Vertex Sparsification for Edge Connectivity

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

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.

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 [7, 65], 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 [34, 54, 48]. 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 $O(\log k/\log\log k)$ [54, 48, 10, 52, 21] and lower bound of $\Omega(\sqrt{\log k}/\log\log k)$ [52] are known. In a different direction, quality 1-sparsifiers (known as *mimicking networks*) with 2^{2^k} vertices were shown to exist [34, 42], and a lower bound of $2^{\Omega(k)}$ is also known [44]. Also, Chuzhoy [13] 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 $\widetilde{O}(k/\mathrm{poly}(\epsilon))$ exists; so far, this is only known for special graph classes, such as quasi-bipartite graphs [5, 4].

This paper aims to study a related graph sparsifier that is suitable for applications in designing fast algorithms for *connectivity problems*. In particular, we consider the

*Aalto University, Finland parinya.chalermsook@aalto.fi
†Indraprastha Institute of Information Technology Delhi, India syamantak@iiitd.ac.in
‡KAIST & IBS Discrete Mathematics Group, 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 at MPI Informatik and while visiting Aalto University.

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 $\min(c, \text{mincut}_G(S, \mathcal{T}\backslash S)) = \min(c, \text{mincut}_H(S, \mathcal{T}\backslash S))$, 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 $c = o(\log n)$ [38]. Additionally, our results are motivated in part by elimination-based graph algorithms [46, 47], 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) involves extending the recursive approach of [13] using *well-linked sets* to a thresholded setting. The construction of [13] 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. [13] then bounds the deterioration in quality from these contractions. The main differences between our approach and [13] 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 [43] to bottom out the recursion.

Additionally, in order to obtain near-linear running times for our constructions, we combine the *expander decomposition* technique [62] with several other combinatorial results that allow us to build the desired connectivity-c mimicking network.

We would like to note that the result in [43] already gives 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 nearly constant $\widetilde{O}(1)$[2] amortized update time. For online fully dynamic algorithms, such results are only known for $c \leq 3$ [36, 37]. Even in the simpler *offline* model introduced by Eppstein [22], 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 [55], 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

[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.

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

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 terminals, 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 [9] 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. Also, 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 [27], near-linear time algorithms for computing global min-cut [39], and most recently, progress in exact [50, 51, 14] and approximate max-flow algorithms [64, 58, 41, 63]. The c-limited edge connectivity case can be solved in $O(mc)$ time statically, and is also implied by ϵ-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.

The question of computing max-flow between multiple pairs of terminal vertices dates back to the Gomory-Hu tree [28], 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 min-cut separating terminals [15], as well as the construction of c-limited Gomory-Hu trees [35, 8].

Such results are in turn used to compute max-flow between multiple pairs of vertices [1]. These problems have received much attention in fine grained complexity, since their directed versions are difficult [3, 2], 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, its construction time is critical for their use in data structures [59]. For a moderate number of terminals (e.g., $k = n^{0.1}$), nearly-linear time constructions of vertex sparsifiers with $\operatorname{poly}(k)$ vertices were previously known only when $c \leq 5$ [59, 55]. To our knowledge, the only results for maintaining exact c connectivity for $c \geq 4$ are incremental algorithms [16, 17, 18, 32], in addition to the aforementioned fully dynamic algorithm for $c = 4, 5$ by [55], 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 $c = o(\log n)$ [38].

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 [46, 47], 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 [46], and can be more easily adapted to data structures that handle dynamic graphs [20, 19].

Important properties of such routines are 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. Particularly, guaranteeing $(1+\epsilon)$-quality 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 [46, 47]. In contrast, most of the work on vertex sparsification to date has been on shortest path metrics [66, 12], and/or utilizes algebraic techniques [43, 23]. 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 $O(\log k/\log\log k)$ [10, 52] and $\Omega(\sqrt{\log k}/\log\log k)$ [52] respectively. [13] 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 [52, 10, 21, 45, 6, 23, 33, 29]. Recently, vertex sparsifiers were also shown to be closely connected with dynamic graph data structures [30, 59, 31, 19].

There has also been work on mimicking networks on special graph classes. [44] presented a mimicking network of size $O(k^2 2^{2k})$ for planar graphs with k terminals, nearly matching the lower bound [40]. When all terminals lie on the same face, mimicking networks of size $O(k^2)$ are known [29, 45]. Graphs with bounded-treewidth have an upper bound $O(k\cdot 2^{2^{\text{tw}(G)}})$ [11].

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 involve contractions, naturally leading to multigraphs. Thus, 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 ∂X, is $E_G(X,V\backslash X)$. For subsets $A,B\subseteq V$, let $\text{mincut}_G(A,B)$ be the size of the minimum cut separating A,B in G. If $A\cap B\neq\emptyset$, then $\text{mincut}_G(A,B)=\infty$. Formally, we have

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

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 $A\cup B\stackrel{\text{def}}{=}A\cup B$ 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 $\mathcal{T}_1$ of $\mathcal{T}$, we have*

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

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 useful observations about $(\mathcal{T},c)$-equivalency.

LEMMA 2.1. *If G and H are $(\mathcal{T},c)$-equivalent, then for any subset of terminals $\widehat{\mathcal{T}}\subseteq\mathcal{T}$ and any $\widehat{c}\le c$, G and H are also $(\widehat{\mathcal{T}},\widehat{c})$-equivalent.*

LEMMA 2.2. *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 2.1). We complement this partitioning process by showing that sparsifiers on disconnected graphs can be built separately (Lemma 2.3).

COROLLARY 2.1. *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 $G \backslash \widehat{E}$, then $H \cup \widehat{E}$ is $(\mathcal{T}, c)$-equivalent to G.*

LEMMA 2.3. *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 [56]. For completeness, we prove the following lemma in the full version of the paper[3].

LEMMA 2.4. *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 2.5. *For any subset of vertices V_1 and V_2 in V, and any set of edges $\widehat{E}$, it holds that*

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

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 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 [13, 60]. Here, we introduce a thresholded version of well-linkedness.[4]

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 $|E_G(A, B)| \geq \min(|\partial A \cap \partial X|, |\partial B \cap \partial X|, c)$.*

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 4.1 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 $V \backslash \mathcal{T} = X_1 \uplus X_2 \uplus \cdots \uplus X_p$, 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 $|\partial X| = |\partial(V \backslash \mathcal{T})| \leq kc$, 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 $|\partial(X_i)| \leq 2c-1$. More precisely, if $|\partial(X_i)| \geq 2c$ 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 $|\partial(X_i)| \leq 2c - 1$ we make tricky manipulation on the boundary edges $\partial(X_i)$ as in Lemma 4.9 and then work on the line graph of X_i. Applying a kernelization result (see Lemma 4.8), which develops from matroid theory (gammoid in particular) and the representative sets lemma (see Theorem 4.3), to the line graph gives rise to a fruitful result (see Lemma

[3] https://arxiv.org/abs/2007.07862

[4] 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 [61]

4.3) 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 in the partition is at most 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 this recursion 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 efficient running time is unlikely to exist, as this seems like a non-trivial instance of the non-uniform sparsest cut problem (in the full version, 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 preserves $(\mathcal{T}, c)$-equivalency. 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 4.1, are governed by their conductance φ, 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 φ, [62] 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 φ (see Lemma 4.4). We now mark all the removed edges as essential, delete them, and mark their endpoints as additional terminals. Corollary 2.1 and Lemma 2.3 show that it suffices to work separately with each remaining component, which are guaranteed to be expanders with conductance at least φ. 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 φ (see Lemma 4.5). 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 2.5) and a cut in G/e is also a cut in G (see Lemma 4.6), 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 parti-

tion via a potential function. This approach 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 partitions X into at most kc pieces, 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 a smaller equivalent graph 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, $G/(E\setminus E')$, 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. The proof of this claim is available in the appendix of our full version.

LEMMA 4.1. *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.*

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 $(\mathcal{T}, c)$-equivalence is justified by Lemma 4.1. Also, Corollary 2.1 and Lemma 2.3 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 $\mathcal{T}'$.

Algorithm 1: PolySizedcNetwork(G)

Input: undirected unweighted multi-graph G
Output: connectivity-c mimicking network H
if $|\partial G| \leq 2c - 1$ **then**
  return $(\mathcal{T}', c)$*-equivalent sparsifier with terminals $\mathcal{T}'$, where $\mathcal{T}'$ is the set of vertices incident to boundary edges ∂G (based on Lemma 4.3).*
else
  if *violating cut (V_1, V_2) exists* **then**
    return PolySizedcNetwork$(G[V_1])$ & PolySizedcNetwork$(G[V_2])$.
  else
    Contract G to a single vertex.
return *connectivity-c mimicking network H*

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 4.2. *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.*

Thus, branching takes place at most $|\partial X|$ times, as 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), by Lemma 4.9, 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 gives rise to the following lemma with its proof presented in Section 4.3.

LEMMA 4.3. *Let $G = (V, E)$ be a graph with a set $\mathcal{T}$ of $O(c)$ terminals and each terminal has degree 1. There is a subset E' of E with $|E'| = O(c^3)$ and $G/(E\setminus E')$ 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 4.2 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 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 $|\bigcup_{i=1}^{p}(\partial X_i)| \leq O(kc^2)$, 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. Another contribution is from the partition pieces X_i with $|\partial(X_i)| \leq 2c-1$. Thus, the total number of edges is at most $kc \cdot O(c^3) = O(kc^4)$ by applying Lemma 4.3.

To verify that the returned graph is indeed a connectivity-c mimicking network, it suffices to apply Lemma 4.1 to argue that we can contract well-linked pieces. Then we can use Corollary 2.1 to delete all boundary edges in $\bigcup_{i=1}^{p}\partial X_i$, and then use Lemma 2.3 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 2.1, 2.2, and Corollary 2.1), and connected components may be handled separately (Lemma 2.3). 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 4.2. *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 4.2, we utilize insights from recent results on finding c-vertex cuts [57, 25, 26], 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 4.1. *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 4.4. *(Theorem 1.2 of [62]) There exists an algorithm* ExpanderDecompose *that for any undirected unweighted graph G and any parameter φ, decomposes in $O(m\varphi^{-1}\log^4 n)$ time G into pieces $\{G_i\}$ of conductance at least φ so that at most $O(m\varphi\log^3 n)$ edges are between the pieces.*

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

$$\min\{vol(S), vol(V\backslash S)\} \leq c\varphi^{-1}. \tag{4.1}$$

In a graph with expansion φ, we can enumerate all cuts of size at most c in time exponential in c and φ. 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 4.5. *In a graph G with n vertices and conductance φ, there exists an algorithm that enumerates all cuts of size at most c with connected smaller side in time $O(n(c\varphi^{-1})^{2c})$.*

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

LEMMA 4.6. *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 (φ-Sparsify) We need a subroutine used to sparsify a graph with conductance φ and terminals $\mathcal{T}$. This subroutine named as φ-Sparsify takes such a graph and enumerates all cuts of size at most c by a smaller side through Lemma 4.5. 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 φ-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 4.5), φ-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 partitions of the terminals $\mathcal{T}$ induced by the enumerated cuts,
2. C is the set of a minimum cut separating a partition of terminals in P,
3. E_0 is the set of edges 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 4.6). 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 2.5). 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 φ-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.

Algorithm 2: EfficientPolySized$(G, \mathcal{T})$

Input: undirected unweighted multi-graph G, terminals $\mathcal{T}$
Output: connectivity-c mimicking network G'
Set $\varphi^{-1} = 4C'c^4 \log^3 n$ for some C'
do
  $\{G_i\} =$ ExpanderDecompose(G, φ).
  Remove the inter-cluster edges among $\{G_i\}$ and add the endpoints of the edges as new terminals to each piece with conductance φ.
  $G'_i = \varphi$-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.
  $G' = \bigcup_i G'_i$ together with inter-cluster edges.
while $|E(G')| < |E(G)|$
return G'

In this way, scanning through each edge in sequence, φ-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 4.7. *For a graph G with conductance φ, n vertices, m edges, and k terminals, the algorithm φ-Sparsify returns a connectivity-c mimicking network with $O(kc^4)$ edges in $O(m + nc(c\varphi^{-1})^{2c})$ time.*

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 Algorithm 2 and with analysis as follows, where C' is a constant such that in Lemma 4.4 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.

Proof of Theorem 1.1 Part 1. By Corollary 2.1 and Lemma 4.7, 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.

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, the total number of edges in G' is bounded by

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

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

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

The running time is dominated by φ-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 4.3: 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. We will make use of the following result in essence.

LEMMA 4.8. ([43]) *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 this lemma addresses a 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 $\widehat{G}$.

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 whenever $\mathcal{S}$ has such a set satisfying the same condition. The previous studies [49, 53, 24] have been eager to find a representative set in polynomial time, which is also of independent interest, for a given representation matrix.

THEOREM 4.3. ([24]) *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 $O(\binom{p+q}{p}tp^\omega + t\binom{p+q}{p}^{\omega-1})$ operations over $\mathbb{F}$, where $\omega < 2.373$ is the matrix multiplication exponent.*

The Subroutine We identify and contract nonessential edges via the result from vertex sparsification preserving vertex connectivity. To exploit this fruitful result, 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 4.8, 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 4.1.

OBSERVATION 4.1. *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.*

Recall in POLYSIZEDCNETWORK that when H has at most $2c - 1$ boundary edges, we mark the endpoints in $V(H)$ of the boundary edges ∂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 4.1, we can assume not only $|\widehat{\mathcal{T}}| = O(c)$, but also each terminal has degree 1.

LEMMA 4.9. *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 ∂H (i.e., $V(H) \cap V(\partial H)$), we may assume that each terminal in $\widehat{\mathcal{T}}$ has degree 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 4.9; the piece has $O(c)$ tentative terminals with degree 1.

Proof of Lemma 4.3. Let $v(e)$ be the corresponding vertex in the line graph $L(G)$ for edges e in E and v_t 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 $L(G)'$ the enlarged one.

Viewing $\mathcal{T}'$ as X in Lemma 4.8, 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: replace terminals t' in Z (if any) by the unique neighbor $v(tv_t)$ of the 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 $G/(E \setminus E')$ 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 $E \setminus E'$, 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 still be found by only contracting edges, and thus our algorithm in Section 4.2 gives sparsifiers with $O(kc^4)$ 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 it for speedup in Section 5.2.2 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 [55] 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 $\mathcal{T} = \mathcal{T}_1 \uplus \mathcal{T}_2$ 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 5.1. *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*

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

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

We can also state Corollary 2.1 and Lemma 2.3 in the language of edge containment.

LEMMA 5.2. *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 $G \setminus \widehat{E}$, then $E^{contain} \cup \widehat{E}$ contains all $(\mathcal{T}, c)$-cuts in G.*

LEMMA 5.3. *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 5.2. *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 $\mathcal{T} = \mathcal{T}_1 \uplus \mathcal{T}_2$ with $\text{mincut}_G(\mathcal{T}_1, \mathcal{T}_2) \leq c$, there exists a cut $F = E(V_1, V_2)$ such that:*

1. *F has size $\text{mincut}_G(\mathcal{T}_1, \mathcal{T}_2)$,*
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 $G \setminus E^{intersect}$.*

Reduction to Cut Intersection Based on Definition 5.2, 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 $G \setminus E^{intersect}$.

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 Algorithm 3. All arguments until now can be summarized as the following stitching lemma.

Algorithm 3: GETCONTAININGEDGES$(G, \mathcal{T}, c)$: Find a set of edges containing all the $(\mathcal{T}, c)$-cuts.

Input: undirected unweighted multi-graph G, terminals $\mathcal{T}$, and connectivity threshold c
Output: set of edges $E^{contain}$ containing all $(\mathcal{T}, c)$-cuts
Initialize $E^{contain} \leftarrow \emptyset$
For $\widehat{c} \leftarrow c, \ldots, 1$ in decreasing order:
(1) $E^{contain} \leftarrow E^{contain} \cup$ GETINTERSECTINGEDGES$(G, \mathcal{T}, \widehat{c})$
(2) $G \leftarrow G \setminus E^{contain}$
(3) $\mathcal{T} \leftarrow \mathcal{T} \cup V(E^{contain})$, where $V(E^{contain})$ is the endpoints of all edges in $E^{contain}$.
return $E^{contain}$

LEMMA 5.4. *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 $(V, E \setminus E^{intersect})$. 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 5.4.

THEOREM 5.1. *For any parameter φ, 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 5.1 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 5.1 Part 2. In essence, we use Theorem 5.1 Part 2 to prove Theorem 1.1 Part 2. 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 5.2). 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 5.5. *Let F be a cut given by the partition $V = V_1 \cup V_2$ in G such that both $G[V_1]$ and $G[V_2]$ are connected, and $\mathcal{T}_i = V_i \cap \mathcal{T}$, $i = 1, 2$, 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.*

Algorithm 4: RECURSIVENONTRIVIALCUTS$(G, \mathcal{T}, c)$: Find a set of edges intersecting all terminal cuts. (RNC for short)

Input: undirected unweighted multi-graph G, terminals $\mathcal{T}$, and connectivity threshold $c > 0$
Output: set of edges $E^{intersect}$ that intersects all $\mathcal{T}$ separating cuts of size at most c

If $|\mathcal{T}| \leq 4$, **return** the union of the min-cuts of all 8 partitions of the terminals.
Initialize $E^{intersect} \leftarrow \emptyset$.
if $\exists$ *non-trivial $\mathcal{T}$-separating cut* (V_1, V_2) *of size* $\leq c$ **then**
    $E^{intersect} \leftarrow E^{intersect} \cup E(V_1, V_2) \cup \text{RNC}(G/V_2, (\mathcal{T} \cap V_1) \cup \{v_2\}, c) \cup \text{RNC}(G/V_1, (\mathcal{T} \cap V_2) \cup \{v_1\}, c)$, where v_1 and v_2 are the vertices formed upon contracting of V_1 and V_2 respectively.
else
    For all vertex v such that $|\text{mincut}(G, v, \mathcal{T} \backslash v)| \leq c$, do $E^{intersect} \leftarrow E^{intersect} \cup \text{mincut}(G, v, \mathcal{T} \backslash v)$ (i.e., add all local terminal cuts to $E^{intersect}$)
return $E^{intersect}$

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 5.3. *A non-trivial $(\mathcal{T}, c)$-cut is a separation of V into $V_1 \cup V_2$ such that $G[V_1]$ and $G[V_2]$ are both connected, and $|V_1 \cap \mathcal{T}| \geq 2$ and $|V_2 \cap \mathcal{T}| \geq 2$.*

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 5.5, 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. In fact, we can 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 5.4. *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 5.6. *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:*

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

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

Combining Lemma 5.5 and 5.6, we obtain the recursive Algorithm 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, then Line 3 finds it and recurses on both sides of the cut using Lemma 5.5. Otherwise, by Lemma 5.6, 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 5.

Lemma 5.7. RecursiveNontrivialCuts *correctly returns a set of $(\mathcal{T}, c)$-cut-intersecting edges of size at most $O(|\mathcal{T}| \cdot c)$.*

We may modify Algorithm 4 to take extra steps for marginal speedup by utilizing expander decomposition.

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. This section illustrates a more efficient algorithm that returns larger sparsifiers, but ultimately leads to the faster running time in Theorem 5.1 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 5.6.

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. It takes $O(m + nc^3 \log n)$ time by using an algorithm in [15].

Theorem 5.2. ([15]) *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.

Lemma 5.8. *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.*

Algorithm 5: RecursiveTerminalCuts$(G, \mathcal{T}, c)$: Find a set of edges that intersect all terminal cuts. (RTC for short)

Input: undirected unweighted multi-graph G, terminals $\mathcal{T}$, and connectivity threshold $c > 0$
Output: set of edges $E^{intersect}$ that intersects all $\mathcal{T}$ separating cuts of size at most c

1 Use Lemma 2.4 to reduce G to having at most nc edges
2 Initialize $E^{intersect} \leftarrow \emptyset$.
3 **while** $|\mathcal{T}| > 4$ **do**
4 Compute a minimum terminal cut $(V_1 \uplus V_2)$ of G
5 If $|E(V_1, V_2)| > c$, **return** $E^{intersect}$
6 **if** *each of V_1 and V_2 contains at least 2 terminals* **then**
7 **return** $E^{intersect} \cup \text{RTC}(G/V_2, (\mathcal{T} \cap V_1) \cup \{v_2\}, c) \cup \text{RTC}(G/V_1, (\mathcal{T} \cap V_2) \cup \{v_1\}, c)$, where v_1 and v_2 are the vertices formed upon contracting of V_1 and V_2 respectively.
8 **else**
9 Assume V_1 contains one terminal s.
10 Record $x \leftarrow |E(V_1, V_2)|$
11 **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, as (V_1, V_2) is a minimum $\mathcal{T}$-separating cut)* **do**
12 Let $(\widehat{V}_1, \widehat{V}_2)$ be such a cut and record $F \leftarrow E(\widehat{V}_1, \widehat{V}_2)$
13 $G \leftarrow G/\widehat{V}_1/F$.
14 $E^{intersect} \leftarrow E^{intersect} \cup F$.
15 For 8 partitions of the terminals, if the corresponding mincut has size at most c, add it to $E^{intersect}$
16 **return** $E^{intersect}$

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 5.4 is also a minimum terminal cut. We can cope with it via an extension of Lemma 5.5. 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 5.9. *Let F be a cut given by the partition $V = V_1 \uplus V_2$ in G such that both $G[V_1]$ and $G[V_2]$ are connected, and $\mathcal{T}_i = V_i \cap \mathcal{T}$, $i = 1, 2$, 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.*

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 5.9 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 5.9 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 Algorithm 5, and the procedure for the repeated contractions to find a maximal s-isolating cut described in the above paragraph is in Line 9.

Discussion of Algorithm 5. We clarify some lines in Algorithm 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 7, and does not execute any of the later lines in the algorithm. In Line 11, 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 13, 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 5.10. *For any graph G, terminals $\mathcal{T}$, and a value c, the algorithm* RECURSIVETERMINALCUTS *runs in $O(n^2c^4 \log n)$ time and returns a set at most $O(|\mathcal{T}|c^2)$ edges that intersect all $(\mathcal{T}, c)$-cuts.*

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 5.11. *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.*

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 ν in time linear in c and ν.

THEOREM 5.3. (THEOREM 3.1 OF [25]) *Let G be a graph and let $v \in V(G)$ be a vertex. For a connectivity parameter c and volume parameter ν, 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 ν.*
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 5.5. (LOCAL CUTS) *For a vertex $v \in G$, define* LocalCut(v) *to be*

$$\min_{\substack{V=V_1 \uplus V_2 \\ v \in V_1 \\ vol(V_1) \le vol(V_2)}} |E(V_1, V_2)|.$$

We now combine Theorem 5.3 with the observation from Equation 4.1 in order to control the volume of the smaller side of the cut in an expander.

LEMMA 5.12. *Let G be a graph with conductance at most φ, and let $\mathcal{T}$ be a set of terminals. If $|\mathcal{T}| \ge 500c^2\varphi^{-1}$ then for any vertex $s \in \mathcal{T}$ we can with high probability in $\widetilde{O}(c^3\varphi^{-1})$ time either compute* LocalCut(s) *or certify that* LocalCut$(s) > c$.

We can substitute this faster cut-finding procedure into RECURSIVETERMINALCUTS to get the faster running time stated in Theorem 5.1 Part 2.

Proof of Theorem 5.1 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 RECURSIVETERMINALCUTS (Algorithm 5). Let $\widehat{\mathcal{T}}$ be the set of terminals at the top level of recursion. The recursion creates $O(|\widehat{\mathcal{T}}|)$ distinct terminals. First, we terminate if $|\mathcal{T}| \le 500c^2\varphi^{-1}$ and use the result of Lemma 5.11. Otherwise, instead of using Theorem 5.2 for line 3, we compute the terminal $s \in \mathcal{T}$ with minimal value of 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 7. Otherwise, we perform the loop in Line 11.

We now give implementation details for computing the terminal $s \in \mathcal{T}$ with minimal value of LocalCut(s).

By Lemma 2.5 we can see that for a terminal s, 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$(s) > c$.

For the runtime, we first bound it from the cases $|\mathcal{T}| \leq 500c^2\varphi^{-1}$. The total number of vertices and edges in the leaves of the recursion tree is $O(mc)$. Therefore, by Lemma 5.11, 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 11 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 5.12. 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 5.1 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 Here 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 $\widetilde{O}(f(c)(g(c)+c)q)$.*

It 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 Algorithm 6 which shows Lemma 6.1. In Algorithm 6 graph G_i for $0 \leq i \leq q$ denotes the current graph after queries $Q_1, \cdots, Q_i$ have been applied.

Algorithm 6: OFFLINECONNECTIVITY(G, c, Q, ℓ, r): OC for short

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_ℓ is a connectivity query between vertices a and b, answers if a and b have connectivity c

1 **if** $\ell = r$ *&* Q_ℓ*: c-connectivity query between a and b* **then**
2 Run a maxflow up to c units from a to b in G and **return** the result
3 **else**
4 Define $m = \lfloor \frac{\ell+r}{2} \rfloor$
5 $E^{\ell,r}_{left} \leftarrow$ the edges that are in all of $G_\ell, \cdots, G_m$ but not in all of $G_\ell, \cdots, G_r$.
6 $\mathcal{T}^{\ell,r}_{left} \leftarrow$ all vertices involved in queries $Q_\ell, \cdots, Q_m$
7 $E^{\ell,r}_{right} \leftarrow$ the edges that are in all of $G_{m+1}, \cdots, G_r$ but not in all of $G_\ell, \cdots, G_r$.
8 $\mathcal{T}^{\ell,r}_{right} \leftarrow$ all vertices involved in queries $Q_{m+1}, \cdots, Q_r$
9 OC$(A(G \cup E^{\ell,r}_{left}, \mathcal{T}^{\ell,r}_{left}, c), c, Q, \ell, m)$
10 OC$(A(G \cup E^{\ell,r}_{right}, \mathcal{T}^{\ell,r}_{right}, c), c, Q, m+1, r)$

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 5. The algorithm then computes all vertices involved in queries in the children of a recursion node in line 6, and computes a connectivity-c mimicking network treating those vertices as terminals, and recurses.

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 6.1. *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 $\text{mincut}^c_G(A_i, B_i)$ 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 [9] 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 only single-exponential in c^4 and $\mathrm{tw}(G)$ (whereas the result in [9] is double-exponential in both c and $\mathrm{tw}(G)$).

Let (T, X) be a tree decomposition of G (see Section 6.2.1). 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.

THEOREM 6.1. *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 $n \exp\left(O(c^4 \mathrm{tw}(G) \log(\mathrm{tw}(G)c)\right)$.*

We outline its proof in the rest of the section, introducing concepts and assumptions used in our result in Section 6.2.1. Then 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 For an undirected graph G, 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 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 $\mathrm{tw}(G)$, is the minimum *width* of any tree decomposition (T, X) for G. The width of (T, X) is given by $\max |X_t| - 1$.

We 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 [9]): (i) T has height $O(\log n)$; (ii) $|X_t| \leq O(\mathrm{tw}(G))$ 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 6.2 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 6.2.

LEMMA 6.2. *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.,*

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

6.2.2 Local Connectivity Rules

We introduce the local connectivity rules which allow us to assign states in a consistent manner to the nodes of T. The states we 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 [9], 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 overcount 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 6.3 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 6.1. (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*

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

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

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

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 6.3. *Let* G *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 6.1. Then, the pairs* $\{(\mathcal{H}'_t, \mathcal{H}_t)\}_{t \in T}$ *satisfy the local definitions if and only if for every* $t \in V(\mathcal{T})$*,*

$$\mathcal{H}'_t \equiv^c_{X_t} G_t \quad \text{and} \quad \mathcal{H}_t \equiv^c_{X_t} G \setminus E(G_t).$$

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:

1. For any leaf $t \in T$, set $D[t, \emptyset, \mathcal{H}] = 0$, and $D[t, \mathcal{H}', \mathcal{H}] = +\infty$ for $\mathcal{H}' \neq \emptyset$;
2. For the root node $\text{root}(T)$, set $D[\text{root}(T), \mathcal{H}', \mathcal{H}] = +\infty$ if $\mathcal{H} \neq \emptyset$;
3. For any demand (v_i, d_i), and $t \in T$ with $v_i \in X_t$, 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:

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

Using these definitions, we can show the feasibility of the dynamic program (i.e. the entries $D[\text{root}(T), \mathcal{H}', \emptyset]$ correspond to feasible solutions) and its optimality (we always obtain the optimum solution to the problem). The full proof is available in the full version.

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

[1] A. Abboud, R. Krauthgamer, and O. Trabelsi, *New algorithms and lower bounds for all-pairs max-flow in undirected graphs*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2020, pp. 48–61.

[2] A. Abboud and V. V. Williams, *Popular conjectures imply strong lower bounds for dynamic problems*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2014, pp. 434–443.

[3] A. Abboud, V. V. Williams, and H. Yu, *Matching triangles and basing hardness on an extremely popular conjecture*, in ACM Symposium on Theory of Computing (STOC), 2015, pp. 41–50.

[4] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng, *On fully dynamic graph sparsifiers*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2016, pp. 335–344.

[5] A. Andoni, A. Gupta, and R. Krauthgamer, *Towards (1+ ε)-approximate flow sparsifiers*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, pp. 279–293.

[6] S. Assadi, S. Khanna, Y. Li, and V. Tannen, *Dynamic sketching for graph optimization problems with applications to cut-preserving sketches*, in Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2015.

[7] A. A. Benczúr and D. R. Karger, *Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time*, in ACM Symposium on Theory of Computing (STOC), 1996, pp. 47–55.

[8] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, *Fast edge splitting and edmonds' arborescence construction for unweighted graphs*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2008, pp. 455–464.

[9] P. Chalermsook, S. Das, G. Even, B. Laekhanukit, and D. Vaz, *Survivable network design for group connectivity in low-treewidth graphs*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM), 2018, pp. 8:1–8:19.

[10] M. Charikar, T. Leighton, S. Li, and A. Moitra, *Vertex sparsifiers and abstract rounding algorithms*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2010, pp. 265–274.

[11] S. Chaudhuri, K. Subrahmanyam, F. Wagner, and C. D. Zaroliagis, *Computing mimicking networks*, Algorithmica, 26 (2000), pp. 31–49.

[12] S. Chechik, *Near-optimal approximate decremental all pairs shortest paths*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2018, pp. 170–181.

[13] J. Chuzhoy, *On vertex sparsifiers with steiner nodes*, in Symposium on Theory of Computing Conference (STOC), 2012, pp. 673–688.

[14] M. B. Cohen, A. Madry, P. Sankowski, and A. Vladu, *Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log W)$ time*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2017, pp. 752–771.

[15] R. Cole and R. Hariharan, *A fast algorithm for computing steiner edge connectivity*, in ACM Symposium on Theory of Computing (STOC), 2003, pp. 167–176.

[16] Y. Dinitz and A. Vainshtein, *The connectivity carcass of a vertex subset in a graph and its incremental maintenance*, in ACM Symposium on Theory of Computing (STOC), 1994, pp. 716–725.

[17] Y. Dinitz and A. Vainshtein, *Locally orientable graphs, cell structures, and a new algorithm for the incremental maintenance of connectivity carcasses*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 302–311.

[18] Y. Dinitz and J. R. Westbrook, *Maintaining the classes of 4-edge-connectivity in a graph on-line*, Algorithmica, 20 (1998), pp. 242–276.

[19] D. Durfee, Y. Gao, G. Goranci, and R. Peng, *Fully dynamic spectral vertex sparsifiers and applications*, in ACM Symposium on Theory of Computing (STOC), 2019, pp. 914–925.

[20] D. Durfee, R. Kyng, J. Peebles, A. B. Rao, and S. Sachdeva, *Sampling random spanning trees faster than matrix multiplication*, in ACM Symposium on Theory of Computing (STOC), 2017, pp. 730–742.

[21] M. Englert, A. Gupta, R. Krauthgamer, H. Racke, I. Talgam-Cohen, and K. Talwar, *Vertex sparsifiers: New results from old techniques*, SIAM Journal on Computing, 43 (2014), pp. 1239–1262.

[22] D. Eppstein, *Offline algorithms for dynamic minimum spanning tree problems*, Journal of Algorithms, 17 (1994), pp. 237–250.

[23] S. Fafianie, E. C. Hols, S. Kratsch, and V. A. Quyen, *Preprocessing under uncertainty: Matroid intersection*, in International Symposium on Mathematical Foundations of Computer Science (MFCS), 2016, pp. 35:1–35:14.

[24] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh, *Efficient computation of representative families with applications in parameterized and exact algorithms*, Journal of the ACM (JACM), 63 (2016), pp. 1–60.

[25] S. Forster, D. Nanongkai, L. Yang, T. Saranurak, and S. Yingchareonthawornchai, *Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2020, pp. 2046–2065.

[26] S. Forster and L. Yang, *A faster local algorithm for detecting bounded-size cuts with applications to higher-connectivity problems*, CoRR, abs/1904.08382 (2019).

[27] A. V. Goldberg and R. E. Tarjan, *Efficient maximum flow algorithms*, Communications of the ACM, 57 (2014), pp. 82–89.

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

[29] G. Goranci, M. Henzinger, and P. Peng, *Improved guarantees for vertex sparsification in planar graphs*, in European Symposium on Algorithms (ESA), 2017, pp. 44:1–44:14.

[30] ———, *The power of vertex sparsifiers in dynamic graph algorithms*, in European Symposium on Algorithms (ESA), 2017, pp. 45:1–45:14.

[31] ———, *Dynamic effective resistances and approximate schur complement on separable graphs*, in European Symposium on Algorithms (ESA), 2018, pp. 40:1–40:15.

[32] G. Goranci, M. Henzinger, and M. Thorup, *Incremental exact min-cut in poly-logarithmic amortized update time*, in Leibniz International Proceedings in Informatics (LIPIcs), vol. 57, 2016.

[33] G. Goranci and H. Räcke, *Vertex sparsification in trees*, in Approximation and Online Algorithms (WAOA), 2016, pp. 103–115.

[34] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde, *Characterizing multiterminal flow networks and computing flows in networks of small treewidth*, Journal of Computer and System Sciences, 57 (1998), pp. 366–375.

[35] R. Hariharan, T. Kavitha, and D. Panigrahi, *Efficient algorithms for computing all low st edge connectivities and related problems*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 127–136.

[36] M. R. Henzinger and V. King, *Randomized fully dynamic graph algorithms with polylogarithmic time per operation*, Journal of ACM (JACM), 46 (1999), pp. 502–516.

[37] J. Holm, K. de Lichtenberg, and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity*, Journal of ACM (JACM), 48 (2001), pp. 723–760.

[38] W. Jin and X. Sun, *Fully dynamic c-edge connectivity in subpolynomial time*, CoRR, abs/2004.07650 (2020).

[39] D. R. Karger, *Minimum cuts in near-linear time*, Journal of ACM (JACM), 47 (2000), pp. 46–76.

[40] N. Karpov, M. Pilipczuk, and A. Zych-Pawlewicz, *An exponential lower bound for cut sparsifiers in planar graphs*, Algorithmica, (2018), pp. 1–14.

[41] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, *An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, pp. 217–226.

[42] A. Khan and P. Raghavendra, *On mimicking networks representing minimum terminal cuts*, Information Processing Letters, 114 (2014), pp. 365–371.

[43] S. Kratsch and M. Wahlstrom, *Representative sets and irrelevant vertices: New tools for kernelization*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2012, pp. 450–459.

[44] R. Krauthgamer and H. Rika, *Mimicking networks and succinct representations of terminal cuts*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2013, pp. 1789–1799.

[45] ———, *Refined vertex sparsifiers of planar graphs*, SIAM Journal on Discrete Mathematics, 34 (2020), pp. 101–129.

[46] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, *Sparsified cholesky and multigrid solvers for connection laplacians*, in ACM Symposium on Theory of Computing (STOC), 2016, pp. 842–850.

[47] R. Kyng and S. Sachdeva, *Approximate gaussian elimination for laplacians - fast, sparse, and simple*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2016, pp. 573–582.

[48] F. T. Leighton and A. Moitra, *Extensions and limits to vertex sparsification*, in ACM Symposium on Theory of Computing (STOC), 2010, pp. 47–56.

[49] L. Lovász, *Flats in matroids and geometric graphs*, in Combinatorial Surveys, 1977, pp. 45–86.

[50] A. Madry, *Navigating central path with electrical flows: From flows to matchings, and back*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2013, pp. 253–262.

[51] A. Madry, *Computing maximum flow with augmenting electrical flows*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2016, pp. 593–602.

[52] K. Makarychev and Y. Makarychev, *Metric extension operators, vertex sparsifiers and lipschitz extendability*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2010, pp. 255–264.

[53] D. Marx, *A parameterized view on matroid optimization problems*, Theoretical Computer Science, 410 (2009), pp. 4471–4479.

[54] A. Moitra, *Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2009, pp. 3–12.

[55] A. Molina and B. Sandlund, *Historical optimization with applications to dynamic higher edge connectivity*, (2018).

[56] H. Nagamochi and T. Ibaraki, *A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7 (1992), pp. 583–596.

[57] D. Nanongkai, T. Saranurak, and S. Yingchareonthawornchai, *Breaking quadratic time for small vertex connectivity and an approximation scheme*, in ACM Symposium on Theory of Computing (STOC), 2019, pp. 241–252.

[58] R. Peng, *Approximate undirected maximum flows in $O(m\text{poly}\log(n))$ time*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2016, pp. 1862–1867.

[59] R. Peng, B. Sandlund, and D. D. Sleator, *Optimal offline dynamic 2, 3-edge/vertex connectivity*, in Algorithms and Data Structures (WADS), 2019, pp. 553–565.

[60] H. Räcke, C. Shah, and H. Täubig, *Computing cut-based hierarchical decompositions in almost linear time*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, pp. 227–238.

[61] B. A. Reed, *Tree width and tangles: A new connectivity measure and some applications*, Surveys in Combinatorics, (1997), pp. 87–162.

[62] T. Saranurak and D. Wang, *Expander decomposition and pruning: Faster, stronger, and simpler*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 2019, pp. 2616–2635.

[63] J. Sherman, *Nearly maximum flows in nearly linear time*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2013, pp. 263–269.

[64] ———, *Area-convexity, l_∞ regularization, and undirected multicommodity flow*, in ACM Symposium on Theory of Computing (STOC), 2017, pp. 452–460.

[65] D. A. Spielman and S. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in ACM Symposium on Theory of Computing (STOC), 2004, pp. 81–90.

[66] J. van den Brand and T. Saranurak, *Sensitive distance and reachability oracles for large batch updates*, in IEEE Symposium on Foundations of Computer Science (FOCS), 2019, pp. 424–435.