Computing Cut-Based Hierarchical Decompositions in Almost Linear Time

Harald Räcke* Chintan Shah† Hanjo Täubig‡

Abstract

We present a fast construction algorithm for the hierarchical tree decompositions that lie at the heart of oblivious routing strategies and that form the basis for approximation and online algorithms for various cut problems in graphs.

Given an undirected graph $G = (V, E, c)$ with edge capacities, we compute a single tree $T = (V_T, E_T, c_T)$, where the leaf nodes of T correspond to nodes in G, such that the tree approximates the cut-structure of G up to a factor of $\mathcal{O}(\log^4 n)$. The best existing construction by Harrelson, Hildrum, and Rao [12] just guarantees a polynomial running time but offers a better approximation guarantee of $\mathcal{O}(\log^2 n \log\log n)$.

Phrasing our results in terms of vertex sparsifiers, we obtain the following: For a graph $G = (V, E)$ with a subset S of terminals, we compute a tree T with at most $2|S|$ vertices (and the leafs of T correspond to nodes in S) such that T is a flow-sparsifier for S in G with quality $\mathcal{O}(\log^2 n \log^2 k)$, where $|V| = n$ and $|S| = k$.

The running time is $\mathcal{O}(\operatorname{polylog} n \cdot T(m, 1/\log^3 n))$ where $T(m, \varepsilon)$ is the time for computing an approximate maxflow in a graph with m edges. The latter is almost linear due to the recent results of Sherman [23] and Kelner et al. [13].

1 Introduction

A hierarchical decomposition of a graph G is a recursive partitioning of the vertex set into smaller and smaller components such that on the final level of the recursion the components just contain single vertices of G. Such a recursive decomposition corresponds in a natural way to a *decomposition tree* T in which leaf nodes correspond to nodes in G, the root corresponds to the whole vertex set, and internal vertices correspond to subsets generated during the partitioning.

*Department of Computer Science, Technische Universität München, Garching, Germany, raecke@in.tum.de

†Department of Computer Science, Technische Universität München, Garching, Germany, shahc@in.tum.de

‡Department of Computer Science, Technische Universität München, Garching, Germany, taeubig@in.tum.de

In the context of designing oblivious routing schemes, Räcke [21] developed *cut-based hierarchical decompositions*—hierarchical decompositions such that the decomposition tree (with suitable edge-weights) closely approximates the cut-structure of the graph G. These have subsequently been used to obtain approximate solutions for a variety of cut-related problems that seem very hard on general graphs but that are efficiently solvable on trees. Examples include: Minimum Bisection, Simultaneous Source Location, Online Multicut, k-multicut etc. (see e.g. [1, 2, 4, 7, 10, 14, 16, 22]).

In 2003, Harrelson, Hildrum, and Rao [12] have designed a hierarchical decomposition that guarantees a factor of $\mathcal{O}(\log^2 n \log\log n)$ between cuts in G and in the decomposition tree T, and this is still the best guarantee for general undirected graphs. Instead of considering a *single tree* to approximate the cut-structure of a graph G, Räcke [22] considered a convex combination of decomposition trees and obtained an approximation guarantee of $\mathcal{O}(\log n)$, which is optimal.

A major drawback of all the above approaches is that the running time required for constructing the decomposition tree (or the distribution over decomposition trees) is quite large. Harrelson et al. [12] give just a polynomial time algorithm, and a suitable implementation of the result in [22] still requires running time $\tilde{\mathcal{O}}(m^2)$. This problem has been partially alleviated by Mądry [18] who approximates the graph by a convex combination of j-trees, which are trees with a few additional edges. This allows him to obtain e.g. an $\mathcal{O}(\log^{2+o(1)} n)$ approximation to the generalized sparsest cut problem with running time roughly $\tilde{\mathcal{O}}(m + n^{4/3})$.

In this paper we give a fast construction algorithm for a cut-based hierarchical decomposition (i.e., a single decomposition tree) such that the approximation guarantee between the decomposition tree and the underlying graph is $\mathcal{O}(\log^4 n)$. While our approximation guarantee is weaker than the guarantee given by [12] we obtain a very fast running time of $\tilde{\mathcal{O}}(m^{1+o(1)})$, and, importantly, do not compute a distribution over trees but a *single tree*.

While quite a few problems like e.g. Minimum Bisection or Generalized Sparsest Cut can be approximated using a distribution over trees, there are important problems that require a single tree. One example, is Min-Max k-Partitioning [4]. In this problem you are asked to partition a graph into balanced components such that the number of edges incident to a single component is minimized. The objective function is not linear in the total number of cut edges and therefore an approximation by a convex combination of trees does not work.[1] (see e.g. [14, 2] for other problems that require a single tree).

One can also interpret our results in terms of *vertex sparsifiers* [20, 17, 6, 9, 19, 11]. In this scenario you are given a large capacitated graph $G = (V, E, c)$ together with a set of terminals $T \subseteq V$, $|T| \ll |V|$. One has to solve some problem (like e.g. a flow problem) *on the terminals*.

For this one can construct a *flow-sparsifier*, which is defined as a graph $H = (V_H, E_H, c_H)$ with $T \subseteq V_H$, $|T| = k$ such that any feasible multicommodity flow problem in G can be supported in H with small congestion, and vice versa. Then one can solve/approximate the problem on H instead of on G with a small loss due to the approximation. The main advantage is that one obtains approximation guarantees that are independent of the size of G and only depend on the size of H (which should not be much larger than k).

However, for this approach to make sense, G must be huge (most approximation algorithms for which the approach is interesting have polylogarithmic approximation guarantees; therefore if $|V| \leq \text{poly}(|V_H|)$ the approach does not have a large effect). As G is very large it is mandatory that the sparsifier can be computed quickly as this dominates the overall running time.

A straightforward extension of our result gives a decomposition tree in which the leaf vertices correspond to terminals. Then the decomposition tree (of height $\mathcal{O}(\log k)$ and size $O(k)$) forms a *flow sparsifier* with quality $\mathcal{O}(\log^2 n \log^2 k)$, where n denotes the number of vertices of G and k denotes the number of terminals. On the one hand, this result is good as it shows for the first time that a flow sparsifier with a reasonable guarantee can be computed in nearly linear time, on the other hand the quality guarantee is not independent of the size of G. It is an interesting open problem whether our construction can be improved to give a guarantee independent of n.

1.1 Techniques. One important building block of our result is an algorithm for the following graph partitioning problem. Given an (unweighted) graph $G = (V, E)$, partition G into components $V_1, \ldots, V_\ell$ such that the set C of edges between different components is well connected. This means if (in G) we add a vertex in the middle of each edge $e \in C$ we can route an all-to-all multicommodity flow problem between all these vertices.

The original results on cut-based decompositions ([21, 5], and also to some extent [12]) relied on a slightly different graph partition. There it was required that one can partition a sub-cluster $S \subset V$ of the graph such that all cut edges *together* with the edges that leave S are well connected (via edges in the induced graph $G[S]$). As this property clearly does not hold for every set $S \subset V$ ($G[S]$ might be disconnected) one has to carefully ensure that one does not call the partitioning routine on bad sets. This is one factor that caused the running time of [12] and [5] to just be polynomial.

We obtain our result by heavily using ideas from the cut matching game introduced by Khandekar, Rao, and Vazirani [15]. For getting an almost linear running time we use the recent breakthrough result due to Sherman [23] and Kelner et al. [13] who show that one can find a $(1-\varepsilon)$-approximation to the maximum s-t flow in an undirected graph in time $O(\text{poly}(1/\varepsilon) \cdot m^{1+o(1)})$. This is presented in Section 3.

In Section 4 we design an oblivious routing scheme with competitive ratio $\mathcal{O}(\log^4 n)$ based on the hierarchical decomposition introduced in Section 3. This shows the required approximation between the decomposition tree and the graph. Note that the routing paths for oblivious routing are not constructed in almost linear time. Doing this naïvely by specifying paths between every s-t pair requires at least time n^2. It is possible to encode the paths more efficiently (e.g. by embedding a hypercube) but even in an expander it is unclear how to do this in nearly linear time.

2 Preliminaries

For keeping the presentation as simple as possible we consider throughout the paper an undirected, unweighted graph $G = (V, E)$. All our results extend to the case of polynomially bounded edge capacities in a straightforward manner. We use n and m to denote the number of vertices and edges, respectively, in G.

In the Demands Multicommodity Flow problem, we are given k source-target pairs (s_i, t_i) with demand D_i for $i = 1, \ldots, k$. We wish to route a flow which satisfies all these demands concurrently while minimizing the congestion in the network, which is the maximum flow passing through any edge in the graph.

We use the following definition introduced by Chekuri et al. [8].

DEFINITION 1. *For a graph $G = (V, E)$ a set $X \subseteq V$ of terminals is called α-flow-linked if there is a feasible*

[1]We thank Robert Krauthgamer for making us aware of this fact.

multicommodity flow with demand $\alpha/|X|$ between every unordered pair of terminals from X.

For our purposes we extend this definition to edges using the following subdivision graph G' obtained from G. We construct the *subdivision graph* $G' = (V', E')$ from G by splitting each edge $e = (u, v)$ via a new vertex x_e into two edges (u, x_e) and (x_e, v). The resulting vertex and edge sets are $V' = V \uplus X_E$ and $E' = \{(u, x_e), (x_e, v) \mid e = (u, v) \in E\}$. We use $X_A = \{x_e \mid e \in A\}$ for an edge subset $A \subseteq E$ to denote the set of corresponding split vertices in G'. For a subset $F \subseteq E$ of edges in G, we say F is α-flow-linked in G if and only if the vertex set X_F is α-flow-linked in G'.

In an oblivious routing algorithm, we are given the network and before any demands arrive, we must specify a fractional flow for routing 1 unit of flow between each possible source-target pair which is then scaled to the value of the demand as it arrives.

In a hierarchical decomposition of G, we start from the set V, and successively partition the set(s), until each set is a singleton vertex. We view this hierarchical decomposition as a tree $T = (V_T, E_T)$, where the root corresponds to the set V, and leaves correspond to singleton sets $\{v\}$ for all $v \in V$.

Consider two adjacent tree vertices $p, c \in V_T$ s.t. p is the parent of child c, i.e., the corresponding vertex sets fulfill $V_c \subseteq V_p \subseteq V$. Within the tree, we assign to edge $(p, c) \in E_T$ the total capacity of all edges between V_c and $V \setminus V_c$, denoted by $\mathrm{cap}(V_c)$.

Given a set of demands between $s_i, t_i \in V$ in G, we can route it in T between $\{s_i\}$ and $\{t_i\}$ along the corresponding unique path with no more congestion than required by an optimal routing in G. The goal is to design a decomposition tree such that every multicommodity flow demand feasible for T can be routed in G with small congestion.

3 The Cutting Scheme

In this section we describe how to construct a hierarchical decomposition that can be used to approximate the cut structure of the input graph $G = (V, E)$. The most straightforward way to define a hierarchical decomposition is to give a cutting scheme that for a subset $S \subseteq V$ partitions S into sub-clusters $S_1, \ldots, S_s$ s.t. $S_1 \uplus \cdots \uplus S_s = S$. Applying this scheme recursively (starting with $S = V$ and stopping if $|S| = 1$) gives a hierarchical decomposition. Usually, one also requires that $|S_i| \leq c \cdot |S|$ holds for some constant $c < 1$ and all $i \in \{1, \ldots, s\}$, as this ensures that the resulting decomposition tree has logarithmic height.

We will not be following this approach but we instead define a cutting scheme that defines a two-level hierarchy. For an input cluster $S \subseteq V$ we define clusters L and R s.t. $L \uplus R = S$ and we define clusters $L_1, \ldots, L_\ell$ and $R_1, \ldots, R_r$ s.t. $L_1 \uplus \cdots \uplus L_\ell = L$ and $R_1 \uplus \cdots \uplus R_r = R$. In addition clusters L_i and R_i fulfill $|L_i| \leq \frac{3}{4}|S|$ and $|R_i| \leq \frac{3}{4}|S|$, respectively.

We do this in two steps. We first partition S into $Z_1, \ldots, Z_z$ with $Z_i \leq \frac{3}{4}|S|$ for $i \in \{1, \ldots, z\}$. Then we partition S again into two sets L and R. The sub-clusters $L_1, \ldots, L_\ell$ and $R_1, \ldots, R_r$ are then obtained by intersecting L and R with each one of the Z_i's (ignoring empty intersections). While the first partition into $Z_1, \ldots, Z_z$ only depends on the induced subgraph $G[S]$, the second partition depends on the *boundary edges of* S (edges that leave the sub-cluster S in G). The second partition also depends on the first partition.

For technical reasons, the following partitioning scheme is only used for clusters S that are sufficiently large ($|S| > 64$). For smaller clusters, we can compute a brute force partitioning in constant time, e.g., according to [12].

3.1 The first partition.

The first partitioning into sets $Z_1, \ldots, Z_z$ aims on the one hand to cut only a small number of edges and to create balanced sub-clusters (i.e., each Z_i has $|Z_i| \leq \frac{3}{4}|S|$). However, this alone is not sufficient. The main goal of this section is to prove the following theorem.

THEOREM 3.1. *There is an algorithm* PartitionA *that partitions the induced subgraph $G[S]$ of a cluster S into connected components $Z_1, \ldots, Z_z$ such that*

- *$Z_1 \uplus \cdots \uplus Z_z = S$,*
- *$\forall i \in \{1, \ldots, z\} : \ |Z_i| \leq \frac{3}{4}|S|$, and*
- *the set of inter-cluster edges*

$$F = \{\{u, v\} \mid u \in Z_i, v \in Z_j, i \neq j\}$$

is α-flow-linked in $G[S]$ for $\alpha = \Omega(1/\log^2 n)$.

PartitionA *runs in time $\mathcal{O}(T(m, \varepsilon) \cdot \mathrm{polylog}(m))$, where m denotes the number of edges of $G[S]$.*

Since the partitioning into $Z_1, \ldots, Z_z$ does not depend on the edges leaving S in G, we can forget about the graph G and only focus on the induced subgraph $G[S]$. To simplify notation we will use $G = (V, E)$ in the remainder of this section to refer to the graph $G[S]$. Similarly, n and m refer to the number of nodes and edges, respectively, in the graph $G[S]$. We say that a subset $F \subseteq E$ of edges induces a *balanced clustering* in G if after removing the edge set F from G every connected component in the resulting graph contains at most $\frac{3}{4}|V|$ vertices.

In order to show Theorem 3.1 we prove two lemmas. The first lemma says that given an edge set that induces a balanced clustering we can either reduce the size of the edge set by a lot or we can argue that a large fraction of the edges are well linked.

Lemma 3.1. *Given a subset $F \subseteq E$ that induces a balanced clustering, we can find in time $\tilde{\mathcal{O}}(T(m,\varepsilon))$, either*

1. *a smaller edge set $F_{\text{new}} \subseteq E$ with $|F_{\text{new}}| \le \frac{7}{8}|F|$ that induces a balanced clustering,* ***or***
2. *an edge set $F_{\text{new}} = A_{\text{new}} \uplus R_{\text{new}}$ with $|A_{\text{new}}| \le |F|$, $|R_{\text{new}}| \le 2|A_{\text{new}}|/\log(n)$, such that F_{new} induces a balanced clustering and A_{new} is $\Omega(1/\log^2 n)$-flow-linked in G.*

Obviously, repeatedly applying Lemma 3.1 ensures that at some point Case 2 must occur. This means we have an edge set that induces a balanced clustering and for which a large fraction of the edges is well-linked. However, our goal is to have *all* the edges well-linked. In order to guarantee this we need the second lemma.

Lemma 3.2. *Given an edge-set $F = A \uplus B$ inducing a balanced clustering, with $|B| \le 2|A|/\log(n)$, we can find in time $\tilde{\mathcal{O}}(T(m,\varepsilon))$, either*

1. *an edge-set $F_{\text{new}} \subseteq E$ with $|F_{\text{new}}| \le \frac{3}{4}|F|$ that induces a balanced clustering,* ***or***
2. *an edge-set C such that $F_{\text{new}} = A \uplus C$ induces a balanced clustering and in G', there exists a multicommodity flow from nodes in X_C to X_A that can be routed with congestion $\mathcal{O}(\log n)$ and*
   - *every node $x_c \in X_C$ sends 1 unit of flow;*
   - *every node $x_a \in X_A$ receives at most 6 units of flow.*

Now, Theorem 3.1 follows by repeatedly applying the previous two lemmas until an application of Lemma 3.2 terminates with Case 2. More precisely, we start with $F = E$ and apply Lemma 3.1. Whenever, the previous application resulted in Case 1 of either Lemma 3.1 or Lemma 3.2 we apply Lemma 3.1; otherwise we apply Lemma 3.2. Note, that whenever Case 1 occurs the cardinality of the edge set decreases by a constant factor. Hence, the total number of applications is at most $\mathcal{O}(\log n)$.

In the end, the fact that edges in A are $\Omega(1/\log^2 n)$-flow-linked and the fact that we can route with congestion $\mathcal{O}(\log n)$ a multicommodity flow such that each edge in C sends 1 unit of flow and each edge in A receives at most 6 units of flow, ensures that edges in F are $\Omega(1/\log^2 n)$-flow-linked. This gives Theorem 3.1.

3.1.1 Proof of Lemma 3.1.

Suppose that we are given a set of edges F that induces a balanced clustering. Our goal is to embed an expander between the nodes $X_F = \{x_e \mid e \in F\}$ into G' with small congestion, following the cut-matching game approach of Khandekar, Rao, and Vazirani [15]. They embed an expander into a graph G with congestion $\mathcal{O}(\log^2(n)/\alpha)$, where α is the expansion of G.

The main difficulty to use this result for our setting is that the subset X_F of nodes in G' may not be expanding, and, in particular, G' may not be an expander. Hence, embedding any expander between nodes of X_F induces high congestion. However, we are allowed to change the set F as long as the new edge set still induces a balanced clustering. Therefore, whenever during the KRV-procedure we discover a cut that does not expand well we will change the edge set F. For this new edge set we will not start the KRV-procedure from scratch as this might be very time consuming, but we will re-use the iterations that we already performed. In total we show that we require only $\mathcal{O}(\log^2 n)$ iterations to arrive at an edge set F_{new} for which we have embedded an expander between a large fraction of the nodes of $X_{F_{\text{new}}}$. The fact that we do not embed an expander between all nodes is due to the use of an approximate Maxflow/Mincut algorithm instead of an exact algorithm.

We can view the analysis of the KRV-cut matching game as follows. Initially, every node v of the graph G generates one unit of flow of a unique commodity. More precisely, *the state* of a node v can be described by an n-dimensional vector f_v that is initially equal to e_v (the vector that is 1 at position v and 0 everywhere else).

Now, in every iteration a perfect matching between vertices is applied. If two vertices are matched they average their flow vectors (by routing flow between themselves in the graph). At any point the total flow at a node remains 1; only the commodities this flow consists of changes throughout the algorithm. In the end if all flow vectors consist of a nearly uniform distribution over commodities it can be shown that the embedded matchings form an expander. More precisely, this is guaranteed to happen if $\sum_v \left\| f_v - \frac{1}{n} \right\|_2^2 \le \frac{1}{4n^2}$, where f_v denotes the flow vector of node v. Indeed, if this equation holds it means in particular that (using the embedded matchings) every node has routed at least $\frac{1}{2n}$ to every other node. This is only possible if the union of matchings forms an expander.

The above only sketches the *analysis* of the KRV-framework. In particular, the flow vectors are not explicitly maintained as their combined size might be quadratic which would result in a large running time.

We proceed differently. We start with a flow vector

f_e of a unique commodity at each node x_e, $e \in F$, where F is our initial edge set. Then we apply fractional, partial matchings. This means we have a weighted graph with vertex set X_F so that the degree (total weight of incident edges) of any node is at most 1. Having an edge of weight w_{uv} between u and v means that $w_{uv}/2$ of the flow vector of u is sent to v and vice versa. We make sure that the partial fractional matching can be routed in G' with congestion 1.

In addition to applying matchings we may move flow vectors or we may delete flow vectors. Whenever we change the edge set F into a new edge set F_{new}, we move flow vectors from old vertices $x_{e_{\text{old}}}$, $e_{\text{old}} \in F \setminus F_{\text{new}}$ to new vertices $x_{e_{\text{new}}}$, $e_{\text{new}} \in F_{\text{new}} \setminus F$. This is also done along paths in the graph with constant congestion. However, some of the new edges may not receive a flow vector. We set $F_{\text{new}} := A_{\text{new}} \uplus R_{\text{new}}$, where A_{new} is the set of *active* edges (i.e. edges that have a flow vector) and R_{new} are the remaining edges in F_{new}. In addition some of the flow vectors at nodes in X_F may be deleted (as e.g. $|F_{\text{new}}| \le |F|$ and we cannot move every flow vector to a new edge).

In the end, let A be the set of *active edges* (cut edges e for which x_e has a flow vector), and let $\mu = \frac{1}{|A|}\sum_{e\in A} f_e$ be the average flow vector among these edges. We terminate if

$$\sum_{e\in A} \|f_e - \mu\|_2^2 \le \frac{1}{16n^2} \tag{3.1}$$

For technical reasons the procedure may also terminate if the current set $|F_{\text{new}}|$ of edges fulfills $|F_{\text{new}}| \le \frac{7}{8}|F|$. Then we start the KRV-procedure from scratch.

Proposition 3.1. *If Equation 3.1 holds after i iterations then the set A is $\Omega(1/i)$-well-linked.*

For a specified time t, let A denote the set of active edges and let $\mu = \frac{1}{|A|}\sum_{e\in A} f_e$ denote the average flow vector of active edges at this time. Then we define the potential at time t as

$$\Phi(t) = \sum_{e\in A} \|f_e - \mu\|_2^2 \ .$$

Note that the initial potential Φ is $|F|-1 \le n^2$, and we terminate if the potential drops below $\frac{1}{16n^2}$. Analogous to the KRV-analysis we show that the expected decrease of potential in every round is at least $\Phi(t)/\mathcal{O}(\log n)$. This gives $\mathcal{O}(\log^2 n)$ iterations in total.

A single iteration is performed as follows. We first project the flow vectors f_e for $e \in A$ onto a random direction r (this can be done efficiently). Let $u_e = \langle f_e, r\rangle$ denote the length of the projection of f_e onto r (i.e., the dot product of the vectors), and let $\bar\mu = \langle \mu, r\rangle$ denote the length of the projection of the average flow vector onto r. For some edge set S, we consider the potential

$$P_S = \sum_{e\in S} |u_e - \bar\mu|^2 \ .$$

Now, we choose $\le |A|/8$ *source edges* and $\ge |A|/2$ *target edges* from A, according to the following lemma.

Lemma 3.3. *Given an edge set A, we can efficiently find a set of* source edges $A_S \subset A$ *and a set of* target edges $A_T \subset A$ *and a separation value η s.t.*

1. *η separates the sets, i.e., either* $\max_{e\in A_S} u_e \le \eta \le \min_{h\in A_T} u_h$ *or* $\min_{e\in A_S} u_e \ge \eta \ge \max_{h\in A_T} u_h$,
2. *$|A_T| \ge |A|/2$, $|A_S| \le |A|/8$,*
3. *for every source edge $e \in A_S$: $|u_e - \eta|^2 \ge \frac{1}{9}|u_e - \bar\mu|^2$,*
4. *$\sum_{e\in A_S} |u_e - \bar\mu|^2 \ge \frac{1}{80}\sum_{e\in A} |u_e - \bar\mu|^2$.*

Proof. Let $L = \{e \in A \mid u_e < \bar\mu\}$ and let $R = \{e \in A \mid u_e \ge \bar\mu\}$, i.e., $L \uplus R = A$), and assume w.l.o.g. that $|L| \le |R|$, i.e., $|L| \le |A|/2$ and $|R| \ge |A|/2$. Observe that $\sum_{e\in L} |u_e - \bar\mu| = \sum_{e\in R} |u_e - \bar\mu| =: \ell$ by the definition of the average. We have $P_L \ge |L| \cdot (\frac{\ell}{|L|})^2 \ge 2\ell^2/|A|$.

If $P_L \ge \frac{1}{20}P_A$, we set $\eta = \bar\mu$, $A_T = R$, and A_S as the $|A|/8$ edges with smallest u_e-values in L (or the whole set L if $|L| < |A|/8$). Then all properties are fulfilled. Otherwise ($P_L < \frac{1}{20}P_A$, $P_R \ge \frac{19}{20}P_A$), we define

$$\begin{aligned} \eta &= \bar\mu + 4\ell/|A|, \\ A_T &= \{e \in A \mid u_e \le \eta\}, \quad \text{and} \\ R' &= \{e \in A \mid u_e \ge \bar\mu + 6\ell/|A|\} \ . \end{aligned}$$

Note that at most half of the elements in R are to the right of η as $\ell/|R| \le 2\ell/|A|$ is an upper bound on the *average distance* of elements of R to $\bar\mu$. This gives $|A_T| \ge |A|/2$. Further, note that every element $e \in R'$ has $|u_e - \eta| \ge \frac{1}{3}|u_e - \bar\mu|$. We choose A_S as the $|A|/8$ largest elements in R' (or $A_S = R'$ if $|R'| < |A|/8$).

It remains to show that this choice fulfills the fourth property. The potential $P_{R'}$ in the set R' is at least

$$\begin{aligned} P_{R'} &= P_R - P_{R\setminus R'} \ \ge \ P_R - |A|\Big(\frac{6\ell}{|A|}\Big)^2 \\ &\ge P_R - 18 \cdot \frac{2\ell^2}{|A|} \ \ge \ P_R - 18P_L \\ &\ge \frac{19}{20}P_A - \frac{18}{20}P_A \ \ge \ \frac{1}{20}P_A \end{aligned}$$

By choosing at most $|A|/8$ elements from R' we only decrease the potential by a factor of 4. Hence, the fourth property holds. □

Now, we compute a partial, fractional matching M between nodes X_{A_S} and nodes X_{A_T} in G', where A_T is the set of target edges. This is done via an approximate maxflow algorithm, which ensures that the matching can be routed in G' with constant congestion. Let for two edges $e, h \in A$, $m_{eh} = m_{he}$ denote the weight of the edge between x_e and x_h in the matching. The algorithm flips a fair random coin and either performs a *matching step* or a *deletion step*. We first describe the matching step.

Matching Step. We apply the partial fractional matching by averaging flow vectors similar to Khandekar et al. [15]. A slight extension of Lemma 3.3 in [15] gives the following :

LEMMA 3.4. *Applying the partial fractional matching M, the potential reduction is $\geq \frac{1}{2}\sum_{\{e,h\}} m_{eh} \cdot \|f_e - f_h\|_2^2$.*

Proof. Let x_i be an $|A|$-dimensional vector that contains the i-th components of all flow vectors for active edges. Further, let $\vec{\mu}_i = \mu_i \cdot \mathbb{1}$ be the $|A|$-dimensional vector that has value μ_i in every component. The total potential can be written as

$$\sum_{e \in A} \|f_e - \mu\|_2^2 = \sum_i \|x_i - \vec{\mu}_i\|_2^2 \ .$$

In order to analyze the potential reduction when applying a fractional perfect matching M (we can assume that M is perfect by adding self-loops) we focus on a single vector x_i and analyze the reduction in $\|x_i - \vec{\mu}_i\|_2^2$. Note that by applying the matching the vector x_i is transformed into vector $\frac{1}{2}x_i + \frac{1}{2}Mx_i$, and the vector μ (and, hence, μ_i) does not change. Therefore, the change in $\|x_i - \vec{\mu}_i\|_2^2$ is

$$\begin{aligned}
&\|x_i - \vec{\mu}_i\|_2^2 - \|\tfrac{1}{2}x_i + \tfrac{1}{2}Mx_i - \vec{\mu}_i\|_2^2 \\
&\quad = \|x_i\|_2^2 - 2\langle x_i, \vec{\mu}_i\rangle + \|\vec{\mu}_i\|_2^2 - \tfrac{1}{4}\|x_i + Mx_i\|_2^2 \\
&\qquad + \langle Mx_i, \vec{\mu}_i\rangle + \langle x_i, \vec{\mu}_i\rangle - \|\vec{\mu}_i\|_2^2 \\
&\quad = \|x_i\|_2^2 - \langle x_i, \vec{\mu}_i\rangle - \tfrac{1}{4}\|x_i + Mx_i\|_2^2 + \langle Mx_i, \vec{\mu}_i\rangle \\
&\quad = \tfrac{3}{4}\|x_i\|_2^2 - \langle x_i, \vec{\mu}_i\rangle - \tfrac{1}{2}\langle x_i, Mx_i\rangle - \tfrac{1}{4}\|Mx_i\|_2^2 \\
&\qquad + \langle Mx_i, \vec{\mu}_i\rangle \\
&\quad = \tfrac{1}{4}(\|x_i\|_2^2 - \|Mx_i\|_2^2) + \tfrac{1}{2}x_i^T(I - M)x_i \\
&\quad \geq \tfrac{1}{2}x_i^T(I - M)x_i \ = \ \tfrac{1}{2}\textstyle\sum_{\{e,h\}} m_{eh} \cdot ((x_i)_e - (x_i)_h)^2
\end{aligned}$$

Here, the fourth equality follows since $\langle x_i, \mathbb{1}\rangle = \langle Mx_i, \mathbb{1}\rangle$ as the total flow for the i-th commodity stays the same. The inequality comes from the fact that $\|x_i\|_2^2 \geq \|Mx_i\|_2^2$ which follows from the Rayleigh-Ritz theorem since the largest eigenvalue of M is at most 1. Summing the above over all i gives the lemma. □

Similar to [15], we use the following lemma on the Gaussian behavior of random projections, see [3].

LEMMA 3.5. *Let v be a vector in $\mathbb{R}^d$, and r be a random unit vector in $\mathbb{R}^d$. Then,*

$$\mathbf{E}[|v \cdot r|^2] = \|v\|_2^2/d$$

and for $x \leq d/16$:

$$\mathbf{Pr}[|v \cdot r|^2 \geq x \cdot \|v\|_2^2/d] \leq e^{-x/4} \ .$$

LEMMA 3.6. *With high probability, all pairs of edges $e, h \in A$ fulfill $n \cdot |u_e - u_h| \leq C \ln n \cdot \|f_e - f_h\|$.*

Proof. Note that $u_e - u_h$ is the projection of $f_e - f_h$ onto the random direction r. Choosing $x = C \ln n$ in Lemma 3.5 gives that the probability that a single pair does not fulfill the equation is at most $e^{-x/4} \leq n^{-C/4}$. Taking a union bound over all (at most n^2 pairs) gives the lemma. □

Now, we can estimate the potential decrease as follows:

$$\begin{aligned}
&\frac{1}{2}\sum_{e \in A_S}\sum_{h \in A_T} m_{eh}\|f_e - f_h\|_2^2 \\
&\quad \geq \frac{n}{2C\ln n}\sum_{e \in A_S}\sum_{h \in A_T} m_{eh}|u_e - u_h|^2 \\
&\quad \geq \frac{n}{2C\ln n}\sum_{e \in A_S}\sum_{h \in A_T} m_{eh}|u_e - \eta|^2 \\
&\quad \geq \frac{n}{18C\ln n}\sum_{e \in A_S}\sum_{h \in A_T} m_{eh}|u_e - \bar{\mu}|^2 \ . \qquad (3.2)
\end{aligned}$$

Here, the first inequality holds with high probability due to Lemma 3.6. The second inequality follows since source and target edges are chosen in such a way that η lies between u_e and u_h (Property 1 in Lemma 3.3). The final inequality is due to Property 3 in Lemma 3.3.

Deletion Step. For describing a deletion step we first need to describe how to find the fractional, perfect matching M. For this we add a super-source s and a super-target t to the graph G' and connect s to all source nodes X_{A_S} with edges of capacity 1, and t to all target nodes X_{A_T} with edges of capacity 1/2. All other edges are assigned a capacity of 2. We use G'_{st} to denote the resulting graph.

In G'_{st} we compute an approximate maxflow between s and t, i.e., we have a feasible flow of some value Z and a cut of capacity at most $(1+\varepsilon)Z$ with $\varepsilon = 1/\log^3 n$. Then, for every source edge $\{s, x_e\}$ that carries at least flow 1/2 we scale the corresponding flow paths to obtain a flow of 1 from s to x_e. We adjust capacities in G'_{st} so that this new flow becomes feasible; we let Z denote the value of the scaled flow. Note that the total capacity of cut edges is still at most $(1+\varepsilon)Z$.

After this rescaling, edge capacities in G'_{st} are still at most constant; the incoming flow along every target edge $\{x_h, t\}$ is at most 1; and the outgoing flow along a source edge (s, x_e) is either 1 or at most 1/2.

The flow can then be decomposed into a fractional, partial matching between source and target nodes. Note that the matching graph contains at most m non-zero edges as each edge corresponds to a flow path and the flow can be decomposed into at most m paths in time $\tilde{\mathcal{O}}(m)$, see [24].

Let C' denote the set of cut edges in G'_{st}. We use $C = \{e \in E \mid \{x_e, z\} \in C', z \in V(G'_{st})\}$, to denote the corresponding set of cut edges in G. The following lemma states that we can obtain an edge set that induces a balanced clustering by suitably combining edges in C with our current set $A \cup R$ of cut edges.

LEMMA 3.7. *Either the set $((A \cup R) \setminus A_S) \cup C$ or the set $((A \cup R) \setminus A_T) \cup C$ induces a balanced clustering.*

Proof. Consider the graph G'_{st} where we remove all the cut edges in C'. We distinguish two cases depending on whether s or t can reach more vertices in this graph. First, assume that we can reach at most half of the vertices (of V) from the super-source s. We show that $D := ((A \cup R) \setminus A_S) \cup C$ induces a balanced clustering.

Consider two nodes u and v that in G are in the same component w.r.t. D but in different components w.r.t. $A \cup R$. There must exist a path in G between u and v that w.r.t. $A \cup R$ is separated only by edges from $A_S \setminus C$ as A_S are the only edges that are removed for constructing D.

However, the fact that these edges are not in C means that in G' the super-source s can reach this path and can also reach u and v. But this means that u and v lie in a component that contains at most half of the vertices from V. This shows that D induces a balanced clustering.

The other case is analogous. □

Observe that

$$\begin{aligned} &|((A \cup R) \setminus A_T) \cup C| \\ &\quad \le \; |A| + |R| - |A_T| + |C| \\ &\quad \le \; |A|/2 + |R| + |C| \\ &\quad \le \; |A|/2 + |F|/\log n + 2(1+\varepsilon)Z \\ &\quad \le \; |A|/2 + |F|/\log n + |A|/4 + |A|/\log n \\ &\quad \le \; \frac{7}{8}|F| \;, \end{aligned}$$

for sufficiently large n. Hence, if $((A \cup R) \setminus A_T) \cup C$ induces a balanced clustering we can stop and Case 1 of the lemma holds. In the following we assume that $((A \cup R) \setminus A_S) \cup C$ induces a balanced clustering. We now describe how we move flow vectors. For every flow path in G'_{st} with weight $w \le 1$ between a source node x_e and a target node x_h we move a w-fraction of the sources flow vector to the *first* cut edge on the flow path (note that a flow path may cross the cut several times as it is just an approximate mincut). The fractional assignment of source vectors to cut edges can be computed in nearly linear time (analogous to extracting a fractional matching from a flow).

We then assign to an edge $h \in G$ all flow vectors that in G'_{st} are assigned to one of the edges incident to x_h. Note that an edge $h \in A \setminus A_S$ will have flow vectors of weight at least 1, as flow vectors from non-source edges do not move. If such an edge also gets assigned flow vectors from A_S these flow vectors are simply ignored.

Other edges sum the received flow vectors, and if their weight is more than 1 the flow vectors are rescaled to 1. Edges, for which the weight of assigned, fractional flow vectors is less than 1 do not receive a flow vector. The following lemma shows that the number of these edges is small.

LEMMA 3.8. *Let C_B denote the set of edges in C for which the total weight of assigned flow vectors sum to less than 1. Then $|C_B| \le 2\varepsilon Z \le \varepsilon|A|/4$.*

Proof. Observe that we only need to worry about cut edges in C that are not also target edges. We view the assignment of flow vectors, as a flow in G'_{st} from the source to the cut edges C'. The total value of the flow is Z. Every cut edge that is not of the form $\{x_h, t\}$ and that does not have flow 1 reaching it, has free capacity at least $\frac{1}{2}$ (in particular this holds for the source edges $\{s, x_e\}$ due to the rescaling). Hence, having more than $2\varepsilon Z$ of these edges would mean that more than εZ capacity among cut edges is unused. However, then the remaining capacity of the cut would not suffice to absorb a flow of value Z as the capacity is at most $(1+\varepsilon)Z$. □

We add all edges in C_B to the set B. Note that these are at most $\varepsilon|A|/4 \le |F|/\log^3 n$, i.e., over all iterations the cardinality of B cannot become larger than $|F|/\log n$. The set $(A - A_T) \cup (C \setminus C_B)$ is our new set of active edges. In the following we analyze the change in potential.

For every source edge e and cut edge $h \in C$, let w_{eh} be the fraction of flow vector f_e that is moved to h (after deleting flow vectors and rescaling). Note that if an edge receives a (fractional) flow vector from A_S it will receive flow vectors of total weight exactly one from A_S. Let D denote the set of these edges.

Note that for all $e \in A_S$ we have $\sum_{h \in D} w_{eh} \le \sum_{h \in A_T} m_{eh}$ since the movement is done along the "matching flow" leaving x_e. The difference in potential that is generated is at least

$$\begin{aligned}
&\sum_{e\in A_S}\|f_e-\mu\|_2^2-\sum_{h\in D}\|f_h-\mu\|_2^2\\
&\quad\geq \sum_{e\in A_S}\|f_e-\mu\|_2^2-\sum_{h\in D}\|\sum_{e\in A_S}w_{eh}(f_e-\mu)\|_2^2\\
&\quad\geq \sum_{e\in A_S}\|f_e-\mu\|_2^2-\sum_{h\in D}\sum_{e\in A_S}w_{eh}\|f_e-\mu\|_2^2\\
&\quad\geq \sum_{e\in A_S}\|f_e-\mu\|_2^2-\sum_{e\in A_S}\sum_{h\in A_T}m_{eh}\|f_e-\mu\|_2^2\\
&\quad= \sum_{e\in A_S}(1-\sum_{h\in A_T}m_{eh})\cdot\|f_e-\mu\|_2^2\\
&\quad\geq \frac{n}{C\ln n}\sum_{e\in A_S}\Big(1-\sum_{h\in A_T}m_{eh}\Big)\cdot|u_e-\bar{\mu}|^2 \qquad (3.3)
\end{aligned}$$

Here, the first inequality uses the fact that the new flow vector is $f_h=\sum_e w_{eh}f_e$. The second inequality follows from $\sum\|\alpha_i v_i\|_2^2\leq\sum\alpha_i\|v_i\|_2^2$ which holds for all vectors v_i and values α_i with $\sum\alpha_i=1$, $\alpha_i>0$. The last inequality follows from Lemma 3.6, and holds with high probability.

Equation 3.2 and Equation 3.3 give high probability lower bounds on the decrease in potential for a matching step and a deletion step, respectively. For both equations we introduce a random variable that describes the amount that we need to subtract from the right hand side such that the equations *always* hold. Let $z_{3.2}$ and $z_{3.3}$, denote these random variables, and observe that they are 0, w.h.p., and that they are polynomially bounded.

Recall that we choose a deletion step or a matching step with probability 1/2. Hence, we can bound the expected loss in potential by

$$\begin{aligned}
&\frac{1}{2}\cdot\frac{n}{18C\ln n}\sum_{e\in A_S}\mathbf{E}[\sum_{h\in A_T}m_{eh}|u_e-\bar{\mu}|^2]-\mathbf{E}[\tfrac{1}{2}z_{3.2}]+\\
&\frac{1}{2}\cdot\frac{n}{C\ln n}\sum_{e\in A_S}\mathbf{E}[(1-\sum_{h\in A_T}m_{eh})|u_e-\bar{\mu}|^2]-\mathbf{E}[\tfrac{1}{2}z_{3.3}]\\
&\quad\geq \frac{n}{36C\ln n}\sum_{e\in A_S}\mathbf{E}\left[|u_e-\bar{\mu}|^2\right]-\tfrac{1}{2}\mathbf{E}[z_{3.2}+z_{3.3}]\\
&\quad\geq \frac{n}{2880C\ln n}\sum_{e\in A}\mathbf{E}\left[|u_e-\bar{\mu}|^2\right]-\tfrac{1}{2}\mathbf{E}[z_{3.2}+z_{3.3}]\\
&\quad\geq \frac{1}{2880C\ln n}\sum_{e\in A}\|f_e-\mu\|_2^2-\tfrac{1}{2}\mathbf{E}[z_{3.2}+z_{3.3}]\\
&\quad= \Omega(\Phi(t)/\log n)\ .
\end{aligned}$$

The second inequality is due to Property 4 in Lemma 3.3, and the following equation is due to Lemma 3.5. The final equation follows since $\mathbf{E}[z_{3.2}+z_{3.3}]=o(1)$.

This shows that after $\mathcal{O}(\log^2 n)$ iterations we have a set $A\uplus R$ of edges such that A is $\Omega(1/\log^2 n)$-flow-linked. Further, $|B|\leq|F|/\log n$. If $|B|\leq 2|A|/\log n$ we have Case 2 of the lemma, otherwise $|A\uplus R|\leq\frac{7}{8}|F|$ for sufficiently large n, and we have Case 1. This finishes the proof of Lemma 3.1.

3.1.2 Proof of Lemma 3.2.

Denote by $P_1,\dots,P_p$, the components of the partition induced by F. We know that $|P_i|\leq\frac{3}{4}n$. Let X_C be the set obtained from Lemma 3.9 for $X_A=\{x_a\in V(G')|a\in A\}$ and $X_B=\{x_b\in V(G')|b\in B\}$. Let C be the edge set in G, corresponding to X_C in G'.

Claim 1. *Either $B\cup C$ or $A\cup C$ induces a balanced clustering.*

Proof. From Lemma 3.9, we know that C separates A from B. Denote by Q_A and Q_B, the parts containing A and B respectively. First observe that both $B\cup C$ and $A\cup C$ induce a clustering. If $|Q_A|\leq\frac{n}{2}$, $B\cup C$ induces a clustering $Q_A, P_1\cap Q_B,\dots,P_p\cap Q_B$. Moreover, this is a balanced clustering since $|Q_A\leq\frac{n}{2}|$ and $|P_i\cap Q_B|\leq|P_i|\leq\frac{3}{4}n$. Similarly, if $|Q_A|>\frac{n}{2}$, $A\cup C$ induces a balanced clustering. □

If $B\cup C$ induces a balanced clustering, we set $F_{\text{new}}=B\cup C$ and claim that we are in Case 1 of the lemma. This is true since $|B\cup C|\leq(2+\varepsilon)|B|\leq\frac{6}{\log n}|A|\leq\frac{3}{4}|F|$ for $\varepsilon\leq 1$ and n large enough.

If, on the other hand, $A\cup C$ induces a balanced clustering, we know from Lemma 3.9 that every node $x_c\in X_C$ sends 1 unit of flow and every node $x_a\in X_A$ receives at most 6 units of flow. We are in Case 2 with $F_{\text{new}}:=A\uplus(C\setminus A)$. This finishes the proof of Lemma 3.2.

3.2 The second partition.

The second partition of $G[S]$ depends on the first partition and on the set B of *boundary edges* of S. We add to the graph $G[S]$ a node x_e for every edge $e\in B$ and connect x_e to v, where $v\in e$ is the node from e that lies in S. These newly added edges are assigned a capacity of $1/\log n$. We use $\bar{G}[S]$ (or just $\bar{G}$ if S is clear from the context) to denote the resulting graph. We use $\bar{G}[S]'$ (or $\bar{G}'$) to denote the subdivision graph of $\bar{G}[S]$, where we added a split vertex to every edge from $G[S]$ (but *not* to the boundary edges).

Let $Z_1,\dots,Z_z$ denote the clustering obtained in the first partition and let F denote the set of *inter-cluster edges* (i.e., $F=\{\{u,v\}\mid u\in Z_i,v\in Z_j,i\neq j\}$). We define the second partition by a subset Y of edges from $\bar{G}[S]$ that are removed. This subset fulfills the following properties.

1. There is a flow in $\bar{G}'$ from nodes in X_Y to nodes in X_B, routable with congestion $\mathcal{O}(\log n)$, s.t.

- *every* node x_e, $e \in Y$ sends out $\mathrm{cap}(e)$ flow,
- a node x_b, $b \in B$ gets at most $6/\log n$ units of flow (recall that these edges have capacity $1/\log n$).

2. There is a flow in $\bar{G}'$ from nodes in X_Y to nodes in X_F, routable with congestion $\mathcal{O}(\log n)$, s.t.
   - *every* node x_e, $e \in Y$ sends out $\mathrm{cap}(e)$ flow,
   - a node x_f, $f \in F$ gets at most 6 units of flow (recall that these edges have capacity 1).
3. The nodes in X_F are separated from nodes in X_B. This means every path in $\bar{G}'$ that starts at $x_b \in X_B$ and ends at $x_f \in X_F$ contains a node from X_Y (which may be x_b or x_f).

In order to find X_Y and the flows we apply the following more general lemma for sets X_F and X_B.

LEMMA 3.9. *Consider two sets of split vertices X_A and X_B in G'. We can find a set X_Y of split-vertices such that the following holds*

- *The set X_Y separates X_A from X_B. This means every path between nodes $x_b \in B$ and $x_a \in X_A$ must contain a vertex $x_e \in X_C$.*
- *There exists a multicommodity-flow in G' from nodes in X_Y to nodes in X_A that can be routed with congestion $\mathcal{O}(\log n)$ s.t.*
  - *Every node $x_e \in X_Y$ sends out $\mathrm{cap}(e)$ units of flow.*
  - *Every node $x_a \in X_A$ receives at most $6\mathrm{cap}(e)$ flow.*
- *There exists a multicommodity-flow G' from nodes in X_Y to nodes in X_B that can be routed with congestion $\mathcal{O}(\log n)$ s.t.*
  - *Every node $x_e \in X_Y$ sends out $\mathrm{cap}(e)$ units of flow.*
  - *Every node $x_b \in X_B$ receives at most $6\mathrm{cap}(e)$ flow.*

The set X_Y and the flows can be computed in almost linear time.

Proof. We construct a graph G'_{st} by adding a super-source s and connect it to every node $x_a \in A$ via an edge of capacity $\mathrm{cap}(a)$, and a super-source t that we connect to every vertex $x_b \in B$ with a set of capacity $\mathrm{cap}(b)$. Now, we compute an approximate maxflow. This means we obtain a flow from s to t with value Z and a cut separating s from t with capacity at most $(1+\varepsilon)Z$. Let Y_1 denote the set of cut edges in this cut. With a slight abuse of notation we define the set X_{Y_1} as the set of *split-vertices* that are incident to an edge in Y_1.

We decompose the flow into flow paths. After this we assign every path to a single vertex $x_y \in X_{Y_1}$ on the path. This could, e.g., be the first such vertex on the path. We call a node $x_e \in X_{Y_1}$ satisfied if the total weight of flow paths assigned to it is at least $\mathrm{cap}(e)/2$. We now delete all satisfied nodes and their incident edges (at most 3 for every node) from G'_{st}. We further adjust the capacities of remaining edges that are incident to either X_B or X_A. For every node $x_a \in X_A$ ($x_b \in X_b$), we reduce the capacities of all edges incident to x_a (x_b) by the amount of flow that is sent along $\{s, x_a\}$ ($\{x_b, t\}$). Then we iterate on the remaining graph.

In the end we use X_Y to be the set of nodes removed in all iterations.

CLAIM 2. *There are at most $\mathcal{O}(\log n)$ iterations.*

Proof. The total capacity of edges that correspond to *satisfied* nodes x_y is at least $Z/2$. To see this observe that the total capacity of *unsatisfied nodes* can be at most $2\varepsilon Z$ as each of these edges e corresponds to an unused capacity of $\mathrm{cap}(e)/2$ over the cut. However, the total unused capacity is at most εZ. Hence, the total capacity of edges corresponding to satisfied nodes is at least $(1+\varepsilon)Z - 2\varepsilon Z = (1-\varepsilon)Z \geq Z/2$ for $\varepsilon \leq 1/2$.

By removing these edges we reduce the mincut between s and t by a constant factor in every iteration. This gives the lemma. □

Combining the flow from all iterations gives a flow that can be routed with congestion $\mathcal{O}(\log n)$. For nodes $x_e \in X_Y \setminus (X_A \cup X_B)$ we have flow paths exclusively assigned to x_e of total value at least $\mathrm{cap}(e)/2$ (as the capacity of the incident edges does not change during the algorithm).

However, this might not be true for nodes in X_A or X_B, as by the time we remove e.g. x_a (which guarantees a flow of at least $\mathrm{cap}(a)/2$ assigned to x_a) the capacity might already have been reduced a lot. Therefore we change the assignment of flow paths to vertices. In addition to assigning a path to the first vertex from X_Y on the path we also assign it to vertices x_a and x_b where these are the first and the last vertex, respectively, on the path (apart from s and t, of course). Then, also vertices in $x_e \in X_A \cup X_B$ are assigned paths of total value equal to at least $\mathrm{cap}(e)/2$.

If every vertex routes all the flow paths assigned to it we have a congestion of $\mathcal{O}(\log n)$, as any flow path is used at most 3 times, and every vertex $x_e \in X_Y$ can send $\mathrm{cap}(e)/2$ to X_A and to X_B. Also every vertex $x_e \in X_A \cup X_B$ sends or receives at most $3 \cdot \mathrm{cap}(e)$. Scaling this flow by a factor of two gives the lemma. □

3.3 Further remarks. PartitionA runs in time $\tilde{\mathcal{O}}(T(m_S, \varepsilon))$, where m_S is the number of edges in the input cluster S. Even though we invoke PartitionA a linear number of times, summing the size of all instances gives at most $\mathcal{O}(m \log m)$ due to the logarithmic height of the decomposition tree. So the total time taken by our cutting scheme is $\tilde{\mathcal{O}}(T(m, \varepsilon) \cdot \log m) = \tilde{\mathcal{O}}(T(m, \varepsilon))$.

For constructing a vertex sparsifier w.r.t. a subset X of terminals, we proceed as follows. In each level of the partitioning, we make sure that a sub-cluster of S contains at most $\frac{3}{4}\tau(S)$ terminals, where $\tau(S)$ denotes the number of terminals in S. The partitioning stops if $\tau(S) \leq 1$. This generates a decomposition tree of height $\mathcal{O}(\log k)$, where $k = |X|$. In Section 4, we show that the decomposition tree T obtained in this section is a flow sparsifier of quality $\mathcal{O}(\log^4 n)$. Here, an $\mathcal{O}(\log^2 n)$ factor is due to the height of T, which is $\mathcal{O}(\log n)$. Having a tree of height $\mathcal{O}(\log k)$ gives rise to a flow sparsifier of quality $\mathcal{O}(\log^2 n \log^2 k)$.

4 The Routing Scheme

In this section we describe how we can use the hierarchical decomposition from the previous section to design an oblivious routing scheme with competitive ratio $\mathcal{O}(\log^4 n)$.

In the following we assume that we are given a demand D_{st} between every pair of vertices s, t from G and the task is to route these demands with close-to optimal congestion. We will show how to do this using the hierarchical decomposition. Note, that even though we assume that we know the demands the actual routing paths that we construct are independent of these and, hence, our routing scheme is oblivious. We can assume w.l.o.g. that the optimum congestion for the given demands is 1 as scaling demands does not affect the competitive ratio.

In order to specify a routing path between s and t we will assume that both s and t send out a message (of size D_{st}). In several iterations these messages are split and distributed to different vertices (this is done independent of other demands). When at some point the distribution of the source and target-message are the same we can construct a flow from s to t by reverting the routing paths of the target-message.

Consider the decomposition tree for our hierarchical decomposition from the previous section. Every two-level cut from the previous section corresponds to a cluster S that is first partitioned into sets L and R that are then partitioned into sets $L_1, \dots, L_\ell$ and $R_1, \dots, R_r$, respectively. The edges incident (in G!) to the lower level components are in the set F of *inter cluster edges*, the set Y of *cut edges*, and the set B of *boundary edges* (note that these sets are not necessarily disjoint).

4.1 Routing to the boundary of S.

In the following we describe a routine that forms a building block for our routing scheme and that moves messages originating in a cluster S to the boundary of S. It will be more convenient to describe the routine on the graph $\bar{G}[S]'$. Initially, all (remaining) messages originating inside S are assumed to reside at nodes $X_F \cup X_Y \cup X_B$. This can e.g. be obtained by first running the routine for sub-clusters $L_1, \dots, L_\ell$ and $R_1, \dots, R_r$. The routine takes these messages; pairs some of these messages up (by routing a source-message and its corresponding target-message to the same distribution over vertices); and finally moves all remaining messages to the boundary of S. We show that we can do this with congestion $\mathcal{O}(\alpha \log^3 n)$, just using edges inside S.

Here $\alpha \geq 1$ is an upper bound on the amount of flow that has been pushed to a boundary node x_b when running the routine for the sub-clusters $L_1, \dots, L_\ell$ and $R_1, \dots, R_r$. Note that this means that initially every boundary node x_b in the graph $\bar{G}[S]'$ has load at most α (only counting load for messages that originate in S), while other nodes may have load 2α because the lower level clusters pushed flow to it from two sides.

We will show that α is constant for every cluster. Then the total load induced on an edge by running our routine for every cluster is at most $\mathcal{O}(\log^4 n)$ as an edge is contained in at most $\mathcal{O}(\log n)$ clusters due to the logarithmic height of the decomposition.

Removing the cut edges Y from $\bar{G}[S]$ partitions the vertex set S into two parts; the vertices that in $\bar{G}[S] \setminus Y$ can reach a boundary node $x_b \in X_B \setminus X_Y$ and those that cannot. We use L to denote the vertices that cannot reach the boundary and R are the remaining vertices. We will refer to L also as the *core* of S.

LEMMA 4.1. *We can route messages for which source and target lie in the core L with congestion $\mathcal{O}(\alpha \cdot \log^3 n)$.*

Proof. In the beginning any message that originates in the core resides at the border of its sub-cluster. Such a node must be in $X_Y \cup X_F$ as in lower levels no edge in Y could have been used for routing and the edges in Y separate the nodes in L from the boundary.

We can now use Property 2 for the set Y (see Section 3.2) and route the flow that resides at vertices in X_Y (and starts in L) to vertices in X_F. Property 2 says that exists a flow which can be routed with congestion $\mathcal{O}(\log n)$ such that every node in X_Y sends a flow that is equal to the capacity of the corresponding edge in $\bar{G}[S]$, which is at least $1/\log n$, and every node in X_F receives at most 6 units of flow. A node in X_Y has at most flow 2α in the beginning. Hence, we can route the fraction of this flow that originates in L to nodes in X_F with congestion at most $\mathcal{O}(\alpha \log^2 n)$. Any node in X_F

receives at most $12\alpha \log n$ units of flow.

After this step we route an all-to-all multicommodity flow problem on the set X_F. Since the set is $\Omega(1/\log^2 n)$-flow-linked we can do this with congestion $\mathcal{O}(\alpha \log^3 n)$. Now, all messages that originate in L are distributed uniformly among vertices in X_F. Hence, we can pair up source- and target-messages if both, the source and the target, lie in L. □

After routing all messages that start and end in the core, we distribute the remaining messages to nodes in X_Y.

Lemma 4.2. *Messages that originate in the core L but for which the counterpart lies outside of L, can be sent to nodes X_Y s.t. each node in X_Y receives a flow of at most 1. This can be done with congestion $\mathcal{O}(\log^3 n)$.*

Proof. After the routing of Lemma 4.1 the messages that originate in L and remain are uniformly distributed among nodes in X_F, and either want to leave or enter the core L. However, Y is a super-set of the edges that leave the core in G. Hence, the total demand of these messages can be at most $|Y|$ as otherwise an optimal algorithm could not solve the flow problem with congestion 1.

Similar to Lemma 4.1 we can send one unit of flow from every node in X_Y to a uniform distribution over X_F with congestion $\mathcal{O}(\log^3 n)$. Reversing this flow distributes the remaining messages from L uniformly among nodes in X_Y. □

Observation 1. *After performing the routing as described in Lemma 4.1 and Lemma 4.2 the nodes in $X_F \setminus X_Y$ do not carry any flow.*

Proof. A node $x_f \in X_F \setminus X_Y$ must be incident to two vertices $u, v \in L$ as otw. either u or v could reach a boundary node (after removing Y) and, hence, also x_f could reach the boundary, which is a contradiction as X_Y separates X_F from X_B. □

The above observation means that we are now in a situation where only vertices in $X_B \cup X_Y$ carry flow. Each vertex in $X_B \setminus X_Y$ carries at most α (it had $\leq \alpha$ in the beginning; and nothing changed); a vertex $X_B \cup X_Y$ carries at most 1 as it only carries flow originating in L; and a vertex in $X_Y \setminus X_B$ carries at most $\alpha + 1$ as it started with α and at most 1 has been added by the routing of Lemma 4.2.

Observation 2. *Assume that each node $X_Y \setminus X_B$ has flow at most $1+\alpha$. We can push this flow to boundary nodes X_B such that each node $x_b \in X_B$ receives at most $12\alpha/\log n$. This can be done with congestion $\mathcal{O}(\alpha \log n)$.*

Proof. Property 1 ensures that there exists a flow with congestion $\mathcal{O}(\log n)$ in $\bar{G}[S]'$ s.t. every node $x_e \in X_Y$ sends 1 unit of flow and every node $x_b \in X_B$ receives at most $6/\log n$ units of flow. Scaling this flow by a factor $(\alpha + 1) \leq 2\alpha$ gives the desired flow. □

Theorem 4.1. *The constructed routing scheme is oblivious and has competitive ratio $\mathcal{O}(\log^4 n)$. Furthermore, any demand that can be routed on the decomposition tree T corresponding to the hierarchical decomposition from Section 3 can be routed in G with congestion $\mathcal{O}(\log^4 n)$.*

Proof. On the lowest level we have clusters that contain just single vertices of V. It is easy to move messages originating at a vertex $v \in V$ to one of the incident edges such that a border node x_b receives at most a flow of 1. Hence, on the lowest level $\alpha = 1$. From Observation 2 the flow α pushed to a boundary node increases by a factor of at most $(1 + 12/\log n)$ in every level. Since, the height of the tree is logarithmic we have $\alpha = \mathcal{O}(1)$.

Together, with the observation that we only have to run our routine once for every cluster, and the fact that an edge is contained in at most $\mathcal{O}(\log n)$ clusters gives the bound on the competitive ratio.

For the second part observe, that the only constraint that we use to restrict the demands, is the fact that we require that the total demand that leaves or enters the core L of a cluster S is at most the capacity of edges leaving L in G. Since, L appears as a set in the hierarchical decomposition, the corresponding parent edge has capacity equal to the number of edges leaving L in G, and, hence, any feasible flow on the tree T will obey this constraint. □

References

[1] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph (Seffi) Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms*, 2(4):640–660, 2006.

[2] Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce Maggs, and Adam Meyerson. Simultaneous source location. *ACM Transactions on Algorithms*, 6(1):Article 16, 2009.

[3] Keith Ball. An elementary introduction to modern convex geometry. In Silvio Levy, editor, *Flavors of Geometry*, volume 31 of *Mathematical Sciences Research Institute Publications*, pages 1–58. Cambridge University Press, 1997.

[4] Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph (Seffi) Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS'11)*, pages 17–26, 2011.

[5] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'03)*, pages 24–33, 2003.

[6] Moses Charikar, F. Thomson Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 265–274, 2010.

[7] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04)*, pages 156–165, 2004.

[8] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC'05)*, pages 183–192, 2005.

[9] Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC'12)*, pages 673–688, 2012.

[10] Roee Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph (Seffi) Naor. Cut problems in graphs with a budget constraint. *Journal of Discrete Algorithms*, 5(2):262–279, 2007.

[11] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. In *Proceedings of the 13th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'10)*, volume 6302 of *Lecture Notes in Computer Science*, pages 152–165, 2010.

[12] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'03)*, pages 34–43, 2003.

[13] Jonathan A. Kelner, Lorenzo Orecchia, Yin Tat Lee, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, 2014.

[14] Rohit Khandekar, Guy Kortsarz, and Vahab Mirrokni. Advantage of overlapping clusters for minimizing conductance. In *Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN'12)*, volume 7256 of *Lecture Notes in Computer Science*, pages 494–505, 2012.

[15] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM*, 56(4):Artikel 19, 2009.

[16] Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. *Algorithmica*, 59(4):489–509, 2011.

[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, 2010.

[18] Aleksander Mądry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 245–254, 2010.

[19] Konstantin Makarychev and Yuri Makarychev. Metric extension operators, vertex sparsifiers and Lipschitz extendability. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 255–264, 2010.

[20] Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS'09)*, pages 3–12, 2009.

[21] Harald Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 43–52, 2002.

[22] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC'08)*, pages 255–264, 2008.

[23] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS'13)*, 2013.

[24] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.