Saranurak, Thatchaphol; Yingchareonthawornchai, Sorrachai

Deterministic Small Vertex Connectivity in Almost Linear Time

Published in:
Proceedings of 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)

DOI:
10.1109/FOCS54457.2022.00080

Published: 30/10/2022

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Saranurak, T., & Yingchareonthawornchai, S. (2022). Deterministic Small Vertex Connectivity in Almost Linear Time. In *Proceedings of 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)* (pp. 789-800). (Annual Symposium on Foundations of Computer Science). IEEE. https://doi.org/10.1109/FOCS54457.2022.00080

Deterministic Small Vertex Connectivity in Almost Linear Time

Thatchaphol Saranurak* Sorrachai Yingchareonthawornchai†

Abstract

In the *vertex connectivity* problem, given an undirected n-vertex m-edge graph G, we need to compute the minimum number of vertices that can disconnect G after removing them. This problem is one of the most well-studied graph problems. From 2019, a new line of work [Nanongkai *et al.* STOC'19;SODA'20;STOC'21] has used randomized techniques to break the quadratic-time barrier and, very recently, culminated in an almost-linear time algorithm via the recently announced maxflow algorithm by Chen *et al.* In contrast, all known deterministic algorithms are much slower. The fastest algorithm [Gabow FOCS'00] takes $O(m(n + \min\{c^{5/2}, cn^{3/4}\}))$ time where c is the vertex connectivity. It remains open whether there exists a subquadratic-time deterministic algorithm for any constant $c > 3$.

In this paper, we give the first deterministic almost-linear time vertex connectivity algorithm for all constants c. Our running time is $m^{1+o(1)}2^{O(c^2)}$ time, which is almost-linear for all $c = o(\sqrt{\log n})$. This is the first deterministic algorithm that breaks the $O(n^2)$-time bound on sparse graphs where $m = O(n)$, which is known for more than 50 years ago [Kleitman'69].

Towards our result, we give a new reduction framework to *vertex expanders* which in turn exploits our new almost-linear time construction of mimicking network for vertex connectivity. The previous construction by Kratsch and Wahlström [FOCS'12] requires large polynomial time and is randomized.

*thsa@umich.edu. University of Michigan, USA

†sorrachai.yingchareonthawornchai@aalto.fi. Aalto University, Finland

Contents

1 Introduction

Vertex connectivity is one of most fundamental measures for robustness of graphs. For any n-vertex m-edge graph $G = (V, E)$, the vertex connectivity κ_G of G is the minimum number of vertices that can disconnect G after removing them. Fast algorithms for computing vertex connectivity has been extensively studied [Kle69, Pod73, ET75b, Eve75, Gal80, EH84, Mat87, BDD+82, LLW88, CR94, NI92, CT91, Hen97, HRG00, Gab06, CGK14]. (See [NSY19] for a discussion.) Since 2019, a new line of work [NSY19, FNS+20, LNP+21] has used randomized techniques to break the long-standing quadratic-time barrier and, very recently, culminated in an almost-linear time algorithm via the recently announced maxflow algorithm by [CKL+22]. However, these new algorithms are all Monte-Carlo and, thus, can make errors.

In contrast, all known deterministic algorithms are much slower. Improved upon previous deterministic algorithms [ET75a, HRG00], in 2000, Gabow [Gab06] gave the previously fastest deterministic algorithm with $O(m(n + \min\{c^{5/2}, cn^{3/4}\}))$ time where c is the vertex connectivity. While linear-time algorithms are known when $c \leq 3$ [Tar72, HT73], it remains open whether there exists a subquadratic-time deterministic algorithm for any constant $c > 3$.

In this paper, we give the first deterministic almost-linear time algorithm for computing vertex connectivity for all constants c. To state our result precisely, we need some terminology. We say (L, S, R) is a *vertex cut* if L, S, R partition a vertex set V but there is no edge between L and R. The size of a vertex cut (L, S, R) is $|S|$. Note that a vertex cut of size less than c certifies that $\kappa_G < c$. If there is no such cut, then $\kappa_G \geq c$, and we say that G is c-connected. Our main result is as follows:

Theorem 1.1. *There is a deterministic algorithm that, given an undirected graph G, in $m^{1+o(1)}2^{O(c^2)}$ time either outputs a vertex cut of size less than c or concludes that G is c-connected.*

When focusing on sparse graphs where $m = O(n)$, the $O(n^2)$-time bound was shown for more than 50 years ago [Kle69]. Theorem 1.1 is the first that deterministically breaks this barrier.

Related Works. Significant effort has been devoted to devising deterministic algorithms that is as fast as their randomized counterparts. Examples include the line of work on deterministic minimum edge cut algorithm [KT15, HRW17, Sar21, LP20] where, recently, Li [Li21] showed a deterministic almost-linear time algorithm for minimum edge cuts even in weighted graphs. Another example are deterministic expander decomposition algorithms and their applications to deterministic Laplacian solvers and approximate max flow [CGL+20].

Historical Note. The previous version of this paper [GLN+19] contains two results: deterministic subquadratic-time algorithms for balanced cuts and vertex connectivity. The paper was split into two newer papers. The newer version of the balanced cut algorithms is [CGL+20], which gave the first almost-linear time deterministic balanced cut algorithm. Our paper is the new version of the vertex connectivity algorithms from [GLN+19].

1.1 Our Approach

Towards our main result (Theorem 1.1), we develop two main algorithmic components. Below, we explain how their combination immediately implies Theorem 1.1. Suppose our goal is just to check whether G is c-connected for $c = O(1)$.

"Almost" Reduction to Vertex Expanders. Our starting point is the observation that c-connectivity can be checked fast on *vertex expanders*. For any vertex cut (L,S,R) of G, its *expansion* is $h(L,S,R) = \frac{|S|}{|S|+\min\{|L|,|R|\}}$. When $h(L,S,R)$ is small, we say that the cut is *sparse*. We say G is a ϕ-vertex expander if $\min_{(L,S,R)} h(L,S,R) \geq \phi$ and we simply call it a vertex expander when $\phi \geq 1/n^{o(1)}$. Observe that for any vertex cut (L,S,R) of size less than c in a vertex expander must be very unbalanced, i.e., $\min\{|L|,|R|\} \leq cn^{o(1)}$. Suppose we are given a vertex $x \in L$ where $|L| \leq cn^{o(1)}$, the local flow algorithms can find the cut of size less than c in $\text{poly}(cn^{o(1)})$ time [NSY19] or in $c^{O(c)}n^{o(1)}$ time [CHI+17]. Thus, by simply calling the local flow algorithm from every vertex, one can check c-connectivity of a vertex expander in almost-linear total time. However, in general the graph might contain a very sparse cut. This is exactly the hard instance of all previous deterministic algorithms.

Now, our new framework essentially allows us to assume that the input graph is a vertex expander. More precisely, give any graph $G = (V,E)$, our subroutine ExpandersOrTerminal(G,c,ϕ) computes in $m^{1+o(1)}/\phi$ time a collection $\mathcal{G}$ of ϕ-vertex expanders and a terminal set $T \subset V$ with the following guarantee:

1. G is c-connected iff every $H \in \mathcal{G}$ is c-connected and the *Steiner connectivity* of T is at least c (i.e. there is no vertex cut of size less than c separating any terminal pair $x,y \in T$).
2. $\mathcal{G}$ has almost-linear total size, i.e., $\sum_{H\in\mathcal{G}} |V(H)| = n^{1+o(1)}$.
3. $|T| \leq \phi n^{1+o(1)}$.

See details in Definition 3.2 and Theorem 3.3. We will set $\phi = 1/n^{o(1)}$ so that $|T| \ll n$ is small.

We employ the reduction as follows. Given G, we call ExpandersOrTerminal(G,c,ϕ) and check c-connectivity of all vertex expanders $H \in \mathcal{G}$ in almost-linear total time, since their total size is almost-linear. If any $H \in \mathcal{G}$ is not c-connected, then we are done. Otherwise, we still need to check the Steiner connectivity of T. Therefore, in almost-linear time, the framework allows us to "focus" on a small terminal set T and this is where our second algorithmic component can help.

Mimicking Network for Vertex Connectivity. Given any $G = (V,E)$ and terminal set T, our second subroutine VertexSparsify(G,T,c) computes a small graph H of size proportional to T and H preserves *all* minimum vertex cuts between terminals T up to size c. More precisely, let $\mu_G(A,B)$ be the minimum number of vertices that, after removing from G, there is no path from A to B left (we allow removing vertices in A and B). The guarantees of H is that, for any $A,B \subseteq T$, $\min\{\mu_G(A,B),c\} = \min\{\mu_H(A,B),c\}$. We also require that $\kappa_H \geq \min\{\kappa_G,c\}$ (otherwise, there might be a new minimum vertex cut in H). We call H a *c-vertex connectivity mimicking network* or, for short a *(T,c)-sparsifier* for G. Below, we show that VertexSparsify can be implemented in almost-linear time. See Definition 3.4 and Theorem 3.5 for details.

Theorem 1.2. *Our subroutine* VertexSparsify(G,T,c) *computes a (T,c)-sparsifier H for G of size $|E(H)| = |T|2^{O(c^2)}$ in $m^{1+o(1)}2^{O(c^2)}$ time.*

How do we exploit this sparsifier? Let T be the terminal set from ExpandersOrTerminal and $H =$ VertexSparsify(G,T,c).[1] Since H preserves all minimum vertex cuts between T and also does not introduce any new minimum cut, it suffices to check if H is c-connected. By setting $\phi = 1/2^{O(c^2)}n^{o(1)}$, we have $|T| \leq n/2^{O(c^2)}$ and so $|E(H)| \leq n/2$. Thus, we have reduced the vertex

[1] For a technical reason, we actually need to compute $H =$ VertexSparsify(G,T',c) where $T' = \cup_{v\in T} N^c_G(v)$ and $N^c_G(v)$ is an arbitrary set of c neighbors of v. See Section 3 for details.

connectivity problem to a graph of half the size in almost-linear time. By repeating this framework at most $O(\log n)$ rounds, we are done. That completes the explanation how the high-level components fit together.

1.2 New Mimicking Networks

The new mimicking network in Theorem 1.2 can be of independent interest.

Let us compare Theorem 1.2 with related results on mimicking network literature. Kratsch and Wahlström [KW20] showed an algorithm that computes a (T, c)-sparsifier (without the condition that $\kappa_H \geq \min\{\kappa_G, c\}$) of size $O(|T|^3)$ in some large polynomial time and their algorithm is randomized. In our applications it is crucial that the size is linear in $|T|$ and the algorithm is deterministic. When we consider edge cuts instead of vertex cuts, c-edge-connectivity mimicking networks of size $|T|\text{poly}(c)$ can be computed in $m^{1+o(1)}c^{O(c)}$ time [CDK+21] or $n^{O(1)}$ time [Liu20].

Theorem 1.2 can be viewed as an adaptation of c-edge connectivity mimicking networks from [LPS19, CDK+21] to c-vertex connectivity, but there are many technical obstacles we need to overcome. Basically, this is because not all techniques for edge cuts generalize to vertex cuts. Even when they do generalize, they require careful and complicated definitions and arguments in order to carry out the approach. For concrete examples, we observe that the "cut enumeration" approach of [CDK+21] fails completely for vertex cuts.[2] So, instead, we take the approach based on covering sets and intersecting sets of [LPS19]. To order to generalize this approach, we arrive at a complicated and delicate definition of *reducing-or-covering* partition-set pair (See Definition 6.1). Nonetheless, while trying to overcome these technical obstacles in constructing mimicking networks, we believe that there is one technical lesson that might be useful beyond this work.

Fast Closure Oracles via Hypergraphs. In our algorithm, we need to perform an operation that is analogous to contracting an edge e, but we do it on a vertex v. This operation is called *neighborhood closure*, or just *closure*, for short. The closure has been use as an important primitive in [KW20]. Given a graph G with a vertex v, *closing* v in G is to add a clique between all neighbors of v, and remove v from G. This operation clearly takes a lot of time when v has large degree. We observe that if we "lift" the problem to hypergraphs, it turns out that the equivalent operation is as follows: *closing* v *on a hypergraph* H is to merge all hyperedges e containing v into one hyperedge, i.e. insert $e' = \cup_{e \ni v} e$ and remove all e that contains v. This simple equivalence is formalized in Observation 7.26.

Now, it turns out that the closure operation on hypergraph is very easy to (implicitly) implement, for example, by using the union-find data structure. More specifically, our algorithm in Section 7 will need to implement a data structure on a graph that handle closure operations in an online manner. The key technique that enables our algorithm to run in almost-linear time is to "lifting" the graph into a hypergraph and implementing closure operations on the hypergraph.

1.3 Organization

We describe necessary background and terminologies in Section 2. We describe the full vertex connectivity algorithm in Section 3. Then, we explain in Section 4 how to implement the reduction to vertex expanders, i.e. the subroutine $\textsc{ExpandersOrTerminal}(G, c, \phi)$. For the rest of the

[2] In [CDK+21], they only show a fast algorithm for enumerating cuts $(S, V \setminus S)$ whose one side induced a connected graph (i.e. $G[S]$ is connected), and then argue that this kind of cuts suffice for their construction. This is not true for vertex cuts. We would need to list vertex cuts (L, S, R) where $G[L]$ is not connected too. While listing cut where $G[L \cup S]$ is connected suffices, it is not clear how to do it fast.

paper, we explain how to compute c-vertex connectivity mimicking network, i.e. the subroutine $\textsc{VertexSparsify}(G, T, c)$. In Section 5, we explain a reduction from constructing mimicking networks to an object called *covering set*. In Section 6, we show a reduction from constructing covering set to another object called *reducing-or-covering partition-set pair*. We give fast implementations for computing a reducing-or-covering partition-set pair in Section 7. Finally, we discuss open problems in Section 8.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph. Throughout this paper, we consider only undirected graphs (unless stated otherwise). For $x, y \in V$, we say that a path in G is (x, y)-*path* if it starts with x and ends with y. For $S, T \subseteq V$, we say that a path P is an (S, T)-*path* in G if it is an (s, t)-path for some $s \in S, t \in T$. We also denote $E_G(S, T)$ to be the set of edges whose one endpoint is in S and the other endpoint is in T. For any $x \in V$, we denote $N_G(x)$ to be the set of neighbors of x in graph G. For any $S \subseteq V$, we denote $N_G(S)$ to be the set of neighbors of some vertex in S that is not in S. We also denote $N_G[S] = S \cup N_G(S)$ and $\mathrm{vol}_G(S) = \sum_{x \in S} \deg_G(x)$ where $\deg_G(x)$ is the degree of x in G, which is the number edges incident to x. For any $X \subseteq V$, we denote $G[X]$ as subgraph of G induced by X. For set notations, we denote set difference as $S - T$. We denote $G - S$ to be the graph G after removing all vertices in S. We also denote $V(G)$ and $E(G)$ to be the set of vertices in G and the set of edges in G, respectively.

Let $S \subseteq V$ and $A, B \subseteq V$. We say that S is a *separator* in G if $G - S$ is not connected. We say that S is an (x, y)-separator in G for some $x, y \in V$ if $G - S$ does not have an (x, y)-path, and $x, y \notin S$. We say that S is an (X, Y)-*separator* in G for some disjoint sets $X, Y \subseteq V$ if $G - S$ does not have an (X, Y)-path and $X \cap S = Y \cap S = \emptyset$. We denote $\kappa_G(x, y)$ to be the minimum size of (x, y)-separator in G or $n - 1$ if such a separator does not exist. *Vertex connectivity* of G is $\kappa_G = \min_{x, y \in V} \kappa_G(x, y)$. We also say that S is an (A, B)-*weak separator* if $G - S$ does not have an (A, B)-path (but possibly $A \cap S \neq \emptyset$ and $B \cap S \neq \emptyset$). We denote $\mu_G(A, B)$ to be the minimum size of (A, B)-weak separator in G. Note that $\mu_G(A, B) \leq \min\{|A|, |B|\}$ because A and B are trivial (A, B)-weak separators. For the purpose of vertex connectivity, we can assume WLOG that G is simple (i.e., no-self loops, and no parallel edges).

A *vertex cut* in $G = (V, E)$ is a triple (L, S, R) that forms a partition of V such that $L \neq \emptyset, R \neq \emptyset$ and $E_G(L, R) = \emptyset$. A *vertex expansion* of (L, S, R) is $h_G(L, S, R) = \frac{|S|}{\min\{|L \cup S|, |S \cup R|\}}$. A *vertex expansion* of a graph G is $h(G) = \min_{(L,S,R)} h_G(L, S, R)$. Let $\phi \in (0, 1)$. We say that a vertex cut (L, S, R) is ϕ-*sparse* if $h_G(L, S, R) < \phi$. We say that G is a ϕ-*vertex expander* if G does not have a ϕ-sparse vertex cut, i.e., $h(G) \geq \phi$.

Definition 2.1. A (c, ν)-*LocalVC algorithm* takes as inputs an initial vertex x in a graph $G = (V, E)$, and two parameters ν, c such that $c \cdot \nu \leq |E|/\tau$ (for some constant $\tau \geq 1$) and outputs either:

- $\perp$ certifying that there is no vertex cut (L, S, R) such that $x \in L, \mathrm{vol}_G(L) \leq \nu$ and $|S| < c$, or
- a vertex set L such that $|N_G(L)| < c$.

Theorem 2.2 ([NSY19, CHI+17]). *There are deterministic (c, ν)-LocalVC algorithms that takes $O(\nu c^{O(c)})$ time and $O(\nu^2 c)$ time, respectively.*

Theorem 2.3 ([NI92]). *Given a graph $G = (V, E_G)$ with n vertices and m edges and a parameter $c > 0$, there is a linear-time algorithm that outputs another graph $H = (V, E_H)$ satisfying the following properties:*

- *G is c-connected if and only if H is c-connected. Furthermore, for all $S \subseteq V$ such that $|S| < c$, S is a separator in G if and only if S is a separator in H, and*
- *H has arboricity c. In particular, $|E_H| \le nc$ and for all $S \subseteq V$, $|E_H(S,S)| \le c|S|$.*

In the language of vertex sparsifier, the following terms will be use throughout the paper.

Definition 2.4. Let G and H be graphs that contain the same terminal set T and $c > 0$ be an integer. We say that G and H are (T, c)-*equivalent* if for all pair $A, B \subseteq T$, we have $\min\{\mu_H(A,B), c\} = \min\{\mu_G(A,B), c\}$.

Definition 2.5. Let H and G be graphs and $c > 0$ be an integer.

- H is *cut-recoverable* for G if $V(H) \subseteq V(G)$ and every separator in H is a separator in G,
- H is *c-cut-recoverable* for G if $V(H) \subseteq V(G)$ and every separator of size $< c$ in H is a separator in G, and
- H is *c-mincut-recoverable* for G if $V(H) \subseteq V(G)$ and for all $s, t \in V(H)$ every min (s,t)-separator of size $< c$ in H is an (s,t)-separator in G.

Proposition 2.6. *If H_1 is cut-recoverable for G, then $\kappa_{H_1} \ge \kappa_G$. If H_2 is c-cut-recoverable for G or c-mincut-recoverable, then $\kappa_{H_2} \ge \min\{c, \kappa_G\}$.*

Proof. If S^* be a min separator in H_1, then S^* is a separator in G, and thus $\kappa_{H_1} = |S^*| \ge \kappa_G$. Now consider H_2. If $\kappa_{H_2} \ge c$, then $\kappa_{H_2} \ge \min\{c, \kappa_G\}$, and we are done. Now, assume $\kappa_{H_2} < c$. Let S^* be a min separator in H. Since H_2 is c-cut-recoverable (or c-mincut-recoverable) and $|S^*| < c$, S^* is a separator in G, and thus $c > \kappa_{H_2} = |S^*| \ge \kappa_G$. Therefore, $\kappa_{H_2} \ge \kappa_G = \min\{c, \kappa_G\}$. $\square$

Observe that cut-recoverable property is transitive: If A is cut-recoverable for B and B is cut-recoverable for C, then A is cut-recoverable for C. Using Definition 2.4 and Definition 2.5, the algorithm in Theorem 2.3 outputs a graph H such that (1) H and G are (V, c)-equivalent, (2) H is c-cut recoverable for G, and (3) H has arboricity c.

3 Vertex Connectivity Algorithm

In this section, we prove Theorem 1.1. We first consider a warm up problem when the input graph is already a ϕ-expander, and then discuss our key tools to handle general graphs.

Warm up. Suppose that G is ϕ-vertex expander. For intuitive purpose, we can think of $\phi = 1/\log n$ (we use $\phi = 1/n^{o(1)}$ in the real algorithm). We show that we can decide c-connectivity in $\tilde{O}(n\phi^{-2})$ time. Since $h(G) \ge \phi$, any vertex cut (L, S, R) where $|S| < c$ must be such that $|L| \le c\phi^{-1}$ or $|R| \le c\phi^{-1}$. Therefore, G is c-connected if and only if there is a set $L \subseteq V$ where $|L| \le c\phi^{-1}$ and $|N(L)| < c$ where $N(L)$ is the neighbors of L. Observe that $\mathrm{vol}(L) \le |L|^2 + |L|c = O(c^2\phi^{-2})$. This is exactly where the *local vertex connectivity* (LocalVC) algorithm introduced in [NSY19] can help us. This algorithm works as follows: given a vertex x in a graph G and parameters ν and c, either (1) certifies that there is no set $L \ni x$ where $\mathrm{vol}(L) \le \nu$ and $|N(L)| < c$, or (2) returns a set L where $|N(L)| < c$. See Definition 2.1 for a formal definition. There are currently two deterministic algorithms for this problem: a $\tilde{O}(\nu^2 c)$-time algorithm [NSY19] and a $O(\nu c^{O(c)})$-time algorithm by a slight adaptation of the algorithm by Chechik et al. [CHI+17]. As $c = O(1)$, we will use the $O(\nu c^{O(c)})$-time algorithm here. From the above observation about the set L, it is enough to run the

LocalVC algorithm from every vertex x with a parameter $\nu = O(c^{-2}\phi^{-2})$ to decide if such L exists. This takes $O(n\phi^{-2})$ total time to decide c-connectivity of G if G is ϕ-expander. When $\phi^{-1} = \log n$, we have a $\tilde{O}(n)$ time algorithm for deciding c-connectivity of ϕ-vertex expanders. That is, we have the following.

Proposition 3.1. *Given an ϕ-vertex expander $G = (V, E)$, and parameters $c > 0$ and $\phi \in (0, 1)$, there is an $\tilde{O}(|V|\frac{c^{O(c)}}{\phi^2})$-time algorithm that outputs either a min separator of size $< c$ or certifies that G is c-connected.*

General Graphs. In general, G is not necessarily a ϕ-vertex expander. To handle the general case, we compute a (c, ϕ)-*expanders-or-terminal pair* of G.

Definition 3.2. Given a graph $G = (V, E)$ and c, ϕ as parameters, and let $\mathcal{G}$ be a set of graphs, $T \subseteq V$ be a set of vertices in G, we say that a pair $(\mathcal{G}, T)$ is a (c, ϕ)-*expanders-or-terminal pair* for G if every $H \in \mathcal{G}$ is a ϕ-vertex expander and c-mincut-recoverable for G (Definition 2.5) and the pair can characterize c-connectivity of G: G is c-connected if and only if

1. every graph $H \in \mathcal{G}$ is c-connected, and
2. for all $x, y \in T$, $\kappa_G(x, y) \geq c$.

We also say that T is a *terminal* set of G.

Our technical tool is that we can compute in almost-linear time a *small* (c, ϕ)-expanders-or-terminal pair in that the size of the terminal set T is small (e.g., $\leq n\phi$) and at the same time the total size of ϕ-vertex-expanders in $\mathcal{G}$ is almost linear. In Section 4, we prove the following:

Theorem 3.3 (Fast Algorithm for Expanders-or-Terminal Pair)**.** *Given a graph $G = (V, E)$ where $m = |E|, n = |V|$ and c, ϕ as inputs where $\phi < 1/(2c\log^2 n)$ and min-degree of $G \geq c$, there is an $O(m^{1+o(1)}/\phi)$ time algorithm, denoted as* ExpandersOrTerminal(G, c, ϕ)*, that outputs a separator of size $< c$, or a (c, ϕ)-expanders-or-terminal pair $(\mathcal{G}, T)$ for G such that*

1. $|T| \leq n^{1+o(1)}\phi$,
2. $\sum_{H\in\mathcal{G}} |V(H)| = O(n^{1+o(1)})$.

By applying Theorem 3.3 on a graph G with appropriate parameters c and ϕ, we obtain a (c, ϕ)-expanders-or-terminal pair $(\mathcal{G}, T)$ for G such that $|T| \leq n^{1+o(1)}\phi$ and total size of ϕ-vertex expanders is almost linear. Since every graph $H \in \mathcal{G}$ is a ϕ-vertex expander, we can decide c-connectivity of all graphs in $\mathcal{G}$ by applying Proposition 3.1 on every ϕ-expander in $\mathcal{G}$. This takes $O(n^{1+o(1)}c^{O(c)}\phi^{-2})$ time. By Definition 3.2, the final step for deciding c-connectivity of G is to verify if $\kappa_G(x, y) \geq c$ for all $x, y \in T$. However, it is not known how to compute $\min_{x,y\in T} \kappa_G(x, y)$ in *deterministically* near-linear time[3].

Instead of computing $\kappa_G(x, y)$ for all $x, y \in T$, we compute *c-vertex connectivity mimicking network*. That is, we want to sparsify the graph while preserving small cuts with respect to terminal set T. The intuition is that if G is not c-connected because there is a vertex cut that separates $x \in T$ from $y \in T$, then there must be some $A, B \subseteq T$ where $|A|, |B| \geq c$ such that $\mu_G(A, B) < c$. Therefore, it is enough to compress G into another graph H so that the vertex cut that corresponds

[3] In fact, the problem generalizes vertex connectivity because when $T = V$ it becomes vertex connectivity. In our case, however, we make progress because we can guarantee that $|T| \leq n^{1+o(1)}\phi$.

to $\mu_G(A, B) < c$ is preserved and at the same time H does not have a new vertex cut of size less than c.

More precisely, let G and H be graphs that contain the same terminal set T. We say that G and H are (T, c)-*equivalent* if for all pair $A, B \subseteq T$, we have $\min\{\mu_H(A, B), c\} = \min\{\mu_G(A, B), c\}$. We say that H is c-*cut-recoverable* for G if $V(H) \subseteq V(G)$ and every separator of size $< c$ in H is a separator in G. By Proposition 2.6, $\kappa_H \geq \min\{c, \kappa_G\}$. We now define c-connectivity mimicking network.

Definition 3.4 (c-Vertex-Connectivity Mimicking Network)**.** We say that a graph H is a *pairwise* (T, c)-*sparsifier* for $G = (V, E)$ that contains a terminal set $T \subseteq V$ if

1. G and H are (T, c)-equivalent, and
2. H is c-cut-recoverable for G. In particular, $\kappa_H \geq \min\{c, \kappa_G\}$.

This leads to our second new technical tool. In Section 5, we prove the following:

Theorem 3.5. *Let $G = (V, E)$ be an undirected graph with n vertices and m edges and a terminal set $T \subseteq V$ and $c > 0$ be a parameter. There is an $O(m^{1+o(1)}2^{O(c^2)})$-time algorithm, denoted as* VertexSparsify(G, T, c)*, that outputs a (T, c)-sparsifier H for G where $|E(H)| \leq c|V(H)|$ and $|V(H)| = O(|T|2^{O(c^2)})$.*

Recall that the terminal set T in (c, ϕ)-expanders-or-terminal pair is of size at most $n^{1+o(1)}\phi$ (Theorem 3.3). After applying Theorem 3.5 using parameters (G, T, c), we obtain a (T, c)-sparsifier H for G where $|E(H)| \leq c|V(H)|$ and $|V(H)| = O(|T|2^{O(c^2)}) = O(n^{1+o(1)}\phi 2^{O(c^2)})$. The key claim is that G is c-connected if and only if H is c-connected. Furthermore, if we set $\phi^{-1} = 10n^{o(1)}2^{O(c^2)}$, then we have that $|V(H)| \leq n/10$ and $|E(H)| \leq nc/10$. Therefore, we can recurse on graph H to decide c-connectivity of G. This can repeat at most $O(\log n)$ time, and by the choice of ϕ, each iteration takes $O(m^{1+o(1)} + n^{1+o(1)}c^{O(c)} + m^{1+o(1)}2^{O(c^2)}) = O(m^{1+o(1)}2^{O(c^2)})$.

We are now ready to prove Theorem 1.1. We describe the algorithm and analyze correctness and its running time. We start with the algorithm description. Recall that ExpandersOrTerminal is the algorithm in Theorem 3.3; and VertexSparsify is the algorithm in Theorem 3.5. We use $\phi^{-1} = 10n^{o(1)}2^{O(c^2)}$. If the graph is ϕ-vertex expander, we can decide c-connectivity efficiently using Proposition 3.1. The algorithm is described in Algorithm 1.

Algorithm 1: Main(G, c)

Input: A graph $G = (V, E)$, and connectivity parameter c
Output: A separator of size $< c$ or $\perp$ certifying that G is c-connected.

if $|V| \leq 100c$ **then** output $\perp$ if $\kappa_G \geq c$, and output a separator of size $< c$ otherwise.
if min-degree of $G < c$ **then return** the set of neighbors of the min degree vertex.
Call ExpandersOrTerminal(G, c, ϕ) where $\phi^{-1} = 10n^{o(1)}2^{O(c^2)}$
if a separator Z is returned **then return** Z.
Otherwise, (c, ϕ)-expanders-or-terminal pair $(\mathcal{G}, T)$ is returned.
if $\exists H \in \mathcal{G}$ s.t. H is not c-connected **then return** the corresponding min separator.
$T \leftarrow \bigcup_{v \in T} N_G^c(v)$ where $N_G^c(v)$ is an arbitrary set of c neighbors of v
$H \leftarrow$ VertexSparsify(G, T, c)
return Main(H, c)

Correctness. We prove that Algorithm 1 outputs a separator of size $< c$ if and only if G is not c-connected. We first prove that if G is c-connected, then Algorithm 1 returns $\perp$. By

Theorem 3.3, we have that $\textsc{ExpandersOrTerminal}(G, c, \phi)$ must return an (ϕ, c)-expanders-or-terminal pair $(\mathcal{G}, T)$ for G. By Definition 3.2, every graph $H \in \mathcal{G}$ is c-connected. By Theorem 3.5, $\textsc{VertexSparsify}(G, T, c)$, that outputs a (T, c)-sparsifier H for G. By Definition 3.4, $\kappa_H \geq \min\{c, \kappa_G\} \geq c$. Therefore, H is also c-connected. Algorithm 1 then recurses on H. By the choice of ϕ, we have $|V(H)| \leq n/10$. The process repeats for at most $O(\log n)$ time, and eventually the graph in the base case is c-connected. Therefore, Algorithm 1 returns $\perp$.

Next, we prove that if G is not c-connected, then the Algorithm 1 returns a separator of size $< c$. By Theorem 3.3, if $\textsc{ExpandersOrTerminal}(G, c, \phi)$ returns a separator, then G is not c-connected, and we are done. Now assume that $\textsc{ExpandersOrTerminal}(G, c, \phi)$ returns an (ϕ, c)-expanders-or-terminal pair $(\mathcal{G}, T)$ for G (Definition 3.2). If one of the graph in $\mathcal{G}$ is not c-connected, then the algorithm returns the corresponding min separator in H of size $< c$. Since every graph in $\mathcal{G}$ is c-mincut-recoverable (Definition 2.5), the same separator is also a separator in G and we are done. Now assume that every graph in $\mathcal{G}$ is c-connected. Since $(\mathcal{G}, T)$ is a (ϕ, c)-expanders-or-terminal pair, there is $x, y \in T$ such that $\kappa_G(x, y) < c$. By Theorem 3.5, $\textsc{VertexSparsify}(G, T, c)$, that outputs a (T, c)-sparsifier H for G. Observe that, prior to running $\textsc{VertexSparsify}(G, T, c)$, T is updated to be $\bigcup_{v \in T} N^c_G(v)$ where $N^c_G(v)$ be an arbitrary set of c neighbors of v.

Claim 3.6. *H is not c-connected.*

Proof. Since $\kappa_G(x, y) < c$, there is a vertex cut (L, S, R) in G where $x \in L \cap T$ and $y \in R \cap T$ and $|S| < c$. Since $N^c_G(x) \subseteq T$ and $N^c_G(y) \subseteq T$, we have that $|T \cap (L \cup S)| \geq c+1$ and $|T \cap (S \cup R)| \geq c+1$. Let $A = T \cap (L \cup S)$ and $B = T \cap (S \cup R)$. We have that S is an (A, B)-weak separator in G and thus $\mu_G(A, B) \leq |S| < c$. Since H and G are (T, c)-equivalent, we have that

$$\min\{\mu_H(A, B), c\} = \min\{\mu_G(A, B), c\} = \mu_G(A, B) < c.$$

Therefore, $\mu_H(A, B) < c$, and thus there is a separator S' such that $H - S'$ does not have a path from A to B (note that S' may contain A and B). Since $|S'| < c$, and $|A| \geq c+1, |B| \geq c+1$, A is not contained entirely in S', and also B is not contained entirely in S'. Therefore, there are $x' \in A - S'$ and $y' \in B - S'$ such that there is no path from x' to y' in $H - S'$. So, S' is a separator of size $< c$ in H. $\square$

Since H is not c-connected and the algorithm recurses on H, and repeat for $O(\log n)$ time. It eventually hits the base case where the input graph is not c-connected. The base case will return a separator S of size $< c$. Since H is a (T, c)-sparsifier for G, H is c-cut-recoverable for G. Since c-cut-recoverable property is transitive (i.e., if A is c-cut-recoverable for B and B is c-cut-recoverable for C, then A is c-cut-recoverable for C), we conclude that S is also a separator in G. Therefore, Algorithm 1 must output a separator of size $< c$.

Running Time. Let n be the number of vertices and m be the number of edges of the input graph G. By Line 2, we can assume that $m \geq nc$. By design, we set $\phi^{-1} = 10n^{o(1)}2^{O(c^2)}$. By Theorem 3.3, $\textsc{ExpandersOrTerminal}(G, c, \phi)$ takes $O(m^{1+o(1)}/\phi) = O(m^{1+o(1)}2^{O(c^2)})$ time. If a separator is returned, we are done. Now, assume (c, ϕ)-expanders-or-terminal pair $(\mathcal{G}, T)$ of G is returned. By Theorem 3.3, every graph in $\mathcal{G}$ is ϕ-vertex expander where $\sum_{H \in \mathcal{G}} |V(H)| = O(n^{1+o(1)})$ and also $|T| \leq n^{1+o(1)}\phi$. By Proposition 3.1, we can solve c-connectivity of every ϕ-expander in $\mathcal{G}$ in $O(\sum_{H \in \mathcal{G}} |V(H)| \cdot c^c \phi^{-2}) = O(n^{1+o(1)}2^{O(c^2)})$ time. By Theorem 3.5, $\textsc{VertexSparsify}(G, T, c)$, that outputs a (T, c)-sparsifier H for G in $O(m^{1+o(1)}2^{O(c^2)})$ time. Finally, we prove that $|E(H)| \leq m/10$ and $|V(H)| \leq n/10$. By Theorem 3.5, we have $|E(H)| \leq |V(H)|c$ and $|V(H)| = O(|T|2^{O(c^2)})$. Since $|T| \leq n^{1+o(1)}\phi$, and $\phi^{-1} = 10n^{o(1)}2^{O(c^2)}$ (with appropriate parameters), we have that $|V(H)| \leq n/10$, and also $|E(H)| \le |V(H)|c \le m/10$. Therefore, we repeat at most $O(\log n)$ time where each time takes total $O(m^{1+o(1)}2^{O(c^2)})$.

4 Expanders or Terminals

This section is devoted to proving Theorem 3.3 which is the basis of our approach. We organize the proof into four subsections. We give an overview of the proof in Section 4.1. In Section 4.2, we prove structural lemma for vertex cuts. In Section 4.3, we show a recursive algorithm that computes the object from Definition 3.2. In Section 4.4, we show that the algorithm can be implemented in almost linear-time using the notion of most balanced sparse cuts.

4.1 Overview

Our new insight for proving Theorem 3.3 is the following structural lemma about vertex cuts:

Lemma 4.1 (Informal)**.** *For any vertex cut (L, S, R) of G, consider forming the graph G_L by*

1. *removing all vertices of R,*
2. *replacing R with a clique K_R of size c, and*
3. *adding a biclique between S and K_R,*

as well as G_R symmetrically. Then, G is c-connected if and only if

1. *for all pair $x, y \in S$, $\kappa(x, y) \ge c$, and*
2. *both G_L and G_R are c-connected.*

See Section 4.2 for the formal statement and its proof. The structural lemma suggests a divide-and-conquer algorithm for computing (c, ϕ)-expanders-or-terminal pair for G. In the base case, if G is already a ϕ-expander, then we return $(\{G\}, \{\})$. Otherwise, G has ϕ-sparse cuts. Among of all ϕ-sparse cuts, we select the one (L, S, R) which maximizes $\min\{|L|, |R|\} + |S|$ with some $n^{o(1)}$ approximation factor from the most balanced one. From an approximate most balanced cut (L, S, R), we construct G_L and G_R as defined in Lemma 4.1, and recurse on G_L and G_R. Let $(\mathcal{G}_L, T_L)$ be a (c, ϕ)-expanders-or-terminal pair for G_L and let $(\mathcal{G}_R, T_R)$ be a (c, ϕ)-expanders-or-terminal pair for G_R. Then, by applying Lemma 4.1, we can prove that $(\mathcal{G}_L \cup \mathcal{G}_R, T_L \cup T_R \cup S)$ is also a (c, ϕ)-expanders-or-terminal pair for G. We refer to Section 4.3 for more details. To bound the running time, suppose that we can compute $n^{o(1)}$-approximate most balanced cut in $O(m^{1+o(1)}/\phi)$ time (see Lemma 4.11). Given this subroutine, we can show that the recursion depth is $O(\log n)$ using a standard argument, and therefore the total running time is $O(m^{1+o(1)}/\phi)$. Furthermore, we should expect the size of terminal set to be $n^{1+o(1)}\phi$ because we only add the new terminals from ϕ-sparse cuts.

4.2 A Divide-and-Conquer Lemma for Vertex Cuts

Definition 4.2 (c-Left and c-Right Graphs)**.** Let (L, S, R) be a vertex cut in graph $G = (V, E)$ and $c > 0$ be a parameter such that $|R| \ge |L| > c$. We define *c-left graph G_L* (w.r.t. (L, S, R)) as the graph G after the following operations:

1. Let $K_R \subseteq R$ be an arbitrary subset of c vertices of R,
2. remove vertices in $R - K_R$,
3. for all pair $x, y \in K_R$, add edge (x, y),
4. for all $x \in S$ and $y \in K_R$, add edge (x, y),
5. (optional) remove all edges in $E(S, S)$.

Intuitively, G_L is G after replacing R with a clique of size c followed by adding edges between the clique and S. Furthermore, each vertex K_R corresponds to one of the c distinct vertices in R.

Similarly, we define *c-right graph* G_R (w.r.t. (L,S,R)) as the graph G after the following operations:

1. Let $K_L \subseteq L$ be an arbitrary subset of c vertices of L,
2. remove vertices in $L - K_L$,
3. for all pair $x, y \in K_L$, add edge (x,y),
4. for all $x \in S$ and $y \in K_L$, add edge (x,y),
5. (optional) remove all edges in $E(S,S)$.

Intuitively, G_R is G after replacing L with a clique of size c followed by adding edges between the clique and S. Furthermore, each vertex K_L corresponds to one of the c distinct vertices in L.

Observe that, by definition, $V(G_L) \subseteq V(G)$ and $V(G_R) \subseteq V(G)$.

Lemma 4.3 (Divide-and-Conquer Lemma for Vertex Cuts)**.** *Given a vertex cut (L,S,R) in $G = (V,E)$ and a parameter $c > 0$ where $|R| \geq |L| \geq c$, G is c-connected if and only if it satisfies the following properties:*

1. $\kappa_G(x,y) \geq c$ for all $x, y \in S$, and

2. Both of its c-left and c-right graphs (i.e., G_L and G_R) are c-connected.

Furthermore, G_L and G_R are c-mincut-recoverable for G.

We divide the proof of Lemma 4.3 into two lemmas. Let (L,S,R) be a vertex cut in $G = (V,E)$, G_L be a c-left graph, and G_R be a c-right graph of G w.r.t. (L,S,R), respectively.

Lemma 4.4. *If G is not c-connected, then at least one of the properties in Lemma 4.3 is false.*

Lemma 4.5. *G_L and G_R are c-mincut-recoverable for G. Therefore, if at least one of the properties in Lemma 4.3 is false, then G is not c-connected.*

We now prove each lemma in turn.

Proof of Lemma 4.4. First, observe that we can assume $|S| \geq c$ and $\kappa_G(x,y) \geq c$ for all $x, y \in S$. This is because if $|S| < c$, then G_L and G_R are not c-connected (since S is a separator in both G_L and G_R) and we are done. Also, if there is a pair of vertices $x, y \in S$ such that $\kappa_G(x,y) < c$, then the first property of Lemma 4.3 is false and we are done too.

Now, since G is not c-connected, G has a vertex cut (L^*, S^*, R^*) where $|S^*| < c$. We will prove that S^* is a separator in G_L or G_R.

Claim 4.6. *We have $S \cap L^* = \emptyset$ or $S \cap R^* = \emptyset$. Also, the two sets $S \cap L^*$ and $S \cap R^*$ cannot be both empty.*

Proof. We prove the first statement. Suppose the two sets are both non-empty. There are $u \in S \cap L^*$ and $v \in S \cap R^*$, and thus $u \in S, v \in S$. So, $\kappa_G(u,v) \geq c$. Since $u \in L^*$ and $v \in R^*$, we have $\kappa_G(u,v) \leq |S^*| < c$, a contradiction. Next, we prove the second statement. Suppose the two sets are both empty. Then, $S \subseteq S^*$, and thus $c \leq |S| \leq |S^*| < c$, a contradiction. ☐

We assume WLOG that $S \cap R^* = \emptyset$ and $S \cap L^* \neq \emptyset$. The case where $S \cap L^* = \emptyset$ is symmetric. Since $S \cap R^* = \emptyset$, we have $L \cap R^* \neq \emptyset$ or $R \cap R^* \neq \emptyset$. It remains to prove the following claim, see Figure 1 for illustration.

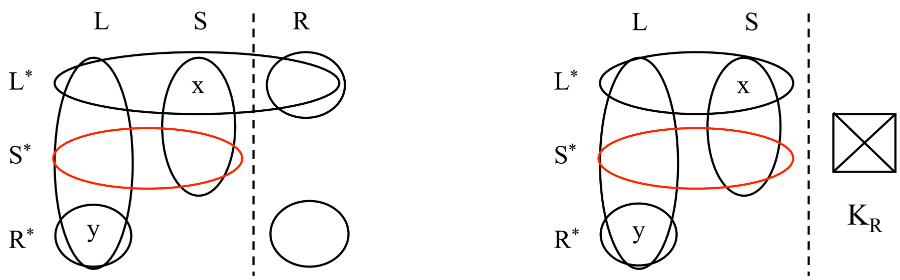

Figure 1: A crossing-diagram for two vertex cuts (L, S, R) and (L^*, S^*, R^*) before and after transformation from G to G_L. If $L \cap R^* \neq \emptyset$, we show that S^* is a vertex cut in G_L

Claim 4.7. *If $L \cap R^* \neq \emptyset$, then S^* is a separator in G_L, and if $R \cap R^* \neq \emptyset$, then S^* is a separator in G_R.*

Proof. We prove the case $L \cap R^* \neq \emptyset$. Let $x \in S \cap L^*$ and $y \in L \cap R^*$. Recall that K_R is a clique that is added to G_L, and denote K_R as the set of vertices of K_R. Observe that $((L^* - R) \cup K_R, S^*, R^* - R)$ forms a partition of vertex set in G_L. It remains to prove that there is no edge from $(L^* - R) \cup K_R$ to $R^* - R$ in G_L. If true, then the partition is a vertex cut in G_L and we are done. Indeed, since (L^*, S^*, R^*) is a vertex cut in G, there is no edge from $(L^* - R, R^* - R)$ in G_L. We next prove that there is no edge from K_R to $R^* - R$. Let v be an arbitrary vertex in K_R. By Definition 4.2, $N_{G_L}(v) \subseteq S \cup K_R$, which is disjoint from L. But observe that $R^* - R \subseteq L$.Therefore, there is no edge from K_R to $R^* - R$, and we are done. The proof for the case $R \cap R^* \neq \emptyset$ is symmetric. □

Remark. The same proof goes through if we further remove edges in $E_{G_L}(S, S)$ in G_L (and also $E_{G_R}(S, S)$ in G_R). □

Proof of Lemma 4.5. We prove the result for G_L (the argument for G_R is symmetric). Fix $s, t \in V$ such that $\kappa_{G_L}(s, t) < c$. Let (L', S', R') be a vertex cut in G_L where $|S'| = \kappa_{G_L}(s, t) < c, s \in L', t \in R'$. Recall that K_R is the clique in G_L.

Claim 4.8. *$K_R \subseteq L'$ or $K_R \subseteq R'$.*

Proof. We first show that $K_R \subseteq L' \cup S'$ or $K_R \subseteq S' \cup R'$. Since $|K_R| = c > |S'|$, we have $K_R \not\subseteq S'$. Since K_R is a clique, K_R cannot be in both L' and R'. Therefore, $K_R \subseteq L' \cup S'$ or $K_R \subseteq S' \cup R'$.

Next we show that $K_R \cap S' = \emptyset$. Suppose otherwise. Let v be an arbitrary vertex in $S' \cap K_R$. Since $K_R \subseteq L' \cup S'$ or $K_R \subseteq S' \cup R'$, we have that $N(v) \subseteq L' \cup S'$ or $N(v) \subseteq R' \cup S'$. If $N(v) \subseteq L' \cup S'$, then $(L' \cup \{v\}, S' - \{v\}, R')$ is a vertex cut. If $N(v) \subseteq R' \cup S'$, then $(L', S' - \{v\}, R' \cup \{v\})$ is a vertex cut. Either way, $S' - \{v\}$ is an (s, t)-separator which is smaller than min (s, t)-separator, a contradiction. □

Since $K_R \subseteq L'$ or $K_R \subseteq R'$, by replacing K_R with the vertex set R, we have that S' is also a separator in G. Furthermore, for any $x \in L'$ and $y \in R'$, we have that S' is also (x, y)-separator in G. This completes the proof of Lemma 4.5.

Remark. In addition, the same proof goes through even if we further remove edges in $E_{G_L}(S, S)$ in G_L (and also $E_{G_R}(S, S)$ in G_R) because of the following claim:

Claim 4.9. *If $K_R \subseteq L'$, then $S \subseteq L' \cup S'$. Similarly, if $K_R \subseteq R'$, then $S \subseteq R' \cup S'$.*

Proof. We prove the first statement. The second statement is similar. Suppose $S \not\subseteq L' \cup S'$. There is a vertex $r \in S \cap R'$. Since there are biclique edges between S and K_R, there is an edge between r and a vertex in K_R. Since $r \in R'$ and $K_R \subseteq L'$, there is an edge between L' and R' in G_L, a contradiction. $\square$

Observe that G can be obtained from G_L by replacing K_R with the vertex set R and adding original edges E' inside S. By Claim 4.9, every edge $(u, v) \in E'$ does not end with L' and R' simultaneously. Using the same argument as above, S' is a separator in G (even we add edges inside S back). $\square$

4.3 An Algorithm for Expanders-or-Terminal Pair

Observe that if G is a ϕ-vertex expander, then $(\{G\}, \{\})$ is (c, ϕ)-expanders-or-terminal pair of G. If G is not a ϕ-expander, there is a ϕ-sparse cut (L, S, R). Then, we can recurse on the c-left and c-right graphs of G w.r.t. (L, S, R) (Definition 4.2). At the end of recursion, we obtain a set of ϕ-expanders and a set of ϕ-sparse cuts. By Lemma 4.3, we can show the pair of sets above form a (c, ϕ)-expander-or-terminal family. Algorithm 2 shows the detail of the algorithm.

Algorithm 2: SlowExpandersOrTerminals$(G, c, \phi, \bar{\phi})$

Input: A graph $G = (V, E)$, and connectivity parameter c and $\phi \in (0, 1/10)$, $\bar{\phi} \geq \phi$ where $\phi \leq (2c)^{-1}$.

Output: A (c, ϕ)-expanders-or-terminal pair of G.

if G is a ϕ-vertex expander **then return** $(\{G\}, \{\})$.
Let (L, S, R) be a $\bar{\phi}$-sparse cut in G.
Let G_L and G_R be c-left and c-right graphs of G w.r.t. (L, S, R).
$(\mathcal{G}_L, T_L) \leftarrow$ SlowExpandersOrTerminals$(G_L, c, \phi, \bar{\phi})$
$(\mathcal{G}_R, T_R) \leftarrow$ SlowExpandersOrTerminals$(G_R, c, \phi, \bar{\phi})$
return $(\mathcal{G}_L \cup \mathcal{G}_R, T_L \cup T_R \cup S)$.

Lemma 4.10. *Algorithm 2 takes a graph $G = (V, E)$ where $m = |E|, n = |V|$ and c, ϕ as inputs and outputs a (c, ϕ)-expanders-or-terminal pair $(\mathcal{G}, T)$ of G.*

Proof. We prove that Algorithm 2 outputs a (c, ϕ)-expanders-or-terminal pair of every connected graph G by induction on number of vertices N. Observe that the parameters $c, \phi, \bar{\phi}$ are fixed throughout the algorithm. If $|V(G)| \leq 1/(2c)$, then G is always a ϕ-expander since $\phi \leq 1/(2c)$, and thus the pair $(\{G\}, \{\})$ is (c, ϕ)-expanders-or-terminal pair of G. For all $N \geq 1/(2c)$, we now assume that Algorithm 2 outputs a (c, ϕ)-expanders-or-terminal pair of every connected graph of at most $N - 1$ vertices, and prove that Algorithm 2 outputs a (c, ϕ)-expanders-or-terminal pair of every connected graph of at most N vertices.

Let G be a graph with N vertices. If G is a ϕ-expander, we are done. Otherwise, we are given a $\bar{\phi}$-sparse cut (L, S, R). Let G_L and G_R be the c-left and c-right graphs of G w.r.t. (L, S, R), respectively. We prove that $n_{G_L} \leq N - 1$ and $n_{G_R} \leq N - 1$. Indeed, since (L, S, R) is $\bar{\phi}$-sparse, $|L| + 1 \geq \bar{\phi}^{-1} \geq 2c$, and thus $|R| \geq |L| \geq 2c - 1 > c$. Therefore, $n_{G_L} = c + |S| + |R| \leq |L| - 1 + |S| + |R| = N - 1$. Similarly, we have $n_{G_R} = |L| + |S| + c \leq |L| + |S| + |R| - 1 = N - 1$. By inductive hypothesis, $(\mathcal{G}_L, T_L)$ is (c, ϕ)-expanders-or-terminal pair for G_L, and similarly $(\mathcal{G}_R, T_R)$ is (c, ϕ)-expanders-or-terminal pair for G_R. Thus, every graph in $\mathcal{G}_L$ and in $\mathcal{G}_R$ is a ϕ-vertex expander.

It remains to prove that $(\mathcal{G}_L \cup \mathcal{G}_R, T_L \cup T_R \cup S)$ is a (c, ϕ)-expanders-or-terminal pair of G. We first prove that if G is c-connected, then both conditions in Definition 3.2 hold. Since G is c-connected, we have that for all $x, y \in V(G)$, $\kappa_G(x, y) \geq c$. In particular, for all $x, y \in T_L \cup T_R \cup S$, $\kappa_G(x, y) \geq c$. Next, we prove that every graph in $\mathcal{G}_L \cup \mathcal{G}_R$ is ϕ-vertex expander and c-connected. Since G is c-connected, Lemma 4.3 implies that G_L and G_R are c-connected and $|S| \geq c$. Since $(\mathcal{G}_L, T_L)$ is (c, ϕ)-expanders-or-terminal pair for G_L, and similarly $(\mathcal{G}_R, T_R)$ is (c, ϕ)-expanders-or-terminal pair for G_R, every graph $H \in \mathcal{G}_R \cup \mathcal{G}_L$ is ϕ-expander, and c-connected.

Finally, we prove that if G is not c-connected, then at least one of the conditions Definition 3.2 is violated for $(\mathcal{G}_L \cup \mathcal{G}_R, T_L \cup T_R \cup S)$. If $\kappa_G(x, y) < c$ for some $x, y \in S$, then we are done. Now assume that for all $x, y \in S, \kappa_G(x, y) \geq c$. So, Lemma 4.3 implies that G_L is not c-connected or G_R is not c-connected. Now, we assume WLOG that G_L is not c-connected. Since $(\mathcal{G}_L, T_L)$ is (c, ϕ)-expanders-or-terminal pair of G_L and G_L is not c-connected, at least the one of the conditions Definition 3.2 is violated for $(\mathcal{G}_L, T_L)$ for G_L. If there is $H \in \mathcal{G}_L$ that is not c-connected, then we are done. Now, assume that there is a pair $x, y \in T_L$ such that $\kappa_{G_L}(x, y) < c$. Since G_L is c-mincut recoverable, we have $\kappa_G(x, y) < c$, and this completes the proof. □

4.4 A Fast Implementation of Algorithm 2

We start with an important primitive for computing sparse cut.

Lemma 4.11 ([LS22]). *Let $G = (V, E)$ be an n-vertex m-edge graph. Given a parameter $0 < \bar{\phi} \leq 1/10$ and $1 \leq r \leq \lfloor \log_{20} n \rfloor$, there is a deterministic algorithm,* BalancedCutOrExpander$(G, \bar{\phi}, r)$, *that computes a vertex cut (L, S, R) (but possible $L = S = \emptyset$) of G such that $|S| \leq \bar{\phi}|L \cup S|$ which further satisfies*

- *either $|L \cup S|, |R \cup S| \geq n/3$; or*
- *$|R \cup S| \geq n/2$ and $G[R \cup S]$ is a ϕ-vertex expander for some $\phi = \bar{\phi}/\log^{O(r^5)} m$.*

The running time of this algorithm is $O(m^{1+o(1)+O(1/r)} \log^{O(r^4)}(m)/\phi)$.

When $G[S \cup R]$ is a 2ϕ-vertex expander, our c-right graph G_R is still an ϕ-vertex expander.

Claim 4.12. *If $G[S \cup R]$ is a 2ϕ-vertex expander, then the c-right graph G_R is ϕ-vertex expander.*

Proof. Suppose there is a ϕ-sparse cut (L', S', R') in G_R. By removing the clique in G_R, we obtain another cut (L'', S'', R'') in G where $L'' = L' - V(K_L)$ $S'' = S' - V(K_L)$ and $R'' = R - V(K_L)$. The expansion $h(L'', S'', R'') = \frac{|S''|}{|L''|+|S''|} \leq \frac{|S'|}{|L'|+|S'|-c} \leq 2\frac{|S'|}{|L'|+|S'|} < 2\phi$. So, (L'', S'', R'') is 2ϕ-sparse in G, a contradiction. □

The full algorithm is described in Algorithm 3. Here, we denote G^{orig} as the original input graph. We assume that the min-degree of G^{orig} is $\geq c$. Thus, Definition 4.2 implies that any c-left and c-right graph have min-degree at least c. Therefore, we can assume that min-degree is at least c. During the recursion, if we obtain a balanced sparse cut (L, S, R) such that $|S| < c$, then we know that the original input graph G^{orig} is not c-connected. Furthermore, if we want to output the corresponding cut, we use the fact that G_L and G_R are c-mincut-recoverable. Therefore, it is enough to compute one min s, t separator where s and t are on the different side of (L, S, R) for an arbitrary $s \in L$ and $t \in R$. This takes $O(mc)$ time by Fold-Fulkerson max-flow algorithm.

Algorithm 3: FastExpandersOrTerminals(G, c, ϕ)

Global variables: the original input graph G^{orig} whose min degree is $\geq c$.
Input: A graph $G = (V, E)$, and connectivity parameter c and $\phi \in (0, 1)$ where $\phi \leq (2c)^{-1}$.
Output: A (c, ϕ)-expanders-or-terminal pair of G.

```
1  r ← log log |V|
2  φ̄ ← 2 · φ · log^{O(r^5)} |E|
3  (L, S, R) ← BalancedCutOrExpander(G, φ̄, r)
4  if L = S = ∅ then
5      By Lemma 4.11, G is a 2φ-vertex expander.
6      return ({G}, {})
7  if |S| < c then
8      terminate and output min (s, t)-separator for an arbitrary s ∈ L and t ∈ R.
9  By Lemma 4.11, (L, S, R) is φ̄-sparse.
10 Let G_L and G_R be c-left and c-right graphs of G w.r.t. (L, S, R).
11 Remove the edges in E_{G_L}(S, S) from G_L.
12 (𝒢_L, T_L) ← FastExpandersOrTerminals(G_L, c, φ)
13 if min{|L ∪ S|, |R ∪ S|} ≥ |V|/3 then
14     Remove the edges in E_{G_R}(S, S) from G_R.
15     (𝒢_R, T_R) ← FastExpandersOrTerminals(G_R, c, φ)
16     return (𝒢_L ∪ 𝒢_R, T_L ∪ T_R ∪ S).
17 else
18     By Claim 4.12, G_R is a φ-vertex expander.
19     return (𝒢_L ∪ {G_R}, T_L ∪ S).
```

Lemma 4.13. *Let G^{orig} be the original input graph with n vertices and m edges, and c, ϕ be parameters where G^{orig} has min degree $\geq c$. Then, Algorithm 3, when applying with the inputs (G^{orig}, c, ϕ) either outputs a separator of size $< c$ in G^{orig} or a (c, ϕ)-expanders-or-terminal pair $(\mathcal{G}, T)$ for G^{orig} satisfying:*

1. $\sum_{H \in \mathcal{G}} |V(H)| \leq n(1 + 3\bar{\phi})^{O(\log n)}$, *and*
2. $|T| \leq \bar{\phi} n (1 + 3\bar{\phi})^{O(\log n)}$,

where $\bar{\phi} = \phi \cdot m^{o(1)}$. The algorithm runs in $m^{1+o(1)}(1 + 4\bar{\phi})^{O(\log n)} \cdot \phi^{-1}$ time.

Lemma 4.13 with appropriate parameters immediately implies Theorem 3.3. The rest of this section is devoted to proving Lemma 4.13.

Correctness. Consider the execution of Algorithm 3 on $(G^{\text{orig}}, c, \phi)$. First we prove that if the condition in Line 8 is true at some point, then G^{orig} is indeed not c-connected. Consider the recursion tree $\mathcal{T}$ starting with G^{orig}. Each internal node can be represented as a graph G along with a $\bar{\phi}$-sparse cut (L, S, R), and we recurse on left subproblem G_L and right subproblem G_R. The leaf nodes represent the instance to be a ϕ-expander. Let G' be the first subproblem in the recursion tree that Line 8 is executed. Since $|S| < c$, we have that G' is not c-connected. Let P be a path from the root node to the subproblem G' in the recursion tree $\mathcal{T}$. By applying Lemma 4.3 along the path P, we derive that G^{orig} is not c-connected, and we are done. From now, we assume that Line 8 is always false (i.e., $|S| \geq c$ every time). By the description of Algorithm 3 along with Lemma 4.11 and Claim 4.12 , the execution of Algorithm 3 is identical to that of Algorithm 2 (where depending on the conditions, G_L and G_R may or may not have edges inside S). By Lemma 4.10, Algorithm 3 outputs a (c, ϕ)-expanders-or-terminal pair for G^{orig}. This completes the correctness proof.

Next, we bound the size of the (c, ϕ)-expanders-or-terminal pair and the running time. We first show that the recursion depth is $O(\log n)$.

Claim 4.14. *The depth of the recursion tree $\mathcal{T}$ is $O(\log n)$.*

Proof. Let G be any internal node of the recursion tree $\mathcal{T}$. Let n_0, n_L, n_R be the number of vertices in G, G_L and G_R, respectively. We prove that the size is reduced by a constant factor each time. Let (L, S, R) be the $\bar{\phi}$-sparse cut in G. If $\min\{|L \cup S|, |R \cup S|\} \geq n_0/3$, then we have $n_L \leq 2n_0/3$ and $n_R \leq 2n_0/3$. Otherwise, we have that G_R is a ϕ-expander and its right subproblem becomes a leaf node. It remains to prove that $n_L \leq 2n_0/3$ in this case. Since $|S| \leq \bar{\phi} n_0$ and $|R \cup S| \geq n_0/2$, we have $|R| \geq n_0/4$, and thus $|L \cup S| = |L| + |S| \leq 3n_0/4 + \bar{\phi} n_0 \leq 0.85n_0$. □

Sizes. If the pair $(\mathcal{G}, T)$ is returned, then Line 8 is always false (i.e., $|S| \geq c$ every time). We now bound the total number of vertices in all graphs in $\mathcal{G}$ and the size of the terminal set T. Let $\mathcal{T}$ be the recursion tree of starting with G^{orig} where the leaf nodes represent ϕ-expanders, and internal nodes are represented by a graph G and its $\bar{\phi}$-sparse cut (L, S, R). Fix an arbitrary internal node in the recursion tree G and (L, S, R). Let G_L and G_R be c-left and c-right graphs of G w.r.t. (L, S, R). Let n_0, n_L and n_R be the number of vertices in G, G_L and G_R, respectively. By definition of G_L and G_R,

$$n_L + n_R = n_0 + |S| + 2c \leq n_0 + 3|S| \leq n_0(1 + 3\bar{\phi}).$$

The first inequality follows because $n_L = |L| + |S| + c$, and $n_R = c + |S| + |R|$. The last inequality follows because (L, S, R) is $\bar{\phi}$-sparse. Therefore, at level i, the total number of vertices is at most $n(1 + 3\bar{\phi})^{i-1}$. Let ℓ be the recursion depth. We have that the total number of vertices at the last level is at most $O(n(1 + 3\bar{\phi})^{\ell+1})$. To bound the number of terminals, observe that it is at most $\bar{\phi}$ times total number of vertices generated by the recursion. Thus, $|T| = O(\bar{\phi} n(1 + 3\bar{\phi})^{\ell+1})$. The results follow by Claim 4.14.

Running Time. Consider the same recursion tree $\mathcal{T}$. We first bound the total number of edges over all the algorithm. Let m_0, m_L and m_R be the number of edges in G, G_L and G_R, respectively. We assume that $\min\{|L \cup S|, |R \cup S|\} \geq n_0/3$ because otherwise G_R is a ϕ-vertex expander, and we only go for the left recursion. By Line 11 and Line 14, both G_L and G_R do not have edges in $E(S, S)$. Therefore,

$$m_L + m_R \leq m_0 + 2(|S|c + c^2) \leq m_0 + 4|S|c \leq m_0 + 4cn_0\bar{\phi} \leq m_0(1 + 4\bar{\phi}).$$

The first inequality follows because the set of new edges are those in the clique in both G_L and G_R and biclique edges. Note that there is no edge inside S for both graphs. The second inequality follows because $|S| \geq c$. The third inequality follows because (L, S, R) is $\bar{\phi}$-sparse. The last inequality follows because min-degrees of G_L and G_R are at least c.

Therefore, at level i, the total number of edges is at most $O(m(1 + 4\bar{\phi})^{i-1})$. Summing over all levels, we have that the total number of edges is $O(m(1 + 4\bar{\phi})^{\ell+1})$ where ℓ is the recursion depth. By Claim 4.14, the recursion depth $\ell = O(\log n)$. Finally, the running time follows because each node v in the recursion tree takes $O(m'^{1+o(1)}/\phi)$ time where m' is the number of edges of the graph at node v.

5 c-Vertex Connectivity Mimicking Networks

This section and the rest are devoted to proving Theorem 3.5. Let $G = (V, E)$ be a graph and $T \subseteq V$ be a terminal set. Our approach for computing a (T, c)-sparsifier for G is to compute a (T, c)-*covering set*.

Definition 5.1. A vertex set $Z \subseteq V$ is (T, c)-*covering* if for all $A, B \subseteq T$ such that $\mu(A, B) \leq c$, Z contains some min (A, B)-weak separator.

Our technical result is the following.

Theorem 5.2. *Given $G = (V, E), T \subseteq V$, and a parameter $c > 0$, we can compute (T, c)-covering set of size $O(|T|2^{O(c^2)})$ in $O(m^{1+o(1)}2^{O(c^2)})$ time.*

We prove Theorem 5.2 in Section 6. Once we have a (T, c)-covering set Z, we show that we can compute a (T, c)-sparsifier for G of small size efficiently.

Lemma 5.3. *Given a (T, c)-covering set Z of G and an integer $c > 0$, there is an $O(mc)$-time algorithm that outputs a (T, c)-sparsifier for G where the number of vertices is $|T \cup Z|$ and the number of edges is at most $c|T \cup Z|$.*

Theorem 5.2 and lemma 5.3 imply Theorem 3.5. Next, we prepare necessary tools for proving Lemma 5.3. We start with the key primitive operation for constructing (T, c)-sparsifier for G, which we call *vertex closure operation* in Section 5.1. Then, we show a fast offline vertex closure oracle in Section 5.2, which is of independent interest. Finally, we prove Lemma 5.3 in Section 5.3.

5.1 Vertex Closure Operation

We start with the definition.

Definition 5.4. *Vertex closure* operation for a vertex v in graph G is defined as follows: add all edges between vertices in $N_G(v)$, and remove v from G. We denote $\mathrm{cl}(G, v)$ as the graph G after closing a vertex v.

Intuitively, one can view vertex closure operation as an analogue of edge contraction operation. If x and y are adjacent to each other via an edge e, after contracting e, there is no edge cut separating x and y as they become the same vertex. Similarly, if x and y share a common neighbor via a vertex v, after closing v, there is no vertex cut separating x and y as they are directly connected by an edge now.

We prove that vertex closure operation does not decrease the vertex connectivity.

Lemma 5.5 (Monotonicity Property of Vertex Closure Operation)**.** *Let v be an arbitrary vertex in G. Then, $\mathrm{cl}(G, v)$ is cut-recoverable, i.e., if S is a separator in $\mathrm{cl}(G, v)$, then S is also a separator in G. In particular, $\kappa_{\mathrm{cl}(G,v)} \geq \kappa_G$.*

Proof. Suppose there is a vertex cut (L, S, R) in $\mathrm{cl}(G, v)$. To undo contraction, we add vertex v and remove edges in $\mathrm{cl}(G, v)$ that do not exist in G. Hence, $(L, S \cup \{v\}, R)$ is a vertex cut in G. Next, we prove that $N_G(v) \cap L = \emptyset$ or $N_G(v) \cap R = \emptyset$. Suppose otherwise. By definition of vertex contraction, there is an edge between L and R in $\mathrm{cl}(G, v)$, contradicting to the fact that (L, S, R) is a vertex cut in $\mathrm{cl}(G, v)$. Assume WLOG that $N_G(v) \cap L = \emptyset$ (the case $N_G(v) \cap R = \emptyset$ is similar). Since $(L, S \cup \{v\}, R)$ is a vertex cut in G and $N_G(v) \cap L = \emptyset$, we have that $(L, S, R \cup \{v\})$ is a vertex cut in G. $\square$

It is convenient to consider a batch version of vertex closure operations. For any subset of vertices $X \subset V$, we denote $\mathrm{cl}(G, X)$ to be the graph G after closing all vertices in X. Note that the resulting graph is the same regardless of the ordering of closure operations.

Proposition 5.6. *For any subset of vertices $X \subset V$ in G, $\mathrm{cl}(G, X)$ is the same graph as the following transformation on G. Let $S_1, \dots, S_\ell$ be connected component of $G[X]$. For each i, we add a clique on $N_G(S_i)$, i.e., adding an edge (u, v) for all pairs (u, v) of vertices in S_i. Then, we remove X from G.*

Proof. It is enough to prove for individual connected component. Fix a connected component S of $G[X]$. We prove that there is an edge (u, v) for every pair of vertices u and v in $N_G(S)$ in $\mathrm{cl}(G, S)$. Fix $u, v \in N_G(S)$. Since S is a connected component, there is a path P from u to v using only vertices from $S \cup \{u, v\}$. For any sequence of closure operation in S, whenever an internal vertex of P is closed, we obtain a path from u to v using only vertices $S \cup \{u, v\}$. After closing the last remaining internal vertex of the path P, we will add an edge from u to v. $\square$

Lemma 5.7 (Offline Closure Oracle)**.** *We are given a graph $G = (V, E)$ with n vertices and m edges and a set of vertices $X \subseteq V$ and an integer $c > 0$. Let $V' = V - X$. Then, there is an $O(mc)$-time deterministic algorithm that outputs a (V', c)-sparsifier for $\mathrm{cl}(G, X)$ with $|V'|$ vertices and at most $c|V'|$ edges.*

We prove Lemma 5.7 in the next section.

5.2 Proof of Lemma 5.7

We describe the algorithm, analyze running time, and argue its correctness. For any subset of vertex $S \subseteq V$, we define a *c-partial clique* on S as follows. If $|S| < c$, then we add an edge to every pair of vertices in S. Otherwise, we select an arbitrary c vertices in S; for each selected vertex, we add an edge to every other vertex in S.

Algorithm. The inputs include G, X and c as defined in the statement.

1. Starting with G. Let $Y_1, \dots, Y_\ell$ be connected components of $G[X]$. For each connected component Y_i, we add a c-partial clique on $N_G(Y_i)$, and remove Y_i. We call the new graph G'.

2. Apply Theorem 2.3 on G' using c as a parameter, and return the resulting graph.

Running Time. It takes $O(m)$ time to find a set of connected components of $G[X]$. The running time for adding partial clique can be computed as follows. For each connected component Y_i, it takes $O(c|N_G(Y_i)|) = O(c \operatorname{vol}_G(Y_i))$ time to add a c-paritial clique. Therefore, the total time in this step is $O(\sum_i c|N_G(Y_i)|) = O(c \sum_i \operatorname{vol}(Y_i)) = O(mc)$. The last equality follows since $Y_1, \dots Y_\ell$ are pairwise disjoint. The new graph has $O(mc)$ edges, and then we apply Theorem 2.3 on the new graph which takes $O(mc)$ time.

Correctness. Let $H = \mathrm{cl}(G, X)$. Observe that H and G' have the same vertex set V'.

Claim 5.8. *G' is a (V', c)-sparsifier for H.*

Therefore, after applying Theorem 2.3 on G' with parameter c, we obtain a (V', c)-sparsifier for H where the set of vertices is V' and the number of edges is at most $|V'|c$. It remains to prove Claim 5.8, which follows from the following two claims.

Claim 5.9. *For every $S \subseteq V'$ such that $|S| < c$, S is a separator in G' if and only if S is a separator in H. Therefore, G' and H are (V', c)-equivalent.*

Claim 5.10. *If $S \subseteq V'$ is a separator in G' such that $|S| < c$, then S is a separator in G. Therefore, G' is c-cut-recoverable for G.*

The two claims are corollaries of the following claim.

Claim 5.11. *If (L, S, R) is a vertex cut in G' where $|S| < c$, then $(L \cup L', S, R \cup R')$ is a vertex cut in G where $L' = \bigcup_{i \in U} Y_i$, and $R' = \bigcup_i Y_i - L'$ for some $U \subseteq \{1, \ldots, \ell\}$.*

Proof. Let G_i be the graph after i iterations of the first step in the algorithm. By design, we have $G_0 = G$, and $G_\ell = G'$. We prove the following for all $i \le \ell$: At the end of iteration i, if S is a separator in G_i of size less than c, then S is a separator in G_{i-1}. Let (L, S, R) be a vertex cut in G_i where $|S| < c$. Since we only add edges between $N(Y_i)$ in G_{i-1} to get G_i, we have that $(L, S \cup Y_i, R)$ is a vertex cut in G_{i-1}. The key claim is $N(Y_i) \subseteq L \cup S$ or $N(Y_i) \subseteq S \cup R$. Indeed, suppose otherwise. Since $|S| < c$, there is one vertex x (out of c vertices) in c-partial clique such that $x \notin S$ and we add an edge from x to every vertex in $N(Y_i)$ (if $|N(Y_i)| < c$, then x can be any vertex outside S). WLOG, assume that $x \in L$. By design, we connect x to every vertex in $N(Y_i)$ including a vertex in R. This implies that there is an edge between L and R in G_{i-1}, contradicting to the fact that (L, S, R) is a vertex cut in G_{i-1}. Since $N(Y_i) \subseteq L \cup S$ or $N(Y_i) \subseteq S \cup R$, we have that either $(L \cup Y_i, S, R)$ or $(L, S, R \cup Y_i)$ is a vertex cut in G_{i-1}. Therefore, S is a separator in G_{i-1}. $\square$

Claim 5.10 follows immediately from Claim 5.11. It remains to derive Claim 5.9.

Proof of Claim 5.9. By Proposition 5.6, G' is a subgraph of H, and thus a separator in H is a separator in G'. It remains to prove that if S where $|S| < c$ is a separator in G', then S is also a separator in H. Let (L, S, R) be a vertex cut in G' where $|S| < c$. By Claim 5.11, $(L \cup L', S, R \cup R')$ is a vertex cut in G where $L' = \bigcup_{i \in U} Y_i$, and $R' = \bigcup_i Y_i - L'$ for some $U \subseteq \{1, \ldots, \ell\}$. Observe that $\mathrm{cl}(G, X)$ will add only edges in the neighbors of Y_i. Since $Y_i \subseteq L'$ or $Y_i \subseteq R'$ for all i, $\mathrm{cl}(G, X)$ does not add an edge between L and R, and thus (L, S, R) is a vertex cut in H. $\square$

5.3 Proof of Lemma 5.3

Let $X = V - (Z \cup T)$ be a set of vertices to be closed, and let $H = \mathrm{cl}(G, X) = (V_H, E_H)$. First, observe that $\kappa_H \ge \kappa_G$ by Lemma 5.5. Next, we claim that H is (T, c)-equivalent to G. If true, then we apply Lemma 5.7 using the graph G and the closure set X to obtain a (V_H, c)-sparsifier for H whose number of vertices is $|T \cup Z|$ and number of edges is at most $c|T \cup Z|$. Since $T \subseteq V_H$, the sparsifier is also a (T, c)-sparsifier for G as desired. By Lemma 5.7, the algorithm takes $O(mc)$ time.

We now prove the claim. We show that for all pair $A, B \subseteq T$, we have that $\min\{c, \mu_G(A, B)\} = \min\{c, \mu_{\mathrm{cl}(G,X)}(A, B)\}$. Fix a pair $A, B \subseteq T$. If $\mu_G(A, B) \ge c$, then $\mu_{\mathrm{cl}(G,X)}(A, B) \ge \mu_G(A, B) \ge c$ (by monotonicity property of closure operations, Lemma 5.5), and we are done. Otherwise, $\mu_G(A, B) < c$. In this case, we prove that $\mathrm{cl}(G, X)$ contains a min (A, B)-weak separator in G. (T, c)-covering set. Hence, S is never closed, i.e., $S \cap X = \emptyset$. Since S and T are not closed, we conclude that S is an (A, B)-weak separator in $\mathrm{cl}(G, X)$. In fact, S is a min (A, B)-weak separator by monotonicity property of closure operations. Therefore, $c > \mu_G(A, B) = |S| = \mu_{\mathrm{cl}(G,X)}(A, B)$.

6 Computing a (T, c)-Covering Sets

This section is devoted to proving Theorem 5.2. We show a reduction from (T, c)-covering sets to (T, c)-*reducing-or-covering partition-set pair*. We set up notations. Denote $\mu(A, B)$ to be the size of a minimum (A, B)-weak separator. Let $\mathcal{S}_{A,B}$ be the set of all min (A, B)-weak separators. Define $\mu^T(A, B) = \min_{S \in \mathcal{S}_{A,B}} |S - T|$. We say that a set S *covers* another set T if $T \subseteq S$.

Definition 6.1. Given a graph $G = (V, E)$ with terminal set T, and parameter $c > 0$, a partition-set pair $(\Pi = \{Z, X_1, \ldots, X_\ell\}, C \subseteq V)$ is (T, c)-*reducing-or-covering* if Π is a partition of V such that Z is an (X_i, X_j)-separator for all $i \neq j$ and for all $A, B \subseteq T$ if $\mu(A, B) \leq c$, then one of the followings is true for some min (A, B)-weak separator S:

1. C covers S,
2. Π *splits* S, i.e., $|S \cap N_G[X_i]| \leq |S| - 1$ for all i,
3. Π T-*hits* S, i.e., $|(S \cap X_i) - T| \leq \mu^T(A, B) - 1$ for all i.

We also say that Z is the *reducer* of the partition Π, and $X_1, \ldots, X_\ell$ is the *non-reducer sequence* of the partition Π.

We first mention here that there exists an almost-linear time algorithm for computing a (T, c)-reducing-or-covering (Π, C) such that $|Z|, |C|$ and the total size of boundaries $\sum_i |N(X_i)|$ are small. This is formalized in Theorem 6.2 below and will be proved in Section 7. We will use it as a key subroutine for constructing a (T, c)-covering set in this section.

Theorem 6.2. *Given a graph G with terminal set T and two parameters $c > 0, \phi \in (0, 1)$ where G has arboricity c and $k = |T|$, there is an $O(m^{1+o(1)}\phi^{-4} \cdot 2^{O(c^2)})$-time algorithm that outputs a (T, c)-reducing-or-covering partition-set pair $(\{Z, X_1, \ldots, X_\ell\}, C)$ of G such that*

- $|Z| = O((k + \phi n^{1+o(1)})c^2)$,
- $|C| = O((k + \phi n^{1+o(1)})2^{O(c^2)})$, *and*
- $\sum_{i=i}^{\ell} |N(X_i)| = O((k + \phi n^{1+o(1)})c^2)$.

Definition 6.1 might not be very intuitive at first. Let us explain the terms "split" and "T-hits" more illustratively here. Let Π be a partition of V from Definition 6.1. Let S be an (A, B)-weak separator in G of size $\leq c$ for some $A, B \subseteq T$. Observe that Π splits S if and only if there exists i and $x, y \in S$ such that $N(X_i)$ is (x, y)-separator. If Π does not split S, then $S \subseteq N[X_i]$ for some i. In this case, note that $(S \cap X_i) - T = S - T - N(X_i)$. So Π T-hits S means that $N(X_i)$ contains enough number of non-terminal vertices of S so that $|S \cap X_i - T| = |S - T - N(X_i)| \leq \mu^T(A, B) - 1$. In particular, $N(X_i)$ must contain at least one vertex from $S - T$, otherwise $|S - T - N(X_i)| \geq \mu^T(A, B)$.

Now, we explain why we say that (Π, C) is "reducing-or-covering". For $i \leq \ell$, define $G_i = G[N[X_i]]$ and $T_i = N(X_i) \cup (X_i \cap T)$. If the set C does not cover S, then the partition Π must "reduces" S in the following sense: either S is split into different G_i so that $|S \cap N[X_i]| \leq c - 1$ for all i or there is i such that $S \subseteq N[X_i]$ and $|S - T_i| \leq \mu_G^T(A, B) - 1$. In other words, S either becomes a smaller cut in G_i or S has the same size in one smaller graph G_i but contains fewer non-terminal vertices in some G_i with the new terminal set T_i.

The above discussion suggests a recursive algorithm for constructing a (T, c)-covering set from a (T, c)-reducing-or-covering (Π, C) where the algorithm recurses on each G_i such that, for every $A, B \subseteq T$, some (A, B)-weak separator is "reduced" in G_i in the above sense. The correctness of this recursion scheme is captured by Lemma 6.4 below. To state it, we need to define a notion of (T, c, c_T)-*covering set*.

Definition 6.3. Given a graph $G = (V, E)$ with terminal set T and parameter $c > 0$, a vertex set $Z \subseteq V$ is (T, c, c_T)-*covering* if for all $A, B \subseteq T$ such that $\mu^T(A, B) \leq c_T$ and $\mu(A, B) \leq c$, Z covers some min (A, B)-weak separator. We say that Z is (T, c)-*covering* if it is (T, c, c)-covering.

Observe that if $c_T > c$, then an empty set is (T, c, c_T)-covering. Also, the terminal set T is a $(T, c, 0)$-covering set. Another trivial (T, c)-covering set is V. The structural lemma below is the key for our recursive algorithm.

Lemma 6.4. *Given a graph G with terminal set T and $c \geq c_T > 0$, let $(\Pi = \{Z, X_1, \dots, X_\ell\}, C)$ be a (T, c)-reducing-or-covering partition-set pair of G. For each $i \in [\ell]$, define $G_i = G[N[X_i]]$, $T_i = (T \cap X_i) \cup N_G(X_i)$ and let Y_i be a $(T_i, c-1, c_T)$-covering set in G_i, $\bar{Y}_i$ be a $(T_i, c, c_T - 1)$-covering set in G_i. Then, $Z \cup C \cup \bigcup_{i \leq \ell}(Y_i \cup \bar{Y}_i) \cup T$ is (T, c, c_T)-covering set in G.*

In Section 6.1, we will show the recursive algorithm based on Lemma 6.4 for constructing a (T, c)-covering set from the subroutine from Theorem 6.2 for constructing a (T, c)-reducing-or-covering partition-set pair. This would prove Theorem 5.2, the main theorem of this section. Then, we will present the proof of Lemma 6.4 in Section 6.2.

6.1 Proof of Theorem 5.2

Algorithm. We describe the algorithm for computing (T, c, c_T)-covering set in G in Algorithm 4. Let ϕ be a parameter to be selected.

Algorithm 4: CoveringSet(G, T, c, c_T, ϕ)

Input: A graph $G = (V, E)$, a terminal set $T \subseteq V$ and parameters $c, c_T, \phi \in (0, 1)$
Output: A (T, c, c_T)-covering set in G.

if $c_T > c$ or $c = 0$ **then return** $\{\}$.
if $c_T = 0$ **then return** T.
Apply Theorem 2.3 on G to obtain a (V, c)-sparsifier for G with arboricity c.
Let $(\{Z, X_1, \dots, X_\ell\}, C)$ be the (T, c)-reducing-or-covering partition-set pair of G obtained by applying Theorem 6.2 using G, T, c, ϕ as inputs.
for $i \in [\ell]$ **do**
  $T_i \leftarrow (T \cap X_i) \cup N_G(X_i)$
  $Y_i \leftarrow$ CoveringSet$(G[N_G[X_i]], T_i, c - 1, c_T, \phi)$
  $\bar{Y}_i \leftarrow$ CoveringSet$(G[N_G[X_i]], T_i, c, c_T - 1, \phi)$
return $Z' = Z \cup C \cup \bigcup_{i \leq \ell}(Y_i \cup \bar{Y}_i) \cup T$.

Correctness. We prove that the output Z' is a (T, c, c_T)-covering set for G by induction on c and c_T. For the base case, if $c_T = 0$, then the terminal set is a covering set by definition. If $c = 0$ or $c_T > c$, then an emptyset is a covering set by definition. Now, suppose Y_i is a $(T_i, c - 1, c_T)$-covering set for $G_i = G[N[X_i]]$, and $\bar{Y}_i$ is a $(T_i, c, c_T - 1)$-covering set for G_i. Combining with the fact that $(\{Z, X_1, \dots, X_\ell\}, C)$ is a (T, c)-reducing-or-covering set of G, Lemma 6.4 implies that Z' is a (T, c, c_T)-covering set in G.

Size. We next bound the size of Z'. Let τ be the constant in the exponent of $2^{O(c^2)}$, and $p(n)$ be a subpolynomial factor in Theorem 6.2, respectively. Let $s(n, k, c, c_T)$ be the size of output of Algorithm 4 where n is the number of vertices, k is the number of terminals, and c, c_T are the parameters as inputs of Algorithm 4. For each $i \leq \ell$, denote $n_i = |N[X_i]|$ and $k_i = |T_i|$.

By definitions, we have $s(n,k,c,0) = k$ and $s(n,k,c,c_T) = 0$ if $c < c_T$ or $c = 0$. Otherwise, by Theorem 6.2, $s(n,k,c,c_T)$ satisfies the following recurrence relation.

$$s(n,k,c,c_T) \le (k + \phi n \cdot p(n)) \cdot 2^{\tau c^2} + \sum_{i \le \ell} s(n_i, k_i, c-1, c_T) + \sum_{i \le \ell} s(n_i, k_i, c, c_T - 1), \tag{1}$$

where

$$\sum_{i \le \ell} k_i \le (k + n\phi \cdot p(n))c^2 \text{ and } \sum_{i \le \ell} n_i \le n + \sum_{i \le \ell} k_i. \tag{2}$$

By careful inductive arguments, we prove the following claim in Appendix A.1 that

Claim 6.5. $s(n,k,c,c_T) \le (k + n\phi \cdot p(n))(4 + c + c_T)^{3(c+c_T)} \cdot (1 + c_T) \cdot 2^{(\tau c^2 + c_T)}$.

When $c_T = c$, Claim 6.5 implies that $s(n,k,c,c) \le k \cdot 2^{O(c^2)} + n\phi \cdot p(n) \cdot g(c)$ for some function of c. Therefore, by setting $\phi = (10 \cdot p(n) \cdot g(c))^{-1}$, we have that $s(n,k,c,c) \le k \cdot 2^{O(c^2)} + n/10$. Now, we repeat the same algorithm for $O(\log n)$ time, we obtain the final covering set of size $O(k \cdot 2^{O(c^2)})$.

Running Time. By Line 3, we can assume that $m \le nc$ and that G has arboricity c. That is, for all $S \subseteq V$, we have $|E_G(S,S)| \le c|S|$. We next bound the total size of subproblems. Let m be the size of the input graph. For all i, let m_i be the size of $G[N[X_i]]$. Denote $\tilde{X}_i = N[X_i]$, we have $m_i = |E_G(\tilde{X}_i, \tilde{X}_i)| \le c|N_G[X_i]|$. Therefore,

$$\sum_i m_i \le c \sum_i |N[X_i]| \le c(\sum_i |X_i| + \sum_i |N(X_i)|) \le cn + cn^{1+o(1)}\phi. \tag{3}$$

Next, we establish the recurrence relation of the running time. By Theorem 6.2, it takes $\tilde{O}(m^{1+o(1)}\phi^{-4}2^{O(c^2)})$ time to compute a (T,c)-reducing-or-covering set system. Let $f(m,\phi,c,c_T)$ be the running time of Algorithm 4. We have that $f(n,k,c,c_T)$ satisfies the following recurrence relation.

$$f(m,k,c,c_T) \le m \cdot p(m)\phi^{-4}2^{\tau c^2} + \sum_i f(m_i, \phi, c-1, c_T) + \sum_i f(m_i, \phi, c, c_T - 1), \tag{4}$$

where $p(m)$ is a subpolynomial factor, $\sum_i m_i \le m(1 + \phi p(m))$, and the base cases take linear time. By the choice of ϕ, and the similar inductive arguments, we can show that $f(n,k,c,c_T) = \tilde{O}(m^{1+o(1)}2^{O(c^2)})$.

6.2 Proof of Lemma 6.4

This section is devoted to proving Lemma 6.4. For all $A, B \subseteq T$ such that $\mu(A,B) \le c, \mu^T(A,B) \le c_T$, if C covers some (A,B)-min weak separator, then we are done. Otherwise, Theorem 6.2 implies that there is a min (A,B)-weak separator S in G such that Π T-hits S or Π splits S. For $i \in [\ell]$, define $S_i = S \cap N[X_i], A_i = A \cap N[X_i]$ and $B_i = B \cap N[X_i]$. We say that S is (T,c,c_T)-*small* in component i if (1) $|S_i| \le c-1$, or (2) $|S_i| = c$ but $|(S \cap X_i) - T| \le c_T - 1$. By definition, S is (T,c,c_T)-small for every component i.

Intuitively, for all i, we should expect S_i to be covered by either Y_i or $\bar{Y}_i$ since Y_i and $\bar{Y}_i$ are $(T,c-1,c_T)$-covering and (T,c,c_T-1)-covering in G_i, respectively. However, Y_i or $\bar{Y}_i$ may cover a different set $S'_i \ne S_i$ that has the same key properties as S_i for our purpose. We show that we can combine S'_i from each component to obtain a single cut S' such that S' is covered by Y_i and $\bar{Y}_i$, and at the same time S' is a min (A,B)-weak separator in G. We next formalize the intuition. We start with the following lemma.

Lemma 6.6 (Swapping Lemma). *If S is a min (A,B)-separator in G, and S is (T,c,c_T)-small in component i, then there exists a separator S'_i in G_i that is covered by $Y_i \cup \bar{Y}_i$ and $(S - S_i) \cup S'_i$ is a min (A,B)-separator in G.*

We defer the proof of Lemma 6.6 to the end of this section. Our proof strategy is to apply Lemma 6.6 for each component. We are now ready to prove Lemma 6.4.

Proof of Lemma 6.4. Let S be the min (A,B)-weak separator in G such that S is (T,c,c_T)-small for every component i (as discussed above). We assume WLOG that S is not covered by $Z \cup C \cup \bigcup_{i\le\ell}(Y_i \cup \bar{Y}_i) \cup T$. We reorder the indices so that S_1 is S_i such that $|S \cap X_i - T|$ is maximized over all i. Observe that $|S \cap X_1 - T| > 0$ because otherwise S must have been covered by $Z \cup \bigcup_{i\le\ell}(Y_i \cup \bar{Y}_i) \cup T$. Also,

$$\text{for all } i > 1, |S \cap X_i - T| \le c_T - 1. \tag{5}$$

Observe that S is (T,c,c_T)-small in component 1. Since S is a min (A,B)-weak separator in G and S is (T,c,c_T)-small in component 1, Lemma 6.6 implies that there exists S'_1 that is covered by $Y_1 \cup \bar{Y}_1$ and $S \cup S'_1 - S_1$ is a min (A,B)-weak separator in G. Therefore, we update $S \leftarrow (S - S_1) \cup S'_1$. Observe that S'_1 is covered by $Y_1 \cup \bar{Y}_1$, and S is a min (A,B)-weak separator.

Next, we repeat the same process for $i = 2,\ldots,\ell$. That is, for each $i \in \{2,\ldots,\ell\}$, we set $S \leftarrow (S - S_i) \cup S'_i$ where S'_i be the separator as stated in Lemma 6.6. If such S'_i always exists, then at the end of the iteration S is a min (A,B)-cut that is covered by $Z \cup \bigcup_{i\le\ell}(Y_i \cup \bar{Y}_i)$ and we are done. It remains to verify that at the beginning of each iteration i, S is a min (A,B)-weak separator and S is (T,c,c_T)-small in component i. We use induction on the number of iterations. The first iteration where $i = 2$ follows from the above discussion. Now assume that at iteration i where $2 \le i \le \ell - 1$, S is a min (A,B)-weak separator and S is (T,c,c_T)-small in component i. Therefore, Lemma 6.6 implies that there exists S'_i that is covered by $Y_i \cup \bar{Y}_i$ and $(S - S_i) \cup S'_i$ is min (A,B)-weak separator. So, we update $S \leftarrow (S - S_i) \cup S'_i$. Observe that for all $j < i$, S_j can be different due to the change in the boundary vertices $\partial X_j := N_G(X_j)$, but S_j is still covered by $Y_j \cup \bar{Y}_j \cup Z$. It remains to show that S is (T,c,c_T)-small in component $i+1$. Since $\mu(A,B) \le c$, we have $|S_{i+1}| \le |S| = \mu(A,B) \le c$. Also, $|S \cap X_{i+1} - T| \le c_T - 1$ because of Equation (5) and the fact that the component X_{i+1} has not been touched yet.[4] Therefore, S is (T,c,c_T)-small in component $i+1$. This completes the proof.

□

It remains to prove Lemma 6.6.

Proof of Lemma 6.6. The lemma is trivial if $S_i = \emptyset$. We now assume $S_i \ne \emptyset$. Let $S_{-i} = S - S_i$. Denote $\partial X_i = N_G(X_i)$ as a set of boundary vertices, and we say that $v \in V_{G_i}$ is a boundary vertex if $v \in \partial X_i$. In G_i, we say that a boundary vertex $v \in \partial X_i$ is an A-*proxy* if $A - A_i \ne \emptyset$, and there is an $(A - A_i, v)$-path in $G - S_{-i}$ that does not use X_i. Similarly, v is a B-*proxy* if $B - B_i \ne \emptyset$ and there is an $(B - B_i, v)$-path in $G - S_{-i}$ that does not use X_i. Let ∂A_i be the set of A-proxy boundary vertices, and ∂B_i be the set of B-proxy boundary vertices.

Intuitively, we can think of ∂A_i as a set of proxy nodes that represents all paths originated from A outside the component X_i that could enter X_i in $G - S_{-i}$, and similarly for ∂B_i.

Claim 6.7. *$\partial A_i \cup A_i \ne \emptyset$ and $\partial B_i \cup B_i \ne \emptyset$ and S_i is $(\partial A_i \cup A_i, \partial B_i \cup B_i)$-weak separator in G_i.*

[4] This is the reason why we showed a separate argument for component 1, otherwise it is less convenient to show that Equation (5) holds for component $i+1$.

Proof. We first prove that $\partial A_i \cup A_i \neq \emptyset$ and $\partial B_i \cup B_i \neq \emptyset$. Since S is a min (A, B)-weak separator, there must be an (A, B)-path P in $G - S_{-i}$. Furthermore, every (A, B)-path in $G - S_{-i}$ must use S_i. By definition of $\partial A_i, \partial B_i$, P must use either A_i or ∂A_i, and B_i or ∂B_i in G_i.

We prove the next claim. Suppose S_i is not a $(\partial A_i \cup A_i, \partial B_i \cup B_i)$-weak separator in G_i. Let P be an $(\partial A_i \cup A_i, \partial B_i \cup B_i)$-path in $G_i - S_i$ starting with a vertex a and ending with a vertex b (it is possible that P is simply a vertex where $a = b$). We define two paths P_a, P_b as follows. If $a \in \partial A_i$, then P_a is an $(A - A_i, a)$-path in $G - S_{-i}$ that does not use X_i. Otherwise, P_a is the vertex a. If $b \in \partial B_i$, then P_b is an $(B - B_i, a)$-path in $G - S_{-i}$ that does not use X_i. Otherwise, P_b is the vertex b. Therefore, the path $P_a \to P \to P_b$ is an (A, B)-path in $G - S$, a contradiction. $\square$

Denote $A'_i = \partial A_i \cup A_i$ and $B'_i = \partial B_i \cup B_i$.

Proposition 6.8. *$\mu_{G_i}(A', B') \leq c - 1$ or $\mu_{G_i}(A', B') = c$ and $\mu^{T_i}_{G_i}(A', B') \leq c_T - 1$*

Proof. Since S is (T, c, c_T)-small in component i, we have $|S_i| \leq c - 1$ or $|S_i| = c$ but $|X_i \cap S - T| \leq c_T - 1$. By Claim 6.7, S is a min (A', B')-weak separator in G_i. Therefore, the result follows. $\square$

We remark that Proposition 6.8 is the key to motivate Definition 6.1 to make this lemma work. Since Y_i is $(T_i, c_T, c - 1)$-covering set in G_i, and $\bar{Y}_i$ is $(T_i, c_T - 1, c)$-covering set in G_i, there is a min (A', B')-separator S'_i in G_i that is covered by $Y_i \cup \bar{Y}_i$. It remains to prove the following.

Claim 6.9. *$(S - S_i) \cup S'_i$ is a min (A, B)-weak separator in G.*

Proof. By Claim 6.7, S_i is a (A', B')-weak separator in G_i. Since S'_i is a min (A', B')-weak separator in G_i, $|S'_i| \leq |S_i|$. Therefore, $|(S - S_i) \cup S'_i| \leq |S| = \mu(A, B)$. It remains to prove that $(S - S_i) \cup S'_i$ is an (A, B)-weak separator in G. Since S is a min (A, B)-weak separator in G, there is an (A, B)-path in $G - S_{-i}$. Furthermore, every (A, B)-path contains a subpath P' that uses A' and B' in $G[X_i]$. Since S'_i is an (A', B')-weak separator in G_i, P' must contain some vertex in S'_i. Therefore, $G - ((S - S_i) \cup S'_i)$ has no (A, B)-path. $\square$

$\square$

7 Computing a (T, c)-Reducing-or-Covering Partition-Set Pair

This section is devoted to proving Theorem 6.2. Given a graph G with terminal set T, we say that a separator *T-reduces* (or simply *reduces*) the non-terminal part of another separator S if it contains a non-terminal vertex of S. We also say that a separator *splits* another separator S if it is an (x, y)-separator in G for some $x, y \in S$. Let $\mathcal{S}$ be a set of separators in G. We say that $\mathcal{S}$ *T-reduces* (or simply *reduces*) the non-terminal part of a separator S if it contains a separator that reduces the non-terminal part of S. We also say that $\mathcal{S}$ *splits* a separator S if it contains a separator that splits S. If S is a min (A, B)-weak separator such that $|S - T| = \mu^T(A, B)$, we say that S is *T-brittle* or just *brittle* for short.

Definition 7.1. Given a graph G with terminal set T and $c > 0$, a separators-set pair $(\mathcal{S}, C \subseteq V)$, where $\mathcal{S}$ is a set of separators in G, is *(T, c)-reducing-or-covering* if for all $A, B \subseteq T$ if $\mu_G(A, B) \leq c$, one of the followings is true:

1. C covers some min (A, B)-weak separator,

2. $\mathcal{S}$ reduces the non-terminal part of some brittle min (A, B)-weak separator, or

3. $\mathcal{S}$ splits some brittle min (A, B)-weak separator.

Organization of Section 7. We first explain a divide and conquer lemmas for computing a (T, c)-reducing-or-covering separators-set pair in Section 7.1. In Section 7.2, we show that divide-and-conquer lemmas naturally induce an algorithm that computes a (T, c)-reducing-or-covering separators-set pair $(\mathcal{S}, C)$ such that total size of all separators in $\mathcal{S}$ is $O(|T|c^2)$ and $|C| = O(|T|2^{O(c^2)})$. In Section 7.3, we explain how to modify the algorithm of Section 7.2 to output a (T, c)-reducing-or-covering partition-set pair with the properties as described in Theorem 6.2. In Section 7.4, we explain how to implement the algorithm in Section 7.3 in $\tilde{O}(|T|^3 mc + |T|m2^{O(c^2)})$ time, which is fast when $|T| \ll n$. In Section 7.5, we show that if the input graph is a ϕ-vertex expander, then the algorithm in Section 7.3 can be implemented in $\tilde{O}(mc\phi^{-1} + \phi^{-5}|T|2^{O(c^2)})$; we will use the algorithm in Section 7.4 as a subroutine. We explain the implementation details in Section 7.6. Finally, in Section 7.7, we apply vertex expander decomposition as preprocessing and apply the algorithm in Section 7.5 on each expander in the decomposition to compute the desired (T, c)-reducing-or-covering partition-set pair with the properties as described in Theorem 6.2 in $\tilde{O}(m^{1+o(1)}\phi^{-4} \cdot 2^{O(c^2)})$ time.

7.1 Divide-and-Conquer Lemmas

We prove divide-and-conquer lemma in this section. We recall the definition of Steiner cuts: A *Steiner cut* (L, S, R) is a vertex cut such that $L \cap T \neq \emptyset$ and $R \cap T \neq \emptyset$. It is minimum when $|S|$ is minimized. Before stating the recursion lemma, we introduce the notion of *left* and *right* graphs of G.

Definition 7.2 (Left and Right Graphs)**.** Let G be a graph with terminal set T and (L, S, R) be a min Steiner cut. We define *left graph* G_L of G and *right graph* G_R of G as follows. We define G_L as G after adding clique edges to the neighbors of $\hat{R} = R \cup (S - T)$, followed by contracting[5] $\hat{R}$ into an arbitrary vertex $t_R \in R \cap T$. Symmetrically, we define G_R as G after adding clique edges to the neighbors of $\hat{L} = L \cup (S - T)$, followed by contracting $\hat{L}$ into an arbitrary vertex $t_L \in L \cap T$. Let $T_L = (T \cap (L \cup S)) \cup \{t_R\}$ be the *left terminal set*, and $T_R = (T \cap (S \cup R)) \cup \{t_L\}$ be the *right terminal set*. We define *strictly left graph* $\check{G}_L$ as $\mathrm{cl}(G_L, t_R)$ and *strictly right graph* $\check{G}_R$ as $\mathrm{cl}(G_R, t_L)$, and *strictly left terminal set* $\check{T}_L = T \cap (L \cup S)$ and *strictly right terminal set* $\check{T}_R = T \cap (S \cup R)$.

The divide-and-conquer lemma is the following.

Lemma 7.3 (Vertex Closure Recursion Lemma)**.** *Let G be a graph and T be a terminal set. Let (L, S, R) be min Steiner cut. Define G_L, T_L, G_R, T_R as in Definition 7.2. If $(\mathcal{S}_L, C_L)$ is a (T_L, c)-reducing-or-covering pair in G_L, and $(\mathcal{S}_R, C_R)$ is a (T_R, c)-reducing-or-covering pair in G_R, then, $(\mathcal{S}_L \cup \{S\} \cup \mathcal{S}_R, C_L \cup C_R)$ is a (T, c)-reducing-or-covering pair in G.*

By using a similar argument in the proof of Lemma 7.3, we also have the following.

Lemma 7.4. *Let G be a graph and T be a terminal set. Let (L, S, R) be min Steiner cut. Define strictly left graphs, strictly left terminal sets, strictly right graphs and strictly right terminal sets $\check{G}_L, \check{T}_L, \check{G}_R, \check{T}_R$ as in Definition 7.2. Let $(\check{\mathcal{S}}_L, \check{C}_L)$ be a $(\check{T}_L, c)$-reducing-or-covering pair in $\hat{G}_L$, and $(\check{\mathcal{S}}_R, \check{C}_R)$ be a $(\check{T}_R, c)$-reducing-or-covering pair in $\check{G}_R$, respectively. If $S \subseteq T$, then $(\check{\mathcal{S}}_L \cup \{S\} \cup \check{\mathcal{S}}_R, \check{C}_L \cup \check{C}_R)$ is a (T, c)-reducing-or-covering pair in G.*

The rest of this section is devoted to proving Lemma 7.3.

Proof of Lemma 7.3. We show that for every $A, B \subseteq T$ such that $\mu_G(A, B) \leq c$ either $C_L \cup C_R$ covers some min (A, B)-weak separator or $\mathcal{S}_L \cup \{S\} \cup \mathcal{S}_R$ splits or reduces non-terminal part of some min brittle (A, B)-weak separator. Let $A, B \subseteq T$ be arbitrary two subsets of terminal set such that $\mu_G(A, B) \leq c$. If one of the following conditions hold, then we are done.

[5] Equivalently, one can view it as replacing $\hat{R}$ with t_R and add an edge to t_R to every vertex in $N(\hat{R})$.

1. $C_L \cup C_R$ covers some min (A, B)-weak separator,

2. $\mathcal{S}_L \cup \{S\} \cup \mathcal{S}_R$ reduces non-terminal part of some min brittle (A, B)-weak separator,

3. S is an (A, B)-weak separator, or

4. S splits some min brittle (A, B)-weak separator.

Assuming none of the above holds, we prove that $\mathcal{S}_L$ or $\mathcal{S}_R$ splits some brittle min (A, B)-weak separator. Let S' be a brittle min (A, B)-weak separator, and (L', S', R') be a corresponding vertex cut where $A \subseteq L' \cup S'$ and $B \subseteq S' \cup R'$. Since S does not reduce the non-terminal part of S', we have $S' \cap S \subseteq T$. Since S does not split S', we have $S' \subseteq L \cup S$ or $S' \subseteq S \cup R$. WLOG, we assume $S' \subseteq L \cup S$. The case for $S' \subseteq S \cup R$ is similar. Since S is a Steiner cut, there is a terminal in R. Since $S' \subseteq L \cup S$, the terminal in R must be in either L' or R'. WLOG, we assume that it is in L'. That is, $T \cap L' \cap R \neq \emptyset$. We summarize our assumption for S'.

Assumption 7.5. *We have the following assumption for S'.*

1. $S' \subseteq L \cup S$*,*

2. $S' \cap S \subseteq T$*, and*

3. $T \cap L' \cap R \neq \emptyset$*.*

From the assumption, we can intuitively expect S' to be some separator in G_L with terminal set T_L and also expect that $\mathcal{S}_L$ will split S'. The goal now is to prove that $\mathcal{S}_L$ splits some brittle min (A, B)-weak separator in G. Note that it may not be S'. Define $A' := (A - R) \cup (S \cap S') \cup \{t_R\}$ and $B' := (B - R) \cup (S \cap S')$ where t_R is the unique vertex in $T_L \cap R$. Note that $A', B' \subseteq T_L$.

Claim 7.6. *Both $A - R$ and $B - R$ are non-empty.*

Proof. Suppose $A - R$ is empty. Since $S' \subseteq L \cup S$, we have $S' \cap R = \emptyset$, and thus $A \subseteq L' \cap R$ and $N_G(L' \cap R) \subseteq S$. By definition of S', we have $B \subseteq S' \cup R'$, and thus $B \cap (L' \cap R) = \emptyset$. Therefore, S is an (A, B)-weak separator, a contradiction. The argument for $B - R$ is similar. □

Claim 7.7. *Any (A', B')-weak separator in G_L is an (A, B)-weak separator in G. Hence, $\mu_{G_L}(A', B') \geq \mu_G(A, B)$.*

Proof. We first set up notations. For any path P in G and a vertex set $X \subseteq V$, we denote $\mathrm{cl}(G, P, X)$ to be the path P after applying vertex closure operations on X in G. By definition of G_L, before replacing $\hat{R}$ with t_R we add clique edges to the neighbors of $\hat{R}$. Therefore, any path in P in G becomes $\mathrm{cl}(G, P, \hat{R})$ in G_L.

Let S'' be any (A', B')-weak separator in G_L. Suppose there is an (A, B)-path P in $G - S''$. Our goal is to show that $\mathrm{cl}(G, P, \hat{R})$ is an (A', B')-path in $G_L - S''$, which is a contradiction. Let $a \in A$, and $b \in B$ be the starting and ending vertices of path P, respectively. We consider three cases.

1. The first case is $b \in B \cap R$. By Assumption 7.5(3), $T \cap L' \cap R$ and $T \cap R' \cap R$ are both non-empty. Since (L, S, R) is a min Steiner cut, we have $L' \cap S = R' \cap S = \emptyset$, and thus P must contain a vertex in $S \cap S' \subseteq T$. Therefore, $\mathrm{cl}(G, P, \hat{R})$ contains a vertex in $S \cap S'$ in $G_L - S''$, a contradiction.

2. The second case is $a \in A - R, b \in B - R$. In this case, $a \notin \hat{R}$ and $b \notin \hat{R}$ since $a, b \in T$ and $\hat{R}$ contains only vertices in R and $S - T$. Therefore, $\mathrm{cl}(G, P, \hat{R})$ starts with a and ends with b, and thus it is an (A', B')-path in $G_L - S''$, a contradiction.

3. The final case is $a \in A \cap R, b \in B - R$. In this case, $a \in \hat{R}$ and $b \notin \hat{R}$, and thus $\mathrm{cl}(G, P, \hat{R})$ starts with a vertex in $N_G(\hat{R})$ and ends at b. By definition of G_L, there is an edge from t_R to every vertex in $N_G(\hat{R})$. Therefore, the path obtained by appending t_R with $\mathrm{cl}(G, P, \hat{R})$ is an $(t_R, B - R)$-path in $G_L - S''$, a contradiction.

□

Claim 7.8. *S' is a min (A', B')-weak separator in G_L. Hence, $\mu_{G_L}(A', B') \le |S'| = \mu_G(A, B) \le c$.*

Proof. We claim that $N_G(\hat{R}) \subseteq (L \cup S) \cap (L' \cup S')$. If true, then S' is an (A', B')-weak separator in G_L (since $B - R$ and $A - R$ are non-empty by Claim 7.6), and thus $\mu_{G_L}(A', B') \le |S'| = \mu_G(A, B)$. We now prove the claim. By Assumption 7.5(3), $R \cap L' \cap T \neq \emptyset$. Since (L, S, R) is a min Steiner cut in G, we have $S \cap R' = \emptyset$. By Assumption 7.5(2), $S' \subseteq L \cup S$, and thus we have $S' \cap R = \emptyset$. By Assumption 7.5(1), $S \cap S' \subseteq T$. Since $\hat{R} = R \cup (S - T)$, the claim follows.

Next, we prove that S' is a min (A', B')-weak separator in G_L. Let S'' be a min (A', B')-weak separator in G_L. By Claim 7.7, we have S'' is also an (A, B)-weak separator in G, and thus $\mu_G(A, B) \le |S''| = \mu_{G_L}(A', B')$. Therefore, $\mu_G(A, B) = \mu_{G_L}(A', B') = |S'|$. So, S' is a min (A', B')-weak separator in G_L. □

Claim 7.9. *Any T_L-brittle min (A', B')-weak separator in G_L is a T-brittle min (A, B)-weak separator in G.*

Proof. Let S'' be a min T_L-brittle min (A', B')-weak separator in G_L. We first show that S'' is a min (A, B)-weak separator in G. By Claim 7.7, S'' is an (A, B)-weak separator in G. So, $|S''| \ge \mu_G(A, B)$. By Claim 7.8, $|S''| = \mu_{G_L}(A', B') \le \mu_G(A, B)$. Therefore, $|S''| = \mu_G(A, B)$. Next, we show that $|S'' - T| = \mu^T_G(A, B)$. If true, then S'' is a T-brittle min (A, B)-weak separator in G. Since S' is a T-brittle weak (A, B)-weak separator in G, $|S' - T| = \mu_G(A, B)$. Since S'' is a T_L-brittle weak (A', B')-weak separator in G_L, $|S'' - T_L| = \mu_{G_L}(A', B')$.

To do so, we show that $S' - T_L = S' - T$ and $S'' - T_L = S'' - T$. Indeed, since $S' \subseteq L \cup S$ and $S' \cap S \subseteq T$, we have that S' does not contain any terminal in $T \cap R$, and thus $S' - T_L = S' - T$. Since $S'' \subseteq V(G_L) = L \cup (S - T) \cup \{t_R\}$ where t_R is the unique terminal in $T \cap R$ and additional terminals from T_L are all in $R - \{t_R\}$, $S'' - T_L = S'' - T$.

It remains to prove that $\mu^T_{G_L}(A', B') = \mu^T_G(A, B)$. If true, then we have

$$|S'' - T| = |S'' - T_L| = \mu^T_{G_L}(A', B') = \mu^T_G(A, B).$$

Since S' is a min (A', B')-weak separator in G_L (Claim 7.8), we have

$$\mu^T_{G_L}(A', B') \le |S' - T_L| = |S' - T| = \mu^T_G(A, B).$$

Since S'' is a min (A, B)-separator in G, we have

$$\mu^T_{G_L}(A', B') = |S'' - T_L| = |S'' - T| \ge \mu^T_G(A, B).$$

□

Claim 7.10. *$\mathcal{S}_L$ splits some T_L-brittle min (A', B')-weak separator in G_L*

Proof. The following facts imply that $\mathcal{S}_L$ must split some T_L-brittle min (A', B')-weak separator in G_L as desired.

- $(\mathcal{S}_L, C_L)$ is a (T_L, c)-reducing-or-covering pair in G_L,

- $\mu_{G_L}(A', B') \leq c$,
- C_L does not cover any min (A', B')-weak separator in G_L,
- $\mathcal{S}_L$ does not reduce non-terminal part of any min T_L-brittle (A', B')-weak separator in G_L.

The first item follows from the definition. We next show the second item. By Claim 7.8, we have $\mu_{G_L}(A', B') = |S'| \leq c$. We prove the third item. Suppose C_L covers a min (A', B')-weak separator Y in G_L. By Claim 7.7, Y is also a min (A, B)-weak separator in G, contradicting that C_L does not cover any min (A, B)-weak separator in G. Finally, the last item follows from Claim 7.9. □

Claim 7.10 and Claim 7.9 imply that $\mathcal{S}_L$ splits some T-brittle min (A, B)-weak separator in G as desired. This completes the proof of Lemma 7.3. □

7.2 Computing a (T, c)-Reducing-or-Covering Separators-Set Pair

Lemma 7.3 naturally gives us a recursive algorithm for computing (T, c)-reducing-or-covering pair as described in Algorithm 5.

Algorithm. Let G be the input graph and T be its terminal set. If T is small enough, we can output $(\emptyset, C)$ where C is the union of a small min (A, B)-weak separator for each $A, B \subseteq T$. Let (L, S, R) be the min Steiner cut. If it does not exist or $|S| \geq c+1$, we are done. We say that (L, S, R) is *balanced* if $\min\{|L \cap T|, |R \cap T|\} \geq 4c^2$, *isolating* if $\min\{|L \cap T|, |R \cap T|\} = 1$. Otherwise, (L, S, R) is *unbalanced non-isolating*. Given a terminal t, (L, S, R) is *t-isolating* if $t \in L$ and $T - \{t\} \subseteq S \cup R$. For each terminal $t \in T$, define $\text{isolate}_G(t)$ to be the size of minimum t-isolating separator in G

Definition 7.11. A vertex cut (L, S, R) is *good* t-isolating if it is a t-isolating cut min Steiner cut and $\text{isolate}_{G_R}(t) > \text{isolate}_G(t)$ or $S \subseteq T$.

For the general case, we apply Lemma 7.3. For efficiency point of view, we will apply Lemma 7.4

if $S \subseteq T$ and (L,S,R) is t-isolating. The full algorithm is described in Algorithm 5.

Algorithm 5: ROC(G,T,c)

Input: A graph $G=(V,E)$, a terminal set $T \subseteq V$ and parameters $c > 0$
Output: A (T,c)-reducing-or-covering separators-set pair in G.

if $|T| \leq 4c^2$ **then**
  Let C be the union of a min-(A,B)-weak separator for all $A,B \subseteq T$ such that $\mu_G(A,B) \leq c$.
  return $(\{\}, C)$
if $G[T]$ is a complete graph **then return** $(\{\}, T)$.
Let (L,S,R) be a min Steiner cut.
if $|S| \geq c+1$ **then return** $(\{\}, T)$.
if (L,S,R) is a t-isolating cut **then**
  Replace (L,S,R) with a good t-isolating cut.
Let G_L, G_R, T_L, T_R be the left and right graphs, the terminal sets as defined in Definition 7.2
if $S \subseteq T$ and (L,S,R) is t-isolating **then**
  Let G_L, G_R, T_L, T_R be the strictly left and right graphs, the strict terminal sets as defined in Definition 7.2
$(\mathcal{S}_L, C_L) \leftarrow$ ROC(G_L, T_L, c)
$(\mathcal{S}_R, C_R) \leftarrow$ ROC(G_R, T_R, c)
return $(\mathcal{S}_L \cup \{S\} \cup \mathcal{S}_R, C_L \cup C_R)$

Analysis. We describe the recursion tree of Algorithm 5 and its structure. Each internal node in the recursion tree can be represented as (G,T) and its left and its right children are (G_L,T_L) and (G_R,T_R) respectively where G_L and G_R are left and right graphs of G with respect to the min Steiner cut (L,S,R) found in the current subproblem (G,T). If $S \subseteq T$ and (L,S,R) is t-isolating, then we further close the unique terminal t_R in G_L and another unique terminal t_L in G_R to obtain strictly left and strictly right graphs respectively.

We state the key properties of this algorithm into two lemmas.

Lemma 7.12. *Given a graph G with terminal set T and a parameter $c > 0$, then over all the executions of Algorithm 5, the number of balanced min Steiner cuts, isolating, and unbalanced non-isolating are at most $O(|T|/c), O(|T|c)$, and $O(|T|c)$, respectively.*

Lemma 7.13. *Algorithm 5 returns a (T,c)-reducing-or-covering separators-set pair $(\mathcal{S}, C)$ such that $|\bigcup_{S\in\mathcal{S}} S| = O(|T|c^2)$ and $|C| = O(|T|2^{O(c^2)})$.*

We explain how to obtain (T,c)-reducing-or-covering partition-set pair from Algorithm 5 in Section 7.3. We will describe fast implementation when the number of terminal is small in Section 7.4 and when the input graph is a vertex expander in Section 7.5. The final algorithm for computing (T,c)-reducing-or-covering partition-set pair is described in Section 7.7. The rest of the section is devoted to proving Lemma 7.12 and Lemma 7.13. We first start with useful properties of left and right graphs.

Proposition 7.14 (Monotonicity of Left and Right Graphs)**.** *Every separator in $G_L, G_R, \check{G}_L, \check{G}_R$ (Definition 7.2) is a separator in G.*

Proof. We prove the results for the left graph G_L. The proof for G_R is similar. Let t_R be an arbitrary terminal in $R \cap T$. Since (L,S,R) is a min Steiner cut in G, there must be a path from

t_R to every vertex in S in $G[S \cup R]$ (otherwise, a subset of S is a Steiner cut, contradicting the minimality of S). Let $\tilde{R} = \hat{R} - \{t_R\}$. Since there is a path from t_R to every vertex in S without using L, we have $\text{cl}(G, \tilde{R}) \subseteq G_L$. Next, we prove that every separator in G_L is a separator in G. Let S' be a separator in G_L. Since $\text{cl}(G, \tilde{R}) \subseteq G_L$, S' is also a separator in $\text{cl}(G, \tilde{R})$. By monotonicity of vertex closure operation (Lemma 5.5), S' is also a separator in G. $\square$

We are now ready to prove Lemma 7.12.

Proof of Lemma 7.12. We define the potential function that we use to keep track of progress of the recursion tree. For each terminal $t \in T$, let $\psi_G(t) = \max\{0, c+1-\text{isolate}_G(t)\}$ if G is not in the base case in the algorithm (i.e., it does not terminate at line 1, 2 and 4 of the algorithm), and $\psi_G(t) = 0$ otherwise. Define $\psi_G(T) = \sum_{t\in T} \psi_G(t)$. Let $\pi_G(T) = \max\{0, |T| - 4c^2\}$. At any time, we have a collection of graphs and their terminal set $\mathcal{G} = \{(G,T)\}$ at leaf nodes (that are not necessarily the base case at the moment) in the recursion tree. Initially, we have only one node which is the input graph and its terminal set. We keep track of the progress via the following potential function

$$\Pi(\mathcal{G}) = \sum_{(G,T)\in\mathcal{G}} \pi_G(T) + \psi_G(T).$$

Initially, $\mathcal{G}$ contains only the input graph and its terminal set. Thus, $\Pi(\mathcal{G}) \leq |T| + |T|c = O(|T|c)$. At any time, $\Pi(\mathcal{G}) \geq 0$. At any time, the recursion tree changes at a leaf node that is not a base. Suppose the leaf node represents a graph G with k terminals where $k \geq 4c^2$. There are three cases.

Case 1: Balanced. We prove that the number of balanced min Steiner cut is at most $O(|T_{\text{input}}|/c)$ where T_{input} is the terminal set at the root of the recursion tree. In this case, (G,T) creates two subproblems (G_L, T_L) and (G_R, T_R) with k_1, and k_2 terminals respectively where $\min\{k_1, k_2\} \geq 4c^2$. By definition, the only duplicate terminals happens at the min Steiner cut S where $|S| \leq c$, and thus we have $k_1 + k_2 \leq k + c$. The change in the potential $\Pi(G)$ happens at the term $\pi_G(T) + \psi_G(T)$ and its two new subproblems $\pi_{G_L}(T_L), \pi_{G_R}(T_R), \psi_{G_L}(T_L), \psi_{G_R}(T_R)$. That is the change due to π is

$$\pi_G(T) - \pi_{G_R}(T_R) - \pi_{G_L}(T_L) = (k - 4c^2) - ((k_1 - 4c^2) + (k_2 - 4c^2)) \geq -c + 4c^2 \geq 3c^2,$$

and the change due to ψ is

$$\psi_G(T) - \psi_{G_L}(T_L) - \psi_{G_R}(T_R) \geq (k - k_1 - k_2 - 2)c \geq -2c^2.$$

Intuitively, ψ increases because there are terminals in $S \cap T$ that are replicated. Each terminal can have ψ value by at most c. Since $|S \cap T| \leq c$, the total increases due to ψ is at most $O(c^2)$. Therefore, the net drop in potential is at least $3c^2 - 2c^2 = c^2$. Since the initial potential is at most $O(|T_{\text{input}}|c)$, the number of balanced min Steiner cuts is at most $O(|T_{\text{input}}|/c)$.

Case 2: Isolating. In this case, we obtain a min Steiner cut (L, S, R) that is good t-isolating for some t and $|S| \leq c$. WLOG, $L \cap T = \{t\}$. Thus, G_L can have at most $c+2$ terminals. So, G_L becomes a base case and the thus the potential for G_L is zero. Hence, the change of the potential function due to π is

$$\pi(G,T) - \pi(G_L, T_L) - \pi(G_R, T_R) = 0.$$

That is, there is no change in the potential due to π. Next, we bound the change of potential due to ψ. Observe that G_R contains the same number of terminals, and $t_L = t$ where t_L is the unique terminal in $T \cap L$ in G_R. For each of terminal $t' \in T_R - \{t_L\}$, we have $\text{isolate}_{G_R}(t') \geq \text{isolate}_G(t')$ because of monotonicity of left and right graphs of G (Proposition 7.14). It remains to bound the change of ψ on $t = t_L$. There are two cases:

- The first case is when $S \subseteq T$. In this case, t_L is closed because we recurse on strictly right graph $\check{G}_R$ of G. Therefore, the terminal t_L disappears in the remaining graph. Since $\text{isolate}_G(t_L) > 0$ and t_L disappears in G_R, the drop in potential due to ψ is at least 1.

- Otherwise, since (L, S, R) is a good t-isolating cut in G, and $S - T \neq \emptyset$, we have $\text{isolate}_{G_R}(t) > \text{isolate}_G(t)$ and the change due to ψ is

$$\psi_G(T) - \psi_{G_L}(T_L \cup \{t_R\}) - \psi_{G_R}(T_R \cap \{t_L\}) \geq 1.$$

Since the initial potential is at most $O(|T_{\text{input}}|c)$, the number of good t-isolating min Steiner cuts is at most $O(|T_{\text{input}}|c)$.

Case 3: Unbalanced Non-isolating. In this case, we obtain a min Steiner cut (L, S, R) such that $\min\{|L \cap T|, |R \cap T|\} \in [2, 4c^2 - 1]$. We assume WLOG $|L \cap T| \leq |R \cap T|$. So, the subproblem (G_L, T_L) becomes a base case, and thus $\pi_{G_L}(T_L) = \psi_{G_L}(T_L) = 0$. Observe that the number of terminals G_R is strictly smaller than that of G. So, $\pi_G(T) - \pi_{G_R}(T_R) \geq 1$. By Proposition 7.14 (monotonicity of left and right graphs of G), we have $\text{isolate}_{G_R}(t') \geq \text{isolate}_G(t')$ for all $t' \in T \cap (S \cup R)$. Furthermore, t_L corresponds to one of the terminal in $L \cap T$, and thus $\text{isolate}_{G_R}(t_L) \geq \text{isolate}_G(t)$ for any $t \in L \cap T$. Therefore, $\psi_G(T) - \psi_{G_R}(T_R) \geq 0$. Since the initial potential is at most $O(|T_{\text{input}}|c)$ where T_{input} is the original input terminal set, the number of unbalanced non-isolating min Steiner cuts is at most $O(|T_{\text{input}}|c)$. ☐

Next, observe that if the min Steiner cut is larger than c, then we are done.

Proposition 7.15. *Let (L, S, R) be a minimum Steiner cut. If $|S| \geq c + 1$, then the terminal set T is (T, c)-covering.*

Proof. Since min Steiner cut is at least $c + 1$, every (A, B)-weak separator of size at most c must be either A or B. Since $A \cup B \subseteq T$, we have that T is a (T, c)-covering set. ☐

We are now ready to prove Lemma 7.13.

Proof of Lemma 7.13. We use induction on the recursion tree from leaves to the root. We consider the base case (leaf nodes). If $|T| \leq 4c^2$, then clearly the union is of all min (A, B)-weak separator for all $A, B \subseteq T$ is (T, c)-covering. If min Steiner cut is at least $c + 1$ or $G[T]$, then Proposition 7.15 implies that T is (T, c)-covering set. For the internal nodes, it follows immediately from Lemma 7.3 if $S \not\subseteq T$, and by Lemma 7.4 otherwise.

Next, we bound the sizes of (T, c)-reducing-or-covering separators-set pair. The total size of all separators in $\mathcal{S}$ is bounded by the total size of min Steiner cuts in the internal nodes of the recursion tree. By Lemma 7.12, the number of internal nodes is $O(|T|c)$, and thus, the number of leaves is also at most $O(|T|c)$. Since each Steiner cut has size at most c, the total contribution of the size due to internal nodes is at most $O(|T|c^2)$. The size of C is bounded by the total size of cuts in the base cases. Since each leave represents a base case, the total contribution due to leaf nodes are at most $O(|T|c \cdot c \cdot 3^{4c^2}) = O(|T| \cdot 2^{O(c^2)})$. ☐

7.3 A Partition from the Recursion Tree

One possible way to obtain (T, c)-reducing-or-covering partition-set pair from a (T, c)-reducing-or-covering separators-set pair is as follows:

Observation 7.16. *Given a (T,c)-reducing-or-covering partition-set pair $(\mathcal{S},C)$, define Z to be the union of all separators in S, and $X_1,\ldots,X_\ell$ to be connected components of $G-Z$. Then, $(\{Z,X_1,\ldots,X_\ell\},C)$ is (T,c)-reducing-or-covering partition-set pair.*

Proof. Fix A,B such that $\mu(A,B)\le c$. If C covers some min (A,B)-weak separator, or $\mathcal{S}$ splits some brittle min (A,B)-weak separator, then we are done. We now assume otherwise. By Definition 7.1, there exists a brittle min (A,B)-weak separator denoted as S in G whose non-terminal part is reduced by $\mathcal{S}$. Since S is not split by $\mathcal{S}$, $S\subseteq N[X_i]$ for some i. Since S is reduced by $\mathcal{S}$, there is a non-terminal vertex in S that is in Z. Therefore, $|(S\cap X_i)-T|\le|S-T|-1\le\mu^T(A,B)-1$. □

However, the total neighbors $\sum_i|N(X_i)|$ in Observation 7.16 can be too large (because of overlapping neighbors), and thus the condition 3 of Theorem 6.2 does not hold. To handle this issue, we show that all the base cases in the recursion of the algorithm for computing (T,c)-reducing-or-covering separators-set pair (Section 7.2) correspond to components $X_1,\ldots,X_\ell$ whose total neighbors is small. We now make the statement precise.

Definition 7.17. Let G^{orig} be the original input graph at the root of the recursion tree of Algorithm 5. Given a recursion tree at the end of Algorithm 5, we define $X_1,\ldots X_\ell$ where ℓ is the number of leaf nodes in the recursion tree as follows. For each leaf node i, we start with $X_i=V(G^{\text{orig}})$. Then, we move along the path from root to leaf i and keep filtering the vertex set in the following sense. At the current node v, let (L_v,S_v,R_v) be a min Steiner cut obtained in the subproblem at v. If we go left, then we set $X_i\leftarrow X_i\cap L_v$. Otherwise, we set $X_i\leftarrow X_i\cap R_v$. We repeat until we reach the leaf node.

Lemma 7.18. *Let $(\mathcal{S},C)$ be a (T,c)-reducing-or-covering obtained from Algorithm 5 with G^{orig},T,c as inputs, and let $X_1,\ldots,X_\ell$ be vertex sets of G^{orig} according to Definition 7.17. Let $Z=\bigcup_{S\in\mathcal{S}}S$. Then, the partition-set pair $((Z,X_1,\ldots,X_\ell),C)$ is (T,c)-reducing-or-covering for G^{orig} where*

- $|Z|=O(|T|c^2)$,
- $|C|=O(|T|\cdot 2^{O(c^2)})$, *and*
- $\sum_i|N_{G^{orig}}(X_i)|=O(|T|c^2)$.

The rest of the section is devoted to proving Lemma 7.18. We will discuss efficient implementation in next subsections.

Analysis. Our goal is to establish the following claims, which imply Lemma 7.18.

Claim 7.19. *$(Z,X_1,\ldots,X_\ell)$ forms a partition of $V(G^{orig})$ such that Z is an (X_i,X_j) for all $i\neq j$.*

Claim 7.20. *The partition-set pair $((Z,X_1,\ldots,X_\ell),C)$ is an (T,c)-reducing-or-covering.*

Claim 7.21. $\sum_i|N_{G^{orig}}(X_i)|=O(|T|c^2)$, $|Z|=O(|T|c^2)$ *and* $|C|=O(|T|\cdot 2^{O(c^2)})$.

For the purpose of analysis, we also associate $X\subseteq V(G^{\text{orig}})$ to each node in the recursion tree. That is, the input to the subproblem is of the form (G,T,c,X). We call X *core set* of the subproblem. Recall that G,T,c are the same as in the input for Algorithm 5. We define X for each node in the recursion tree. Initially, $X=V(G^{\text{orig}})$ at the root node. At the current node with (G,T,c,X), we obtain a min Steiner cut (L,S,R) of G with terminal set T. According to Algorithm 5, we recurse on left graph G_L of G with terminal set T_L and right graph G_R of G with terminal set T_R (if (L,S,R) is a isolating Steiner cut and $S\subseteq T$, then we define G_L to be strictly left graph and G_R to be strictly right graph of G instead). In this analysis, we recurse left with (G_L, T_L, c, X_L) as inputs and right with (G_R, T_R, c, X_R) as inputs where $X_L := X \cap L$ and $X_R := X \cap R$. Observe that $X_L \subseteq V(G_L)$ and $X_R \subseteq V(G_R)$, respectively.

Lemma 7.22. *If (G, T, c, X) is the current node, and (L, S, R) is the min Steiner cut found in the algorithm, then S is an (L, R)-separator in G^{orig}.*

Proof. Observe that the graph G in the recursion tree is obtained by a sequence of transformations in Definition 7.2 starting from G^{orig}. By Proposition 7.14, S is also a separator in G^{orig}. Since L and R are on different sides of the separator S in the graph G, S must separate L and R in G^{orig} as well. □

We are now ready to prove each of the claims above.

Proof of Claim 7.19. We prove that $(Z, X_1, \ldots, X_\ell)$ is a partition of $V(G^{\text{orig}})$. We show that $X_i \cap X_j = \emptyset$ for all $i \neq j$. Suppose there are i and j such that $X_i \cap X_j \neq \emptyset$. Let $v \in X_i \cap X_j$. Let p be the LCA (longest common ancestor) node of leaf i and leaf j in the recursion tree. Let X_p be the core set of the subproblem at p. So, $v \in X_p$. Since p is not the leaf, we must obtain a min Steiner cut (L, S, R) at p. Since $v \in X_i$ and $v \in X_j$, we have $v \in L \cap R$, so $L \cap R \neq \emptyset$, a contradiction. Next, we prove that $Z \cap X_i = \emptyset$ for all i. By design, X_i is obtained by filtering from $V(G^{\text{orig}})$ with either left or right part of min Steiner cuts along the path from root to leaf. Therefore, X_i does not contain any vertex from min Steiner cuts.

It remains to prove that Z is an (X_i, X_j)-separator in G^{orig} for all $i \neq j$. Suppose there is an (X_i, X_j)-path P in $G^{\text{orig}} - Z$ for some i, j. Let (L^*, S^*, R^*) be the min Steiner cut obtained at the LCA p of leaf i and leaf j. Since $S^* \subseteq Z$, P is also an (X_i, X_j)-path in $G^{\text{orig}} - S^*$. Finally, we prove that S^* is an (X_i, X_j)-separator in G^{orig} which leads to a contradiction. By recursion tree and Definition 7.17, $X_i \subseteq L^*$ and $X_j \subseteq R^*$. By Lemma 7.22, S^* is (L^*, R^*)-separator in G^{orig}. Since $X_i \subseteq L^*$ and $X_j \subseteq R^*$, S^* is an (X_i, X_j)-separator in G^{orig}.

□

Proof of Claim 7.20. Fix $A, B \subseteq T$ such that $\mu_{G^{\text{orig}}}(A, B) \leq c$. If C covers some min (A, B)-separator or $\mathcal{S}$ reduces the non-terminal part of some brittle min (A, B)-weak separator, then we are done. Now, we assume that $\mathcal{S}$ splits some brittle min (A, B)-weak separator S'. We prove that $|S' \cap N_{G^{\text{orig}}}[X_i]| \leq |S'| - 1$ for all i. To do so, we show that $N_{G^{\text{orig}}}[X_i]$ does not cover S' for all i. Since $\mathcal{S}$ splits S', there could be many separators in $\mathcal{S}$ that splits S'. Let $S \in \mathcal{S}$ be the min Steiner cut (L, S, R) in one of the subproblem in the recursion tree where $x \in L$ and $y \in R$ (or symmetrically $y \in L$ and $x \in R$). Such an S exists by selecting the first level in the recursion tree that S' is split (if S' is not split at root and its non-terminal part is never reduced, then S' must be split at either left subtree or right subtree). Let $x \in N_{G^{\text{orig}}}[X_i]$ for some i. We prove that $y \notin N_{G^{\text{orig}}}[X_i]$. Indeed, since x and y are on different side of S in the recursion, we have $y \notin X_i$. By Lemma 7.22, S is an (L, R)-separator in G^{orig}. Since $X_i \subseteq L$ and $y \in R$, $y \notin N_{G^{\text{orig}}}(X_i)$. This completes the proof. □

Proof of Claim 7.21. The bounds for Z and C follow from Lemma 7.13. We next bound the sum of total neighbors. At any time, we have a collection of graphs and their terminal set $\mathcal{G} = \{(G, T, c, X)\}$ at leaf nodes (that are not necessarily the base case at the moment) in the recursion tree. Initially, we have only one node which is the input graph and its terminal set. We keep track of the progress via the following potential function

$$\Pi(\mathcal{G}) = \sum_{(G,T,c,X) \in \mathcal{G}} |N_{G^{\text{orig}}}(X)|.$$

Notice the neighbors of each set corresponds to those in the original input graph G^{orig}.

Initially, $\mathcal{G}$ contains only the input graph and its terminal set. Thus, $\Pi(\mathcal{G}) = 0$. At any time, the recursion tree changes at a leaf node that is not a base case. We claim that the potential can increase at most $4c$ whenever a min Steiner cut is obtained at any subproblem. If this is the case, then at the end of the recursion we have $\Pi(\mathcal{G}) = \sum_{(G,T,c,X)\in\mathcal{G}} |N_{G^{\text{orig}}}(X)| = \sum_{i\leq\ell} |N_{G^{\text{orig}}}(X_i)| = O(kc^2)$ (because the number of internal nodes is at most $O(kc)$ by Lemma 7.12).

It remains to prove the claim. We set up notations and make observation. At the current node with (G,T,c,X), suppose we obtain a min Steiner cut (L,S,R). The tree will create two children (G_L,T_L,c,X_L) and (G_R,T_R,c,X_R) as described above. Suppose $|N_{G^{\text{orig}}}(X)| = k_0$.

Claim 7.23. *$|N_{G^{orig}}(X_L)| + |N_{G^{orig}}(X_R)| \leq k_0 + 3c$.*

Proof. We first show that

$$N_{G^{\text{orig}}}(X_L) \subseteq N_{G^{\text{orig}}}(X) \cup S \text{ and } N_{G^{\text{orig}}}(X_R) \subseteq N_{G^{\text{orig}}}(X) \cup S. \tag{6}$$

Indeed, since $X_L = X \cap L \subseteq X$, we have $N_{G^{\text{orig}}}(X_L) \subseteq N_{G^{\text{orig}}}(X) \cup X$. Since (L,S,R) is a vertex cut in G and $X \subseteq V(G)$, we have $N_{G^{\text{orig}}}(X_L) \subseteq N_{G^{\text{orig}}}(X) \cup S$. The proof for $N_{G^{\text{orig}}}(X_R)$ is similar. Second, S is an (X_L,X_R)-separator in G^{orig} because Lemma 7.22 implies that S is (L,R)-separator in G^{orig} and $X_L \subseteq L$ and $X_R \subseteq R$.

Since S is an (X_L,X_R)-separator in G^{orig}, we have that for every vertex v in G^{orig} if $v \in N_{G^{\text{orig}}}(X_L) \cap N_{G^{\text{orig}}}(X_R)$, then $v \in S$. In particular, there are at most $|S| \leq c$ vertices in $N_{G^{\text{orig}}}(X)$ that can be both neighbors of X_L and X_R. By Equation (6) and the fact that at most $|S| \leq c$ of $N_{G^{\text{orig}}}(X)$ can be both neighbors of X_L and X_R, we conclude that $|N_{G^{\text{orig}}}(X_L)| + |N_{G^{\text{orig}}}(X_R)| \leq (|N_{G^{\text{orig}}}(X)| + |S|) + 2|S| = k_0 + 3|S| \leq k_0 + 3c$. ◻

Therefore, Claim 7.23 imply that the potential can increase at most $3c$, and we are done. ◻

7.4 Fast Implementation for Few Terminals

We describe the implementation of Algorithm 5 that is fast when the number of terminal is small (e.g., $|T| = \log^{O(1)}(n)$).

Lemma 7.24. *Algorithm 5 can be simulated by another algorithm that runs in $\tilde{O}(k^3mc + km2^{O(c^2)})$ time where m is the number of edges and k is the number of terminals of the input graph for Algorithm 5.*

We first describe the key challenge for fast implementation. Given G with n vertices and m edges, recall that G_L (G_R) is constructed by adding clique edges between neighbors of $\hat{R}$ ($\hat{L}$). The key challenge is that the size of the graph G_L and G_R can be $O(n^2)$ even if the G is sparse. Our approach is to simulate Algorithm 5 using hypergraphs instead. The intuition is that each clique in the graph can be viewed as a hyperedge in the corresponding hypergraph. That is, adding clique edges in G can be viewed as adding a hyperedge that is formed by the union of existing hyperedges. We next formalize the intuition.

Hypergraphs. We use bipartite representation of a hypergraph $H = (V_{\mathsf{v}}, V_{\mathsf{e}}, E)$ where V_{v} is a set of vertices, and V_{e} is a set of hyperedges, and the set of edges in bipartite graph E represents vertex and hyperedge incidence in H. The size of hypergraph H is $|E|$. Given an hyperedge $e \in V_{\mathsf{e}}$, we denote $N_H(e) \subseteq V_{\mathsf{v}}$ to be the set of vertices that are incident to e.

Definition 7.25. Given a hypergraph $H = (V_{\mathsf{v}}, V_{\mathsf{e}}, E)$, $\mathsf{Clique}(H)$ is a graph $G = (V_{\mathsf{v}}, E')$ where E' is obtained by adding clique edges of $N_H(e)$ for every hyperedge $e \in V_{\mathsf{e}}$. We say that a graph G *represents* a hypergraph H if $G = \mathsf{Clique}(H)$.

Observation 7.26. *Let H and G be a hypergraph and a graph such that $G = \mathsf{Clique}(H)$. Then, a vertex cut (L, S, R) in G is a vertex cut in H and vice versa.*

Proof. If (L, S, R) is a vertex cut in G, then there is no edge between L and R. Since $\mathsf{Clique}(H) = G$, every hyperedge cannot be incident to both L and R. Therefore, (L, S, R) is a vertex cut in H. If (L, S, R) is a vertex cut in H, then every hyperedge cannot be incident to both L and R. Since $\mathsf{Clique}(H) = G$, there is no edge that is incident to L and R in G, and thus (L, S, R) is a vertex cut in G. □

Proposition 7.27 (Line 9)**.** *Given a hypergraph H of size p such that $G = \mathsf{Clique}(H)$, a terminal set T, and a min Steiner cut (L, S, R), there is an algorithm that runs in $O(p)$ time and outputs two hypergraphs H_L and H_R such that $G_L = \mathsf{Clique}(H_L)$ and $G_R = \mathsf{Clique}(H_R)$, respectively where G_L and G_R are left and right graphs of G with respect to (L, S, R) (Definition 7.2). Furthermore, the size of each hypergraph is no more than that of H.*

Proof. We explain the construction of H_R. The construction of H_L is similar. Let $\hat{L} = L \cup (S - T)$. We view H as a bipartite graph, and construct H_L as follows. Given (L, S, R), we first compute $\hat{E} = N_H(\hat{L})$ the set of hyperedges incident to $\hat{L}$. Then, we compute $N_H(\hat{E})$ the set of vertices that are incident to $\hat{E}$. Then, we add a terminal t_L to V_{v}, and add a new hyperedge $\hat{e}$ to V_{e} where $\hat{e}$ is incident to every vertex in $\{t_L\} \cup N_H(\hat{E})$. Observe that $\hat{e}$ represents a clique between the neighbors of $\hat{L}$ in the original graph. Finally, we clean up by deleting all hyperedges in $\hat{E}$ and vertices in $\hat{L}$. By design, $G_R = \mathsf{Clique}(H_R)$, and we can implement in $O(p)$ time. □

We state three subroutines that implement Lines 2, 5 and 8 of Algorithm 5 using hypergraphs.

Proposition 7.28. *Let H be a hypergraph of size p and T be a set of k terminals. There are the following deterministic algorithms whose proofs are almost the same as the graph version (where we can view hypergraphs as bipartite graphs).*

1. *(Line 2) Given H, T, compute a min Steiner cut of size at most c or certifies that min Steiner cut is at least c in $O(k^2pc)$ time.*

2. *(Line 5) Given H and $A, B \subseteq T$, compute a minimum (A, B)-weak separator of size at most c, or certifies that $\mu(A, B) \geq c + 1$ in $O(pc)$ time.*

3. *(Line 8) Given H and $t \in T$, compute a maximal t-isolating cut of size at most c (therefore, a good t-isolating cut), or correctly report that the size of t-isolating cut is at least $c + 1$ in $O(pc)$ time.*

We are now ready to prove Lemma 7.24.

Proof of Lemma 7.24. We simulate Algorithm 5 by using a hypergraph H instead of a graph G as an input such that $G = \mathsf{Clique}(H)$. We apply Proposition 7.28 to implement Lines 2, 5 and 8 of Algorithm 5. Next, we describe how to implement Line 9. Given a min Steiner cut (L, S, R), we apply Proposition 7.27 to construct H_L and H_R such that $G_L = \mathsf{Clique}(H_L)$ and $G_R = \mathsf{Clique}(H_R)$, and recurse.

We now analyze the modified algorithm based on hypergraphs. Consider the recursion tree of the modified algorithm where we use hypergraphs instead of graphs as inputs. Each node in the recursion tree is slightly changed from (G, T) to (H, T) where H is a hypergraph and G. By induction on the depth of recursion tree and Proposition 7.27, we have that at any time $G = \mathsf{Clique}(H)$. Furthermore, the size of hypergraphs is at most m, and thus the time to construct H_L and H_R according to Proposition 7.27 is also $O(m)$. Since $G = \mathsf{Clique}(H)$ at any time, Lemma 7.12 implies that the number of balanced min Steiner cuts, good t-isolating cuts, and unbalanced non-isolating cuts (all in hypergraph versions) are at most $O(k/c), O(kc)$ and $O(kc)$ respectively. Thus, the number of calls to min Steiner cuts is at most $O(kc)$, and the number of calls to maximal t-isolating Steiner cut is at most $O(kc)$. The number of calls to the base case is at most $O(kc)$. Therefore, the running time is $O(kc(k^2mc + mc) + kc \cdot cm2^{O(c^2)}) = O(k^3mc + km2^{O(c^2)})$. □

Corollary 7.29. *Given a graph G with terminal set T and a parameter $c > 0$, there is an algorithm that computes a (T, c)-reducing-or-covering partition-set pair $((Z, X_1, \dots, X_\ell), C)$ for G such that*

- $|Z| = O(|T|c^2)$,
- $|C| = O(|T| \cdot 2^{O(c^2)})$, *and*
- $\sum_i |N(X_i)| = O(|T|c^2)$.

The algorithm takes $\tilde{O}(|T|^3mc + |T|m2^{O(c^2)})$ time.

Proof. Let $(\mathcal{S}, C)$ be a (T, c)-reducing-or-covering obtained from Lemma 7.24 with G, T, c as inputs. We compute $X_1, \dots, X_\ell$ from the recursion tree of the algorithm in Lemma 7.24 according to Definition 7.17. This takes at most $O(n)$ time per internal node in the recursion tree to label every node according to the Steiner cut. Since there are $O(|T|c)$ internal nodes, it takes total $O(n|T|c)$ additional time to compute $X_1, \dots, X_\ell$, which is subsumed by the running time Lemma 7.24. □

7.5 Fast Implementation for Expanders with Many Terminals

This section is devoted to proving the following lemma.

Lemma 7.30. *Given a ϕ-vertex expander graph G with terminal set T and a parameter $c > 0$, there is an algorithm that computes a (T, c)-reducing-or-covering partition-set pair $((Z, X_1, \dots, X_\ell), C)$ for G such that*

- $|Z| = O(|T|c^2)$,
- $|C| = O(|T| \cdot 2^{O(c^2)})$, *and*
- $\sum_i |N(X_i)| = O(|T|c^2)$.

The algorithm takes $\tilde{O}(mc\phi^{-1} + \phi^{-5}|T|2^{O(c^2)})$ time.

We first describe a simpler goal of simulating Algorithm 5 fast as stated in Lemma 7.31. Then, we prove Lemma 7.30 by showing that the algorithm in Definition 7.17 for extracting a (T, c)-reducing-or-covering partition-set pair from Algorithm 5 can be implemented efficiently.

Lemma 7.31. *If the input graph is a ϕ-vertex expander, then Algorithm 5 can be simulated by another algorithm that runs in $\tilde{O}(mc\phi^{-1} + \phi^{-5}k2^{O(c^2)})$ time where m is the number of edges and k is the number of terminals of the input graph for Algorithm 5.*

We describe the challenges for Lemma 7.31. Observe that the number of calls to min Steiner cut is $O(kc)$ (Lemma 7.12), and thus the previous algorithm can take up to $O(mkc)$ even if we assume linear-time algorithm for computing min Steiner cut. To get a better running time, we will exploit the crucial fact that ϕ-expander can only have unbalanced small separators. Therefore, we can compute min Steiner cut, and good t-isolating cut in sublinear time using local algorithms. We now make the intuition precise.

Definition 7.32. Let $G = (V, E)$ be a ϕ-expander graph. For any seed vertex $v \in V$, define $\mathcal{L}_G(v) = \{L \subseteq V : v \in L, \text{ and } |L| \leq \frac{c}{\phi}\}$, and $\text{local}_G(v) = \min_{L \in \mathcal{L}_G(v)} |N_G(L)|$. A minimizer vertex set $L \ni v$ is called *local cut* containing the seed vertex v. If $|N(L)| \leq c$, we say that L is a *local mincut* of size at most c.

We describe a different implementation of Algorithm 5 that is based on local algorithms. Let $t_{\text{local}}(c, \phi)$ be the running time to compute the local cut given a seed vertex or certify that min local cut has size at least $c + 1$. We first modify the base case. That is, if the number of terminals is smaller than $2c\phi^{-1}$ then we run the previous algorithm. From now we assume that the number of terminal is at least $2c\phi^{-1}$. The key property is that every local cut is Steiner cut and vice versa.

Proposition 7.33. *If G is a ϕ-expander, and $|T| \geq 2c\phi^{-1}$, then every local cut of size at most c is a Steiner cut of size at most c and vice versa.*

Computing min Steiner cuts. We maintain a list of value $\ell(t)$ for each terminal t with the invariant that $\ell(t) \leq \text{local}_G(t)$. With this invariant and Proposition 7.33, the local cut containing the seed node t' such that $\text{local}_G(t') = \min_t \ell(t)$ is a min Steiner cut. By monotonicity operations of constructing G_L and G_R, $\text{local}_G(t)$ does not decrease for all t. Therefore, to maintain the invariant, it is enough to record $\ell(t)$ to be the latest value of $\text{local}_G(t)$ checked by computing local cut containing t so far. The value $\ell(t)$ represents the lower bound of the current value of $\text{local}_G(t)$. To find a Steiner cut, we compute a min local cut given a seed vertex on the terminal t whose $\ell(t)$ is minimized, and update the value of $\ell(t)$. If the new local cut is not the smallest among $\ell(t)$, we keep that cut and recompute the next terminal whose ℓ is minimized. To find smallest $\ell(t)$, we use priority queue. Observe that the value of $\ell(t)$ can changed at most c time, and so the number of increase key operations is at most $O(kc)$, each operation takes $O(\log n)$ time. Since the number of Steiner cuts is at most $O(kc)$ over the entire algorithm, the total time due to computing min Steiner cuts is:

$$O(kc \cdot t_{\text{local}}(c, \phi) + kc \log n) \tag{7}$$

Computing good t-isolating cuts. Suppose we are given a local mincut L such that $(L, N(L), V - N[L])$ is a min Steiner cut and t-isolating. We describe how to get a good t-isolating cut from L. We construct G_L and G_R according to Lemma 7.3. Observe that G_L immediately becomes a base case. So, we work on G_R. Observe that $t = t_L$ in this case. Then, we compute the next local mincut using t as a seed vertex and repeat until one of the following happens: (1) the local cut is t-isolating and $\text{isolate}(t)$ increases or (2) the local cut is t-isolating and $N(L) \subseteq T$ or (3) the local cut is not t-isolating. For case (1), the previous local cut is a good t-isolating cut. For case (2), current local cut is a good t-isolating cut. Finally, for case (3), we obtain a min Steiner cut and it reduces to the previous step.

Next, we analyze the running time. Given a local mincut, the number of iterations until $\text{isolate}(t)$ increases or $N(L) \subseteq T$ is at most $2c\phi^{-1}$ since G is ϕ-expander. Each iteration takes $t_{\text{local}}(c, \phi)$ time. The value of $\text{isolate}(t)$ can increase at most c times, and there are at most k terminals. The case $N(L) \subseteq T$ and case (3) can also occur at most $O(kc)$ times. Therefore, the total time due to computing good t-isolating cuts (excluding the time to compute G_L and G_R) is:

$$O(kc^2\phi^{-1} \cdot t_{\text{local}}(c,\phi)) \tag{8}$$

Next, we explain how to efficiently compute G_L and G_R using hypergraphs. For brevity, we will focus on constructing left and right graphs of G with respect to (L, S, R). The construction of strictly left and strictly right graphs is similar.

Simulating with Hypergraphs. For the same reason as in the previous section, in order to construct G_L and G_R according to Line 9, we simulate the algorithm using hypergraph H such that $G = \mathsf{Clique}(H)$ as follows. Throughout this section, we view the hypergraph as a bipartite graph $H = (V_{\mathsf{v}}, V_{\mathsf{e}}, E)$ and denote G_{orig} as the initial input graph. Initially, H is a ϕ-expander, and furthermore for all $v \in V_{\mathsf{v}}$, $\deg_H(v) = \deg_G(v)$. To support efficient construction of G_L and G_R, we need a hypergraph that is equipped with additional operation as follows.

Definition 7.34. *Mergable hypergraph H* is a hypergraph $(V_{\mathsf{v}}, V_{\mathsf{e}}, E)$ that supports the following operation: $\textsc{Merge}(\hat{E} \subseteq V_{\mathsf{e}})$: contract $\hat{E}$ into a single node in V_{e} (i.e., taking the union of hyperedges in $\hat{E}$), there may be parallel edges from this operation.

Lemma 7.35 (Hypergraph With Efficient Hyperedge Merging)**.** *There is a mergable hypergraph that supports* $\textsc{Merge}(\hat{E})$ *operation in amortized $O(\alpha(m) \cdot |\hat{E}|) = \tilde{O}(|\hat{E}|)$ where $\alpha(m)$ is the inverse Ackermann function and m is the number of hyperedges, and the time to access an edge in E is $O(\alpha(m)) = \tilde{O}(1)$ per access.*

Intuitively, we can view mergable hypergraphs as hypergraphs but with $O(\alpha(m)) = \tilde{O}(1)$ overhead for accessing an individual edge in the bipartite representation. For the rest of the section, when we say hypergraphs we mean mergable hypergraphs, and the cost for accessing edges in its bipartite representation is $O(\alpha(m)) = \tilde{O}(1)$.

We maintain the invariant that at any time, $H = \mathsf{Clique}(G)$ and H is an expander with low volume in the following sense.

Definition 7.36. We say that a hypergraph H is a (c,ϕ)-*low-volume expander* if for all vertex cut (L, S, R) such that $|L| \leq |R|$ and $|S| \leq c$, we have $\text{vol}_H(L) \leq (\frac{2c}{\phi})^2$ and H is a ϕ-expander.

A key feature of low-volume expander is that we can compute local cuts in sublinear time.

Lemma 7.37. *If H is a (c,ϕ)-low volume expander, then for any seed vertex t, we can compute min local cut containing t or correctly report that local min cut containing t has size at least $c+1$ in $\tilde{O}(c^{O(c)}\phi^{-2})$ time. Therefore, $t_{\text{local}}(c,\phi) = \tilde{O}(c^{O(c)}\phi^{-2})$.*

Observe that initially the hypergraph is indeed a (c,ϕ)-low-volume expander. We will show that at any time every hypergraph during the recursion is (c,ϕ)-low-volume expanders.

Definition 7.38. Given a hypergraph H and a graph G, we say that H is *degree-dominated* by G, denoted $H \leq_{\deg} G$ if for all $v \in V_H$, if $v \in V_G$, then $\deg_H(v) \leq \deg_G(v)$; otherwise, $\deg_H(v) \leq 1$.

Lemma 7.39. *Let $H = (V_{\mathsf{v}}, V_{\mathsf{e}}, E)$ be a hypergraph that is a (c,ϕ)-low-volume expander and degree-dominated by G_{orig} and $G = \mathsf{Clique}(H)$, and T be a terminal set. Let $G_L, G_R, T_L, T_R, t_L, t_R$ be the graphs and terminal sets defined in Definition 7.2. Given a pair of vertex sets (L, S) of H such that $N_H(L) = S$ and $|S| \leq c$, there is an algorithm that outputs a hypergraph H_L with terminal set T_L and modify H to be another hypergraph H_R with terminal set T_R satisfying the following properties:*

1. *$G_L = \mathsf{Clique}(H_L)$ and $G_R = \mathsf{Clique}(H_R)$,*

2. *$H_L \leq_{\deg} G_{\text{orig}}$ and $H_R \leq_{\deg} G_{\text{orig}}$,*

3. *H_L and H_R are both (c, ϕ)-low-volume vertex expanders, and*

4. *H_L has at most $2c\phi^{-1}$ vertices and is of size at most $O(c^3\phi^{-2})$.*

The algorithm runs in $\tilde{O}((c\phi^{-1})^3 + (c\phi^{-1}) \cdot \text{vol}_H(S - T))$ time.

We prove Lemma 7.35, Lemma 7.37 and Lemma 7.39 in Section 7.6. We are now ready to prove Lemma 7.31. We start with algorithm, followed by analysis.

Modified Algorithm. We describe the modified version of Algorithm 5. Let G_{orig} be the initial input graph that is ϕ-vertex expander, and T be the set of terminals for G_{orig}. We first construct a mergable hypergraph H such that $G_{\text{orig}} = \mathsf{Clique}(H)$. This step takes $O(m)$ time. Observe that H is (c, ϕ)-low volume vertex expander. The new base case is that whenever the number of terminals is smaller than $2c\phi^{-1}$. If that happens, we apply Lemma 7.24. Otherwise, we compute min Steiner cuts and good t-isolating cuts as discussed above. When a local mincut L is found, we apply Lemma 7.39 to implement Line 9, and recurse on H_L and H_R. Observe that H_L will always go to the base case since the number of vertices is at most $2c\phi^{-1}$. By Lemma 7.39, we have that at any time the hypergraph H_R is (c, ϕ)-low-volume vertex expander, and $H_R \leq_{\deg} G_{\text{orig}}$.

Recursion Tree. We analyze the modified algorithm. We have a recursion tree where each node is a pair of hypergraph and its terminal set (H, T). If the current node is (H, T) with (L, S, R) as a min Steiner cut, the left child is (H_L, T_L) and the right child is (H_R, T_R). Note that the recursion tree is quite simple: the left node always go to the base case. By induction, we can show that $G = \mathsf{Clique}(H)$ at any time. By the correctness of the base case (Lemma 7.24), the modified algorithm indeed simulates Algorithm 5.

Proof of Lemma 7.31. It remains to analyze the running time of the modified algorithm. Observe that at any time H is (c, ϕ)-low volume vertex expander. By Lemma 7.37, we have $t_{\text{local}}(c, \phi) = \tilde{O}(c^{O(c)}\phi^{-2})$. The running time due to computing min Steiner cuts is given by Equation (7) which is:

$$\tilde{O}(kc \cdot t_{\text{local}}(c, \phi) + kc \log n) = \tilde{O}(k\phi^{-2}c^{O(c)}). \tag{9}$$

The running time due to computing good t-isolating cut is given by Equation (8) (excluding the time to construct G_L, G_R) which is:

$$O(kc^2\phi^{-1} \cdot t_{\text{local}}(c, \phi)) = \tilde{O}(k\phi^{-3}c^{O(c)}). \tag{10}$$

We now describe the total cost for constructing subproblems H_L and H_R given the current hypergraph H. Note that the time includes the construction of G_L, G_R when we compute good t-isolating cuts. By Lemma 7.39, the running time is $\tilde{O}((c\phi^{-1})^3 + (c\phi^{-1}) \cdot \text{vol}_H(S - T))$ per internal node in the recursion tree. Since the number of internal nodes is at most kc, the first term sums to $\tilde{O}(kc^4\phi^{-3})$. The bound the second term, observe that each node $v \in S - T$ is non-terminal, and v corresponds to the original vertex in G_{orig}. Since $H \leq_{\deg} G_{\text{orig}}$, we have that $\deg_H(v) \leq \deg_{G_{\text{orig}}}(v)$ for all $v \in S - T$. Since each v appears in $S - T$ at most once, we have that the total cost over the entire recursion is at most $\tilde{O}((c\phi^{-1}) \cdot \sum_{v \in V} \deg_{G_{\text{orig}}}(v)) = \tilde{O}(c\phi^{-1} \cdot m)$. Therefore, the running time due to Line 9 is:

$$\tilde{O}(kc^4\phi^{-3} + c\phi^{-1} \cdot m). \tag{11}$$

Finally, we bound the running time due to the base cases. By Lemma 7.39, each H_L becomes a base case of size $O(c^3\phi^{-2})$. Since there are at most $O(kc)$ min Steiner cuts throughout the recursion, the total size of the base cases is $m' = O(kc^4\phi^{-2})$. Let k' be the number of terminal in the base case. By definition, $k' \leq 2c\phi^{-1}$. By Lemma 7.24, the total running time due to the base cases is given by

$$\tilde{O}(k'^3m'c + k'm'2^{O(c^2)}) = \tilde{O}(c^8\phi^{-5}k + c^5\phi^{-3}2^{O(c^2)}k) = \tilde{O}(\phi^{-5}2^{O(c^2)}k). \tag{12}$$

Summing all the cost from Equations (9) to (12), we obtain the desired running time. □

Extracting A (T, c)-Reducing-or-Covering Partition-Set Pair. Finally, we prove Lemma 7.30.

Proof of Lemma 7.30. By Lemma 7.31, the modified algorithm simulates Algorithm 5 efficiently. Given a recursion tree of the modified algorithm, our next goal is to compute the partition $(Z, X_1, \ldots, X_\ell)$ according to Lemma 7.18 efficiently. In order to extract reducing-or-covering partition-set pair, we apply Corollary 7.29 on the new base cases. More precisely, we define *the right spine* of the recursion tree as the path from root to the first right-only descendant of the root that is not a base case. The right spine of the recursion tree can be viewed as an execution on the original graph until it becomes a base case. Indeed, observe that at the root node we start with the entire graph. Then, whenever we find a local cut, construct H_L and recurse on the left (which immediately goes into the base case that calls Corollary 7.29), and continue on the remaining graph.

We now describe the desired output $((Z, X_1, \ldots, X_\ell), C)$. Observe that a node v is a base case if and only if v is not in the right spine of the recursion tree. For each base case v, we apply Corollary 7.29 to obtain a reducing-or-covering partition-set pair (Π_v, C_v) at the subproblem in node v in the recursion tree. Recall that Π_v consists of the reducer and a non-reducer sequence. The final output is $\Pi = (Z, X_1, \ldots, X_\ell)$ where the reducer of Π is $Z = Z_{\text{base}} \cup Z_{\text{spine}}$ where Z_{base} is the union of the sets of the reducer of the partition Π_v over all base case v, and Z_{spine} is the union of Steiner cuts along the right spine of the recursion tree, and the non-reducer of Π can be obtained by the union of of all non-reducer sequence of all partition Π_v over all the base cases v. Finally, C is a union of C_v over all base cases v. It follows by design that the pair (Π, C) is a (T, c)-reducing-or-covering of G_{orig} with the desired properties because we compute the pair (Π, C) according to Lemma 7.18. The running time to compute the pair (Π, C) is dominated by the algorithm to complete the recursion tree. □

7.6 Implementation Details

We prove Lemma 7.35, Lemma 7.37 and Lemma 7.39 in this section. For convenience, we treat mergable hypergraphs as normal hypergraphs and hide the $O(\alpha(m))$ factor overhead in the polylog factors. We may call merge operation when needed.

7.6.1 Proof of Lemma 7.35

To implement mergable hypergraph, we view $H = (V_{\mathsf{v}}, V_{\mathsf{e}}, E)$ as a bipartite graph. Furthermore, instead of viewing V_{e} as a set of vertices, we view V_{e} as a collection of sets. More precisely, recall that initially our hypergraph satisfies the following $G_{\text{orig}} = \mathsf{Clique}(H)$. Suppose initially $V_{\mathsf{e}} = \{e_1, \ldots, e_m\}$. We view $V_{\mathsf{e}} = \{\{e_1\}, \ldots, \{e_m\}\}$. To implement $\textsc{Merge}(\hat{E})$, we replace each set in $\hat{E}$ with $\bigcup_{C \in \hat{E}} C$. We implement set union operation using union-find data structures $\mathcal{U}$ where the ground set is V_{e} in the initial hypergraph. For any edge $(u, e) \in E$ in the original bipartite graph, we have the corresponding edge $(u, \mathcal{U}.\text{find}(e))$ in the mergable hypergraph. Note that this allows parallel edges in bipartite graph after merging operations. Also, union and find operations take amortized $O(\alpha(m))$ time where m is the size of the hypergraph.

7.6.2 Proof of Lemma 7.37

Recall that $H = (V_{\mathsf{v}}, V_{\mathsf{e}}, E)$. We view H interchangably as a bipartite graph and a hypergraph. The local cut algorithm is the following: Given a seed vertex t, we apply LocalVC algorithm on the bipartite graph H using volume parameter $\nu = (c+1)(c\phi^{-1})^2$ and cut parameter c. If LocalVC outputs a local cut L, then we binary search on value c until we find the $c = \text{local}(t)$. Otherwise, we report that there is no local mincut of size at most c. Since LocalVC runs in $O(k^k \nu)$ where $k \leq c$, and $\nu = O(c^3\phi^{-2})$, the total time is $\tilde{O}(c^{O(c)}\phi^{-2})$.

We argue the correctness. Suppose there is a vertex cut (L, S, R) in the hypergraph H such that $t \in L, |S| \leq c, |L| \leq |R|$. The neighbor of L, denoted as $N_B(L)$ in bipartite graph represents all hyperedges incident to L. Denote $L' = N_B(L) \cup L$ the set of vertices L and their hyperedges in the bipartite graph. By definition, $N_B(L') = S$ and $|N_B(L')| = |S| \leq c$. Therefore, $\text{vol}_B(L') = \text{vol}_B(L) + \text{vol}_B(L) \cdot |S| \leq (c\phi^{-1})^2(c+1)$. The last inequality follows since H is (ϕ, c)-low volume expander (so we have $\text{vol}_B(L) = \text{vol}_H(L) \leq O((c\phi^{-1})^2))$. Since $\text{vol}_B(L') \leq (c+1)(c\phi^{-1})^2$ and $N_B(L') = |S| \leq c$, LocalVC will output a local cut. If there is no such (L, S, R), then LocalVC always reports there is no local cut.

7.6.3 Proof of Lemma 7.39

Since H is (ϕ, c)-low volume vertex expander, we have

$$\text{vol}_H(L) \leq O((c\phi^{-1})^2) \text{ and } |L| + |S| \leq 2c\phi^{-1}. \tag{13}$$

Construction of H_L. We first show that we can construct H_L satisfying aforementioned properties in $O((c\phi^{-1})^3 + (c\phi^{-1}) \cdot \text{vol}_H(S - T))$ time. Recall the set $\hat{R} = R \cup (S - T)$ as defined in Lemma 7.3. Our algorithm is to identify neighborhood of $\hat{R}$ in $L \cup (S - T)$, denoted as $N(\hat{R})$, without reading the entire set of $\hat{R}$. Let H'_L be the hypergraph induced by $H[L \cup S]$.

Next, we describe the algorithm that compute $N(\hat{R})$. Observe that if $v \in L$, then any hyperedge containing v must incident to at most $|L| + |S|$ vertices. Therefore, we can check for every vertex $v \in L$ if v is a neighbor of $\hat{R}$ in total $O(\text{vol}_H(L) \cdot (|L| + |S|)) \overset{(13)}{=} O((c\phi^{-1})^3)$ time. Next, observe that for every hyperedge e containing a vertex v in S, if e is incident to more than $2c\phi^{-1} \overset{(13)}{\geq} |L| + |S|$ vertices, v must be incident to $\hat{R}$. So, we need to check only hyperedges incident to $S - T$ that incident at most $O(c\phi^{-1})$ vertices. This takes $O(\text{vol}_H(S - T) \cdot c\phi^{-1})$ time.

Once $N(\hat{R})$ is found, we modify H'_L as follows. We remove $S - T$, remove hyperedges incident to $N(\hat{R})$, and add a vertex t_R and a new hyperedge to that incident to every vertex in $N(\hat{R}) \cup \{t_R\}$. The modified hypergraph corresponds to H_L where $G_L = \mathsf{Clique}(H_L)$. The number of vertices in H_L is at most $|L| + |S| \overset{(13)}{\leq} 2c\phi^{-1}$. Furthermore, every old hyperedge in G_L must be incident to $L \cup (S - T)$. Let E' be the set of hyperedges that incident to L. Since $|S| \leq c$, the total size of E' is at most $O(c \cdot \text{vol}_H(L))$. Therefore, the size of G_L is at most $O(\text{vol}_H(L) + \text{vol}_H(E')) = O(c \cdot (c\phi^{-1})^2) = O(c^3\phi^{-2})$. Also, observe that the degree of each vertex does not increase. The terminal t_R has degree exactly 1. Combining with the fact that $H \leq_{\text{deg}} G_{\text{orig}}$, we conclude that $H_L \leq_{\text{deg}} G_{\text{orig}}$.

Construction of H_R. We next describe the construction of H_R in $\tilde{O}((c\phi^{-1})^2 + \mathrm{vol}_H(S-T))$ time. Recall the set $\hat{L} = L \cup (S-T)$ as defined in Lemma 7.3. Let E_1 be the set of hyperedges incident to $S-T$, and E_2 be the set of hyperedges incident to vertices only in $\hat{L}$. We now describe the construction.

1. Compute $\hat{E} = E_1 - E_2$. This step takes $\tilde{O}(\mathrm{vol}_H(L) + \mathrm{vol}_H(S-T)) \overset{(13)}{=} \tilde{O}((c\phi^{-1})^2 + \mathrm{vol}_H(S-T))$ time.
2. Apply Merge($\hat{E}$) operation and obtain a new hyperedge $\hat{e}$ ($\hat{e}$ is added to V_{e}). By Lemma 7.35, this step takes $\tilde{O}(|\hat{E}|) = \tilde{O}(\mathrm{vol}_H(S-T))$ time.
3. Remove $\hat{L}$ from H. This step takes $\tilde{O}(\mathrm{vol}_H(L)) \overset{(13)}{=} \tilde{O}((c\phi^{-1})^2)$ time.
4. Add a terminal t_L and add $(t_L, \hat{e})$ edge to E.
5. Add for all $s \in S \cap T$, add $(s, \hat{e})$ edge to E.

By design, every vertex in $N(\hat{L})$ is incident to the hyperedge $\hat{e}$. Combining with the fact that $G = \mathsf{Clique}(H)$, we have $G_R = \mathsf{Clique}(H_R)$. We next show that $H_R \leq_{\deg} G_{\text{orig}}$. By Definition 7.34, Merge operation does not reduce the degree in step 2. Observe that the vertex t_L has degree exactly 1 by step 3. The only potential increase in degree (by one) is the vertices in $S \cap T$ in step 4. However, since S is a min Steiner cut, every node $v \in S$ must have an hyperedge that is incident to L. Since we remove $\hat{L}$ in step 3, the degree must be reduced by at least 1. Therefore, the degree of v does not increase at the end of step 5. Combining with the fact that $H \leq_{\deg} G_{\text{orig}}$ we conclude that $H_R \leq_{\deg} G_{\text{orig}}$.

H_L and H_R are both (c,ϕ)-low volume vertex expander. We now prove the last remaining property for H_L and H_R. We prove for H_L (the case H_R is similar). Let (L, S, R) be a vertex cut in H_L where $|S| \leq c$. By monotonicity property (Proposition 7.14), we have that S must be a separator in H. By similar argument in Proposition 7.14, H_L and H_R are ϕ-vertex expanders. Therefore, there is a vertex cut (L', S, R') in H such that L' becomes L and R' becomes R after transforming H to H_L. Since the degree of each vertex does not increase, we have $\mathrm{vol}_{H_L}(L) \leq \mathrm{vol}_H(L')$. Since H is a (c,ϕ)-low volume expander, we also have $\mathrm{vol}_H(L') \leq (c\phi^{-1})^2$. Therefore, $\mathrm{vol}_{H_L}(L) \leq \mathrm{vol}_H(L') \leq (c\phi^{-1})^2$. We conclude that H_L is a (c,ϕ)-low volume vertex expander.

7.7 The Final Algorithm

This section is devoted to proving Theorem 6.2. We start with an important tool: vertex expander decomposition.

Theorem 7.40 ([LS22])**.** *There is a deterministic algorithm that, given a graph $G = (V, E)$ with n vertices and m edges and a parameter $\phi \in (0, 1/10 \log n)$, computes in $m^{1+o(1)}/\phi$ time a partition $\{Z, X_1, \ldots, X_\ell\}$ of V and $\{Y_1, \ldots, Y_\ell\}$ such that*

- $Z = \bigcup_i (Y_i - X_i)$,
- *for all i, $N[X_i] \subseteq Y_i \subseteq X_i \cup Z$,*
- *for all i, $G[Y_i]$ is a ϕ-vertex expander, and*
- $\sum_i |Y_i - X_i| \leq \phi n^{1+o(1)}$.

The first item implies that $N(X_i) \subseteq Z$ for all i. That is, all X_i are separated from each other by Z. The set Y_i is the set containing $N[X_i]$ such that $G[Y_i]$ is a vertex expander, but Y_i is not too big in the sense that the "additional" part $Y_i \setminus X_i \subseteq S$ and also $\sum_i |Y_i \setminus X_i| \leq \phi n^{1+o(1)}$. Note also that a vertex could be counted in $\sum_i |Y_i \setminus X_i|$ many times. That is, we actually bound the total number of "occurrences" of all vertices in $Y_i \setminus X_i$ overall i.

Observe that each $G[X_i]$ is possibly not connected. This is unavoidable, for example, when G is a star.

Our plan is to apply expander decomposition as a preprocessing step before applying Lemma 7.30 on each expander. Once we obtain reducing-or-covering partition-set pair for each expander, we need to "compose" them. To do so, we need the notion of composibility of (T, c)-reducing-or-covering partition-set pair for the decomposition. We state it in a general form as follows. Recall the definitions of reducer and non-reducer sequence in Definition 6.1.

Lemma 7.41. *Let $(Z, X_1, \ldots, X_\ell)$ be a partition of $V(G)$ such that Z is (X_i, X_j)-separator for all distinct i, j. For $i \in [\ell]$, let Y_i be a set of vertices such that $N_G[X_i] \subseteq Y_i \subseteq X_i \cup Z$. Let $G_i = G[Y_i]$ and $T_i = (T \cap X_i) \cup (Y_i - X_i)$. For $i \in [\ell]$, let (Π_i, C_i) be (T_i, c)-reducing-or-covering partition-set pair in G_i. We define* $\textsc{Compose}(Z, (\Pi_1, C_1), \ldots, (\Pi_\ell, C_i))$ *that outputs $((Z', X'_1, \ldots, X'_{\ell'}), C')$ where each set in the output is defined as follows.*

- *$Z' = Z \cup \bigcup_{i \leq \ell} Z_i$ where Z_i is the reducer of Π_i,*
- *$C' = \bigcup_{i \leq \ell} C_i$, and*
- *The sequence $X'_1, \ldots X'_{\ell'}$ is obtained as follows. We first subtract Z from every set in the non-reducer sequences of the partition Π_i. Then, for all partition Π_i, we append all the resulting non-reducer sequences.*

Then, $((Z', X'_1, \ldots, X'_{\ell'}), C') = \textsc{Compose}(Z, (\Pi_1, C_1), \ldots, (\Pi_\ell, C_\ell))$ *is a (T, c)-reducing-or-covering partition-set pair in G where*

- *$|Z'| \leq |Z| + \sum_i |Z_i|$ where Z_i is the reducer of Π_i,*
- *$|C'| \leq \sum_i |C_i|$, and*
- *$\sum_{i' \in [\ell']} |N_G(X'_{i'})| \leq \sum_{i \in [\ell]} \sum_j |N_{G_i}(Q_{i,j})| + \sum_{i \in [\ell]} |Y_i - X_i|$ where $Q_{i,j}$ is the j-th set in the non-reducing sequence of Π_i.*

We prove Lemma 7.41 in Appendix A.2 since the proof is similar to that of Lemma 7.3.

We are now ready to prove Theorem 6.2. We describe the algorithm and analysis.

Algorithm. We describe the algorithm in Algorithm 6. Note that the input graph has arboricity c. First, we apply expander decomposition. For each expanders, we apply Algorithm 5 using terminal set in the boundary $Y_i - X_i$ and $T \cap X_i$. Since each graph is an expander, we can implement Algorithm 5 fast using Lemma 7.30. Finally, we compose all the reducing-or-covering partition-set pair to output the final (T,c)-reducing-or-covering partition-set pair in G.

Algorithm 6: FastROC(G,T,c,ϕ)

Input: A graph $G=(V,E)$, a terminal set $T\subseteq V$ and parameters $c>0, \phi\in(0,1/10\log|V|)$ where G has arboricity c.

Output: A (T,c)-reducing-or-covering partition-set pair in G.

Apply Theorem 7.40 using G and ϕ to obtain $S, X_1,\dots,X_\ell$ and $Y_1,\dots Y_\ell$.
for $i\in[\ell]$ **do**
  Let $T_i=(T\cap X_i)\cup(Y_i-X_i)$.
  $(\Pi_i,C_i)=\mathrm{ROC}(G[Y_i],T_i,c)$ // Algorithm 5
return Compose$(Z,(\Pi_1,C_1),\dots,(\Pi_\ell,C_\ell))$ where Compose is defined in Lemma 7.41.

We denote $|T|=k$ and $m=|E|$. For $i\le\ell$, denote $|T_i|=k_i$ and $|E(G[Y_i])|=m_i$. The total size of terminals is:

$$\sum_i k_i=\sum_i(|T\cap X_i|+|Y_i-X_i|)=O(k+\phi n^{1+o(1)}) \tag{14}$$

Next, the total instances size is:

$$\sum_i m_i=\sum_i|E_G(Y_i,Y_i)|\le\sum_i c|Y_i|=\sum_i(c|Y_i-X_i|+c|X_i|)=O(c\phi n^{1+o(1)}+nc) \tag{15}$$

The first inequality follows because G has arboricity c (i.e., for all $S\subseteq V(G)$, $|E(S,S)|\le c|S|$).

Correctness. Let $(\Pi'=(Z',X'_1,\dots,X'_{\ell'}),C')$ be the output of Algorithm 6. For $i\le\ell$, let Z_i be the reducer of partition Π_i, and let Q_{ij} be j-th set of the non-reducer sequence of Π_i. Since (Π_i,C_i) is (T_i,c)-reducing-or-covering for all i, Lemma 7.41 implies that (Π',C') is (T,c)-reducing-or-covering where the sizes of $|Z'|,|C'|,\sum_{i'\in[\ell']}|N_G(X'_{i'})|$ are stated in Lemma 7.41. We now argue each one accordingly. First, we bound the size of Z'.

$$|Z'|\le|Z|+\sum_i|Z_i|\le\phi n^{1+o(1)}+O((\sum_i k_i)c^2)\overset{(14)}{=}O((k+\phi n^{1+o(1)})c^2).$$

The first inequality follows by Lemma 7.41. For the second inequality, $|Z|\le\phi n^{1+o(1)}$ because $Z=\bigcup_i(Y_i-X_i)$, and $\sum_i|Y_i-X_i|\le\phi n^{1+o(1)}$ (Theorem 7.40), and the term $\sum_i|Z_i|\le O(\sum_i k_ic^2)$ follows from Lemma 7.30. Next, we bound the size of C'.

$$|C'|\le\sum_i|C_i|\le(\sum_i k_i)2^{O(c^2)}\overset{(14)}{=}O((k+\phi n^{1+o(1)})\cdot 2^{O(c^2)})=O((\phi n^{1+o(1)}+k)\cdot 2^{O(c^2)}).$$

The first inequality follows from Lemma 7.41. The second inequality follows from Lemma 7.30. Finally, we bound the total neighbors.

$$\begin{aligned}\sum_{i'\in[\ell']}|N_G(X'_{i'})|&\le\sum_{i\in[\ell]}\sum_j|N_{G_i}(Q_{i,j})|+\sum_{i\in[\ell]}|Y_i-X_i|\\&\le O(\sum_i k_ic^2)+\phi n^{1+o(1)}\\&\overset{(14)}{=}O((k+\phi n^{1+o(1)})c^2).\end{aligned}$$

The first inequality follows from Lemma 7.41. The second inequality follows from Lemma 7.30.

Running Time. By Theorem 7.40, the expander decomposition algorithm takes $O(m^{1+o(1)}\phi^{-1})$ time and guarantees that $G[Y_i]$ is a ϕ-vertex expander for all i. By Lemma 7.31, the time to compute R_i is $\tilde{O}(m_i c\phi^{-1} + \phi^{-5}k2^{O(c^2)})$. Therefore, the total running time is:

$$\begin{aligned}
&\tilde{O}(m^{1+o(1)}\phi^{-1} + (\sum_i m_i)\cdot c\phi^{-1} + \phi^{-5}(\sum_i k_i)\cdot 2^{O(c^2)}) \\
&\overset{(14 \text{ and } 15)}{=} \tilde{O}(m^{1+o(1)}c\phi^{-1} + (\phi^{-5}k + \phi^{-4}n^{1+o(1)})\cdot 2^{O(c^2)}) \\
&= \tilde{O}(m^{1+o(1)}\phi^{-4}\cdot 2^{O(c^2)}).
\end{aligned}$$

8 Open Problems

Deterministic Vertex Connectivity Algorithms. It remains an outstanding open problem whether there exists a linear-time deterministic vertex connectivity algorithm as asked by Aho, Hopcroft, and Ullman in 1974 ([AHU74] Problem 5.30). Our results answer this question affirmatively up to a subpolynomial factor in the running time when the connectivity $c = o(\sqrt{\log n})$. Can one improve the dependency on c in our running time to $m^{1+o(1)}\operatorname{poly}(c)$? This would follow immediately by our framework if one can improve Theorem 1.2 so that a (T,c)-sparsifier of size $|T|\operatorname{poly}(c)$ can be obtained in $m^{1+o(1)}\operatorname{poly}(c)$ time. For general connectivity c, even a deterministic algorithm with $\tilde{O}(mn)$ time bound is still open.

How fast can we compute vertex connectivity in directed graphs? The algorithm by [FNS+20] runs in $\tilde{O}(mc^2)$ time, which is near-linear for all $c = \operatorname{polylog}(n)$. When c can be big, the algorithm by [LNP+21] takes $n^{2+o(1)}$ time when we apply the almost-linear-time max flow algorithm by [CKL+22]. However, both of these algorithms are Monte-Carlo. In contrast, the fastest deterministic algorithm [Gab06] still requires in $O(mn + m\cdot\min\{c^{5/2}, c\cdot n^{3/4}\})$ time. No almost linear time deterministic algorithm is known even when we promise that $c = O(1)$.

Weighted Vertex Connectivity. Given a vertex-weighted graph, the *weighted vertex connectivity* problem is to compute a vertex cut with minimum total weight. The state-of-the-art of this natural extension is by Henzinger Rao and Gabow [HRG00] who gave a $\tilde{O}(mn)$-time Monte-Carlo algorithm and a $\tilde{O}(\kappa_0 mn)$-time deterministic algorithm where κ_0 is the unweighted vertex connectivity. It is an outstanding open problem whether this bound can be improved.

Mimicking Networks. Our algorithm for mimicking network is of independent interest. An immediate open problem is to compute a mimicking network of size $k\operatorname{poly}(c)$ in $m^{1+o(1)}\operatorname{poly}(c)$ time where k is the number of terminals. The same question holds for the edge version of the mimicking networks. For the purpose of vertex connectivity algorithm in our framework, it is enough for the sparsifier to preserve only a separator of size at most c that splits terminal set T (instead of all possible pair $A, B \subseteq T$ such that $\mu_G(A,B) < c$). Using a weaker notion of sparsifier, is it possible to obtain such a sparsifier of size $k\operatorname{poly}(c)$ in $m^{1+o(1)}\operatorname{poly}(c)$ time?

Acknowledgement

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672 and No 759557. We thank Yu Gao, Jason Li, Danupon Nanongkai, and Richard Peng for discussion during the early stage of this project. We thank the 2021 HIM program Discrete Optimization during which part of this work was developed. We thank the anonymous reviewer for suggesting a simple proof of Lemma 5.7.

References

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. 44

[BDD+82] Michael Becker, W. Degenhardt, Jürgen Doenhardt, Stefan Hertel, Gerd Kaninke, W. Kerber, Kurt Mehlhorn, Stefan Näher, Hans Rohnert, and Thomas Winter. A probabilistic algorithm for vertex connectivity of graphs. *Inf. Process. Lett.*, 15(3):135–136, 1982. 1

[CDK+21] Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1206–1225. SIAM, 2021. 3

[CGK14] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *PODC*, pages 156–165. ACM, 2014. 1

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020. 1

[CHI+17] Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F Italiano, Veronika Loitzenbauer, and Nikos Parotsidis. Faster algorithms for computing maximal 2-connected subgraphs in sparse directed graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1900–1918. SIAM, 2017. 2, 4, 5

[CKL+22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint arXiv:2203.00671*, 2022. 1, 44

[CR94] Joseph Cheriyan and John H. Reif. Directed s-t numberings, rubber bands, and testing digraph k-vertex connectivity. *Combinatorica*, 14(4):435–451, 1994. Announced at SODA'92. 1

[CT91] Joseph Cheriyan and Ramakrishna Thurimella. Algorithms for parallel k-vertex connectivity and sparse certificates (extended abstract). In *STOC*, pages 391–401. ACM, 1991. 1

[EH84] Abdol-Hossein Esfahanian and S. Louis Hakimi. On computing the connectivities of graphs and digraphs. *Networks*, 14(2):355–366, 1984. 1

[ET75a] Shimon Even and R Endre Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975. 1

[ET75b] Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. 1

[Eve75] Shimon Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM J. Comput.*, 4(3):393–396, 1975. 1

[FNS+20] Sebastian Forster, Danupon Nanongkai, Thatchaphol Saranurak, Liu Yang, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In *SODA*, pages 2046–2065. SIAM, 2020. 1, 44

[Gab06] Harold N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006. Announced at FOCS'00. 1, 44

[Gal80] Zvi Galil. Finding the vertex connectivity of graphs. *SIAM J. Comput.*, 9(1):197–199, 1980. 1

[GLN+19] Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex. *arXiv preprint arXiv:1910.07950*, 2019. 1

[Hen97] Monika Rauch Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *J. Algorithms*, 24(1):194–220, 1997. 1

[HRG00] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000. Announced at FOCS'96. 1, 44

[HRW17] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1919–1938, 2017. 1

[HT73] John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. 1

[Kle69] D Kleitman. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory*, 16(2):232–233, 1969. 1

[KT15] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In *STOC*, pages 665–674. ACM, 2015. 1

[KW20] Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM (JACM)*, 67(3):1–50, 2020. Announced at FOCS'12. 3

[Li21] Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–395, 2021. 1

[Liu20] Yang P Liu. Vertex sparsification for edge connectivity in polynomial time. *arXiv preprint arXiv:2011.15101*, 2020. 3

[LLW88] Nathan Linial, László Lovász, and Avi Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8(1):91–102, 1988. Announced at FOCS'86. 1

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 317–329, 2021. 1, 44

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society, 2020. 1

[LPS19] Yang P Liu, Richard Peng, and Mark Sellke. Vertex sparsifiers for c-edge connectivity. *arXiv preprint arXiv:1910.10359*, 2019. 3

[LS22] Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. *arXiv preprint arXiv:2205.03930*, 2022. To appear at FOCS'22. 13, 41

[Mat87] David W. Matula. Determining edge connectivity in o(nm). In *FOCS*, pages 249–251. IEEE Computer Society, 1987. 1

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. 1, 4

[NSY19] Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *STOC*, pages 241–252. ACM, 2019. 1, 2, 4, 5

[Pod73] VD Podderyugin. An algorithm for finding the edge connectivity of graphs. *Vopr. Kibern*, 2:136, 1973. 1

[Sar21] Thatchaphol Saranurak. A simple deterministic algorithm for edge connectivity. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 80–85. SIAM, 2021. 1

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. Announced at FOCS'71. 1

A Omitted Proofs

A.1 Proof of Claim 6.5

We prove by induction on c and c_T. We omit the subpolymonial factor for simplicity of the presentation. The base cases follow trivially. Next, we want to prove that

$$s(n,k,c,c_T) \leq (k+n\phi)(4+c+c_T)^{3(c+c_T)} \cdot (1+c_T) \cdot 2^{(\tau c^2 + c_T)} \tag{16}$$

Assuming that Equation (16) holds for parameters $(c-1, c_T)$ and (c, c_T-1), we prove that Equation (16) holds for the parameter (c, c_T). Observe that

$$\sum_i (k_i + \phi n_i) \overset{(2)}{\leq} (k+n\phi)c^2 + \phi n + \phi(k+n\phi)c^2 \leq 3(k+n\phi)c^2 \tag{17}$$

Applying the induction hypothesis on Equation (1), we obtain:

$$\begin{aligned}
s(n,k,c,c_T) &\le (k+\phi n)2^{\tau c^2} + \sum_i (k_i+\phi n_i)(3+c+c_T)^{3(c+c_T)-3}\cdot(1+c_T)\cdot 2^{(\tau(c-1)^2+c_T)} \\
&\quad + \sum_i (k_i+\phi n_i)(3+c+c_T)^{3(c+c_T)-3}\cdot((1+c_T)-1)\cdot 2^{(\tau c^2+c_T-1)} \\
&\overset{(17)}{\le} \sum_i (k_i+\phi n_i)(3+c+c_T)^{3(c+c_T)-3}\cdot(1+c_T)\cdot 2^{(\tau(c-1)^2+c_T)} \\
&\quad + \sum_i (k_i+\phi n_i)(3+c+c_T)^{3(c+c_T)-3}\cdot(1+c_T)\cdot 2^{(\tau c^2+c_T-1)} \\
&\le \sum_i (k_i+\phi n_i)(3+c+c_T)^{3(c+c_T)-3}\cdot(1+c_T)\cdot(2^{(\tau(c-1)^2+c_T)}+2^{(\tau c^2+c_T-1)}) \\
&\le \sum_i (k_i+\phi n_i)(3+c+c_T)^{3(c+c_T)-3}\cdot(1+c_T)\cdot 2^{(\tau c^2+c_T)} \\
&\overset{(17)}{\le} (k+n\phi)(3c^2)(3+c+c_T)^{3(c+c_T)-3}\cdot(1+c_T)\cdot 2^{(\tau c^2+c_T)} \\
&\le (k+n\phi)(4+c+c_T)^{3(c+c_T)}\cdot(1+c_T)\cdot 2^{(\tau c^2+c_T)}.
\end{aligned}$$

Observe $2^{(\tau(c-1)^2+c_T)}+2^{(\tau c^2+c_T-1)} \le 2\cdot 2^{(\tau c^2+c_T-1)} = 2^{(\tau c^2+c_T)}$ since $\tau(c^2-2c+1)+c_T \le \tau c^2+c_T-1$.

A.2 Proof of Lemma 7.41

The fact that $|Z'| \le |Z| + \sum_i |Z_i|$ where Z_i is the reducer of Π_i and that $|C'| \le \sum_i |C_i|$ follow by design. Next, we show that $\sum_{i'\in[\ell']} |N_G(X'_{i'})| \le \sum_{i\in[\ell]}\sum_j |N_{G_i}(Q_{i,j})| + \sum_{i\in[\ell]} |Y_i - X_i|$. Observe that, for all i,j,

$$N_G(Q_{ij} - Z) = N_{G_i}(Q_{ij} - Y_i) \subseteq N_{G_i}(Q_{ij}) \cup (Y_i \cap Q_{ij}). \tag{18}$$

Since $X'_{i'} = Q_{ij} - Z$ for some i,j, and Q_{ij} are disjoint among those with the same i, we have $\sum_{i'\in[\ell']} |N_G(X'_{i'})| \le \sum_{i\in[\ell]}\sum_j |N_{G_i}(Q_{i,j})| + \sum_{i\in[\ell]} |Y_i - X_i|$.

Furthermore, Equation (18) implies that $N_G(Q_{ij} - Z) \subseteq Z_i \cup Z \subseteq Z'$ for all i,j. Therefore, Z' is an (X'_i, X'_j)-separator in G for all i,j. It remains to prove the following claim.

We prove that $(\Pi' = (Z', X'_1, \dots, X'_{\ell'}), C')$ is a (T,c)-reducing-or-covering partition-set pair in G. Fix a pair $A,B \subseteq T$ such that $\mu_G(A,B) \le c$. If C' cover some min (A,B)-weak separator, then we are done. If Z' contains a non-terminal vertex of some brittle min (A,B)-weak separator, then we are also done. Assume otherwise. Let S be a brittle min (A,B)-weak separator in G. If Z is an (x,y)-separator for some distinct $x,y \in S$, then we are done with (A,B). Assume otherwise. There is i such that $S \cap X_i \neq \emptyset$. WLOG, we assume that $S \subseteq X_i \cup Z$ and $S \cap Z \subseteq T$. If $S \not\subseteq Y_i$, then there are $x,y \in S$ such that $x \in X_i$ and $y \notin Y_i$, and thus Π' splits S and we are done. Now, assume $S \subseteq Y_i$.

Let $A_i = A \cap X_i, B_i = B \cap X_i$. Denote $\partial X_i = Y_i - X_i$ as the set of *boundary* vertices. In G_i, we say that a boundary vertex $v \in \partial X_i$ is *A-proxy* if $A - A_i \neq \emptyset$, and there is an $(A - A_i, v)$-path in G that does not use X_i. Similarly, v is *B-proxy* if $B - B_i \neq \emptyset$ and there is an $(B - B_i, v)$-path in G that does not use X_i. Let ∂A_i be the set of A-proxy boundary vertices, and ∂B_i be the set of B-proxy boundary vertices. Let $A' = \partial A_i \cup A_i$ and $B' = \partial B_i \cup B_i$.

Claim A.1. *S is an (A', B')-weak separator in G_i.*

Proof. The proof is similar to that in Claim 6.7. □

Claim A.2. *An (A', B')-weak separator in G_i is an (A, B)-weak separator in G.*

Proof. Since S is an (A, B)-separator in G, every (A, B)-path in G must use S. Since $S \subseteq Y_i$, every (A, B)-path in G contains an (A', B')-subpath in G_i. Let S' be an (A', B')-weak separator in G_i. Suppose S' is not an (A, B)-weak separator in G. There must be an (A, B)-path P in G that does not use S'. Therefore, P contains a subpath from A' to B' in G_i that does not use S', a contradiction. $\square$

By Claim A.1 and Claim A.2, $\mu_{G_i}(A', B') = \mu_G(A, B) \leq c$. Since $S \cap Z \subseteq T$, $S - T_i = S - T$. Hence, Claim A.1 implies that $\mu^T_{G_i}(A', B') \leq |S - T_i| = |S - T| = \mu^T_G(A, B)$. Since S is a min (A', B')-weak separator in G_i and C_i does not cover any min (A, B)-weak separator in G_i, Π_i either splits or T_i-hits some min (A', B')-weak separator S' in G_i. By Claim A.2 and the fact that $\mu_{G_i}(A', B') = \mu_G(A, B)$, S' is a min (A, B)-weak separator in G. We prove that Π' either T-hits or splits S' in G, and we are done. If Π_i T_i-hits S', then $|(Q_{ij} \cap S') - T_i| \leq \mu^T_{G_i}(A, B) - 1 \leq \mu^T_G(A, B) - 1$ for all j. Therefore, $|((Q_{ij} - Z) \cap S') - T| \leq \mu^T_G(A, B) - 1$ for all j. Since $S' \subseteq Y_i$, $|((Q_{i'j} - Z) \cap S') - T| \leq \mu^T_G(A, B) - 1$ for all $i' \neq i$ and for all j. Therefore, Π' T-hits S'. If Π_i splits S' in G_i, then, for all j, $|S' \cap N_{G_i}[Q_{ij}]| \leq |S'| - 1$, and thus $|S' \cap N_G[Q_{ij} - Z]| \leq |S'| - 1$. Observe that $S' \subseteq Y_i$. Therefore, Π' splits S' in G.