widetilde{O}(m)$.

□

5.3.1 Flow Sparsification of Unweighted Graphs: Proof of Lemma 31

In this subsection, we prove Lemma 31. Our starting point is the following decomposition statement, which shows that we can form a partition of an unweighted graph where most edges do not cross the boundaries and the subgraphs induced within each set of this partition are near-expanders. The following lemma is implicit in Spielman and Teng's local clustering approach to spectral sparsification [32].

Lemma 32 (Decomposition Lemma)**.** *For an unweighted graph $G = (V, E)$, in $\tilde{O}(m)$-time we can produce a partition $V_1, \ldots, V_k$ of V and a collection of sets $S_1, \ldots, S_k \subseteq V$ with the following properties:*

- *For all i, S_i is contained in V_i.*
- *For all i, there exists a set T_i with $S_i \subseteq T_i \subseteq V_i$, such that*

$$\Phi(G(T_i)) \geq \Omega\left(\frac{1}{\log^2 n}\right).$$

- *At least half of the edges are found within the sets $\{S_i\}$, i.e.*

$$\sum_{i=1}^{k} |E(S_i)| = \sum_{i=1}^{k} |\{e = \{a, b\} : a \in S_i, b \in S_i\}| \geq \frac{1}{2}|E|.$$

To design an algorithm satisfying the requirements of Lemma 31, we start by appling the Decomposition Lemma to our unweighted input graph $G = (V, E)$ to obtain the partition $\{V_i\}_{i\in[k]}$ and the sets $\{S_i\}_{i\in[k]}$. We let $G_i \stackrel{\text{def}}{=} (S_i, E(S_i))$. To reduce the number of edges, while preseving cuts, we apply a spectral sparsification algorithm to each G_i. Concretely, by applying the spectral sparsification by effective resistances of Spielman and Srivastava [29] to each G_i, we obtain weighted graphs $G'_i = (S_i, E'_i \subseteq E(S_i), w'_i)$ in time $\sum_{i=1}^{k} \widetilde{O}(|E(S_i)|) \leq \widetilde{O}(|E|)$ with $|E'_i| \leq \widetilde{O}(|S_i|)$ and the property that cuts are preserved[6] for all i:

$$\forall S \subseteq S_i \ , \ (1-\varepsilon) \cdot \delta_{G_i}(S) \leq w'_i(\delta(S)) \leq (1+\varepsilon) \cdot \delta_{G_i}(S).$$

Moreover, the spectral sparsification of [29] constructs the weights $\{w'_i(e)\}_{e\in E'_i}$ such that

$$\forall e \in E'_i \ , \ \frac{1}{\mathsf{poly}(n)} \leq \frac{1}{\mathsf{poly}(|S_i)} \leq w'_i(e) \leq |S_i| \leq n.$$

[6]The spectral sparsification result actually yields the stronger spectral approximation guarantee, but for our purposes the cut guarantee suffices.

To complete the description of the algorithm, we output the partition $(F, \bar{F})$ of E, where

$$F \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} E(S_i).$$

We also output the set of weighted sparsified edges F'.

$$F' \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} E'_i.$$

The weight $w_{F'}(e)$ of edge $e \in F'$ is given by finding i such that $e \in E'_i$ and setting $w_{F'}(e) = w'_i(e)$.

We now depart from Spielman and Teng's construction by endowing our F' with an embedding onto G. The embedding $\mathbf{H} : \mathbb{R}^{F'} \to \mathbb{R}^{E}$ of the graph $H = (V, F', w_{F'})$ to G is constructed by using the oblivious electrical-flow routing of $E(S_i)$ into $G(V_i)$. More specifically, as the sets $\{V_i\}$ partition V, the embedding $\mathbf{H}$ can be expressed as the following direct sum over the orthogonal subspaces $\{\mathbb{R}^{E(V_i)\times E'_i}\}$.

$$\mathbf{H} \stackrel{\text{def}}{=} \left(\bigoplus_{i=1}^{k} \mathbf{B}_{E(V_i)} \mathcal{L}^{\dagger}_{G(V_i)} \mathbf{B}^{T}_{E(V_i)} \mathbf{I}_{(E(V_i),E'_i)} \right),$$

where $\mathbf{I}_{(E(V_i),E'_i)}$ is the identity mapping of the edges $E'_i \subseteq E(V_i)$ of F' over V_i to the edges $E(V_i)$ of V_i in G. Notice that there is no dependence on the resistances over G as G is unweighted.

This complete the description of the algorithm. We are now ready to give the proof of Lemma 31.

Proof of Lemma 31. The algorithm described above performs a decomposition of the input graph $G = (V, E)$ in time $\widetilde{O}(m)$ by the Decomposition Lemma. By the result of Spielman and Srivastava [29], each G_i is sparsified in time $\widetilde{O}(|E(S_i)|)$. Hence, the sparsification step requires time $\widetilde{O}(m)$ as well. This shows that the algorithm runs in $\widetilde{O}(m)$-time, as required.

By the Decomposition Lemma, we know that $|F| = \sum_{i=1}^{k} |E(S_i)| \geq \frac{|E|}{2}$, which satisfies the requirement of the Lemma. Moreover, by the spectral sparsification result, we know that $|F'| = \sum_{i=1}^{k} |E'_i| \leq \sum_{i=1}^{k} \widetilde{O}(|S_i|) \leq \widetilde{O}(n)$, as required. We also saw that by construction the weights $w_{F'}$ are bounded:

$$\forall e \in F' \ , \ \frac{1}{\mathsf{poly}(n)} \leq w_{F'}(e) \leq n.$$

To obtain the cut-approximation guarantee, we use the fact that for every i, by spectral sparsification,

$$\forall S \subseteq S_i \ , \ (1-\varepsilon) \cdot \delta_{G_i}(S) \leq w'_i(\delta(S)) \leq (1+\varepsilon) \cdot \delta_{G_i}(S).$$

We have $H' = (V, F', w_{F'})$ and $H = (V, F)$. Consider now $T \subseteq V$ and apply the previous bound to $T \cap S_i$ for all i. Because $F' \subseteq F = \cup_{i=1}^{k} E(S_i)$, we have that summing over the k bounds yields

$$\forall T \subseteq V \ , \ (1-\varepsilon)|\delta_H(T)| \leq w_{F'}(\delta(T)) \leq (1+\varepsilon)|\delta_H(T)|,$$

which is the desired cut-approximaton guarantee.

Finally, we are left to prove that the embedding $\mathbf{H}$ from $H' = (V, F', w_{F'})$ to $G = (V, E)$ has low congestion and can be applied efficiently. By definition of congestion,

$$\mathrm{cong}(\mathbf{H}) = \max_{\vec{x} \in \mathbb{R}^{F'}} \frac{\left\|\mathbf{H}\vec{x}\right\|_\infty}{\left\|\mathbf{U}^{-1}_{F'}\vec{x}\right\|_\infty} = \left\| |\mathbf{H}| \mathbf{U}_{F'} \mathbb{1}_{F'} \right\|_\infty = \left\| \left| \bigoplus_{i=1}^{k} \mathbf{B}_{E(V_i)} \mathcal{L}^{\dagger}_{G(V_i)} \mathbf{B}^{T}_{E(V_i)} \mathbf{I}_{(E(V_i),E'_i)} \right| \mathbf{U}_{F'} \mathbb{1}_{F'} \right\|_\infty .$$

Decomposing $\mathbb{R}^{E}$ into the subspaces $\{\mathbb{R}^{E(V_i)}\}$ and $\mathbb{R}^{F'}$ into the subspaces $\{\mathbb{R}^{E'_i}\}$ we have:

$$\mathrm{cong}(\mathbf{H}) \leq \max_{i \in [k]} \left\| \left| \mathbf{B}_{E(V_i)} \mathcal{L}^{\dagger}_{G(V_i)} \mathbf{B}^{T}_{E(V_i)} \mathbf{I}_{(E(V_i),E'_i)} \right| \mathbf{U}_{E'_i} \mathbb{1}_{E'_i} \right\|_\infty .$$

For each $i \in [k]$, consider now the set of demands D_i over V_i, $D_i \stackrel{\text{def}}{=} \{\vec{\chi}_e\}_{e \in E'_i}$, given by the edges of E'_i with their capacities w'_i. That is, $\vec{\chi}_e \in \mathbb{R}^{V_i}$ is the demand corresponding to edge $e \in E'_i$ with weight $w'_i(e)$. Consider also the electrical routing $\mathbf{A}_{\mathcal{E},i} = \mathbf{B}_{E(V_i)}\mathcal{L}^{\dagger}_{G(V_i)}$ over $G(V_i)$. Then:

$$\text{cong}(\mathbf{H}) \leq \max_{i \in [k]} \text{cong}(\mathbf{A}_{\mathcal{E},i} D_i)$$

Notice that, by construction, D_i is routable in $G'_i = (S_i, E'_i, w'_i)$ and $\text{opt}_{G'_i}(D_i) = 1$. But, by our use of spectral sparsifiers in the construction, G'_i is an ε-cut approximation of G_i. Hence, by the flow-cut gap of Aumann and Rabani [4], we have:

$$\text{opt}_{G_i}(D_i) \leq \log(|D_i|) \cdot \text{opt}_{G'_i}(D_i) \leq \widetilde{O}(1).$$

When we route D_i oblivious in $G(V_i)$, we can consider the $E(S_i)$-competitive ratio $\rho^{E(S_i)}(\mathbf{A}_{\mathcal{E},i})$ of the electrical routing $\mathbf{A}_{\mathcal{E},i} = \mathbf{B}_{E(V_i)}\mathcal{L}^{\dagger}_{G(V_i)}$, as D_i is routable in $E(S_i)$, because $E'_i \subseteq E(S_i)$. We have

$$\text{cong}(\mathbf{H}) \leq \max_{i \in [k]} \rho^{E(S_i)}_{G(V_i)}(\mathbf{A}_{\mathcal{E},i}) \cdot \text{opt}^{E(S_i)}_{G(V_i)}(D_i) = \max_{i \in [k]} \rho^{E(S_i)}_{G(V_i)}(\mathbf{A}_{\mathcal{E},i}) \cdot \text{opt}_{G_i}(D_i),$$

Finally, putting these bounds together, we have:

$$\text{cong}(\mathbf{H}) \leq \max_{i \in [k]} \rho^{E(S_i)}_{G(V_i)}(\mathbf{A}_{\mathcal{E},i}) \cdot \text{opt}_{G_i}(D_i) \leq \widetilde{O}(1) \cdot \max_{i \in [k]} \rho^{E(S_i)}_{G(V_i)}(\mathbf{A}_{\mathcal{E},i}).$$

But, by the Decomposition Lemma, there exists T_i with $S_i \subseteq T_i \subseteq V_i$ such that

$$\Phi(G(T_i)) \geq \Omega\left(\frac{1}{\log^2 n}\right).$$

Then, by Lemma 29, we have that:

$$\rho^{E(S_i)}_{G(V_i)}(\mathbf{A}_{\mathcal{E},i}) \leq O\left(\frac{\log \text{vol}(G(T_i))}{\Phi(G(T_i))^2}\right) \leq \widetilde{O}(1).$$

This concludes the proof that $\text{cong}(\mathbf{H}) \leq \widetilde{O}(1)$. To complete the proof of the Lemma, we just notice that $\mathbf{H}$ can be invoked in time $\widetilde{O}(m)$. A call of $\mathbf{H}$ involves solving k-electrical-problems, one for each $G(V_i)$. This can be done in time $\sum_{i=1}^{k} \widetilde{O}(|E(V_i)|) \leq \widetilde{O}(m)$, using any of the nearly-linear Laplacian system solvers available, such as [11]. □

6 Removing Vertices in Oblivious Routing Construction

In this section we show how to reduce computing an efficient oblivious routing on a graph $G = (V, E)$ to computing an oblivious routing for t graphs with $\widetilde{O}(\frac{|V|}{t})$ vertices and at most $|E|$ edges. Formally we show

Theorem 33 (Node Reduction *(Restatement)*)**.** *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 $\widetilde{O}(\frac{|E| \log(U)}{t})$ 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 $\mathbf{A} \in \mathbb{R}^{E \times V}$ on G such that*

$$\mathcal{T}(\mathbf{A}) = \widetilde{O}\left(t \cdot |E| + \sum_{i=1}^{t} \mathcal{T}(\mathbf{A}_i)\right) \quad \text{and} \quad \rho(\mathbf{A}) = \widetilde{O}\left(\max_i \rho(\mathbf{A}_i)\right)$$

We break this proof into several parts. First we show how to embed G into a collection of t graphs consisting of trees minus some edges which we call *patrial tree embeddings* (Section 6.1). Then we show how to embed a partial tree embedding in an "almost j-tree" [19], that is a graph consisting of a tree and a subgraph on at most j vertices, for $j = 2t$ (Section 6.2). Finally, we show how to reduce oblivious routing on an almost j-tree to oblivious routing on a graph with at most $O(j)$ vertices by removing degree-1 and degree-2 vertices (Section 6.3). Finally, in Section 6.4 we put this all together to prove Theorem 17.

We remark that much of the ideas in the section were either highly influenced from [19] or are direct restatements of theorems from [19] adapted to our setting. We encourage the reader to look over that paper for further details regarding the techniques used in this section.

6.1 From Graphs to Partial Tree Embeddings

To prove Theorem 17, we make heavy use of spanning trees and various properties of them. In particular, we use the facts that for every pair of vertices there is a unique *tree path* connecting them, that every edge in the tree *induces a cut* in the graph, and that we can embed a graph in a tree by simply routing ever edge over its tree path and that the congestion of this embedding will be determined by the *load* the edges place on tree edges. We define these quantities formally below.

Definition 34 (Tree Path)**.** *For undirected graph $G = (V, E)$, spanning tree T, and all $a, b \in V$ we let $P_{a,b} \subseteq E$ denote the unique path from a to b using only edges in T and we let $\vec{p}_{a,b} \in \mathbb{R}^E$ denote the vector representation of this path corresponding to the unique vector sending one one unit from a to b that is nonzero only on T (i.e. $\mathbf{B}^T\vec{p}_{a,b} = \vec{\chi}_{a,b}$ and $\forall e \in E \setminus T$ we have $\vec{p}_{a,b}(e) = 0$)*

Definition 35 (Tree Cuts)**.** *For undirected $G = (V, E)$ and spanning tree $T \subseteq E$ the* edges cut by e, $\partial_T(F)$, *and the edges cut by F, $\partial_T(e)$, are given by*

$$\partial_T(e) \stackrel{\text{def}}{=} \{e' \in E \mid e' \in P_e\} \quad \text{and} \quad \partial_T(F) \stackrel{\text{def}}{=} \cup_{e \in F} \partial(e)$$

Definition 36 (Tree Load)**.** *For undirected capacitated $G = (V, E, \vec{\mu})$ and spanning tree $T \subseteq E$ the load on edge $e \in E$ by T, $\text{cong}_T(e)$ is given by $\text{load}_T(e) = \sum_{e' \in E | e \in P_{e'}} \vec{\mu}_{e'}$*

While these properties do highlight the fact that we could just embed our graph into a collection of trees to simplify the structure of our graph, this approach suffers from a high computational cost [25]. Instead we show that we can embed parts of the graph onto collections of trees at a lower computational cost but higher complexity. In particular we will consider what we call partial tree embeddings.

Definition 37 (Partial Tree Embedding [7])**.** *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*

$$\forall e \in E' \quad : \quad \vec{\mu}'(e) = \begin{cases} \text{load}_T(e) & \text{if } e \in T \setminus F. \\ \vec{\mu}(e) & \text{otherwise} \end{cases}$$

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.

$$\forall e \in E \quad : \quad \mathbf{M}_H(e) = \begin{cases} \vec{p}_e & e \notin \partial_T(F) \\ \mathbb{1}_e & \text{otherwise} \end{cases}$$

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, i.e. $\forall e \in E'$, $\mathbf{M}'_H(e) = \mathbb{1}_e$.

[7] This is a restatement of the $H(T, F)$ graphs in [19].

Note that by definition $\text{cong}(\mathbf{M}_H) \leq 1$, i.e. a graph embeds into its partial tree embedding with no congestion. However, to get embedding guarantees in the other direction more work is required. For this purpose we use a lemma from Madry [19] saying that we can construct a convex combination or a distribution of partial tree embeddings we can get such a guarantee.

Lemma 38 (Probabilistic Partial Tree Embedding [8])**.** *For any undirected capacitated graph $G = (V, E, \vec{\mu})$ and $t > 0 \in \mathbb{Z}$ in $\widetilde{O}(t \cdot m)$ time we can find a collection of partial tree embeddings $H_1 = H(G, T_1, F_k), \ldots, H_t = H(G, T_k, F_k)$ and coefficients $\lambda_i \geq 0$ with $\sum_i \lambda_i = 1$ such that $\forall i \in [t]$ we have $|F_i| = \widetilde{O}(\frac{m \log U}{t})$ and such that $\forall i \in [t]$ we have $|F_i| = \widetilde{O}(\frac{m \log U}{t})$ and such that $\sum_i \lambda_i \mathbf{M}'_{H_i}$ embeds $G' = \sum_i \lambda_i G_i$ into G with congestion $\widetilde{O}(1)$*

Using this lemma, we can prove that we can reduce constructing an oblivious routing for a graph to constructing oblivious routings on several partial tree embeddings.

Lemma 39. *Let the H_i be graphs produced byLemma 38 and for all i let $\mathbf{A}_i$ be an oblivious routing algorithm for H_i. It follows that $\mathbf{A} = \sum_i \lambda_i \mathbf{M}'_{H_i} \mathbf{A}_i$ is an oblivious routing on G with $\rho(\mathbf{A}) \leq \widetilde{O}(\max_i \rho(\mathbf{A}_i) \log n)$ and $\mathcal{T}(\mathbf{A}) = O(\sum_i \mathcal{T}(\mathbf{A}_i))$*

Proof. The proof is similar to the proof of Lemma 15. For all i let $\mathbf{U}_i$ denote the capacity matrix of graph G_i. Then using Lemma 10 we get

$$\rho(\mathbf{A}) = \left\|\mathbf{U}^{-1}\mathbf{A}\mathbf{B}^T\mathbf{U}\right\|_\infty = \left\|\sum_{i=1}^{t} \lambda_i \mathbf{U}^{-1}\mathbf{M}'_{H_i}\mathbf{A}_i\mathbf{B}^T\mathbf{U}\right\|_\infty$$

Using that $\mathbf{M}_{H_i}$ is an embedding and therefore $\mathbf{B}^T_{H_i}\mathbf{M}_{H_i} = \mathbf{B}^T$ we get

$$\rho(\mathbf{A}) = \left\|\sum_{i=1}^{t} \lambda_i \mathbf{U}^{-1}\mathbf{M}'_{H_i}\mathbf{A}_i\mathbf{B}^T_{H_i}\mathbf{M}_{H_i}\mathbf{U}\right\|_\infty \leq \max_{j,k} \left\|\sum_{i=1}^{t} \lambda_i \mathbf{U}^{-1}\mathbf{M}'_i\mathbf{U}_i\right\|_\infty \cdot \rho(\mathbf{A}_j) \cdot \text{cong}(\mathbf{M}_{H_k})$$

The result follows from $\sum_i \lambda_i \mathbf{M}'_{H_i}$ being an embedding of congestion of at most $\widetilde{O}(1)$ and $\text{cong}(\mathbf{M}_{H_k}) \leq 1$. □

6.2 From Partial Tree Embeddings To Almost-j-trees

Here we show how to reduce constructing an oblivious routing for a partial tree embedding to constructing an oblivious routing for what Madry [19] calls an "almost j-tree," the union of a tree plus a subgraph on at most j vertices. First we define such objects and then we prove the reduction.

Definition 40 (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 41. *For undirected capacitated $G = (V, E, \vec{\mu})$ and partial tree embedding $H = H(G, T, F)$ in $\widetilde{O}(|E|)$ time we can construct an almost $2 \cdot |F|$-tree $G' = (V, E', \vec{\mu}')$ with $|E'| \leq |E|$ and an embedding $\mathbf{M}'$ from G' to H such that H is embeddable into G' with congestion 2, $\text{cong}(\mathbf{M}') = 2$, and $\mathcal{T}(\mathbf{M}') = \widetilde{O}(|E|)$.*

Proof. For every $e = (a, b) \in E$, we let $v^1(e) \in V$ denote the first vertex on tree path $P_{(a,b)}$ incident to F and we let $v^2(e) \in V$ denote the last vertex incident to F on tree path $P_{(a,b)}$. Note that for every $e = (a, b) \in T$ we have that $(v^1(e), v^2(e)) = e$.

We define $G' = (V, E', \vec{\mu}')$ to simply be the graph that consists of all these $(v^1(e), v^2(e))$ pairs

$$E' = \{(a, b) \mid \exists e \in E \text{ such that } (a, b) = (v^1(e), v^2(e))\}$$

[8] This in an adaptation of Corollary 5.6 in [19]

and we define the weights to simply be the sums

$$\forall e' \in E' \quad : \quad \vec{\mu}'(e') \stackrel{\text{def}}{=} \sum_{e \in E \mid e=(v^1(e'),v^2(e'))} \vec{\mu}(e)$$

Now to embed H in G' we define $\mathbf{M}$ by

$$\forall e = (a,b) \in E \quad : \quad \mathbf{M}\mathbb{1}_e = \vec{p}_{a,v^1(e)} + \mathbb{1}_{(v^1(e),v^2(e))} + \vec{p}_{v^2(e),b}$$

and to embed G' in H we define $\mathbf{M}'$ by

$$\forall e' \in E \quad : \quad \mathbf{M}'\mathbb{1}_{e'} = \sum_{e=(a,b)\in E \mid e'=(v^1(e),v^2(e))} \frac{\vec{\mu}(e)}{\vec{\mu}'(e')} \left[\vec{p}_{v^1(e),a} + \mathbb{1}_{(a,b)} + \vec{p}_{b,v^2(e)}\right]$$

In other words we route edges in H along the tree until we encounter nodes in F and then we route them along added edges and we simply route the other way for the reverse embedding. By construction clearly the congestion of the embedding in either direction is 2.

To bound the running time, we note that by having every edge e in H maintain its $v^1(e)$ and $v^2(e)$ information, having every edge e' in E' maintain the set $\{e \in E | e' = (v^1(e), v^2(e))\}$ in a list, and using link cut trees [28] or the static tree structure in [11] to update information along tree paths we can obtain the desired value of $\mathcal{T}(\mathbf{M}')$. □

6.3 From Almost-J Trees to Less Vertices

Here we show that by "greedy elimination" [31] [12] [14], i.e. removing all degree 1 and degree 2 vertices in $O(m)$ time we can 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. Again, we remark that the lemmas in this section are derived heavily from [19] but repeated for completeness and to prove additional properties that we will need for our purposes.

We start by showing that an almost-j-tree with no degree 1 or degree 2 vertices has at most $O(j)$ vertices.

Lemma 42. *For any almost j-tree $G = (V, E)$ with no degree 1 or degree 2 vertices, we have $|V| \leq 3j - 2$.*

Proof. Since G is an almost j-tree, there is some $J \subseteq V$ with $|J| \leq j$ such that the removal of all edges with both endpoints in J creates a forest. Now, since $K = V - J$ is incident only to forest edges clearly the sum of the degrees of the vertices in K is at most $2(|V| - 1)$ (otherwise there would be a cycle). However, since the minimum degree in G is 3, clearly this sum is at least $3(|V| - j)$. Combining yields that $3|V| - 3j \leq 2|V| - 2$. □

Next, we show how to remove degree one vertices efficiently.

Lemma 43 (Removing Degree One Vertices)**.** *Let $G = (V, E, \vec{\mu})$ be an unweighted capacitated graph, let $a \in V$ be a degree 1 vertex, let $e = (a, b) \in E$ be the single edge incident to a, and let $G' = (V', E', \vec{\mu}')$ be the graph that results from simply removing e and a, i.e. $V' = V \setminus \{a\}$ and $E' = E \setminus \{e\}$. Given $a \in V$ and an oblivious routing algorithm $\mathbf{A}'$ in G' in $O(1)$ time we can construct an oblivious routing algorithm $\mathbf{A}$ in G such that*

$$\mathcal{T}(\mathbf{A}) = O(\mathcal{T}(\mathbf{A}') + 1) \text{ , and } \rho(\mathbf{A}) = \rho(\mathbf{A}')$$

Proof. For any demand vector $\vec{\chi}$, the only way to route demand at a in G is over e. Therefore, if $\mathbf{B}\vec{f} = \vec{\chi}$ then $\vec{f}(e) = \vec{\chi}$. Therefore, to get an oblivious routing algorithm on G, we can simply send demand at a over edge e, modify the demand at b accordingly, and then run the oblivious routing algorithm on G' on the remaining vertices. The routing algorithm we get is the following

$$\mathbf{A} \stackrel{\text{def}}{=} \mathbf{I}_{E' \to E}\mathbf{A}'(\mathbf{I} + \mathbb{1}_b\mathbb{1}_a^T) + \mathbb{1}_e\mathbb{1}_a^T$$

Since all routing algorithms send this flow on e we get that $\rho(\mathbf{A}) = \rho(\mathbf{A}')$ and since the above operators not counting $\mathbf{A}$ have only $O(1)$ entries that are not the identity we can clearly implement the operations in the desired running time. □

Using the above lemma we show how to remove all degree 1 and 2 vertices in $O(m)$ time while only increasing the congestion by $O(1)$.

Lemma 44 (Greedy Elimination)**.** *Let $G = (V, E, \vec{\mu})$ be an unweighted capacitated graph and let $G' = (V', E', \vec{\mu}')$ be the graph the results from iteratively removing vertices of degree 1 and replacing degree 2 vertices with an edge connecting its neighbors of the minimum capacity of its adjacent edges. We can construct G' in $O(m)$ time and given an oblivious routing algorithm $\mathbf{A}'$ in G' in $O(1)$ time we can construct an oblivious routing algorithm $\mathbf{A}$ in G such that* [9]

$$\mathcal{T}(\mathbf{A}) = O(\mathcal{T}(\mathbf{A}') + |E|) \ , \quad , \text{ and } \rho(\mathbf{A}) \leq 4 \cdot \rho(\mathbf{A}')$$

Proof. First we repeatedly apply Lemma 43 repeatedly to in reduce to the case that there are no degree 1 vertices. By simply array of the degrees of every vertex and a list of degree 1 vertices this can be done in $O(m)$ time. We denote the result of these operations by graph K.

Next, we repeatedly find degree two vertices that have not been explored and explore this vertices neighbors to get a path of vertices, $a_1, a_2, \ldots, a_k \in V$ for $k > 3$ such that each vertex $a_2, \ldots, a_{k-1}$ is of degree two. We then compute $j = \arg\min_{i \in [k-1]} \vec{\mu}(a_i, a_{i+1})$, remove edge (a_j, a_{j+1}) and add an edge (a_1, a_k) of capacity $\vec{\mu}(a_j, a_{j+1})$. We denote the result of doing this for all degree two vertices by K' and note that again by careful implementation this can be performed in $O(m)$ time.

Note that clearly K is embeddable in K' with congestion 2 just by routing every edge over itself except the removed edges which we route by the path plus the added edges. Furthermore, K' is embeddable in K with congestion 2 again by routing every edge on itself except for the edges which we added which we route back over the paths they came from. Furthermore, we note that clearly this embedding and the transpose of this operator is computable in $O(m)$ time.

Finally, by again repeatedly applying Lemma 43 to K' until there are no degree 1 vertices we get a graph G' that has no degree one or degree two vertices (since nothing decreased the degree of vertices with degree more than two). Furthermore, by Lemma 43 and by Lemma 15 we see that we can compose these operators to compute $\mathbf{A}$ with the desired properties. □

6.4 Putting It All Together

Here we put together the previous components to prove the main theorem of this section.

Node Reduction Theorem 17. Using Lemma 38 we can construct $G' = \sum_{i=1}^{t} \lambda_i G_i$ and embeddings $\mathbf{M}_1, \ldots, \mathbf{M}_t$ from G_i to G. Next we can apply Lemma 41 to each G_i to get almost-j-trees $G'_1, \ldots, G'_t$ and embeddings $\mathbf{M}'_1, \ldots, \mathbf{M}'_t$ from G'_i to G_i. Furthermore, using Lemma 44 we can construction graphs $G''_1, \ldots, G''_t$ with the desired properties (the congestion ratio property follows from the fact that we only add capacities during these reductions)

Now given oblivious routing algorithms $\mathbf{A}''_1, \ldots, \mathbf{A}''_t$ on the G''_i and again by Lemma 44 we could get oblivious routing algorithms $\mathbf{A}'_1, \ldots, \mathbf{A}'_t$ on the G'_i with constant times more congestion. Finally, by the guarantees of Lemma 15 we have that $\mathbf{A} \stackrel{\text{def}}{=} \sum_{i=1}^{t} \lambda \mathbf{M}_i \mathbf{M}'_i \mathbf{A}'_i$ is an oblivious routing algorithm that satisfies the requirements. □

7 Nonlinear Projection and Maximum Concurrent Flow

7.1 Gradient Descent Method for Nonlinear Projection Problem

In this section, we strengthen and generalize the **MaxFlow** algorithm to a more general setting. We believe this algorithm may be of independent interest as it includes maximum concurrent flow problem, the compressive sensing problem, etc. For some norms, e.g. $\|\cdot\|_1$ as typically of interest compressive sensing, the Nesterov algorithm [21] can be used to replace gradient descent. However, this kind of accelerated method

[9] Note that the constant of 4 below is improved to 3 in [19].

is not known in the general norm settings as good proxy function may not exist at all. Even worse, in the non-smooth regime, the minimization problem on the $\|\cdot\|_p$ with $p > 2$ can be proven to be difficult [20]. For these reasons we focus here on the gradient descent method which is always applicable.

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 45.

1. *There are a family of convex differentiable functions f_t such that for all $\vec{x} \in L$, we have*

$$\|\vec{x}\| \leq f_t(\vec{x}) \leq \|\vec{x}\| + Kt$$

   and the Lipschitz constant of ∇f_t is $\frac{1}{t}$.

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

In other words we assume that there is a family of regularized objective functions f_t and a projection matrix $\mathbf{P}$, which we can think of as an approximation algorithm of this projection problem.

Now, let $\vec{x}^*$ be a minimizer of $\min_{\vec{x}\in L} \|\vec{x} - \vec{y}\|$. Since $\vec{x}^* \in L$, we have $\mathbf{P}\vec{x}^* = \vec{x}^*$ and hence

$$\begin{aligned} \|\mathbf{P}\vec{y} - \vec{y}\| &\leq \|\vec{y} - \vec{x}^*\| + \|\vec{x}^* - \mathbf{P}\vec{y}\| \\ &\leq \|\vec{y} - \vec{x}^*\| + \|\mathbf{P}\vec{x}^* - \mathbf{P}\vec{y}\| \\ &\leq (1 + \|\mathbf{P}\|) \min_{\vec{x}\in L} \|\vec{x} - \vec{y}\|. \end{aligned} \tag{4}$$

Therefore, the approximation ratio of $\mathbf{P}$ is $1 + \|\mathbf{P}\|$ and we see that our problem is to show that we can solve nonlinear projection using a decent linear projection matrix. Our algorithm for solving this problem is below.

NonlinearProjection
Input: a point $\vec{y}$ and $\text{OPT} = \min_{\vec{x}\in L} \|\vec{x} - \vec{y}\|$.
1. Let $\vec{y_0} = (\mathbf{I} - \mathbf{P})\,\vec{y}$ and $\vec{x_0} = 0$.
2. For $j = 0, \cdots$, until $2^{-j}\|\mathbf{P}\| \leq \frac{1}{2}$
3. If $2^{-j}\|\mathbf{P}\| > 1$, then let $t_j = \frac{2^{-(j+2)}\|\mathbf{P}\|\text{OPT}}{K}$ and $k_j = 3200\|\mathbf{P}\|^2 K$.
4. If $2^{-j}\|\mathbf{P}\| \leq 1$, then let $t_j = \frac{\varepsilon \text{OPT}}{2K}$ and $k_j = \frac{800\|\mathbf{P}\|^2 K}{\varepsilon^2}$.
5. Let $g_j(\vec{x}) = f_{t_j}(\mathbf{P}\vec{x} - \vec{y_j})$ and $\vec{x_0} = 0$.
6. For $i = 0, \cdots, k_j - 1$
7. $\vec{x}_{i+1} = \vec{x}_i - \frac{t}{\|\mathbf{P}\|_\infty^2}(\bigtriangledown g_j(\vec{x}_i))^\#$.
8. Let $\vec{y_{j+1}} = \vec{y_j} - \mathbf{P}\vec{x}_{k_j}$.
9. Output $\vec{y} - \vec{y_{\text{last}}}$.

Note that this algorithm and its proof are quite similar to Theorem 4 but modified to scale parameters over an outer loop. By changing the parameter t we can decrease the dependence of the initial error.[10]

Theorem 46. *Assume the conditions in Assumption 45 are satisfied. Let $\mathcal{T}$ be the time needed to compute $\mathbf{P}x$ and $\mathbf{P}^T x$ and $x^\#$. Then, **NonlinearProjection** outputs a vector $\vec{x}$ with $\|\vec{x}\| \leq (1 + \varepsilon)\min_{\vec{x}\in L} \|\vec{x} - \vec{y}\|$ and the algorithm takes time*

$$O\left(\|\mathbf{P}\|^2 K\,(\mathcal{T} + m)\left(\frac{1}{\varepsilon^2} + \log\|\mathbf{P}\|\right)\right).$$

[10] This is an idea that has been applied previously to solve linear programming problems [23].

Proof. We prove by induction on j that when $2^{-(j-1)}\|\mathbf{P}\| \geq 1$ we have $\|\vec{y_j}\| \leq (1 + 2^{-j}\|\mathbf{P}\|)\,\mathrm{OPT}$.

For the base case ($j = 0$), (46) shows that $\|\vec{y_0}\| \leq (1 + \|\mathbf{P}\|)\,\mathrm{OPT}$.

For the inductive case we assume that the assertion holds for some j. We start by bounding the corresponding R in Theorem 1 for g_j, which we denote R_j. Note that

$$g_j(\vec{x}_0) = f_{t_j}(-\vec{y_j}) \leq \|\vec{y_j}\| + Kt_j \leq (1 + 2^{-j}\|\mathbf{P}\|_\infty)\,\mathrm{OPT} + Kt_j.$$

Hence, the condition that $g_j(\vec{x}) \leq g_j(\vec{x}_0)$ implies that

$$\|\mathbf{P}\vec{x} - \vec{y_j}\| \leq (1 + 2^{-j}\|\mathbf{P}\|_\infty)\,\mathrm{OPT} + Kt_j.$$

Take any $\vec{y} \in X^*$, let $\vec{c} = \vec{x} - \mathbf{P}\vec{x} + \vec{y}$, and note that $\mathbf{P}\vec{c} = \mathbf{P}\vec{y}$ and therefore $\vec{c} \in X^*$. Using these facts, we can bound R_j as follows

$$\begin{aligned}
R_j &= \max_{\vec{x}\in\mathbb{R}^E \,:\, g_j(\vec{x}) \leq g_j(\vec{x}_0)} \left\{ \min_{\vec{x}^* \in X^*} \|\vec{x} - \vec{x}^*\| \right\} \\
&\leq \max_{\vec{x}\in\mathbb{R}^E \,:\, g_j(\vec{x}) \leq g_j(\vec{x}_0)} \|\vec{x} - \vec{c}\| \\
&\leq \max_{\vec{x}\in\mathbb{R}^E \,:\, g_j(\vec{x}) \leq g_j(\vec{x}_0)} \|\mathbf{P}\vec{x} - \mathbf{P}\vec{y}\| \\
&\leq \max_{\vec{x}\in\mathbb{R}^E \,:\, g_j(\vec{x}) \leq g_j(\vec{x}_0)} \|\mathbf{P}\vec{x}\| + \|\mathbf{P}\vec{y}\| \\
&\leq 2\|\vec{y_0}\| + \|\mathbf{P}\vec{x} - \vec{y_j}\| + \|\mathbf{P}\vec{y} - \vec{y_j}\| \\
&\leq 2\|\vec{y_0}\| + 2\|\mathbf{P}\vec{x} - \vec{y_j}\| \\
&\leq 4\,(1 + 2^{-j}\|\mathbf{P}\|_\infty)\,\mathrm{OPT} + 2Kt_j.
\end{aligned}$$

Similar to Lemma 3, the Lipschitz constant L_j of g_j is $\|\mathbf{P}\|^2/t_j$. Hence, Theorem 1 shows that

$$\begin{aligned}
g_j(\vec{x_{k_j}}) &\leq \min_{\vec{x}} g_j(\vec{x}) + \frac{2 \cdot L_j \cdot R_j^2}{k_j + 4} \\
&\leq \min_{\vec{x}} \|\mathbf{P}\vec{x} - \vec{y_j}\| + \frac{2 \cdot L_j \cdot R_j^2}{k_j + 4} + Kt_j
\end{aligned}$$

So, we have

$$\begin{aligned}
\|\mathbf{P}\vec{x}_{k_j} - \vec{y_j}\| &\leq f_{t_j}(\mathbf{P}\vec{x_{k_j}} - \vec{y_j}) \\
&\leq \mathrm{OPT} + Kt_j + \frac{2\|\mathbf{P}\|^2}{t_j(k_j + 4)} \left(4\,(1 + 2^{-j}\|\mathbf{P}\|)\,\mathrm{OPT} + 2Kt_j\right)^2.
\end{aligned}$$

When $2^{-j}\|\mathbf{P}\| > 1$, we have

$$t_j = \frac{2^{-(j+2)}\|\mathbf{P}\|\mathrm{OPT}}{K} \quad \text{and} \quad k_j = 3200\|\mathbf{P}\|^2 K$$

and hence

$$\|\vec{y_{j+1}}\| = \|\mathbf{P}\vec{x}_{k_j} - \vec{y_j}\| \leq (1 + 2^{-j-1}\|\mathbf{P}\|)\,\mathrm{OPT}.$$

When $2^{-j}\|\mathbf{P}\| \leq 1$, we have

$$t_j = \frac{\varepsilon\mathrm{OPT}}{2K} \quad \text{and} \quad k_j = \frac{800\|\mathbf{P}\|^2 K}{\varepsilon^2}$$

and hence

$$\|\vec{y_{j+1}}\| = \|\mathbf{P}\vec{x}_{k_j} - \vec{y_j}\| \leq (1 + \varepsilon)\,\mathrm{OPT}.$$

Since $\vec{y_{\mathrm{last}}}$ is $\vec{y}$ plus some vectors in L, $\vec{y} - \vec{y_{\mathrm{last}}} \in L$ and $\|\vec{y} - \vec{y_{\mathrm{last}}} - \vec{y}\| = \|\vec{y_{\mathrm{last}}}\| \leq (1 + \varepsilon)\,\mathrm{OPT}$.

□

7.2 Maximum Concurrent Flow

For an arbitrary set of demands $\vec{\chi}_i \in \mathbb{R}^V$ with $\sum_{v\in V} \vec{\chi}_i(v) = 0$ for $i = 1, \cdots, k$, we wish to solve the following *maximum concurrent flow* problem

$$\max_{\alpha\in R, \vec{f}\in\mathbb{R}^E} \alpha \text{ subject to } \mathbf{B}^T\vec{f}_i = \alpha\vec{\chi}_i \text{ and } \left\|\mathbf{U}^{-1}\sum_{i=1}^{k}|\vec{f}_i|\right\|_\infty \leq 1.$$

Similar to Section 3.2, it is equivalent to the problem

$$\min_{\vec{c}\in\mathbb{R}^{E\times[k]}} \left\| \sum_{i=1}^{k} |\vec{\alpha}_i + (\mathbf{Q}\vec{x})_i| \right\|_\infty$$

where $\mathbf{Q}$ is a projection matrix onto the subspace $\{\mathbf{B}^T\mathbf{U}\vec{x_i} = 0\}$, the output maximum concurrent flow is

$$\vec{f}_i(\vec{x}) = \mathbf{U}(\vec{\alpha}_i + (\mathbf{Q}\vec{x})_i)/\left\| \sum_{i=1}^{k} |\vec{\alpha}_i + (\mathbf{Q}\vec{x})_i| \right\|_\infty ,$$

and $\mathbf{U}\vec{\alpha_i}$ is any flow such that $\mathbf{B}^T\mathbf{U}\vec{\alpha_i} = \vec{\chi}_i$. In order to apply **NonlinearProjection**, we need to find a regularized norm and a good projection matrix. Let us define the norm

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

The problem is simply $\left\|\vec{\alpha} + \mathbf{Q}\vec{x}\right\|_{1;\infty}$ where $\mathbf{Q}$ is a projection matrix from $\mathbb{R}^{E\times[k]}$ to $\mathbb{R}^{E\times[k]}$ onto some subspace. Since each copy $\mathbb{R}^E$ is same, there is no reason that there is coupling in$\mathbf{Q}$ between different copies of $\mathbb{R}^E$. In the next lemma, we formalize this by the fact that any good projection matrix$\mathbf{P}$ onto the subspace $\{\mathbf{B}^T\mathbf{U}\vec{x} = 0\} \subset \mathbb{R}^E$ extends to a good projection $\mathbf{Q}$ onto the subspace $\{\mathbf{B}^T\mathbf{U}\vec{x_i} = 0\} \subset \mathbb{R}^{E\times[k]}$. Therefore, we can simply extends the good circulation projection $\mathbf{P}$ by formula $(\mathbf{Q}\vec{x})_i = \mathbf{P}\vec{x}_i$. Thus, the only last piece needed is a regularized $\left\|\cdot\right\|_{1;\infty}$. However, it turns out that smoothing via conjugate does not work well in this case because the dual space of $\left\|\cdot\right\|_{1;\infty}$ involves with $\left\|\cdot\right\|_\infty$, which is unfavorable for this kind of smoothing procedure. It can be proved that there is no such good regularized $\left\|\cdot\right\|_{1;\infty}$. Therefore, we could not do $O(m^{1+o(1)}k)$ using this approach, however, $O(m^{1+o(1)}k^2)$ is possible by using a bad regularized $\left\|\cdot\right\|_{1;\infty}$.

Lemma 47. *Let* $smaxL1_t(\vec{x}) = \text{smax}_t\left(\sum_{i=1}^{k}\sqrt{(x_i(e))^2 + t^2}\right)$. *It is a convex continuously differentiable function. The Lipschitz constant of* $\nabla smaxL1_t$ *is* $\frac{2}{t}$ *and*

$$\left\|\vec{x}\right\|_{1;\infty} - t\ln(2m) \leq smaxL1_t(\vec{x}) \leq \left\|\vec{x}\right\|_{1;\infty} + kt.$$

Proof. 1) It is clear that $\text{smax}L1_t$ is smooth.

2) $\text{smax}L1_t$ is convex.

Since smax_t is increasing for positive values and $\sqrt{x^2+t^2}$ is convex, for any $\vec{x}, \vec{y} \in \mathbb{R}^{E\times[k]}$ and $0 \leq t \leq 1$, we have

$$\begin{aligned}
\text{smax}L1_t(t\vec{x} + (1-t)\vec{y}) &= \text{smax}_t\left(\sum_{i=1}^{k}\sqrt{((tx_i + (1-t)y_i)(e))^2 + t^2}\right) \\
&\leq \text{smax}_t\left(\sum_{i=1}^{k}\left(t\sqrt{(x_i(e))^2 + t^2} + (1-t)\sqrt{(y_i(e))^2 + t^2}\right)\right) \\
&\leq t\text{smax}L1_t(\vec{x}) + (1-t)\text{smax}L1_t(\vec{y}).
\end{aligned}$$

3) The Lipschitz constant of $\nabla \text{smax}L1_t$ is $\frac{2}{t}$.
Note that smax_t (not its gradient) has Lipschitz constant 1 because for any $\vec{x}, \vec{y} \in \mathbb{R}^E$,

$$\begin{aligned}
& |\text{smax}_t(\vec{x}) - \text{smax}_t(\vec{y})| \\
= & \left| t \ln \left(\frac{\sum_{e\in E} \left(\exp(-\frac{x(e)}{t}) + \exp(\frac{x(e)}{t}) \right)}{2m} \right) - t \ln \left(\frac{\sum_{e\in E} \left(\exp(-\frac{y(e)}{t}) + \exp(\frac{y(e)}{t}) \right)}{2m} \right) \right| \\
= & t \left| \ln \left(\frac{\sum_{e\in E} \left(\exp(-\frac{x(e)}{t}) + \exp(\frac{x(e)}{t}) \right)}{\sum_{e\in E} \left(\exp(-\frac{y(e)}{t}) + \exp(\frac{y(e)}{t}) \right)} \right) \right| \\
\leq & t \left| \ln \left(\max_{e\in E} \exp(\frac{|x-y|(e)}{t}) \right) \right| \\
= & \left\| \vec{x} - \vec{y} \right\|_\infty .
\end{aligned}$$

Also, by the definition of derivative, for any $\vec{x}, \vec{y} \in \mathbb{R}^n$ and $t \in \mathbb{R}$, we have

$$\text{smax}_t(\vec{x} + t\vec{y}) - \text{smax}_t(\vec{x}) = t\left\langle \nabla \text{smax}_t(\vec{x}), \vec{y} \right\rangle + o(t).$$

and it implies $\left|\left\langle \nabla \text{smax}_t(\vec{x}), \vec{y} \right\rangle\right| \leq \left\| \vec{y} \right\|_\infty$ for arbitrary $\vec{y}$ and hence

$$\left\| \nabla \text{smax}_t(\vec{x}) \right\|_1 \leq 1. \tag{5}$$

For notational simplicity, let $s_1 = \text{smax}L1_t$, $s_2 = \text{smax}_t$ and $s_3(x) = \sqrt{x^2 + t^2}$. Thus, we have

$$s_1(\vec{x}) = s_2 \left(\sum_{i=1}^{k} s_3(x_i(e)) \right).$$

Now, we want to prove

$$\left\| \nabla s_1(\vec{x}) - \nabla s_1(\vec{y}) \right\|_{\infty;1} \leq \frac{2}{t} \left\| \vec{x} - \vec{y} \right\|_{1;\infty}.$$

Note that

$$\frac{\partial s_1(\vec{x})}{\partial x_i(e)} = \frac{\partial s_2}{\partial e} \left(\sum_i s_3(x_i(e)) \right) \frac{ds_3}{dx} \left(x_i(e) \right).$$

Hence, we have

$$\begin{aligned}
\left\| \nabla s_1(x) - \nabla s_1(y) \right\|_{\infty;1} = & \sum_e \max_i \left| \frac{\partial s_2}{\partial e} \left(\sum_j s_3(x_j(e)) \right) \frac{ds_3}{dx} \left(x_i(e) \right) - \frac{\partial s_2}{\partial e} \left(\sum_j s_3(y_j(e)) \right) \frac{ds_3}{dx} \left(y_i(e) \right) \right| \\
\leq & \sum_e \max_i \left| \frac{\partial s_2}{\partial e} \left(\sum_j s_3(x_j(e)) \right) \frac{ds_3}{dx} \left(x_i(e) \right) - \frac{\partial s_2}{\partial e} \left(\sum_j s_3(x_j(e)) \right) \frac{ds_3}{dx} \left(y_i(e) \right) \right| \\
& + \sum_e \max_i \left| \frac{\partial s_2}{\partial e} \left(\sum_j s_3(x_j(e)) \right) \frac{ds_3}{dx} \left(y_i(e) \right) - \frac{\partial s_2}{\partial e} \left(\sum_j s_3(y_j(e)) \right) \frac{ds_3}{dx} \left(y_i(e) \right) \right| \\
= & \sum_e \left| \frac{\partial s_2}{\partial e} \left(\sum_j s_3(x_j(e)) \right) \right| \max_i \left| \frac{ds_3}{dx} \left(x_i(e) \right) - \frac{ds_3}{dx} \left(y_i(e) \right) \right| \\
& + \sum_e \max_i \frac{ds_3}{dx} \left(y_i(e) \right) \left| \frac{\partial s_2}{\partial e} \left(\sum_j s_3(x_j(e)) \right) - \frac{\partial s_2}{\partial e} \left(\sum_j s_3(y_j(e)) \right) \right|.
\end{aligned}$$

Since s_3 has $\frac{1}{t}$-Lipschitz gradient, we have

$$\left|\frac{ds_3}{dx}(x) - \frac{ds_3}{dx}(y)\right| \leq \frac{1}{t}|x-y|.$$

By (5), we have

$$\sum_e \left|\frac{\partial s_2}{\partial e}(x(e))\right| \leq 1.$$

Hence, we have

$$\begin{aligned}
& \sum_e \max_i \left|\frac{ds_3}{dx}(x_i(e)) - \frac{ds_3}{dx}(y_i(e))\right| \left|\frac{\partial s_2}{\partial e}\left(\sum_i s_3(x_i(e))\right)\right| \\
\leq\ & \frac{1}{t}\max_{i,e}|x_i(e)-y_i(e)| \sum_e \left|\frac{\partial s_3}{\partial e}\left(\sum_i s_3(x_i(e))\right)\right| \\
=\ & \frac{1}{t}\|\vec{x}-\vec{y}\|_{1;\infty}.
\end{aligned}$$

Since s_3 is 1-Lipschitz, we have

$$\left|\frac{ds_3}{dx}\right| \leq 1.$$

Since s_2 has $\frac{1}{t}$-Lipschitz gradient in $\|\cdot\|_\infty$, we have

$$\sum_e \left|\frac{\partial s_2}{\partial e}(x) - \frac{\partial s_2}{\partial e}(y)\right| \leq \frac{1}{t}\|\vec{x}-\vec{y}\|_\infty.$$

Hence, we have

$$\begin{aligned}
& \sum_e \max_i \frac{ds_3}{dx}(y_i(e)) \left|\frac{\partial s_2}{\partial e}\left(\sum_j s_3(x_j(e))\right) - \frac{\partial s_2}{\partial e}\left(\sum_j s_3(y_j(e))\right)\right| \\
\leq\ & \sum_e \left|\frac{\partial s_2}{\partial e}\left(\sum_j s_3(x_j(e))\right) - \frac{\partial s_2}{\partial e}\left(\sum_j s_3(y_j(e))\right)\right| \\
\leq\ & \frac{1}{t}\|\sum_i s_3(x_i(e)) - \sum s_3(y_i(e))\|_\infty \\
\leq\ & \frac{1}{t}\|\sum_i |x_i(e)-y_i(e)|\ \| \\
=\ & \frac{1}{t}\|\vec{x}-\vec{y}\|_{1;\infty}
\end{aligned}$$

Therefore, we have

$$\|\nabla s_1(\vec{x}) - \nabla s_1(\vec{y})\|_{\infty;1} \leq \frac{2}{t}\|\vec{x}-\vec{y}\|_{1;\infty}.$$

4) Using the fact that

$$\|x(e)\| \leq \sum_{i=1}^{k} \sqrt{(x_i(e))^2 + t^2} \leq \|x(e)\|_1 + kt$$

and smax is 1-Lipschitz, we have

$$\|\vec{x}\|_{1;\infty} - t\ln(2m) \leq \mathrm{smax}L1_t(\vec{x}) \leq \|\vec{x}\|_{1;\infty} + kt.$$

□

The last thing needed is to check is that the # operator is easy to compute.

Lemma 48. *In $\|\cdot\|_{1;\infty}$, the # operator is given by an explicit formula*

$$\left(\vec{x}^{\#}\right)_i(e) = \begin{cases} ||\vec{x}||_{1;\infty}\text{sign}(x_i(e)) & \text{if } i \text{ is the smallest index such that } \min_j |x_j(e)| = x_i(e) \\ 0 & \text{otherwises} \end{cases}.$$

Proof. It can be proved by direct computation.

□

Now, all the conditions in the Assumption 45 are satisfiedTherefore, Theorem 46 and Theorem 19 gives us the following theorem:

Theorem 49. *Given an undirected capacitated graph $G = (V, E, \vec{\mu})$ with capacity ratio U. Assume $U = \mathsf{poly}(|V|)$. There is an algorithm finds an $(1-\varepsilon)$ approximate Maximum Concurrent Flow in time*

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

Proof. Let $\mathbf{A}$ be the oblivious routing algorithm given by Theorem 19. And we have $\rho(\mathbf{A}) \leq 2^{O\left(\sqrt{\log|V|\log\log|V|}\right)}$. Let us define the scaled circulation projection matrix $\mathbf{P} = \mathbf{I} - \mathbf{U}\mathbf{A}\mathbf{B}^T\mathbf{U}^{-1}$. Lemma 12 shows that $\|\mathbf{P}\|_\infty \leq 1 + 2^{O\left(\sqrt{\log|V|\log\log|V|}\right)}$.

Let the multi-commodity circulation projection matrix $\mathbf{Q} : \mathbb{R}^{E\times[k]} \to \mathbb{R}^{E\times[k]}$ defined by $(\mathbf{Q}\vec{x})_i = \mathbf{P}\vec{x_i}$. Note that the definition of $\|\mathbf{Q}\|_{1;\infty}$ is similar to $\rho(\mathbf{Q})$. By similar proof as Lemma 10, we have $\|\mathbf{Q}\|_{1;\infty} = \|\mathbf{P}\|_\infty$. Hence, we have $\|\mathbf{Q}\|_{1;\infty} \leq 1 + 2^{O\left(\sqrt{\log|V|\log\log|V|}\right)}$. Also, since $\mathbf{P}$ is a projection matrix on the subspace $\{\vec{x} \in \mathbb{R}^E : \mathbf{B}^T\mathbf{U}\vec{x} = 0\}$, $\mathbf{Q}$ is a projection matrix on the subspace $\{\vec{x} \in \mathbb{R}^{E\times[k]} : \mathbf{B}^T\mathbf{U}\vec{x_i} = 0\}$.

By Lemma 47, the function $\text{smax}L1_t(\vec{x})$ is a convex continuously differentiable function such that the Lipschitz constant of $\nabla\text{smax}L1_t$ is $\frac{2}{t}$ and

$$\|\vec{x}\|_{1;\infty} - t\ln(2m) \leq \text{smax}L1_t(\vec{x}) \leq \|\vec{x}\|_{1;\infty} + kt.$$

Given an arbitrary set of demands $\vec{\chi_i} \in \mathbb{R}^V$, we find a vector $\vec{y}$ such that

$$\mathbf{B}^T\mathbf{U}\vec{y} = -\vec{\chi_i}.$$

Then, we use the **NonlinearProjection** to solve

$$\min_{\mathbf{B}^T\mathbf{U}\vec{x}=0} \|\vec{x} - \vec{y}\|_{1;\infty}$$

using a family of functions $\text{smax}L1_t(\vec{x}) + t\ln(2n)$ and the projection matrix $\mathbf{Q}$. Since each iteration involves calculation of gradients and # operator, it takes $O(mk)$ each iteration. And it takes $\tilde{O}\left(\|\mathbf{Q}\|^2_{1;\infty}K/\varepsilon^2\right)$ iterations in total where $K = k + \ln(2m)$. In total, it **NonlinearProjection** outputs a $(1+\varepsilon)$ approximate minimizer $\vec{x}$ in time

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

And it gives a $(1-\varepsilon)$ approximate maximum concurrent flow $\vec{f_i}$ by the formula

$$\vec{f_i} = \mathbf{U}(\vec{x_i} - \vec{y_i})/\|\vec{x} - \vec{y}\|_{1;\infty}.$$

□

8 Acknowledgements

We thank Jonah Sherman for agreeing to coordinate submissions and we thank Satish Rao, Jonah Sherman, Daniel Spielman, Shang-Hua Teng. This work was partially supported by NSF awards 0843915 and 1111109 and a NSF Graduate Research Fellowship (grant no. 1122374).

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows*. Elsevier North-Holland, Inc., New York, NY, USA, 1989.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M. R. Reddy. Applications of network optimization. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 1–75. North-Holland, 1995.

[3] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. Available at http://www.cs.princeton.edu/~arora/pubs/MWsurvey.pdf.

[4] 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.

[5] 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.

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

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

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

[20] Arkadiui Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

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

[22] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2003.

[23] Yurii Nesterov. Rounding of convex sets and efficient gradient methods for linear programming problems. *Available at SSRN 965658*, 2004.

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

[25] 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.

[26] Alexander Schrijver. *Combinatorial Optimization, Volume A*. Number 24 in Algorithms and Combinatorics. Springer, 2003.

[27] Jonah Sherman. Breaking the multicommodity flow barrier for $O\left(\sqrt{\log n}\right)$-approximations to sparsest cut. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science*, 2009.

[28] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 114–122, New York, NY, USA, 1981. ACM.

[29] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 563–568, New York, NY, USA, 2008. ACM.

[30] 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.

[31] 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.

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

A Some Facts about Norm and Functions with Lipschitz Gradient

In this section, we present some basic fact used in this paper about norm and dual norm. Also, we presented some lemmas about convex functions with Lipschitz gradient. See [6, 22] for comprehensive discussion.

A.1 Norms

Fact 50.

$$\vec{x} = 0 \quad \Leftrightarrow \vec{x}^{\#} = 0.$$

Proof. If $\vec{x} = 0$ then $\forall \vec{s} \neq 0$ we have $\langle \vec{x}, \vec{s} \rangle - \frac{1}{2}\|\vec{s}\|^2 < 0$ but $\langle \vec{x}, \vec{x} \rangle - \frac{1}{2}\|\vec{x}\|^2 = 0$. So we have $\vec{x}^{\#} = 0$. If $\vec{x} \neq 0$ then let $\vec{s} = \frac{\langle \vec{x}, \vec{x} \rangle}{\|\vec{x}\|^2}\vec{x}$ with this choice we have $\langle \vec{x}, \vec{s} \rangle - \frac{1}{2}\|\vec{s}\|^2 = \frac{1}{2}\frac{\langle \vec{x}, \vec{x} \rangle^2}{\|\vec{x}\|^2} > 0$. However, for $\vec{s} = 0$ we have that $\langle \vec{x}, \vec{s} \rangle - \frac{1}{2}\|\vec{s}\|^2 = 0$ therefore we have $\vec{x}^{\#} \neq 0$. □

Fact 51.

$$\forall \vec{x} \in \mathbb{R}^n \quad : \quad \langle x, x^{\#} \rangle = \|x^{\#}\|^2.$$

Proof. If $\vec{x} = 0$ then $\vec{x}^{\#} = 0$ by Claim 50 and we have the result. Otherwise, again by claim 50 we know that $\vec{x}^{\#} \neq 0$ and therefore by the definition of $\vec{x}^{\#}$ we have

$$1 = \arg\max_{c \in \mathbb{R}} \langle x, c \cdot x^{\#} \rangle - \frac{1}{2}\|c \cdot x^{\#}\|^2 = \arg\max_{c \in \mathbb{R}} c \cdot \langle x, x^{\#} \rangle - \frac{c^2}{2}\|x^{\#}\|^2$$

Setting the derivative of with respect to c to 0 we get that $1 = c = \frac{\langle \vec{x}, \vec{x}^{\#} \rangle}{\|\vec{x}^{\#}\|^2}$. □

Fact 52.

$$\forall \vec{x} \in \mathbb{R}^n \quad : \quad \|\vec{x}\|^* = \|\vec{x}^{\#}\|.$$

Proof. Note that if $\vec{x} = 0$ then the claim follows from Claim (50) otherwise we have

$$\|\vec{x}\|^* = \max_{\|y\| \leq 1} \langle x, y \rangle = \max_{\|y\| = 1} \langle x, y \rangle \leq \max_{y \in \mathbb{R}^n} \frac{\langle x, y \rangle}{\|y\|}$$

From this it is clear that $\|\vec{x}\|^* \geq \|\vec{x}^{\#}\|$. To see the other direction consider a $\vec{y}$ that maximizes the above and let $\vec{z} = \frac{\langle \vec{x}, \vec{y} \rangle}{\|\vec{y}\|^2}\vec{y}$

$$\langle \vec{x}, \vec{z} \rangle - \frac{1}{2}\|\vec{z}\|^2 \leq \langle \vec{x}, \vec{x}^{\#} \rangle - \frac{1}{2}\|\vec{x}^{\#}\|^2$$

and therefore

$$\|\vec{x}\|^{*2} - \frac{1}{2}\|\vec{x}\|^{*2} \leq \frac{1}{2}\|\vec{x}^{\#}\|^2$$

□

Fact 53. *[Cauchy Shcwarz]*

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \quad : \quad \langle \vec{y}, \vec{x} \rangle \leq \|\vec{y}\|^* \|\vec{x}\|.$$

Proof. By the definition of dual norm, for all $\|\vec{x}\| = 1$, we have $\langle \vec{y}, \vec{x} \rangle \leq \|\vec{y}\|^*$. Hence, it follows by linearity of both side. □

A.2 Functions with Lipschitz Gradient

Lemma 54. *Let f be a continuously differentiable convex function. Then, the following are equivalence:*

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \ : \ \left\| \bigtriangledown f(\vec{x}) - \bigtriangledown f(\vec{y}) \right\|^* \leq L \cdot \left\| \vec{x} - \vec{y} \right\|$$

and

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \ : \ f(\vec{x}) \leq f(\vec{y}) + \left\langle \bigtriangledown f(\vec{y}), \vec{x} - \vec{y} \right\rangle + \frac{L}{2} \left\| \vec{x} - \vec{y} \right\|^2.$$

For any such f and any $\vec{x} \in \mathbb{R}^n$, we have

$$f(\vec{x} - \frac{1}{L} \nabla f(\vec{x})^{\#}) \leq f(\vec{x}) - \frac{1}{2L} \left\| \bigtriangledown f(\vec{x}) \right\|^{*2}.$$

Proof. From the first condition, we have

$$\begin{aligned}
f(\vec{y}) &= f(\vec{x}) + \int_0^1 \frac{d}{dt} f(\vec{x} + t(\vec{y} - \vec{x}))dt \\
&= f(\vec{x}) + \int_0^1 \left\langle \nabla f(\vec{x} + t(\vec{y} - \vec{x})), \vec{y} - \vec{x} \right\rangle dt \\
&= f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \int_0^1 \left\langle \nabla f(\vec{x} + t(\vec{y} - \vec{x})) - \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle dt \\
&\leq f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \int_0^1 \left\| \nabla f(\vec{x} + t(\vec{y} - \vec{x})) - \nabla f(\vec{x}) \right\|^* \left\| \vec{y} - \vec{x} \right\| dt \\
&\leq f(x) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \int_0^1 Lt \left\| \vec{y} - \vec{x} \right\|^2 dt \\
&= f(x) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \frac{L}{2} \left\| \vec{y} - \vec{x} \right\|^2.
\end{aligned}$$

Given the second condition. For any $\vec{x} \in \mathbb{R}^n$. let $\phi_{\vec{x}}(\vec{y}) = f(\vec{y}) - \left\langle \nabla f(\vec{x}), \vec{y} \right\rangle$. From the convexity of f, for any $\vec{y} \in \mathbb{R}^n$

$$f(\vec{y}) - f(\vec{x}) \geq \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle.$$

Hence, $\vec{x}$ is a minimizer of $\phi_{\vec{x}}$. Hence, we have

$$\begin{aligned}
\phi_{\vec{x}}(\vec{x}) &\leq \phi_{\vec{x}}(\vec{y} - \frac{1}{L} \nabla \phi_{\vec{x}}(\vec{y})^{\#}) \\
&\leq \phi_{\vec{x}}(\vec{y}) - \left\langle \nabla \phi_{\vec{x}}(\vec{y}), \frac{1}{L} \nabla \phi_{\vec{x}}(\vec{y})^{\#} \right\rangle + \frac{L}{2} \left\| \frac{1}{L} \nabla \phi_{\vec{x}}(\vec{y})^{\#} \right\|^2 \qquad \text{(First part of this lemma)} \\
&= \phi_{\vec{x}}(\vec{y}) - \frac{1}{2L} \left\| \nabla \phi_{\vec{x}}(\vec{y})^{\#} \right\|^2 \\
&= \phi_{\vec{x}}(\vec{y}) - \frac{1}{2L} \left(\left\| \nabla \phi_{\vec{x}}(\vec{y}) \right\|^* \right)^2.
\end{aligned}$$

Hence,

$$f(\vec{y}) \geq f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \frac{1}{2L} \left(\left\| \nabla f(\vec{y}) - \nabla f(\vec{x}) \right\|^* \right)^2.$$

Adding up this inequality with $\vec{x}$ and $\vec{y}$ interchanged, we have

$$\begin{aligned}
\frac{1}{L} \left(\left\| \nabla f(\vec{y}) - \nabla f(\vec{x}) \right\|^* \right)^2 &\leq \left\langle \nabla f(\vec{y}) - \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle \\
&\leq \left\| \bigtriangledown f(\vec{y}) - \bigtriangledown f(\vec{x}) \right\|^* \left\| \vec{y} - \vec{x} \right\|.
\end{aligned}$$

The last inequality follows from similar proof in above for $\phi_{\vec{x}}$. □

The next lemma relate the Hessian of function with the Lipschitz parameter L and this lemma gives us a easy way to compute L.

Lemma 55. *Let f be a twice differentiable function such that for any $\vec{x}, \vec{y} \in \mathbb{R}^n$*

$$0 \leq \vec{y}^T \left(\nabla^2 f(\vec{x})\right) \vec{y} \leq L||\vec{y}||^2.$$

Then, f is convex and the gradient of f is Lipschitz continuous with Lipschitz parameter L.

Proof. Similarly to Lemma 54, we have

$$\begin{aligned} f(\vec{y}) &= f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \int_0^1 \left\langle \nabla f(\vec{x} + t(\vec{y} - \vec{x})) - \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle dt \\ &= f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \int_0^1 t(\vec{y} - \vec{x})^T \nabla^2 f(\vec{x} + \theta_t(\vec{y} - \vec{x}))(\vec{y} - \vec{x}) dt \end{aligned}$$

where the $0 \leq \theta_t \leq t$ comes from mean value theorem. By the assumption, we have

$$\begin{aligned} f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle &\leq f(\vec{y}) \\ &\leq f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \int_0^1 tL\left\|\vec{y} - \vec{x}\right\|^2 dt \\ &\leq f(\vec{x}) + \left\langle \nabla f(\vec{x}), \vec{y} - \vec{x} \right\rangle + \frac{L}{2}\left\|\vec{y} - \vec{x}\right\|^2. \end{aligned}$$

And the conclusion follows from Lemma 54. □