A Note on Cut-Approximators and Approximating Undirected Max Flows

Richard Peng
M.I.T.
rpeng@mit.edu

December 1, 2014

Abstract

We show a closer algorithmic connection between constructing cut-approximating hierarchical tree decompositions and computing approximate maximum flows in undirected graphs. This leads to the first $O(m \text{polylog}(n))$ time algorithms for both problems.

1 Introduction

The maximum flow problem asks for a flow of minimum congestion that meets a demand. Sherman \cite{She13a} and Kelner, Lee, Orecchia, and Sidford \cite{KLOS14} gave algorithms for computing approximate undirected maximum flows in $O(m^{1+o(1)} \epsilon^{-2})$ time using congestion-approximators and oblivious routing schemes respectively. These algorithms have in turn been used by Räcke, Shah and Täubig \cite{RST14} to construct higher quality cut-preserving hierarchical tree decompositions for undirected graphs in $O(m^{1+o(1)})$ time. A natural question from these results is whether the $m^{o(1)}$ factor can be improved.

In this note, we show that combining these results leads to faster algorithms. Our approach is based on invoking these algorithms recursively while reducing problem sizes using ultra-sparsifiers. We believe further runtime improvements based on this approach are likely.

To simplify presentation, we state the exponents on the log factors as constants $e_i$. However, as the control of constants is important for recursive calls, we will track them explicitly as $c_i$ instead of using the big-O notation. We start by formally describing previous works.
2 Background

Our presentation follows standard notations. A flow \( f \) in a graph \( G \) meets demands \( b \) if for all vertices \( v \), the total amount of flow entering/leaving \( v \) is \( b_v \). For a set of capacities \( u \), the congestion of the flow \( f \) is the maximum of \( f_e/u_e \) over all edges. A cut is defined by a subset of vertices \( S \): its demand is the total demand of vertices in \( S \), and its capacity is the total capacity of edges leaving \( S \). The ratio between demand and capacity of any cut \( S \) is a lower bound for the minimum congestion needed to route the demands. The maxflow-mincut theorem states that the minimum congestion is in fact equal to the maximum demand/capacity ratio over all cuts \( S \).

Congestion approximators are structures that approximate the minimum congestion required to route any demand on a graph.

**Definition 2.1** (Definition 1.1. in [She13b]). An \( \alpha \)-congestion-approximator of \( G \) is a matrix \( R \) such that for any demand vector \( b \),

\[
\|Rb\|_\infty \leq \text{opt}(b) \leq \alpha \|Rb\|_\infty,
\]

where \( \text{opt}(b) \) is the minimum congestion required to route the demands \( b \) in \( G \).

The approximate undirected maximum flow problem can be viewed as finding a flow/cut pair whose objectives are close. Formally, given graph \( G \), demands \( b \), and error parameter \( \epsilon > 0 \), the goal is to find a flow/cut pair whose congestion and demand/capacity ratios are within a factor of \( 1 + \epsilon \) of each other. We will refer to such solutions as \((1 + \epsilon)\)-approximate flow/cut solutions for the demands \( b \). Sherman [She13a] used congestion-approximators to give algorithms for computing these solutions.

**Theorem 2.2** (Theorem 1.2. from [She13b]). There is a routine Approximator-MaxFlow that, given demands \( b \) and access to an \( \alpha \)-congestion-approximator \( R \), makes \( c_1 \alpha^2 \log e n \epsilon^{-2} \) iterations and returns an \((1 + \epsilon)\)-approximate flow/cut solution for these demands. Each iteration takes \( c_1 m \) time, plus a multiplication by \( R \) and \( R^T \).

An algorithm based on oblivious routing schemes was shown by Kelner, Lee, Orecchia, and Sidford [KLOST14]. Such schemes return flows whose congestions are at most \( \alpha \text{opt}(b) \), instead of just an approximation to \( \text{opt}(b) \). Räcke, Shah and Täubig [RST14] showed the following result on efficiently constructing such schemes:

**Theorem 2.3** (Theorem 4.1 from [RST14], paraphrased). Given a graph \( G \) with \( n \) vertices and \( m \) edges, we can construct a tree \( T \) that corresponds to an oblivious routing scheme with competitive ratio \( c_2 \log^{e_2} n \) using a sequence of maximum flow calls on graphs of size \( m_1, \ldots, m_k \) with error \( c_3 \log^{-e_3} n \) such that

\[
\sum_{i=1}^{k} m_i \leq c_4 m \log^{e_4} n.
\]
Such trees can be used as congestion approximators by summing up the demands across each edge of the tree. This was observed in the second paragraph of the abstract of the paper by Räcke et al. \cite{RST14}.

**Corollary 2.4.** There is a routine \textsc{CongestionApproximator} that takes a graph $G$, constructs a $c_2 \log^{e_2} n$-congestion-approximator $R$ by computing a series of approximate undirected maximum flows with error $c_3 \log^{-e_3} n$ on graphs of sizes $m_1, \ldots, m_k$ such that

$$\sum_{i=1}^{k} m_i \leq c_4 m \log^{e_4} n.$$ 

Furthermore, matrix-vector multiplications in $R$ and $R^T$ can be performed in $c_2 n$ time.

### 3 Recursive Algorithm

The goal of this note is to connect these routines recursively using ultra-sparsifiers. The following construction of ultra-sparsifiers can be obtained from \cite{KMP14} and \cite{AN12}.

**Theorem 3.1.** There is a routine \textsc{Ultra-Sparsify} that takes a graph $G = (V, E, u_G)$ with $n$ vertices and $m$ edges, and any parameter $\kappa > 1$, returns in $O(m \log n \log \log n)$ time a graph $H = (V, E_H, u_H)$ with $n - 1 + mc_5 \log^{e_5} n / \kappa$ edges such that the cut-structure of $H$ is within a factor of $\kappa$ of the cut-structure of $G$.

Note that since we only need to preserve cuts, the Spielman-Teng construction of ultra-sparsifiers \cite{ST14} using cut-sparsifiers \cite{BK96} also leads to a similar bound.

Ultra-sparsifiers are graphs with low Euler characteristics, and therefore leads to reductions in problem sizes. Such reductions play central roles in algorithms that utilize ultra-sparsifiers \cite{ST14, Mad10a, KMP14, Pen13, She13a, KLOS14}.

**Lemma 3.2** (Lemma 5.8 of \cite{Mad10b}, paraphrased). There are routines \textsc{Reduce} and \textsc{Convert} such that when given a graph $H$ with $n$ vertices and $m = n - 1 + m'$ edges, $H' = \text{Reduce}(H)$ is a graph such that:

1. $H'$ has at most $c_6 m'$ edges, and
2. given an $\alpha$-congestion-approximator $R'$ for $H'$, $R = \text{Convert}(H, H', R')$

   (a) is an $c_6 \alpha$-congestion-approximator for $H$, and

   (b) matrix-vector multiplications involving $R$ and $R^T$ can each be performed using one matrix-vector multiplication involving $R'$ and $R'^T$ plus an overhead of $c_6 m$ time.

Using these two steps as size reductions leads to a recursive algorithm. Its pseudocode is given in Figure 1.
**Theorem 3.3.** When \( \epsilon = c_3 \log^{-c_2} n \), \textsc{RecursiveApproxMaxFlow} returns an \((1 + \epsilon)\)-approximate flow/cut solution in time 

\[
O(m \log^{e_1+2(e_2+e_3+e_4+e_5)} n).
\]

**Proof.** The guarantees of \textsc{CongestionApproximator} from Corollary 2.4 gives that \( R' \) is a \( c_2 \log^{e_2} \)-congestion-approximator for \( H \).

Lemma 3.2 then gives that \( R \) as returned by \textsc{Convert} on Line 4 of \textsc{RecursiveApproxMaxFlow} is a \( c_6 \cdot c_2 \log^{e_2} \)-congestion-approximator for \( H \).

Combining this with the the guarantees of Theorem 2.1 gives that \( R \) is an congestion-approximator for \( G \) with quality:

\[
c_6 \cdot c_2 \log^{e_2} \cdot \kappa = 2c_2c_4c_5c_6^2 \log^{e_2+e_4+e_5} n.
\]

Furthermore, Corollary 2.4 and Lemma 3.2 gives that the cost of matrix-vector multiplications involving \( R \) and \( R^T \) can be bounded by \( c_2n + c_6m \). Theorem 2.2 then gives that the cost of the call to \textsc{ApproximatorMaxFlow} is at most

\[
c_1 \alpha^2 \log^{e_1} n \epsilon^{-2} = c_1 \left(2c_2c_4c_5c_6^2 \log^{e_2+e_4+e_5} n \right)^2 \log^{e_1} n \left(c_3 \log^{-c_2} n \right)^{-2} \left(2c_2n + c_6m \right)
\]

\[
\leq 4c_1c_2c_3^{-2}c_4^{2}c_5c_6^{3} m \log^{e_1+2(e_2+e_3+e_4+e_5)} n.
\]

We will denote the constant \( 4c_1c_2c_3^{-2}c_4^{2}c_5c_6^{3} \) as \( C \) from here on.

Let the running time of \textsc{RecursiveApproxMaxFlow}(\( G, c_3 \log^{-c_2} n, b \)) on \( G \) with \( m \) edges be \( T(m) \). The above calculations give the following recurrence:

\[
T(m) \leq \sum_{i=1}^{k} T(m_k) + Cm \log^{e_1+2(e_2+e_3+e_4+e_5)} n,
\]
where

\[ \sum_{i=1}^{k} m_i \leq c_4 \log^{e_4} |E(H')|. \]

To bound the size of \( H' \), note that ULTRASPARSIFY is invoked on Line 1 of RECURSIVEAPPROXMAXFLOW with \( \kappa = 2c_4c_5c_6 \log^{e_4+e_5} n \). By Theorem 3.1 we have:

\[ |E(H)| \leq n - 1 + mc_5 \log^{e_5} n / \kappa = n - 1 + \frac{m}{2c_4c_6 \log^{e_4} n}. \]

Lemma 3.2 then gives that the number of edges in \( H' \) is at most:

\[ \frac{m}{2c_4 \log^{e_4} n}, \]

which implies

\[ \sum_{i=1}^{k} m_i \leq \frac{m}{2}. \]

So the total sizes of the recursive calls to RECURSIVEAPPROXMAXFLOW is geometrically decreasing. We can then show by guess-and-check that

\[ T(m) \leq 2Cm \log^{e_1+2(e_2+e_3+e_4+e_5)} n. \]

Formally this process is via induction, but it boils down to directly substituting in the desired complexity, and invoking the fact that \( \sum_i m_i \leq \frac{m}{2} \). This gives:

\[
\begin{align*}
T & \leq \sum_{i=1}^{k} T(m_k) + Cm \log^{e_1+2(e_2+e_3+e_4+e_5)} n \\
& \leq \sum_i Cm_i \log^{2(e_1+e_2+e_3+e_4)+e_4} n + 2Cm \log^{e_1+2(e_2+e_3+e_4+e_5)} n \\
& = \left( 2 \sum_i m_i + m \right) C \log^{e_1+2(e_2+e_3+e_4+e_5)} n \\
& \leq 2Cm \log^{e_1+2(e_2+e_3+e_4+e_5)} n.
\end{align*}
\]

Invoking this algorithm in Theorem 2.3 also gives an \( O(m \text{poly}(\log(n))) \) time algorithm for constructing hierarchical tree decompositions.

**Corollary 3.4.** Given a graph \( G \), we can construct a tree that corresponds to an oblivious routing scheme with competitive ratio \( O(\log^2 n) \) in \( O(m \log^{e_1+2(e_2+e_3+e_4)+3e_4} n) \) time.

Modifying the value of \( \epsilon \) at the first level of the recursion gives a similar runtime bound for smaller values of \( \epsilon \):

**Corollary 3.5.** If \( \epsilon \leq c_3 \log^{-e_3} n \), RECURSIVEAPPROXMAXFLOW returns an \((1 + \epsilon)\)-approximate flow/cut solution in \( O(m \log^{e_1+2(e_2+e_4+e_5)} n \epsilon^{-2}) \) time.
References

[AN12] Ittai Abraham and Ofer Neiman. Using petal decompositions to build a low stretch spanning tree. In Proceedings of the 44th symposium on Theory of Computing, STOC ’12, pages 395–406, New York, NY, USA, 2012. ACM.

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, STOC ’96, pages 47–55, New York, NY, USA, 1996. ACM.

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 217–226, 2014.

[KMP14] I. Koutis, G. Miller, and R. Peng. Approaching optimality for solving sdd linear systems. SIAM Journal on Computing, 43(1):337–354, 2014.

[Mad10a] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In FOCS, pages 245–254. IEEE Computer Society, 2010.

[Mad10b] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. CoRR, abs/1008.1975, 2010.

[Pen13] Richard Peng. Algorithm Design Using Spectral Graph Theory. PhD thesis, Carnegie Mellon University, Pittsburgh, August 2013. CMU CS Tech Report CMU-CS-13-121.

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 227–238, 2014.

[She13a] Jonah Sherman. Nearly maximum flows in nearly linear time. In 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, pages 263–269, 2013.

[She13b] Jonah Sherman. Nearly maximum flows in nearly linear time. CoRR, abs/1304.2077, 2013.

[ST14] D. Spielman and S. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. SIAM Journal on Matrix Analysis and Applications, 35(3):835–885, 2014.