# Vertex Sparsification for Edge Connectivity *

Parinya Chalermsook† Syamantak Das‡ Yunbum Kook§ Bundit Laekhanukit¶
Yang P. Liu|| Richard Peng** Mark Sellke†† Daniel Vaz‡‡

February 19, 2024

**Abstract**

*Graph compression* or *sparsification* is a basic information-theoretic and computational question. A major open problem in this research area is whether $(1+\epsilon)$-approximate cut-preserving vertex sparsifiers with size close to the number of terminals exist. As a step towards this goal, we study a thresholded version of the problem: for a given parameter $c$, find a smaller graph, which we call *connectivity-c mimicking network*, which preserves connectivity among $k$ terminals exactly up to the value of $c$. We show that connectivity-$c$ mimicking networks with $O(kc^4)$ edges exist and can be found in time $m(c \log n)^{O(c)}$. We also give a separate algorithm that constructs such graphs with $k \cdot O(c)^{2c}$ edges in time $mc^{O(c)} \log^{O(1)} n$.

These results lead to the first data structures for answering fully dynamic offline $c$-edge-connectivity queries for $c \geq 4$ in polylogarithmic time per query, as well as more efficient algorithms for survivable network design on bounded treewidth graphs.

*Final version available at doi:10.1137/1.9781611976465.74

†Aalto University, Finland parinya.chalermsook@aalto.fi

‡Indraprastha Institute of Information Technology Delhi, India syamantak@iiitd.ac.in

§KAIST, South Korea and Discrete Mathematics Group, Institute for Basic Science, South Korea yb.kook@kaist.ac.kr. Part of this work was done while visiting Georgia Tech.

¶Shanghai University of Finance and Economics, China bundit@sufe.edu.cn

||Stanford University yangpliu@stanford.edu

**Georgia Tech richard.peng@gmail.com. Part of this work was done while visiting MSR Redmond.

††Stanford University msellke@stanford.edu

‡‡Operations Research Group, Technische Universität München, Germany, daniel.vaz@tum.de. Part of this work was done while being at MPI Informatik and while visiting Aalto University.

# 1 Introduction

*Graph compression* or *sparsification* is a basic information-theoretic and computational question of the following nature: can we compute a "compact" representation of a graph, with fewer vertices or edges, that preserves important information? Important examples include spanners, which preserve distances approximately up to a multiplicative factor, and cut and spectral sparsifiers [BK96, ST04], which preserve cuts and the Laplacian spectrum up to an approximation factor of $(1+\epsilon)$. Such edge sparsifiers allow us to reduce several algorithmic problems on dense graphs to those on sparse graphs, at the cost of a $(1+\epsilon)$ approximation factor. On the other hand, some computational tasks, such as routing or graph partitioning, require reducing the number of vertices (instead of edges), that is, *vertex sparsification*.

The notion of vertex sparsification we consider here is that of *cut sparsification*, introduced by [HKNR98, Moi09, LM10]. In this setting, we are given an edge-capacitated graph $G$ and a subset $\mathcal{T} \subseteq V(G)$ of $k$ vertices called *terminals*, and we want to construct a smaller graph $H$ that maintains all the minimum cuts between every pair of subsets of $\mathcal{T}$ up to a multiplicative factor $q$, called the *quality of the sparsifier*. More formally, we want to find a graph $H$ which contains $\mathcal{T}$ as well as possibly additional vertices, such that for any $S \subseteq \mathcal{T}$, the minimum cut between $S$ and $\mathcal{T} \setminus S$ in $G$ and $H$ agree up to the multiplicative factor of $q$. Ideally, the size of the sparsifier (i.e., $|V(H)|$) should only depend on $|\mathcal{T}|$ and not the size of $G$.

There have been several results regarding tradeoffs between the quality $q$ and the size of cut sparsifiers. One line of work considers the case where the sparsifier $H$ has no additional vertices beyond the terminals. Here, an upper bound of $O(\log k/\log\log k)$ [Moi09, LM10, CLLM10, EGK+10, MM10] and lower bound of $\Omega(\sqrt{\log k}/\log\log k)$ [MM10] are known. In a different direction, quality 1-sparsifiers (known as *mimicking networks*) with $2^{2^k}$ vertices were shown to exist [HKNR98, KR14], and a lower bound of $2^{\Omega(k)}$ is also known [KR13]. Also, Chuzhoy [Chu12] studied the problem of obtaining the best possible trade-offs between quality and size of sparsifiers, and showed that quality-3 sparsifiers with $O(Z^3)$ vertices exist, where $Z$ is the total capacity of all terminals. A major open problem is whether a quality-$(1+\epsilon)$ cut sparsifier of size $\widetilde{O}(k/\text{poly}(\epsilon))$ exists; so far, this is only known for special graph classes, such as quasi-bipartite graphs [AGK14, ADK+16].

The aim of this paper is to study a related graph sparsifier that is suitable for applications in designing fast algorithms for *connectivity problems*. In particular, we consider the following problem: given an edge-capacitated graph $G$ with $k$ terminals $\mathcal{T}$ and a constant $c$, construct a graph $H$ with $\mathcal{T} \subseteq V(H)$ that maintains all minimum cuts up to size $c$ among terminals. Precisely, we want, for all subsets $S \subseteq \mathcal{T}$, that $\min(c, \text{mincut}_G(S, \mathcal{T}\setminus S)) = \min(c, \text{mincut}_H(S, \mathcal{T}\setminus S))$, where, for disjoint subsets $A, B \subseteq V(G)$, we define $\text{mincut}_G(A, B)$ as the value of a minimum cut between $A$ and $B$ in graph $G$. In this case, we call $H$ a *connectivity-c mimicking network* of $G$ (See Definition 2.1).

Our main result (Theorem 1.1) shows that every graph $G$ with integer edge capacities admits a connectivity-$c$ mimicking network with $O(kc^4)$ edges (so $O(kc^4)$ vertices as well), and we show a near-linear time algorithm to compute it.

**Theorem 1.1** *Given any edge-capacitated graph $G$ with $n$ vertices, $m$ edges, along with a set $\mathcal{T}$ of $k$ terminals and a value $c$, there are algorithms that construct a connectivity-c mimicking network $H$ of $G$ with*

1. *$O(kc^4)$ edges in time $O(m \cdot (c \log n)^{O(c)})$,*
2. *$k \cdot O(c)^{2c}$ edges in time $O(m \cdot c^{O(c)} \log^{O(1)} n)$.*

In fact, the algorithm for Part 1 constructs the optimal *contraction-based mimicking network*, so any existential improvement to the size bound of such mimicking networks would immediately translate to an efficient algorithm.[1] Our second algorithm is more efficient, while blowing up the size of the mimicking network obtained. We believe that our dependence on $c$ is suboptimal – we were only able to construct instances that require at least $2kc$ edges in the connectivity-$c$ mimicking network, and are inclined to believe that an upper bound of $O(kc)$ is likely.

Theorem 1.1 has direct applications in fixed-parameter tractability and dynamic graph data structures (see Sections 1.1 and 6). In fact, our results and techniques have already been used to give a deterministic $n^{o(1)}$ update time fully dynamic algorithm for $c$-connectivity for all $c = o(\log n)$ [JS20]. Additionally, our results are motivated in part by elimination-based graph algorithms [KLP+16, KS16], which we discuss in Section 1.2. In this way, we believe that achieving $(1+\epsilon)$-quality cut sparsifiers of size $\widetilde{O}(k/\text{poly}(\epsilon))$, an analogue of approximate Schur complements for cuts, may have broad applications in graph algorithms and data structures.

## 1.1 Our Results

Our proof of existence of connectivity-$c$ mimicking networks with $O(kc^4)$ edges (and thus $O(kc^4)$ vertices as well) involves extending the recursive approach of [Chu12] using *well-linked sets* to a thresholded setting. The construction of [Chu12] for cut sparsifiers maintains a partition of the vertices of the graph $G$. For each partition piece, the algorithm either finds a sparse cut to recurse on, or contracts the piece. [Chu12] then bounds the deterioration in quality from these contractions. The main differences between our approach and [Chu12] are:

- We introduce an extension of well-linkedness to a thresholded $c$-connectivity setting.
- We do not run the recursion all the way down. Instead, we use a kernelization result on mimicking networks via gammoid representative sets [KW12] to bottom out the recursion.

Additionally, in order to obtain near-linear running times for our constructions, we combine the *expander decomposition* technique [SW19] with several other combinatorial results that allow us to build the desired connectivity-$c$ mimicking network.

We would like to note that the results of [KW12, FLPS16] already give connectivity-$c$ mimicking networks of size $\text{poly}(k, c)$. However, the dependence on $k$ is at least quadratic, and the algorithms for computing them run in at least quadratic time, as these results use linear algebra on matroids. Therefore, their results do not give more efficient algorithms for the applications of dynamic connectivity and subset $c$-EC below.

Theorem 1.1 has applications in data structures for dynamic edge connectivity. The problem of dynamic $c$-edge-connectivity is to design an algorithm which supports edge additions, deletions, and $c$-edge-connectivity queries between pairs of vertices as efficiently as possible, preferably in

[1] Formally, if there exists a connectivity-$c$ mimicking network with $kf(c)$ edges that can be obtained by only contracting edges in $G$, for some function $f$, then our algorithm finds a mimicking network with at most $O(kf(c))$ edges.

nearly constant $\widetilde{O}(1)$[2] amortized update time. For online fully dynamic algorithms, such results are only known for $c \leq 3$ [HK99, HdLT01]. Even in the simpler *offline* model introduced by Eppstein [Epp94], where the algorithm sees all queries at the beginning, the only result for $c \geq 4$ is, to our knowledge, an unpublished offline fully dynamic algorithm for $c = 4, 5$ by Molina and Sandlund [MS18], which requires about $\sqrt{n}$ time per query. The fact that even offline algorithms are not known for dynamic $c$-connectivity when $c > 5$ shows a serious gap in understanding of dynamic flow algorithms. We make significant progress towards shrinking this gap, and show in Section 6.1 that, by combining Theorem 1.1 Part 2 with a divide and conquer algorithm for processing queries, we achieve nearly constant amortized time for offline fully dynamic $c$-edge-connectivity.

**Theorem 1.2** *There is an offline algorithm that on an initially empty graph $G$ answers $q$ edge insertion, deletion, and $c$-connectivity queries between arbitrary pairs of vertices in amortized $\widetilde{O}(c^{O(c)})$ time per query.*

Finally, connectivity-$c$ mimicking networks are perhaps the most natural object that can be used to "pass along" connectivity information between sub-problems in the dynamic programming framework. We illustrate this concept by presenting an additional application. The Subset $c$-Edge-Connectivity (or Subset $c$-EC) problem is the following: given a graph $G = (V, E)$ with costs on edges, and a terminal set, find the cheapest subgraph $H$ in which every pair of terminals is $c$-connected. We show in Section 6.2 that Theorem 1.1 speeds up the running time for solving this problem in low treewidth graphs.

**Theorem 1.3** *There is an algorithm that exactly solves* Subset $c$-EC *on an input graph $G$ with $n$ vertices in time $n \exp\left(O(c^4 \operatorname{tw}(G) \log(\operatorname{tw}(G)c))\right)$, where $\operatorname{tw}(G)$ denotes the treewidth of $G$.*

This is an improvement over [CDE+18] in which the running time was doubly-exponential in both $c$ and $\operatorname{tw}(G)$. Furthermore, the existence of a conditional lower bound of $(3-\epsilon)^{\operatorname{tw}(G)}$ even when $c = 1$, under the assumption of the strong exponential time hypothesis, implies that the dependence of our running time on $\operatorname{tw}(G)$ is almost optimal. Additionally, our dynamic programming based algorithm shows that any improvement to the edge bound $O(kc^4)$ in Theorem 1.1 gives an improvement for Theorem 1.3.

### 1.2 Related Work

We believe that our work has potential connections to dynamic data structures, elimination-based graph algorithms, and approximation algorithms and sparsification.

**Static and Dynamic $c$-Edge-Connectivity Algorithms.** The study of efficient algorithms for computing graph connectivity has a long history, including the study of max-flow algorithms [GT14], near-linear time algorithms for computing global min-cut [Kar00], and most recently, progress in exact [Mad13, Mad16, CMSV17] and approximate max-flow algorithms [She17, Pen16, KLOS14, She13]. The $c$-limited edge connectivity case can be solved in $O(mc)$ time statically, and is also implied by $\epsilon$-approximate routines by setting $\epsilon < 1/c$. As a result, it is a natural starting point for developing routines that can answer multiple flow queries on the same graph.

[2] Throughout, we use $\widetilde{O}(\cdot)$ to hide $\operatorname{poly}\log(n)$ factors. In particular, $\widetilde{O}(1) = \operatorname{poly}\log(n)$.

The question of computing max-flow between multiple pairs of terminal vertices dates back to the Gomory-Hu tree [GH61], which gives a tree representation of all $s$-$t$ min-cuts. However, such tree structures do not extend to arbitrary subsets of vertices, and to date, have proven difficult to maintain dynamically. As a result, previous works on computing cuts between a subset of vertices have gone through the use of tree-packing based certificates. These include results on computing the minimum cut separating terminals [CH03], as well as the construction of $c$-limited Gomory-Hu trees [HKP07, BHKP08].

Such results are in turn used to compute max-flow between multiple pairs of vertices [AKT19]. These problems have received much attention in fine grained complexity, since their directed versions are difficult [AWY15, AW14], and it is not known whether computing $(1+\epsilon)$-approximate versions of these is possible. From this perspective, the $c$-limited version is a natural starting point towards understanding the difficulty of computing $(1+\epsilon)$-approximate all-pairs max-flows in both static and dynamic graphs.

For the problem of finding a connectivity-$c$ mimicking network, the construction time of these vertex sparsifiers is critical for their use in data structures [PSS19]. For a moderate number of terminals (e.g., $k = n^{0.1}$), nearly-linear time constructions of vertex sparsifiers with $\text{poly}(k)$ vertices were previously known only when $c \leq 5$ [PSS19, MS18]. To our knowledge, the only results for maintaining exact $c$ connectivity for $c \geq 4$ are incremental algorithms [DV94, DV95, DW98, GHT16], in addition to the aforementioned fully dynamic algorithm for $c = 4, 5$ by Molina and Sandlund [MS18], which took about $\sqrt{n}$ time per query.

Furthermore, since this work was originally released, the concept of connectivity-$c$ mimicking networks along with the techniques of this work has been used to design deterministic $n^{o(1)}$ time fully dynamic algorithms for exact $c$-connectivity for all $c = o(\log n)$ [JS20].

**Elimination-based graph algorithms:** The study of connectivity-$c$ mimicking networks in this paper can also be viewed in the context of vertex reduction / elimination based graph algorithms. Such algorithms are closely related to the widely used and highly practically effective multigrid methods, which until very recently have been viewed as heuristics with unproven bounds. Even in the static setting, the only worst-case bounds for multi-grid and elimination based algorithms have been in the setting of linear systems [KLP+16, KS16], by utilizing a combination of vertex and edge sparsifications. Compared to the tree-like Laplacian solvers, sparse vertex elimination has a multitude of advantages: they are readily parallelizable [KLP+16], and can be more easily adapted to data structures that handle dynamic graphs [DKP+17, DGGP19].

Important properties of such routines is that the size of sparsifier is linear in the number of terminals, the construction can be computed in nearly linear time, and they are $(1+\epsilon)$-quality approximations. In particular, guaranteeing a $(1+\epsilon)$ approximation is essential as there are often multiples stages in elimination algorithms, so losing $\omega(1)$-quality at each stage would be detrimental. Our vertex-elimination routine combines all these properties and thus meets all criteria of previous elimination based routines [KLP+16, KS16]. On the other hand, most of the work on vertex sparsification to date has been on shortest path metrics [vdBS19, Che18], and/or utilizes algebraic techniques [vdBS19, KW12, FHKQ16]. As a result, these routines, when interpreted as vertex elimination routines, either incur errors, or have size super-linear in the number of terminals.

**Other notions of approximate sparsification.** Without using any additional vertices, the best known upper and lower bounds on the quality of vertex cut sparsifiers are $O(\log k/\log\log k)$ [CLLM10, MM10] and $\Omega(\sqrt{\log k}/\log\log k)$ [MM10] respectively. [Chu12] presents a quality-$O(1)$, size-$O(C^3)$ sparsifier, computable in time $\text{poly}(n)\cdot 2^C$, where $C$ denotes the total capacity of the edges incident on the terminals. It is open whether there are quality-$(1+\epsilon)$ and size $\text{poly}(k/\epsilon)$ vertex sparsifiers for edge connectivity, and we see Theorem 1.1 as a first step towards achieving this goal.

Additionally, there has been significant work on vertex sparsification in approximation algorithms [MM10, CLLM10, EGK+14, KR17, KW12, AKLT15, FKQ16, FHKQ16, GR16, GHP17a]. Recently, vertex sparsifiers were also shown to be closely connected with dynamic graph data structures [GHP17c, PSS19, GHP18, DGGP19].

There has also been work on mimicking networks on special graph classes. Krauthgamer and Rika [KR13] presented a mimicking network of size $O(k^2 2^{2k})$ for planar graphs with $k$ terminals, nearly matching the lower bound [KPZP18]. When all terminals lie on the same face, mimicking networks of size $O(k^2)$ are known [GHP17b, KR17]. An upper bound of $O(k\cdot 2^{2^{\text{tw}(G)}})$ is known for bounded-treewidth graphs [CSWZ00].

### 1.3 Structure of the Paper

In Section 2, we give preliminaries for our algorithms. In Section 3, we sketch our approach to the main results, the existence of a connectivity-$c$ mimicking network with $O(kc^4)$ edges and an algorithm to construct connectivity-$c$ mimicking networks of the optimal size, which we elaborate in detail in Section 4. In Section 5, we take a different approach to make an algorithm more efficient. We finalize our paper with detailed explanation on applications in Section 6.

## 2 Preliminaries

Our focus will be on cuts with at most $c$ edges. Our algorithms will involve contractions, which naturally lead to multigraphs. Therefore, we view capacitated graphs $(G, w)$ as multigraphs with $\min(w_e, c)$ copies of an edge $e$. Hence, we only deal with undirected, unweighted multigraphs.

Furthermore, we assume that each terminal vertex $t\in\mathcal{T}$ has degree at most $c$ through the following operation: for $t\in\mathcal{T}$ add a new vertex $t'$ and $c$ edges between $t$ and $t'$. As any cut separating $t$ and $t'$ has size at least $c$, this operation preserves all cuts of size at most $c$.

### 2.1 Cuts, Minimum Cuts, and $(\mathcal{T}, c)$-equivalency

For a graph $G=(V,E)$ and disjoint subsets $A, B\subseteq V$, let $E_G(A,B)$ denote the edges with one endpoint in $A$ and the other in $B$. The set of cuts in $G$ consists of $E_G(X, V\backslash X)$ for $X\subseteq V$. For a subset $X\subseteq V$ the *boundary* of $X$, denoted by $\partial X$, is $E_G(X, V\backslash X)$. For subsets $A, B\subseteq V$, define $\text{mincut}_G(A,B)$ to be the minimum cut separating $A, B$ in $G$. If $A\cap B\neq\emptyset$, then $\text{mincut}_G(A,B)=\infty$. Formally, we have

$$\text{mincut}_G(A,B)=\min_{\substack{S\subseteq V\\ A\subseteq S, B\subseteq V\backslash S}}|E_G(S, V\backslash S)|.$$

Furthermore, we let $\text{mincut}(G,A,B)$ be the edges in a minimum cut between $A, B$ in $G$, so that $\text{mincut}_G(A,B)=|\text{mincut}(G,A,B)|$. If multiple minimum cuts exist, the choice is arbitrary, and

does not affect our results. For disjoint $A, B \subseteq V$, we sometimes write their disjoint union as $A \mathbin{\dot{\cup}} B \stackrel{\text{def}}{=} A \cup B$ to emphasize that $A, B$ are disjoint. We define the *thresholded minimum cut* as $\text{mincut}^c_G(A, B) \stackrel{\text{def}}{=} \min(c, \text{mincut}_G(A, B))$. This definition then allows us to formally define $(\mathcal{T}, c)$-equivalence.

**Definition 2.1** *Let $G$ and $H$ be graphs both containing terminals $\mathcal{T}$. We say that $G$ and $H$ are $(\mathcal{T}, c)$-equivalent if for any subset $\mathcal{T}_1 \subseteq \mathcal{T}$ of the terminals we have that*

$$\text{mincut}^c_H(\mathcal{T}_1, \mathcal{T} \setminus \mathcal{T}_1) = \text{mincut}^c_G(\mathcal{T}_1, \mathcal{T} \setminus \mathcal{T}_1).$$

*If $G$ and $H$ are $(\mathcal{T}, c)$-equivalent, then we also say that $H$ is a connectivity-$c$ mimicking network for $G$.*

A *terminal cut* is any cut that has at least one terminal from $\mathcal{T}$ on both sides of the cut. The minimum terminal cut is the terminal cut with the smallest number of edges. We denote by $(\mathcal{T}, c)$-cuts the terminal cuts with at most $c$ edges.

We present several useful observations about the notion of $(\mathcal{T}, c)$-equivalence.

**Lemma 2.2** *If $G$ and $H$ are $(\mathcal{T}, c)$-equivalent, then for any subset of terminals $\widehat{\mathcal{T}} \subseteq \mathcal{T}$ and any $\widehat{c} \leq c$, $G$ and $H$ are also $(\widehat{\mathcal{T}}, \widehat{c})$-equivalent.*

**Lemma 2.3** *If $G$ and $H$ are $(\mathcal{T}, c)$-equivalent, then for any additional set of edges $\widehat{E}$ with endpoints in $\mathcal{T}$, $G \cup \widehat{E}$ and $H \cup \widehat{E}$ are also $(\mathcal{T}, c)$-equivalent.*

When used in the reverse direction, this lemma says that we can remove edges, as long as we include their endpoints as terminal vertices (Corollary 2.4). We complement this partitioning process by showing that sparsifiers on disconnected graphs can be built separately (Lemma 2.5).

**Corollary 2.4** *Let $\widehat{E}$ be a set of edges in $G$ with endpoints $V(\widehat{E})$, and $\mathcal{T}$ be terminals in $G$. If $H$ is $(\mathcal{T} \cup V(\widehat{E}), c)$-equivalent to $G \setminus \widehat{E}$, then $H \cup \widehat{E}$ is $(\mathcal{T}, c)$-equivalent to $G$.*

**Lemma 2.5** *If $G_1$ is $(\mathcal{T}_1, c)$-equivalent to $H_1$, and $G_2$ is $(\mathcal{T}_2, c)$-equivalent to $H_2$, then the vertex-disjoint union of $G_1$ and $G_2$, is $(\mathcal{T}_1 \cup \mathcal{T}_2, c)$-equivalent to the vertex-disjoint union of $H_1$ and $H_2$.*

When considering connectivity-$c$ mimicking networks, we can restrict our attention to sparse graphs [NI92]. For completeness, we prove the following lemma in Appendix A.1.

**Lemma 2.6** *Given any graph $G = (V, E)$ on $n$ vertices and any $c \geq 0$, we can find in $O(cm)$ time a graph $H$ on the same $n$ vertices, but with at most $c(n-1)$ edges, such that $G$ and $H$ are $(V, c)$-equivalent.*

### 2.2 Contractions

For a graph $G$ and an edge $e \in E(G)$, we let $G/e$ denote the graph obtained from $G$ by identifying the endpoints of $e$ as a single vertex; we say that we have *contracted* the edge $e$. The new vertex is marked as a terminal if at least one of its endpoints was a terminal. For a subset of edges $\widehat{E} \subseteq E$, we let $G/\widehat{E}$ denote the graph obtained from $G$ by contracting all edges in $\widehat{E}$. For any vertex set $X \subseteq V$, we denote by $G/X$ the graph obtained from $G$ by contracting every edge in $G[X]$.

For multigraphs, minimum cuts are monotonically increasing under contractions.

**Lemma 2.7** *For any subset of vertices $V_1$ and $V_2$ in $V$, and any set of edges $\widehat{E}$, it holds that*

$$\text{mincut}_G(V_1, V_2) \le \text{mincut}_{G/\widehat{E}}(V_1, V_2).$$

All our mimicking networks in Theorem 1.1 are produced by contracting edges of $G$.

# 3 Overview of our Approach

In this section, we give an overview for our proof of Theorem 1.1 Part 1. We first present our contraction-based approach to construct connectivity-$c$ mimicking networks with $O(kc^4)$ edges, and then show how to generically convert contraction-based approaches into efficient algorithms.

**Existence of connectivity-$c$ mimicking networks with $O(kc^4)$ edges.** Recall that, in our setup, we have a graph $G$ with a set of $k$ terminals $\mathcal{T} \subseteq V$, and wish to construct a graph $H$ with $O(kc^4)$ edges which is $(\mathcal{T}, c)$-equivalent to $G$. Our algorithm constructs $H$ by contracting edges of $G$ whose contraction does not affect the terminal cuts of size at most $c$. To find these *non-essential edges*, we intuitively perform a recursive procedure to identify *essential edges*, i.e., edges that are involved in terminal cuts of size at most $c$. After finding this set of essential edges in $G$, we contract all remaining edges.

At a high level, this recursive procedure finds a "small cut" in $G$, marks these edges as essential, and recurses on both halves. We formalize this notion of small cut through the definition of well-linkedness, variations of which have seen use throughout flow approximation algorithms [Chu12, RST14]. Here, we introduce a thresholded version of well-linkedness.[3]

**Definition 3.1** *For a graph $G$, we call a subset $X \subseteq V$* connectivity-$c$ well-linked *if for every bipartition $(A, B)$ of $X$, we have $|E_G(A, B)| \ge \min(|\partial A \cap \partial X|, |\partial B \cap \partial X|, c)$.*

If a bipartition $(A, B)$ of $X$ satisfies $|E_G(A, B)| < \min(|\partial A \cap \partial X|, |\partial B \cap \partial X|, c)$, we say that $E_G(A, B)$ is a *violating cut*, as it certifies that $X$ is not connectivity-$c$ well-linked. In this way, a violating cut corresponds to the "small cut" in $G$ whose edges we mark as essential. Conversely, we show in Lemma 4.2 that *all edges* inside a connectivity-$c$ well-linked set are non-essential, i.e., may be freely contracted.

Our full recursive algorithm is as follows. We maintain a partition of $V \backslash \mathcal{T} = X_1 \cup X_2 \cup \cdots \cup X_p$, where $p$ denotes the number of pieces, and track the potential function $\sum_{i=1}^{p} |\partial(X_i)|$. Initially, we let there be a single piece $X = V \backslash \mathcal{T}$, so that the potential value is $|\partial X| = |\partial(V \backslash \mathcal{T})| \le kc$, by our assumption in Section 2 that all terminals have degree at most $c$. We recursively refine the partition until each $X_i$ is either connectivity-$c$ well-linked or $|\partial(X_i)| \le 2c - 1$. More precisely, if $|\partial(X_i)| \ge 2c$ but $X_i$ is not connectivity-$c$ well-linked, let $E_G(A, B)$ be a violating cut of $X_i$ for a bipartition $(A, B)$ of $X_i$; we then remove $X_i$ and add $A, B$ to our partition. After this partitioning process terminates, the well-linked pieces among $X_1, X_2, \cdots, X_p$ may be contracted as discussed. For the pieces with $|\partial(X_i)| \le 2c - 1$ we make tricky manipulation on the boundary edges $\partial(X_i)$ as in Lemma 4.14 and then work on the line graph of $X_i$. Applying a kernelization result (see Lemma 4.11), which develops from matroid theory (gammoid in particular) and the representative sets lemma (see

[3]There are two notions of well-linkedness in the literature: edge linkedness and vertex linkedness. Here, our work focuses on edge linkedness. For discussions and definitions of vertex linkedness, we refer the readers to [Ree97]

Theorem 4.12), to the line graph gives rise to a fruitful result (see Lemma 4.4) which is more tailored to our edge-cut problem. It allows us to contract those pieces down to $O(c^3)$ edges and maintain $(\mathcal{T}, c)$-equivalence.

It suffices to argue that the number of pieces $p$ in the partition is at most $O(kc)$ at the end, so that our total edge count is $O(kc \cdot c^3) = O(kc^4)$. To show this, note that by Definition 3.1, for a violating cut $E_G(A, B)$ of $X$, we have that $\max(|\partial A|, |\partial B|) \leq |\partial X| - 1$ and $|\partial A| + |\partial B| \leq 2(c-1) + |\partial X|$. The former shows that our recursive procedure terminates, and combining the latter with our potential function bounds the number of pieces at the end. A more formal analysis is given in Section 4.1.

Note that the only non-constructive part of the above proof is the assumption that we can find a violating cut. However, we believe that an algorithm with very efficient running time is unlikely to exist, as this seems like a non-trivial instance of the non-uniform sparsest cut problem (in Appendix B, we present an algorithm with running time $2^{O(c^2)}k^2m$, which could be of independent interest). Hence, we present another procedure that does not rely on computing a violating cut.

**Efficient algorithm for constructing contraction-based connectivity-$c$ mimicking networks.** Our above analysis, in fact, shows that all but $O(kc^4)$ edges of $G$ may be contracted while still giving a graph which is $(\mathcal{T}, c)$-equivalent to $G$ (see Theorem 4.1).

A natural high level approach for an algorithm would be to go through the edges $e$ of $G$ sequentially and check whether contracting $e$ results in a $(\mathcal{T}, c)$-equivalent graph. If so, we contract $e$, and otherwise, we do not. Our analysis shows that at most $O(kc^4)$ edges will remain at the end, and in fact that proving a better existential bound improves the guarantees of such an algorithm.

Unfortunately, we do not know how to decide whether contracting an edge $e$ maintains all $(\mathcal{T}, c)$-cuts even in polynomial time. To get around this, we enforce particular structure on our graph by performing an *expander decomposition*. Expanders, defined formally in Definition 4.6, are governed by their conductance $\varphi$, and satisfy that any cut of size at most $c$ has at most $c\varphi^{-1}$ vertices on the smaller side. For a fixed parameter $\varphi$, [SW19] have given an efficient algorithm to remove $O(m\varphi \log^3 n)$ edges from $G$ such that all remaining components are expanders with conductance at least $\varphi$ (see Lemma 4.7). We now mark all the removed edges as essential, delete them, and mark their endpoints as additional terminals. Corollary 2.4 and Lemma 2.5 show that it suffices to work separately with each remaining component, which are guaranteed to be expanders with conductance at least $\varphi$. Note that the total number of terminals is now $k + O(m\varphi \log^3 n)$.

In order to check each edge $e$ and decide whether it can be safely contracted, we first enumerate all cuts in the graph with at most $c$ edges, of which there are at most $n(c\varphi^{-1})^{2c}$, using the fact that the small side of any cut of size at most $c$ has at most $c\varphi^{-1}$ vertices in a graph of conductance $\varphi$ (see Lemma 4.8). For each cut, we find the induced terminal partition, and all involved edges. This allows us to find the minimum cut value for any terminal partition, as long as it is at most $c$. Now, for an edge $e$ we check for all minimum cuts of size at most $c$ that it is involved in, whether there is another minimum cut separating terminals that does not involve $e$. If so, $e$ may be contracted, and otherwise, it cannot. In case we contract $e$, we delete the minimum cuts containing $e$ in the enumeration. Since cuts are monotone under contraction (see Lemma 2.7) and a cut in $G/e$ is also a cut in $G$ (see Lemma 4.9), the remaining minimum cuts in the enumeration correspond to the minimum cuts of $G/e$. Hence, we do not have to rebuild the set of all small cuts during the algorithm. As there are at most $n(c\varphi^{-1})^{2c}$ total cuts of size at most $c$, this algorithm may be executed in time $\widetilde{O}(nc(c\varphi^{-1})^{2c})$ using some standard data structures.

Finally, we discuss how to make our algorithm efficient, even though the total number of terminals increased to $k + O(m\varphi \log^3 n)$ after the expander decomposition. We set $\varphi^{-1} = O(c^4 \log^3 n)$, and note that the number of edges in our connectivity-$c$ mimicking network for $G$ is $O(kc^4 + mc^4 \varphi \log^3 n) \leq m/2$ as long as $m$ is a constant factor larger than $kc^4$. Now we repeat this procedure until our connectivity-$c$ mimicking network has $O(kc^4)$ edges, which requires $O(\log m)$ iterations. Details are given in Section 4.2.

# 4 Existence and Algorithm for Sparsifiers with $O(kc^4)$ edges

We first show the existence of a connectivity-$c$ mimicking network with $O(kc^4)$ edges in Section 4.1, based on contractions of connectivity-$c$ well-linked sets and replacement of sets with sparse boundary. Then, in Section 4.2, we design a $O(m(c \log n)^{O(c)})$ time algorithm to find a connectivity-$c$ mimicking network whose size matches the best guarantee achievable via contractions.

## 4.1 Existence of Sparsifiers with $O(kc^4)$ edges via Contractions

Given a graph $G$ and $k$ terminals $\mathcal{T}$, our construction of a connectivity-$c$ mimicking network with $O(kc^4)$ edges leverages a recursion scheme, where we maintain a partition of the vertices $X = V \backslash \mathcal{T}$, and track the total number of boundary edges of the partition as a potential function. This approach naturally introduces the notion of well-linkedness, a standard tool for studying flows and cuts, in order to refine the partition. Additionally, we must stop recursion at sets with sufficiently sparse boundary to guarantee that the recursion terminates without branching exponential times. The recursion results in a decomposition of $V \setminus \mathcal{T}$ into at most $kc$ clusters, each of which is either a connectivity-$c$ well-linked set or a set with sparse boundary. Then we contract the connectivity-$c$ well-linked sets and change the sets with sparse boundary into equivalent connectivity-$c$ mimicking networks with $O(c^3)$ edges. This procedure results in the following theorem.

**Theorem 4.1** *For a graph $G$ with $k$ terminals, there is a subset $E'$ of $E(G)$ such that the size of $E'$ is $O(kc^4)$ and the graph with all edges except $E'$ contracted, $G/(E \backslash E')$, is $(\mathcal{T}, c)$-equivalent to $G$.*

To this end, we elaborate the procedure with details in Section 4.1.1. To handle sets with sparse boundary, we use a known kernelization result based on matroid theory and the representative sets lemma. Unfortunately, these results mostly discuss vertex cuts, so in Section 4.3 we build a gadget to transform a given graph, from which we wish to obtain a connectivity-$c$ mimicking network, into a new graph whose minimum *vertex* cut of any partition of terminals corresponds to a minimum *edge* cut of the corresponding partition of terminals in the original graph.

### 4.1.1 Existence Proof: PolySizedcNetwork

As discussed in Section 3, it is desirable to find connectivity-$c$ well-linked sets, because they can be contracted without changing connectivity. Its proof is deferred to Appendix A.2.

**Lemma 4.2** *Let $X$ be a connectivity-$c$ well-linked set in $G$, and $\mathcal{T}$ be terminals disjoint with $X$ (i.e. $X \cap \mathcal{T} = \emptyset$). Then $G/X$ is $(\mathcal{T}, c)$-equivalent to $G$.*

$H = \textsc{PolySizedcNetwork}(G)$
Input: undirected unweighted multi-graph $G$.
Output: a connectivity-$c$ mimicking network $H$.

If $|\partial G| \leq 2c - 1$:

- Return, based on Lemma 4.4, a $(\mathcal{T}', c)$-equivalent connectivity-$c$ mimicking network with terminals $\mathcal{T}'$, where $\mathcal{T}'$ is the set of vertices incident to boundary edges $\partial G$.

Else:

- Find a violating cut $(V_1, V_2)$ if it exists and then, return PolySizedcNetwork($G[V_1]$) and PolySizedcNetwork($G[V_2]$).
- If no violating cut exists, contract $G$ to a single vertex.

Return a connectivity-$c$ mimicking network $H$.

Figure 1: Pseudocode for PolySizedcNetwork

The recursive procedure PolySizedcNetwork takes a subset $X$ of $V \setminus \mathcal{T}$ and bisects it if there exists a violating cut. Since finding a violating cut $E_G(A, B)$ of $X$ guarantees that $|\partial A|, |\partial B| < |\partial X|$, the recursion ends up reaching the base case in which the number of boundary edges is at most $2c - 1$. If there are no violating cuts, contracting $X$ also halts the recursion.

Hence, PolySizedcNetwork$(V \setminus \mathcal{T})$ just partitions $V \setminus \mathcal{T}$ into pieces being either connectivity-$c$ well-linked sets or sets whose number of boundary edges is at most $2c - 1$. The contraction of a connectivity-$c$ well-linked set for $(\mathcal{T}, c)$-equivalency is justified by Lemma 4.2. Also, Corollary 2.4 and Lemma 2.5 justify the replacement of a set $X$ with no terminals by a $(\mathcal{T}', c)$-equivalent graph, where we introduce boundary vertices in $X$ as the tentative terminals $\mathcal{T}'$.

The number of edges in a connectivity-$c$ mimicking network returned by the procedure depends on (i) the number of pieces, and (ii) how small equivalent sparsifiers a subgraph with $O(c)$ boundary edges has.

For (i), the number of smaller pieces being either connectivity-$c$ well-linked or $|\partial X| \leq 2c - 1$ simply matches with the number of branching during the recursion induced by the existence of a violating cut. It is bounded by the following decreasing invariant, which decreases by at least 1 for each branching.

**Lemma 4.3** *When* PolySizedcNetwork *splits a given $X$ into $\{X_i\}_{i=1}^{l}$, $\sum_{i \leq l} \max(|\partial(X_i)| - 2c + 1, 0)$ is a decreasing invariant with respect to separation induced by a violating cut for some $X_i$.*

**Proof:** The number of pieces only increases when a piece $X$ is divided into $A$ and $B$ by finding a violating cut. Denote $k = |\partial X|, k_1 = |\partial A \cap \partial X|, k_2 = |\partial B \cap \partial X|$, and $l = |E_G(A, B)|$. Note that $|\partial A| = k_1 + l$ and $|\partial B| = k_2 + l$. From the definition of a violating cut, we have $l < \min(k_1, k_2, c)$ and clearly, $|\partial A|, |\partial B| < |\partial X|$. When $X$ is divided, the term $|\partial X| - 2c + 1$ is replaced by $\max(|\partial A| - 2c + 1, 0) + \max(|\partial B| - 2c + 1, 0)$. If either $A$ or $B$ happens to have less than $2c - 1$ boundary edges, the latter term is strictly smaller. Now, let $|\partial A|, |\partial B| \geq 2c - 1$. Then,

$$\begin{aligned} |\partial A| - 2c + 1 + |\partial B| - 2c + 1 &= k_1 + l - 2c + 1 + k_2 + l - 2c + 1 \\ &= k - 2c + 1 + 2(l - c) + 1 < k - 2c + 1. \end{aligned}$$

□

The above lemma says that the number of branching is bounded by $|\partial X|$ since the decreasing invariant begins with $|\partial X|$. By the assumption that terminals have degree at most $c$, we have $|\partial(V \setminus \mathcal{T})|$ pieces, which is upper bounded by $kc$.

For (ii), due to a tricky preprocessing on boundary edges as in Lemma 4.14, we may assume that a set with $O(c)$ boundary edges can be viewed as a set with $O(c)$ tentative terminals, each of which has degree 1. This preprocessing gives rise to the following lemma with its proof presented in Section 4.3.

**Lemma 4.4** *Let $G = (V, E)$ be a graph with a set $\mathcal{T}$ of $O(c)$ terminals and each terminal have degree 1. There is a subset $E'$ of $E$ with $|E'| = O(c^3)$ and $G/(E \backslash E')$ is a connectivity-c mimicking network for $G$.*

We can combine series of lemmas to show Theorem 4.1.

**Proof of Theorem 4.1.** We show that PolySizedcNetwork returns a connectivity-$c$ mimicking network with $O(kc^4)$ edges for a graph $G$. First, applying Lemma 4.3 shows that the total number $p$ of pieces in the partition $V \setminus \mathcal{T} = X_1 \cup X_2 \cup \cdots \cup X_p$ is at most $kc$, as $|\partial(V \setminus \mathcal{T})| \leq kc$ by our assumption that all terminals have degree at most $c$.

To bound the total number of edges in the final sparsifier, we must analyze two contributions. First, the total number of boundary edges over all partition pieces is at most $\left|\bigcup_{i=1}^{p}(\partial X_i)\right| \leq O(kc^2)$, as the total number of boundary edges may increase by $c$ each time we split a partition piece into two pieces, and there are at most $kc$ pieces. The other contribution is from the partition pieces $X_i$ with $|\partial(X_i)| \leq 2c - 1$. The total number of edges from this is at most $kc \cdot O(c^3) = O(kc^4)$ by applying Lemma 4.4.

To verify that the returned graph is indeed a connectivity-$c$ mimicking network, it suffices to apply Lemma 4.2 to argue that we can contract well-linked pieces. Then we can use Corollary 2.4 to delete all boundary edges in $\bigcup_{i=1}^{p} \partial X_i$, and then use Lemma 2.5 and build a connectivity-$c$ mimicking network on each $X_i$ separately. □

### 4.2 Algorithm for the Optimal Sparsifiers: Contracting Non-Essential Edges

We use many forms of graph partitioning and operations, such as adding and deleting edges among terminals (Lemma 2.2, 2.3, and Corollary 2.4), and connected components may be handled separately (Lemma 2.5). These observations form the basis of our divide-and-conquer scheme, which repeatedly deletes edges, adds terminals, and works on connected components of disconnected graphs. Our approach in fact utilizes *expander decomposition* elaborated in Section 4.2.1 to split a graph into several expanders. Removing the inter-cluster edges, it sparsifies each expander by contracting non-essential edges in the expander, the contraction of each still preserves the value of a minimum cut up to $c$ between any partition of terminals. Then it glues together all the sparsified expanders via the inter-cluster edges to obtain a connectivity-$c$ mimicking network. This one pass reduces the number of edges by half. Repeating several passes leaves *essential* edges in the end, leading to the connectivity-$c$ mimicking network of the optimal size, which is currently $O(kc^4)$.

**Theorem 4.5** *For a graph $G$ with $n$ vertices, $m$ edges, and $k$ terminals, there exists an algorithm which successfully finds a connectivity-$c$ mimicking network with $O(kc^4)$ edges in $O(m(c\log n)^{O(c)})$.*

In fact, our algorithm guarantees a stronger property: If there exists a connectivity-$c$ mimicking network with $kf(c)$ edges that can be obtained by only contracting edges in $G$, for some function $f$, then our algorithm finds a connectivity-$c$ mimicking network with $O(kf(c))$ edges. We present the proof in three parts. In Section 4.2.1 and 4.2.2, we explain two sub-routines that are used in our algorithm. The description of the algorithm is in Section 4.2.3.

#### 4.2.1 Enumeration of Small Cuts via Expander Decomposition

To achieve Theorem 4.5, we utilize insights from recent results on finding $c$-vertex cuts [NSY19a, NSY19b, FY19], namely that in a well connected graph, any cut of size at most $c$ must have a very small side. This notion of connectivity is formalized through the notion of graph conductance.

**Definition 4.6** *In an undirected unweighted graph $G=(V,E)$, denote the volume of a subset of vertices, $vol(S)$, as the total degrees of its vertices. The conductance of a cut $S$ is then*

$$\Phi_G(S) = \frac{|\partial(S)|}{\min\{vol(S), vol(V\backslash S)\}},$$

*and the conductance of a graph $G=(V,E)$ is the minimum conductance of a subset of vertices:*

$$\Phi(G) = \min_{S\subseteq V} \Phi_G(S).$$

We use expander decomposition to reduce to the case where the graph has high conductance.

**Lemma 4.7** *(Theorem 1.2 of [SW19]) There exists an algorithm* ExpanderDecompose *that for any undirected unweighted graph $G$ and any parameter $\varphi$, decomposes in $O(m\varphi^{-1}\log^4 n)$ time $G$ into pieces $\{G_i\}$ of conductance at least $\varphi$ so that at most $O(m\varphi\log^3 n)$ edges are between the pieces.*

Note that if a graph has conductance $\varphi$, any cut $(S, V\backslash S)$ of size at most $c$ must have

$$\min\{vol(S), vol(V\backslash S)\} \leq c\varphi^{-1}. \tag{1}$$

In a graph with expansion $\varphi$, we can enumerate all cuts of size at most $c$ in time exponential in $c$ and $\varphi$. As a side note, the time complexity of both the results in Theorem 1.1 are dominated by the $c^{O(c)}$ term, essentially coming from this enumeration. Hence, a more efficient algorithm on enumeration may open up the possibility toward a faster algorithm for finding a connectivity-$c$ mimicking network.

**Lemma 4.8** *In a graph $G$ with $n$ vertices and conductance $\varphi$, there exists an algorithm that enumerates all cuts of size at most $c$ with connected smaller side in time $O(n(c\varphi^{-1})^{2c})$.*

**Proof:** We first enumerate over all starting vertices. For a starting vertex $u$, we repeatedly perform the following process.

1. Perform a DFS from $u$ until it reaches more than $c\varphi^{-1}$ vertices.

2. Pick one of the edges among the reached vertices as a cut edge.
3. Remove that edge, and recursively start another DFS starting at $u$.

After we have done this process at most $c$ times, we check whether the edges form a valid cut, and store it if so.

By Equation 1, the smaller side of the cut involves at most $c\varphi^{-1}$ vertices. Consider such a cut with $S$ as the smaller side, $F = E(S, V \setminus S)$, and $|S| \leq c\varphi^{-1}$. Then if we picked some vertex $u \in S$ as the starting point, the DFS tree rooted at $u$ must contain some edge in $F$ at some point. Performing an induction with this edge removed then gives that the DFS starting from $u$ will find this cut. Because there can be at most $O((c\varphi^{-1})^2)$ different edges picked among the vertices reached, the total work performed in the $c$ layers of recursion is $O((c\varphi^{-1})^{2c})$. □

It suffices to enumerate all such cuts once at the start, and reuse them as we perform contractions.

**Lemma 4.9** *If $F$ is a cut in $G/\widehat{E}$, then $F$ is also a cut in $G$.*

Note that this lemma also implies that an expander stays so under contractions. So, we do not even need to re-partition the graph as we recurse.

#### 4.2.2 Sparsifying Procedure ($\varphi$-SPARSIFY )

We need a subroutine used to sparsify a graph with conductance $\varphi$ and terminals $\mathcal{T}$. This subroutine named as $\varphi$-SPARSIFY takes such a graph and enumerates all cuts of size at most $c$ by a smaller side through Lemma 4.8. Then it sparsifies the expander by checking if the contraction of each edge still preserves $(\mathcal{T}, c)$-equivalency. Formally, we can contract an edge $e$ while preserving $(\mathcal{T}, c)$-equivalency if and only if for any partition $(\mathcal{T}_1, \mathcal{T}_2)$ of terminals $\mathcal{T}$, there exists a $(\mathcal{T}_1, \mathcal{T}_2)$-mincut not containing the edge $e$. For convenience, we call such an edge $e \in E(G)$ as *contractible* in $G$. Sequentially checking all edges in $G$ and contracting some if possible, we show that $\varphi$-SPARSIFY only leaves at most $O(|\mathcal{T}|c^4)$ "essential edges" which appear in a minimum cut of any partition of the terminals.

Through the enumeration of all cuts of size at most $c$ by a smaller side of the cut (Lemma 4.8), $\varphi$-SPARSIFY forms an auxiliary graph $H$ for efficient tracking of minimum cuts of partitions of terminals as follows: $V(H)$ is the disjoint union of $P$, $C$, and $E_0$, where

1. $P$ is the set of an induced partition of terminals $\mathcal{T}$ during the enumeration,
2. $C$ is the set of a minimum cut separating a partition of terminals in $P$,
3. $E_0$ is the set of edges included in a minimum cut in $C$,

and for $p \in P, c \in C$, and $e \in E_0$, add an edge $pc$ to $E(H)$ if $c$ is a minimum cut of $p$, and an edge $ce$ to $E(H)$ if $e \in c$.

For a given query edge $e \in E(G)$, the algorithm deletes all nodes (minimum cuts) $N(e) \subseteq C$, also removing the incident edges to $N(e)$. Then it checks if there is a node (partition $p$) in $P$ whose degree becomes 0 after the deletion. If so, it means that the edge $e$ appears in all minimum cuts of the partition $p$, leading the algorithm to undo the deletion. Otherwise, it means that the algorithm may contract $e$ and actually obtains a $(\mathcal{T}, c)$-equivalent graph $G/e$.

In the case that it contracts a contractible edge $e$, we should make sure that the auxiliary graph from $G/e$ is equal to $H$ with $N(e)$ deleted. First of all, a minimum cut $F$ of size at most $c$ inducing

a partition of terminals in $G/e$ is also a cut of the partition in $G$ (see Lemma 4.9). As $G/e$ and $G$ are $(\mathcal{T}, c)$-equivalent, $F$ must be a minimum cut of the partition in $G$ as well. For the opposite direction, a minimum cut of a partition of terminals in $G$, which does not contain $e$, is also a minimum cut of the partition in $G/e$, since the value of minimum cuts non-decreases under contraction (see Lemma 2.7). Therefore, we only need to enumerate all cuts of size at most $c$ $O(1)$ times and to create an auxiliary graph at the very beginning of $\varphi$-SPARSIFY, and simply update the auxiliary graph in response to contraction of edges without re-enumerating all cuts of size at most $c$ in contracted graphs.

In this way, scanning through each edge in sequence, $\varphi$-SPARSIFY checks if each edge is contractible in $G$ with contractible edges (in their turns) already contracted. In the end, it returns a $(\mathcal{T}, c)$-equivalent graph $G/X$, where $X$ is the set of contractible edges in each turn.

**Lemma 4.10** *For a graph $G$ with conductance $\varphi$, $n$ vertices, $m$ edges, and $k$ terminals, the algorithm $\varphi$-SPARSIFY returns a connectivity-$c$ mimicking network with $O(kc^4)$ edges in $O(m + nc(c\varphi^{-1})^{2c})$ time.*

**Proof:** Observe that once an edge $e$ is not contractible in its turn, $e$ never becomes contractible when checking other subsequent edges. In other words, for $i < j$, if the algorithm checks the edge $e$ in $i^{th}$ iteration and marks it as non-contractible, then $e$ is still not contractible when checking another edge in $j^{th}$ iteration. Hence, after checking all edges, the remaining edges $E^*$are not contractible in $G/(E - E^*)$. As Theorem 4.1 guarantees the existence of contractible edges as long as the number of remaining edges are larger than $O(kc^4)$, so the graph with the contractible edges contracted, $G/(E - E^*)$, returned by $\varphi$-SPARSIFY has at most $O(kc^4)$ edges as well.

For the running time, the algorithm enumerates all minimum cuts of size at most $c$ in $O(n \cdot (c\varphi^{-1})^{2c})$ and updating an auxiliary graph requires as many references as the number of edges in the auxiliary graph. As each minimum cut has size at most $c$, double counting on the number of edges results in the running time $O(m + nc \cdot (c\varphi^{-1})^{2c})$. □

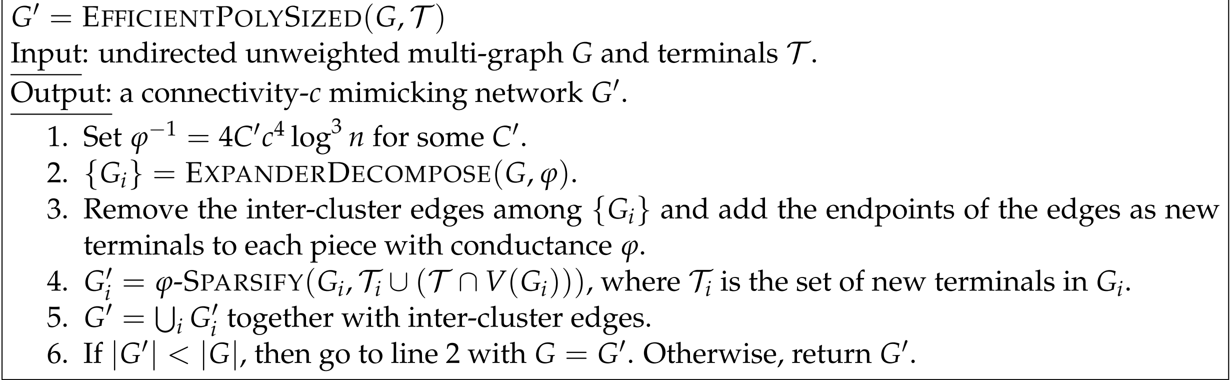
### 4.2.3 Putting things together

Now we join all the sparsified graphs via the removed inter-cluster edges and reduce the total number of edges by half. Repeating this procedure until no more edges are contractible, we can build a $(\mathcal{T}, c)$-equivalent graph with at most $O(kc^4)$ essential edges. We present the algorithm EFFICIENTPOLYSIZED with details in Figure 2 and with analysis as follows, where $C'$ is a constant such that in Lemma 4.7 the number of edges between the pieces are at most $C' m\varphi \log^3 n$. Theorem 1.1 part 1 follows from the analysis of EFFICIENTPOLYSIZED.

**Theorem 4.5** *For a graph $G$ with $n$ vertices, $m$ edges, and $k$ terminals, the algorithm EFFICIENTPOLYSIZED successfully finds a connectivity-$c$ mimicking network with $O(kc^4)$ edges in $O(m \cdot (c \log n)^{O(c)})$.*

**Proof:** By Corollary 2.4 and Lemma 4.10, EFFICIENTPOLYSIZED successfully finds a connectivity-$c$ mimicking network of $G$. For the size, we prove a more general statement that if the optimal number of edges in a connectivity-$c$ mimicking network of a graph with $k$ terminals is $k \cdot p(c)$ for a polynomial $p$, then EFFICIENTPOLYSIZED returns a sparsifier with $O(kp(c))$ edges.

We show by induction that after $i^{th}$ iteration, the number of remaining edges is at most $kp(c) \sum_{r=0}^{i-1} \frac{1}{2^r} + \frac{m}{2^i}$, which is bounded by $2kp(c) + \frac{m}{2^i}$. Hence, after $O(\log m)$ iterations, the algorithms yields a connectivity-$c$ mimicking network with $O(kp(c))$ edges.



$G' = \text{EFFICIENTPOLYSIZED}(G, \mathcal{T})$

Input: undirected unweighted multi-graph $G$ and terminals $\mathcal{T}$.

Output: a connectivity-$c$ mimicking network $G'$.

1. Set $\varphi^{-1} = 4C'c^4 \log^3 n$ for some $C'$.
2. $\{G_i\} = \text{EXPANDERDECOMPOSE}(G, \varphi)$.
3. Remove the inter-cluster edges among $\{G_i\}$ and add the endpoints of the edges as new terminals to each piece with conductance $\varphi$.
4. $G_i' = \varphi\text{-SPARSIFY}(G_i, \mathcal{T}_i \cup (\mathcal{T} \cap V(G_i)))$, where $\mathcal{T}_i$ is the set of new terminals in $G_i$.
5. $G' = \bigcup_i G_i'$ together with inter-cluster edges.
6. If $|G'| < |G|$, then go to line 2 with $G = G'$. Otherwise, return $G'$.

Figure 2: Pseudocode for EFFICIENTPOLYSIZED

Observe that $\varphi = 1/(4C'p(c)\log^3 n)$ satisfies $\varphi \cdot (C'p(c)\log^3 n + C'\log^3 n) \leq \frac{1}{2}$. In the first iteration, the total number of terminals is bounded by $k + mC'\varphi \log^3 n$. Hence, in line 5, the total number of edges in $G'$ is bounded by

$$(k + mC'\varphi \log^3 n)p(c) + mC'\varphi \log^3 n \leq kp(c) + m\varphi \cdot (C'p(c)\log^3 n + C'\log^3 n) \leq kp(c) + \frac{m}{2}$$

Using the similar argument for $i^{th}$ iteration and induction hypothesis, we have

$$(k + (kp(c)\sum_{r=0}^{i-2}\frac{1}{2^r} + \frac{m}{2^{i-1}})C'\varphi \log^3 n)p(c) + (kp(c)\sum_{r=0}^{i-2}\frac{1}{2^r} + \frac{m}{2^{i-1}})C'\varphi \log^3 n$$
$$\leq kp(c) + (kp(c)\sum_{r=0}^{i-2}\frac{1}{2^r} + \frac{m}{2^{i-1}})/2 \leq kp(c)\sum_{r=0}^{i-1}\frac{1}{2^r} + \frac{m}{2^i}.$$

For time complexity part, it is dominated by $\varphi$-SPARSIFY which takes time $O(m \cdot c^{O(c)} \log^{6c} n \cdot \log m) = O(m(c \log n)^{O(c)})$ as desired. □

### 4.3 Proof of Lemma 4.4: Transforming Edge Cuts to Vertex Cuts

As seen above, POLYSIZEDCNETWORK replaces a set with sparse boundary by a connectivity-$c$ mimicking network. Here we present a key lemma used for this subroutine, which reduces our problem to the problem of identifying essential vertices in preserving the value of minimum *vertex* cuts. The notion of vertex cuts is closely related with the notion of vertex-disjoint paths, which takes advantages of a well-developed theory from gammoid and representative sets. The following result in [KW12] is what we will make use of in essence.

**Lemma 4.11 ([KW12])** *Let $G = (V, E)$ be a directed graph, and $X \subseteq V$ a set of terminals. We can identify a set $Z$ of $O(|X|^3)$ vertices such that for any $A, B \subseteq X$, a minimum $(A, B)$-vertex cut in $G$ is contained in $Z$.*

Note that the above lemma addresses a minimum vertex cut, not an edge cut and it holds under digraphs setting. However, we can still replace digraphs with undirected graphs; for given an

undirected graph $G = (V, E)$, simply orient each edge in the both directions to obtain a directed graph $\widehat{G}$ and then apply the above result to $\widehat{G}$.

This simple but amazing result develops in the context of proving the usefulness of *matroid theory* to *kernelization*. As a set of vectors in vector fields enjoys removal of redundant vectors through the notion of linear independence, a general notion of independence potentially leads to identification of *important* objects in a problem. A matroid is one which abstracts and generalizes the notion, while especially developed in abstracting central notions in graph theory. Formally, for a given set $E$, a matroid $M = (E, \mathcal{I})$ means a collection $\mathcal{I}$ of all the 'independent' subsets of $E$. As motivated from linear independence in vector fields, the most basic matroid is derived from a matrix $A$ over a field $\mathbb{F}$ and the sets of all sets of columns linearly independent over $\mathbb{F}$. Some matroids $M$ happen to have an intrinsic matrix $A_M$ over $\mathbb{F}$ which represents the matroid in the way described above. We call such matroids as *linear* matroids over $\mathbb{F}$ and its matrix $A_M$ as a representation matrix.

When a directed graph $G = (V, E)$ with source vertices $S \subseteq V$ is given, a gammoid collects all the sets $T \subseteq V$ to which there exists $|T|$ vertex-disjoint paths from a subset of $S$. The size of the largest such $T$ is called the rank of the gammoid and naturally corresponds to the rank of matrices over $\mathbb{F}$. Among various matroids originated from graph theory, a *gammoid* naturally bridges the gap between vertex cuts and matroid theory since the size of a minimum vertex cut between two sets has to do with the number of vertex-disjoint paths between the two sets.

The last key notion is a $q$-representative set. For a given matroid $(E, \mathcal{I})$, a family $\mathcal{S}$ of subsets of size $p$ and any given $Y \subseteq E$ with $|Y| \leq q$, a $q$-representative set $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ contains a set $\widehat{X} \subseteq E$ with $\widehat{X} \cap Y = \emptyset$ and $\widehat{X} \cup Y$ being independent (i.e., $\widehat{X} \cup Y \in \mathcal{I}$) whenever $\mathcal{S}$ has such a set satisfying the same condition.

The previous studies [Lov77, Mar09, KW12, FLPS16] have been eager to find a representative set in polynomial time, which is also of independent interest, for a given representation matrix and we introduce the following theorem.

**Theorem 4.12 ([FLPS16])** *Let $M = (E, \mathcal{I})$ be a linear matroid of rank $p + q = k$ given together with its representation matrix $A_M$ over a field $\mathbb{F}$. Let $\mathcal{S} = \{S_1, ..., S_t\}$ be a family of independent sets of size $p$. Then a $q$-representative family $\widehat{\mathcal{S}}$ for $\mathcal{S}$ with at most $\binom{p+q}{p}$ sets can be found in $O(\binom{p+q}{p} t p^\omega + t \binom{p+q}{p}^{\omega-1})$ operations over $\mathbb{F}$, where $\omega < 2.373$ is the matrix multiplication exponent.*

Since a gammoid is a linear matroid and its notion of independence highly has to do with a minimum vertex cuts of two sets, the notion of a representative set has a key connection to the following problem: In a directed graph $G$ with vertex subsets $S$ and $T$, can we find a set $Z$ which contains a minimum $(A, B)$ vertex cut for any $A \subseteq S$ and $B \subseteq T$? Even though a naïve answer can be $Z = V(G)$, the identification of a representative set significantly reduces the size of $Z$ with dependencies on $p, q$, and $|\mathcal{S}|$ in the above setting.

**The Subroutine** We identify and contract unnecessary edges especially by using knowledge from vertex sparsification preserving a vertex connectivity. To exploit such fruitful results, we leverage a natural correspondence between edges and vertices via working on the *line graph* of a graph. In this way, edges appearing in a minimum edge cut of a partition of terminals in the original graph are given by identifying their corresponding vertices in the line graph through Lemma 4.11, which contains a minimum vertex cut of any partition of terminals.

Before making this connection clear, we can make a further assumption by preprocessing the boundary edges of an induced subgraph. This preprocessing relies on the following, which readily follows from Lemma 4.2.

**Observation 4.13** *Let $G$ be a graph with terminals $\mathcal{T}$ and $v \in V(G)$. A subdivision of an edge $uv$, which is to replace an edge $uv$ with a path $uwv$ through a new vertex $w$, results in a $(\mathcal{T}, c)$-equivalent graph.*

**Proof:** The set $\{u, w\}$ is a connectivity-$c$ well-linked set. □

Recall in POLYSIZEDCNETWORK that when $H$ has at most $2c - 1$ boundary edges, we mark the endpoints in $V(H)$ of the boundary edges $\partial H$ (i.e., $V(H) \cap V(\partial H)$) as tentative terminals $\widehat{\mathcal{T}}$ and then replace $H$ with a smaller equivalent one. In this case, despite $|\widehat{\mathcal{T}}| \leq 2c - 1$, we do not know how many incident edges $\widehat{\mathcal{T}}$ would have. By Observations 4.13, we can assume not only that $|\widehat{\mathcal{T}}| = O(c)$, but also that each terminal has degree 1.

**Lemma 4.14** *When working on an induced subgraph $H$ of a graph $G$ with its tentative terminals $\widehat{\mathcal{T}}$ coming from the endpoints of the boundary edges in $\partial H$ (i.e., $V(H) \cap V(\partial H)$), we may assume that each terminal in $\widehat{\mathcal{T}}$ has degree 1.*

**Proof:** Apply a subdivision of each boundary edge $uv \in \partial H$ with $v \in V(H)$, introducing one vertex $w_{uv}$. We extend $H$ to the induced subgraph $G[V(H) \cup \{w_{uv} : v \in V(H) \cap V(\partial H)\}]$, denoted by $H'$, and its boundary $\partial(H')$ becomes the edges $uw_{uv}$ for each $u \in V(H)^c \cap V(\partial H)$.

When we work on $H'$ via Corollary 2.4, new terminals $\widehat{\mathcal{T}}'$ becomes $\{w_{uv} | uv \in \partial H\}$, and it is clear that each terminal $w_{uv}$ has the unique neighbor $v$ in $H'$ (i.e., $\deg_{H'}(w_{uv}) = 1$). □

Note that this manipulation on boundary of a piece has no impact on boundary of other pieces. When sparsifying a set with $O(c)$ boundary edges, we can make the stronger assumption as in Lemma 4.14; the piece has $O(c)$ tentative terminals and each terminal has degree 1.

**Lemma 4.4** *Let $G = (V, E)$ be a graph with a set $\mathcal{T}$ of $O(c)$ terminals and each terminal have degree 1. There is a subset $E'$ of $E$ with $|E'| = O(c^3)$ and $G/(E \backslash E')$ is a connectivity-$c$ mimicking network for $G$.*

**Proof:** Let $v(e)$ be the corresponding vertex in the line graph $L(G)$ for an edge $e$ in $E$ and $v_t$ be the unique neighbor in $G$ of each terminal $t$ in $\mathcal{T}$. We enlarge the line graph $L(G)$ by adding a copy $t'$ of $t$ with its unique edge $t'v(tv_t)$. Also let $\mathcal{T}'$ be the set of such terminal copies and $L(G)'$ be the enlarged one.

Viewing $\mathcal{T}'$ as $X$ in Lemma 4.11, we have a subset $Z$ of $V(L(G)')$ with $O(c^3)$ vertices, which contains a minimum $(A', B')$-vertex cut of any bipartition $(A', B')$ of $\mathcal{T}'$. We slightly change $Z$ as follows: remove terminals $t'$ in $Z$ (if any) and add the unique neighbor $v(tv_t)$ of each terminal $t'$. Note that this perturbed set, denoted by $Z'$, still has $O(c^3)$ vertices but no intersection with $\mathcal{T}'$.

We show $Z'$ also contains a minimum vertex cut between any bipartition of terminals. Suppose that a partition $(A', B')$ of $\mathcal{T}'$ has a minimum vertex cut $C$ overlapping with $\mathcal{T}'$. For any $t' \in C \cap \mathcal{T}'$, the minimum vertex cut $C$ does not include the unique neighbor $v(tv_t)$ of $t'$; otherwise $C - t'$ is a smaller minimum $(A', B')$-vertex cut. Thus we can replace $C$ by another minimum cut $C - t' + v(tv_t)$. Repeating this operation on all terminals in $C$, we end up having a minimum

$(A', B')$-vertex cut disjoint from $\mathcal{T}'$ such that the modified minimum vertex cut is contained in $Z'$ in light of the construction of $Z'$.

Lastly, we take $E'$ as $\{e \in E(G) : v(e) \in Z'\}$ and claim $G/(E \backslash E')$ is $(\mathcal{T}, c)$-equivalent to $G$. For partition $(A, B)$ of $\mathcal{T}$ in $G$, a minimum edge cut $C$ between $A$ and $B$ corresponds to a vertex cut between $A'$ and $B'$ that consists of the corresponding vertices of the edges in $C$, thus $\text{mincut}_{L(G)'}(A', B') \leq \text{mincut}_G(A, B)$. By similar reasoning for the opposite direction, we have $\text{mincut}_{L(G)'}(A', B') \geq \text{mincut}_G(A, B)$ and thus $\text{mincut}_{L(G)'}(A', B') = \text{mincut}_G(A, B)$. It implies that even after contracting all edges in $E \backslash E'$, we still retain an edge cut of size $\text{mincut}_G(A, B)$. □

The last paragraph makes more sense by relying on the max-flow and min-cut theorem and the Menger's theorem. For example, there are $\text{mincut}_G(A, B)$ edge-disjoint paths between $A$ and $B$, which can be exactly transformed into $\text{mincut}_G(A, B)$ vertex-disjoint paths between $A'$ and $B'$. It implies that a minimum $(A', B')$-vertex cut has size at least $\text{mincut}_G(A, B)$ (i.e., $\text{mincut}_G(A, B) \leq \text{mincut}_{L(G)'}(A', B')$).

After sparsifying the enlarged piece $H'$ with $O(c)$ boundary edges into a smaller equivalent graph with $O(c^3)$ edges, the all edges $w_{uv}v$ for each $v \in H$ still remain, since $Z'$ contains the vertex $v(w_{uv}v)$ and so $E'$ also contains $w_{uv}v$. As $\{w_{uv}, v\}$ is a connectivity-$c$ well-linked set, we may contract it as if there were no any operations introducing additional vertices $w_{uv}$ at the very beginning. Therefore, our sparsifier with $O(kc^4)$ edges can be still found by only contracting edges, which means outputs of our algorithm in Section 4.2 have at most $O(kc^4)$ edges as well.

# 5 More Efficient Algorithms for Connectivity-$c$ Mimicking Networks

In this section we present a faster algorithm at the expense of the size by using the notion of "important" edges elaborated in Section 5.1.

Equipped with these notions, we prove the existence of connectivity-$c$ mimicking networks constructed from important edges in Section 5.2.1. Then we revisit in Section 5.2.2 the original approach for speedup (in Section 5.2.1) and achieve a result implying Theorem 1.1 Part 2 by utilizing expander decomposition and local cut algorithms in Section 5.2.3.

## 5.1 Equivalence, Cut Containment, and Cut Intersection

Our faster recursive algorithm works more directly with the notion of equivalence defined in Definition 2.1. This algorithm identifies a set of important edges, $\widehat{E}$, and forms $H$ by contracting all edges in $E \setminus \widehat{E}$.

Observe that as long as $\widehat{E}$ is small, contracting $E \setminus \widehat{E}$ still results in a graph with few vertices and edges. Therefore, our goal is find a set $\widehat{E}$ of important edges to keep in $H$ such that the size of $\widehat{E}$ is not much larger than $|\mathcal{T}|$. We will show for the purpose of being $(\mathcal{T}, c)$-equivalent, a sufficient condition is that every $(\mathcal{T}, c)$-cut can be formed using edges from only $\widehat{E}$. This leads to the definition of $\widehat{E}$ containing all $(\mathcal{T}, c)$-cuts, which was also used in [MS18] for the $c \leq 5$ setting.

**Definition 5.1 (Cut containment)** *In a graph $G = (V, E)$ with terminals $\mathcal{T}$, a subset of edges $E^{contain} \subseteq E$ is said to contain all $(\mathcal{T}, c)$-cuts if for any partition $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ with $\text{mincut}_G(\mathcal{T}_1, \mathcal{T}_2) \leq c$ there is a cut $F \subseteq E^{contain}$ such that*

1. *$F$ has size equal to $\text{mincut}_G(\mathcal{T}_1, \mathcal{T}_2)$,*

2. *$F$ is also a cut between $\mathcal{T}_1$ and $\mathcal{T}_2$. That is, $\mathcal{T}_1$ and $\mathcal{T}_2$ are disconnected in $G \setminus F$.*

Note that this is different than containing all the minimum cuts: on a length $n$ path with two endpoints as terminals, any intermediate edge contains a minimum terminal cut, but there are up to $n-1$ different such minimum cuts.

If $E^{contain}$ contains all $(\mathcal{T}, c)$-cuts, we may contract all edges in $E \setminus E^{contain}$ to obtain a $(\mathcal{T}, c)$-equivalent graph $H$ of $G$.

**Lemma 5.2** *If $G = (V, E)$ is a connected graph with terminals $\mathcal{T}$, and $E^{contain}$ is a subset of edges that contain all $(\mathcal{T}, c)$-cuts, then the graph*

$$H = G / \left( E \backslash E^{contain} \right)$$

*is $(\mathcal{T}, c)$-equivalent to $G$, and has at most $|E^{contain}| + 1$ vertices.*

**Proof:** Consider any cut using entirely edges in $E^{contain}$: contracting edges from $E \backslash E^{contain}$ will bring together vertices on the same side of the cut. Therefore, the separation of vertices given by this cut also exists in $H$ as well.

To bound the size of $H$, observe that contracting all edges of $G$ brings it to a single vertex. That is, $H / E^{contain}$ is a single vertex: uncontracting an edge can increase the number of vertices by at most 1, so $H$ has at most $|E^{contain}| + 1$ vertices. □

We can also state Corollary 2.4 and Lemma 2.5 in the language of edge containment.

**Lemma 5.3** *Let $\widehat{E}$ be a set of edges in $G$ with endpoints $V(\widehat{E})$, and $\mathcal{T}$ be terminals in $G$. If edges $E^{contain}$ contain all $(\mathcal{T} \cup V(\widehat{E}), c)$-cuts in $G \backslash \widehat{E}$, then $E^{contain} \cup \widehat{E}$ contains all $(\mathcal{T}, c)$-cuts in $G$.*

**Lemma 5.4** *If the edges $E_1^{contain} \subseteq E(G_1)$ contain all $(\mathcal{T}_1, c)$-cuts in $G_1$, and the edges $E_2^{contain} \subseteq E(G_2)$ contain all $(\mathcal{T}_2, c)$-cuts in $G_2$, then $E_1^{contain} \cup E_2^{contain}$ contains all the $(\mathcal{T}_1 \cup \mathcal{T}_2, c)$-cuts in the vertex disjoint union of $G_1$ and $G_2$.*

These motivate us to gradually build up $E^{contain}$ through a further intermediate definition.

**Definition 5.5** *In a graph $G = (V, E)$ with terminals $\mathcal{T}$, a subset of edges $E^{intersect} \subseteq E$ intersects all $(\mathcal{T}, c)$-cuts for some $c > 0$ if for any partition $\mathcal{T} = \mathcal{T}_1 \mathbin{\dot\cup} \mathcal{T}_2$ with $\mathrm{mincut}_G(\mathcal{T}_1, \mathcal{T}_2) \leq c$, there exists a cut $F = E(V_1, V_2)$ such that:*

1. *$F$ has size $\mathrm{mincut}_G(\mathcal{T}_1, \mathcal{T}_2)$,*
2. *$F$ induces the same separation of $\mathcal{T}$: $V_1 \cap \mathcal{T} = \mathcal{T}_1$, $V_2 \cap \mathcal{T} = \mathcal{T}_2$.*
3. *$F$ contains at most $c-1$ edges from any connected component of $G \backslash E^{intersect}$.*

**Reduction to Cut Intersection** Based on Definition 5.5, we can reduce the problem of finding a set $E^{contain}$ which contains all $(\mathcal{T}, c)$-cuts to the problem of finding a set $E^{intersect}$ which intersects all small cuts. Formally, the deletion of an intersecting edge set $E^{intersect}$ separates $(\mathcal{T}, c)$-cuts of $G$ into edge sets of size $c-1$. Each of these smaller cuts happens on one of the connected components of $G \setminus E^{intersect}$, and can thus be considered independently when we construct the containing sets of $G \setminus E^{intersect}$.

$E^{contain} = \text{GETCONTAININGEDGES}(G, \mathcal{T}, c)$
Input: undirected unweighted multi-graph $G$, terminals $\mathcal{T}$, cut threshold $c$.
Output: set of edges $E^{intersect}$ that intersects all $(\mathcal{T}, c)$-cuts.

1. Initialize $E^{contain} \leftarrow \emptyset$.
2. For $\widehat{c} \leftarrow c \dots 1$ in decreasing order:
   (a) $E^{contain} \leftarrow E^{contain} \cup \text{GETINTERSECTINGEDGES}(G, \mathcal{T}, \widehat{c})$.
   (b) $G \leftarrow G \backslash E^{contain}$.
   (c) $\mathcal{T} \leftarrow \mathcal{T} \cup V(E^{contain})$, where $V(E^{contain})$ is the endpoints of all edges in $E^{contain}$.
3. Return $E^{contain}$.

Figure 3: Pseudocode for finding a set of edges that contain all the $(\mathcal{T}, c)$-cuts.

This is done by first finding an intersecting set $E^{intersect}$, and then recursing on each component in the (disconnected) graph with $E^{intersect}$ removed, but with the endpoints of $E^{intersect}$ included as terminals as well. This will increase the number of edges and terminals, but allow us to focus on $(c-1)$-connectivity in the components, leading to our recursive scheme. The overall algorithm simply iterates this process until $c$ reaches 1, as shown in Figure 3. All arguments until now can be summarized as the following stitching lemma.

**Lemma 5.6** *Let $G = (V, E)$ be a graph with terminals $\mathcal{T}$, and $E^{intersect} \subseteq E$ be a set of edges that intersects all $(\mathcal{T}, c)$-cuts. For $\widehat{\mathcal{T}} = \mathcal{T} \cup V\left(E^{intersect}\right)$, let $E^{contain} \subseteq E \setminus E^{intersect}$ be a set of edges that contains all $(\widehat{\mathcal{T}}, c-1)$ cuts in the graph $(V, E \backslash E^{intersect})$. Then $E^{contain} \cup E^{intersect}$ contains all $(\mathcal{T}, c)$-cuts in $G$.*

**Proof:** Consider a partition $\mathcal{T} = \mathcal{T}_1 \uplus \mathcal{T}_2$ with $\text{mincut}_G(\mathcal{T}_1, \mathcal{T}_2) \leq c$. Since $E^{intersect}$ intersects all $(\mathcal{T}, c)$-cuts, there is cut of size $\hat{c}$ separating $\mathcal{T}_1$ and $\mathcal{T}_2$, which has at most $c-1$ edges in each component of $G \backslash E^{intersect}$.

Combining this with Lemmas 5.3 and 5.4 shows that if $E^{contain}$ contains all $(\mathcal{T} \cup V(E^{intersect}), c-1)$-cuts in $G \backslash E^{intersect}$, then $E^{contain} \cup E^{intersect}$ contains all $(\mathcal{T}, c)$-cuts in $G$. □

The following theorem shows the bounds for generating a set of edges $E^{intersect}$ that intersects all $(\mathcal{T}, c)$-cuts. Its correctness follows from Lemma 5.6.

**Theorem 5.7** *For any parameter $\varphi$, value $c$, and graph $G$ with terminals $\mathcal{T}$, there exists an algorithm that generates a set of edges $E^{intersect}$ that intersects all $(\mathcal{T}, c)$-cuts:*

1. *with size at most $O((\varphi m \log^4 n + |\mathcal{T}|) \cdot c)$ in $\widetilde{O}(m(c\varphi^{-1})^{2c})$ time.*
2. *with size at most $O((\varphi m \log^4 n + |\mathcal{T}|) \cdot c^2)$ in $\widetilde{O}(m\varphi^{-2}c^7)$ time.*

While Theorem 5.7 Part 1 developed in Section 5.2.1 provides a slow subroutine, we are able to modify the argument in Section 5.2.2 and then take further steps, expander decomposition and local cut algorithms, in Section 5.2.3 to obtain Theorem 5.7 Part 2.

In essence, we use Theorem 5.7 Part 2 to prove Theorem 1.1 Part 2 in Appendix A.3. As the size of $E^{contain}$ multiplies by $O(c^2)$ every iteration, the total size of $E^{contain}$ at the end is $O(c)^{2c}$, as desired in Theorem 1.1 Part 2.

## 5.2 Efficient Algorithm: Recursive Constructions

In this section, we give recursive algorithms for finding sets of edges that intersect all $(\mathcal{T}, c)$-cuts (as defined in Definition 5.5). In Section 5.2.1, we show the existence of a small set of edges that intersects all $(\mathcal{T}, c)$-cuts. In Section 5.2.2, we give a polynomial-time construction that outputs a set of edges whose size is slightly larger than those given in Section 5.2.1.

Our routines are based on recursive contractions. Suppose we have found a terminal cut $F = E(V_1, V_2)$. Then any cut $\widehat{F}$ overlapping with both $G[V_1]$ and $G[V_2]$, or $F$, will have only at most $c - 1$ edges in common with $G[V_1]$ or $G[V_2]$. Thus, it suffices for us to focus on cuts that lie entire in one half, which we assume without loss of generality is $G[V_1]$.

Since none of the edges in $F$ and $G[V_2]$ are used, we can work equivalently on the graph with all these edges contracted. The progress made by this, on the other hand, may be negligible: consider, for example, the extreme case of $F$ being a matching, and no edges are present in $G[V_2]$. On the other hand, if $G[V_2]$ is connected, then it will become a single terminal vertex in addition to $V_1$, and the two halves that we recurse on add up to a size that's only slightly larger than $G$.

Thus, our critical starting point is to look for cuts $(V_1, V_2)$, where both $G[V_1]$ and $G[V_2]$ are connected, and contain two or more terminals. We first look for such cuts through exhaustive enumeration in Section 5.2.1, and show that when none are found, we can simply terminate by taking all minimum cuts with one terminal on one side, and the other terminals on the other side. Unfortunately, we do not have a polynomial time algorithm for determining the existence of a cut $(V_1, V_2)$ with size at most $c$ such that both $G[V_1]$ and $G[V_2]$ are connected and have at least 2 terminals.

In Section 5.2.2, we take a less direct, but poly-time computable approach based on computing the minimum terminal cut among the terminals $\mathcal{T}$. Both sides of this cut are guaranteed to be connected by the minimality of the cut. However, we cannot immediately recurse on this cut due to it possibly containing only one terminal on one side. We address this by defining maximal terminal separating cuts: minimum cuts with only that terminal on one side, but containing as many vertices as possible. The fact that such cuts can only grow $c$ times until their sizes exceed $c$ allows us to bound the number of cuts recorded by the number of terminals, times an extra factor of $c$, for a total of $O(kc^2)$ edges in the sparsifier.

### 5.2.1 Existence

Our divide-and-conquer scheme relies on the following observation about when $(\mathcal{T}, c)$-cuts are able to interact completely with both sides of a cut.

**Lemma 5.8** *Let $F$ be a cut given by the partition $V = V_1 \uplus V_2$ in $G = (V, E)$ such that both $G[V_1]$ and $G[V_2]$ are connected, and $\mathcal{T}_1 = V_1 \cap \mathcal{T}$ and $\mathcal{T}_2 = V_2 \cap \mathcal{T}$ be the partition of $\mathcal{T}$ induced by this cut. If $E_1^{intersect}$ intersects all $(\mathcal{T}_1 \cup \{v_2\}, c)$-terminal cuts in $G/V_2$, the graph formed by contracting all of $V_2$ into a single vertex $v_2$, and similarly $E_2^{intersect}$ intersects all $(\mathcal{T}_2 \cup \{v_1\}, c)$-terminal cuts in $G/V_1$, then $E_1^{intersect} \cup E_2^{intersect} \cup F$ intersects all $(\mathcal{T}, c)$-cuts in $G$ as well.*

**Proof:** Consider some cut $\widehat{F}$ of size at most $c$. If $\widehat{F}$ uses an edge from $F$, then it has at most $c - 1$ edges in $G \backslash F$, and thus in any connected component as well. If $\widehat{F}$ has at most $c - 1$ edges in $G[V_1]$, then every connected component in $(G \setminus F) \setminus E_1^{intersect}$ has at most $c - 1$ edges from $\widehat{F}$. This follows

because removing $F$ has already disconnected $V_1$ and $V_2$, and removing $E_1^{intersect}$ can only further disconnects the remaining components.

The only remaining case is when $\widehat{F}$ is entirely contained in one of the sides. Without loss of generality assume that $\widehat{F}$ is entirely contained in $V_1$, i.e., $\widehat{F} \subseteq E(G[V_1])$. Because no edges from $G[V_2]$ are removed and $G[V_2]$ is connected, all of $\mathcal{T}_2$ must be on one side of the cut, and can therefore be represented by a single vertex $v_2$. Using the induction hypothesis on the cut $\widehat{F}$ in $G/V_2$ with the terminal separation given by all of $\mathcal{T}_2$ replaced by $v_2$ gives that $\widehat{F}$ has at most $c-1$ edges in any connected component of $(G/V_2)\setminus E_1^{intersect}$. Since connected components remain intact under contracting connected subsets, we conclude that $\widehat{F}$ has at most $c-1$ edges in any connected components of $G\setminus E_1^{intersect}$ as well. □

However, to make progress on such a partition, we need to contract at least two terminals together with either $V_1$ or $V_2$. This leads to our key definition of a non-trivial $\mathcal{T}$-separating cut:

**Definition 5.9** *A non-trivial $(\mathcal{T}, c)$-cut is a separation of $V$ into $V_1 \cup V_2$ such that:*

1. *the subgraphs induced by $V_1$ and $V_2$ (i.e., $G[V_1]$ and $G[V_2]$) are both connected.*
2. *$|V_1 \cap \mathcal{T}| \geq 2$, $|V_2 \cap \mathcal{T}| \geq 2$.*

Such cuts are critical for partitioning and recursing on the two resulting pieces. The connectivity of $G[V_1]$ and $G[V_2]$ is necessary for applying Lemma 5.8, and $|V_1 \cap \mathcal{T}| \geq 2$, $|V_2 \cap \mathcal{T}| \geq 2$ are necessary to ensure that making this cut and recursing makes progress.

We now study the set of graphs $G$ and terminals $\mathcal{T}$ for which a non-trivial cut exists. For example, consider the case when $G$ is a star graph (a single vertices with $n-1$ vertex connected to it) and all vertices are terminals. In this graph, the side of the cut not containing the center can only have a single vertex; hence, there are no non-trivial cuts.

We can, in fact, prove the converse: if no such interesting separations exist, we can terminate by only considering the $|\mathcal{T}|$ separations of $\mathcal{T}$ formed with one terminal on one of the sides. We define these cuts to be the $s$-isolating cuts.

**Definition 5.10** *For a graph $G$ with terminal set $\mathcal{T}$ and some $s \in \mathcal{T}$, an $s$-isolating cut is a partition of the vertices $V = V_A \cup V_B$ such that $s$ is the only terminal in $V_A$, i.e., $s \in V_A$, $(\mathcal{T}\setminus\{u\}) \subseteq V_B$.*

**Lemma 5.11** *If $\mathcal{T}$ is a subset of at least 4 terminals in an undirected graph $G$ that has no non-trivial $\mathcal{T}$-separating cut of size at most $c$, then the union of all $s$-isolating cuts of size at most $c$:*

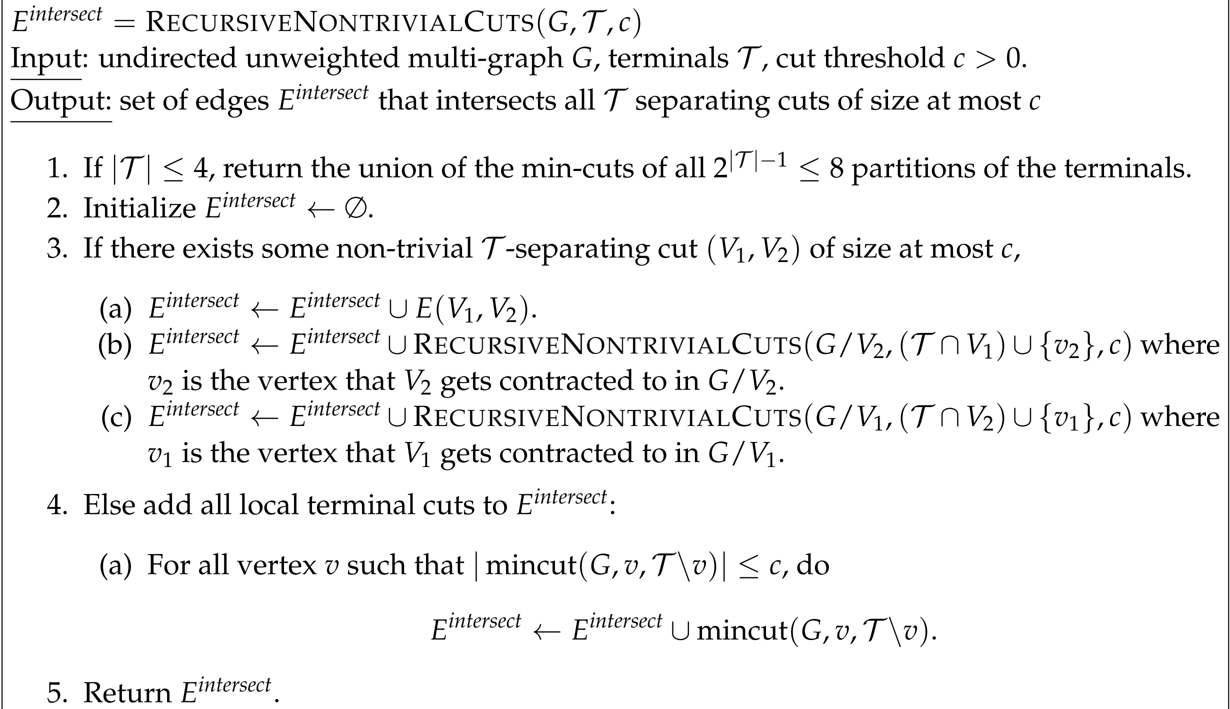
$$E^{intersect} \leftarrow \bigcup_{\substack{s \in \mathcal{T} \\ \text{mincut}_G(\{s\}, \mathcal{T}\setminus\{s\}) \leq c}} \text{mincut}\left(G, \{s\}, \mathcal{T}\setminus\{s\}\right),$$

*contains all $(\mathcal{T}, c)$-cuts of $G$.*

**Proof:** Consider a graph with no non-trivial $\mathcal{T}$-separating cut of size at most $c$, but there is a partition of $\mathcal{T}$, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, such that the minimum cut between $\mathcal{T}_1$ and $\mathcal{T}_2$, $V_1$ and $V_2$, has at most $c$ edges, and $|\mathcal{T}_1|, |\mathcal{T}_2| \geq 2$.

Let $F$ be one such cut, and consider the graph

$$\widehat{G} = G/\left(E\setminus F\right),$$



$E^{intersect} = \textsc{RecursiveNontrivialCuts}(G, \mathcal{T}, c)$

Input: undirected unweighted multi-graph $G$, terminals $\mathcal{T}$, cut threshold $c > 0$.

Output: set of edges $E^{intersect}$ that intersects all $\mathcal{T}$ separating cuts of size at most $c$

1. If $|\mathcal{T}| \leq 4$, return the union of the min-cuts of all $2^{|\mathcal{T}|-1} \leq 8$ partitions of the terminals.
2. Initialize $E^{intersect} \leftarrow \emptyset$.
3. If there exists some non-trivial $\mathcal{T}$-separating cut $(V_1, V_2)$ of size at most $c$,
   (a) $E^{intersect} \leftarrow E^{intersect} \cup E(V_1, V_2)$.
   (b) $E^{intersect} \leftarrow E^{intersect} \cup \textsc{RecursiveNontrivialCuts}(G/V_2, (\mathcal{T} \cap V_1) \cup \{v_2\}, c)$ where $v_2$ is the vertex that $V_2$ gets contracted to in $G/V_2$.
   (c) $E^{intersect} \leftarrow E^{intersect} \cup \textsc{RecursiveNontrivialCuts}(G/V_1, (\mathcal{T} \cap V_2) \cup \{v_1\}, c)$ where $v_1$ is the vertex that $V_1$ gets contracted to in $G/V_1$.
4. Else add all local terminal cuts to $E^{intersect}$:
   (a) For all vertex $v$ such that $|\operatorname{mincut}(G, v, \mathcal{T} \backslash v)| \leq c$, do
   $$E^{intersect} \leftarrow E^{intersect} \cup \operatorname{mincut}(G, v, \mathcal{T} \backslash v).$$
5. Return $E^{intersect}$.

Figure 4: Algorithm for finding a set of edges that intersects all terminal cuts of size $\leq c$.

that is, we contract all edges except the ones on this cut. Note that $\widehat{G}$ has at least 2 vertices.

Consider a spanning tree $T$ of $\widehat{G}$. By minimality of $F$, each node of $T$ must contain at least one terminal. Otherwise, we can keep one edge from such a node without affecting the distribution of terminal vertices.

We now show that no vertex of $T$ can contain $|\mathcal{T}| - 1$ terminals. If $T$ has exactly two vertices, then one vertex must correspond to $\mathcal{T}_1$ and one must correspond to $\mathcal{T}_2$, so no vertex has $|\mathcal{T}| - 1$ terminals. If $T$ has at least 3 vertices, then because every vertex contains at least one terminal, no vertex in $T$ can contain $|\mathcal{T}| - 1$ vertices.

Also, each leaf of $T$ can contain at most one terminal, otherwise deleting the edge adjacent to that leaf forms a non-trivial cut.

Now consider any non-leaf node of the tree, say $r$. As $r$ is a non-leaf node, it has at least two different neighbors that lead to leaf vertices.

Let us make $r$ the root of this tree and consider some neighbor of $r$, say $x$. If the subtree rooted at $x$ has more than 2 terminals, then cutting the $rx$ edge results in two components, each containing at least two terminals (the component including $r$ has at least one other neighbor that contains a terminal). Thus, the subtree rooted at $x$ can contain at most one terminal, and must therefore be a singleton leaf.

Hence, the only possible structure of $T$ is a star centered at $r$ (which may contain multiple terminals) in which each leaf has exactly one terminal in it. This in turn implies that $\widehat{G}$ must also be

a star, i.e., $\widehat{G}$ has the same edges as $T$ but possibly with multi-edges. This is because any edges incident to a leaf of a star forms a connected cut.

By minimality, each cut separating the root from a leaf is a minimal cut for that single terminal, and these cuts are disjoint. Thus, taking the union of edges of all these singleton cuts gives a cut that partitions $\mathcal{T}$ in the same way and has the same size. □

Note that Lemma 5.11 is not claiming all the $(\mathcal{T}, c)$-cuts of $\mathcal{T}$ are singletons. Instead, it says that any $(\mathcal{T}, c)$-cut can be formed from a union of single terminal cuts.

Combining Lemma 5.8 and 5.11, we obtain the recursive algorithm in Figure 4, which demonstrates the existence of $O(|\mathcal{T}| \cdot c)$ sized $(\mathcal{T}, c)$-cut-intersecting subsets. If there is a non-trivial $\mathcal{T}$-separating cut, the algorithm in Line 3 finds it and recurses on both sides of the cut using Lemma 5.8. Otherwise, by Lemma 5.11, the union of the $s$-isolating cuts of size at most $c$ contains all $(\mathcal{T}, c)$-cuts, so the algorithm keeps the edges of those cuts in Line 4a.

**Lemma 5.12** RECURSIVENONTRIVIALCUTS *as shown in Figure 4 correctly returns a set of $(\mathcal{T}, c)$-cut-intersecting edges of size at most $O(|\mathcal{T}| \cdot c)$.*

**Proof:** The correctness of the algorithm can be argued by induction. The base case, where we terminate by adding all min-cuts with one terminal on one side, follows from Lemma 5.11, while the inductive case follows from applying Lemma 5.8.

It remains to bound the size of $E^{intersect}$ returned. Once again there are two cases: The first case is when we terminate with the union of singleton cuts. Each such cut has size at most $c$, thus summing to the total of $|\mathcal{T}| \cdot c$.

The second is the recursive case, which can be viewed as partitioning $k \geq 4$ terminals into two instances of sizes $k_1$ and $k_2$ where $k_1 + k_2 = k + 2$ and $k_1, k_2 \geq 3$. Note that the total values of $|\text{TERMINALS}| - 2$ across all the recursion instances is strictly decreasing, and is always positive. So, the recursion can branch at most $|\mathcal{T}|$ times, implying that the total number of edges added is at most $O(c \cdot |\mathcal{T}|)$. □

In fact, we may modify RECURSIVENONTRIVIALCUTS to take extra steps for marginal speedup by utilizing expander decomposition.

**Proof of Theorem 5.7 Part 1.** First, we perform expander decomposition, remove the inter-cluster edges, and add their endpoints as terminals.

Now, we describe the modifications to RECURSIVENONTRIVIALCUTS that make it efficient.

Lemma 2.3 and Lemma 2.5 allow us to consider the pieces separately.

Now at the start of each recursive call, enumerate all cuts of size at most $c$, and store the vertices on the smaller side, which by Equation 1 above has size at most $O(c\varphi^{-1})$. When such a cut is found, we only invoke recursion on the smaller side (in terms of volume). For the larger piece, we can continue using the original set of cuts found during the search.

To use a cut from a pre-contracted state, we need to:

1. check if all of its edges remain (using a union-find data structure).
2. check if both portions of the graph remain connected upon removal of this cut – this can be done by explicitly checking the smaller side, and certifying the bigger side using a dynamic

connectivity data structure by removing all edges from the smaller side.

Since we contract each edge at most once, the total work done over all the larger side is at most

$$\widetilde{O}\left(m\left(c\varphi^{-1}\right)^{2c}\right),$$

where we have included the logarithmic factors from using the dynamic connectivity data structure. Furthermore, the fact that we only recurse on things with half as many edges ensures that each edge participates in the cut enumeration process at most $O(\log n)$ times. Combining these then gives the overall running time. □

### 5.2.2 Polynomial-Time Construction

It is not clear to us how the previous algorithm in Section 5.2.1 could be implemented in polynomial time. While incorporating expander decomposition makes our algorithm faster, its running time still has the $\log^{O(c)} n$ term (as stated in Theorem 5.7 Part 1). In this section, we give a more efficient algorithm that returns sparsifiers of larger size, but ultimately leads to the faster running time given in Theorem 5.7 Part 2. It was derived by working backwards from the termination condition of taking all the cuts with one terminal on one side in Lemma 5.11.

Recall that a terminal cut is a cut with at least one terminal one both sides. The algorithm has the same high level recursive structure, but it instead only finds the minimum terminal cut or certifies that its size is greater than $c$. This takes $O(m + nc^3 \log n)$ time using an algorithm by Cole and Hariharan [CH03].

**Theorem 5.13 (Minimum Terminal Cut [CH03])** *Given graph $G$ with terminals $\mathcal{T}$ and constant $c$, there is an $O(m + nc^3 \log n)$ time algorithm which computes the minimum terminal cut on $\mathcal{T}$ or certifies that its size is greater than $c$.*

It is direct to check that both sides of a minimum terminal cut are connected. This is important towards our goal of finding a non-trivial $\mathcal{T}$-separating cut, defined in Definition 5.9.

**Lemma 5.14** *If $(V_A, V_B)$ is the global minimum $\mathcal{T}$-separating cut in a connected graph $G$, then both $G[V_A]$ and $G[V_B]$ must be connected.*

**Proof:** Suppose for the sake of contradiction that $V_A$ is disconnected as $V_A = V_{A1} \mathbin{\dot\cup} V_{A2}$. Without loss of generality assume $V_{A1}$ contains a terminal. Also, $V_B$ contains at least one terminal because $(V_A, V_B)$ is $\mathcal{T}$-separating.

Then as $G$ is connected, there is an edge between $V_{A1}$ and $V_B$. Then the cut $(V_{A1}, V_{A2} \cup V_B)$ has strictly fewer edges crossing, and also terminals on both sides, a contradiction to $(V_A, V_B)$ being the minimum $\mathcal{T}$-separating cut. □

The only bad case that prevents us from recursing is when the minimum terminal cut has a single terminal $s$ on some side. That is, one of the $s$-isolating cuts from Definition 5.10 is also a minimum terminal cut. We can cope with it via an extension of Lemma 5.8. Specifically, we show that for a cut with both sides connected, we can contract a side of the cut along with the cut edges before recursing.

**Lemma 5.15** *Let $F$ be a cut given by the partition $V = V_1 \cupdot V_2$ in $G = (V, E)$ such that both $G[V_1]$ and $G[V_2]$ are connected, and $\mathcal{T}_1 = V_1 \cap \mathcal{T}$ and $\mathcal{T}_2 = V_2 \cap \mathcal{T}$ be the partition of $\mathcal{T}$ induced by this cut. If $E_1^{intersect}$ intersects all $(\mathcal{T}_1 \cup \{v_2\}, c)$-terminal cuts in $G/V_2/F$, the graph formed by contracting all of $V_2$ and all edges in $F$ into a single vertex $v_2$, and similarly $E_2^{intersect}$ intersects all $(\mathcal{T}_2 \cup \{v_1\}, c)$-terminal cuts in $G/V_1/F$, then $E_1^{intersect} \cup E_2^{intersect} \cup F$ intersects all $(\mathcal{T}, c)$-cuts in $G$.*

**Proof:** Consider some cut $\widehat{F}$ of size at most $c$. If $\widehat{F}$ uses an edge from $F$, then it has at most $c - 1$ edges in $G \backslash F$, and thus in any connected component as well. If $\widehat{F}$ has at most $c - 1$ edges in $G[V_1]$, then no connected component in $V_1$ can have $c$ or more edges because removing $F$ already disconnected $V_1$ and $V_2$, and removing $E_1^{intersect}$ can only further disconnect the remaining components.

The only remaining case is when $\widehat{F}$ is entirely contained on one of the sides. Without loss of generality assume $\widehat{F}$ is entirely contained in $V_1$, i.e., $\widehat{F} \subseteq E(G[V_1])$. Because no edges from $G[V_2]$ and $F$ are removed and $G[V_2]$ is connected, all edges in $G[V_2]$ and $F$ must not be cut and hence can be contracted into a single vertex $v_2$. Using the induction hypothesis on the cut $\widehat{F}$ in $G/V_2/F$ with the terminal separation given by all of $\mathcal{T}_2$ replaced by $v_2$ gives that $\widehat{F}$ has at most $c - 1$ edges in any connected component of $(G/V_2/F) \backslash E_1^{intersect}$. Since connected components are unchanged under contracting connected subsets, we get that $\widehat{F}$ has at most $c - 1$ edges in any connected components of $G \backslash E_1^{intersect}$ as well. □

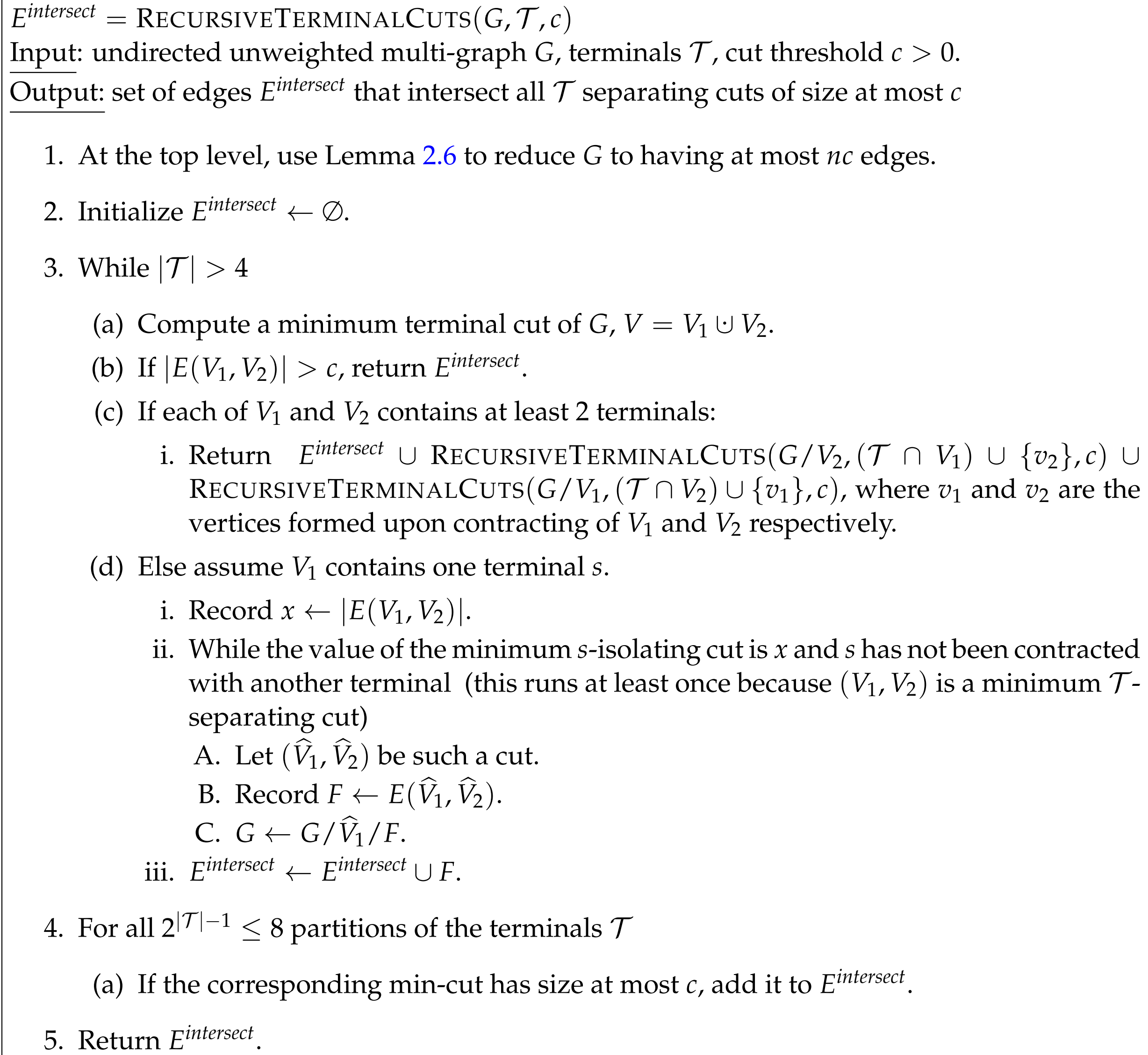
Now, a natural way to handle the case where a minimum terminal cut has a single terminal $s$ on some side is to use Lemma 5.15 to contract across the cut to make progress. However, it may be the case that for some $s \in \mathcal{T}$, there are many minimum $s$-isolating cuts: consider for example the length $n$ path with only the endpoints as terminals. If we always pick the edge closest to $s$ as the minimum $s$-isolating cut, we may have to continue $n$ rounds, and thus add all $n$ edges to our set of intersecting edges.

To remedy this, we instead pick a "maximal" $s$-isolating minimum cut. One way to find a maximal $s$-isolating cut is to repeatedly contract across an $s$-isolating minimum cut using Lemma 5.15 until its size increases. At that point, we add the last set of edges found in the cut to the set of intersecting edges. We have made progress because the value of the minimum $s$-isolating cut in the contracted graph must have increased by at least 1. While there are many ways to find a maximal $s$-isolating minimum cut, the way described here extends to our analysis in Section 5.2.3.

Pseudocode of this algorithm is shown in Figure 5, and the procedure for the repeated contractions to find a maximal $s$-isolating cut described in the above paragraph is in Line 3d.

**Discussion of algorithm in Figure 5.** We clarify some lines in the algorithm of Figure 5. If the algorithm finds a non-trivial $\mathcal{T}$-separating cut as the minimum terminal cut, it returns the result of the recursion in Line 3(c)i, and does not execute any of the later lines in the algorithm. In Line 3(d)ii, in addition to checking that the $s$-isolating minimum cut size is still $x$, we also must check that $s$ does not get contracted with another terminal. Otherwise, contracting across that cut makes global progress by reducing the number of terminals by 1. In Line 3(d)iiC, note that we can still view $s$ as a terminal in $G \leftarrow G/\widehat{V}_1/F$, as we have assumed that this contraction does not merge $s$ with any other terminals.

**Lemma 5.16** *For any graph $G$, terminals $\mathcal{T}$, and a value $c$, the algorithm* RecursiveTerminalCuts *as shown in Figure 5 runs in $O(n^2 c^4 \log n)$ time and returns a set at most $O(|\mathcal{T}| c^2)$ edges that intersect*



$E^{intersect} = \textsc{RecursiveTerminalCuts}(G, \mathcal{T}, c)$

Input: undirected unweighted multi-graph $G$, terminals $\mathcal{T}$, cut threshold $c > 0$.

Output: set of edges $E^{intersect}$ that intersect all $\mathcal{T}$ separating cuts of size at most $c$

1. At the top level, use Lemma 2.6 to reduce $G$ to having at most $nc$ edges.
2. Initialize $E^{intersect} \leftarrow \emptyset$.
3. While $|\mathcal{T}| > 4$
   (a) Compute a minimum terminal cut of $G$, $V = V_1 \cup V_2$.
   (b) If $|E(V_1, V_2)| > c$, return $E^{intersect}$.
   (c) If each of $V_1$ and $V_2$ contains at least 2 terminals:
      i. Return $E^{intersect} \cup \textsc{RecursiveTerminalCuts}(G/V_2, (\mathcal{T} \cap V_1) \cup \{v_2\}, c) \cup \textsc{RecursiveTerminalCuts}(G/V_1, (\mathcal{T} \cap V_2) \cup \{v_1\}, c)$, where $v_1$ and $v_2$ are the vertices formed upon contracting of $V_1$ and $V_2$ respectively.
   (d) Else assume $V_1$ contains one terminal $s$.
      i. Record $x \leftarrow |E(V_1, V_2)|$.
      ii. While the value of the minimum $s$-isolating cut is $x$ and $s$ has not been contracted with another terminal (this runs at least once because $(V_1, V_2)$ is a minimum $\mathcal{T}$-separating cut)
         A. Let $(\widehat{V}_1, \widehat{V}_2)$ be such a cut.
         B. Record $F \leftarrow E(\widehat{V}_1, \widehat{V}_2)$.
         C. $G \leftarrow G/\widehat{V}_1/F$.
      iii. $E^{intersect} \leftarrow E^{intersect} \cup F$.
4. For all $2^{|\mathcal{T}|-1} \leq 8$ partitions of the terminals $\mathcal{T}$
   (a) If the corresponding min-cut has size at most $c$, add it to $E^{intersect}$.
5. Return $E^{intersect}$.

Figure 5: Recursive algorithm using minimum terminal cuts for finding a set of edges that intersect all terminal cuts of size $\leq c$.

*all $(\mathcal{T}, c)$-cuts.*

**Proof:** We assume $m \leq nc$ throughout, as we can reduce to this case in $O(mc)$ time by Lemma 2.6. In line 3a, we use Theorem 5.13.

Note that the recursion in Line 3(c)i can only branch $O(|\mathcal{T}|)$ times, by the analysis in Lemma 5.12. Similarly, the case where $s$ gets contracted with another terminal in Line 3(d)ii can only occur $O(|\mathcal{T}|)$ times.

Therefore, we only create $O(|\mathcal{T}|)$ distinct terminals throughout the algorithm. Let $s$ be a terminal

created at some point during the algorithm. By monotonicity of cuts in Lemma 2.7, the minimum $s$-isolating cut can only increase in size $c$ times. Hence, $E^{intersect}$ is the union of $O(|\mathcal{T}|c)$ cuts of size at most $c$. Therefore, $E^{intersect}$ has at most $O(|\mathcal{T}|c^2)$ edges.

To bound the runtime, we use the total number of edges in the graphs in our recursive algorithm as a potential function. Thus, initially, this potential function has value $m$. Note that the recursion of Line 3(c)i can increase the potential function by $c$; hence, the total potential function increase throughout the algorithm is bounded by $m + O(c|\mathcal{T}|) = O(nc)$.

Each loop of Line 3(d)ii decreases our potential function by at least 1 from contractions. Thus, the total runtime of the loop involving Line 3(d)ii can be bounded by

$$O(mc) + O\left(m + nc^3 \log n\right) = O\left(nc^3 \log n\right),$$

where the former term is from running a maxflow algorithm up to flow $c$, and the latter is from applying Theorem 5.13. As the total increase in the total potential function is at most $O(nc)$, the loop in Line 3(d)ii can only execute $O(nc)$ times, for a total runtime of $O(n^2c^4 \log n)$ as desired. □

Our further speedup of this routine in Section 5.2.3 also uses a faster variant of RECURSIVETERMINALCUTS as base case, which happens when $|\mathcal{T}|$ is too small. Here the main observation is that a single maxflow computation is sufficient to compute a "maximal" $s$-isolating minimum cut, instead of the repeated contractions performed in RECURSIVETERMINALCUTS.

**Lemma 5.17** *For any graph $G$, terminals $\mathcal{T}$, and a value $c$, there is an algorithm that runs in $O(mc + n|\mathcal{T}|c^4 \log n)$ time and returns a set at most $O(|\mathcal{T}|c^2)$ edges that intersect all $(\mathcal{T}, c)$-cuts.*

**Proof:** We modify RECURSIVETERMINALCUTS as shown in Figure 5 and its analysis as given in Lemma 5.16 above. Specifically, we modify how we compute a maximal $s$-isolating minimum cut in Line 3(d)ii. For any partition $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, by submodularity of cuts it is known that there is a unique maximal subset $V_1 \subseteq V$ such that

$$\begin{aligned}
\mathcal{T}_1 &\subseteq V_1, \\
\mathcal{T}_2 &\subseteq V \setminus V_1, \\
|E(V_1, V_2)| &= |\text{mincut}(G, \mathcal{T}_1, \mathcal{T}_2)|.
\end{aligned}$$

Also, this maximal set can be computed in $O(mc)$ time by running the Ford-Fulkerson augmenting path algorithm with $\mathcal{T}_2$ as source and $\mathcal{T}_1$ as sink. The connectivity value of $c$ means at most $c$ augmenting paths need to be found, and the set $V_2$ can be set to the vertices that can still reach the sink set $\mathcal{T}_2$ in the residual graph [FH75]. Now set $V_1 = V \setminus V_2$. Thus by setting $\mathcal{T}_1 \leftarrow \{s\}$, we can use the corresponding computed set $V_1$ as the representative of the maximal $s$-isolating minimum terminal cut.

Now we analyze the runtime of this procedure. First, we reduce the number of edges to at most $nc$ in $O(mc)$ time. As in the proof of Lemma 5.16, all graphs in the recursion have at most $O(nc)$ edges. The recursion in Line 3(c)i can only branch $|\mathcal{T}|$ times, and we only need to compute $O(c|\mathcal{T}|)$ maximal $s$-isolating minimum terminal cuts throughout the algorithm. Each call Theorem 5.13 takes $O(m + nc^3 \log n) = O(nc^3 \log n)$ time, for a total runtime of $O(nc^3 \log n \cdot c|\mathcal{T}|) = O(n|\mathcal{T}|c^4 \log n)$ as desired. □

### 5.2.3 Using Local Cut Algorithms

A local cut algorithm is a tool that has recently been developed. Given a vertex $v$, there exists a local cut algorithm that determines whether there is a cut of size at most $c$ such that the side containing $v$ has volume at most $\nu$ in time linear in $c$ and $\nu$.

**Theorem 5.18 (Theorem 3.1 of [NSY19b])** *Let $G$ be a graph and let $v \in V(G)$ be a vertex. For a connectivity parameter $c$ and volume parameter $\nu$, there is an algorithm running in time $\widetilde{O}(c^2\nu)$ that with high probability either*

1. *Certifies that there is no cut of size at most $c$ such that the side with $v$ has volume at most $\nu$.*
2. *Returns a cut of size at most $c$ such that the side with $v$ has volume at most $130c\nu$.*

We now formalize the notion of the smallest cut that is *local* around a vertex $v$.

**Definition 5.19 (Local cuts)** *For a vertex $v \in G$, define* $\mathrm{LocalCut}(v)$ *to be*

$$\min_{\substack{V = V_1 \dot\cup V_2 \\ v \in V_1 \\ vol(V_1) \leq vol(V_2)}} |E(V_1, V_2)|.$$

We now combine Theorem 5.18 with the observation from Equation 1 in order to control the volume of the smaller side of the cut in an expander.

**Lemma 5.20** *Let $G$ be a graph with conductance at most $\varphi$, and let $\mathcal{T}$ be a set of terminals. If $|\mathcal{T}| \geq 500c^2\varphi^{-1}$ then for any vertex $s \in \mathcal{T}$ we can with high probability in $\widetilde{O}(c^3\varphi^{-1})$ time either compute* $\mathrm{LocalCut}(s)$ *or certify that* $\mathrm{LocalCut}(s) > c$.

**Proof:** We run binary search on the size of the minimum terminal cut with $s$ on the smaller side, and apply Theorem 5.18. The smaller side of a terminal cut has volume at most $c\varphi^{-1}$. Therefore, if $|\mathcal{T}| \geq 500c^2\varphi^{-1}$, then the cut returned by Theorem 5.18 for $\nu = c\varphi^{-1}$ will always be a terminal cut, as $130\nu c \leq |\mathcal{T}|/2$. The runtime is $\widetilde{O}(\nu c^2) = \widetilde{O}(c^3\varphi^{-1})$ as desired. □

We can substitute this faster cut-finding procedure into RECURSIVETERMINALCUTS to get the faster running time stated in Theorem 5.7 Part 2.

**Proof of Theorem 5.7 Part 2.** First, we perform expander decomposition, remove the inter-cluster edges, and add their endpoints as terminals.

Now, we describe the modifications we need to make to Algorithm RECURSIVETERMINALCUTS as shown in Figure 5. Let $\widehat{\mathcal{T}}$ be the set of terminals at the top level of recursion. The recursion gives that at most $O(|\widehat{\mathcal{T}}|)$ distinct terminals are created in the recursion.

First, we terminate if $|\mathcal{T}| \leq 500c^2\varphi^{-1}$ and use the result of Lemma 5.17. Otherwise, instead of using Theorem 5.13 for line 3a, we compute the terminal $s \in \mathcal{T}$ with minimal value of $\mathrm{LocalCut}(s)$. This gives us a minimum terminal cut. If the corresponding cut is a non-trivial $\mathcal{T}$-separating cut then we recurse as in Line 3(c)i. Otherwise, we perform the loop in Line 3(d)ii.

We now give implementation details for computing the terminal $s \in \mathcal{T}$ with minimal value of $\mathrm{LocalCut}(s)$. By Lemma 2.7 we can see that for a terminal $s$, $\mathrm{LocalCut}(s)$ is monotone throughout

the algorithm. For each terminal $s$, our algorithm records the previous value of LocalCut$(s)$ computed. Because this value is monotone, we need only check vertices $s$ whose value of LocalCut$(s)$ could still possibly be minimal. Now, either LocalCut$(s)$ is certified to be minimal among all $s$, or the value of LocalCut$(s)$ is higher than the previously recorded value. Note that this can only occur $O(c|\widehat{\mathcal{T}}|)$ times, as we stop processing a vertex $s$ if LocalCut$(s) > c$.

We now analyze the runtime. We first bound the runtime from the cases $|\mathcal{T}| \le 500c^2\varphi^{-1}$. The total number of vertices and edges in the leaves of the recursion tree is at most $O(mc)$. Therefore, by Lemma 5.17, the total runtime from these is at most

$$\widetilde{O}(500c^2\varphi^{-1} \cdot mc \cdot c^4) = \widetilde{O}(m\varphi^{-1}c^7).$$

Now, the loop of Line 3(d)ii can only execute $c\varphi^{-1}$ times because the volume of any $s$-isolating cut has size at most $c\varphi^{-1}$. Each iteration of the loop requires $\widetilde{O}(c^3\varphi^{-1})$ time by Lemma 5.20. Therefore, the total runtime of executing the loop and calls to it is bounded by

$$\widetilde{O}\left(c|\widehat{\mathcal{T}}| \cdot c\varphi^{-1} \cdot c^3\varphi^{-1}\right) = \widetilde{O}(|\widehat{\mathcal{T}}|\varphi^{-2}c^5).$$

Combining these shows Theorem 5.7 Part 2. □

# 6 Applications

We now discuss the applications of our connectivity-$c$ mimicking networks in dynamic graph data structures and parameterized algorithms.

## 6.1 Dynamic Offline $c$-edge-connectivity

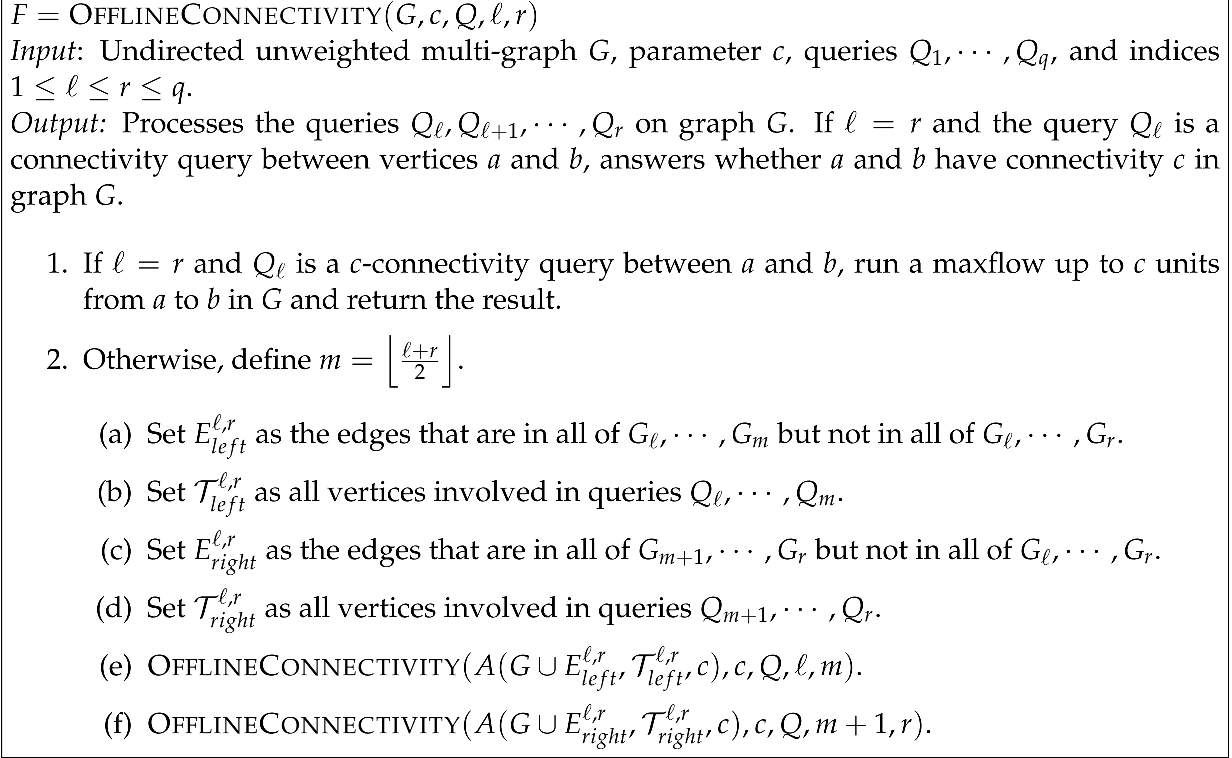
In this section we formally show how to use connectivity-$c$ mimicking network to obtain offline dynamic connectivity routines.

**Lemma 6.1** *Suppose that an algorithm $A(G', S, c)$ returns a connectivity-c mimicking network with $f(c)|S|$ edges for terminals $S$ on a graph $G'$ in time $\widetilde{O}(g(c)|E(G')|)$. Then there is an offline algorithm that on an initially empty graph $G$ answers $q$ edge insertion, deletion, and c-connectivity queries in total time $\widetilde{O}(f(c)(g(c)+c)q)$.*

Lemma 6.1 directly implies an analogous result when graph $G$ is not initially empty, as we can make the first $m$ queries simply insert the edges of $G$.

We now state the algorithm OfflineConnectivity in Figure 6 which shows Lemma 6.1. In Figure 6 graph $G_i$ for $0 \le i \le q$ denotes the current graph after queries $Q_1, \cdots, Q_i$ have been applied.

**Description of algorithm OfflineConnectivity.** The algorithm does a divide and conquer procedure, computing connectivity-$c$ mimicking network on the way down the recursion tree. As the algorithm moves down the recursion tree, it adds edges to our graph that will exist in all children of the recursion node. This is done in line 2a and 2c. The algorithm then computes all vertices involved in queries in the children of a recursion node in lines 2b and 2d, and computes a connectivity-$c$ mimicking network treating those vertices as terminals, and recurses.



$F = \text{OFFLINECONNECTIVITY}(G, c, Q, \ell, r)$

*Input*: Undirected unweighted multi-graph $G$, parameter $c$, queries $Q_1, \cdots, Q_q$, and indices $1 \leq \ell \leq r \leq q$.

*Output:* Processes the queries $Q_\ell, Q_{\ell+1}, \cdots, Q_r$ on graph $G$. If $\ell = r$ and the query $Q_\ell$ is a connectivity query between vertices $a$ and $b$, answers whether $a$ and $b$ have connectivity $c$ in graph $G$.

1. If $\ell = r$ and $Q_\ell$ is a $c$-connectivity query between $a$ and $b$, run a maxflow up to $c$ units from $a$ to $b$ in $G$ and return the result.
2. Otherwise, define $m = \left\lfloor \frac{\ell+r}{2} \right\rfloor$.
   (a) Set $E^{\ell,r}_{left}$ as the edges that are in all of $G_\ell, \cdots, G_m$ but not in all of $G_\ell, \cdots, G_r$.
   (b) Set $\mathcal{T}^{\ell,r}_{left}$ as all vertices involved in queries $Q_\ell, \cdots, Q_m$.
   (c) Set $E^{\ell,r}_{right}$ as the edges that are in all of $G_{m+1}, \cdots, G_r$ but not in all of $G_\ell, \cdots, G_r$.
   (d) Set $\mathcal{T}^{\ell,r}_{right}$ as all vertices involved in queries $Q_{m+1}, \cdots, Q_r$.
   (e) $\text{OFFLINECONNECTIVITY}(A(G \cup E^{\ell,r}_{left}, \mathcal{T}^{\ell,r}_{left}, c), c, Q, \ell, m)$.
   (f) $\text{OFFLINECONNECTIVITY}(A(G \cup E^{\ell,r}_{right}, \mathcal{T}^{\ell,r}_{right}, c), c, Q, m+1, r)$.

Figure 6: Algorithm for offline connectivity

**Proof:** Correctness directly follows from the algorithm description and Lemma 2.3 – we are only adding edges to our connectivity-$c$ mimicking network as we progress down the recursion tree. It suffices to bound the runtime.

It is straightforward to compute all the sets $E^{\ell,r}_{left}$ and $E^{\ell,r}_{right}$ in $O(q \log q)$ time. Additionally, $\sum_{\ell,r} |E^{\ell,r}_{left}| + |E^{\ell,r}_{right}| \leq O(q \log q)$, where the sum runs over all pairs $(\ell, r)$ encountered in an execution of OFFLINECONNECTIVITY.

The graph $G$ in a call to $\text{OFFLINECONNECTIVITY}(G, c, Q, \ell, r)$ has at most $O(f(c)(r - \ell))$ edges by the guarantees of algorithm $A$. Therefore, the runtime of calls to algorithm $A$ is bounded by

$$\widetilde{O}(g(c)) \cdot \sum_{\ell,r} \left( (r - \ell) f(c) + |E^{\ell,r}_{left}| + |E^{\ell,r}_{right}| \right)$$
$$\leq \widetilde{O}(g(c)) \cdot \left( O(q \log q) + \sum_{k=0}^{\log q} 2^k \cdot O(f(c) q \cdot 2^{-k}) \right) \leq \widetilde{O}(g(c) f(c) q).$$

The cost of running line 1 is bounded by $O(c \cdot f(c) q)$ as each graph in the leaf recursion nodes has at most $O(f(c))$ vertices. Hence the total runtime is at most $\widetilde{O}(f(c)(g(c) + c) q)$ as desired. □

Combining Lemma 6.1 and Theorem 1.1 Part 2 immediately gives a proof of Theorem 1.2.

Additionally, by adding / deleting edges from source / sinks, we can query for $c$-edge connectivity between multiple sets of vertices efficiently on a static graph.

**Corollary 6.2** *Given a graph $G$ with $m$ edges, as well as query subsets $(A_1, B_1), (A_2, B_2) \ldots (A_k, B_k)$, we can compute the value of $\mathrm{mincut}^c_G(A_i, B_i)$ for all $1 \leq i \leq k$ in $\widetilde{O}\left((m + \sum_i |A_i| + |B_i|)c^{O(c)}\right)$ time.*

## 6.2 Parameterized Algorithms for Network Design

In this section, we consider the rooted survivable network design problem (rSNDP), in which we are given a graph $G$ with edge-costs, as well as $h$ demands $(v_i, d_i) \in V \times \mathbb{Z}$, $i \in [h]$, and a root $r \in V$. The goal is to find a minimum-cost subgraph that contains, for every demand $(v_i, d_i)$, $i \in [h]$, $d_i$ edge-disjoint paths connecting $r$ to $v_i$.

We will show how to solve rSNDP optimally in the running time of $f(c, \mathrm{tw}(G))n$, where $c = \max_i d_i$ is the maximum demand, and $\mathrm{tw}(G)$ is the treewidth of $G$. Our algorithm uses the ideas of Chalermsook et al. [CDE+18] together with connectivity-$c$ mimicking networks. Our running time is $n \exp\left(O(c^4 \,\mathrm{tw}(G) \log(\mathrm{tw}(G)c)\right)$ which is double-exponential in $c$, but only single-exponential in $\mathrm{tw}(G)$ (whereas the result by Chalermsook et al. [CDE+18] is double-exponential in both $c$ and $\mathrm{tw}(G)$).

Let $(T, X)$ be a tree decomposition of $G$ (see Section 6.2.1 for a definition). The main idea of our algorithm is to assign, to each $t \in T$, a state representing the connectivity of the solution restricted to $X_t$. By assigning these states in a manner that they are consistent across $T$, we can piece together the solutions by looking at the states for each individual node. We will show that representing connectivity by two connectivity-$c$ mimicking networks is sufficient for our purposes, and that we can achieve consistency across $T$ by using very simple local rules between the state for a node $t$ and the states for its children $t_1$, $t_2$. These rules can be applied using dynamic programming to compute the optimum solution.

**Theorem 6.3** *There is an exact algorithm for* rSNDP *on a graph $G$ with treewidth $\mathrm{tw}(G)$ and maximum demand $c$ with a running time of $n \exp\left(O(c^4 \,\mathrm{tw}(G) \log(\mathrm{tw}(G)c)\right)$.*

The rest of this section is dedicated to proving the theorem above. In Section 6.2.1 we introduce some concepts and assumptions used in our result; in Section 6.2.2 we show how to represent the solution locally using connectivity-$c$ mimicking networks, and how to make sure that all these local representations are consistent; finally, in Section 6.2.3 we show how to use these ideas to solve rSNDP.

### 6.2.1 Preliminaries

**Tree Decomposition** Let $G$ be an undirected graph. A *tree decomposition* is a pair $(T, X)$ where $T$ is a tree and $X = \{X_t \subseteq V(G)\}_{t \in V(T)}$ is a collection of *bags* such that:

1. $V(G) = \bigcup_{t \in V(T)} X_t$, that is, every $v \in V(G)$ is contained in some bag $X_t$;
2. For any edge $uv \in E$, there is a bag $X_t$ that contains both $u$ and $v$, i.e., $u, v \in X_t$;
3. For each vertex $v \in V(G)$, the collection of nodes $t$ whose bags $X_t$ contain $v$ induces a connected subgraph of $T$, that is, $T[\{t \in V(T) : v \in X_t\}]$ is a (connected) subtree.

We will use the term *node* to refer to an element $t \in V(T)$, and *bag* to refer to the corresponding subset $X_t$.

The treewidth of $G$, denoted $\text{tw}(G)$, is the minimum *width* of any tree decomposition $(T, X)$ for $G$. The width of $(T, X)$ is given by $\max |X_t| - 1$.

Let $G$ be a graph and $(T, X)$ be its tree decomposition. We will say that each edge $uv \in E(G)$ *belongs* to a unique bag $X_t$, and write $e \in E_t$ if $t \in T$ is the node closest to the root such that $u, v \in X_t$. For a subset $S \subseteq V(T)$, we define $X(S) := \bigcup_{t \in S} X_t$. Given a node $t \in V(T)$, we denote by $T_t$ the subtree of $T$ rooted at $t$ and by $p(t)$ the parent node of $t$ in $V(T)$. We also define $G_t$ as the subgraph with vertices $X(T_t)$ and edges $E(G_t) = \bigcup_{t' \in T_t} E_{t'}$. For each $v \in V$, we denote by $t_v$ the node closest to the root for which $v \in X_{t_v}$.

Throughout this section, we will consider a tree decomposition $(T, X)$ of $G$ satisfying the following properties (see [CDE+18]): (i) $T$ has height $O(\log n)$; (ii) $|X_t| \leq O(\text{tw}(G))$ for all $t \in T$; (iii) every leaf bag contains no edges ($E_t = \emptyset$ for all leaves $t \in T$); (iv) every non-leaf has exactly 2 children. Additionally, we add the root $r$ to every bag $X_t$, $t \in T$.

**Vertex Sparsification** In our application of connectivity-$c$ mimicking networks to rSNDP, we need graphs that preserve the thresholded minimum cuts *for any disjoint sets* $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{T}$ (i.e. $\mathcal{T}_1 \cup \mathcal{T}_2$ may not include all terminals). Lemma 6.4 shows that this formulation is equivalent to that of Definition 2.1. We write $G \equiv^c_{\mathcal{T}} H$ if $G$ and $H$ are $(\mathcal{T}, c)$-equivalent according to the definition of Lemma 6.4.

**Lemma 6.4** *Let $G$, $H$ be graphs both containing a set of terminals $\mathcal{T}$. $G$ and $H$ are $(\mathcal{T}, c)$-equivalent if and only if for any* disjoint *subsets of terminals $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{T}$, the thresholded minimum cuts are preserved in $H$, i.e.,*

$$\text{mincut}^c_H(\mathcal{T}_1, \mathcal{T}_2) = \text{mincut}^c_G(\mathcal{T}_1, \mathcal{T}_2).$$

**Proof:** Note that if the condition above holds, $G$ and $H$ are trivially $(\mathcal{T}, c)$-equivalent, since for any partition $\mathcal{T} = \mathcal{T}_1 \uplus \mathcal{T}_2$, $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint.

We now prove that if Definition 2.1 is satisfied, thresholded minimum cuts are preserved for any disjoint subsets of terminals. Let $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{T}$ be disjoint sets of terminals. Let $(A_G, B_G)$, $(A_H, B_H)$ be the minimum cuts separating $\mathcal{T}_1$ and $\mathcal{T}_2$ in $G$ and $H$, respectively. We know that

$$\text{mincut}^c_G(A_H \cap \mathcal{T}, B_H \cap \mathcal{T}) \geq \text{mincut}^c_G(A_G \cap \mathcal{T}, B_G \cap \mathcal{T}),$$

since $(A_G, B_G)$ is the minimum cut separating $\mathcal{T}_1$, $\mathcal{T}_2$ in $G$, and $\text{mincut}^c_G(A_H \cap \mathcal{T}, B_H \cap \mathcal{T})$ represents a cut which also separates $\mathcal{T}_1$, $\mathcal{T}_2$. A similar statement is also true for $H$.

Furthermore, we know $\text{mincut}^c_G(A_H \cap \mathcal{T}, B_H \cap \mathcal{T}) = \text{mincut}^c_H(A_H \cap \mathcal{T}, B_H \cap \mathcal{T})$ (and similarly for $(A_G, B_G)$) by $(\mathcal{T}_c)$-equivalence of $G$ and $H$.

Combining everything, we get,

$$\begin{aligned}\text{mincut}^c_H(A_H \cap \mathcal{T}, B_H \cap \mathcal{T}) &= \text{mincut}^c_G(A_H \cap \mathcal{T}, B_H \cap \mathcal{T}) \geq \text{mincut}^c_G(A_G \cap \mathcal{T}, B_G \cap \mathcal{T}) \\ &= \text{mincut}^c_H(A_G \cap \mathcal{T}, B_G \cap \mathcal{T}) \geq \text{mincut}^c_H(A_H \cap \mathcal{T}, B_H \cap \mathcal{T})\end{aligned}$$

The circular chain of inequalities implies that

$$\text{mincut}^c_G(A_G \cap \mathcal{T}, B_G \cap \mathcal{T}) = \text{mincut}^c_H(A_H \cap \mathcal{T}, B_H \cap \mathcal{T})$$

The definition of $(A_G, B_G)$, $(A_H, B_H)$ then implies that

$$\text{mincut}^c_G(\mathcal{T}_1, \mathcal{T}_2) = \text{mincut}^c_G(A_G \cap \mathcal{T}, B_G \cap \mathcal{T}) = \text{mincut}^c_H(A_H \cap \mathcal{T}, B_H \cap \mathcal{T}) = \text{mincut}^c_H(\mathcal{T}_1, \mathcal{T}_2)$$

□

### 6.2.2 Local Connectivity Rules

In this section, we will introduce the local connectivity rules which will allow us to assign states in a consistent manner to the nodes of $T$. The states we will consider consist of two connectivity-$c$ mimicking networks roughly corresponding to the connectivity of the solution in $E(G_t)$ and $E \setminus E(G_t)$. We then present some rules that make these states consistent across $T$, while only being enforced for a node and its children.

We remark that this notation deviates from the one used by Chalermsook et al. [CDE+18], in which states represent connectivity in $E(G_t)$ and $E$. We do so because taking the union of overlapping connectivity-$c$ mimicking networks would lead to overcounting of the number of edge-disjoint paths.

The following local definition of connectivity introduces the desired consistency rules that we can use to define a dynamic program for the problem. Lemma 6.5 shows that a collection of connectivity-$c$ mimicking networks satisfy the local definition if and only if they represent the connectivity in $G$ with terminals given by a bag.

**Definition 6.5 (Local Connectivity)** *We say that the pairs of connectivity-$c$ mimicking networks $\{(\mathcal{H}'_t, \mathcal{H}_t)\}_{t \in V(T)}$ satisfy the* local connectivity definition *if*

$$\begin{aligned} \mathcal{H}'_t &\equiv^c_{X_t} (X_t, \emptyset) && \text{for every leaf node } t \text{ of } T \\ \mathcal{H}_{\text{root}(T)} &\equiv^c_{X_t} (X_t, \emptyset) \end{aligned}$$

*and for every internal node $t \in V(T)$ with children $t_1$ and $t_2$,*

$$\begin{aligned} \mathcal{H}'_t &\equiv^c_{X_t} (X_t, E_t) \cup \mathcal{H}'_{t_1} \cup \mathcal{H}'_{t_2} \\ \mathcal{H}_{t_1} &\equiv^c_{X_t} (X_t, E_t) \cup \mathcal{H}'_{t_2} \cup \mathcal{H}_t \end{aligned}$$

*where $A \equiv^c_{X_t} B$ means that* $\text{mincut}^c_A(S_1, S_2) = \text{mincut}^c_B(S_1, S_2)$ *for all disjoint sets $S_1, S_2 \subseteq X_t$.*

**Lemma 6.6** *Let $G = (V, E)$ be a graph, and $(\mathcal{T}, X)$ its tree decomposition satisfying [the usual properties]. For every $t \in V(T)$, let $(\mathcal{H}'_t, \mathcal{H}_t)$ be a pair as in Definition 6.5.*

*Then, the pairs $\{(\mathcal{H}'_t, \mathcal{H}_t)\}_{t \in T}$ satisfy the local definitions if and only if for every $t \in V(\mathcal{T})$,*

$$\begin{aligned} \mathcal{H}'_t &\equiv^c_{X_t} G_t \\ \mathcal{H}_t &\equiv^c_{X_t} G \setminus E(G_t) \end{aligned}$$

*where $A \equiv^c_{X_t} B$ means that* $\text{mincut}^c_A(S_1, S_2) = \text{mincut}^c_B(S_1, S_2)$ *for all disjoint sets $S_1, S_2 \subseteq X_t$.*

**Proof:** We start by proving the statement for $\mathcal{H}'$ by bottom-up induction, and then the one for $\mathcal{H}$ by top-down induction. We will show that

Let $t \in \mathcal{T}$ be a leaf of the tree decomposition. Then $E(G_t) = \emptyset$, so the statement immediately follows. Consider now an internal node $t$ with children $t_1$, $t_2$, and assume that the claim follows for $t_1$, $t_2$. We will define $H'_t = (X_t, E_t) \cup \mathcal{H}'_{t_1} \cup \mathcal{H}'_{t_2}$, and prove that $H'_t \equiv^c_{X_t} G_t$. That implies that $\mathcal{H}'_t \equiv^c_{X_t} H'_t$ (that is, $\mathcal{H}'_t$ satisfies the local connectivity definition) if and only if $\mathcal{H}'_t \equiv^c_{X_t} G_t$.

Let $S_1, S_2 \subseteq X_t$, and $F$ be the cutset for a mincut between $S_1$ and $S_2$ in $E(G_t)$. We will use $c_G(S_1, S_2) = \text{mincut}^c_G(S_1, S_2)$ for conciseness (in this proof only). Then

$$\begin{aligned}
&c_{G_t}(S_1, S_2) \\
&\quad = \min(c, |F|) \\
&\quad = \min(c, |F \cap E_t| + |F \cap E(G_{t_1})| + |F \cap E(G_{t_2})|) \\
&\quad \geq \min\big(c, c_{E_t}(S_1, S_2) + c_{E(G_{t_1})}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}) + c_{E(G_{t_2})}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})\big) \\
&\quad = \min\big(c, c_{E_t}(S_1, S_2) + c_{\mathcal{H}'_{t_1}}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}) + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})\big) \\
&\quad \geq c_{H'_t}(S_1, S_2)
\end{aligned}$$

The third inequality follows because each of the three terms corresponds to a min-cut between $S_1$ and $S_2$ for the respective edge sets. The fourth inequality follows by induction hypothesis, and the final one follows by definition of $H'_t$. For this last step, we crucially use that $X_{t_1} \cap X_{t_2} \subseteq X_t$, which means that any cut for $E_t$, $\mathcal{H}'_{t_1}$ and $\mathcal{H}'_{t_2}$ uses disjoint edges and disjoint vertices outside of $X_t$. These edges provide an upper bound for the cut $c_{H'_t}$.

Analogously, we can prove that $c_{G_t} \leq c_{H'_t}$, by taking a set of edges $F'$ of $H'_t$ that realizes the minimum cut in that graph. The same steps then apply to prove the desired inequality.

$$\begin{aligned}
c_{H'_t}(S_1, S_2) &= \min(c, |F'|) \\
&= \min(c, |F' \cap E_t| + |F' \cap E(\mathcal{H}'_{t_1})| + |F' \cap E(\mathcal{H}'_{t_2})|) \\
&\geq \min\big(c, c_{E_t}(S_1, S_2) + c_{\mathcal{H}'_{t_1}}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}) + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})\big) \\
&= \min\big(c, c_{E_t}(S_1, S_2) + c_{G_{t_1}}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}) + c_{G_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})\big) \\
&\geq c_{G_t}(S_1, S_2)
\end{aligned}$$

This concludes the first part of the proof.

For the second part of the proof, we will use top-down induction. For $t = r$, notice that $E \setminus E(G_t) = \emptyset$, so the statement follows. We now prove the equality for a node $t_1$ with parent $t$ and sibling $t_2$. Let $H_{t_1} = (X_t, E_t) \cup \mathcal{H}'_{t_2} \cup \mathcal{H}_t$, and prove that $H_t \equiv^c_{X_t} G \setminus G_t$. This implies the statement, as it shows that $\mathcal{H}_t \equiv^c_{X_t} H_t$ (that is, $\mathcal{H}_t$ satisfies the local connectivity definition) if and only if $\mathcal{H}_t \equiv^c_{X_t} G \setminus G_t$.

Let $S_1, S_2 \subseteq X_{t_1}$, and $F$ be the cutset for a mincut between $S_1$ and $S_2$ in $E \setminus E(G_{t_1})$. Then

$$\begin{aligned}
c_{E\setminus E(G_{t_1})}(S_1, S_2) &= \min(c, |F|) \\
&= \min(c, |F \cap E_t| + |F \cap (E \setminus E(G_t))| + |F \cap E(G_{t_2})|) \\
&\geq \min(c, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{E\setminus E(G_t)}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad\qquad + c_{E(G_{t_2})}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&= \min(c, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{\mathcal{H}_t}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad\qquad + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\geq c_{H_{t_1}}(S_1, S_2)
\end{aligned}$$

Similarly to the proof above, we use the fact that $F \cap E_t$, $F \cap (E \setminus E(G_t))$, $F \cap E(G_{t_2})$ are cuts in the subgraphs $E_t$, $G \setminus E(G_t)$, $E(G_{t_2})$ respectively. The last step follows from the fact that the three terms correspond to cuts in $E_t$, $\mathcal{H}_t$ and $\mathcal{H}'_{t_2}$, and therefore their union forms a cut in $\mathcal{H}_t \cup E_t \cup \mathcal{H}'_{t_2}$. Since $H_{t_1} \equiv^c_{X_{t_1}} \mathcal{H}_t \cup E_t \cup \mathcal{H}'_{t_2}$, the inequality follows. The converse follows similarly:

$$\begin{aligned}
c_{H_{t_1}}(S_1, S_2) &= \min(c, |F|) \\
&= \min(c, |F \cap E_t| + |F \cap E(\mathcal{H}_t)| + |F \cap E(\mathcal{H}'_{t_2})|) \\
&\geq \min(c, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{\mathcal{H}_t}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad\qquad + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&= \min(c, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{E\setminus E(G_t)}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad\qquad + c_{E(G_{t_2})}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\geq c_{E\setminus E(G_{t_1})}(S_1, S_2)
\end{aligned}$$

This completes the proof. □

#### 6.2.3 Dynamic Program for rSNDP

In this section, we present an algorithm for rSNDP on bounded-treewidth graphs, which uses dynamic programming to compute a solution bottom-up. Our goal is to assign two connectivity-$c$ mimicking networks $\mathcal{H}'_t$, $\mathcal{H}_t$ to each node $t \in T$, corresponding to the connectivity of the solution in $E(G_t)$ and $E \setminus E(G_t)$. We argue that any solution for $G_t$, $t \in T$ that is compatible with a state $(\mathcal{H}'_t, \mathcal{H}_t)$ can be interchangeably used, which implies that the dynamic program will obtain the minimum-cost solution.

We define a dynamic programming table $D$, with entries $D[t, \mathcal{H}', \mathcal{H}]$, $t \in T$, $\mathcal{H}', \mathcal{H}$ a connectivity-$c$ mimicking networks with terminal set $X_t$. The entry $D[t, \mathcal{H}', \mathcal{H}]$ represents the minimum cost of a solution $F$ that is consistent with $\mathcal{H}'$ (i.e. $F \equiv^c_{X_t} \mathcal{H}'$), such that $F \cup \mathcal{H}_t$ satisfies all the demands contained in $G_t$.

We compute $D[t, \mathcal{H}', \mathcal{H}]$ as follows:

- For any leaf $t$, set $D[t, \emptyset, \mathcal{H}] = 0$ and $D[t, \mathcal{H}', \mathcal{H}] = +\infty$ for $\mathcal{H}' \neq \emptyset$;
- For the root node root$(T)$, set $D[\text{root}(T), \mathcal{H}', \mathcal{H}] = +\infty$ if $\mathcal{H} \neq \emptyset$;

- For any demand $(v_i, d_i)$, and $t \in T$ such that $v_i \in X_t$, set $D[t, \mathcal{H}', \mathcal{H}] = +\infty$ if $\mathcal{H}' \cup \mathcal{H}$ contains fewer than $d_i$ edge-disjoint paths connecting $r$ to $v_i$.

For all other entries of $T$, compute it recursively as:

$$
\begin{aligned}
D[t, \mathcal{H}', \mathcal{H}] = \min \Big\{ w(Y) + D[t_1, \mathcal{H}'_1, \mathcal{H}_1] + D[t_2, \mathcal{H}'_2, \mathcal{H}_2] : Y \subseteq E_t,& \\
\mathcal{H}' \equiv^c_{X_t} Y \cup \mathcal{H}'_1 \cup \mathcal{H}'_2,& \\
\mathcal{H}_1 \equiv^c_{X_{t_1}} Y \cup \mathcal{H} \cup \mathcal{H}'_2,& \\
\mathcal{H}_2 \equiv^c_{X_{t_2}} Y \cup \mathcal{H} \cup \mathcal{H}'_1 \Big\}&
\end{aligned}
$$

We now want to prove that the dynamic program is feasible, i. e. that the entries $D[\text{root}(T), \mathcal{H}', \emptyset]$ correspond to feasible solutions; and that it is optimal, meaning that we will obtain the optimum solution to the problem.

To prove that the dynamic program is feasible, notice that, by definition, any solution obtained induces a choice of $Y_t$, $\mathcal{H}'_t$, $\mathcal{H}_t$ for each $t \in T$. Let $Y = \cup_{t \in T}$. The recursion formula of the dynamic program implies that the pairs $\{(\mathcal{H}'_t, \mathcal{H}_t)\}_{t \in T}$ satisfy the local connectivity definition with regard to the graph $(V, Y)$.

By Lemma 6.6, this implies that

$$
\mathcal{H}'_t \equiv^c_{X_t} G_t[Y], \mathcal{H}_t \equiv^c_{X_t} G[Y] \setminus E(G_t),
$$

and hence, $\mathcal{H}'_t \cup \mathcal{H}_t \equiv^c_{X_t} G[Y]$.

Let $(v_i, d_i)$ be a demand and $t \in T$ be a node such that $v_i \in X_t$. Since we know that $\mathcal{H}'_t \cup \mathcal{H}_t$ contains $d_i$ edge-disjoint paths from $r$ to $v_i$ (otherwise $D[t, \mathcal{H}'_t, \mathcal{H}_t] = +\infty$), then we know that the minimum cut separating $r$ from $v_i$ has at least $d_i$ edges, which implies that $Y$ must also contain $d_i$ edge-disjoint paths connecting $r$ and $v_i$.

For the converse, we will prove that any feasible solution $F$ can be captured by the dynamic program. Given $F$, it is sufficient to take $\mathcal{H}'_t$, $\mathcal{H}_t$ to be connectivity-$c$ mimicking networks for $G_t[F]$, $G[F] \setminus E(G_t)$, respectively. By Lemma 6.6 (applied to graph $(V, F)$), we know that $\{(\mathcal{H}'_t, \mathcal{H}_t)\}_{t \in T}$ satisfy the local connectivity definition for $(V, F)$, and therefore $D[t, \mathcal{H}'_t, \mathcal{H}_t]$ can be computed recursively from $D[t_1, \mathcal{H}'_{t_1}, \mathcal{H}_{t_1}]$, $D[t_2, \mathcal{H}'_{t_2}, \mathcal{H}_{t_2}]$, $Y_t = F \cap E_t$.

Let $(v_i, d_i)$ be a demand and $t \in T$ be a node such that $v_i \in X_t$. Since $F$ is a feasible solution, it contains $d_i$ edge-disjoint paths from $r$ to $v_i$, and therefore $\text{mincut}^c_F(\{r\}, \{v_i\}) \geq d_i$. This implies that $\text{mincut}^c_{\mathcal{H}'_t \cup \mathcal{H}_t}(\{r\}, \{v_i\}) \geq d_i$, and thus $\mathcal{H}'_t \cup \mathcal{H}_t$ contains $d_i$ edge-disjoint paths from $r$ to $v_i$ (and is a valid entry of $T$).

We conclude that the dynamic program above computes an optimum solution for rSNDP. By Theorem 1.1, there is a connectivity-$c$ mimicking network containing $O(wc^4)$ edges (and $O(wc^4)$ vertices as well), for any graph with $w$ terminals. Hence, there are at most

$$
\left(|V|^2\right)^{|E|} = \left(w^2 c^8\right)^{wc^4} = \exp\left(O(c^4 w \log(wc))\right)
$$

possibilities for such connectivity-$c$ mimicking networks. This implies that the dynamic programming table has $n \exp\left(O(c^4 \operatorname{tw}(G) \log(\operatorname{tw}(G)c)\right)$ entries. The work required to compute the value of each entry takes time

$$\exp\left(O(c^4 \operatorname{tw}(G) \log(\operatorname{tw}(G)c)\right) \cdot w^{O(c)} \cdot \operatorname{poly}(c, w)$$

(considering all combinations of states for children nodes, all disjoint subsets of terminals, compute the min-cuts and check if they match).

We conclude that the running time of the algorithm is $n \exp\left(O(c^4 \operatorname{tw}(G) \log(\operatorname{tw}(G)c)\right)$, which completes the proof.

**Acknowledgements:** Parinya Chalermsook has been supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557) and by the Academy of Finland Research Fellowship, under the grant number 310415. Richard Peng is partially supported by the US National Science Foundation under grant number 1846218. Yunbum Kook has been supported by the Institute for Basic Science (IBS-R029-C1). Yang P. Liu has been supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program. Bundit Laekhanukit has been supported by the 1000-talents award by the Chinese government and by Science and Technology Innovation 2030 – "New Generation of Artificial Intelligence" Major Project No.(2018AAA0100903), NSFC grant 61932002, Program for Innovative Research Team of Shanghai University of Finance and Economics (IRTSHUFE) and the Fundamental Research Funds for the Central Universities. Daniel Vaz has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

## References

[ADK+16] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng. On fully dynamic graph sparsifiers. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 335–344, Oct 2016.

[AGK14] Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\varepsilon)$-approximate flow sparsifiers. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 279–293. Society for Industrial and Applied Mathematics, 2014.

[AKLT15] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Dynamic sketching for graph optimization problems with applications to cut-preserving sketches. In *FSTTCS*, 2015. Available at https://arxiv.org/abs/1510.03252.

[AKT19] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. *CoRR*, abs/1901.01412, 2019. Available at: http://arxiv.org/abs/1901.01412.

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

[AWY15] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 41–50, 2015.

[BHKP08] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Fast edge splitting and edmonds' arborescence construction for unweighted graphs. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 455–464. Society for Industrial and Applied Mathematics, 2008.

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

[CDE+18] Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CH03] Richard Cole and Ramesh Hariharan. A fast algorithm for computing steiner edge connectivity. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 167–176, 2003.

[Che18] Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 170–181, 2018.

[Chu12] Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 673–688. ACM, 2012.

[CLLM10] Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 265–274, 2010.

[CMSV17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log W)$ time (extended abstract). In *SODA*, pages 752–771. SIAM, 2017.

[CSWZ00] Shiva Chaudhuri, KV Subrahmanyam, Frank Wagner, and Christos D Zaroliagis. Computing mimicking networks. *Algorithmica*, 26(1):31–49, 2000.

[DGGP19] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 914–925, 2019. Available at: https://arxiv.org/abs/1906.10530.

[DKP+17] David Durfee, Rasmus Kyng, John Peebles, Anup B Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 730–742. ACM, 2017. Available at: https://arxiv.org/abs/1611.07451.

[DV94] Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 716–725. ACM, 1994.

[DV95] Ye. Dinitz and A. Vainshtein. Locally orientable graphs, cell structures, and a new algorithm for the incremental maintenance of connectivity carcasses. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 302–311, 1995.

[DW98] Yefim Dinitz and Jeffery R. Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20(3):242–276, 1998.

[EGK+10] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *CoRR*, abs/1006.4586, 2010.

[EGK+14] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014.

[Epp94] David Eppstein. Offline algorithms for dynamic minimum spanning tree problems. *J. Algorithms*, 17(2):237–250, September 1994.

[FH75] Delbert Ray Fulkerson and Gary Harding. On edge-disjoint branchings. Technical report, CORNELL UNIV ITHACA NY DEPT OF OPERATIONS RESEARCH, 1975.

[FHKQ16] Stefan Fafianie, Eva-Maria C. Hols, Stefan Kratsch, and Vuong Anh Quyen. Preprocessing under uncertainty: Matroid intersection. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 35:1–35:14, 2016. Available at https://core.ac.uk/download/pdf/62922404.pdf.

[FKQ16] Stefan Fafianie, Stefan Kratsch, and Vuong Anh Quyen. Preprocessing under uncertainty. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47, pages 33:1–33:13, 2016.

[FLPS16] Fedor V Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM (JACM)*, 63(4):1–60, 2016.

[FY19] Sebastian Forster and Liu Yang. A faster local algorithm for detecting bounded-size cuts with applications to higher-connectivity problems. *CoRR*, abs/1904.08382, 2019.

[GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):pp. 551–570, 1961.

[GHP17a] Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 44:1–44:14, 2017. Available at: https://arxiv.org/abs/1702.01136.

[GHP17b] Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[GHP17c] Gramoz Goranci, Monika Henzinger, and Pan Peng. The power of vertex sparsifiers in dynamic graph algorithms. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 45:1–45:14, 2017. Available at: https://arxiv.org/abs/1712.06473.

[GHP18] Gramoz Goranci, Monika Henzinger, and Pan Peng. Dynamic effective resistances and approximate schur complement on separable graphs. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 40:1–40:15, 2018. Available at: https://arxiv.org/abs/1802.09111.

[GHT16] Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in poly-logarithmic amortized update time. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 57. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[GR16] Gramoz Goranci and Harald Räcke. Vertex sparsification in trees. In *Approximation and Online Algorithms - 14th International Workshop, WAOA 2016, Aarhus, Denmark, August 25-26, 2016, Revised Selected Papers*, pages 103–115, 2016. Available at: https://arxiv.org/abs/1612.03017.

[GT14] Andrew V. Goldberg and Robert Endre Tarjan. Efficient maximum flow algorithms. *Commun. ACM*, 57(8):82–89, 2014.

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[HK99] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[HKNR98] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

[HKP07] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient algorithms for computing all low st edge connectivities and related problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 127–136. Society for Industrial and Applied Mathematics, 2007.

[JS20] Wenyu Jin and Xiaorui Sun. Fully dynamic c-edge connectivity in subpolynomial time. *CoRR*, abs/2004.07650, 2020.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, January 2000.

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

[KLP+16] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 842–850. ACM, 2016. Available at http://arxiv.org/abs/1512.01892.

[KPZP18] Nikolai Karpov, Marcin Pilipczuk, and Anna Zych-Pawlewicz. An exponential lower bound for cut sparsifiers in planar graphs. *Algorithmica*, pages 1–14, 2018.

[KR13] Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1789–1799. SIAM, 2013.

[KR14] Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014.

[KR17] Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. *CoRR*, abs/1702.05951, 2017.

[KS16] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 573–582, 2016. Available at http://arxiv.org/abs/1605.02353.

[KW12] Stefan Kratsch and Magnus Wahlstrom. Representative sets and irrelevant vertices: New tools for kernelization. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS '12, pages 450–459, 2012. Available at https://arxiv.org/abs/1111.2195.

[LM10] F Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 47–56. ACM, 2010.

[Lov77] László Lovász. Flats in matroids and geometric graphs. In *Combinatorial Surveys (Proc. 6th British Combinatorial Conference*, pages 45–86, 1977.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 253–262. IEEE, 2013. Available at http://arxiv.org/abs/1307.2205.

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *FOCS*, pages 593–602. IEEE Computer Society, 2016.

[Mar06] Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

[Mar09] Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.

[MM10] Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 255–264, 2010.

[Moi09] Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 3–12. IEEE, 2009.

[MS18] Antonio Molina and Bryce Sandlund. Historical optimization with applications to dynamic higher edge connectivity. 2018.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

[NSY19a] Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 241–252, 2019. Available at: https://arxiv.org/abs/1904.04453.

[NSY19b] Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small vertex connectivity in near-linear time and queries. *CoRR*, abs/1905.05329, 2019. Available at : http://arxiv.org/abs/1905.05329.

[Pen16] Richard Peng. Approximate undirected maximum flows in $O(m\text{poly}\log(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1862–1867, 2016.

[PSS19] Richard Peng, Bryce Sandlund, and Daniel Dominic Sleator. Optimal offline dynamic 2, 3-edge/vertex connectivity. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, pages 553–565, 2019. Available at: https://arxiv.org/abs/1708.03812.

[Ree97] Bruce A Reed. Tree width and tangles: A new connectivity measure and some applications. *Surveys in combinatorics*, pages 87–162, 1997.

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

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *FOCS*, pages 263–269. IEEE Computer Society, 2013.

[She17] Jonah Sherman. Area-convexity, $l_\infty$ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 452–460, 2017.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90. ACM, 2004.

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635. SIAM, 2019. Available at: https://arxiv.org/abs/1812.08958.

[vdBS19] Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. *CoRR*, abs/1907.07982, 2019.

# A Deferred Proofs

## A.1 Proof of Lemma 2.6

Consider the following routine: repeat $c$ iterations of finding a maximal spanning forest from $G$, remove it from $G$ and add it to $H$.

Each of the steps takes $O(m)$ time, for a total of $O(mc)$. Also, a maximal spanning tree has the property that for every non-empty cut, it contains at least one edge from it. Thus, for any cut $\partial(S)$, the $c$ iterations add at least

$$\min\left\{c, \left|\partial\left(S\right)\right|\right\}$$

edges to $H$, which means that up to a value of $c$, all cuts in $G$ and $H$ are the same.

## A.2 Proof of Lemma 4.2

Let $G' = G/X$ be the contracted graph and $v_X$ be the contracted vertex in $G'$ that is obtained by contracting $G[X]$. Since we do not contract the terminals, it suffices to show that, for any two subsets $X_A, X_B \subseteq \mathcal{T}$, we have $\text{mincut}^c_{G'}(X_A, X_B) = \text{mincut}^c_G(X_A, X_B)$.

Starting with $\text{mincut}^c_{G'}(X_A, X_B) \geq \text{mincut}^c_G(X_A, X_B)$, we can see that all the edges in $G'$ are also in $G$, which implies that any cutset in $G'$ is also in $G$. We conclude that the size of the minimum cut

in $G$ must be at most the size of the minimum cut in $G'$, for any pair of terminals sets. In general, we can say that contraction of edges only ever increases connectivity, which implies the above.

Let us now show the converse, that is, $\text{mincut}^c_{G'}(X_A, X_B) \leq \text{mincut}^c_G(X_A, X_B)$. Since we are in the unweighted setting, it is sufficient to consider $|X_A|, |X_B| \leq c$. Suppose that $\text{mincut}^c_{G'}(X_A, X_B) = \ell \leq c$. Then there must be $\ell$ disjoint paths connecting $X'_A \subseteq X_A$ to $X'_B \subseteq X_B$ such that $|X'_A| = |X'_B| = \ell$. Denote the set of such paths in $G'$ by $\mathcal{P}'$.

We will construct the set of edge-disjoint paths $\mathcal{P}$ in $G$ connecting $X'_A$ to $X'_B$, thus implying that $\text{mincut}^c_G(X_A, X_B) \geq \ell$. We write $\mathcal{P}'$ as $\mathcal{P}' = \mathcal{P}'_1 \cup \mathcal{P}'_2$ where $\mathcal{P}'_1$ are the paths that do not go through the contracted vertex $v_X$. We add the paths in $\mathcal{P}'_1$ to $\mathcal{P}$, since they correspond to edge disjoint paths in the original graph $G$. For paths in $\mathcal{P}'_2$, we will need to specify their behavior inside the contracted set $G[X]$. Let $E_{in} \subseteq \partial(X)$ be the set of boundary edges of $X$ that paths in $\mathcal{P}'_2$ use to enter $v_X$; analogously, we define $E_{out} \subseteq \partial(X)$. Notice that $|E_{in}| = |E_{out}| = |\mathcal{P}'_2| \leq c$. Since $X$ is connectivity-$c$ well-linked, there is a collection of disjoint paths $\mathcal{P}_X$ connecting $E_{in}$ to $E_{out}$. We stitch the three parts of the paths in $\mathcal{P}'_2$ and $\mathcal{P}_X$ together to add to $\mathcal{P}$: (1) a subpath of some path $P \in \mathcal{P}'_2$ from a node in $X'_A$ to $E_{in}$, (2) a path in $\mathcal{P}_X$ from that edge in $E_{in}$ to an edge in $E_{out}$, and (3) a subpath of some path $Q \in \mathcal{P}'_2$ from the same an edge in $E_{out}$ to a node in $X'_B$. We remark that, even though $\mathcal{P}$ contains $\ell$ edge-disjoint paths connecting $X'_A$ to $X'_B$, the pairing induced by $\mathcal{P}$ and $\mathcal{P}'$ may be different.

### A.3 Proof of Theorem 1.1 Part 2

By Lemma 5.6, Algorithm 3 computes a set $E^{contain}$ of edges that contains all $(\mathcal{T}, c)$-cuts. Reduce to $m \leq nc$ by Lemma 2.6. Let $C_{int}$ be a constant such that part 2 of Theorem 5.7 gives us a set $E^{intersect}$ of edges intersecting all $(\mathcal{T}, c)$-cuts of size at most $C_{int}(\varphi m \log^4 n + |\mathcal{T}|)c^2$ in $\widetilde{O}(m\varphi^{-2}c^7)$ time. Let $E^{contain}_i$ be the set of edges $E^{contain}$ after the iteration $\widehat{c} = i$. Let $\widehat{\mathcal{T}}$ be the terminals at the start of the algorithm. We show by induction that before processing $\widehat{c} = i$ in the second line of Figure 3 that

$$\left|E_i^{contain}\right| \leq (4C_{int})^{c-i} \frac{(c!)^2}{(i!)^2} \left(\varphi m \log^4 n + \left|\widehat{\mathcal{T}}\right|\right)$$

and

$$\left|V(E_i^{contain})\right| \leq 2\,(4C_{int})^{c-i} \frac{(c!)^2}{(i!)^2} \left(\varphi m \log^4 n + \left|\widehat{\mathcal{T}}\right|\right).$$

Since $|V(\widehat{E})| \leq 2|\widehat{E}|$ for any set of edges $\widehat{E}$, it suffices to bound $|E^{contain}_i|$. The induction hypothesis holds for $i = c$. By Part 2 of Theorem 5.7 we have the size of $E^{contain}$ after processing $\widehat{c} = i$ is at most

$$\begin{aligned}
&C_{int}\left(\varphi m \log^4 n + |V(E_i^{contain})|\right)\widehat{c}^2 + |E_i^{contain}| \\
&\leq C_{int}\left(\varphi m \log^4 n + 2\,(4C_{int})^{c-i} \frac{(c!)^2}{(i!)^2}\left(\varphi m \log^4 n + \left|\widehat{\mathcal{T}}\right|\right)\right) i^2 + (4C_{int})^{c-i}\frac{(c!)^2}{(i!)^2}\left(\varphi m \log^4 n + \left|\widehat{\mathcal{T}}\right|\right) \\
&\leq (4C_{int})^{c-i+1} \frac{(c!)^2}{(i-1)!^2}\left(\varphi m \log^4 n + \left|\widehat{\mathcal{T}}\right|\right)
\end{aligned}$$

as desired. Taking $i = 0$ shows that the final size of $E^{contain}$ is at most $(4C_{int})^c (c!)^2 (\varphi m \log^4 n + |\widehat{\mathcal{T}}|)$.

Then, we use the choice of conductance threshold

$$\varphi = \frac{1}{5c\,(4C_{int})^c\,(c!)^2 \log^4 n}.$$

Because $m \leq nc$, the final size of $E^{contain}$ is at most

$$(4C_{int})^c\, c! \left(\varphi m \log^4 n + \left|\widehat{\mathcal{T}}\right|\right) \leq \frac{n}{5} + (4C_{int})^c\, c! \left|\widehat{\mathcal{T}}\right|.$$

Now, we apply Lemma 5.2 to produce a graph $H$ with at most $\frac{n}{5} + (4C_{int})^c (c!)^2 |\widehat{\mathcal{T}}| + 1$ vertices that is $(\widehat{\mathcal{T}}, c)$-equivalent to $G$. Now, we can repeat the process on $H$ $O(\log n)$ times. The number of vertices in the graphs we process decrease geometrically until they have at most $2(4C_{int})^c (c!)^2 |\widehat{\mathcal{T}}| = O(c)^{2c} |\widehat{\mathcal{T}}|$ many vertices.

Now, combining the runtime of $\widetilde{O}(m\varphi^{-2}c^7)$ along with our choice of $\varphi$ above gives a vertex sparsifier with $|\widehat{\mathcal{T}}| \cdot O(c)^{2c}$ edges in time $O(m \cdot c^{O(c)} \cdot \log^{O(1)} n)$, as desired.

# B Efficiently Finding a Violating Cut

Although our proof in Section 4 of existence of connectivity-$c$ mimicking networks with $O(kc^4)$ edges uses the concept of a violating cut, we do not explicitly find the violating cuts. In this section, we present a parametrized algorithm running in time $2^{O(c^2)} k^2 m$ for finding violating cuts.

Let $G = (V, E)$ be a graph and $\mathcal{T} \subseteq V$ a set of terminals, and let $X = V \setminus \mathcal{T}$ be the set of non-terminal vertices in $G$. For simplicity, we will assume that our terminals are in one-to-one correspondence with $\partial(X) = E_G(X, V(G) - X)$, that is, that all edges in $\partial(X)$ have different endpoints outside $X$. By abuse of notation, we write $\mathcal{T} = \partial(X)$ and $k = |\partial(X)|$. Furthermore, this assumption implies that all terminals have degree 1.

Observe first that a violating cut can be found in $k^{O(c)}\widetilde{O}(m)$ time by simply computing all possible minimum cuts separating any disjoint subsets of terminals $\mathcal{T}_0, \mathcal{T}_1 \subseteq \mathcal{T}$ of size $q \leq c$ whose minimum cut contains less than $q$ edges. However, as we are aiming for a running time of $f(c)\text{poly}(k, m)$, we cannot afford to enumerate all the possible minimum cuts to find the "correct" disjoint subsets $\mathcal{T}_0, \mathcal{T}_1 \subseteq \mathcal{T}$.

Our algorithm actually solves a more general problem. We say that a cut $(A_0, A_1)$ of $G$ is a valid $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut if

- $Q_0 \subseteq A_0 \setminus \mathcal{T}$ and $Q_1 \subseteq A_1 \setminus \mathcal{T}$.
- $|A_j \cap \mathcal{T}| \geq c_j$ for $j = 0, 1$.
- $E_G(A_0, A_1)$ contains at most $\ell$ edges.

In other words, $Q_0$ and $Q_1$ are the non-terminals that are "constrained" to be on different sides. The values of $c_0$ and $c_1$ are the minimum required number of terminals on the sides of $A_0$ and $A_1$ respectively. We will refer to the two sides of the cuts as *zero side* and *one side*, respectively.

**Observation B.1** *Given a subroutine that finds a valid $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut in time given by some function $T(m, k, \max(c_0, c_1, \ell))$, there exists an algorithm that either returns a violating cut in $G$ or reports that such a cut does not exist in time $O(cT(m, k, c))$.*

In the rest of the section, we shall describe an algorithm that finds a valid $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut. Let $c = \max(c_0, c_1, \ell)$. Our algorithm has two steps, encapsulated in the following two lemmas.

**Lemma B.2 (Reduction)** *There is an algorithm that runs in time $2^{O(c^2)} \cdot k^2 \cdot m$, and reduces the problem of finding a valid $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut to at most $2^{O(c^2)}$ instances of finding valid $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut where $\min(c'_0, c'_1) = 0$.*

We remark that each of these instances may have different parameters (of the constrained cut). The only property they have in common is that $\min(c'_0, c'_1) = 0$; that is, there is only a one-sided terminal requirement.

**Lemma B.3 (Base case)** *For $\ell \leq c$, there is an algorithm that finds a valid $(Q_0, Q_1, 0, c, \ell)$-constrained cut (and analogously, $(Q_0, Q_1, c, 0, \ell)$-constrained) in time $2^{O(c^2)} \cdot k^2 \cdot m$.*

The following theorem follows in a straightforward manner since every violating cut is also $(\emptyset, \emptyset, \ell + 1, \ell + 1, \ell)$-constrained, for some $\ell \in [c - 1]$.

**Theorem B.4** *There is an algorithm that runs in time $2^{O(c^2)} \cdot k^2 \cdot m$ and either returns a violating cut or reports that such a cut does not exist.*

## B.1 The reduction to the base case

In this subsection, we prove Lemma B.2. The main ingredient for doing so is the following lemma.

**Lemma B.5** *There is a reduction from $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut to solving at most $2^{O(c)}$ instances of finding valid $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut where $(c'_0 + c'_1) < (c_0 + c_1)$.*

In other words, this lemma allows us to reduce the number of required terminals on at least one of the sides by one. Applying Lemma B.5 recursively will allow us to turn an input instance of $(Q_0, Q_1, c_0, c_1, \ell)$ constrained cut into at most $2^{O(c^2)}$ instances of the base problem: This follows from the fact that at every recursive call, the value of at least one of $c_0$ and $c_1$ decreases by at least one. Therefore, the depth of the recursion is at most $2c$, and the "degree" of the recursion tree is at most $2^{O(c)}$ as guaranteed by the above lemma.

Let $(G, \mathcal{T})$ be an input. We now proceed to prove Lemma B.5, i.e., we show how to compute a $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut in $(G, \mathcal{T})$.

**Our algorithm:** Let $(A'_0, A'_1)$ be a minimum cut in $G$ such that $Q_0 \subseteq A'_0$ and $Q_1 \subseteq A'_1$ and each side contains at least one terminal. This cut can be found by a standard minimum $s$-$t$ cut algorithm. Observe that the value of this cut is at most $\ell$ if there is a valid constrained cut.

Such a cut can be used for our recursive approach to solve smaller subproblems by recursing on $G[A'_i]$ as follows. Denote by $\mathcal{T}_i = A'_i \cap \mathcal{T}$ for $i = 0, 1$. By definition, each set $\mathcal{T}_i$ is non-empty, and this is crucial for us.

If $|E_G(A'_0, A'_1)| > \ell$, the procedure terminates and reports no valid solution. Or, if $|\mathcal{T}_i| \geq k_i$ for all $i = 0, 1$, then we have found our desired constrained cut. Otherwise, assume that $|\mathcal{T}_0| < k_0$ (the other case is symmetric). We create a collection of $2^{O(c)}$ instances of smaller subproblems as follows.

**Sub-Instances.**

- First, we guess the "correct" way to partition terminals in $\mathcal{T}_0$ into $\mathcal{T}_0 = \mathcal{T}_0^0 \cup \mathcal{T}_0^1$. There are at most $2^c$ possible guesses.
- Second, we guess the "correct" partition of the (non-terminal) boundary vertices in $V(E_G(A'_0, A'_1)) - \mathcal{T}$ into $B_0 \cup B_1$ where $B_0$ and $B_1$ are the vertices supposed to be on the zero-side and one-side respectively. Let $\tilde{E} = E_G(B_0, B_1)$. There are $2^c$ possible guesses.

Now we will solve subproblems in $G[A'_0]$ and $G[A'_1]$. Notice that $G[A'_0]$ has small number of terminals, so we could solve it by brute force. For $G[A'_1]$, we will solve it recursively.

Let $E_0$ be the minimum cut in $G[A'_0]$ that separates $S_0 = Q_0 \cup (B_0 \cap A'_0) \cup \mathcal{T}_0^0$ and $T_0 = (B_1 \cap A'_0) \cup \mathcal{T}_0^1$. Next, we solve an instance of valid $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut in $G[A'_1]$ with terminal set $\mathcal{T}_1$, where $Q'_0 = (B_0 \cap A'_1)$, $Q'_1 = Q_1 \cup (B_1 \cap A'_1)$, $c'_0 = \max(c_0 - |\mathcal{T}_0^0|, 0)$, $c'_1 = \max(c_1 - |\mathcal{T}_0^1|, 0)$, and $\ell' = \ell - |\tilde{E}| - |E_0|$. Let $E_1$ be a $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut. Our algorithm outputs $E_0 \cup E_1 \cup \tilde{E}$.

**Analysis.** Clearly, $c'_0 + c'_1 < c_0 + c_1$. The following lemma will finish the proof.

**Lemma B.6** *There is a $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut in $(G, \mathcal{T})$ if and only if there exist correct guesses $(B_0, B_1, \mathcal{T}_0^0, \mathcal{T}_0^1)$ such that a $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut exists in $(G[A'_1], \mathcal{T}_1)$.*

**Proof:** First, we prove the "if" part. Suppose that there exists such a guess $(B_0, B_1, \mathcal{T}_0^0, \mathcal{T}_0^1)$. We claim that $E_0 \cup E_1 \cup \tilde{E}$ is actually a $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut that we are looking for. Observe that the size of the cut is at most $\ell$.

We argue that there are two subsets of terminals $\widetilde{\mathcal{T}_0}$ of size $c_0$ and $\widetilde{\mathcal{T}_1}$ of size $c_1$ that are separated after removing $E_0 \cup E_1 \cup \tilde{E}$. Let $\mathcal{T}_1^0$ and $\mathcal{T}_1^1$ be the sets of terminals in $\mathcal{T}_1$ that are on the side of $Q'_0$ and $Q'_1$, respectively (in particular, $\mathcal{T}_1^0$ cannot reach $Q'_1$ in $G[A'_1]$ after removing $E_1$). Notice that $|\mathcal{T}_0^0 \cup \mathcal{T}_1^0| \geq c_0$ and $|\mathcal{T}_0^1 \cup \mathcal{T}_1^1| \geq c_1$. The following claim completes the proof of the "if" part. □

**Claim B.7** *$Q_0 \cup \mathcal{T}_0^0 \cup \mathcal{T}_1^0$ and $Q_1 \cup \mathcal{T}_0^1 \cup \mathcal{T}_1^1$ are not connected in $G$ after removing $\tilde{E} \cup E_0 \cup E_1$.*

**Proof:** Let us consider a path $P$ from $Q_0$ to $Q_1$ in $G$; we view it such that the first vertex starts in $Q_0$ and so on until the last vertex on the path is in $Q_1$. Let $u$ be the last vertex the path from the start lies completely in $G[A'_0]$ and $v$ be the first vertex such that the path from $v$ to the end lies completely in $G[A'_1]$. Break path $P$ into $P_1 P_2 P_3$ where $P_1$ is the path from the first vertex to $u$, $P_2$ is

the path from $u$ to $v$, and $P_3$ the path from $v$ to the last vertex of $P$ in $Q_1$. If $|\{u,v\} \cap B_0| = 1$, then we are done because $P_2$ contains some edge in $\tilde{E}$. So, it must be that (i) $u, v \in B_0$ or (ii) $u, v \in B_1$. In case (i), we have $v \in Q'_0$ while the last vertex of $P$ is in $Q_1 \subseteq Q'_1$, so path $P_3$ is path in $G[A'_1]$ connecting $Q'_0$ to $Q'_1$. Hence, $P_3$ contains an edge in $E_1$. In case (ii), we have that $u \in T_0$, while the first vertex in $P$ is in $Q_0 \subseteq S_0$. Therefore, path $P_1$ is a path in $G[A'_0]$ connecting $S_0$ to $T_0$, which must be cut by $E_0$.

Similar analysis can be done when considering the path $P$ that connects $Q_0$ and $\mathcal{T}_1^1$, or between $\mathcal{T}_0^0$ and $Q_1 \cup \mathcal{T}_1^1$. The only (somewhat) different case is when the path $P$ connects $Q_0$ to $\mathcal{T}_0^1$. Assume that $P$ is not completely contained in $G[A'_0]$ (otherwise, it would be trivial). Let $u$ be the last vertex on $P$ such that the path from the start to $u$ lies completely inside $G[A'_0]$, and let $v$ be the first vertex on $P$ such that the path from $v$ to the end of $P$ lies completely inside $G[A'_0]$. Again, we break $P$ into three subpaths $P_1P_2P_3$ similarly to before. If $u \in B_1$, then we are done because $P_1$ would contain an edge in $E_0$; or, if $v \in B_0$, then we are also done since $P_3$ would contain an edge in $E_0$. Therefore, $u \in B_0$ and $v \in B_1$, implying that $P_2$ must contain an edge in $\tilde{E}$. □

To prove the "only if" part, assume that $(A_0, A_1)$ is a valid $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut. We argue that there is a choice of guess such that the subproblem also finds a valid $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut. We define $B_i = V(E_G(A_0, A_1)) \cap A_i$ for $i = 0, 1$, and $\mathcal{T}_0^i = \mathcal{T}_0 \cap A_i$ for $i = 0, 1$. With these choices, we have determined the values of $Q'_0$, $Q'_1$, $c'_0$ and $c'_1$. The following claim will finish the proof.

**Claim B.8** *There exists a cut $E_0$ that separates $S_0$ and $T_0$ in $G[A'_0]$ and a cut $E_1$ that is a $(Q'_0, Q'_1, c'_0, c'_1, \ell')$-constrained cut.*

**Proof:** First, we remark that $|E_G(A_0, A_1)| \leq \ell$ and

$$E_G(A_0, A_1) = E_G(B_0, B_1) \cup E_G(A'_0 \cap A_0, A'_0 \cap A_1) \cup E_G(A'_1 \cap A_0, A'_1 \cap A_1)$$

To complete the proof of the claim, it suffices to show that $E_G(A'_0 \cap A_0, A'_0 \cap A_1)$ is an $(S_0, T_0)$ cut in $G[A'_0]$ and that $E_G(A'_1 \cap A_0, A'_1 \cap A_1)$ is a valid constrained cut in $G[A'_1]$.

The first claim is simple: Since $S_0 \subseteq A_0$ and $T_0 \subseteq A_1$, any path from $S_0$ to $T_0$ in $G[A'_0]$ must contain an edge in $E_G(A'_0 \cap A_0, A'_0 \cap A_1)$.

The second claim is also simple: (i) $Q'_0 \subseteq A_0$ and $Q'_1 \subseteq A_1$, so the edge set $E_G(A'_1 \cap A_0, A'_1 \cap A_1)$ separates $Q'_0$ and $Q'_1$, (ii) For $i = 0, 1$, the number of terminals on the $Q'_i$-side must be at least $c_i - |\mathcal{T}_0^i|$ because otherwise this would contradict the fact that $(A_0, A_1)$ is a $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut. □

**Lemma B.9** *Let $c = \max\{\ell, c_0, c_1\}$. The algorithm to reduce the problem of finding a $(Q_0, Q_1, c_0, c_1, \ell)$-constrained cut with $\min c_0, c_1 > 0$ to the problem of finding a $(Q'_0, Q'_1, c'_0, c'_1, \ell)$-constrained cut with $\min c_1, c_0 = 0$ terminates in time $2^{O(c^2)} \cdot k^2 \cdot O(m)$.*

**Proof:** Lemma B.5 implies that the depth of the recursion tree is at the most $2c$ and that each recursive step reduces to solving $2^{O(c)}$ sub-instances. Hence, the total number of nodes in the recursion tree is $2^{O(c^2)}$.

The total runtime outside the recursive calls is dominated by a minimum $s-t$-cut computation. However, we observe that we are only interested in minimum cuts that are of value at the most $c$. Hence, such a cut can be found in time $O(mc)$ using any standard augmentation path based algorithm. Also, recall that we are looking for cuts that have at least one terminal on each side and hence we need to make $k^2$ guesses. The total runtime for this procedure is $k^2 \cdot O(mc)$ and we have the lemma. □

## B.2 Handling the base case

In this subsection, we prove Lemma B.3, i.e., we present an algorithm that finds a $(Q_0, Q_1, c_0, 0, \ell)$-constrained cut $(A'_0, A'_1)$. We first consider the case of $c_0 = 0$: since neither side of the cut is required to contain a terminal, we can simply compute a minimum-cut between $Q_0$ and $Q_1$. If one of these is empty (say $Q_1$), we take $A'_0 = V(G)$, $A'_1 = \emptyset$. In any case, let $E_1$ be the edges of the cut. Now there are two possibilities: if $|E_1| \leq \ell$, then our cut is a solution to the subproblem; if $|E_1| > \ell$, then there is no cut separating $Q_0$ from $Q_1$ with at most $\ell$ edges, and therefore, there is no valid constrained cut.

We can now focus on the case where $c_0 > 0$. We can further assume that $|\mathcal{T}| \geq c_0$; otherwise, there is no feasible solution. For simplification, we also assume that $Q_0$ is connected; if it is not, we can add auxiliary edges to make it connected in the run of the algorithm, which we can remove afterwards (these edges will never be cut since $Q_0 \subseteq A'_0$).

**Important Cuts.** The main tool we will be using is the notion of important cuts, introduced by Marx [Mar06] (see [CFK+15] and references within for other results using this concept).

**Definition B.10 (Important cut)** *Let $G$ be a graph and $X, Y \subseteq V(G)$ be disjoint subsets of vertices of $G$.*

*A cut $(S_X, S_Y)$, $X \subseteq S_X$, $Y \subseteq S_Y$ is an important cut if it has (inclusionwise) maximal reachability (from $X$) among all cuts with at most as many edges. In other words, there is no cut $(S'_X, S'_Y)$, $X \subseteq S'_X$, $Y \subseteq S'_Y$, such that $|E(S'_X, S'_Y)| \leq |E(S_X, S_Y)|$ and $S_X \subsetneq S'_X$.*

**Proposition B.11 ([CFK+15])** *Let $G$ be an undirected graph and $X, Y \subseteq V(G)$ two disjoint sets of vertices.*

*Let $(S_X, S_Y)$ be an $(X, Y)$-cut. Then there is an important $(X, Y)$-cut $(S'_X, S'_Y)$ (possibly $S_X = S'_X$) such that $S_X \subseteq S'_X$ and $|E(S'_X, S'_Y)| \leq |E(S_X, S_Y)|$.*

**Theorem B.12 ([CFK+15])** *Let $G$ be an undirected graph, $X, Y \subseteq V(G)$ be two disjoint sets of vertices and $c \geq 0$ be an integer. There are at most $4^c$ important $(X, Y)$-cuts of size at most $c$. Furthermore, the set of all important $(X, Y)$-cuts of size at most $c$ can be enumerated in time $O(4^c \cdot c \cdot m)$.*

**Proposition B.13** *Let $G$ be an undirected graph and $X, Y \subseteq V(G)$ two disjoint sets of vertices, and let $(S_X, S_Y)$ be an important $(X, Y)$-cut.*

*Then $(S_X, S_Y)$ is also an important $(X', Y)$-cut for all $X' \subseteq S_X$.*

**Proof:** Assume that the statement is false for contradiction. Then there is an important cut $(S'_X, S'_Y)$ for $(X', Y)$, with $|E(S'_X, S'_Y)| \leq |E(S_X, S_Y)|$ and $S_X \subsetneq S'_X$ by Proposition B.11. But then, $X \subseteq S_X \subseteq S'_X$, which means $(S'_X, S'_Y)$ is an $(X, Y)$-cut, and therefore, $(S_X, S_Y)$ is not an important cut for $(X, Y)$, which is a contradiction. □

**Cut profile vectors.** In order to make the exposition of the algorithm clearer, we introduce the concept of cut profile vectors.

**Definition B.14** *Let $c, \ell \geq 0$. A* cut profile vector *is a vector of $\lambda \leq c$ pairs of numbers $\{(\kappa_i, \ell_i)\}_{i \in [\lambda]}$, with $\kappa_i \in [c-1]$, $\ell_i \in [\ell]$, satisfying*

$$c \leq \sum_{i=1}^{\lambda} \kappa_i \leq 2c, \qquad \sum_{i=1}^{\lambda} \ell_i \leq \ell$$

*Each of the pairs $(\kappa_i, \ell_i)$ is called a* slot *of this profile. We say a cut $(A, B)$ is compatible with a slot $(\kappa_i, \ell_i)$ if $|A \cap \mathcal{T}| = \kappa_i$ and $|E(A, B)| = \ell_i$*

**Observation B.15** *There are at most $c^c \cdot \ell^c$ different cut profile vectors.*

Given a cut $(A, B)$, a cut profile vector represents the bounds for terminals covered and cut edges for each of the components of $G[A]$: there are $\lambda$ connected components, and component $C_i$ contains $\kappa_i$ terminals and has $\ell_i$ cut edges. Our algorithm will enumerate all the possible cut profile vectors and, for each of them, try to find a solution that fits the constraints given by the input. If there is a solution to the problem, there must be a corresponding profile vector, and therefore the algorithm finds a solution.

**Algorithm** Our algorithm works by guessing the number of connected components of $G[A]$ as well as the number of terminals that are contained in each component, and then proceeding to find cuts that fit these guesses. This is made easier by the following two facts: 1. there is a solution such that $A$ is a disjoint union of important $(Q_0, Q_1)$-cuts or $(t, Q_1)$-cuts, $t \in \mathcal{T}$; 2. we can find a set of $O(k^2)$ terminals such that there is a solution where each connected component of $G[A]$ contains one of these terminals.

The strategy of the algorithm is as follows: it starts by guessing the component $C_0$ that contains $Q_0$, out of all important $(Q_0, Q_1)$-cuts. If $(C_0, \bar{C}_0)$ is feasible, it returns. Otherwise, it guesses the cut profile vector of the solution. Then it tries to greedily fill all of the slots using important cuts containing disjoint sets of terminals. The goal of this stage is not yet to obtain a solution, but to accumulate terminals for the second stage. This process of trying to fill each slot is repeated $c$ times so that we may have $c$ candidates for each slot. All of the terminals contained in each of the candidates found this way form our base set of terminals, denoted $S$. The solution is finally obtained by enumerating tuples of up to $c$ components out of important $(t, Q_1)$-cuts, for $t \in S$.

We refer to Figure B.2 for a formal description of the algorithm.

We will now show that if there is a solution to the problem, our algorithm always finds a solution. This implies that, when we output "No Valid Solution", there is no solution. From now on, we assume that there is a solution to the problem. Let $(A, B)$ be a solution that minimizes the number of connected components of $G[A]$.

Let $\mathcal{C}_0$ be the set of all important cuts $(C, \bar{C})$ for $(Q_0, Q_1)$, and let $\mathcal{C}$ be the set of all important cuts $(C, \bar{C})$ for $(t, Q_1)$, for any $t \in \mathcal{T}$.

**Lemma B.16** *There is a solution $(A', B')$ such that every connected component $C$ of $G[A']$ corresponds to an important cut $(C, \bar{C})$ in $\mathcal{C}_0$ or $\mathcal{C}$. Furthermore, the number of connected components of $G[A']$ is not greater than that of $G[A]$.*

**Proof:** We will show an iterative process that turns a solution $(A, B)$ into a solution $(A', B)$ where every component corresponds to an important cut as above.

Let $C$ be a component of $G[A]$ that does not correspond to an important cut in $\mathcal{C}_0$ or $\mathcal{C}$. Notice that $C$ cannot contain a proper non-empty subset of $Q_0$ since $Q_0 \subseteq A$ and we assume that $Q_0$ is connected. If $C$ does not contain any terminals or $Q_0$, we move $C$ to $B$ (resulting in the cut $(A \setminus C, B \cup C)$). Since $C$ is a connected component of $G[A]$, all of the neighbors of $C$ are in $B$, and therefore moving $C$ to $B$ does not add any cut edges.

In the remaining case, $C$ contains a terminal $t \in \mathcal{T}$ or $Q_0$ but is not an important cut. By Proposition B.11, there is an important cut $(C', \bar{C}')$ with at most as many cut edges as $(C, \bar{C})$ and $C \subsetneq C'$. We can replace $C$ by a component corresponding to an important cut by taking the cut $(A \cup C', B \setminus C')$. This is still a valid solution since all terminals contained in $A$ are contained in $A \cup C'$, and $Q_0 \subseteq A$, $Q_1 \subseteq B \setminus C'$. Additionally, the number of edges crossing the cut does not increase: since $|E(C', \bar{C}')| \leq |E(C, \bar{C})|$, the number of edges added to the cutset is at most the number of edges removed.

We can apply the operations above until the constraints in the lemma are satisfied. Notice that when applying the operations above, the number of components of $G[A]$ never increases and the number of vertices in $A$ connected to terminals in $G[A]$ never decreases. Furthermore, each operation changes at least one of the two quantities above, so this process must finish after a finite number of operations. □

**Lemma B.17** *If a feasible cut $(A, B)$ exists, then our algorithm returns a feasible solution.*

**Proof:** Due to Lemma B.16, we can assume that every connected component of $G[A]$ corresponds to an important cut. Now, let $C_0^*, C_1^*, \ldots, C_\lambda^*$ be the connected components of $G[A]$, with $C_0^* \in \mathcal{C}_0$ being the component that contains $Q_0$. Let $\{(\kappa_i, \ell_i)\}_\lambda$ be the cut profile vector corresponding to the cuts $(C_i^*, \bar{C}_i^*)$ for $i \in \{1, \ldots, \lambda\}$ (excluding $C_0^*$), meaning that $\kappa_i$, $\ell_i$ are the number of terminals in $C_i^*$ and the number of edges in the cutset, $E(C_i^*, \bar{C}_i^*)$, respectively. Notice that, if $C_0^*$ or $C_0^* \cup C_i^*$ (for some $i \in [\lambda]$) contain at least $c$ terminals, then we can remove all the other components of $A$. In this case, the algorithm finds $C_0 \in \mathcal{C}_0$ or $C_0 \in \mathcal{C}_0$, $C_1 \in \mathcal{C}$ by enumeration and returns a valid solution. Otherwise, all the components contain at most $c - 1$ terminals each (and thus $A$ induces a slot vector as in Definition B.14).

Consider the iteration of the algorithm in which the cut profile vector defined above is considered and $C_0 = C_0^*$. The next part of the algorithm (Lines 22–29) greedily fills the slots with compatible important cuts from $\mathcal{C}$, while making sure that each set contains a disjoint set of terminals from the others. Though it seems that our goal at this stage is to obtain a feasible solution, what we intend is to obtain a set of terminals, denoted $S$, such that the set of important cuts for terminals in $S$ contains a feasible solution. For instance, if $S$ contains at least one terminal from each $C_i^*$, $i \in [\lambda]$, our goal is achieved.

The above considerations motivate the following definition. We say a slot $i$ is *hit* by $S$ if $S \cap C_i^* \neq \emptyset$. Notice that slot $i$ is hit by $S$ if $C_{ji} = C_i^*$ for some $j$, since the terminals in $C_i^*$ is added to $S$. Slot $i$ is also hit by $S$ if, for some $j$, we cannot find a set $C_{ji}$, since that implies that $C_{ji} = C_i^*$ is not a valid choice, and thus $S \cap C_i^* \neq \emptyset$. Furthermore, if slot $i$ is not hit by $S$, then $C_{ji}$ is found in all $(c + 1)$ rounds.

Let $\mathcal{C}_S \subseteq \mathcal{C}$ be the subset of important cuts containing terminals in $S$ (by Proposition B.13 these are the important $(t, Q_1)$-cuts for $t \in S$). It is now suficient to show that there is a sequence of $\lambda$ cuts $\{(C_i, \bar{C}_i)\}_{i\in\lambda}$ from $\mathcal{C}_S$, such that all $C_i$ contain disjoint sets of terminals (also disjoint with the terminals in $C_0$), and such that $(C_i, \bar{C}_i)$ is compatible with $(\kappa_i, \ell_i)$. Taking $C = C_0 \cup \bigcup_{i=1}^{\lambda} C_i$, we obtain a feasible solution $(C, \bar{C})$, which may be different from $(A, B)$, but has the same number of connected components as $G[A]$, and the same numbers of terminals contained in each component and cut edges separating each component from the other side of the cut. Since the algorithm enumerates all such sequences of $\lambda$ sets, it will find either $(C, \bar{C})$ or a different feasible solution.

We now define the sets $C_i$: if a slot $i$ is hit by $S$ we can set $C_i = C_i^*$, since there is $t \in C_i^* \cap S$, and therefore, $(C_i^*, \bar{C}_i^*) \in \mathcal{C}_S$. This cut is trivially compatible with $(\kappa_i, \ell_i)$, and is disjoint to all other sets defined similarly. Let $I_H$ be the set of all $i \in [\lambda]$ such that slot $i$ is hit by $S$, and let $C^* = \bigcup\{C_i : i \in I_H\}$. All that is left to prove is that, for every slot $i$ that is not hit by $S$, there is an important cut $(C_i, \bar{C}_i) \in \mathcal{C}_S$, which contains terminals not in any previous $C_{i'}$, $i' \leq i$, or in $C^*$. Notice that we have covered at most $c$ terminals so far (if we covered more, then the components so far are sufficient and therefore the number of components of $G[A]$ is not minimal). Since there are $c + 1$ important cuts $(C_{ji}, \bar{C}_{ji})$, $j \in [c + 1]$, all compatible with slot $i$ and containing disjoint sets of terminals (since the terminals of $C_{ji}$ are added to $S$ after being picked), there must be one set $C_{ji}$ that does not contain any of the at most $c$ terminals in $\bigcup_{i'<i} C_{i'}$, or in $C^*$, and we can set $C_i = C_{ji}$. Therefore, a sequence $\{(C_i, \bar{C}_i)\}_{i\in\lambda}$ exists, and the algorithm outputs a feasible solution. □

Finally, we analyze the running time.

**Lemma B.18** *The described algorithms terminates in time* $2^{O(c^2)} \cdot k + 2^{O(c)} \cdot k \cdot m$.

**Proof:** Computing all the relevant important cuts takes time $2^{O(c)} \cdot k \cdot m$. There are at most $c^{O(c)}$ cut profile vectors, and for each of these the algorithm fills the slots at most $c$ times, which takes time $c^2 \cdot k \cdot 2^{O(c)}$; then, once it has computed $S$, it enumerates at most $c$ components out of $2^{O(c)} \cdot c^2$ possible important cuts, which takes time $\left(2^{O(c)} \cdot c^2\right)^c = 2^{O(c^2)}$. Once the right combination of components is found, it takes $O(n)$ time to obtain the corresponding feasible cut. The total running time is

$$2^{O(c)} \cdot k \cdot m + c^{O(c)}\left(c^2 \cdot k \cdot 2^{O(c)} + 2^{O(c^2)}\right) = 2^{O(c^2)} \cdot k + 2^{O(c)} \cdot k \cdot m$$

□

1: **function** Constrained-Cut($G$, $\mathcal{T}$, $Q_0$, $Q_1$,$c$, $\ell$)
2: **if** $c = 0$ **then**
3: Compute a min-cut $(A'_0, A'_1)$ such that $Q_0 \subseteq A'_0, Q_1 \subseteq A'_1$
4: **if** $|E(A'_0, A'_1)| \leq \ell$ **then**
5: **return** $(A'_0, A'_1)$
6: **else**
7: **return** NO VALID SOLUTION
8: **end if**
9: **end if**

10: Compute the set $\mathcal{C}_0$ of all important $(Q_0, Q_1)$-cuts with at most $\ell$ cut edges
11: Compute the set $\mathcal{C}$ of all important $(t, Q_1)$-cuts for $t \in \mathcal{T}$ with at most $\ell$ cut edges

12: Find an important cut $(C_0, \bar{C}_0) \in \mathcal{C}_0$ such that $|C_0 \cap \mathcal{T}| \geq c$
13: **if** $(C_0, \bar{C}_0)$ exists **then**
14: **return** $(C_0, \bar{C}_0)$
15: **end if**
16: Find important cuts $(C_0, \bar{C}_0) \in \mathcal{C}_0$, $(C_1, \bar{C}_1) \in \mathcal{C}$, such that $(C_0 \cap \mathcal{T}) \cap (C_1 \cap \mathcal{T}) = \emptyset$, $|C_0 \cap \mathcal{T}| + |C_1 \cap \mathcal{T}| \geq c$, and $\left|E(C_0 \cup C_1, \bar{C}_0 \cap \bar{C}_1)\right| \leq \ell$
17: **if** $(C_0, \bar{C}_0)$, $(C_1, \bar{C}_1)$ exist **then**
18: **return** $(C_0 \cup C_1, \bar{C}_0 \cap \bar{C}_1)$
19: **end if**

20: **for** all cut profile vectors $\{(\kappa_i, \ell_i)\}_\lambda$, and all $(C_0, \bar{C}_0) \in \mathcal{C}_0$ **do**
21: $S \leftarrow C_0 \cap \mathcal{T}$
22: **for** $j \in \{1, \ldots, c+1\}$ **do** *// Round j*
23: **for** $i \in \{1, \ldots, \lambda\}$ **do**
24: Find $(C_{ji}, \bar{C}_{ji}) \in \mathcal{C}$ compatible with slot $(\kappa_i, \ell_i)$, such that $C_{ji} \cap S = \emptyset$
25: **if** $C_{ji}$ exists **then**
26: $S \leftarrow S \cup (C_{ji} \cap \mathcal{T})$
27: **end if**
28: **end for**
29: **end for**

30: Let $\mathcal{C}_S = \left\{(C, \bar{C}) \mid C \cap S \neq \emptyset\right\}$
31: Find (by enumeration) $\left\{(C_i, \bar{C}_i)\right\}_{i \in \lambda}$, with $(C_i, \bar{C}_i) \in \mathcal{C}_S$ compatible with slot $i$, and all sets $C_i \cap \mathcal{T}$ are disjoint (including $C_0 \cap \mathcal{T}$)
32: **if** $\left\{(C_i, \bar{C}_i)\right\}_{i \in \lambda}$ exists **then**
33: Let $C = C_0 \cup \bigcup_{i=1}^{\lambda} C_i$
34: **return** $(C, \bar{C})$
35: **end if**
36: **end for**

37: **return** NO VALID SOLUTION *// No solution found for any cut profile*
38: **end function**

Figure 7: Algorithm to find a constrained cut in the base case