Fast Approximation Algorithms for Cut-based Problems in Undirected Graphs (Extended Abstract)

Aleksander Mądry*
Massachusetts Institute of Technology
Cambridge, MA
madry@mit.edu

Abstract— We present a general method of designing fast approximation algorithms for cut-based minimization problems in undirected graphs. In particular, we develop a technique that given any such problem that can be approximated quickly on trees, allows approximating it almost as quickly on general graphs while only losing a poly-logarithmic factor in the approximation guarantee.

To illustrate the applicability of our paradigm, we focus our attention on the undirected sparsest cut problem with general demands and the balanced separator problem. By a simple use of our framework, we obtain poly-logarithmic approximation algorithms for these problems that run in time close to linear.

The main tool behind our result is an efficient procedure that decomposes general graphs into simpler ones while approximately preserving the cut-flow structure. This decomposition is inspired by the cut-based graph decomposition of Räcke that was developed in the context of oblivious routing schemes, as well as, by the construction of the ultrasparsifiers due to Spielman and Teng that was employed to preconditioning symmetric diagonally-dominant matrices.

Keywords-cut-based problems; graph partitioning; generalized sparsest cut; balanced separator; fast approximation algorithms; graph decomposition;

1. Introduction

Cut-based graph problems are ubiquitous in optimization. They have been extensively studied – both from theoretical and applied perspective – in the context of flow problems, theory of Markov chains, geometric embeddings, clustering, community detection, VLSI layout design, and as a basic primitive for divide-and-conquer approaches (cf. [1]). For the sake of concreteness, in this paper we define an optimization problem $\mathcal{P}$ to be an *(undirected) cut-based (minimization) problem* if its every instance $P \in \mathcal{P}$ can be cast as a task of finding – for a given input undirected graph $G = (V, E, u)$ – a cut C^* such that $C^* = \text{argmin}_{\emptyset \neq C \subset V} u(C) f_P(C)$, where $u(C)$ is the capacity of the cut C in G and f_P is a non-negative function that depends on P, but not on the graph G. It is not hard to see that a large number of graph problems fits this definition. For illustrative purposes, in this paper we focus on two of them that are fundamental in graph partitioning: the generalized sparsest cut problem and the balanced separator problem.

In the *generalized sparsest cut* problem, we are given a graph $G = (V, E, u)$ and a demand graph $D = (V, E_D, d)$. Our task is to find a cut C^* that minimizes the *(generalized) sparsity* $u(C)/d(C)$ among all the cuts C of G. Here, $d(C)$ is the total demand $\sum_{e \in E_D(C)} d(e)$ of the demand edges $E_D(C)$ of D cut by the cut C. Casted in our terminology, the problem is to find a cut C^* being $\text{argmin}_C u(C) f_P(C)$ with $f_P(C) := 1/d(C)$.

An important problem that is related to the generalized sparsest cut problem is the *sparsest cut* problem. In this problem one aims at finding a cut C^* that minimizes the *sparsity* $u(C)/\min\{|C|, |\overline{C}|\}$ among all the cuts C of G. Note that if we considered an instance of generalized sparsest cut problem corresponding to the demand graph D being complete and each demand being $1/|V|$ (this special case is sometimes called the *uniform sparsest cut* problem) then the corresponding generalized sparsity of every cut C is within factor of two of its sparsity. Therefore, up to this constant factor, one can view the sparsest cut problem as a special case of the generalized sparsest cut problem.

The *balanced separator* problem corresponds to a task of finding a cut that minimizes the sparsity among all the *c-balanced cuts* C in G, i.e. among all the cuts C with $\min\{|C|, |\overline{C}|\} \geq c|V|$, for some specified constant $c > 0$ called the *balance constant* of the instance. In our framework, the function $f_P(C)$ corresponding to this problem is equal to $1/\min\{|C|, |\overline{C}|\}$ if $\min\{|C|, |\overline{C}|\} \geq c|V|$ and equal to $+\infty$ otherwise.

1.1. Previous Work on Graph Partitioning

The captured by the above-mentioned problems task of partitioning a graph into relatively large pieces while minimizing the number of edges cut is a fundamental combinatorial problem (see [1]) with a long history of

*Supported by NSF contract CCF-0829878 and by ONR grant N00014-05-1-0148.

theoretical and applied investigation. Since most of the graph partitioning problems – in particular, the ones we consider – are NP-hard, one needs to settle for approximation algorithms. In case of the sparsest cut problem and the balanced separator problem, there were two main types of approaches to approximating them.

First type of approach was based on spectral methods. Inspired by Cheeger's inequality [2] discovered in the context of Riemannian manifolds, Alon and Milman [3] transplanted it to discrete setting and presented an algorithm for the sparsest cut problem that can be implemented to run in nearly-linear time (cf. [4]), but its approximation ratio depends on the conductance[1] $\Phi(G)$ of the graph and can be as large as $\Omega(n)$ in the worst case. Later, Spielman and Teng [5] (see also [6] and [7] for a follow-up work) used this approach to design an algorithm for the balanced separator problem. However, both the running time and the approximation guarantee still depend on the conductance of the graph, leading to an $\Omega(n)$ approximation ratio and large running time in the worst case.

The second type of approach relies on flow methods. It was started by Leighton and Rao [8] who used linear programming relaxation of the maximum concurrent flow problem to establish first poly-logarithmic approximation algorithms for many graph partitioning problems. In particular, an $O(\log n)$-approximation for the sparsest cut and the balanced separator problems. Both these algorithms can be implemented to run in $\widetilde{O}(n^2)$ time.

These spectral and flow-based approaches were combined in a seminal result of Arora, Rao, and Vazirani [9] to give an $O(\sqrt{\log n})$-approximation for the sparsest cut and the balanced separator problems.

In case of the generalized sparsest cut problem, Leighton and Rao [8] presented a $O(\log r)$-approximation algorithm, where r is the number of vertices of the demand graph that are endpoints of some demand edge. Subsequently, Chawla, Gupta, and Räcke [10] extended the techniques from [9] to obtain an $O(\log^{3/4} r)$-approximation. This approximation ratio was later improved by Arora, Lee, and Naor [11] to $O(\sqrt{\log r}\log\log r)$.

1.2. Fast Approximation Algorithms for Graph Partitioning Problems

More recently, a lot of effort was put into designing algorithms for the graph partitioning problems that are very efficient while still having poly-logarithmic approximation guarantee. Arora, Hazan, and Kale [12] combined the concept of expander flows that was introduced in [9] together with multicommodity flow computations to obtain $O(\sqrt{\log n})$-approximation algorithms for the sparsest cut and the balanced separator problems that run in $\widetilde{O}(n^2)$ time.

Subsequently, Khandekar, Rao, and Vazirani [13] designed a primal-dual framework for graph partitioning algorithms and used it to achieve $O(\log^2 n)$-approximations for the same two problems running in $\widetilde{O}(m + n^{3/2})$ time needed to perform graph sparsification of Benczúr and Karger [14] followed by a poly-logarithmic number of maximum single-commodity flow computations. In [15], Arora and Kale introduced a general approach to approximately solving semi-definite programs which, in particular, led to $O(\log n)$-approximation algorithms for the sparsest cut and the balanced separator problems that also run in $\widetilde{O}(m + n^{3/2})$ time. Later, Orecchia, Schulman, Vazirani, and Vishnoi [16] obtained the same approximation ratio and running time as [15] by extending the framework from [13]. Recently, Sherman [17] presented an algorithm that for any $\varepsilon > 0$ works in $\widetilde{O}(m + n^{3/2+\varepsilon})$ time – corresponding to performing sparsification and $\widetilde{O}(n^{\varepsilon})$ maximum single-commodity flow computations – and achieves an approximation ratio of $O(\sqrt{\log n/\varepsilon})$. Therefore, for any fixed $\varepsilon > 0$, his algorithm has the best-known approximation guarantee while still having running time close to the time needed for maximum single-commodity flow computation.

Unfortunately, despite this substantial progress in designing efficient poly-logarithmic approximation algorithms for the sparsest cut problem, if one is interested in the generalized sparsest cut problem then the multicommodity flow based $O(\log r)$-approximation algorithm of Leighton and Rao [8] – that can be implemented to run in $\widetilde{O}(n^2 \log U)$ time – is still the fastest one known.

1.3. Our Contribution

In this paper we present a general method of designing fast approximation algorithms for undirected cut-based minimization problems.[2] In particular, we design a procedure that given an integer parameter $k \geq 1$ and any undirected graph G with m edges, n vertices, and having integral edge capacities in the range $[1, \ldots, U]$, produces in $\widetilde{O}(m + 2^k n^{1+1/(2^k-1)} \log U)$

[1] *Conductance* $\Phi(G)$ of a graph $G = (V, E, u)$ is defined as $\Phi(G) := \min_{\emptyset \neq C \subset V} \frac{u(C)}{U(C)U(\overline{C})}$, where $U(C) := \sum_{(v,w)\in E, v \in C} u((v,w))$ is the total capacity-weighted degree of the vertices in C.

[2] In fact, as we briefly mention in section 4.4, our framework can be applied to even more general class of problems: the multicut-based minimization problems.

(Uniform) sparsest cut and balanced separator problem:

Algorithm	Approximation ratio	Running time
AM [3]	$O(\Phi^{-1/2})$	$\widetilde{O}(m)$
AP [7]	$\widetilde{O}(\Phi^{-1/2})$	$\widetilde{O}(m\Phi^{-3/2})$
LR [8]	$O(\log n)$	$\widetilde{O}(n^2)$
ARV [9]	$O(\sqrt{\log n})$	polynomial time
AHK [12]	$O(\sqrt{\log n})$	$\widetilde{O}(n^2)$
KRV [13]	$O(\log^2 n)$	$\widetilde{O}(m + n^{3/2})$
AK [15]	$O(\log n)$	$\widetilde{O}(m + n^{3/2})$
OSVV [16]	$O(\log n)$	$\widetilde{O}(m + n^{3/2})$
S [17]	$O(\sqrt{\log n/\varepsilon})$	$\widetilde{O}(m + n^{3/2+\varepsilon})$
here $k = 1$	$\frac{\log^{3/2+o(1)} n}{\sqrt{\varepsilon}}$	$\widetilde{O}(m + n^{6/5+\varepsilon})$
here $k = 2$	$\frac{\log^{5/2+o(1)} n}{\sqrt{\varepsilon}}$	$\widetilde{O}(m + n^{12/11+\varepsilon})$
here $k \geq 1$	$\frac{\log^{(1+o(1))(k+1/2)} n}{\sqrt{\varepsilon}}$	$\widetilde{O}(m)+$
		$+\widetilde{O}(2^k n^{\frac{3\cdot 2^k}{(3\cdot 2^k-1)}+\varepsilon})$

Generalized sparsest cut problem:

Algorithm	Approximation ratio	Running time
LR [8]	$O(\log r)$	$\widetilde{O}(n^2 \log U)$
CGR [10]	$O(\log^{3/4} r)$	polynomial time
ALN [11]	$O(\sqrt{\log r} \lg \lg r)$	polynomial time
here $k = 2$	$\log^{2+o(1)} n$	$\widetilde{O}(m + \lvert E_D \rvert)+$
	$\log^{2+o(1)} n$	$+\widetilde{O}(n^{4/3} \log U)$
here $k = 3$	$\log^{2+o(1)} n$	$\widetilde{O}(m + \lvert E_D \rvert)+$
	$\log^{3+o(1)} n$	$+\widetilde{O}(n^{8/7} \log U)$
here $k \geq 1$	$\log^{(1+o(1))k} n$	$\widetilde{O}(m + \lvert E_D \rvert)+$
	$\log^{(1+o(1))k} n$	$+\widetilde{O}(2^k n^{\frac{2^k}{2^k-1}} \log U)$

Figure 1. Here, n denotes the number of vertices of the input graph G, m the number of its edges, Φ its conductance, U is its capacity ratio, and r is the number of vertices of the demand graph $D = (V, E_D, d)$ that are endpoints of some demand edges. Also, $\widetilde{O}(\cdot)$ notation suppresses poly-logarithmic factors. The algorithm of Alon and Milman applies only to the sparsest cut problem.

time a small number of trees $\{T_i\}_i$. These trees have a property that, with high probability, for any $\alpha \geq 1$, we can find an $(\alpha \log^{(1+o(1))k} n)$-approximation to given instance of *any* cut-based minimization problem on G by just obtaining some α-optimal solution for each T_i and choosing the one among them that leads to the smallest objective value in G (see Theorem 4.1 for more details). As a consequence, we are able to transform any α-approximation algorithm for a cut-based problem that works only on tree instances, to an $(\alpha \log^{(1+o(1))k} n)$-approximation algorithm for general graphs, while paying a computational overhead of $\widetilde{O}(m + 2^k n^{1+1/(2^k-1)} \log U)$ time that, as k grows, quickly becomes close to linear.

We illustrate the applicability of our paradigm on two fundamental graph partitioning problems: the undirected (generalized) sparsest cut and the balanced separator problems. By a simple use of our framework we obtain, for any integral $k \geq 1$, a $(\log^{(1+o(1))k} n)$-approximation algorithm for the generalized sparsest cut problem that runs in time $\widetilde{O}(m + |E_D| + 2^k n^{1+1/(2^k-1)} \log U)$, where $|E_D|$ is the number of the demand edges in the demand graph. Furthermore, in case of the sparsest cut and the balanced separator problems, we combine our techniques with the algorithm of Sherman [17] to obtain approximation algorithms that have even better dependence on k in the running time. Namely, for any $k \geq 1$ and $\varepsilon > 0$, we present $(\log^{(1+o(1))(k+1/2)} n/\sqrt{\varepsilon})$-approximation algorithms[3] for these problems that run in time $\widetilde{O}(m + 2^k n^{1+1/(3\cdot 2^k-1)+\varepsilon})$. We summarize our results together with the previous work in Figure 1. One can see that even for small values of k, the running times of our algorithms beat the multicommodity flow barrier of $\Omega(n^2)$ and the bound of $\Omega(m + n^{3/2})$ time corresponding to performing sparsification and single-commodity flow computation. Unfortunately, in the case of the generalized sparsest cut problem our approximation guarantee has poly-logarithmic dependence on n. This is in contrast to the previous algorithms for this problem that have their approximation guarantee depending on r.

1.4. Overview of Our Techniques

Our approach is inspired by the cut-based graph decomposition of Räcke [18] that was developed in the context of oblivious routing schemes (see [19], [20], [21], [22] for some previous work on this subject). This decomposition is a powerful tool in designing approximation algorithms for various undirected cut-based problem. It is based on finding for a given graph G with n vertices and m edges, a convex combination $\{\lambda_i\}_i$ of decomposition trees $\{T_i\}_i$ such that: G is embeddable into each T_i, and this convex combination can be embedded into G with $O(\log n)$ congestion. One employs this decomposition by first approximating

[3]As was the case in all the previous work we described, we only obtain a pseudo-approximation for the balanced separator problem. Recall that an *α-pseudo-approximation* for the balanced separator problem with balance constant c is a c'-balanced cut C – for some other constant c' – such that C's sparsity is within α of the sparsest c-balanced cut. For the sake of convenience, in the rest of the paper we don't differentiate between pseudo- and true approximation.

the desired cut-based problem on each tree T_i – this usually yields much better approximation ratio than general instances – and then extracting from obtained solutions a solution for the graph G while incurring only additional $O(\log n)$ factor in the approximation guarantee.

The key question motivating our results is: can the above paradigm be applied to obtain *very efficient* approximation algorithms for cut-based graph problems? After all, one could envision a generic approach to designing fast approximation algorithms for such problems in which one decomposes first an input graph G into a convex combination of structurally simple graphs G_i (e.g. trees), solves the problem quickly on each of these easy instances and then combines these solutions to obtain a solution for the graph G, while losing some (e.g. poly-logarithmic) factor in approximation guarantee as a price of this speed-up. Clearly, the viability of such scenario depends critically on how fast a suitable decomposition can be computed and how 'easy' are the graphs G_is from the point of view of the problems we want to solve.

We start investigation of this approach by noticing that if one is willing to settle for approximation algorithms that are of Monte Carlo-type then, given a decomposition of the graph G into a convex combination $\{(\lambda_i, G_i)\}_i$, one does not need to compute the solution for each of G_is to get the solution for G. It is sufficient to just solve the problem on a small number of G_is that are sampled from the distribution described by λ_is.

However, even after making this observation, one still needs to compute the decomposition of the graph to be able to sample from it and, unfortunately, Räcke's decomposition – that was aimed at obtaining algorithms that are just polynomial-time – has serious limitations when one is interested in time-efficiency. In particular, the running time of his decomposition procedure is dominated by $\widetilde{O}(m)$ all-pair shortest path computations and is thus prohibitively large from our point of view – see section 3.1 for more details.

To circumvent this problem, we design an alternative and more general graph decomposition (see Theorem 3.6). Similarly to the case of Räcke's decomposition, our construction is based on embedding graph metrics into tree metrics. However – instead of the embedding result of Fakcharoenphol, Talwar, and Rao [23] that was used by Räcke – we employ the nearly-linear time algorithm of Abraham, Bartal, and Neiman [24] for finding low-average-stretch spanning trees. This choice allows for a much more efficient implementation of our decomposition procedure at a cost of decreasing the quality of provided approximation from $O(\log n)$ to $\widetilde{O}(\log n)$. Also, even more crucially, inspired by the notion of the ultrasparsifiers of Spielman and Teng [5], [25], we allow flexibility in choosing the type of graphs into which our graph is decomposed. In this way, we are able to offer a trade-off between the structural simplicity of these graphs and the time needed to compute the corresponding decomposition.

Finally, by recursive use of our decomposition – together with sparsification technique – we are able to leverage the above-mentioned flexibility to design a procedure that can have its running time be arbitrarily close to nearly-linear and, informally speaking, allow sampling from a decomposition of our input graph G into graphs whose structure is arbitrarily simple, but at a cost of getting proportionally worse quality of the reflection of the cut structure of G – see Theorem 3.7 for more details.

2. Preliminaries

In this paper we will be concerned with an undirected graph $G = (V, E, u)$ having vertex set V, edge set E, and integer capacities $u : E \to \mathbb{Z}^+$ on the edges. By the *capacity ratio U of G* we mean the maximum possible ratio between capacities of two edges of G i.e. $U := \max_{e,e' \in E} u(e)/u(e')$.

For a given graph $G = (V, E, u)$, by a *cut C* in G we mean any vertex set $\emptyset \neq C \subset V$. We denote by $E(C)$ the set of all the edges of G with exactly one endpoint in C – we say that these edges are *cut* by C. By $\overline{C}$ we will denote the set $V \setminus C$. Finally, we define the *capacity* $u(C)$ of a cut to be the total capacity $u(E(C))$ of all the edges in $E(C)$.

2.1. Maximum Concurrent Flow Problem

For a given graph $G = (V, E, u)$, by a *multicommodity flow* $f = (f^1, \ldots, f^k)$ we mean a set of k flows f^i in G, where each flow f^i routes a commodity i from some *source* s_i to some *sink* t_i. Let us also define $|f(e)|$ as the total flow routed along the edge e by f, i.e. $|f(e)| := \sum_i |f^i(e)|$. We say that a multicommodity flow f is *feasible* if for every edge e, $|f(e)| \leq u(e)$.

One of the most popular multicommodity flow problems is the *maximum concurrent flow* problem. In this problem, in addition to the graph $G = (V, E, u)$, we are given a *demand graph* $D = (V, E_D, d)$. The objective is to find a feasible multicommodity flow f in G such that for each demand edge $e = (v, w) \in E_D$ there is a flow f^e that routes $\theta d(e)$ units of corresponding commodity from v to w and the *flow rate* θ is maximized.

2.2. Embedability

A notion we will be dealing extensively with is the notion of graph embedding.

Definition 2.1. *For given graphs $G = (V, E, u)$ and $\overline{G} = (V, \overline{E}, \overline{u})$, by an* embedding f of G into $\overline{G}$ *we mean a multicommodity flow $f = (f^{e_1}, \ldots, f^{e_{|E|}})$ in $\overline{G}$ with $|E|$ commodities indexed by the edges of G, such that for each $1 \leq i \leq |E|$, the flow f^{e_i} routes in $\overline{G}$ $u(e_i)$ units of flow between endpoints of e_i.*

One may view an embedding f of G into $\overline{G}$ as a concurrent flow in $\overline{G}$ whose source-sink pairs and corresponding demands are given by the endpoints of edges of G and their corresponding capacities. Note that the above definition does not require that f is feasible. We proceed to the definition of embedability.

Definition 2.2. *For given $t \geq 1$, graphs $G = (V, E, u)$, and $\overline{G} = (V, \overline{E}, \overline{u})$, we say that G* is t-embeddable into $\overline{G}$ *if there exists an embedding f of G into $\overline{G}$ such that for all $e \in \overline{E}$, $|f(e)| \leq t\overline{u}(e)$. We say that G* is embeddable into $\overline{G}$ *if G is 1-embeddable into $\overline{G}$.*

Intuitively, the fact that G is t-embeddable into $\overline{G}$ means that we can fractionally pack all the edges of G into $\overline{G}$ with all its capacities $\overline{u}$ multiplied by t. Also, it is easy to see that for given graphs G and $\overline{G}$, one can always find the smallest possible t such that G is t-embeddable into $\overline{G}$ by just solving maximum concurrent flow problem in $\overline{G}$ in which we treat G as a demand graph with the demands given by its capacities.

3. Graph Decompositions and Fast Approximation Algorithms for Cut-based Problems

The central concept of our paper is the notion of $(\alpha, \mathcal{G})$-decomposition. This notion is a generalization of the cut-based graph decomposition introduced by Räcke [18].

Definition 3.1. *For any $\alpha \geq 1$ and some family of graphs $\mathcal{G}$, by an* $(\alpha, \mathcal{G})$-decomposition *of a graph $G = (V, E, u)$ we mean a set of pairs $\{(\lambda_i, G_i)\}_i$, where for each i, $\lambda_i > 0$ and $G_i = (V, E, u_i)$ is a graph in $\mathcal{G}$, such that:*

(a) $\sum_i \lambda_i = 1$
(b) G is embeddable into each G_i
(c) There exist embeddings f_i of each G_i into G such that for each $e \in E$, $\sum_i \lambda_i |f_i(e)| \leq \alpha u(e)$.

Moreover, we say that such decomposition is k-sparse *if it consists of at most k different G_i.*

The crucial property of $(\alpha, \mathcal{G})$-decomposition is that it captures α-approximately the cut structure of the graph G. We formalize this statement in the following easy to prove fact.

Fact 3.2. *For any $\alpha \geq 1$, graph $G = (V, E, u)$, and a family of graphs $\mathcal{G}$, if $\{(\lambda_i, G_i)\}_i$ is an $(\alpha, \mathcal{G})$-decomposition of G then for any cut C of G:*

- *for all i, the capacity $u_i(C)$ of C in G_i is at least its capacity $u(C)$ in G;*
- $\mathbb{E}_{\vec{\lambda}}[u(C)] := \sum_i \lambda_i u_i(C) \leq \alpha u(C)$.

Note that the condition (a) from Definition 3.1 implies that $\{(\lambda_i, G_i)\}_i$ is a convex combination of the graphs $\{G_i\}_i$. Therefore, one of the implications of the Fact 3.2 is that for any cut C in G, not only the capacity of C in every G_i is lowerbounded by its capacity $u(C)$ in G, but also there always exists a graph G_j in which the capacity of C is at most $\alpha u(C)$. As one can easily convince oneself, for any $\beta \geq 1$, this property alone allows to reduce a task of $\alpha\beta$-approximation of any cut-based minimization problem in G to a task of β-approximating this problem in each of G_is.

In fact, if one is willing to settle for Monte Carlo-type approximation guarantees, one can reduce the task of approximation of such problem to a task of approximating it in only a *small number* of G_is. To make this statement precise, let us introduce the following definition.

Definition 3.3. *For given $G = (V, E, u)$, $\alpha \geq 1$, and $1 \geq p > 0$, we say that a collection $\{G_i\}_i$ of random graphs[4] $G_i = (V, E_i, u_i)$,* α-preserves the cuts of G with probability p *if for every cut C of G:*

- *for all i, the capacity $u_i(C)$ of C in G_i is at least its capacity $u(C)$ in G;*
- *with probability at least p, there exists i such that $u_i(C) \leq \alpha u(C)$.*

With a slight abuse of notation, we will say that a random graph G' is α-preserving the cuts of G with probability p if the corresponding singleton family $\{G'\}$ is doing it. Now, a useful connection between $(\alpha, \mathcal{G})$-decomposition of graph G and obtaining graphs $O(\alpha)$-preserving the cuts of G with some probability is given by the following fact whose proof is a straight-forward application of Markov's inequality.

Fact 3.4. *Let $G = (V, E, u)$, $\alpha \geq 1$, $1 > p > 0$ and let $\{(\lambda_i, G_i)\}_i$ be some $(\alpha, \mathcal{G})$-decomposition of G. If G' is a random graph chosen from the set $\{G_i\}_i$ according to distribution given by λ_is i.e. $Pr[G' = G_i] = \lambda_i$, then G' 2α-preserves cuts of G with probability $1/2$.*

[4]We formalize the notion of random graphs by viewing each G_i as a graph on vertex set V chosen according to some underlying distribution over all such graphs.

Therefore – as we will formally prove later – for any $\beta \geq 1$, we can obtain a Monte Carlo $2\alpha\beta$-approximation algorithm for any minimization cut-based problem in G, by β-approximating it in a sample of $O(\ln |V|)$ G_is that were chosen according to distribution described by λ_is.

3.1. Finding a Good $(\alpha, \mathcal{G})$-decomposition Efficiently

In the light of the above discussion, we focus our attention on the task of developing a $(\alpha, \mathcal{G})$-decomposition of graphs into some family $\mathcal{G}$ of structurally simple graphs that enjoys relatively small value of α. Of course, since our emphasis is on obtaining algorithms that are very efficient, an important aspect of the decomposition that we will be interested in is the time needed to compute it.

With this goal in mind, we start by considering the following theorem due to Räcke [18] that describes the quality of decomposition one can achieve when decomposing the graph G into a family $\mathcal{G}_T$ of its decomposition trees (cf. [18] for a formal definition of a decomposition tree).

Theorem 3.5 ([18])**.** *For any graph $G = (V, E, u)$, an $\widetilde{O}(m)$-sparse $(O(\log n), \mathcal{G}_T)$-decomposition of G can be found in polynomial time, where $n = |V|$ and $m = |E|$.*

Due to structural simplicity of decomposition trees, as well as, the quality of cut preservation being the best – up to a constant – achievable in this context, this theorem was used to design good approximation algorithms for a number of cut-based minimization problems.

Unfortunately, from our point of view, the usefulness of Räcke's decomposition is severely limited by the fact that the time needed to construct it is not acceptable when one is aiming at obtaining fast algorithms. More precisely, the running time of the decomposing algorithm of [18] is dominated by $\widetilde{O}(m)$ executions of the algorithm of Fakcharoenphol, Talwar and Rao [23] that solves the minimum communication cost tree problem with respect to a cost function devised from the graph G (cf. [18] for details). Each such execution requires, in particular, computation of the shortest-path metric in G with respect to some length function, which employs the well-known all-pair shortest path algorithm running in $O(\min\{mn, n^{2.376}\})$ time. This results in – entirely acceptable when one is just interested in obtaining polynomial-time approximation algorithms, but prohibitive in our case – $\widetilde{O}(m \min\{mn, n^{2.376}\})$ total running time.

One could try to obtain a faster implementation of Räcke's decomposition by using a nearly-linear time low-average-stretch spanning tree construction due to Abraham, Bartal, and Neiman [24] in place of the algorithm of [23]. This would lead to an $\widetilde{O}(m^2)$ time decomposition of G into its spanning trees that has a – slightly worse – quality of $\widetilde{O}(\log n)$ instead of the previous $O(\log n)$. However, although we will use [24] in our construction, $\widetilde{O}(m^2)$ time is still not sufficiently fast for our purposes.

Our key idea for circumventing this running time bottleneck is allowing ourselves more flexibility in the choice of the family of graphs into which we will decompose G. Namely, we will be considering $(\alpha, \mathcal{G})$-decompositions of G into objects that are still structurally simpler than G, but not as simple as trees.

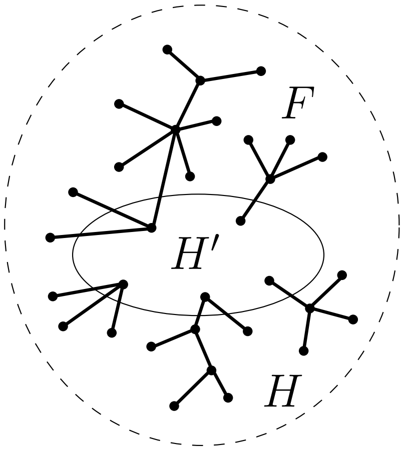

Figure 2. An example of a j-tree H, its core H', and its envelope F consisting of bold edges.

To this end, for $j \geq 1$, we say that a graph $H = (V_H, E_H, u_H)$ is a *j-tree* (cf. Figure 2) if it is a connected graph being a union of: a subgraph H' of H induced by some vertex set $V'_H \subseteq V_H$ with $|V'_H| \leq j$; and of a forest F on V_H whose each connected component has exactly one vertex in V'_H. We will call the subgraph H' the *core* of H and the forest F the *envelope* of H.[5]

Now, if we define $\mathcal{G}_V[j]$ to be the family of all the j-trees on the vertex set V, the following theorem – being the heart of our framework – holds. Its proof is deferred to the full version of the paper.

Theorem 3.6. *For any graph $G = (V, E, u)$ and $t \geq 1$, we can find in time $\widetilde{O}(tm)$ a t-sparse $(\widetilde{O}(\log n), \mathcal{G}_V[\widetilde{O}(\frac{m \log U}{t})])$-decomposition $\{(\lambda_i, G_i)\}_i$*

[5]Note that given a j-tree we can find its envelope in linear time. Therefore, throughout the paper we always assume that the j-trees we are dealing with have their envelopes and cores explicitly marked.

of G, where $m = |E|$, $n = |V|$, and U is the capacity ratio of G. Moreover, the capacity ratio of each G_i is $O(mU)$.

Intuitively, the above theorem shows that if we allow $j = \widetilde{O}(\frac{m \log U}{t})$ to grow, the sparsity of the corresponding decomposition of G into j-trees – and thus the time needed to compute it – will decrease proportionally.

Note that a 1-tree is just an ordinary tree, so by taking t in the above theorem sufficiently large, we obtain a decomposition of G into trees in time $\widetilde{O}(m^2 \log U)$.[6] Therefore, we see that compared to the decomposition result of Räcke (cf. Theorem 3.5), we obtain in this case a decomposition of G into objects that are even simpler than decomposition trees and with more efficient implementation, but at a cost of slightly worse quality.

However, the aspect of this theorem that we will find most useful is the above-mentioned trade-off between the simplicity of the j-trees into which the decomposition decomposes G and the time needed to compute it. This flexibility in the choice of t can be utilized in various ways.

For instance, one should note that, in some sense, the core of a j-tree H captures all the non-trivial cut-structure of H. In particular, it is easy to see that maximum flow computations in H essentially reduce to maximum flow computations in H's core. So, one might hope that for some cut-based problems the complexity of solving such problem in H is proportional to the complexity of its solving in the – possibly much smaller than H – core of H (this is, for example, indeed the case for the balanced separator and the sparsest cut problems – see section 4.3). Thus, one could use Theorem 3.6 for appropriate choice of t, to get a faster algorithm for such problem by just computing first the decomposition and then leveraging the existing algorithms to solve the problem on a small number of sampled j-trees, while paying for this speed-up an additional $\widetilde{O}(\log n)$ factor in the approximation quality.

Even more importantly, our ability to choose in Theorem 3.6 sufficiently small value of t, as well as, the sparsification technique and recursive application of the theorem to the cores of $\widetilde{O}(\frac{m \log U}{t})$-trees sampled from the computed decomposition, allows establishing the following theorem – its proof appears in the full version of the paper.

Theorem 3.7. *For any $1 \geq l \geq 0$, integral $k \geq 1$, and any graph $G = (V, E, u)$, we can find in $\widetilde{O}(m + 2^k n^{(1+\frac{1-l}{2^k-1})} \log U)$ time a collection of $(2^{k+1} \ln n)$ n^l-trees $\{G_i\}_i$ that $(\log^{(1+o(1))k} n)$-preserve the cuts of G with high probability. Moreover, the capacity ratio of each G_i is $n^{(2+o(1))k} U$, where U is the capacity ratio of G, $n = |V|$, and $m = |E|$.*

As one can see, the above theorem allows obtaining a collection of j-trees that α-preserve cuts of G – for *arbitrary* $j = n^l$ – in time *arbitrarily* close to nearly-linear, but at a price of α growing accordingly as these two parameters decrease.

Note that one can get a cut-preserving collection satisfying the requirements of the theorem by just finding an $(\widetilde{O}(\log n), n^l)$-decomposition of G via Theorem 3.6 and sampling – in the spirit of Fact 3.4 – $O(\log n)$ n^l-trees from it so as to ensure that each cut is preserved with high probability. Unfortunately, the running time of such procedure would be too large. Therefore, our approach to establishing Theorem 3.7 can be heuristically viewed as an algorithm that in case when the time required by Theorem 3.6 to compute a decomposition of G into n^l-trees is not acceptable, does not try to compute and sample from such decomposition directly. Instead, it performs its sampling by finding a decomposition of G into j-trees, for some value of j bigger than n^l, then samples a j-tree from it and recurses on this sample. Now, the desired collection of n^l-trees is obtained by repeating this sampling procedure enough times to make sure that the cut preserving probability is high enough, with the value of j chosen so as to bound the number of recursive calls by $k - 1$. Since each recursive call introduces an additional $\widetilde{O}(\log n)$ distortion in the faithfulness of the reflection of the cut structure of G, the quality of the cut preservation of the collection generated via such algorithm will be bounded by $\log^{(1+o(1))k} n$.

4. Applications

We proceed to demonstrating how the tools and ideas presented in the previous section lead to a fast approximation algorithms for cut-based graph problems. In particular, we apply our techniques to the (generalized) sparsest cut and the balanced separator problems. Later we remark on the fact that our framework is also applicable to multicut-based minimization problems.

The following theorem encapsulates our way of employing our framework – its proof appears in the full version of the paper.

Theorem 4.1. *For any $\alpha \geq 1$, integral $k \geq 1$, $1 \geq l \geq 0$, and undirected graph $G = (V, E, u)$ with $n = |V|$, $m = |E|$, and U being its capacity ratio, we can find in $\widetilde{O}(m + 2^k n^{(1+\frac{1-l}{2^k-1})} \log U)$ time, a collection of*

[6]In fact, one might show that if our goal is to decompose G into trees then the running time of our algorithm is just $\widetilde{O}(m^2)$.

$(2^{k+1}\ln n)$ n^l*-trees* $\{G_i\}_i$*, such that for* any *instance* P *of* any *cut-based minimization problem* $\mathcal{P}$ *the following holds with high probability. If* $\{C_i^*\}_i$ *is a collection of cuts of* G *with each* C_i^* *being an* α*-optimal solution to* P *on the* n^l*-tree* G_i *then at least one of* C_i^* *is an* $(\alpha \log^{(1+o(1))k} n)$*-optimal solution to* P *on the graph* G*.*

When we take l equal to zero, the above theorem implies that if we have an α-approximation algorithm for a given cut-based problem on trees that runs in $T(m,n,U)$ time then, for any $k \geq 1$, we can get an $(\alpha \log^{(1+o(1))k} n)$-approximation algorithm for it in general graphs and the running time of this algorithm will be just $\widetilde{O}(m + 2^k n^{(1+1/(2^k-1))} \log U) + (2^{k+1}\ln n)T(m,n,U)$. Note that the computational overhead introduced by our framework, as k grows, quickly approaches nearly-linear. Therefore, if we are interested in designing fast poly-logaritmic approximation algorithms for some cut-based minimization problem, we can just focus our attention on finding a fast approximation algorithm for its tree instances.

Also, an interesting feature of our theorem is that the procedure producing the graphs $\{G_i\}_i$ is completely oblivious to the cut-based problem we want to solve – the fact that this problem is a cut-based minimization problem is all we need to make our approach work.

4.1. Computing Maximum Concurrent Flow Rate on Trees

As Theorem 4.1 suggests, we should focus on designing fast approximation algorithms for tree instances of our problems. The basic tool we will use in this task is the ability to compute maximum concurrent flow rate on trees in nearly-linear time. The main reason why such a fast algorithms exist stems from the fact that if we have a demand graph $D = (V, E_D, d)$ and a tree $T = (V, E_T, u_T)$, there is a unique way of satisfying these demands in T. Namely, for each demand edge $e \in E_D$ the flow of $d(e)$ units of corresponding commodity has to be routed along the unique path $\mathsf{path}_T[D](e)$ joining two endpoints of e in the tree T. As a result, we know that if we want to route in T a concurrent flow of rate $\theta = 1$ then the total amount of flow flowing through a particular edge h of T is equal to

$$u^T[D](h) := \sum_{e \in E_D, h \in \mathsf{path}_T[D](e)} d(e).$$

Interestingly, as it is showed in the full version of the paper, we can compute $u^T[D](h)$ for all $h \in E_T$ in nearly-linear time.

Lemma 4.2. *For any tree* $T = (V, E_T, u_t)$*, and demand graph* $D = (V, E_D, d)$*, we can compute* $u^T[D](h)$*, for all* $h \in E_T$*, in* $\widetilde{O}(|E_D| + |V|)$ *time.*

Now, the crucial thing to notice is that the best achievable flow rate θ^* of the maximum concurrent flow is equal to $\min_{h \in E_T} \frac{u_T(h)}{u^T[D](h)}$. Therefore, Lemma 4.2 implies the following corollary.

Corollary 4.3. *For any tree* $T = (V, E_T, u_T)$ *and demand graph* $D = (V, E_D, d)$*, we can find in* $\widetilde{O}(|E_D| + |V|)$ *time the optimal maximum concurrent flow rate* θ^* *and an edge* $h^* \in E_T$ *such that* $\theta^* = \frac{u_T(h^*)}{u^T[D](h^*)}$*.*

Note that we only compute the optimal flow rate of the concurrent flow and not the actual flows. In some sense, this is unavoidable – one can easily construct an example of maximum concurrent flow problem on tree, where the representation of any (even only approximately) optimal flow has size $\Omega(|E_D||V|)$.

4.2. Generalized Sparsest Cut Problem

We proceed to designing a fast approximation algorithm for the generalized sparsest cut problem. We start by noticing that for a given tree $T = (V, E_T, u_T)$, demand graph $D = (V, E_D, d)$, and an edge h of this tree, the quantity $\frac{u_T(h)}{u^T[D](h)}$ is exactly the (generalized) sparsity of the cut that cuts in T only the edge h. Therefore, Corollary 4.3 together with the fact that the sparsity of the sparsest cut is always an upper bound on the maximum flow rate achievable, gives us the following corollary.

Corollary 4.4. *For any given tree* $T = (V, E, u)$ *and demand graph* $D = (V, E_D, d)$*, an optimal solution to the generalized sparsest cut problem can be computed in* $\widetilde{O}(|E_D| + |V|)$ *time.*

Now, by applying Theorem 4.1 together with a preprocessing step of sparsification of D, we are able to obtain a poly-logarithmic approximation for the generalized sparsest cut problem in time close to linear. The proof of the following theorem appears in the full version of the paper.

Theorem 4.5. *For any graph* $G = (V, E, u)$*, demand graph* $D = (V, E_D, d)$*, and integral* $k \geq 1$*, there exists a Monte Carlo* $\log^{(1+o(1))k} n$*-approximation algorithm for generalized sparsest cut problem that runs in time* $\widetilde{O}(m + |E_D| + 2^k n^{(1+1/(2^k-1))} \log U)$*, where* $n = |V|$*,* $m = |E|$*, and* U *is the capacity ratio of* G*.*

4.3. Balanced Separator and Sparsest Cut Problem

We turn our attention to the balanced separator problem. Analogously to the case of the generalized sparsest cut problem above, to employ our approach we need an efficient algorithm for the tree instances of the balanced separator problem. Unfortunately, although we can solve this problem on trees optimally via dynamic programming, there seems to be no algorithm that does it very efficiently – ideally, in nearly-linear time. Therefore, we circumvent this problem by settling for a fast but approximate solution.

Namely, we use the result of Sherman [17] who shows that, for any $\varepsilon > 0$, the balanced separator problem – as well as the sparsest cut problem – can be $O(\sqrt{\log n/\varepsilon})$-approximated in a graph G – with n vertices and m edges – in time $\widetilde{O}(m + n^{3/2+\varepsilon})$. This running time corresponds to sparsifying G and then using the fastest known algorithm for the maximum flow problem due to Goldberg and Rao [26] to perform maximum flow computations on a sequence of n^{ε} graphs that are derived in a certain way from the sparsified version of G.

Unfortunately, $\widetilde{O}(m + n^{3/2+\varepsilon})$ running time is still not sufficiently fast for our purposes. At this point, however, we recall that maximum flow computation on a j-tree reduces to the task of finding the maximum flow in its core. So, if we want to perform a maximum flow computation in a j-tree that has its core sparsified (i.e. the core has only $\widetilde{O}(j)$ edges), the real complexity of this task is proportional to j as opposed to being proportional to n. This motivates us to obtaining an implementation of Sherman's algorithm on j-trees that achieves better running time – the proof of the following lemma appears in the full version of the paper.

Lemma 4.6. *For any j-tree $G = (V, E, u)$ and $\varepsilon > 0$, we can $O(\sqrt{\log n/\varepsilon})$-approximate the balanced separator and the sparsest cut problems in $\widetilde{O}(m+n^{\varepsilon}(n+j^{3/2}))$ time, where $m = |E|$ and $n = |V|$.*

Now, we can use Theorem 4.1 and Lemma 4.6 – for the right choice of $j = n^l$ – together with a simple preprocessing making the capacity ratio of the graphs we are dealing with polynomially bounded, to obtain the following result – its proof is deferred to the full version of the paper.

Theorem 4.7. *For any $\varepsilon > 0$, integral $k \geq 1$, and graph $G = (V, E, u)$, we can $(\log^{(1+o(1))(k+1/2)} n/\sqrt{\varepsilon})$-approximate the sparsest cut and the balanced separator problems in time $\widetilde{O}(m + 2^k n^{1+\frac{1}{3\cdot 2^k - 1}+\varepsilon})$, where $m = |E|$ and $n = |V|$.*

Recall that our main motivation to establishing Lemma 4.6 was our inability to solve the balanced separator problem on trees efficiently. However, the obtained trade-off between the running time of the resulting algorithm and the quality of the approximation provided for both the balanced separator and the sparsest cut problems is much better than the one we got for the generalized sparsest cut problem (cf. Theorem 4.5). This shows us that sometimes it is beneficial to take advantage of the flexibility in the choice of l given by Theorem 4.1 by combining it with an existing fast approximation algorithm for our cut-based problem that allows a faster implementation on j-tree instances.

4.4. Extension to Multicut-based Problems

We briefly remark on the applicability of our framework to a class of undirected multicut-based minimization problems which generalize the undirected cut-based minimization ones we consider in the paper. To this end, let us define a problem $\mathcal{P}$ to be an *(undirected) multicut-based (minimization) problem* if its every instance $P \in \mathcal{P}$ on a graph $G = (V, E, u)$ can be cast as a task of finding a partition $\{C_i^*\}_{i=1}^{k^*}$ of vertices of V that minimizes the quantity $u(\{C_i\}_{i=1}^k) f_P(\{C_i\}_{i=1}^k)$ over all partitions $\{C_i\}_{i=1}^k$ of the vertex set V. Here, $u(\{C_i\}_{i=1}^k)$ is the total capacity of all the edges of G whose endpoints are in different C_is. Once again, we require f_P to be a non-negative function that can depend on P, but does not depend on the graph G. Note that in our definition we do not restrict a priori the number of sets into which V is partitioned in considered partitions – this can be however specified by appropriate choice of the function f_P. Two important examples of problems that are captured by this definition are the multiway cut problem (cf. [27]) and the multicut problem (cf. [28]).

To see why our framework can be extended to handle multicut-based problems, note that for any partition $\{C_i\}_{i=1}^k$, we can express the total capacity of edges cut by it as a linear combination of the capacities of the cuts corresponding to each set C_i. Namely, $u(\{C_i\}_{i=1}^k) = \frac{1}{2}\sum_{i=1}^k u(C_i)$. Therefore, if G is our input graph then any graph that approximates the cuts of G also approximates its multicuts. This implies, in particular, that sparsification preserves the multi-cuts up to a constant factor. Furthermore, by linearity of expectation, the analogs of Fact 3.2 and 3.4 hold for multicuts as well. It turns out, however, that this is all that we need to prove an extension of Theorem 4.1 that handles minimization multicut-based problems.

Acknowledgments

We are grateful to Sanjeev Arora, Michel Goemans, Piotr Indyk, Jonathan Kelner, Shiva Kintali, Ofer Neiman, Richard Peng, Daniel Spielman, and anonymous reviewers for helpful comments and discussion.

References

[1] D. B. Shmoys, "Cut problems and their application to divide-and-conquer," in *Approximation Algorithms for NP-hard Problems*, D. Hochbaum, Ed., 1997, pp. 192–235.

[2] J. Cheeger, "A lower bound for the smallest eigenvalue of the laplacian," in *Problems in Analysis, eds Gunning RC*, 1970.

[3] N. Alon and V. Milman, "λ_1, isoperimetric inequalities for graphs and superconcentrators," *Journal of Combinatorial Theory, Series B*, vol. 38, pp. 73–88, 1985.

[4] J. Kuczyński and H. Woźniakowski, "Estimating the largest eigenvalues by the power and lanczos algorithms with a random start," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 4, pp. 1094–1122, 1992.

[5] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC'04: Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, 2004, pp. 81–90.

[6] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *FOCS'06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006, pp. 475–486.

[7] R. Andersen and Y. Peres, "Finding sparse cuts locally using evolving sets," in *STOC'09: Proceedings of the 41st Annual ACM symposium on Theory of Computing*, 2009, pp. 235–244.

[8] T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *Journal of the ACM*, vol. 46, no. 6, pp. 787–832, 1999.

[9] S. Arora, S. Rao, and U. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *Journal of the ACM*, vol. 56, no. 2, pp. 1–37, 2009.

[10] S. Chawla, A. Gupta, and H. Räcke, "Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut," in *SODA'05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005, pp. 102–111.

[11] S. Arora, J. R. Lee, and A. Naor, "Euclidean distortion and the sparsest cut," in *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, 2005, pp. 553–562.

[12] S. Arora, E. Hazan, and S. Kale, "$O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time," in *FOCS'04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004, pp. 238–247.

[13] R. Khandekar, S. Rao, and U. Vazirani, "Graph partitioning using single commodity flows," in *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 2006, pp. 385–390.

[14] A. A. Benczúr and D. 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*, 1996, pp. 47–55.

[15] S. Arora and S. Kale, "A combinatorial, primal-dual approach to semidefinite programs," in *STOC'07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, 2007, pp. 227–236.

[16] L. Orecchia, L. J. Schulman, U. V. Vazirani, and N. K. Vishnoi, "On partitioning graphs via single commodity flows," in *STOC'08: Proceedings of the 40th Annual ACM Symposium on the Theory of Computing*, 2008, pp. 461–470.

[17] J. Sherman, "Breaking the multicommodity flow barrier for $O(\sqrt{logn})$-approximations to sparsest cut," in *FOCS'09: Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009, pp. 363–372.

[18] H. Räcke, "Optimal hierarchical decompositions for congestion minimization in networks," in *STOC'08: Proceedings of the 40th Annual ACM Symposium on the Theory of Computing*, 2008, pp. 255–264.

[19] ——, "Minimizing congestion in general networks," in *FOCS'02: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 43–52.

[20] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke, "Optimal oblivious routing in polynomial time," in *STOC'03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003, pp. 383–388.

[21] M. Bienkowski, M. Korzeniowski, and H. Räcke, "A practical algorithm for constructing oblivious routing schemes," in *SPAA'03: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, 2003, pp. 24–33.

[22] C. Harrelson, K. Hildrum, and S. Rao, "A polynomial-time tree decomposition to minimize congestion," in *SPAA'03: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, 2003, pp. 34–43.

[23] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," in *STOC'03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003, pp. 448–455.

[24] I. Abraham, Y. Bartal, and O. Neiman, "Nearly tight low stretch spanning trees," in *FOCS'08: Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, 2008, pp. 781–790, full version available at arXiv:0808.2017.

[25] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *CoRR*, vol. abs/cs/0607105, 2006.

[26] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *Journal of the ACM*, vol. 45, no. 5, pp. 783–797, 1998.

[27] D. R. Karger, P. Klein, C. Stein, M. Thorup, and N. E. Young, "Rounding algorithms for a geometric embedding of minimum multiway cut," in *STOC '99: Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999, pp. 668–678.

[28] N. Garg, V. V. Vazirani, and M. Yannakakis, "Approximate max-flow min-(multi)cut theorems and their applications," *SIAM Journal on Computing*, vol. 25, no. 2, pp. 235–251, 1996.