Cactus Representations in Polylogarithmic Max-flow via Maximal Isolating Mincuts

Zhongtian He
Princeton University

Shang-En Huang*
Boston College

Thatchaphol Saranurak†
University of Michigan

Abstract

A *cactus* representation of a graph, introduced by Dinitz *et al.* in 1976, is an edge sparsifier of $O(n)$ size that exactly captures *all* global minimum cuts of the graph. It is a central combinatorial object that has been a key ingredient in almost all algorithms for the connectivity augmentation problems and for maintaining minimum cuts under edge insertions (e.g. [Naor *et al.* SICOMP'97], [Cen *et al.* SODA'22], [Henzinger ICALP'95]). This sparsifier was generalized to *Steiner cactus* for a vertex set T, which can be seen as a vertex sparsifier of $O(|T|)$ size that captures all partitions of T corresponding to a T-Steiner minimum cut, and also *hypercactus*, an analogous concept in hypergraphs. These generalizations further extend the applications of cactus to the Steiner and hypergraph settings.

In a long line of work on fast constructions of cactus and its generalizations, a near-linear time construction of cactus was shown by Karger and Panigrahi [SODA'09]. Unfortunately, their technique based on tree packing inherently does not generalize. The state-of-the-art algorithms for Steiner cactus and hypercactus are still slower than linear time by a factor of $\Omega(|T|)$ [Dinitz and Vainshtein STOC'94] and $\Omega(n)$ [Chekuri and Xu SODA'17], respectively.

We show how to construct both Steiner cactus and hypercactus using polylogarithmic calls to max flow, which gives the first almost-linear time algorithms of both problems. The constructions immediately imply almost-linear-time connectivity augmentation algorithms in the Steiner and hypergraph settings, as well as speed up the incremental algorithm for maintaining minimum cuts in hypergraphs by a factor of n.

The key technique behind our result is a novel variant of the influential *isolating mincut technique* [Li and Panigrahi FOCS'20, Abboud *et al.* STOC'21] which we called *maximal isolating mincuts*. This technique makes the isolating mincuts to be "more balanced" which, we believe, will likely be useful in future applications.

*Supported by NSF Grant No. CCF-2008422.
†Supported by NSF CAREER grant 2238138.

Contents

1 Introduction

In a weighted undirected $G = (V, E)$ with n vertices and m edges, a *global minimum cut* (or *mincut*, for short) is a cut with minimum weight separating some pair of vertices. More than 40 years ago, Dinitz *et al.* [DKL76] showed that, even though G may have as many as $\binom{n}{2}$ distinct mincuts, there exists a $O(n)$-size data structure called a *cactus representation* or simply a *cactus* of G that captures *all* mincuts of G in a strong way as follows.

A cactus of G is a tuple (H, ϕ) where H is a graph with $O(n)$ edges and $\phi : V(G) \to V(H)$ is a mapping such that, for any vertex set $A \subset V$, a mincut in G separates A and $V \setminus A$ iff a mincut in H separates $\phi(A)$ and $\phi(V \setminus A)$. Hence, H preserves all mincuts of G. Furthermore, H and its mincuts are highly-structured: H is a *cactus graph*[1] with edge weights from $\{1, 2\}$ and its mincut contains either two edges of weight 1 from the same cycle, or an edge of weight 2. Precisely by the ability of cactus to capture all mincuts via extremely simple structure, cactus has become the key ingredient in almost all algorithms for the connectivity augmentation problems [Gab91a, NGM97, BK00, CLP22] and for maintaining mincuts on graphs undergoing edge insertions [Hen97, DW98, GHT18]. Cactus can also be viewed as one of the first graph sparsifiers, predating other notions such as spanners [ADD+93], cut sparsifiers [BK96], and spectral sparsifiers [ST11].

Steiner cactus and hypercactus. To capture Steiner mincuts which are more general than global mincuts, a generalization of cactus called *Steiner cactus* was introduced by Dinitz and Vainshtein [DV94].[2] Recall that, for any vertex set $\mathcal{T} \subseteq V$, a *$\mathcal{T}$-Steiner mincut* is a cut with minimum weight separating some pair of vertices in $\mathcal{T}$. A *$\mathcal{T}$-Steiner cactus* of G is a tuple (H, ϕ) where H is a graph with $O(|\mathcal{T}|)$ edges and $\phi : \mathcal{T} \to V(H)$ is a mapping such that, for any $A \subset \mathcal{T}$, a $\mathcal{T}$-Steiner mincut in G separates A and $\mathcal{T} \setminus A$ iff a mincut in H separates $\phi(A)$ and $\phi(\mathcal{T} \setminus A)$. The graph H is also a cactus graph with the same simple structure as in a normal cactus. Note that, when $\mathcal{T} = V$, $\mathcal{T}$-Steiner cactus of G is simply a cactus of G.

The notion of Steiner cactus can be placed nicely into a more modern concept of *vertex sparsifiers* (also called *mimicking networks*) [HKNR98, Moi09, LM10, KR13, KR14]. The goal in this area is, given a terminal set $\mathcal{T} \subseteq V$, to construct a small graph H and a mapping ϕ such that, for every $A \subset \mathcal{T}$, $\text{mincut}_G(A, \mathcal{T} \setminus A) = \text{mincut}_H(\phi(A), \phi(\mathcal{T} \setminus A))$ where $\text{mincut}_G(X, Y)$ denotes the size of minimum cuts separating the sets X and Y in G. Unfortunately, there is a lower bound of $|E(H)| = \Omega(2^{|\mathcal{T}|})$ [KR13], and perhaps the most prominent open problem in this area is, when $(1 + \epsilon)$-approximation is allowed, whether the bound $|E(H)| = \text{poly}(|\mathcal{T}|)$ is possible. Interestingly, Steiner cactus implies that the linear bound $|E(H)| = O(|\mathcal{T}|)$ is actually possible without any approximation when we restrict ourselves to the sets $A \subset \mathcal{T}$ separated by some $\mathcal{T}$-Steiner mincuts.

Another generalization of cactus is called *hypercactus*. Cheng [Che99] (and later [FJ99]) showed an existence of hypercactus (H, ϕ) where H is a hypergraph of linear size $\sum_{e \in E(H)} |e| = O(n)$, yet, for every set $A \subset V$, a mincut in G separates A and $V \setminus A$ iff a mincut in H separates $\phi(A)$ and $\phi(V \setminus A)$. Similar to cactus, H is highly structured: each mincut in H contains either two size-2 edges of weight 1 from the same cycle of size-2 edges, or a (hyper)edge of weight 2. This compact data structure is perhaps even more surprising than cactus because the total size of a hypergraph can be exponential and a hypergraph may contain exponentially many distinct mincuts.[3]

[1] A *cactus graph* is a graph where each edge appears in at most one cycle.

[2] A Steiner cactus was introduced as a core structure inside a more involved structure called *carcass* [DV94, DN95, DV00]. The detailed proofs of the existence of Steiner cactus were given in [DN, Fle99, FF09]

[3] Consider a hypergraph $G = (V, E)$ with a single hyperedge containing all vertices, every cut $(S, V \setminus S)$ is a mincut.

Both Steiner cactus and hypercactus naturally extend the reach of applications of cactus. Cole et al. [CH03] used a Steiner cactus to speed up the algorithm for the *uniform survivable network* problem. They exploited the fact that Steiner cactus can be efficiently maintained under edge insertions between terminals. Hypercactus were used for hypergraph connectivity augmentation algorithms [Che99], and incremental algorithms for hypergraph mincuts [GK19].

Fast Algorithms. Because of the elegance and utility of cactus and its generalization, a long line of work has been devoted on fast algorithms for constructing them. Historically, cactus construction has been much more challenging than a more well-known problem of computing a single mincut (e.g. [Kar93, Kar00, KW96, MW00]), as we need to capture the structure of *all* mincuts.

Karzanov and Timofeev [KT86] outlined the first algorithm that constructs a cactus of a graph in $\Theta(n^3)$ time. Their algorithm was parallelized by Naor and Vazirani [NV91] and refined by Nagamochi and Kameda [NK94]. Later on, faster algorithms were developed [Gab91a, KS96, NNI00, Fle99] where the latter two algorithms ran in $\tilde{O}(nm)$ time. Finally, the line of work culminated in a near-linear $\tilde{O}(m)$-time algorithm by Karger and Panigrahi [KP09].[4]

The state-of-the-art constructions for Steiner cactus and hypercactus are significantly slower. Dinitz and Vainshtein [DV94, DV00] showed how to compute a $\mathcal{T}$-Steiner cactus using $\Theta(|\mathcal{T}|)$ max flow calls, which is $\Omega(m|\mathcal{T}|)$ time. Cole et al. [CH03] showed an $\tilde{O}(m + \lambda_G(T)n)$-time algorithm on unweighted graphs where $\lambda_G(T)$ denotes the value of $\mathcal{T}$-Steiner mincut, but in general $\lambda_G(T)$ may be big especially in weighted graphs. To construct a hypercactus of a hypergraph with total size $p = \sum_{e \in E} |e|$, the only algorithm with explicit running time was by Chekuri and Xu [CX17], which takes $O(pn + n^2 \log n)$ time.

To summarize, the fastest constructions for Steiner cactus and hypercactus are still slower than linear time by a factor of $\Omega(|T|)$ and $\Omega(n)$, respectively. This suggests a natural question of how fast one can compute them.

1.1 Our Results

We give a novel approach for constructing a cactus that generalizes to both Steiner cactus and hypercactus using polylogarithmic calls to max-flows. Let $\text{MaxFlow}(m)$ denote the running time of solving max flow in a graph with m edges. Since $\text{MaxFlow}(m) = m^{1+o(1)}$ by [CKL+22], we obtain the first almost-linear time algorithms of both problems. The Steiner cactus algorithm is summarized below.

Theorem 1.1. *There is a randomized Monte-Carlo algorithm that, given an undirected weighted graph G with m edges and a terminal set $\mathcal{T} \subseteq V$, the algorithm computes a $\mathcal{T}$-Steiner cactus of G in $\tilde{O}(\text{MaxFlow}(O(m)))$ time w.h.p.*

On hypergraphs, we even obtain an algorithm for computing a *Steiner hypercactus*, which naturally generalizes both a hypercactus and a Steiner cactus (see Definition 4.1 for the formal definition). The result is summarized below.

Theorem 1.2. *There is a randomized Monte-Carlo algorithm that, given a weighted hypergraph G with total size $p = \sum_{e \in E(G)} |e|$ and a terminal set $\mathcal{T} \subseteq V$, compute a $\mathcal{T}$-Steiner hypercactus of G in $\tilde{O}(\text{MaxFlow}(O(p)))$ time w.h.p.*

We note that if we consider set of hyperedges across a mincut, there are at most $O(n^2)$ distinct sets.

[4] Throughout the paper, $\tilde{O}(\cdot)$ hides $\text{polylog}(n)$ terms and $\widehat{O}(\cdot)$ hides additional $n^{o(1)}$ terms.

It is implicit in [FJ99, CX17] that a Steiner hypercactus admits polynomial time algorithms, but their construction requires at least n calls to max flows, which takes at least $\Omega(pn)$ time. Even for the more special problem of computing hypercactus, the best-known construction by [CX17] takes $\tilde{O}(pn)$ time.[5] Theorem 1.2 gives the first almost-linear time construction.

As discussed above, a cactus and its generalizations are central objects in many algorithms. Consequently, our almost-linear constructions immediately imply several applications. We defer the definition of the problems and the proofs of these applications to Appendix E.

Corollary 1.3. *There are randomized almost-linear time algorithms that can w.h.p. compute*

- *the optimal solution of the Steiner connectivity augmentation problem*[6]*, and*
- *the optimal value of the hypergraph +1-Steiner-connectivity augmentation problem.*

Corollary 1.3 improved a polynomial algorithm for the hypergraph +1-connectivity by Cheng [Che99] by speeding up it to almost-linear time and generalizing it to the Steiner version.

Lastly, we also improved the update time of the incremental algorithm for maintaining hypergraph mincuts from $O(\lambda n)$ [GK19] to of $\widehat{O}(\lambda)$.

Corollary 1.4. *There is an algorithm that, given an unweighted hypergraph $G = (V, E)$ undergoing hyperedge insertions, maintains a mincut in time $\widehat{O}(\lambda)$ amortized update time where λ denotes the mincut value at the end of the updates.*

1.2 Techniques

Below, we explain why the two most promising techniques in the literature fail to solve our problems, which motivates our new algorithmic tool called *maximal isolating mincuts*.

First Technical Barrier: Tree-packing Fails. Perhaps the most natural approach for devising fast algorithms for both Steiner cactus and hypercactus is to extend the techniques of the only known near-linear time algorithm by Karger and Panigrahi [KP09] for a normal cactus. However, their algorithm relied on the *tree packing technique* [Gab91b, Kar00], which requires solving the so-called *2-respecting mincuts* problem. Although they devised an ingenious way to deal with 2-respecting mincuts, at least quadratic time is likely required for k-respecting mincuts for any $k > 2$.[7]

Now, in the Steiner and hypergraph settings, it turns out that one needs to compute 4-respecting and $O(\log n)$-respecting mincuts, respectively, because fast algorithms for tree packing in these settings have worse quality by at least a factor of 2 [Meh88] and $\Omega(\log n)$ [CS07], respectively. Therefore, the tree-packing approach seems futile to us.

The same technical barrier was previously illustrated on the problem of computing a *single* mincut. Karger's [Kar00] near-linear time global mincut algorithm was based on tree-packing, and it took 20 years before almost-linear algorithms for Steiner mincut and hypergraph mincut [LP20, CQ21] were found using a very different technique, called *isolating mincuts*.

[5] Their result is based on the *MA-ordering* technique, which does not work well with Steiner mincuts.

[6] Very recently, Cen *et al.* [CHLP23] independently showed an almost-linear time algorithm for the Steiner connectivity augmentation problem.

[7] Abboud *et al.* [AKL+21] devised a new technique for dealing with k-respecting mincuts using $\log^{O(k)} n$ calls to max flow. The algorithm, however, is based on the isolating mincuts technique, which also fails to solve our problems. Furthermore, the exponential dependency on k makes their technique futile for hypergraphs where $k = O(\log n)$.

Second Technical Barrier: Minimal Isolating Mincuts Fails. The *isolating mincuts* technique was recently discovered by [LP20, AKT21]. They show that given a graph $G = (V, E)$ and a terminal set $\mathcal{T} \subseteq V$, one can compute a t-mincut of $\mathcal{T}$ (i.e., a minimum cut separating t from $\mathcal{T} \setminus t$) for all $t \in \mathcal{T}$ using logarithmic calls to max flow, instead of $|\mathcal{T}|$ many calls. In fact, they showed how to compute a *minimal* t-mincut of $\mathcal{T}$ for all $t \in \mathcal{T}$, i.e., the unique mincut separating t from $\mathcal{T} \setminus t$ that is "closest" to t. These cuts are called the *minimal isolating mincuts of* $\mathcal{T}$. The technique instantly became very influential and found many applications [AKT21, LNP+21, CQ21, LP21, MN21, CLP22, LNPS22, LPS22, AKT22, AKL+21], many of which crucially exploit the minimal property of these isolating mincuts.

Unfortunately, minimal isolating mincuts are ineffective for constructing a cactus: it cannot even distinguish a cycle from a clique! More precisely, consider a clique K and a cycle C with n vertices. Scale the weight edges so that the weighted degree of each node of the two graphs agrees. All mincuts of K consist of n singletons cut, while all mincuts of C contain all $\binom{n}{2}$ arcs of C. Now, for every vertex set $\mathcal{T}$, minimal isolating mincuts of $\mathcal{T}$ in both K and C are always a collection of singleton cuts. That is, minimal isolating mincuts alone fail to distinguish the mincut structures between the clique K and the cycle C.

Our New Tool: Maximal Isolating Mincuts. The above counter-example naturally suggests we consider *maximal* isolating mincuts. That is, given a terminal set $\mathcal{T}$, compute a maximal t-mincut of $\mathcal{T}$ for all $t \in \mathcal{T}$, which is the unique mincut separating t from $\mathcal{T} \setminus t$ that is "furthest" from t.

At first glance, maximal isolating mincuts seem unsuitable for almost-linear time algorithms because it is not even clear whether these cuts admit a near-linear space representation. In contrast, minimal isolating mincuts consist of disjoint vertex sets and so have linear size. For example, when $\mathcal{T} = \{s, t\}$, it is easy to see that the two maximal mincuts can overlap, and almost all vertices may be in both cuts. Therefore, it is conceivable that maximal isolating mincuts of $\mathcal{T}$ requires $\Omega(n|\mathcal{T}|)$ space, quadratic in the worse case.

Perhaps surprisingly, we show that these cuts' total size is linear and can also be computed using polylogarithmic calls to max flow. We devote Section 3 to proving this structural result.

Let us reconsider the toy problem of "cycle vs. clique" above. Indeed, maximal isolating mincuts can resolve this problem. Suppose we sample a terminal set $\mathcal{T} \subset V$ and then compute maximal isolating mincuts $\mathcal{T}$. While all the minimal isolating mincuts in the clique remain singleton cuts, some maximal isolating mincuts will be non-singleton cuts in the cycle. So, in this simple example, we can distinguish the two graphs.

Cactus via Maximal Isolating Mincuts. It turns out that maximal isolating mincuts are powerful enough to capture *all* mincuts. By highly exploiting their structure, we show in Section 4 how to construct Steiner cactus using polylogarithmic maxflow calls. At a high level, our algorithm significantly improves the divide-and-conquer approach by [CX17] with linear recursion depth to only logarithmic. To achieve this, the high-level idea is, in the divide step, our maximal-isolating-cut-based approach can *split* the graph such that every part has size smaller by a constant factor, except at most one part that we guarantee that no more recursion is needed. (See Figure 2 for an illustration.) Hence, the recursion depth is logarithmic. In contrast, the algorithm of [CX17] cannot certify this and might need to recurse on the biggest part for $\Omega(n)$ rounds.

First Challenge in Hypergraphs: Defining Maximal Isolating Mincuts Let us discuss the technical challenges in generalizing our techniques to hypergraphs. First, it is tricky even to define the notion of maximal isolating mincuts for hypergraphs. A natural extension is a partition of vertices $(X, V \setminus X)$ that separates a specified terminal $t \in X \cap \mathcal{T}$ from other terminals such that $|X|$ is maximized while the number of boundary edges $|\partial X|$ is minimized. However, the total output size of maximal isolating mincuts can be quadratic under this definition. Hence, it is useless for almost-linear time algorithms. For example, consider a hypergraph $G = (V, \{V\})$ with only one single hyperedge containing all the vertices. Designate half of the vertices as terminals, i.e., $\mathcal{T} \subset V$ and $|\mathcal{T}| = |V|/2$. In this case, every maximal isolating mincut has size $|V| - |\mathcal{T}| + 1 = \Omega(|V|)$.

It turns out the "right" definition is the following tweak — we require that all mincuts $(X, V \setminus X)$ the algorithm is considering must be *connected at the X side*, that is, after removing the boundary edges ∂X, all vertices in X must still be connected. Under this definition, in Section 5.1, we show that all nice structural results we had on graphs transfer to hypergraphs. In particular, we can bound the total size of maximal isolating mincuts to be linear. As a sanity check, all maximal isolating mincuts in the above example are single vertices under the new definition, thereby the total output size becomes $O(|V|)$.

Second Challenge in Hypergraphs: Computing Hypercactus The second significant challenge is because a hypercactus contains hyperedges of rank higher than two. Without very careful treatment, our divide-and-conquer algorithms, including previous algorithms by [CX17], will split these higher-rank hyperedges of rank r in the divide step into $\Omega(r)$ pieces. Now, in the conquer step, the algorithm will need to merge them back and each merging step requires at least linear time (in fact, one max flow call) because we need to perform some test to know the topology of the gluing parts. This incurs the running time of $\Omega(pr) = \Omega(pn)$ where p is the total size of the hypergraph, which is too slow.

We completely bypass this difficulty showing that our divide-and-conquer algorithm simply never splits these higher-rank hyperedges in a non-trivial way! This is done by again exploiting the "right" definition of the maximal isolating mincuts described above. Compared with the algorithm by [CX17], our final algorithm in Section 5.2 is arguably simpler as we do not perform the test related to higher-rank hyperedges and runs in almost-linear time.

2 Preliminaries

Let $G = (V, E, w)$ be an undirected, connected, and weighted graph with positive edge weights $w : E \to \mathbb{R}^+$. A cut is a partition $(X, V \setminus X)$, when the context is clear we use X (or $V \setminus X$) representing the cut $(X, V \setminus X)$ for brevity. For any subsets $X, Y \subseteq V$ we define the *value* between X and Y to be $\mathcal{C}(X, Y) = \sum_{u \in X, v \in Y} w(u, v)$. When $Y = V \setminus X$ we simply denote $\mathcal{C}(X, V \setminus X)$ by $\mathcal{C}(X)$. A nonempty subset $X \subset V$ is said to be a (global) *mincut* if $\mathcal{C}(X) = \min_{S: \emptyset \subset S \subset V} \mathcal{C}(S)$. The set of boundary edges incident to X is denoted by ∂X. For two disjoint subsets of vertices A and B, a cut $(X, V \setminus X)$ *separates* A and B if $X \subseteq A$ and $Y \subseteq B$.

We say two subsets of vertices X and $Y \subseteq V$ are *crossing* if all of $X \cap Y$, $X \setminus Y$, $Y \setminus X$, and $V \setminus (X \cup Y)$ are nonempty. The value function $\mathcal{C}$ satisfies *submodularity* and *posi-modularity* (see [NI00]) on crossing subsets.

Lemma 2.1 ([NI00])**.** *Let $X, Y \subseteq V$ be subsets that are crossing. Then,*

- *(Submodularity)* $\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cap Y) + \mathcal{C}(X \cup Y)$, *and*
- *(Posi-modularity)* $\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \setminus Y) + \mathcal{C}(Y \setminus X)$. □

Let $X \subseteq V$ be any subset of vertices, we define the *contracted graph* G/X to be the graph obtained from G by (1) contracting all vertices in X into one vertex, (2) replacing all multi-edges with a single edge with the corresponding edge weight, and finally (3) removing all self-loops.

Steiner Mincuts and t-Isolating Cuts. Let $\mathcal{T} \subseteq V$ be a terminal vertex set of size at least 2. For any proper subset $A \subsetneq \mathcal{T}$, an *A-cut* of $\mathcal{T}$ is a cut that separates A and $\mathcal{T} \setminus A$. An *A-mincut* of $\mathcal{T}$ is a minimum valued A-cut of $\mathcal{T}$, denoted by the vertex set X_A. Conveniently, for any vertex $t \in \mathcal{T}$ we define a cut X_t to be *t-isolating mincut* of $\mathcal{T}$ if X_t is a $\{t\}$-mincut of $\mathcal{T}$. A *$\mathcal{T}$-Steiner mincut* is defined to be a minimum valued A-mincut among all proper subsets $A \subsetneq \mathcal{T}$. The value of a $\mathcal{T}$-Steiner mincut on the graph G is denoted as $\lambda_G(\mathcal{T})$.

A-mincuts of $\mathcal{T}$ are not necessarily unique. Fortunately, with the submodularity and posi-modularity mentioned in Lemma 2.1, these A-mincuts behave similarly to global mincuts in the sense that there is a unique *minimal* and *maximal* A-mincut of $\mathcal{T}$.

Definition 2.2 (Maximal and Minimal A-Mincuts of $\mathcal{T}$)**.** We say that an A-mincut X_A of $\mathcal{T}$ is *maximal* (resp. minimal), if for any other A-mincut X'_A of $\mathcal{T}$, we have $X_A \supseteq X'_A$ (resp. $X_A \subseteq X'_A$).

Isolating Mincuts. Given a graph G and a set of terminal vertex $\mathcal{T}$, the *isolating mincuts of* $\mathcal{T}$ is a collection of t-isolating mincuts that separates each single terminal vertex $t \in \mathcal{T}$ from $\mathcal{T} \setminus \{t\}$. Li and Panigrahi [LP20] gave an algorithm that computes a minimal isolating cut efficiently.

Theorem 2.3 ([LP20, Isolating Cut Lemma])**.** *Given a graph $G = (V, E)$ and any terminal set $\mathcal{T} \subseteq V$, there is an algorithm that returns the minimal t-isolating mincuts for all $t \in \mathcal{T}$ in $O(\log |\mathcal{T}| \cdot \text{MaxFlow}(2n, 2m))$ time.*

The function $\text{MaxFlow}(x, y)$ denotes the time needed for solving an st-maxflow instance with x vertices and y edges. We assume that the function $\text{MaxFlow}(x, y)$ is $\Omega(x + y)$, is non-decreasing, and is superadditive.[8]

3 Maximal Isolating Mincuts

The maximal minimum isolating cut problem states that, given a graph $G = (V, E)$ with n vertices and m edges and a terminal set $\mathcal{T}$, the goal is to obtain the maximal isolating mincut X_v for each terminal $v \in \mathcal{T}$. In this section, we give an efficient algorithm for solving the maximal isolating mincuts problem in almost linear time, which summarizes as the following Theorem 3.1.

Theorem 3.1. *There exists an algorithm that, given an undirected weighted graph $G = (V, E)$ and a terminal set $\mathcal{T} \subseteq V$, in $O(\log |\mathcal{T}| \cdot \text{MaxFlow}(3n, 4m))$ time computes the maximal isolating mincut of all terminals $v \in \mathcal{T}$ with respect to $\mathcal{T}$.*

[8] An integral function $f(x, y)$ is said to be *superadditive* if for any $x_1, y_1, x_2, y_2 \in \mathbb{N}$ we have $f(x_1 + x_2, y_1 + y_2) \geq f(x_1, y_1) + f(x_2, y_2)$. The function is *non-decreasing* if both $f(x + 1, y) \geq f(x, y)$ and $f(x, y + 1) \geq f(x, y)$ holds.

Key Insight. The crux of our maximal isolating mincut algorithm is an observation[9] to any set of three pairwise crossing mincuts to three disjoint subsets of $\mathcal{T}$ — their intersection is always an empty set due to *only* posi-modularity but not submodularity.

To formally prove this, we first state a standard fact about posi-modularity in Lemma 3.2 (see its proof in Appendix A.1). Then, we state the key observation as the Pairwise Intersection Only Lemma below.

Lemma 3.2 (Disjoint & Posi-modularity)**.** *Let $A, B \subseteq \mathcal{T}$ be two nonempty subsets of terminals with $A \cap B = \emptyset$. Let X_A (resp. X_B) be an A-mincut (resp. B-mincut) of $\mathcal{T}$. Then, $X_A \setminus X_B$ is an A-mincut of $\mathcal{T}$, and $X_B \setminus X_A$ is a B-mincut of $\mathcal{T}$.*

Lemma 3.3 (Pairwise Intersection Only)**.** *Given a graph G, let $A, B, C \subseteq \mathcal{T}$ be three disjoint nonempty subsets of terminals. Let $X_A, X_B, X_C \subseteq V$ be any A-mincut, B-mincut, and C-mincut of $\mathcal{T}$ respectively. Then $X_A \cap X_B \cap X_C = \emptyset$.*

Proof. The proof is only interesting whenever the intersection of any two mincuts is non-empty (i.e. crossing). Thus, without loss of generality, we assume that $X_A \cap X_B \neq \emptyset$, $X_B \cap X_C \neq \emptyset$, and $X_C \cap X_A \neq \emptyset$.

Define $X'_A = (X_A \setminus X_B) \setminus X_C$, $X'_B = (X_B \setminus X_C) \setminus X_A$ and $X'_C = (X_C \setminus X_A) \setminus X_B$. These sets are non-empty since $A \subseteq X'_A$, $B \subseteq X'_B$, and $C \subseteq X'_C$. By posi-modularity (Lemma 3.2) we know that X'_A, X'_B, and X'_C are also A-mincut, B-mincut, and C-mincut of $\mathcal{T}$ respectively, and thus

$$\underbrace{(\mathcal{C}(X_A) + \mathcal{C}(X_B) + \mathcal{C}(X_C)) - (\mathcal{C}(X'_A) + \mathcal{C}(X'_B) + \mathcal{C}(X'_C))}_{\text{(LHS)}} = 0. \qquad (\star)$$

Assume for contradiction that $X := X_A \cap X_B \cap X_C \neq \emptyset$. We now claim that there is no edge from X to $X_A \setminus X$. Note that with the assumption that G is connected, we have $\mathcal{C}(X, V \setminus X_A) > 0$. The claim implies that

$$\mathcal{C}(X_A) = \mathcal{C}(X_A \setminus X) - \mathcal{C}(X_A \setminus X, X) + \mathcal{C}(X, V \setminus X_A) > \mathcal{C}(X_A \setminus X),$$

which contradicts the fact that X_A is A-mincut.

We shall prove the claim by two steps: (1) every edge in the graph contributes to non-negative values to the left-hand side (LHS) of ($\star$). Therefore, any edge that contributes to a positive amount to LHS should not exist in G by Equation ($\star$). (2) For any boundary edge $(u, v) \in \partial X$ with $u \in X$ and $v \notin X$, we have $v \notin X_A \cup X_B \cup X_C$, which implies the claim.

1. We consider the cases that e contributes to how many terms of $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$ and $\mathcal{C}(X'_C)$. Since X'_A, X'_B, and X'_C are disjoint and an edge e incidents to two vertices, e contributes to at most two of the three terms. There is nothing to prove if e contributes to 0 of them.

   If e contributes to one of them, without loss of generality $\mathcal{C}(X'_A)$. Then e contains some vertex in X'_A, and also some vertex v in $V \setminus X'_A = (V \setminus X_A) \cup X_B \cup X_C$. Therefore v is in either $V \setminus X_A$, X_B or X_C, hence also contributes to either $\mathcal{C}(X_A), \mathcal{C}(X_B)$ or $\mathcal{C}(X_C)$ respectively.

   Otherwise e contributes to two of them, without loss of generality $\mathcal{C}(X'_A)$ and $\mathcal{C}(X'_B)$, then e contains some vertex $u \in X'_A$ and $v \in X'_B$. By the definition of X'_A and X'_B, we have $u \in X_A$, $u \notin X_B$, $v \in X_B$, $v \notin X_A$. So e also contributes to $\mathcal{C}(X_A)$ and $\mathcal{C}(X_B)$.

[9] A similar observation can also be found in Dinitz and Vainshtein [DV94, 3-Star Lemma].

2. We prove (2) using (1) by showing that the contribution to LHS is positive if $u \in X$ and $v \in (X_A \cup X_B \cup X_C) \setminus X$. Again we consider the cases that e contributes to how many terms of $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$, and $\mathcal{C}(X'_C)$. Note that e can not contribute to more than one of them since X'_A, X'_B, and X'_C are disjoint and one of the endpoints $u \in X$ is in neither of them.

   If e contributes to none of them, then we need to show that e contributes to at least one of $\mathcal{C}(X_A), \mathcal{C}(X_B)$, and $\mathcal{C}(X_C)$. This claim is directly from the fact that $X = X_A \cap X_B \cap X_C$, $u \in X$, and $v \notin X$.

   Otherwise, e contributes to one of them, without loss of generality $\mathcal{C}(X'_A)$. By the definition of X'_A, we have $v \notin X_B$ and $v \notin X_C$. Therefore e also contributes to $\mathcal{C}(X_B)$, and $\mathcal{C}(X_C)$. □

Bounding the output size. Before we describe our algorithm, we emphasize that the total size of all maximal v-isolating mincuts is $O(n)$, as a consequence of Lemma 3.3:

Lemma 3.4. *Let G be a graph and $\mathcal{T}$ be a set of terminals. For each $v \in \mathcal{T}$, let X_v be any v-isolating mincut. Then, $\sum_{v \in \mathcal{T}} |X_v| \leq 2n$.*

Proof. Every vertex $u \in V$ belongs to at most two isolating mincuts. Suppose by contradiction that there exist three terminal vertices v_1, v_2, and v_3 such that $u \in X_{v_1} \cap X_{v_2} \cap X_{v_3}$. By Lemma 3.3 we know that $X_{v_1} \cap X_{v_2} \cap X_{v_3} = \emptyset$, a contradiction. Hence, a counting argument shows that $\sum_{v \in T} |X_v| \leq 2n$. □

We remark that for our purpose we apply Lemma 3.4 where X_v is the maximal v-isloating mincut for all $v \in \mathcal{T}$. Lemma 3.4 implies that the total output size is $O(n)$. Thus, all maximal v-isolating mincuts can be stored explicitly. Now, we are ready for introducing the algorithm.

A Divide and Conquer Algorithm. The existing algorithms [LP20, CQ21] for finding minimal isolating mincuts use *one-shot recursion* by first computing $O(\log |\mathcal{T}|)$ cuts on G. These cuts partition the vertex set into $O(|\mathcal{T}|)$ subsets, and isolate every terminal from $\mathcal{T}$. The minimal t-isolating mincuts can then be obtained within each part (and contracting everything outside the part). This does not work for us! The most evident reason is that the maximal t-isolating mincuts may not be disjoint.

Instead of using one-shot recursion, we can also consider a divide and conquer algorithm that is equivalent to the existing algorithms. The algorithm considers one cut at a time. Each cut splits the terminal set (and the vertex set) into two parts, creating two subproblems. Each subproblem is obtained by contracting all vertices on one side of the cut into a single vertex. The subsequent cuts are now affecting both subproblems, splitting each subproblem into another two subproblems, and so on and so forth.

Our algorithm solves the maximal isolating mincut problem using a divide and conquer approach very similar to the algorithm described above, but with a slight twist. In our algorithm, every subproblem is derived from G via contractions, and there will be *at most one* contracted vertex in each subproblem. We call such a special vertex p the *pivot* of a subproblem. Notice that $p = \texttt{null}$ when the algorithm first enters the recursion, as there is no contracted vertex. Throughout the execution, the algorithm recursively splits the terminal set $\mathcal{T} \setminus \{p\}$ arbitrarily into two halves $\mathcal{T} \setminus \{p\} = A \cup B$. Then, the algorithm computes X_A (the maximal A-mincut of $\mathcal{T}$) and X_B (the maximal B-mincut of $\mathcal{T}$) by solving *two* s-Maximal st-Mincut instances (which can be solved using one st-MaxFlow and a linear time post processing).

We observe (via submodularity, see Lemma 3.5) that for each terminal $v \in A$, the maximal v-isolating mincut X_v must be fully contained in X_A, i.e., $X_v \subseteq X_A$. Hence, in order to find out the maximal v-isolating mincuts for all vertices $v \in A$, it is safe to contract everything outside X_A and recursively compute maximal isolating mincuts on the contracted graph $G_A := G/(V \setminus X_A)$. Notice that the pivot p as well as all terminal vertices in B are outside of X_A, so within the contracted graph G_A the contracted vertex p_A shall be considered as a terminal vertex in the subproblem.

Similarly, the algorithm contracts everything outside X_B which leads to the pivot vertex p_B, and recursively computes maximal isolating mincuts on $G/(V \setminus X_B)$ too. This divide and conquer procedure continues until the number of terminal vertices becomes a constant, where the maximal v-isolating mincut can be computed for each vertex $v \in \mathcal{T}$ individually using a max-flow. The algorithm is summarized in Algorithm 1.

Algorithm 1 Maximal isolating mincuts.

```
1: procedure MaxIsoMincut(G, T)
2:     Call MaxIsoMincutWithPivot(G, T, null).
3: end procedure
4: procedure MaxIsoMincutWithPivot(G, T, p)
5:     if |T| ≤ 4 then                                          ▷ Base case.
6:         Obtain the maximal v-isolating mincut X_v for each v ∈ T.   ▷ Use MaxFlow.
7:     else
8:         Partition T \ {p} arbitrarily into two similarly sized sets A ∪ B = T \ {p}.
9:         Compute X_A, the maximal A-mincut of T.              ▷ Use MaxFlow.
10:        Compute X_B, the maximal B-mincut of T.              ▷ Use MaxFlow.
11:        Let G_A ← G/(V \ X_A) where V \ X_A gets contracted to p_A.
12:        Let G_B ← G/(V \ X_B) where V \ X_B gets contracted to p_B.
13:        Invoke MaxIsoMincutWithPivot(G_A, A ∪ {p_A}, p_A).
14:        Invoke MaxIsoMincutWithPivot(G_B, B ∪ {p_B}, p_B).
15:    end if
16: end procedure
```

Analysis We now prove the correctness and the runtime. As mentioned above, the correctness of Algorithm 1 depends on a standard fact on submodularity on two nesting subsets (for completeness see its proof in Appendix A.2):

Lemma 3.5 (Nesting & Submodularity)**.** *Let G be a graph and let $\mathcal{T}$ be the set of terminals. Consider two nonempty subsets A and B of terminals such that $A \subseteq B \subsetneq \mathcal{T}$. Let X_A (resp. X_B) be any A-mincut (resp. B-mincut) of $\mathcal{T}$. Then, $X_A \cap X_B$ is a A-mincut of $\mathcal{T}$. Respectively, $X_A \cup X_B$ is a B-mincut of $\mathcal{T}$.*

Consider the recursion tree throughout executing Algorithm 1. We say that a subproblem is a *leaf* if Algorithm 1 is executed for the subproblem and thus no further recursions are invoked.

Lemma 3.6 (Correctness)**.** *Fix a terminal vertex $v \in \mathcal{T}$. There is a unique (leaf) subproblem where the maximal isolating mincut for v is computed. Let $\hat{X}_v$ be the cut returned at Line 6 for vertex v. Then $\hat{X}_v = X_v$ is the maximal v-isolating mincut on the graph G.*

Proof. Fix $v \in \mathcal{T}$. Since in each divide step, exactly one subproblem contains v so there will be exactly one leaf subproblem where Line 6 is invoked for v. Clearly $\hat{X}_v$ does not contain any

contracted vertex (since the only contracted vertex in the subproblem is the pivot.) Now, to show that $\hat{X}_v = X_v$, it suffices to show that in each divide step, all vertices in X_v are not contracted. Indeed, by submodularity in Lemma 3.5, suppose without loss of generality that $v \in A$ then we must have $X_v \subseteq X_A$ (otherwise X_A is not a maximal isolating mincut of A since $X_v \cup X_A$ is a larger-sized isolating mincut of A.) □

Lemma 3.7 (Runtime)**.** *MaxIsoMincut*$(G, \mathcal{T})$ *runs in* $O(\log |\mathcal{T}| \cdot \text{MaxFlow}(3n, 4m))$ *time.*

Proof. It suffices to bound the sum of graph sizes in all subproblems throughout the execution of Algorithm 1. First of all, the maximum depth of the recursion tree is $\lceil \log |\mathcal{T}| \rceil$ since in each recursive call the number of non-pivot terminals is reduced to half. In addition, the number of subproblems in each recursion depth i is at most $\min\{2^i, |\mathcal{T}|\}$.

Now, we focus on a particular recursion depth $i > 0$. Let $\{(G_j, \mathcal{T}_j, p_j)\}$ be all the subproblems whose recursion depth is i. We observe that all terminals except pivot go to exactly one subproblem so the subsets $\mathcal{T}'_j := \mathcal{T}_j \setminus \{p_j\}$ are disjoint. Moreover, by Lines 11-12 we know that for each j, removing the pivot p_j from G_j it is exactly the maximal $\mathcal{T}'_j$-mincut of $\mathcal{T}$ on G.

Using the Pairwise Intersection Only Lemma (Lemma 3.3), we are able to conclude that every vertex in the input graph G occurs in at most two subproblems at recursion depth i. Therefore, the total number of vertices across all subproblems at depth i is $\sum_j |V(G_j)| \le 2n + \min\{2^i, |\mathcal{T}|\} \le 2n + |\mathcal{T}| \le 3n$. To analyze the total number of edges across all subproblems at depth i, we notice that each edge has at least one non-pivot endpoint. Thus, the total number of edges can be bounded by the sum of all vertex degrees $\sum_j |E(G_j)| \le 4m$.

Finally, in each subproblem $(G', \mathcal{T}')$, where the graph G' has n' vertices and m' edges, the algorithm computes the maximal A-mincut of $\mathcal{T}$ (denoted by X_A) and the maximal B-mincut of $\mathcal{T}$ (denoted by X_B) by the following steps. First, the algorithm creates a flow graph G'_{flow} where the source vertex s is obtained by merging all vertices in A and the sink vertex t is obtained by merging all vertices in B. This graph has at most n' vertices and at most m' (undirected) edges since G' is formed from the input graph G by a sequence of contraction. Then, the algorithm finds any st-MaxFlow f within $\text{MaxFlow}(n', m')$ time. Finally, the algorithm examines the residual graph $G'^{(f)}_{\text{flow}}$ in $O(n' + m')$ time: X_A is exactly the set of vertices that do *not* reach t and X_B is exactly the set of vertices that are *not* reachable from s.

Therefore, by summing up the runtime per recursion depth, we obtain an upper bound to the desired total runtime $O(\log |\mathcal{T}| \cdot \text{MaxFlow}(3n, 4m))$. □

Proof of Theorem 3.1. Theorem 3.1 follows directly by Algorithm 1, Lemma 3.6, and Lemma 3.7. □

4 Steiner Cactus Construction

In this section, we apply the maximal isolating mincut algorithm from Section 3 to construct a a *Steiner cactus representation* that succinctly represents all $\mathcal{T}$-mincuts on a given graph G with terminal set $\mathcal{T}$ using $O(|\mathcal{T}|)$ space.

Definition 4.1 (Steiner Cactus, see also [DV94, CX17])**.** Given a graph G and a terminal set $\mathcal{T}$, a *$\mathcal{T}$-Steiner cactus* (H, ϕ) is a weighted cactus graph H with a mapping $\phi : \mathcal{T} \to V(H)$ such that (1) edges in a cycle have weights $\lambda_G(\mathcal{T})/2$ and edges not in a cycle have weights $\lambda_G(\mathcal{T})$, (2) an A-mincut of $\mathcal{T}$ is a $\mathcal{T}$-Steiner mincut if and only if a global mincut on H separates $\phi(A)$ and $\phi(\mathcal{T} \setminus A)$.

We say that a node $v \in V(H)$ in a cactus is *non-empty* if there exists a terminal t where $\phi(t) = v$. There may exist *empty* nodes in H. Notice that cactus and Steiner cactus representations of a graph may not be unique.[10] Our Steiner cactus algorithm is now summarized below as Theorem 4.2.

Theorem 4.2. *Let G be a graph with n vertices and m edges. Let $\mathcal{T}$ be a set of terminals. There exists a randomized Monte Carlo algorithm such that, with probability $1 - 8n^{-10}$, the algorithm correctly computes a $\mathcal{T}$-Steiner cactus in $O((\log^4 n) \cdot \text{MaxFlow}(3n, 4m + 8n \log |\mathcal{T}|))$ time.*

4.1 Divide and Conquer Approach: Prior Works

Chekuri and Xu's algorithm [CX17] finds a linear-sized *hypercactus representation* that represents all (global) mincuts on a hypergraph. This hypercactus representation degenerates to a cactus representation on a normal graph and also in the Steiner setting. We now briefly describe their framework in terms of constructing a $\mathcal{T}$-Steiner cactus.

The main idea of Chekuri and Xu's algorithm is to successively find *$\mathcal{T}$-splits* — $\mathcal{T}$-mincuts that have at least two terminal vertices on both sides. After obtaining a $\mathcal{T}$-split $(X, V \setminus X)$, a *simple refinement* conceptually *decomposes* the graph G into two graphs G_1 and G_2, where each of them is obtained from a copy of G with all vertices from one side (either X or $V \setminus X$) are contracted. Notice that the contracted vertices will be treated as terminals in the decomposed graphs, which we call *anchor vertices*.

Definition 4.3. A *split* (or a *$\mathcal{T}$-split*) of a terminal set $\mathcal{T}$ on a graph G is a $\mathcal{T}$-mincut $(X, V \setminus X)$ such that both X and $V \setminus X$ have at least two terminal vertices, i.e., $|\mathcal{T} \cap X|, |\mathcal{T} \setminus X| \geq 2$.

Definition 4.4 (Simple Refinement and Anchor Vertex)**.** Fix a graph $G = (V, E)$ and a terminal set $\mathcal{T}$. We say that $\{(G_1, \mathcal{T}_1), (G_2, \mathcal{T}_2)\}$ is a *simple refinement* of G if G_1 and G_2 are graphs obtained through a $\mathcal{T}$-split $(X, V \setminus X)$ of G and a new *anchor vertex*[11] a as follows.

- $G_1 := G/(V \setminus X)$ such that $V \setminus X$ gets contracted to a.
- $\mathcal{T}_1 := (\mathcal{T} \cap X) \cup \{a\}$.
- $G_2 := G/X$ such that X gets contracted to a.
- $\mathcal{T}_2 := (\mathcal{T} \cap (V \setminus X)) \cup \{a\}$.

Chekuri and Xu's algorithm maintains a *decomposition* $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ (initialized with the input graph $\{(G, \mathcal{T})\}$), iteratively finds a $\mathcal{T}_i$-split to any graph $(G_i, \mathcal{T}_i) \in \mathcal{G}$, and replaces the graph G_i with its simple refinements. Since each simple refinement creates an anchor vertex that appears in both decomposed graphs, at any time, the decomposition $\mathcal{G}$ admits a *decomposition tree*.

Definition 4.5 (Decomposition)**.** Fix a graph G and a terminal set $\mathcal{T}$. A *decomposition* $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ is a collection of graphs and terminal vertices obtained by performing an arbitrary sequence of simple refinements.

Finally, the algorithm halts when no splits exist in the current decomposition. In this case, the decomposition is called a *prime decomposition*.

[10] See the work of Nagamochi and Kameda [NK94] for canonical cactus representations of a graph.

[11] In [CX17] the authors called these vertices *marker vertices*.

Definition 4.6 (Prime Decomposition)**.** Fix a graph G and a terminal set $\mathcal{T}$. We say that a decomposition $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ is *prime* if each graph G_i does not contain a $\mathcal{T}_i$-split.

In Chekuri and Xu's algorithm, they add a post-processing step turning a prime decomposition into a *canonical decomposition* (see also [Che99, Cun83]), where every mincut of G can be found in exactly one graph from the canonical decomposition. Indeed, it is possible for some $\mathcal{T}$-mincuts of G not being preserved anymore in a prime decomposition. For example, if the algorithm decomposes a graph using a $\mathcal{T}$-split, then all the $\mathcal{T}$-mincuts that cross with that split no longer exist in the simple refinement. The only guarantee to any decomposition $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ is that every $\mathcal{T}_i$-mincut in G_i corresponds to some $\mathcal{T}$-mincut in G (by "expanding" the anchor vertices with the terminal vertices in $\mathcal{T}$). Fortunately, all $\mathcal{T}$-mincuts that the algorithm has missed in one divide and conquer step belong to the same cycle on a cactus representation of $\mathcal{T}$. In Section 4.4 we show that even without the post-processing step, we are still able to efficiently *glue* the cactus of decomposed graphs (via anchor vertices) such that a cycle on a cactus representation can still be constructed. Thus, all $\mathcal{T}$-mincuts are preserved.

However, iteratively finding splits and repeatedly invoking simple refinements have a worst case $\Omega(m|\mathcal{T}|)$ runtime, which is too slow. This worst case occurs when the splits used for simple refinements were imbalanced. In the rest of the section, we resolve this issue via maximal isolating mincuts, obtaining an algorithm for Steiner cactus in poly-logarithmic max-flow time.

4.2 Our Divide and Conquer Framework via a Sequence of Splits

Our divide and conquer algorithm is based on the idea of Chekuri and Xu [CX17], where the goal is to output a prime decomposition through a series of simple refinements. However, in each subproblem, instead of seeking one split at a time, our algorithm uses multiple splits in G and generates a *good* decomposition (see Definition 4.8) with high probability. The guarantee of a good decomposition leads to an $O(\log |\mathcal{T}|)$ upper bound to the recursion depth, achieving a poly-logarithmic max-flow runtime.

In the rest of this subsection, we formulate a divide and conquer framework (see Algorithm 2) for computing a $\mathcal{T}$-Steiner cactus. The implementation of this framework has to overcome two non-trivial challenges, namely (1) computing a collection of splits that generates a good decomposition (Lemma 4.10) and (2) merging the sub-cactus returned from the subproblems into a $\mathcal{T}$-Steiner cactus (Lemma 4.11). We overcome both challenges using our maximal isolating mincut algorithm and describe the details in Section 4.3 and Section 4.4. By assuming Lemmas 4.10 and 4.11, we establish a proof to Theorem 4.2 at the end of this subsection.

Preprocessing. To enable the power of the maximal isolating mincut algorithm, we rely on the following two handy properties after preprocessing:

1. We may assume that we have already known the value of the Steiner mincut $\lambda_G(\mathcal{T})$.
2. We may assume that for any two terminal vertices u and $v \in \mathcal{T}$, there exists a $\mathcal{T}$-Steiner mincut that separates u and v.

These two assumptions can both be achieved using the isolating cut algorithm from Li and Panigrahi [LP20]. The first assumption can be made directly via an almost-linear time algorithm [CQ21] that computes a $\mathcal{T}$-mincut and its value. The second assumption can be made by preprocessing the graph with a "λ-connected component algorithm" implicitly mentioned in Li and Panigrahi [LP21].

We summarize the second preprocessing step below in Lemma 4.7 and prove them in Appendix B.1 for completeness.

Lemma 4.7 (Preprocessing [LP21, run one step of Algorithm 4])**.** *Given a graph G and a terminal set $\mathcal{T}$, there exists an algorithm such that, with probability $1-n^{-11}$ the algorithm outputs a partition of $\mathcal{T}$ such that $\lambda(u,v) = \lambda_G(\mathcal{T})$ if and only if u and v belongs to different parts. This algorithm runs in $O(\log^2 n \cdot \text{MaxFlow}(2n, 2m))$ time.*

Good Decomposition. Let us now consider decomposing the graph G using one or more splits at a time in Chekuri and Xu's algorithm. Suppose we have an ideal oracle that always finds a *balanced $\mathcal{T}'$-split* where both sides contain at least $\frac{1}{4}|\mathcal{T}'|$ terminal vertices in a graph G' with terminal vertex set $\mathcal{T}'$. Then, performing one simple refinement in each subproblem $(G', \mathcal{T}')$ suffices to bound the recursion depth by $O(\log |\mathcal{T}|)$. Unfortunately, such an exemplary oracle does not always exist. In an extreme scenario, consider a graph G and a terminal set $\mathcal{T}$ whose Steiner cactus representation could be a star. Since all $\mathcal{T}$-mincuts are trivial, there is even no split for $\mathcal{T}$. Fortunately, if there is no further split for $\mathcal{T}$ on G, then $\{(G, \mathcal{T})\}$ itself is already a prime decomposition so this is a base case in our divide and conquer algorithm. Motivated by this, we define the *good decomposition* that suffices to bound the recursion depth as follows.

Definition 4.8 (Good Decomposition)**.** Given a graph G and a set of terminal vertices $\mathcal{T}$, a decomposition $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ of G is said to be *good* with respect to $\mathcal{T}$ if $\mathcal{G}$ has the following property. Let $\mathcal{T}_i$ be the set of terminal vertices in G_i. For all i except at most one special index i^*, $|\mathcal{T}_i| \le \frac{3}{4}|\mathcal{T}| + 1$, and there exists a Steiner cactus representation of $\mathcal{T}_{i^*}$ in G_{i^*} that is a star.

Induced Decomposition from a Collection of Disjoint Splits. Consider a collection of $\mathcal{T}$-splits $\mathcal{S} = \{X_1, X_2, \ldots, X_k\}$ where the presented subsets in $\mathcal{S}$ are *disjoint*. The disjointness leads to a robust procedure for performing lots of simple refinements to G using the $\mathcal{T}$-splits from $\mathcal{S}$ in any order. It is straightforward to check that the resulting decomposition is unique up to relabeling the anchor vertices, and we call the result *the decomposition induced by $\mathcal{S}$* on G.

Now, we formally establish sufficient criteria that our maximal isolating mincut algorithm will achieve with high probability.

Definition 4.9 (Good Split Collection)**.** Given a graph G and a set of terminals $\mathcal{T}$, we say that a collection of $\mathcal{T}$-splits $\mathcal{S} = \{X_i\}$ is a *good split collection* if (1) for any $i \neq j$ we have $X_i \cap X_j = \emptyset$, and (2) the decomposition induced by $\mathcal{S}$ is a good decomposition.

With the above Definition 4.9, we are able to summarize and highlight the first step in the divide and conquer framework in Lemma 4.10 (proved in Section 4.3). The entire divide and conquer algorithm is presented in Algorithm 2.

Lemma 4.10. *Given a graph $G = (V, E)$ and a set of terminals $\mathcal{T}$, there exists a randomized Monte Carlo algorithm such that, with probability $1 - n^{-11}$, the algorithm returns a good split collection $\mathcal{S}$ in $O((\log^3 n) \cdot \text{MaxFlow}(3n, 4m))$ time.*

[12] In the hypergraph setting, we replace this procedure with StarOrBrittleCactus, see Section 5.4.

Algorithm 2 A divide and conquer framework that computes a $\mathcal{T}$-Steiner cactus.

Require: a graph $G = (V, E)$, a terminal set $\mathcal{T}$.
Ensure: a $\mathcal{T}$-Steiner cactus of $\mathcal{G}$.

1: **procedure** ComputeSteinerCactus($G, \mathcal{T}$)
2: **if** $|\mathcal{T}| \leq 3$ **then** ▷ Either a triangle or a path.
3: **return** TrivialCactus($G, \mathcal{T}$).
4: **else**
5: Obtain $\mathcal{S}$, a good collection of $\mathcal{T}$-splits on G. ▷ See Lemma 4.10 and Section 4.3.
6: **if** $\mathcal{S} = \emptyset$ **then**
7: **return** StarCactus($G, \mathcal{T}$).[12] ▷ There will be two types of stars, see Section 4.4.
8: **end if**
9: Compute the decomposition $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ induced by $\mathcal{S}$ over G.
10: Obtain $H_i \leftarrow$ ComputeSteinerCactus($G_i, \mathcal{T}_i$) for all i.
11: **return** MergeCactus($G, \mathcal{T}, \{H_i\}$). ▷ See Lemma 4.11 and Section 4.4.
12: **end if**
13: **end procedure**

Merging Cactus from Subproblems. The last piece for accomplishing the divide and conquer algorithm is to merge the cactus returned from each subproblem. On the bright side, with the help of anchor vertices, we do have the proximity of how two cactus should be combined. However, the merging procedure is a bit subtle as we have to make sure that every $\mathcal{T}$-Steiner mincut is preserved in the combined cactus. We summarize the correctness and the runtime guarantee here in Lemma 4.11 and establish the details in Section 4.4.

Lemma 4.11. *Fix a subproblem $(G = (V, E), \mathcal{T})$ in Algorithm 2. Assume all splits generated from the subproblems $\{(G_i, \mathcal{T}_i)\}$ derived from $(G, \mathcal{T})$ are good, and each subproblem returns a correct $\mathcal{T}_i$-Steiner cactus of G_i. Then, the procedures TrivialCactus, StarCactus, and MergeCactus returns a $\mathcal{T}$-Steiner cactus of G in $O(\log |\mathcal{T}| \cdot \text{MaxFlow}(2n, 2m))$ time.*

Now, with Lemmas 4.10 and 4.11, we are able to prove the main Theorem 4.2 by completing the (relatively trivial) implementation details of Line 9 and analyzing its runtime.

Proof of Theorem 4.2. Let $\mathcal{S} = \{X_1, X_2, \ldots, X_\ell\}$ be a good split collection returned from Line 5. To obtain a decomposition $\mathcal{G}$ induced by $\mathcal{S}$ (Line 9), the algorithm first computes the induced subgraphs $G[X_i]$ for all i in linear time. Then, the algorithm simulates the simple refinement of X_i by creating an anchor vertex a_i for each split X_i, and for each edge (u, v) that across the split $u \in X_i$ but $v \notin X_i$, the algorithm either adds a new edge from (u, a_i), or adds the weight to an existing edge (u, a_i). Finally, the algorithm duplicates the graph G and contract each subset X_i into a single anchor vertex a_i, forming the last decomposed graph $(G_{\ell+1}, \mathcal{T}_{\ell+1})$. The implementation of Line 9 takes linear time $O(|V(G)| + |E(G)|)$ in total.

Runtime. Consider the recursion tree of subproblems from Algorithm 2. By definition of a good decomposition, we know that the recursion depth satisfies the following recurrence relation: $\text{MaxDepth}(k) = \text{MaxDepth}(\lfloor \frac{3}{4}k \rfloor + 1) + 1$ whenever $k > 3$ and $\text{MaxDepth}(k) = 0$ whenever $k \leq 3$. By solving the recurrence relation we obtain $\text{MaxDepth}(k) = O(\log k)$.

Now, it suffices to bound the total subproblem sizes within the same recursion depth. We first claim that the number of vertices that occur across all subproblems in the same recursion depth is at most $2n$ using a potential method. For each subproblem $(G, \mathcal{T})$ we define an invariant potential $\Phi(G, \mathcal{T}) := |V(G)| + |\mathcal{T}| - 4$. Notice that in the case where at least one recursion step is performed we must have $|\mathcal{T}| \geq 3$ and hence $\Phi(G, \mathcal{T}) > 0$. If $\{(G_1, \mathcal{T}_1), (G_2, \mathcal{T}_2)\}$ is a simple refinement of G, observe that

$$\Phi(G_1, \mathcal{T}_1) + \Phi(G_2, \mathcal{T}_2) = (|V(G_1)| + |V(G_2)|) + (|\mathcal{T}_1| + |\mathcal{T}_2|) - 8 = (|V(G)| + 2) + (|\mathcal{T}| + 2) - 8 = \Phi(G, \mathcal{T}).$$

Thus, consider the induced decomposition $\{(G_i, \mathcal{T}_i)\}$ on a good split collection of size k, we know that $\Phi(G, \mathcal{T}) = \sum_{i=1}^{k+1} \Phi(G_i, \mathcal{T}_i)$. Therefore, the sum of all potentials within the same recursion depth can be upper bounded by the root problem's potential. Since in every subproblem we have $|\mathcal{T}| \geq 3$, we conclude that the total number of vertices across all subproblems at any particular recursion depth (or any collection of subproblems that are not related to each other) is at most $\Phi(G, |\mathcal{T}|) = n + |\mathcal{T}| - 4 \leq 2n$.

As a consequence, we also deduce that there are at most $4|\mathcal{T}| - 9$ subproblems in the recursion tree, by noticing that the recursion tree is a branching tree with at most $2|\mathcal{T}| - 4$ leaf subproblems (every leaf subproblem contains at least one anchor vertex and every vertex in $V(G) \setminus \mathcal{T}$ occurs in exactly one leaf subproblem.)

To bound the total number of edges across all subproblems within the same recursion depth, we observe that after computing an induced decomposition from a good split collection, the total number of edges is increased by at most $\sum_{i=1}^{\ell} |V(G_i)| \leq 2n$ (notice that we charge the number of the newly generated edges in the last decomposed graph $(G_{\ell+1}, \mathcal{T}_{\ell+1})$ to the edges across each split). Hence, we know that at any recursion depth there are at most $m + 2n \log |\mathcal{T}|$ edges in total.

Finally, we add up the runtime needed per recursion depth. Fix any recursion depth, for each subproblem $(G_j, \mathcal{T}_j)$, by Lemma 4.10 the runtime spent in Line 5 is at most $O((\log^3 n) \cdot \text{MaxFlow}(3|V(G_j)|, 4|E(G_j)|))$, the runtime spent for Line 9 is linear in the graph size $O(|V(G_j)| + |E(G_j)|)$, and by Lemma 4.11 merging cactus takes $O(\log |\mathcal{T}_j| \cdot \text{MaxFlow}(|V(G_j)|, |E(G_j)|))$ time. Hence, by denoting $k = |\mathcal{T}|$, the runtime of Algorithm 2 is

$$\begin{aligned}
&\underbrace{O(\log k)}_{\text{recursion depth}} \cdot O(\underbrace{(m + 2n \log k)}_{\text{Line 9}} + \underbrace{(\log^3 n) \cdot \text{MaxFlow}(3n, 4(m + 2n \log k))}_{\text{Line 3, Line 5, Line 7, and Line 11}}) \\
&= O(m \log k + n \log^2 k + (\log^3 n) \cdot \text{MaxFlow}(3n, 4m + 8n \log k)) \\
&= O((\log^4 n) \cdot \text{MaxFlow}(3n, 4m + 8n \log k)).
\end{aligned}$$

Finally, note that we spent time for preprocessing the input graph by called Lemma 4.7 using $O(\log^2 n \cdot \text{MaxFlow}(2n, 2m))$ time, but this is subsumed by the above bound.

Correctness. By Lemma 4.10, with probability $1 - n^{-11}$ the returned collection is good in Line 5. Throughout execution there are at most $4|\mathcal{T}| - 9 \leq 4n$ invocations to Lemma 4.10. Hence, with a union bound we know that with probability $1 - 4n^{-10}$ the collections of splits from all subproblems are good. Now, by applying the union bound again to Lemma 4.11 we know that the returned cactus is a $\mathcal{T}$-Steiner cactus of G with probability at least $1 - 4n^{-10}$. Therefore, with another union bound we know that with probability $1 - 8n^{-10}$ Algorithm 2 correctly outputs a $\mathcal{T}$-Steiner cactus. □

4.3 Computing a Good Split Collection

In this subsection, we aim to prove Lemma 4.10. Specifically, we propose Algorithm 3, and then we show that with probability at least $1-n^{-11}$, a good split collection can be computed in almost-linear time via $O(\log^2 n)$ maximal isolating mincut algorithms.

Algorithm Description. Algorithm 3 works as follows. The algorithm set up $\lceil \log |\mathcal{T}| \rceil$ different sample rates, namely $2^{-1}, 2^{-2}, \ldots, 2^{-\lceil \log |\mathcal{T}| \rceil}$. For each sample rate 2^{-i}, the algorithm samples each terminal vertex with probability 2^{-i} and forms a set $\mathcal{T}_i$. Then, the algorithm computes the maximal isolating mincuts for the set $\mathcal{T}_i$, and keeps the maximal v-isolating mincut of $\mathcal{T}_i$ if the cut is a $\mathcal{T}$-split and its value equals to $\lambda_G(\mathcal{T})$ (i.e., keeps only the non-trivial $\mathcal{T}$-Steiner mincuts.) To ensure a high probability result, we repeat the whole sampling procedure another $\Theta(\log n)$ times. Let $\mathcal{S}$ be the collection of splits that the algorithm has found so far.

Recall from Definition 4.9 that there are two cases where a split collection is considered to be good: either we find a balanced split whose both sides have at least $\frac{1}{4}|\mathcal{T}|$ terminals, or we find a collection of disjoint sets where the contracted graph (obtained by contracting all these sets) does not contain a split anymore.

Once obtaining the collection of splits $\mathcal{S}$, the algorithm checks if there exists any balanced split by simply checking the size of each set in $\mathcal{S}$. If there is such a balanced split, returning the split itself is sufficient (Line 11). Otherwise, every set in $\mathcal{S}$ now contains either less than $\frac{1}{4}|\mathcal{T}|$ or more than $\frac{3}{4}|\mathcal{T}|$ terminals. The algorithm discards all sets containing more than $\frac{3}{4}|\mathcal{T}|$ terminals and then keeps the maximal subsets among the splits in the collection.

In the case where no balanced split is found, the algorithm does an additional post-processing in Line 13-15. The purpose of this post-processing is to obtain a set of disjoint $\mathcal{T}$-splits that satisfy Definition 4.9. Specifically, in Line 13 the algorithm get rid of all subsets with only one terminal. In Line 14 only maximal subsets are kept and in Line 15 the disjointness of these subsets are enforced. This completes the description of Algorithm 3.

To simplify the correctness proof, we introduce the notion of *irredundant* $\mathcal{T}$-Steiner cactus.

Definition 4.12. A $\mathcal{T}$-Steiner cactus (H, ϕ) of G is said to be *irredundant*, if for every edge e on H, the contraction H/e is no longer a $\mathcal{T}$-Steiner cactus of G.

We remark that the irredundant cactus is somewhat similar to the notion of a normal cactus defined in the work of Nagamochi and Kameda [NK94], except that we still allow tree edges in H.

Intuition of Correctness. The high probability correctness comes from case analysis to *any* $\mathcal{T}$-Steiner cactus of G. Let (H, ϕ) be any irredundant $\mathcal{T}$-Steiner cactus of G. Define a *balanced edge-cut* on H to be a minimum edge cut of H (either one edge or two edges in a cycle) such that the number of terminals on both sides is between $\frac{1}{4}|\mathcal{T}|$ and $\frac{3}{4}|\mathcal{T}|$. Our analysis depends on whether or not a balanced edge-cut exists on H.[13]

Case 1: Balanced Cuts Exist. In the first case where there is a balanced edge-cut, the correctness relies on the sparsest sampling rate $2^{-\lceil \log |\mathcal{T}| \rceil}$. In particular, we rely on a sampled terminal set $\mathcal{T}' = \{u, v, r\}$ of exactly 3 vertices, where two corresponding nodes $\phi(u)$ and $\phi(v)$ are in the "larger

[13] The existence of a balanced edge-cut on H is equivalent to the existence of a balanced split on G. However, we believe the proof is easier to see through if we analyze the algorithm's behavior on H.

Algorithm 3 Computing a Good Split Collection

procedure GoodSplitCollection($G, \mathcal{T}, \lambda := \lambda_G(\mathcal{T})$)
  Initialize $\mathcal{S} \leftarrow \emptyset$.
  repeat the following procedure $\lceil 12 \cdot 1024e \cdot \ln n \rceil$ times **do**
    for $i = 1, 2, \ldots, \lceil \log |\mathcal{T}| \rceil$ **do**
      Sample each terminal vertex with probability 2^{-i}, denote the set by $\mathcal{T}_i$.
      $\mathcal{X} \leftarrow$MaxIsoCut$(G, \mathcal{T}_i)$.
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{X \in \mathcal{X} \mid \mathcal{C}(X) = \lambda\}$.
    end for
  end repeat
  if there exists a balanced split $X \in \mathcal{S}$ where $|\mathcal{T} \cap X|, |\mathcal{T} \setminus X| \geq \frac{1}{4}|\mathcal{T}|$ **then**
    return $\{X\}$. ▷ A single balanced split.
  else
    $\mathcal{S} \leftarrow \{X_i \in \mathcal{S} \mid 2 \leq |X_i \cap \mathcal{T}| \leq \frac{1}{4}|\mathcal{T}|\}$. ▷ Keep only small $\mathcal{T}$-splits.
    $\mathcal{S} \leftarrow \{X_i \in \mathcal{S} \mid X_i \not\subseteq X_j \text{ for all } i \neq j\}$. ▷ Obtain only the maximal subsets.
    For each $X_i \in \mathcal{S}$, set $X_i = X_i \setminus \cup_{j<i} X_j$. ▷ Enforce disjointness to the subsets.
    return $\mathcal{S}$.
  end if
end procedure

side" of the cut and the third corresponding node $\phi(r)$ is in the "smaller side" of the cut. Then, it is possible to prove that with constant probability, the maximal r-isolating cut of $\mathcal{T}'$ contains the right amount of terminal vertices — between $\frac{1}{4}|\mathcal{T}|$ and $\frac{3}{4}|\mathcal{T}|$ (the upper bound comes from Lemma 4.14). Thus, a balanced split will be found in $\mathcal{S}$ with high probability because the sampling procedure with the sparsest sampling rate is repeated $O(\log n)$ times. We formalize the first case here as Lemma 4.13, and give an illustration in Figure 1.

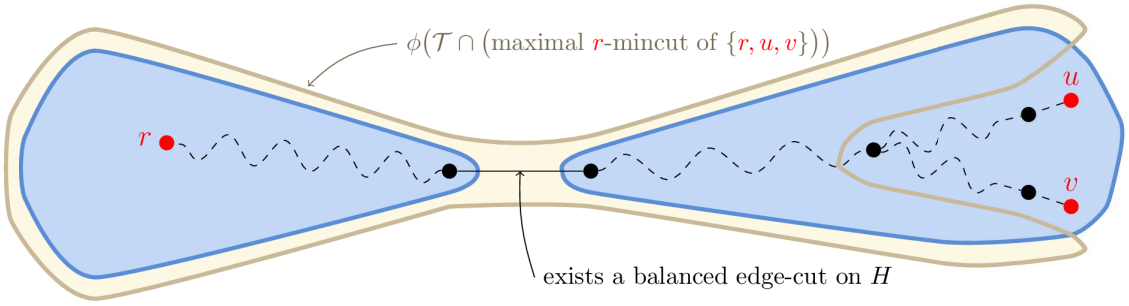

Figure 1: When there is a balanced edge-cut on a $\mathcal{T}$-Steiner cactus H of G, a balanced $\mathcal{T}$-split on G will be found with high probability.

Lemma 4.13. *Let G be the graph with terminal set $\mathcal{T}$ and let (H, ϕ) be a $\mathcal{T}$-Steiner cactus of G. Suppose there is a balanced edge-cut on H. Then, with probability $1 - n^{-11}$ there is a balanced split in $\mathcal{S}$ returned from Algorithm 3.*

To prove Lemma 4.13, we introduce a helper lemma that shows the benefit of having assumption 2. That is, if we have sampled two vertices u and v in the "large side" and sampled a single

vertex r in the "smaller side" of a balanced split, then the maximal r-isolating cut of $\{u, v, r\}$ has to contain *at most* $\frac{1}{2}|\mathcal{T}|$ terminals.

Lemma 4.14. *Let G be the graph with a set $\mathcal{T}$ of terminals that satisfies Assumption 2. Let $r \in \mathcal{T}$ be a terminal such that any r-isolating mincut is a $\mathcal{T}$-mincut. If we sample terminals $u, v \in \mathcal{T} - \{r\}$ uniformly at random, then with probability at least $1/4$, any $\{u, v\}$-mincut of $\{u, v, r\}$ has at least $\frac{1}{2}|\mathcal{T}|$ terminals.*

Proof. Let (H, ϕ) be a $\mathcal{T}$-Steiner cactus of G. Recall that Assumption 2 states that any two terminals $u, v \in \mathcal{T}$ can be separated by some $\mathcal{T}$-mincut. This implies that $\phi(u) \neq \phi(v)$ as all $\mathcal{T}$-mincuts are preserved. From the assumption that r-isolating mincut is a $\mathcal{T}$-mincut, we know that $\phi(r)$ has degree 1 or has degree 2 within a cycle in H. Consider a specialized DFS traversal of H starting from $\phi(r)$. Upon visiting a vertex from a cycle edge, the DFS traversal always tends to choose any edge that leaves the cycle. Let $(r, v_1, v_2, \ldots, v_{|\mathcal{T}|-1})$ be the unique permutation of $\mathcal{T}$ where $(\phi(r), \phi(v_1), \phi(v_2), \ldots, \phi(v_{|\mathcal{T}|-1}))$ is the order (subsequence) of visited vertices by the DFS traversal, i.e. the pre-order. Notice that the DFS traversal only returns to $\phi(r)$ at the very end. Then for any two indices i and j such that $1 \leq i < j \leq |\mathcal{T}|$, maximal r-isolating mincut of $\{v_i, v_j, r\}$ must not contain any vertices in $\{v_i, v_{i+1}, \ldots, v_j\}$ and hence the result follows by counting the fraction of pairs (at least $1/4$) whose position in the permutation differs by at least $\frac{1}{2}|\mathcal{T}|$. □

Lemma 4.14 implies that with constant probability, the mincut of our concern is balanced. Now we are ready to prove Lemma 4.13.

Proof of Lemma 4.13. Suppose there is a balanced edge-cut on H, i.e. a $\mathcal{T}$-Steiner mincut $(X, V \setminus X)$ where $\frac{1}{4}|\mathcal{T}| \leq |X \cap \mathcal{T}| \leq \frac{1}{2}|\mathcal{T}|$. Consider the phase that Algorithm 3 samples each terminal vertex with probability $2^{-\lceil \log |\mathcal{T}| \rceil}$, then with probability at least

$$\begin{aligned}
&\binom{\frac{1}{2}|\mathcal{T}|}{2} \cdot \frac{1}{4}|\mathcal{T}| \left(1 - \frac{1}{2^{\lceil \log |\mathcal{T}| \rceil}}\right)^{|\mathcal{T}|-3} \cdot \left(\frac{1}{2^{\lceil \log |\mathcal{T}| \rceil}}\right)^3 \\
\geq\ & \frac{|\mathcal{T}|^3}{32e} \left(1 - \frac{2}{|\mathcal{T}|}\right) \left(1 - \frac{1}{2(|\mathcal{T}|-1)}\right)^{-3} \left(\frac{1}{2(|\mathcal{T}|-1)}\right)^3 \\
\geq\ & \frac{1}{256e} .
\end{aligned}$$

there will be one sampled terminal r in X, and exactly two sampled terminals u and v in $V \setminus X$. We denote the event described above by $\mathcal{E}$. When $\mathcal{E}$ happens, since $(X, V \setminus X)$ is a $\mathcal{T}$-Steiner mincut, we know that the maximal r-mincut of $\{r, u, v\}$ must contain entire X, hence containing at least $\frac{1}{4}|\mathcal{T}|$ terminals. Now we will use Lemma 4.14 to prove that, conditioned on $\mathcal{E}$, with probability at least $1/4$, the maximal r-mincut of $\{r, u, v\}$ has at most $\frac{3}{4}|\mathcal{T}|$ terminals.

Indeed, conditioned on event $\mathcal{E}$, the maximal r-mincut of $\{r, u, v\}$ is disjoint to the minimal $\{u, v\}$-mincut of $\{r, u, v\}$ (otherwise it contradicts to Disjoint & Posi-modularity Lemma 3.2). Consider the graph G/X with r' being the contracted terminal vertex. Let $\mathcal{T}' := (\mathcal{T} \setminus X) \cup \{r'\}$ be the contracted terminal set. Since X itself is a $\mathcal{T}$-Steiner mincut, $\{r'\}$ is a $\mathcal{T}'$-Steiner mincut on G/X and hence the criteria of Lemma 4.14 are met. Therefore, by Lemma 4.14, with probability at least $1/4$, the minimal $\{u, v\}$-mincut of $\{r, u, v\}$ on G contains at least $\frac{1}{2}|\mathcal{T}'| \geq \frac{1}{4}|\mathcal{T}|$ terminals. This implies that the maximal r-mincut of $\{r, u, v\}$ contains at most $\frac{3}{4}|\mathcal{T}|$ terminals.

As a consequence, we know that with probability $1/(1024e)$, a balanced split will be found in one sampling procedure. With repeating the sampling procedure for $\lceil 12 \cdot 1024e \cdot \ln n \rceil$ times, Algorithm 3 returns a balanced split with probability at least $1 - n^{-12} \geq 1 - n^{-11}$ as desired. □

Case 2: No Balanced Cut. Now let us consider the second case where there is no balanced edge-cut. An illustration for this case is provided in Figure 2. In this case, our algorithm should be able to obtain lots of disjoint $\mathcal{T}$-splits such that, after contracting smaller sides of all these $\mathcal{T}$-splits, the remaining graph (which could still be large) is guaranteed to have a star shaped cactus representation. Not surprisingly, the center of this star can be traced back (by undoing the contractions) to a *centroid* node on H, given the non-presence of a balanced edge-cut.

Recall that H is an irredundant $\mathcal{T}$-Steiner cactus representation (H, ϕ) of G. A *centroid* v is a node on H such that, every edge or cycle incident to v defines a $\mathcal{T}$-Steiner mincut whose corresponding mincut on H not containing v has at most $\frac{1}{4}|\mathcal{T}|$ terminals. We will soon prove (in Lemma 4.16) that no balanced edge-cut on H implies a unique centroid node v on H. This centroid node v naturally partitions $\mathcal{T} \setminus \phi^{-1}(v)$ into sets of terminals $T_1 \sqcup T_2 \sqcup \cdots \sqcup T_k$, where for each i, $\phi(T_i)$ belongs to the same connected component in $H - v$. Moreover, since (H, ϕ) is a cactus representation for G, we know that for each T_i there exists a $\mathcal{T}$-Steiner mincut that separates T_i and $\mathcal{T} \setminus T_i$. Let X'_i be the maximal T_i-mincut of $\mathcal{T}$, and define the collection $\mathcal{S}' = \{X'_i\}$.

Fix a particular i such that $1 \leq i \leq k$, and consider sampling each vertex in $\mathcal{T}$ at the sampling rate $2^{-\lceil \log |T_i| \rceil}$. We can then prove (in Lemma 4.17) that, with constant probability, *exactly one* terminal $w \in T_i$ is sampled, together with *at least one* terminal from any two other subsets (namely $x \in T_j$ and $y \in T_{j'}$, for some $j \neq j' \neq i \neq j$) being sampled.

The following lemma ensures that X'_i can be precisely discovered by our maximal isolating mincut algorithm, illustrated in Figure 2(b).

Lemma 4.15. *Let v be a centroid node on H. Let $w, x, y \in \mathcal{T}$ be three terminals such that $\phi(w), \phi(x)$, and $\phi(y)$ belongs to distinct connected components in $H - v$. Let X be the maximal w-isolating mincut of $\{w, x, y\}$, then the corresponding cut of X in H must not contain v.*

Proof. Let Y be a mincut on H that corresponds to X. Suppose by contradiction that Y contains v. But since $\phi(x), \phi(y) \notin Y$, the cut value of Y would be at least $2\lambda_G(\mathcal{T})$, a contradiction to Y being a mincut of H. □

Notice that by Lemma 4.15, whenever the algorithm seeks the maximal w-isolating mincut of this sampled terminal set, the algorithm obtains exactly the set X'_i. Is the collection $\mathcal{S}'$ serves for our purpose? Not really — $\mathcal{S}'$ may not be a good split collection (Definition 4.9). For example, some mincut $X'_i \in \mathcal{S}'$ may contain exactly one terminal vertex — simply removing these mincuts is an easy fix. What's worse, there could be two mincuts X'_i and X'_j that are not disjoint (e.g., Figure 2(a)). Fortunately, an additional post-processing step can be further applied: whenever there exists $X'_i \cap X'_j \neq \emptyset$ (say $i > j$), we *prune* the larger indexed one by replacing X'_i with $X'_i \setminus X'_j$. By posi-modularity (Lemma 3.2), $T_i \cap T_j = \emptyset$ implies that X'_j is still a T_j-mincut of $\mathcal{T}$. It is straightforward to check that at the end of the post-processing step we have obtained a pruned set $\mathcal{S}'_{\text{pruned}} = \{X_i\}$ where $X_i = X'_i \setminus \cup_{j<i} X'_j$ and every X_i contains at least two terminal vertices.

The post-processing steps mentioned above correspond to Line 15 of Algorithm 3. We prove as a corollary of Lemma 4.17 (Corollary 4.20) that the collection $\mathcal{S}$ returned by Algorithm 3 is exactly the same as $\mathcal{S}'_{\text{pruned}}$ with high probability. At the end of the analysis, we prove in Lemma 4.23 that $\mathcal{S}'_{\text{pruned}}$ is actually a good split collection.

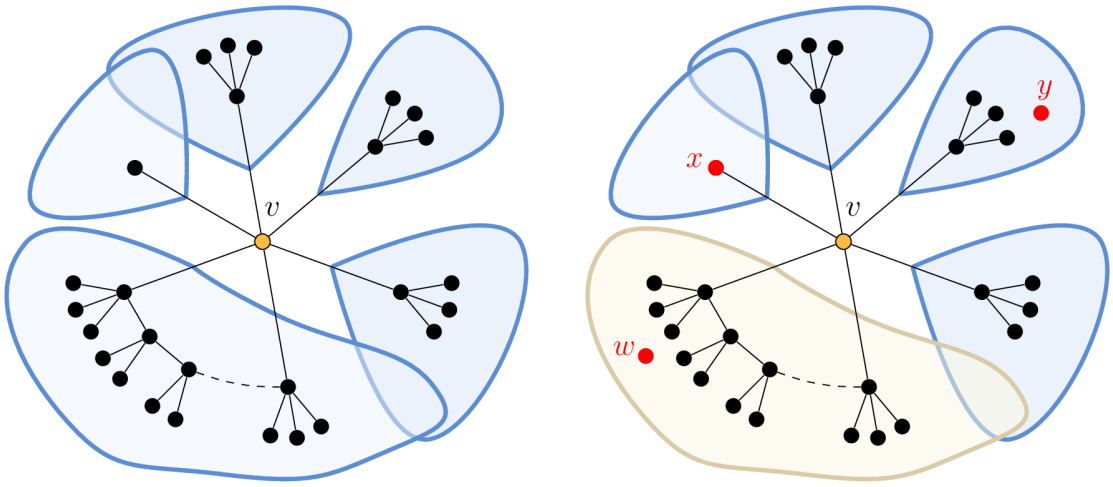

Figure 2: An illustration of a cactus representation H. The colored regions correspond to $\mathcal{S}'$, which is a collection of $\mathcal{T}$-Steiner mincuts of G. (a) When there is no balanced edge-cut, there must be a centroid node v on H. (b) If we sample three terminals w, x, y that are mapped into distinct connected components in $H - v$, then the maximal w-mincut of $\{w, x, y\}$ correspond to exactly the connected component of $H - v$ where w belongs to.

Formalizing the Proof to Case 2. The rest of this subsection devotes to formalize the high-level idea described above. Let G be the graph with terminal set $\mathcal{T}$ and let (H, ϕ) be an irredundant $\mathcal{T}$-Steiner cactus of G. Assume that there is no balanced edge-cut on H. We first show that there exists a unique centroid node on H.

Lemma 4.16. *there exists a unique centroid node v on H whose all incident 1-edges and 2-edges from the same cycle correspond to $\mathcal{T}$-Steiner mincuts of at most $\frac{1}{4}|\mathcal{T}|$ terminals in the side not containing v.*

Proof. The existence of such a centroid node can be proved as follows. We first replace each cactus cycle on H with a star (adding an additional node that represents the cactus cycle), forming a tree H'. We note that each edge on H' still represents a mincut on H. It is well-known that any tree contains a centroid with respect to any vertex weight. Let v be any centroid node on the tree H' when empty nodes have weight zero and non-empty nodes have weight one. If v is a node on H, then by definition v is a centroid node on H. Otherwise, if v is an additional node that represents a cactus cycle, then since each incident edge of v on the tree H' corresponds to a mincut with at most $\frac{1}{4}|\mathcal{T}|$ terminals, one can obtain a balanced edge-cut on H greedily along the cactus cycle represented by v, thereby a contradiction.

To show uniqueness, suppose by contradiction that there are two centroids u and v on H. Consider any mincut on H that separates u and v. By the fact of being a centroid, the side of this mincut not containing u (resp. not containing v) has at most $\frac{1}{4}|\mathcal{T}|$ terminals. Howerver, this implies that the total number of terminals is at most $\frac{1}{2}|\mathcal{T}|$, a contradiction. □

Let $T_1 \sqcup T_2 \sqcup \ldots \sqcup T_k$ be the partition of terminal vertices of $\mathcal{T} \setminus \phi^{-1}(v)$ where two terminal vertices x and y belong to the same T_i if and only if $\phi(x)$ and $\phi(y)$ are in the same connected component of $H - v$. For each i, let X'_i be the maximal T_i-mincut of $\mathcal{T}$ on G, and let $\mathcal{S}' = \{X'_i\}$ be

the collection of all these mincuts. We first establish the relation between the collection $\mathcal{S}'$ and the mincuts computed from the algorithm:

Lemma 4.17. *Let $\hat{\mathcal{S}}$ be the collection of $\mathcal{T}$-Steiner mincut right after the execution of Line 14. With probability at least $1 - n^{-11}$,*

1. *for all $X'_i \in \mathcal{S}'$ such that $|X'_i \cap \mathcal{T}| \geq 2$, we have $X'_i \in \hat{\mathcal{S}}$.*
2. *$\hat{\mathcal{S}} \subseteq \mathcal{S}'$.*

We first establish the following fact and a helper claim:

Fact 4.18. *Let X be any $\mathcal{T}$-Steiner mincut. Consider any corresponding edge-cut that separates $\phi(X \cap \mathcal{T})$ and $\phi(\mathcal{T} \setminus X)$ on H. Suppose that the centroid v belongs to the $\phi(X \cap \mathcal{T})$ side of the edge-cut, then $|X \cap \mathcal{T}| > \frac{1}{4}|\mathcal{T}|$.*

Proof. Since (H, ϕ) is a cactus representation of G, any minimum edge-cut of H must have one side whose nodes are all within the same connected component of $H - v$[14]. The condition that v belongs to the $\phi(X \cap \mathcal{T})$ side implies that all nodes in $\phi(\mathcal{T} \setminus X)$ belong to the same connected component of $H - v$. Now, using the assumption that v is a centroid node, we know that $|\mathcal{T} \setminus X| \leq \frac{1}{4}|\mathcal{T}|$ and hence $|X \cap \mathcal{T}| \geq \frac{3}{4}|\mathcal{T}| > \frac{1}{4}|\mathcal{T}|$. □

Proposition 4.19. *Fix any $X'_i \in \mathcal{S}'$, the maximal T_i-mincut of $\mathcal{T}$. For any $\mathcal{T}$-Steiner mincut $X \not\subseteq X'_i$ but $X \cap T_i \neq \emptyset$, we must have $|X \cap \mathcal{T}| > \frac{1}{4}|\mathcal{T}|$.*

Proof of Proposition 4.19. Let $T_X = X \cap \mathcal{T}$ be the set of terminal vertices in X. Consider the corresponding edge-cut on H that separates $\phi(T_X)$ and $\phi(\mathcal{T} \setminus T_X)$. The condition of the proposition implies that v belongs to the $\phi(T_X)$ side of the edge-cut. Using Fact 4.18 we obtain $|T_X| > \frac{1}{4}|\mathcal{T}|$. □

Intuitively speaking, Proposition 4.19 validates Line 13-15 of Algorithm 3: any $\mathcal{T}$-Steiner mincut that is either crossing or containing some $X'_i \in \mathcal{S}'$ will contain too many terminals and will be removed. As a consequence, Line 13 protects X'_i from being accidentally removed in Line 14 and being "chopped" in Line 15. Now we formally prove Lemma 4.17.

Proof of Lemma 4.17.

Part 1. Fix an $X'_i \in \mathcal{S}'$. We first notice that by Proposition 4.19, any $\mathcal{T}$-Steiner mincut X that is a superset of X'_i has more than $\frac{1}{4}|\mathcal{T}|$ terminal vertices. Hence, such mincut X will be excluded by executing Line 13 of the algorithm. Now it suffices to show that X'_i will be found and appeared in $\hat{\mathcal{S}}$ with high probability.

Consider sampling terminal vertices with probability $p := 2^{-\lceil \log |T_i| \rceil}$. By Lemma 4.15, it suffices to lower bound the probability of the event where (1) no other terminals in T_i is sampled and (2) some terminals are sampled from at least two other sets. The following analysis further restricts condition (2): we first form a partition $\mathcal{T} \setminus \phi^{-1}(v) = T_i \sqcup Q_1 \sqcup Q_2$ and compute the probability that at least one terminal from each of Q_1 and Q_2 are sampled.

Indeed, since each part has a size at most $\frac{1}{4}|\mathcal{T}|$, by a straightforward greedy algorithm, it is possible to group all the parts except T_i into two large sets, whose sizes are between $\frac{1}{4}|\mathcal{T}|$ and

[14] This sentence even holds when v is not a centroid.

$\frac{5}{8}|\mathcal{T}|$[15]. Let Q_1 and Q_2 be such two sets. We have $\mathcal{T} \setminus \phi^{-1}(v) = T_i \sqcup Q_1 \sqcup Q_2$. Now it suffices to prove that with constant probability, exactly one terminal is sampled from T_i and at least one terminal is sampled from each of Q_1 and Q_2. By a standard probability argument, we know that sampling exactly one terminal from T_i has a probability of at least $1/(2e)$. For each $j \in \{1, 2\}$, sampling at least one terminal from Q_j has a probability at least:

$$1 - (1-p)^{|Q_j|} \geq 1 - \left(1 - \frac{1}{2|T_i|}\right)^{|Q_j|} \geq 1 - e^{-1/2}. \qquad (|Q_j| \geq \tfrac{1}{4}|\mathcal{T}| \geq |T_i|)$$

Hence, the success probability per sampling at the particular scale is at least

$$\frac{1}{2e}\left(1 - e^{-1/2}\right)^2 \geq 0.02$$

By repeating the sampling procedure at least $\lceil 12 \cdot (1/0.02) \cdot \ln n \rceil$ times, we know that Algorithm 3 obtains a $\mathcal{T}$-Steiner mincut that separates T_i and $\mathcal{T} \setminus T_i$ with probability $1 - n^{-12}$. By applying another union bound over all i we obtain the success probability $1 - |\mathcal{T}|n^{-12} \geq 1 - n^{-11}$ as desired.

Part 2. To show that $\hat{\mathcal{S}} \subseteq \mathcal{S}'$, assume by contradiction that there is $X \in \hat{\mathcal{S}}$ but $X \notin \mathcal{S}'$. By Line 7 of Algorithm 3, we know that X is a $\mathcal{T}$-Steiner mincut. By Line 13, we know that $2 \leq |X \cap \mathcal{T}| \leq \frac{1}{4}|\mathcal{T}|$. Using Fact 4.18, we know that there must exist a part T_i such that $X \cap \mathcal{T} \subseteq T_i$. This implies that $X \subsetneq X'_i$. However, from Part 1 we knew that with probability $1 - n^{-12}$, $X'_i \in \hat{\mathcal{S}}$. According to Line 14, X will be removed from $\mathcal{S}$ so $X \notin \hat{\mathcal{S}}$, a contradiction. □

Recall that $\mathcal{S}'_{\text{pruned}}$ is defined by, first removing all $X'_i \in \mathcal{S}'$ where $|X'_i \cap \mathcal{T}| = 1$, and then replace each X'_i with $X_i = X'_i \setminus \cup_{j<i} X'_j$. Since these steps are identical to Line 13 and Line 15 of Algorithm 3, we obtain the following corollary.

Corollary 4.20. *Suppose that Lemma 4.17 holds. Then, after the execution of Line 15, $\mathcal{S} = \mathcal{S}'_{\text{pruned}}$.*

Proof. The post-processing that defines S'_{pruned} is exactly the same as Line 15. □

Once we prove that with high probability, the returned collection $\mathcal{S}$ from Algorithm 3 is the same as $\mathcal{S}'_{\text{pruned}}$, we can put all our effort now proving that $\mathcal{S}'_{\text{pruned}}$ is exactly what we want for the divide-and-conquer algorithm. That is, we aim to show that $\mathcal{S}'_{\text{pruned}}$ is a good split collection.

Proposition 4.21. *Each $X_i \in \mathcal{S}'_{\text{pruned}}$ is a $\mathcal{T}$-split.*

Proof. First, X_i is a $\mathcal{T}$-Steiner mincut by the definition $X_i = X'_i \setminus \cup_{j<i} X'_j$ and by posi-modularity (Lemma 3.2). Moreover, since each X_i contains at least two terminal vertices so X_i is a $\mathcal{T}$-split. □

Lemma 4.22. *Consider a particular pruned mincut $X_i \in \mathcal{S}'_{\text{pruned}}$. Then, the mincut on H that separates $\phi(X_i \cap \mathcal{T})$ with $\phi(\mathcal{T} \setminus X_i)$ is incident to v. That is, after the contraction of all X_i, the contracted graph has a cactus that is a star shape with ≥ 4 leaves.*

[15]The greedy strategy iteratively merges each subset to the smaller pile of two. In the end, the difference between the two piles' sizes is at most $\frac{1}{4}|\mathcal{T}|$. Since the sum of the two sizes is at least $\frac{3}{4}|\mathcal{T}|$, the smaller pile must have at least $\frac{1}{4}|\mathcal{T}|$ elements.

Proof. This is straightforward to check using the fact that H is irredundant. To bound the number of leaves, we first deduce that $|\mathcal{T}| \geq 5$ — this is because at least one split is found and there is no balanced split. Moreover, using Assumption 2, there can be at most one terminal being the center of the star. Since each split contains strictly less than $\frac{1}{4}|\mathcal{T}|$ terminals, we conclude that the star shape has at least 4 leaves. □

Lemma 4.23. *$\mathcal{S}'_{\text{pruned}}$ is a good split collection.*

Proof. It suffices to check that $\mathcal{S}'_{\text{pruned}} = \{X_i\}$ satisfies Definition 4.9. By Proposition 4.21, we know that all X_i are indeed $\mathcal{T}$-splits. Furthermore, by the construction of X_i we know that for any $i \neq j$, $X_i \cap X_j = \emptyset$. Thus, condition (1) of Definition 4.9 is satisfied.

Now, since all $\mathcal{T}$-Steiner mincuts in $\mathcal{S}'_{\text{pruned}}$ are disjoint, the decomposition $\{(G_i, \mathcal{T}_i)\}$ induced by $\mathcal{S}$ is well-defined. Without loss of generality we may apply simple refinements to G in the order of $X_1, X_2, \ldots, X_k$. Let $(G_1, \mathcal{T}_1), \ldots, (G_{k+1}, \mathcal{T}_{k+1})$ be the decomposed graphs[16]. By Lemma 4.22, we know that there exists a cactus representation that is a star shape for the very last graph $(G_{k+1}, \mathcal{T}_{k+1})$. Therefore, by Definition 4.8, $\{(G_i, \mathcal{T}_i)\}$ is a good decomposition. This implies that condition (2) of Definition 4.9 holds and $\mathcal{S}$ is a good split collection. □

Proof of Lemma 4.10. The correctness of Lemma 4.10 follows from Lemma 4.13 and Lemma 4.23. To analyze the runtime, we first note that Algorithm 3 involves $O(\log^2 n)$ maximal isolating mincut computations, which contributes a total of $O((\log^3 n) \cdot \text{MaxFlow}(3n, 4m))$ time by Theorem 3.1. Moreover, by Lemma 3.4 we know that the total size of subsets after Line 9 is at most $O(n \log^2 n)$.

It is not hard to see that each of the remaining steps can be implemented in time linear to the total size of the found mincuts, which is $O(n \log^2 n)$: in Line 10 it suffices to scan through every subset in $\mathcal{S}$ and count the number of terminals; similar analysis holds for Line 13. To implement Line 14, it suffices to scan through all the subsets $X_i \in \mathcal{S}$ and mark the size of $|X_i|$ at each terminal vertex $u \in \mathcal{T} \cap X_i$. For each terminal vertex u, we simply select the largest sized X_i that contains u and get rid of all the others. This step also take $O(n \log^2 n)$ time. In Line 15, the algorithm initializes a boolean array A and iterates through each set $X_1, X_2, \ldots, X_k$. For each set X_i, the algorithm checks for each vertex $u \in X_i$ whether or not u has been set in A. If set, the algorithm discard u from X_i. Otherwise, the algorithm marks $A[u] \leftarrow \texttt{true}$ and keeps u in the set. Thus, it takes only a linear scan for Line 15. In conclusion, the total runtime of Algorithm 3 is $O((\log^3 n) \cdot \text{MaxFlow}(3n, 4m))$. □

4.4 Returning a Correct Cactus

In this section, we prove Lemma 4.11. In particular, we will implement the procedures TrivialCactus$(G, \mathcal{T})$, StarCactus$(G, \mathcal{T})$, and MergeCactus$(G, \mathcal{T}, \{H_i\})$. Then, we will prove the correctness and analyze the runtime. In this section, we assume that all the other divide and conquer steps (especially a good split collection is returned from Line 5 of Algorithm 2) are correct.

Hollow 3-Stars vs 3-Cycle. A *hollow 3-star* is an induced subgraph on H that has 4 nodes where an empty node connects to exactly 3 other nodes. Nagamochi and Kameda [NK94] pointed out that if we replace this hollow 3-star induced subgraph with a 3-cycle (thereby removing the center empty node) from H, the resulting graph preserves the same set of mincuts on H.

[16]Note that for $1 \leq i \leq k$, we must have $\mathcal{T}_i \supsetneq T_i$: $\mathcal{T}_i$ contains exactly one more anchor vertex.

To ensure a correct cactus being returned, we implement the procedures of Algorithm 2 with the following invariant:

Invariant 4.24. *Let (H, ϕ) be the $\mathcal{T}$-Steiner cactus returned by any subproblem $(G, \mathcal{T})$. Then H is irredundant, and does not contain a hollow 3-star as an induced subgraph.*

Now, we start proving Lemma 4.11.

(Line 3) Trivial Cactus. Since $|\mathcal{T}| \leq 3$, by computing the mincut for every partition of $\mathcal{T}$ we obtain a $\mathcal{T}$-Steiner cactus in $O(\text{MaxFlow}(n, m))$ time. Notice that a returned cactus can be chosen to satisify Invariant 4.24 as there are only $O(1)$ ways to construct such a cactus.

(Line 7) Star Cactus. In this case, there is no $\mathcal{T}$-split that is a $\mathcal{T}$-Steiner mincut. Since $|\mathcal{T}| \geq 4$ when Line 7 is reached in Algorithm 2, we claim that there exists a star shaped $\mathcal{T}$-Steiner cactus of G.

Fact 4.25. *Let G be a graph and $\mathcal{T}$ be a set of terminals with $|\mathcal{T}| \geq 4$. Suppose that every $\mathcal{T}$-Steiner mincut is trivial (i.e., a t-isolating mincut of $\mathcal{T}$ for some $t \in \mathcal{T}$). Then, there exists (H, ϕ), a $\mathcal{T}$-Steiner cactus of G such that H is a star graph.*

Proof. Let $A \subseteq \mathcal{T}$ be the subset of terminals t whose t-isolating mincut is a $\mathcal{T}$-Steiner mincut. Construct H as a star graph with A being the set of leaves. The rest vertices in $\mathcal{T} \setminus A$ (possibly empty) are mapped to the center of H. Then H preserves all $\mathcal{T}$-Steiner mincuts. □

With the assumption 2, we know that there is at most one terminal vertex t whose t-isolating mincut of $\mathcal{T}$ is not a $\mathcal{T}$-Steiner mincut. Hence, after invoking one Isolating Cut Lemma (Theorem 2.3) and checking the isolating mincut values, the StarCactus procedure is able to return a $\mathcal{T}$-Steiner cactus of G in $O(\log |\mathcal{T}| \cdot \text{MaxFlow}(2n, 2m))$ time. By assumption 2 and $|\mathcal{T}| \geq 4$, no hollow 3-star can be formed and thus Invariant 4.24 holds.

(Line 11, Case 1.) Merging from a Balanced Split. There are two cases when Line 11 is reached, depending on whether a balanced split is found or not. Suppose that a balanced split is found so the good split collection contains exactly one $\mathcal{T}$-split $|\mathcal{S}| = 1$.

The MergeCactus procedure relies on the following useful observation.

Fact 4.26. *Let $(G, \mathcal{T})$ be a subproblem and let $t \in \mathcal{T}$ be an* anchor vertex *generated in some ancestor problems. Let (H, ϕ) be any irredundant $\mathcal{T}$-Steiner cactus of G. Then, the node $\phi(t)$ on H has either degree 1 (a leaf), or degree 2 but in a cycle.*

Proof. Since t is an anchor vertex, $\{t\}$ is a $\mathcal{T}$-Steiner mincut of G. Hence, there exists a mincut on H that separates $\phi(t)$ with $\phi(\mathcal{T} \setminus \{t\})$. Since H is irredundant, the $\phi(t)$-side of the mincut must be a single node (i.e., $\phi(t)$ itself), and the statement follows. □

Now, let us assume that the balanced split in a good split collection $\mathcal{S}$ decomposes the graph G into two subproblems $(G_1, \mathcal{T}_1)$ and $(G_2, \mathcal{T}_2)$, with a shared anchor vertex $a \in \mathcal{T}_1 \cap \mathcal{T}_2$. Let (H_1, ϕ_1) and (H_2, ϕ_2) be the cactus returned from the two subproblems respectively.

Using Fact 4.26, there are only constantly many situations to be handled:

Leaf-Leaf Case. If both anchor nodes $\phi_1(a)$ and $\phi_2(a)$ have degree 1, say edge $(x, \phi_1(a))$ in H_1 and edge $(\phi_2(a), y)$ in H_2, then the procedure forms the merged cactus by simply connecting H_1 and H_2 with an edge (x, y) and then delete the anchor nodes $\phi_1(a)$ and $\phi_2(a)$. Notice that Invariant 4.24 holds since (x, y) cannot be further contracted, and this operation does not produce a hollow 3-star, simply because $|\mathcal{T}| \geq 4$.

Leaf-Cycle Case. If the anchor vertex a has degree 1 in one of the cactus and degree 2 in the other, without loss of generality, edge $(\phi_1(a), x)$ in H_1, edge $(\phi_2(a), y)$ and $(\phi_2(a), z)$ in H_2. Then the procedure simply connects H_1 and H_2 with edges (x, y) and (x, z) and then deletes the anchor nodes $\phi_1(a)$ and $\phi_2(a)$. Invariant 4.24 holds here too.

Cycle-Cycle Case. The last case is nontrivial. Now both anchor nodes $\phi_1(a)$ and $\phi_2(a)$ have degree 2, say edges $(\phi_1(a), x_1)$ and $(\phi_1(a), y_1)$ in H_1 and edges $(\phi_2(a), x_2)$ and $(\phi_2(a), y_2)$ in H_j. We need to take care of the relation between these two cycles, as in this case there may be missing $\mathcal{T}$-Steiner mincuts. There will be three possible outcomes in the merged cactus, and the algorithm has to identify the correct formulation.

1. The two cycles are separated from each other in the cactus, and there is an empty node in the middle.
2. They form one larger cycle together, concatenated by edges (x_1, x_2) and (y_1, y_2).
3. They form one larger cycle together, concatenated by edges (x_1, y_2) and (y_1, x_2).

Test via Max-Flows. Fortunately, it is possible to distinguish the three cases mentioned above. All we need to do is to obtain the mincut values between terminal sets $\{x_1, x_2\}$ and $\{y_1, y_2\}$, and between terminal sets $\{x_1, y_2\}$ and $\{x_2, y_1\}$ respectively. This can be done by running st-mincut oracles on G with the terminal sets $\{x_1, x_2\}$ (or $\{x_1, y_2\}$) contracted to s and $\{y_1, y_2\}$ (or $\{x_2, y_1\}$ respectively) contracted to t. If none of them equals to the known value $\lambda_G(\mathcal{T})$, then it is the case (1). Otherwise, $\{x_1, x_2\}$ or $\{x_1, y_2\}$ is a $\mathcal{T}$-Steiner mincut corresponding to case (2) and (3) respectively. This can be done in $O(\text{MaxFlow}(n, m))$ time.

Correctness. Here we briefly prove that the combined cactus is indeed a $\mathcal{T}$-Steiner cactus of G. Let H' be some irredundant $\mathcal{T}$-Steiner cactus of G. Let $\mathcal{S} = \{X\}$ and let $a \in \mathcal{T}_1 \cap \mathcal{T}_2$ be the anchor vertex. Since H' is irredundant, there is a unique mincut on H' that separates $\phi(X \cap \mathcal{T})$ and $\phi(\mathcal{T} \setminus X)$. It is straightforward to check that if X corresponds to a 1-edge-cut of H', then every mincut in G is preserved in the subproblems already. If X corresponds to a 2-edge-cut on H', then at least one anchor node $\phi_1(a)$ or $\phi_2(a)$ must be in a cycle in the returned cactus. In this case, the above procedure recovers the cycle correctly. Invariant 4.24 holds for the cycle-cycle case too as no new star can be formed.

(Line 11, Case 2.) Merging When There Is No Balanced Split. The other case when invoking Line 11 is that no balanced split exists and thus a good split collection with one or more splits is returned. Suppose that $|\mathcal{S}| = \ell \geq 1$ and the graph is decomposed into $\ell + 1$ subproblems $\{(G_i, \mathcal{T}_i)\}$, with the last subgraph $(G_{\ell+1}, \mathcal{T}_{\ell+1})$ containing all the anchor vertices generated in this recursion step. By Lemma 4.22 and the algorithm, the returned cactus from the subproblem $(G_{\ell+1}, \mathcal{T}_{\ell+1})$ must be a star graph of at least 4 leaves. Let H_{center} be this star graph.

The algorithm attaches each cactus H_i from other subproblems $(G_i, \mathcal{T}_i)$ to the star graph H_{center} via the **Leaf-Leaf Case** or the **Leaf-Cycle Case**. Since no tests are required, this step can be done in linear time $O(|V(G)|+|E(G)|)$. The correctness argument is the same as the balanced-split case since we can view this process as sequentially merging two cactus at a time. For the same reason, Invariant 4.24 holds too.

Conclusion in Runtime. We conclude that the bottleneck of the runtime happens whenever the algorithm invokes the StarCactus procedure, which invokes one isolating cut algorithm and runs in $O(\log|\mathcal{T}| \cdot \text{MaxFlow}(2n, 2m))$ time. All other operations require at most one max flow procedure so MergeCactus takes up to $O(\text{MaxFlow}(n, m))$ time.

5 Steiner Hypercactus Construction

In this section, we generalize our algorithms in Sections 3 and 4 to hypergraphs; we give an almost-linear time construction of a *Steiner hypercactus representation* that succinctly represents all Steiner hyperedge mincuts. This is the first almost-linear time algorithm even for normal hypercactus representation.

Preliminaries on (Steiner) Hypercactus Representation. A hypergraph $G = (V, E)$ consists of a vertex set V and a hyperedge set E where each edge e is a subset of vertices. Let $n = |V|, m = |E|$ and $p = \sum_{e\in E}|e|$. Given a subset of vertices $X \subseteq V$, the *induced subhypergraph* $G[X]$ is defined by removing all outside vertices $V \setminus X$ and all the incident edges, i.e. $G[X] = (X, E_X)$ where $E_X = \{e \in E \mid \forall v \in e, v \in X\}$. Let $\mathcal{T} \subseteq V$ be a terminal vertex set. A $\mathcal{T}$-Steiner cut is a cut of G that separates $\mathcal{T}$. A $\mathcal{T}$-Steiner mincut is a minimum valued $\mathcal{T}$-Steiner cut.

A *hypercactus* is a hypergraph that satisfies the following properties:

- H is connected.
- Every rank-2 edge on H is in at most one cycle, and this cycle must contain only rank-2 edges.
- For every hyperedge of rank $r > 2$, removing this hyperedge partitions H into *exactly* r connected components, where each connected component is also a hypercactus.

Fleiner and Jordán [FJ99] showed that there exists a hypercactus graph H that represents all $\mathcal{T}$-Steiner hyperedge mincuts.

Definition 5.1 (Steiner Hypercactus, see also [FJ99])**.** Given a hypergraph G and a terminal set $\mathcal{T}$, a *$\mathcal{T}$-Steiner hypercactus* (H, ϕ) is a weighted hypercactus graph H with a mapping $\phi : \mathcal{T} \to V(H)$ such that (1) edges in a cycle have weights $\lambda_G(\mathcal{T})/2$ and edges or hyperedges have weights $\lambda_G(\mathcal{T})$, and (2) an A-mincut of $\mathcal{T}$ is a $\mathcal{T}$-Steiner mincut if and only if a global mincut on H separates $\phi(A)$ and $\phi(\mathcal{T} \setminus A)$.

Our Result. Our main result is an algorithm for constructing Steiner hypercactus for any hypergraphs using polylogarithmic maxflow calls.

Theorem 5.2. *Let G be a hypergraph with n vertices, m edges and p total volume of edges. Let $\mathcal{T}$ be a set of terminals. There exists a randomized Monte Carlo algorithm such that, with probability*

$1 - 8n^{-10}$, *the algorithm correctly computes a* $\mathcal{T}$*-Steiner hypercactus in* $O(\log^4 n) \cdot \text{MaxFlow}(O(n+m), O(p + n \log |\mathcal{T}|))$ *time.*

The rest of the section is organized as follows. In Section 5.1, we show that by carefully modifying a definition of A-cuts in a subtle way, we can generalize our maximal isolating mincuts algorithm from Section 3 to hypergraphs. Then, we present our divide-and-conquer algorithm for constructing Steiner cactus in Section 5.2, generalizing our algorithm in Section 4.

5.1 Maximal Isolating Mincuts on Hypergraphs

To overcome the first challenge to fast algorithms for computing maximal isolating mincuts in hypergraphs as discussed in Section 1.2, we carefully give a new definition of A-cuts of $\mathcal{T}$ in Definition 5.3 with additional *connectivity constraint*. Then, we give a generalization of Theorem 3.1 in Theorem 5.4.

Definition 5.3. Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$. For any proper subset of terminals $A \subsetneq \mathcal{T}$, a cut X is *A-cut* of T if it satisfies the following conditions:

- The cut $(X, V \setminus X)$ separates A and $\mathcal{T} \setminus A$, with $A \subseteq X$.
- After removing the boundary edges ∂X, for any $u \in X$, there exists a path from u to some vertex $v \in A$. That is, $(G/A)[X]$ is connected.

We use the same terminology for related concepts. An *A-mincut* of $\mathcal{T}$ is a minimum valued A-cut of $\mathcal{T}$, denoted as X_A. For any vertex $t \in \mathcal{T}$, a cut X_t is *t-isolating mincut* of $\mathcal{T}$ if X_t is a $\{t\}$-mincut of $\mathcal{T}$. We say that an A-mincut X_A of $\mathcal{T}$ is *maximal* (resp. minimal), if for any other A-mincut X'_A of $\mathcal{T}$, we have $X_A \supseteq X'_A$ (resp. $X_A \subseteq X'_A$).

By crucially exploiting Definition 5.3, we are able to generalize the maximal isolating mincuts algorithm from Theorem 3.1 to hypergraphs.

Theorem 5.4. *There exists an algorithm that, given an undirected weighted hypergraph* $G = (V, E)$ *and a terminal set* $\mathcal{T} \subseteq V$*, in* $O(\log |\mathcal{T}|) \cdot \text{MaxFlow}(O(n+m), O(p))$ *time computes the maximal* v*-isolating mincuts of* $\mathcal{T}$ *for all terminals* $v \in \mathcal{T}$*.*

The new definition of A-cuts leads to some inconveniences. For example, an A-mincut of $\mathcal{T}$ may not be a $(\mathcal{T} \setminus A)$-mincut of $\mathcal{T}$ in a hypergraph. Also, suppose X_A and X_B are A-mincut and B-mincut, respectively, where $A \subseteq B$. In normal graphs, $X_A \cap X_B$ would be an A-mincut by Lemma 3.5, but in hypergraphs $X_A \cap X_B$ might not be A-mincut with respect to our definition because $(G/A)[X_A \cap X_B]$ might not be connected. Anyhow, this inconvenience is easy to deal with.

The technical reason why we need to work with this new definition is that it allows us to show the hypergraph version of the Pairwise Intersection Only Lemma 3.3, which is the key to efficiency. Given this lemma, we verify that all ideas in Section 3 indeed generalize to hypergraphs and prove Theorem 5.4 in Appendix C. We need to reprove everything since we are working with the new basic definition.

5.2 Our Divide and Conquer Framework

In this section, we finally give an almost-linear time construction for Steiner hypercactus, based on the maximal isolating mincut algorithms for hypergraphs from Section 5.1. Let us describe our algorithm.

Preprocessing. First, we preprocess the hypergraphs in the same way that we did for normal graphs. After preprocessing in $O(\log^2 n) \cdot \text{MaxFlow}(O(n+m), O(p))$ time, we may assume that for any two terminal vertices there is a $\mathcal{T}$-Steiner mincut that separates them (see Lemma B.1.)

Modified Algorithm 2. Our main divide-and-conquer algorithm for Steiner hypercactus is based on the same framework as Section 4.2. Recall that a hypergraph consisting of a single (weighted) hyperedge containing all vertices is called a *brittle*. We modify Algorithm 2 to make it compute a $\mathcal{T}$-Steiner hypercactus as follows.

1. In Line 5, we called Algorithm 3 to return a good split collection. In Line 6 of Algorithm 3, we now use the maximal isolating mincut algorithm on hypergraphs (Theorem 5.4) instead.
2. The hypercactus returned from the StarCactus procedure — the procedure now may return a brittle instead of a star so we now call it StarOrBrittleCactus, which will be discussed in Section 5.4.
3. The implementation of MergeCactus is changed and will be specified in Section 5.4.

Key Property: Never Split Higher-rank Hyperedges. As discussed in Section 1.2, the second challenge to efficiently compute a Steiner hypercactus representation is because a hypercactus contains hyperedges of rank more than two. Splitting these higher-rank hyperedges leads to slow run time. Our key observation is that the Modified Algorithm 2 never splits these hyperedges in a non-trivial way (proved at the end of the section).

Lemma 5.5. *Consider* any *$\mathcal{T}$-Steiner hypercactus (H, ϕ). Let e be an hyperedge on H of rank ≥ 3. Then, the Modified Algorithm 2 above never finds a split such that e gets decomposed to at least two smaller hyperedges with rank ≥ 3 in at least two subproblems.*

Therefore, the algorithm never splits higher-rank hyperedges in a hypercactus. This motivates us to define a *good decomposition* for hypergraphs in almost the same way as defined for normal graphs (Definition 4.8), except that we treat brittles as a base case similar to stars.

Definition 5.6 (Good Decomposition for Hypergraphs)**.** Given a hypergraph G and a set of terminal vertices $\mathcal{T}$, a decomposition $\mathcal{G} = \{(G_i, \mathcal{T}_i)\}$ of G is said to be *good* with respect to $\mathcal{T}$ if $\mathcal{G}$ has the following property. Let $\mathcal{T}_i$ be the set of terminal vertices in G_i. For all i except at most one special index i^*, $|\mathcal{T}_i| \leq \frac{3}{4}|\mathcal{T}| + 1$, and there exists a Steiner cactus representation of $\mathcal{T}_{i^*}$ in G_{i^*} that is a star or a brittle.

Now, the definition of a *good split collection* on a hypergraph is the same as Definition 4.9 on normal graph. Similar to Section 4, there are two main steps in the analysis of the algorithms.

First, Lemma 5.7 summarizes the result that computes a good split collection on hypergraphs.

Lemma 5.7. *Given a hypergraph $G = (V, E)$ and a set of terminals $\mathcal{T}$, there exists a randomized Monte Carlo algorithm such that, with probability $1 - n^{-11}$, the algorithm returns a good split collection $\mathcal{S}$ in $O(\log^3 n) \cdot \text{MaxFlow}(O(n+m), O(p))$ time.*

Second, Lemma 5.8 summarizes the result for returning a correct hypercactus.

Lemma 5.8. *Fix a subproblem $(G = (V, E), \mathcal{T})$ in Algorithm 2. Assume all splits generated from the subproblems $\{(G_i, \mathcal{T}_i)\}$ derived from $(G, \mathcal{T})$ are good, and each subproblem returns a correct $\mathcal{T}_i$-Steiner hypercactus of G_i. Then, with probability $1 - n^{-11}$, the procedures* TrivialCactus, StarOrBrittleCactus, *and* MergeCactus *returns a $\mathcal{T}$-Steiner hypercactus of G in $O(\log |\mathcal{T}|) \cdot \text{MaxFlow}(O(n+m), O(p))$ time. The randomization comes only from the almost-linear time max-flow algorithm.*

With Lemma 5.7 and Lemma 5.8, we can prove our main result Theorem 5.2. Since the proof goes in very similar way to the analogous theorem for normal graphs (Theorem 4.2), we defer the proof to Appendix D. Now, we proceed to prove Lemma 5.7 and Lemma 5.8 in respective subsections.

5.3 Computing a Good Split Collection

Now we aim to prove Lemma 5.7 using the same approach as in Section 4.3. Specifically, we show that Algorithm 3 also works for hypergraph.

Recall that H denotes a hypercactus. Motivated by Lemma 5.5, we observe that our algorithm never decomposes a rank ≥ 4 hyperedge on H into two smaller hyperedges of rank ≥ 3. This leads us to consider *singular hyperedge cut* only. Define a *singular hyperedge cut* $(Q, V(H) \setminus Q)$ of H if there exists hyperedge $e \in H$ such that $|Q \cap e| = 1$.

We call a mincut of H *accessible* if it is a singular hyperedge cut, or all edges across the boundary are normal edges (either one edge or two edges in a cycle). Note that with Lemma 5.5, all $\mathcal{T}$-Steiner mincuts returned from Algorithm 2 correspond to accessible mincuts on H. Define a *balanced edge-cut* on hypercactus H to be an accessible mincut of H such that the number of terminals on both sides is between $\frac{1}{4}|\mathcal{T}|$ and $\frac{3}{4}|\mathcal{T}|$. Again, to show correctness of Algorithm 3, our analysis depends on whether or not a balanced edge-cut exists on H.

Case 1: Balanced Cuts Exist. In the first case where there is a balanced edge-cut, The proof is exactly the same as normal graph, except that we need to plug in a hypergraph version of Lemma 4.14.

Lemma 5.9. *Let G be the hypergraph with a set $\mathcal{T}$ of terminals that satisfies Assumption 2. Let $r \in \mathcal{T}$ be a terminal such that any r-isolating mincut is a $\mathcal{T}$-mincut. If we sample terminals $u, v \in \mathcal{T} - \{r\}$ uniformly at random, then with probability at least $1/4$, maximal $\{r\}$-isolating mincut of $\mathcal{T}' = \{u, v, r\}$ has at most $\frac{1}{2}|\mathcal{T}|$ terminals.*

Proof. Let (H, ϕ) be a $\mathcal{T}$-Steiner hypercactus of G. By Assumption 2 every vertex $v \in \mathcal{T}$ will be mapped to different vertices in H. From the assumption that r-isolating mincut is a $\mathcal{T}$-mincut, we know that $\phi(r)$ is a *leaf* node on H, i.e. either has degree 1 (may connected to a normal edge or a hyperedge) or has degree 2 within a cycle in H. Consider a specialized DFS traversal of H starting from $\phi(r)$: where upon visiting a vertex from an cycle edge, the DFS traversal always tends to choose any edge that leaves the cycle; upon first visiting a vertex from a specific hyperedge, the DFS traversal will visit the other veritces in the hyperedge with arbitrary order. Let $(r, v_1, v_2, \ldots, v_{|\mathcal{T}|-1})$ be the a permutation of $\mathcal{T}$ where $(\phi(r), \phi(v_1), \phi(v_2), \ldots, \phi(v_{|\mathcal{T}|-1}))$ is the order (subsequence) of visited vertices by the DFS traversal, i.e. the pre-order. Then for any two indices i and j such that $1 \leq i < j \leq |\mathcal{T}|$, we will show that maximal r-isolating mincut of $\{v_i, v_j, r\}$ must not contain any vertices in $\{v_i, v_{i+1}, \ldots, v_j\}$ by Lemma 5.5 and hence the result follows by counting the fraction of pairs (at least $1/4$) whose position in the permutation differs by at least $\frac{1}{2}|\mathcal{T}|$.

It remains to show that maximal r-isolating mincut of $\{v_i, v_j, r\}$ is disjoint with $\{v_i, v_{i+1}, \ldots, v_j\}$. There are three cases of the maximal r-isolating mincut cutting the hypercactus: (1) on a normal edge (2) on cycle edges (3) on a hyperedge. The former two cases are identical to the normal graph. For the third case, Lemma 5.5 implies that the maximal r-isolating mincut X_r contains either 1 or $|e| - 1$ vertices of this hyperedge e (not 0 or all the vertices since it cuts through this hyperedge). If X_r contains one node u in this hyperedge, then u is the first one visited by the DFS traversal starting from root r. Therefore in this case, all the nodes in subtree rooted at u are not contained in X_r except u, while the subtree contains the entire set $\{v_i, v_{i+1}, \ldots, v_j\}$, and hence imples the claim. If X_r contains all nodes in this hyperedge except one node u, then the subtree rooted at u are not contained in X_r, and again implies that the set $\{v_i, v_{i+1}, \ldots, v_j\}$ is disjoint with X_r. □

Lemma 5.10. *Let G be the hypergraph with terminal set $\mathcal{T}$ and let (H, ϕ) be a $\mathcal{T}$-Steiner hypercactus of G. Suppose there is a balanced edge-cut on H. Then, with probability $1 - n^{-11}$ there is a balanced split in $\mathcal{S}$ returned from Algorithm 3.*

Proof Sketch. By plugging in Lemma 5.9 into the proof of Lemma 4.13, we get the proof of Lemma 5.10. □

Case 2: No Balanced Cuts. In the second case there is no balanced edge-cut. Note that the definition of irredundant $\mathcal{T}$-Steiner hypercactus H is the same as Definition 4.12, since one can never contract a hyperedge in H without losing Steiner mincuts.

Lemma 5.11. *Let G be a hypergraph with terminal set $\mathcal{T}$ and let (H, ϕ) be an irredundant $\mathcal{T}$-Steiner hypercactus of G. Suppose there is no balanced edge-cut on H, then with probability $1 - n^{-11}$ the following statements holds:*

1. *Either (1) there exists a unique centroid node v on H whose all incident 1-edges and 2-edges from the same cycle corresponds to $\mathcal{T}$-Steiner mincuts of at most $\frac{1}{4}|\mathcal{T}|$ terminals, or (2) there exists a unique hyperedge e on H such that each connected component in $H - e$ corresponds to $\mathcal{T}$-Steiner mincuts of at most $\frac{1}{4}|\mathcal{T}|$ terminals.*
2. *Define $\mathcal{S}'_{\text{pruned}}$ in the same way as in Section 4.3. Let $\mathcal{S} = \{X_i\}$ be the returned collection of $\mathcal{T}$-Steiner mincuts from Algorithm 3. Then $\mathcal{S} = \mathcal{S}'_{\text{pruned}}$. (Recall that $\mathcal{S}'_{\text{pruned}}$ contains all $\mathcal{T}$-Steiner mincuts for each component T_i described in part 1 as long as it contains at least 2 terminals).*
3. *Moreover, the mincut on H that separates $\phi(X_i)$ with the rest vertices is incident to v. That is, after the contraction of all X_i, the contracted graph has a hypercactus that is a star shape with ≥ 4 leaves or a brittle containing ≥ 4 nodes.*
4. *$\mathcal{S}$ is a good split collection.*

Proof.

Part 1. For each hyperedge $e \in H$, replace e by a node v and a bunch of normal edges $\{(u, v) \mid u \in e\}$, and denote this cactus as H'. Then the uniqueness of centriod of H' follows the same proof as Lemma 4.16.

There is no balanced edge-cut on H, then by the definition of balanced edge-cut there must be a centroid node v on H' such that by removing v the cactus H' shattered into connected components. Then centroid node v mapped to a node or a hyperedge in H, corresponds to (1) and (2) respectively.

Part 2. Let $\{T_i\}$ be the partition of all terminal vertices (possibly excluding v if v is non-empty) within each connected components. Then by the assumption where no balanced cut exists, we have $\cup_i T_i = \mathcal{T} \setminus \{v\}, \forall i, |T_i| \leq \frac{1}{4}|\mathcal{T}|$. Similar to Lemma 4.15 (but now the "centroid" v could be either a node or a hyperedge), if three terminal vertices w, x, y are sampled such that $\phi(w), \phi(x)$, and $\phi(y)$ belongs to three distinct connected component in $H - v$, then the maximal w-mincut of $\{w, x, y\}$ (denoted as X_w) has a corresponding mincut in H that is exactly the connected component W of $\phi(w)$ in $H - v$.

It is straightforward to check from definition that any A-mincut of $\mathcal{T}$ will be corresponding to an accessible mincut on H. By Lemma 5.5, our algorithm finds X_w, and it does not contain any vertex outside $\phi^{-1}(W)$.

Now, again by Lemma 5.5, the algorithm always finds accessible mincuts. Note that Fact 4.18 and Proposition 4.19 work for accessible mincuts. Thus, the algorithm ensures that X_w stays maximal in Line 14.

The rest analysis about lower bounding the probability is the same as Lemma 4.17. Hence, with desired probability the returned collection $\mathcal{S}$ contains all desired $\mathcal{T}$-Steiner mincuts.

Part 3. By Part 2, $\mathcal{S} = \mathcal{S}'_{\text{pruned}}$. This is straightforward to check using the fact that H is irredundant. The star case is proved in Lemma 4.22. To bound the number of nodes in the brittle, recall that each split contains strictly less than $\frac{1}{4}|\mathcal{T}|$ terminals, so we conclude that the brittle has at least 5 nodes.

Part 4. By Part 2, $\mathcal{S} = \mathcal{S}'_{\text{pruned}}$. If case (1) in Part 1 is true (i.e., there exists a unique centroid node on H), then the arguments follow exactly the same as in normal graph. Otherwise, there exists a unique hyperedge e on H such that each connected component in $H - e$ corresponds to $\mathcal{T}$-Steiner mincuts of at most $\frac{1}{4}|\mathcal{T}|$ terminals. By Line 15 in Algorithm 3, condition (1) of Definition 4.9 is satisfied.

Now, since all $\mathcal{T}$-Steiner mincuts in $\mathcal{S}$ are disjoint, the decomposition $\{(G_i, \mathcal{T}_i)\}$ induced by $\mathcal{S}$ is well-defined. Without loss of generality we may apply simple refinements to G in the order of $X_1, X_2, \ldots, X_k$. With this ordering, for each $1 \leq i \leq k$, we have $|\mathcal{T}_i| \leq |\mathcal{T} \cap X_i| + 1 \leq \frac{1}{4}|\mathcal{T}| + 1$. Moreover, by Part 3, we know that there exists a cactus representation that is a brittle shape for the very last graph $(G_{k+1}, \mathcal{T}_{k+1})$. Thus, by Definition 5.6, $\{(G_i, \mathcal{T}_i)\}$ is a good decomposition. This implies that condition (2) Definition 4.9 holds and $\mathcal{S}$ is a good split collection. □

Proof of Lemma 5.7. The correctness of Lemma 5.7 follows from Lemma 5.10 and Lemma 5.11. It is straightforward to check (with the proof of Lemma 4.10) that the runtime of Modified Algorithm 3 is $O(\log^3 n) \cdot \text{MaxFlow}(O(n+m), O(p))$. □

5.4 Returning a Correct Hypercactus

In this subsection, we prove Lemma 5.8. In particular, we implement the procedures TrivialCactus$(G, \mathcal{T})$, MergeCactus$(G, \mathcal{T}, \{H_i\})$, and StarOrBrittleCactus$(G, \mathcal{T})$. The first procedure is exactly the same as stated in Section 4.4.

To ensure a correct hypercactus is returned, we implement the procedures of Algorithm 2 with the following invariant:

Invariant 5.12. *Let (H, ϕ) be the $\mathcal{T}$-Steiner hypercactus returned by any subproblem $(G, \mathcal{T})$. Then H is irredundant and does not contain a hollow 3-star as an induced subgraph.*

Star or Brittle Cactus. In this case, there is no $\mathcal{T}$-split that is an accessible $\mathcal{T}$-Steiner mincut. Since $|\mathcal{T}| \geq 4$ when Line 7 is reached in Algorithm 2, we claim that there exists a star or brittle shaped $\mathcal{T}$-Steiner hypercactus of G.

Fact 5.13. *Let G be a hypergraph and $\mathcal{T}$ be a set of terminals with $|\mathcal{T}| \geq 4$. Suppose that every accessible $\mathcal{T}$-Steiner mincut is trivial (i.e., a t-isolating mincut of $\mathcal{T}$ for some $t \in \mathcal{T}$). Then, there exists (H, ϕ), a $\mathcal{T}$-Steiner hypercactus of G such that H is a star graph or a brittle.*

This fact follows from the definition of accessible mincut. Let $A \subset \mathcal{T}$ be arbitrary subset with size $|A| = 2$. To distinguish the star case and brittle case, we simply test the mincut value between A and $\mathcal{T} \setminus A$, which can be done in $O(\text{MaxFlow}(n+m, p))$ time. And contructing the star cactus is the same as Section 4.4. The StarOrBrittleCactus procedure is able to return a $\mathcal{T}$-Steiner hypercactus of G in $O(\log |\mathcal{T}|) \cdot \text{MaxFlow}(O(n+m), O(p))$ time. By assumption 2 and $|\mathcal{T}| \geq 4$, no hollow 3-star can be formed and thus Invariant 5.12 holds.

(Line 11, Case 1.) Merge from a Balanced Split. There are two cases when Line 11 is reached, depending on whether a balanced split is found or not. Suppose that a balanced split is found so the good split collection contains exactly one $\mathcal{T}$-split $|\mathcal{S}| = 1$.

The MergeCactus procedure rely on the following useful observation, which can be proved in the same way as Fact 4.26.

Fact 5.14. *Let $(G, \mathcal{T})$ be a subproblem and let $t \in \mathcal{T}$ be an* anchor vertex *generated in some ancestor problems. Let (H, ϕ) be any irredundant $\mathcal{T}$-Steiner hypercactus of G. Then, the node $\phi(t)$ on H has either degree 1, or degree 2 but in a cycle.*

Now, let us assume that the balanced split in a good split collection $\mathcal{S}$ decomposes the graph G into two subproblems $(G_1, \mathcal{T}_1)$ and $(G_2, \mathcal{T}_2)$, with a shared anchor vertex $a \in \mathcal{T}_1 \cap \mathcal{T}_2$. Let (H_1, ϕ_1) and (H_2, ϕ_2) be the cactus returned from the two subproblems respectively. Using Fact 4.26, there are only a constant situations to be handled, depending on whether $\phi_1(a)$ and $\phi_2(a)$ are leaves (connected to a normal edge), degree 1 nodes on brittle, or on cycle. The **Leaf-Leaf Case**, the **Leaf-Cycle Case**, and the **Cycle-Cycle Case** are implemented in Section 4.4. In addition, we only need to handle the following cases. Note that Invariant 5.12 still holds after these operations.

Brittle-Brittle Case. If both anchor nodes $\phi_1(a)$ and $\phi_2(a)$ have degree 1 and connected to hyperedges in both G_1 and G_2, then we simply make a to be a normal node on hypercactus, and connect G_1 and G_2 via a.

Leaf-Brittle Case. If both anchor nodes $\phi_1(a)$ and $\phi_2(a)$ have degree 1, while one connected to a normal edge and the other connected to a hyperedge, say $(x, \phi_1(a))$ and hyperedge e where $\phi_2(b) \in e$, then the procedure simply delete anchor vertex a and replace hyperedge e by hyperedge $(e \cup \{x\}) \setminus \{a\}$, which connects H_1 and H_2.

Cycle-Brittle Case. If one anchor node connects to a cycle and the other connects to a hyperedge, say $(x, \phi_1(a)), (y, \phi_1(a))$ and hyperedge e where $\phi_2(b) \in e$, then we simply make a to be a normal node on hypercactus, and connect G_1 and G_2 via a.

(Line 11, Case 2.) Merging When There Is No Balanced Split. The other case when invoking Line 11 is that no balanced split exists and thus a good split collection with one or more splits were returned. Suppose that $|\mathcal{S}| = \ell \geq 1$ and the graph is decomposed into $\ell+1$ subproblems $\{(G_i, \mathcal{T}_i)\}$, with the last subgraph $(G_{\ell+1}, \mathcal{T}_{\ell+1})$ containing all the anchor vertices generated in this recursion step. By part 3 of Lemma 5.11 and the algorithm, the returned cactus from the subproblem $(G_{\ell+1}, \mathcal{T}_{\ell+1})$ must be a star graph of at least 4 leaves, or a brittle. Let H_{center} be this graph.

The algorithm attaches each cactus H_i from other subproblems $(G_i, \mathcal{T}_i)$ to the center graph H_{center} via **Leaf-Leaf Case**, **Leaf-Cycle Case**, and **Leaf-Brittle Case** if H_{center} is a star graph; or via **Leaf-Brittle Case**, **Cycle-Brittle Case**, and **Brittle-Brittle Case** if H_{center} is a brittle. Since no tests are required, this step can be done in linear time $O(|V(G)|+|E(G)|)$. The correctness argument is the same as the balanced-split case, since we can view this process as sequentially merging two cactus at a time. With the same reason, Invariant 5.12 holds too.

5.5 Proof of Lemma 5.5

In this section, we prove Lemma 5.5, the structural lemma that was crucial for our algorithm. First, we shows that, although a hyperedge of rank r on a hypercactus implies 2^r mincuts on G, almost all of these mincuts on G have the same set of boundary edges.

Lemma 5.15. *Let $G = (V, E)$ be hypergraph and $\mathcal{T}$ be a terminal vertex set with $|\mathcal{T}| \geq 4$. Suppose that there exists a brittle hypergraph H that is a $\mathcal{T}$-Steiner hypercactus of G. That is, for every proper subset $A \subsetneq \mathcal{T}$, there exists a $\mathcal{T}$-Steiner mincut that separates A and $\mathcal{T} \setminus A$. Then, there exists a unique set of hyperedges $E' \subseteq E$ such that, whenever $|A| \leq |\mathcal{T}| - 2$, E' is the set of boundary hyperedges to the maximal A-mincut X_A of $\mathcal{T}$, i.e. $E' = \partial X_A$.*

Proof. It suffices to prove that: $\partial X_a = \partial X_A$ for all $a \in A \subset \mathcal{T}$ such that $|A| \leq |\mathcal{T}| - 2$, where X_a (resp. X_A) is the maximal a-isolating (resp. A-) mincut of $\mathcal{T}$. We shall first prove a simpler statement $\partial X_a = \partial X_b$ for all $a, b \in \mathcal{T}$.

For any $a, b, c \in \mathcal{T}$, suppose X_{ab}, X_{ac} and $X_{\overline{bc}}$ are the maximal $\{a, b\}$-mincut, maximal $\{a, c\}$-mincut and maximal $\mathcal{T} \setminus \{b, c\}$-mincut of $\mathcal{T}$ respectively. By assumption, $\mathcal{C}(X_{ab}) = \mathcal{C}(X_{ac}) = \mathcal{C}(X_{\overline{bc}}) = \lambda_G(T)$.

Let $Z_a = X_{ab} \cap X_{ac} \cap X_{\overline{bc}}$, $Z_b = (X_{ab} \setminus X_{ac}) \setminus X_{\overline{bc}}$, $Z_c = (X_{ac} \setminus X_{ab}) \setminus X_{\overline{bc}}$ and $Z_{\overline{abc}} = (X_{\overline{bc}} \setminus X_{ab}) \setminus X_{ac}$. Then $\mathcal{C}(Z_a) = \mathcal{C}(Z_b) = \mathcal{C}(Z_c) = \mathcal{C}(Z_{\overline{abc}}) = \lambda_G(T)$ by submodularity and posimodularity (Lemma 2.1). Next we use the proof method similar to Lemma C.6. That is, the following invariant equality always holds.

$$\underbrace{(\mathcal{C}(X_{ab}) + \mathcal{C}(X_{ac}) + \mathcal{C}(X_{\overline{bc}})) - (\mathcal{C}(Z_b) + \mathcal{C}(Z_c) + \mathcal{C}(Z_{\overline{abc}}))}_{\texttt{(LHS)}} = 0 \ .$$

Using the same argument with the proof of Lemma C.6, we have

1. For any hyperedge e, the total contribution of the weight e to the LHS of the equality is non-negative.

2. For any hyperedge e connecting X_a such that e contributes 0 to the LHS of the equality, either e contributes to none of $\mathcal{C}(X_{ab})$, $\mathcal{C}(X_{ac})$ and $\mathcal{C}(X_{\overline{bc}})$, or e contributes to all of $\mathcal{C}(Z_b)$, $\mathcal{C}(Z_c)$, and $\mathcal{C}(Z_{\overline{abc}})$.

Let X_a be the maximal a-isolating mincut of $\mathcal{T}$. First observe that $X_a \subseteq Z_a$, otherwise contradicts to maximality of either X_{ab}, X_{ac} or $X_{\overline{bc}}$ by Nesting & Submodularity Lemma C.1. So X_a must be the connected component in $G[Z_a]$ connected to a, by the definition of maximal $\{a\}$-mincut of $\mathcal{T}$. Therefore $\partial X_a = \partial Z_a$.

For any edge e in ∂Z_a, it always contributes to either $\mathcal{C}(X_{ab})$, $\mathcal{C}(X_{ac})$ or $\mathcal{C}(X_{\overline{bc}})$. By property 2, it will also contributes to all of $\mathcal{C}(Z_b)$, $\mathcal{C}(Z_c)$, and $\mathcal{C}(Z_{\overline{abc}})$, furthermore by the equality, contributes to all of $\mathcal{C}(X_{ab})$, $\mathcal{C}(X_{ac})$ and $\mathcal{C}(X_{\overline{bc}})$. Combining with the fact that $\mathcal{C}(X_{ab}) = \mathcal{C}(Z_a) = \lambda_G(T)$, we have $\partial X_{ab} = \partial Z_a = \partial X_a$.

Since a, b are arbitrary terminals in $\mathcal{T}$, we also have $\partial X_{ab} = \partial X_b$ by swapping a and b in the argument above. Therefore, for any $a, b \in \mathcal{T}$, $\partial X_a = \partial X_b$ and the statement follows.

After replacing b by $A \setminus \{a\}$ and using the same argument, we have $\partial X_a = \partial X_A$ for all $a \in A \subset \mathcal{T}$ such that $|A| \leq |\mathcal{T}| - 2$. □

Lemma 5.16. *Consider a $\mathcal{T}$-Steiner hypercactus (H, ϕ) of G. Let $\mathcal{T}' \subseteq \mathcal{T}$ be a subset of $\mathcal{T}$ with $|\mathcal{T}'| \geq 2$. Let e be a hyperedge on H with rank ≥ 3. Then for all $t \in \mathcal{T}'$ such that the maximal t-isolating mincut X_t of $\mathcal{T}'$ is a $\mathcal{T}'$-Steiner mincut, any corresponding mincut that separates $\phi(X_t \cap \mathcal{T})$ and $\phi(\mathcal{T} \setminus X_t)$ on H includes 0, 1, $|e| - 1$, or $|e|$ nodes in e.*

Proof. For any $u \in e$, let $r(u) = \phi^{-1}(v)$ be a terminal vertex in G where v is some arbitrary fixed node in the connected component of H connected to u when removing e. There exists such v with non-empty pre-image by the definition of hypercactus. Let $\hat{T}_0 = \{r(u) \mid u \in e\}$, the hypercactus of $\hat{T}_0$ is a brittle.

Suppose a contradiction there exists X_t to be a maximal t-isolating mincut of $\mathcal{T}'$, and mincut $(Q, V(H) \setminus Q)$ on H that separates $\phi(X_t \cap \mathcal{T})$ and $\phi(\mathcal{T} \setminus X_t)$, such that $2 \leq |Q \cap e| \leq |e| - 2$. Let $v \in e$ be the node in the same connected component with $\phi(t)$ when removing e from H. Define $\hat{\mathcal{T}} = (\hat{T}_0 \setminus \{r(v)\}) \cup \{t\}$. Then, the hypercactus representation of $\hat{\mathcal{T}}$ is also a brittle. Let $A = \{r(u) \mid u \in (Q \cap e) \setminus \{v\}\} \cup \{t\}$ which is a subset of $\hat{\mathcal{T}}$. Observe that the hyperedge e on H itself must be the *only* mincut that separates $\phi(X_A \cap \mathcal{T})$ and $\phi(\mathcal{T} \setminus X_A)$ (resp. separates Q and $V(H) \setminus Q$). So, for consider any two vertices a, b such that their mapped vertices on H are in the same connected component of $H - e$, we have $a \in X_A$ if and only if $b \in X_A$. Respectively, $a \in X_t$ if and only if $b \in X_t$.

Now we have $A \subseteq X_t$ since $r(u) \in A$ implies that $u \in (Q \cap e)$ so there is some $a \in X_t$ such that $\phi(a)$ is in the same component of $H - e$ with u, further implies that $r(u) \in X_t$. Now we claim that $X_t = X_A$. First, X_t does not contain other terminals in $\hat{\mathcal{T}} \setminus A$, so $X_t \subseteq X_A$ (otherwise, $X_t \cup X_A$ is a larger sized (connected) A-mincut of $\mathcal{T}$ by submodularity). On the other hand, since $A \subseteq X_t$ and that X_A does not contain any terminal in $(\mathcal{T}' \setminus \{t\})$ (otherwise, X_A contains a terminal $a \in \mathcal{T}' \setminus \{t\}$, let $u \in e$ be the node in the same connected component with $\phi(a)$ in $H - e$. Then, $r(u) \in A \subseteq X_t$, which further implies that $a \in X_t$, contradicting to the fact that X_t separates a and t.) By the same submodularity argument we have $X_A \subseteq X_t$.

There exists w be a vertex in A other than t, since $|A| \geq 2$. By Lemma 5.15 ($|A| \leq |e| - 2 = |\hat{\mathcal{T}}| - 2$), $\partial X_A = \partial X_w$ where X_w is the maximal w-isolating mincut of $\hat{T}$, so $\partial X_t = \partial X_A = \partial X_w$. ∂X_w separates w and t by definition, and $\partial X_t = \partial X_w$, contradicts to t and w are connected in $G[X_t]$. □

Lemma 5.16 directly implies Lemma 5.5, since the splits used in the Modified Algorithm 2 are maximal isolating mincuts of some terminal sets $\mathcal{T}' \subseteq \mathcal{T}$.

6 Conclusion and Open Problems

We develop a new approach based on *maximal isolating cuts* for computing the cactus representation of all mincuts that gives the first almost-linear time algorithms for computing Steiner hypercactus, which generalizes both Steiner cactus and hypercactus, each of which generalizes the standard cactus for global edge mincuts.

A natural question is whether our framework works with even more generalized settings than hypergraph connectivity, such as *element connectivity*. Let $U \subseteq V$ be a set of terminals. An element mincut between s and t is the smallest mixed cut $C \subset (E \cup (V - U))$ whose removal disconnects s and t. There exists a hypercactus representation that captures all global element mincuts as well [FJ99] (in the same sense as Steiner cactus captures all Steiner mincuts). Given our result, one can also hope that it admits an almost-linear time construction.

However, element cuts do not fall into the setting of symmetric submodular set functions. Instead, they are captured by a more general notion of *bisubmodular* set functions (see, e.g., [CQ21]). To make our approach works, it seems we need to generalize the notion of posi-modularity for bisubmodular set functions, but it is unclear how to come up with the right definition. More concretely, what is the usable version of the Pairwise Insertion Only Lemma (Lemma 3.3) for element cuts?

Since cactus representation exists for arbitrary symmetric submodular set functions [FJ99], it is also interesting whether there are algorithms with small *query complexity* for cactus construction. Our algorithm carefully decomposes graphs into small pieces and works on each of them separately so that the total running time is almost-linear. This approach that works on small pieces in parallel does not seem to work with the setting for arbitrary symmetric submodular set functions where we count the number of queries.

Acknowledgement

We thank the anonymous reviewers for their constructive suggestions.

References

[ADD+93] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. 1

[AKL+21] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. *CoRR*, abs/2111.04958, 2021. 3, 4

[AKT21] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory–hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1725–1737, 2021. 4

[AKT22] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. APMF < APSP? Gomory-Hu tree for unweighted graphs in almost-quadratic time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1135–1146. IEEE, 2022. 4

[BK96] András A Benczúr and David R Karger. Approximating st minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55, 1996. 1

[BK00] András A Benczúr and David R Karger. Augmenting undirected edge connectivity in $O(n^2)$ time. *Journal of Algorithms*, 37(1):2–36, 2000. 1, 46

[CH03] Richard Cole and Ramesh Hariharan. A fast algorithm for computing steiner edge connectivity. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 167–176, 2003. 2

[Che99] Eddie Cheng. Edge-augmentation of hypergraphs. *Math. Program.*, 84(3):443–465, 1999. 1, 2, 3, 12, 46

[CHLP23] Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Steiner connectivity augmentation and splitting-off in poly-logarithmic maximum flows. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2023. 3

[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. Appeared at FOCS'22. 2

[CLP22] Ruoxu Cen, Jason Li, and Debmalya Panigrahi. Augmenting edge connectivity via isolating cuts. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3237–3252. SIAM, 2022. 1, 4, 46

[CQ21] Chandra Chekuri and Kent Quanrud. Isolating cuts, (bi-)submodularity, and faster algorithms for connectivity. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 50:1–50:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 3, 4, 8, 12, 35, 40

[CS07] Joseph Cheriyan and Mohammad R Salavatipour. Packing element-disjoint steiner trees. *ACM Transactions on Algorithms (TALG)*, 3(4):47–es, 2007. 3

[Cun83] W.H Cunningham. Decomposition of submodular functions. *Combinatorica*, 3:53–68, 1983. 12

[CX17] Chandra Chekuri and Chao Xu. Computing minimum cuts in hypergraphs. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1085–1100. SIAM, 2017. 2, 3, 4, 5, 10, 11, 12, 47

[DKL76] Efim A. Dinits, Alexander V. Karzanov, and Micael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. *Studies in Discrete Optimization*, pages 209–306, 1976. 1

[DN] Yefim Dinitz and Zeev Nutov. Cactus tree type models for families of bisections of a set. 1

[DN95] Yefim Dinitz and Zeev Nutov. A 2-level cactus model for the system of minimum and minimum+ 1 edge-cuts in a graph and its incremental maintenance. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 509–518, 1995. 1

[DV94] Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 716–725. ACM, 1994. 1, 2, 7, 10

[DV00] Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM Journal on Computing*, 30(3):753–808, 2000. 1, 2

[DW98] Ye Dinitz and Jeffery Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20(3):242–276, 1998. 1

[FF09] Tamás Fleiner and András Frank. A quick proof for the cactus representation of mincuts. 2009. 1

[FJ99] Tamás Fleiner and Tibor Jordán. Coverings and structure of crossing families. *Math. Program.*, 84(3):505–518, 1999. 1, 3, 26, 35

[Fle99] Lisa Fleischer. Building chain and cactus representations of all minimum cuts from hao–orlin in the same asymptotic run time. *Journal of Algorithms*, 33(1):51–72, 1999. 1, 2

[Gab91a] Harold N Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 812–821. IEEE Computer Society, 1991. 1, 2

[Gab91b] Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 112–122, 1991. 3

[GHT18] Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in polylogarithmic amortized update time. *ACM Transactions on Algorithms (TALG)*, 14(2):1–21, 2018. 1

[GK19] Rahul Raj Gupta and Sushanta Karmakar. Incremental algorithm for minimum cut and edge connectivity in hypergraph. In *International Workshop on Combinatorial Algorithms*, pages 237–250. Springer, 2019. 2, 3, 47

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

[HKNR98] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998. 1

[Kar93] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30. Citeseer, 1993. 2

[Kar00] David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000. 2, 3

[KP09] David R Karger and Debmalya Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 246–255. SIAM, 2009. 2, 3

[KR13] Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1789–1799. SIAM, 2013. 1

[KR14] Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014. 1

[KS96] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996. 2

[KT86] Alexander V Karzanov and Eugeniy A Timofeev. Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. *Cybernetics*, 22(2):156–162, 1986. 2

[KW96] Regina Klimmek and Frank Wagner. A simple hypergraph min cut algorithm. 1996. 2

[LM10] F Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 47–56, 2010. 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. 4

[LNPS22] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Fair cuts, approximate isolating cuts, and approximate gomory-hu trees in near-linear time. *arXiv preprint arXiv:2203.00751*, 2022. 4

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 85–92. IEEE, 2020. 3, 4, 6, 8, 12, 40, 46

[LP21] Jason Li and Debmalya Panigrahi. Approximate gomory–hu tree is faster than n–1 max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1738–1748, 2021. 4, 12, 13, 40

[LPS22] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1124–1134. IEEE, 2022. 4

[Meh88] Kurt Mehlhorn. A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988. 3

[MN21] Sagnik Mukhopadhyay and Danupon Nanongkai. A note on isolating cut lemma for submodular function minimization. *arXiv preprint arXiv:2103.15724*, 2021. 4

[Moi09] Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 3–12. IEEE, 2009. 1

[MW00] Wai-Kei Mak and DF Wong. A fast hypergraph min-cut algorithm for circuit partitioning. *Integration*, 30(1):1–11, 2000. 2

[NGM97] Dalit Naor, Dan Gusfield, and Charles Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997. 1

[NI00] Hiroshi Nagamochi and Toshihide Ibaraki. Polyhedral structure of submodular and posi-modular systems. *Discret. Appl. Math.*, 107(1-3):165–189, 2000. 5

[NK94] Hiroshi Nagamochi and Tiko Kameda. Canonical cactus representation for minimum cuts. *Japan Journal of Industrial and Applied Mathematics*, 11(3):343–361, 1994. 2, 11, 16, 23

[NNI00] Hiroshi Nagamochi, Yoshitaka Nakao, and Toshihide Ibaraki. A fast algorithm for cactus representations of minimum cuts. *Japan journal of industrial and applied mathematics*, 17(2):245–264, 2000. 2

[NV91] Dalit Naor and Vijay V Vazirani. Representing and enumerating edge connectivity cuts in rnc. In *Workshop on Algorithms and Data Structures*, pages 273–285. Springer, 1991. 2

[ST11] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. 1

A Omitted Proofs from Section 3

A.1 Proof of Lemma 3.2

If $X_A \cup X_B = V$, then it implies that $\mathcal{C}(X_A) = \mathcal{C}(V \setminus X_A) = \mathcal{C}(X_B \setminus X_A) \geq \mathcal{C}(X_B) = \mathcal{C}(X_A \setminus X_B) \geq \mathcal{C}(X_A)$ since the complement of X_A is a B-cut of $\mathcal{T}$ and the complement of X_B is an A-cut of $\mathcal{T}$. Suppose that $X_A \cup X_B \subsetneq V$. By posi-modularity we have

$$\mathcal{C}(X_A) + \mathcal{C}(X_B) \geq \mathcal{C}(X_A \setminus X_B) + \mathcal{C}(X_B \setminus X_A).$$

Since A and B are disjoint, $X_A \setminus X_B$ is an A-cut and $X_B \setminus X_A$ is a B-cut. Hence, we have $\mathcal{C}(X_A \setminus X_B) \geq \mathcal{C}(X_A)$ and $\mathcal{C}(X_B \setminus X_A) \geq \mathcal{C}(X_B)$. Combining with the posi-modularity we have $\mathcal{C}(X_A) = \mathcal{C}(X_A \setminus X_B)$ and $\mathcal{C}(X_B) = \mathcal{C}(X_B \setminus X_A)$ as desired.

□

A.2 Proof of Lemma 3.5

By the submodularity of cut value function in Lemma 2.1,

$$\mathcal{C}(X_A) + \mathcal{C}(X_B) \geq \mathcal{C}(X_A \cap X_B) + \mathcal{C}(X_A \cup X_B) \ .$$

Since both X_A and $X_A \cap X_B$ are A-cuts and X_A is A-mincut, we have $\mathcal{C}(X_A) \leq \mathcal{C}(X_A \cap X_B)$. Similarly, both X_B and $X_A \cup X_B$ are B-cuts and X_B is B-mincut implies $\mathcal{C}(X_B) \leq \mathcal{C}(X_A \cup X_B)$. Combining with the submodularity, we have $\mathcal{C}(X_A) = \mathcal{C}(X_A \cap X_B)$ and $\mathcal{C}(X_B) = \mathcal{C}(X_A \cup X_B)$. Therefore, $X_A \cap X_B$ is a A-mincut of $\mathcal{T}$, and $X_A \cup X_B$ is a B-mincut of $\mathcal{T}$.

□

B Omitted Proofs from Section 4

B.1 Proof of Lemma B.1

We show how to perform preprocessing in hypergraph, which implies Lemma 4.7 as a corollary for normal graph.

Lemma B.1 (Preprocessing)**.** *Given a hypergraph G and a terminal set $\mathcal{T}$, there exists an algorithm such that, with probability $1 - n^{-11}$ the algorithm outputs a partition of $\mathcal{T}$ such that $\lambda(u, v) = \lambda_G(\mathcal{T})$ if and only if u and v belongs to different parts. This algorithm runs in $O(\log^2 n) \cdot \text{MaxFlow}(O(n + m), O(p))$ time.*

Proof. The algorithm is exactly the same as invoking one step in Algorithm 4 (CutThresholdStep with post-processing) in [LP21] with the slightest modification. The algorithm works as follows. The algorithm first computes $\lambda_G(\mathcal{T})$, the value of $\mathcal{T}$-Steiner mincut on G using Chekuri and Quanrud's algorithm [CQ21]. Then, the algorithm samples each terminal vertex with different sampling rates 2^{-i} for all $i = 0, 1, 2, \ldots, 2^{-\lceil \log |\mathcal{T}| \rceil}$. For each sampled vertex set $\mathcal{T}_i$ from the sampling rate 2^{-i}, the algorithm computes the *minimal* isolating mincut using the Isolating Cut Lemma [LP20], and keeps only the mincut having the same value as $\lambda_G(\mathcal{T})$. Repeat the sampling procedure for $\Theta(\log n)$ times for ensuring that with high probability, every vertex v obtains a minimal $\mathcal{T}$-Steiner mincut that contains v, as long as the size of this mincut is at most $|\mathcal{T}|/2$. Let $\mathcal{T}_{\text{large}}$ be the set of all terminal vertices that does not obtain such a mincut. Then these $\mathcal{T}$-Steiner mincuts together with $\mathcal{T}_{\text{large}}$ *is* a partition of $\mathcal{T}$ that we are looking for. □

C Proof of Maximal Isolating Mincuts on Hypergraphs

In this section, we prove Theorem 5.4. First, we verify that The Nesting & Submodularity property also holds in hypergraphs, and leads to the uniqueness of maximal and minimal A-mincuts of $\mathcal{T}$ in a hypergraph.

Lemma C.1 (Nesting & Submodularity on Hypergraph)**.** *Let G be a hypergraph and let $\mathcal{T}$ be the set of terminals. Consider two nonempty subsets A and B of terminals such that $A \subseteq B \subsetneq \mathcal{T}$. Let X_A (resp. X_B) to be any A-mincuts (resp. B-mincuts) of $\mathcal{T}$. Then, $\mathcal{C}(X_A \cap X_B) = \mathcal{C}(X_A)$. Respectively, $\mathcal{C}(X_A \cup X_B) = \mathcal{C}(X_B)$, and $X_A \cup X_B$ is a B-mincut of $\mathcal{T}$.* □

The difference of Lemma C.1 with Lemma 3.5 in a normal graph is that we cannot say $X_A \cap X_B$ is A-mincut of $\mathcal{T}$ but only their cut values are the same, since $(G/A)[X_A \cap X_B]$ may not be connected and hence violates Definition 5.3.

The proof of Nesting & Submodularity Lemma is almost identical as in proof of Lemma 3.5 since the cut value function $\mathcal{C}$ also preserves submodularity in hypergraph, except that we need to argue $X_A \cup X_B$ is a B-mincut of $\mathcal{T}$ which is straightforward to verify that $(G/B)[X_A \cup X_B]$ is connected. Fortunately, Definition 5.3 does not affect the desired properties of maximal and minimal A-mincuts.

Lemma C.2. *For any $A \subseteq \mathcal{T}$, there exists a unique maximal (resp. minimal) A-mincut of $\mathcal{T}$.*

Proof. Suppose by contradiction X_A and X'_A are two different maximal A-mincuts of $\mathcal{T}$. By Nesting & Submodularity Lemma C.1, $X_A \cup X'_A$ is also a A-mincut of $\mathcal{T}$, contradicting the maximality of X_A and X'_A.

Suppose by contradiction X_A and X'_A are two different minimal A-mincuts of $\mathcal{T}$. Again by Nesting & Submodularity Lemma C.1, we have $\mathcal{C}(X_A \cap X'_A) = \mathcal{C}(X_A)$. Suppose r_A is the vertex contracted from A in $(G/A)[X_A \cap X'_A]$. Let Y be the connected component in $(G/A)[X_A \cap X'_A]$ connected to r_A, and $X = Y \cup A \setminus \{r_A\}$ which is the corresponding set of vertices in G. Then the boundary edges $\partial X \subseteq \partial(X_A \cap X'_A)$. Therefore, $\mathcal{C}(X) \leq \mathcal{C}(X_A \cap X'_A) = \mathcal{C}(X_A)$ implies that X is a also a A-mincuts of $\mathcal{T}$, contradicting the minimality of X_A and X'_A. □

The reason that we want $(G/A)[X_A]$ to be connected in A-mincut of T is because this definition leads to Pairwise Intersection Only Lemma on hypergraphs. Recall that the proof of Lemma 3.3 in a normal graph aims to show that there is no edge between $X = X_A \cap X_B \cap X_C$ and $X_A \setminus X$, contradicts to the connectivity of $(G/A)[X_A]$. The proof for hypergraph is a bit different, since there may exist a hyperedge connecting X and $X_A \setminus X$. Fortunately, we will show that these hyperedges can only be cut edges of X_A and hence removed by Definition 5.3 when we analyze the connectivity of $(G/A)[X_A]$.

The Disjoint & Posi-modularity property also holds in a hypergraph, and serves as the key to prove Pairwise Intersection Only Lemma on hypergraph. We need the definition of relaxed A-mincut before showing the Disjoint & Posi-modularity for hypergraph.

Definition C.3. Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$. For any proper subset of terminals $A \subsetneq \mathcal{T}$, a cut X is a *relaxed A-cut* of T if it satisfies the first conditions of A-cut in Definition 5.3, i.e., the cut $(X, V \setminus X)$ separates A and $\mathcal{T} \setminus A$, with $A \subseteq X$. A *relaxed A-mincut* of $\mathcal{T}$ is a minimum valued relaxed A-cut of $\mathcal{T}$.

Lemma C.4. *Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$. For any proper subset of terminals $A \subsetneq \mathcal{T}$, the value of A-mincut equals the value of relaxed A-mincut.*

Proof. The "$\geq$" direction is straightforward. For the other direction the proof is similar to the uniqueness of minimal A-mincut. We suppose a contradiction that there exists a relaxed A-mincut X_A such that $\mathcal{C}(X_A)$ is strictly smaller than the value of A-mincut. Suppose r_A is the vertex contracted from A in $(G/A)[X_A \cap X'_A]$. Let Y be the connected component in $(G/A)[X_A \cap X'_A]$ connected to r_A, and $X = Y \cup A \setminus \{r_A\}$ which is the corresponding set of vertices in G. Then the boundary edges $\partial X \subseteq \partial(X_A)$. Therefore, $\mathcal{C}(X) \leq \mathcal{C}(X_A)$ which is smaller than the value of A-mincut, contradicting the definition of A-mincut. □

The proof of Disjoint & Posi-modularity Lemma is the same as the normal graph since the cut value function $\mathcal{C}$ also preserves posi-modularity in a hypergraph.

Lemma C.5 (Disjoint & Posi-modularity)**.** *Let $A, B \subseteq \mathcal{T}$ be two nonempty subsets of terminals with $A \cap B = \emptyset$. Let X_A (resp. X_B) be an relaxed A-mincut (resp. relaxed B-mincut) of $\mathcal{T}$. Then, $X_A \setminus X_B$ is a relaxed A-mincut of $\mathcal{T}$, and $X_B \setminus X_A$ is a relaxed B-mincut of $\mathcal{T}$.* □

Lemma C.6 (Pairwise Intersection Only on Hypergraph)**.** *Given a hypergraph G, let $A, B, C \subseteq \mathcal{T}$ be three disjoint nonempty subsets of terminals. Let $X_A, X_B, X_C \subseteq V$ be any A-mincut of $\mathcal{T}$, B-mincut of $\mathcal{T}$, and C-mincut of $\mathcal{T}$ respectively. Then $X_A \cap X_B \cap X_C = \emptyset$.*

Proof. The proof is only interesting whenever the intersection of any two isolating mincuts is non-empty (i.e. crossing). Thus, without loss of generality, we assume that $X_A \cap X_B \neq \emptyset$, $X_B \cap X_C \neq \emptyset$, and $X_C \cap X_A \neq \emptyset$.

Define $X'_A = (X_A \setminus X_B) \setminus X_C$, $X'_B = (X_B \setminus X_C) \setminus X_A$ and $X'_C = (X_C \setminus X_A) \setminus X_B$. These sets are non-empty since $A \subseteq X'_A$, $B \subseteq X'_B$, and $C \subseteq X'_C$. By posi-modularity (Lemma C.5) we know that X'_A (resp. X'_B and X'_C) is relaxed A-mincuts (resp. B and C).

Assume for contradiction that $X := X_A \cap X_B \cap X_C \neq \emptyset$. We now aim to prove that there is no path from X to X'_A in the induced graph $(G/A)[X_A]$, which leads to a contradiction to Definition 5.3. Similar as the proof for the normal graph, we consider the equality

$$\underbrace{(\mathcal{C}(X_A) + \mathcal{C}(X_B) + \mathcal{C}(X_C)) - (\mathcal{C}(X'_A) + \mathcal{C}(X'_B) + \mathcal{C}(X'_C))}_{\texttt{(LHS)}} = 0 \ .$$

Note that X'_A (resp. X'_B and X'_C) is relaxed A-mincut (resp. B and C) while X_A (resp. X_B and X_C) is A-mincut (resp. B and C), the equality still holds since Lemma C.4.

Next, it suffices to prove the following combinatorial property.

1. For any hyperedge e, the total contribution of the weight e to the LHS of the equality is non-negative.

2. For any hyperedge e connecting X such that e contributes 0 to the LHS of the equality, either e contributes to none of $\mathcal{C}(X_A), \mathcal{C}(X_B)$ and $\mathcal{C}(X_C)$, or e contributes to all of $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$ and $\mathcal{C}(X'_C)$.

To see why it is sufficient, by first property, no hyperedge contributes positive to LHS of the equality. And by the second property, only these two kinds of hyperedges can be connecting to X. For the first case, e contributes to none of $\mathcal{C}(X_A), \mathcal{C}(X_B)$ and $\mathcal{C}(X_C)$ means that e only contains vertex in X. For the second case, e is not in the induced graph $(G/A)[X_A]$ by definition of induced subhypergraph and hence X is not connected with $X_A \setminus X$ in the induced graph, contradicts to the connectivity assumption in Definition 5.3. It remains to prove the two properties.

1. We further consider the cases that e contributes to how many terms of $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$ and $\mathcal{C}(X'_C)$. There is nothing to prove if e contributes to none of them.

   If e contributes to one of them, without loss of generality $\mathcal{C}(X'_A)$, then e contains some vertex in X'_A, and also some vertex v in $V \setminus X'_A = (V \setminus X_A) \cup X_B \cup X_C$. Therefore v is in either $V \setminus X_A$, X_B or X_C, hence also contributes to either $\mathcal{C}(X_A), \mathcal{C}(X_B)$ or $\mathcal{C}(X_C)$ respectively.

   If e contributes to two of them, without loss of generality $\mathcal{C}(X'_A)$ and $\mathcal{C}(X'_B)$, then e contains some vertex in X'_A and in X'_B. So e also contributes to $\mathcal{C}(X_A)$ and $\mathcal{C}(X_B)$.

Otherwise e contributes to all the three terms $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$ and $\mathcal{C}(X'_C)$, then it is direct to see that e also contributes to $\mathcal{C}(X_A), \mathcal{C}(X_B)$ and $\mathcal{C}(X_C)$.

2. Similar to the proof of the first property, we consider the cases that e contributes to how many terms of $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$ and $\mathcal{C}(X'_C)$. This time e must contain some vertex in X.

   If e contributes to none of them, the it also contributes to none of $\mathcal{C}(X_A), \mathcal{C}(X_B)$ and $\mathcal{C}(X_C)$ since the total contribution is 0, which satisfies the claim.

   If e contributes to one of them, without loss of generality $\mathcal{C}(X'_A)$, then e contains some vertex in X'_A. Since e also contains vertex in X, it contributes to both $\mathcal{C}(X_B)$ and $\mathcal{C}(X_C)$, which gives a positive total contribution to LHS. So this case cannot happen.

   If e contributes to two of them, without loss of generality $\mathcal{C}(X'_A)$ and $\mathcal{C}(X'_B)$, then e contains some vertex in X'_A and in X'_B. Since e also contains vertex in X, it contributes to all of $\mathcal{C}(X_A), \mathcal{C}(X_B)$ and $\mathcal{C}(X_C)$, which also gives a positive total contribution to LHS. So this case cannot happen.

   Otherwise e contributes to all the three terms $\mathcal{C}(X'_A), \mathcal{C}(X'_B)$ and $\mathcal{C}(X'_C)$, which satisfies the claim. □

Bounding the output size. The total size of all maximal v-isolating mincuts on hypergraph is also $O(n)$, as a consequence of Lemma C.6. This results and its proof is the same as normal graph:

Lemma C.7. *Let G be a hypergraph and $\mathcal{T}$ be a set of terminals. For each $v \in \mathcal{T}$, let X_v be the maximal v-isolating mincut. Then, $\sum_{v\in\mathcal{T}} |X_v| \le 2n$.* □

A Divide and Conquer Algorithm. The maximal isolating mincut algorithm is exactly the same as Algorithm 1, and we shall reduce computing the max-flow and s-minimal (resp. maximal) s-t mincut on hypergraph to compute the max-flow on normal graph.

Compute s-Maximal st-Mincut on a Hypergraph via Max-Flow. Given a hypergraph $G = (V, E)$, a source s and a sink t, the hypergraph can also be viewed as a bipartite graph $G' = (U', V', E')$ such that $U' = V, V' = E$ and $(u, e) \in E'$ iff $u \in e$. The flow network in the hypergraph G can be viewed as flow network in the bipartite graph G', with vertex capacity on $e \in V'$. So we can compute the max-flow of G via computing the vertex capacity max-flow of graph G'.

The s-maximal s-t mincut can be obtained by a post-processing of max-flow. The algorithm first computes the s-t maxflow, and then examines the residue flow graph $G'^{(f)}_{\text{flow}}$: the s-minimal s-t mincut is the set of vertices in U reachable from s, and the s-maximal s-t mincut can be obtained by first computing $X \subset U$ to be the set of vertices not reachable to t, then find the connected component of the induced subhypergraph $G[X]$ connected by s.

The correctness proof of Algorithm 1 follows the same proof of Lemma 3.6, by replacing the Nesting & Submodularity Lemma using the hypergraph version Lemma C.1.

Lemma C.8 (Correctness)**.** *Fix a terminal vertex $v \in \mathcal{T}$. There is a unique (leaf) subproblem where the maximal isolating mincut for v is computed. Let $\hat{X}_v$ be the cut returned at Line 6 of Algorithm 1 for vertex v. Then $\hat{X}_v = X_v$ is the maximal v-isolating mincut on the hypergraph G.* □

Lemma C.9 (Runtime)**.** *MaxIsoMincut$(G, \mathcal{T})$ runs in time $O(\log|\mathcal{T}|)\cdot\text{MaxFlow}(O(n+m), O(p))$ time on hypergraph G.*

Proof. It suffices to bound the sum of graph sizes in all subproblems throughout the execution of Algorithm 1. First of all, the maximum depth of the recursion tree is $\lceil\log|\mathcal{T}|\rceil$ since in each recursive call the number of non-pivot terminals is reduced to half. In addition, the number of subproblems in each recursion depth i is at most $2^i \le |\mathcal{T}|$.

Now, we focus on a particular recursion depth $i > 0$. Let $\{(G_j, \mathcal{T}_j, r_j)\}$ (r_j denote the pivot vertex in hypergraph to avoid ambiguity) be all the subproblems whose recursion depth is i. We observe that all terminals except pivot never goes to two subproblems so the subsets $\mathcal{T}'_j := \mathcal{T}_j \setminus \{r_j\}$ are disjoint. Moreover, by Lines 11-12 we know that for each j, removing the pivot r_j from G_j it is exactly the maximal $\mathcal{T}'_j$-mincut of $\mathcal{T}$ on G.

Using the Pairwise Intersection Only Lemma on Hypergraph (Lemma C.6), we are able to conclude that every vertex in the input graph G occurs in at most two subproblems at recursion depth i. Therefore, the total number of vertices across all subproblems at depth i is $\sum_j |V(G_j)| \le 2n + 2^i \le 2n + |\mathcal{T}| \le 3n$.

Next we bound the total volume of edges across all subproblems at depth i.

$$\begin{aligned}
\sum_j \sum_{e\in E(G_j)} |e| &= \sum_j \sum_{e\in E(G_j)} \big|\{u \in e \mid u \in V(G_j)\}\big| \\
&\le \sum_j \sum_{e\in E(G_j)} 2\big|\{u \in e \mid u \in V(G_j)\setminus\{r_j\}\}\big| \\
&\le 2\sum_{u\in V(G)} 2d(u) \\
&= 4p\ .
\end{aligned}$$

The last inequality is because every vertex in the input graph G occurs in at most two subproblems at recursion depth i.

Therefore, in each subproblem $(G', \mathcal{T}')$, where the graph G' has n' vertices and m' edges with total volume p', the algorithm computes maximal A-mincut (and B-mincut) of $\mathcal{T}$ in $\text{MaxFlow}(O(n'+m'), O(p'))$ time. By summing up the runtime per recursion depth, we obtain an upper bound to the desired total runtime $O(\log|\mathcal{T}|)\cdot\text{MaxFlow}(O(n+m), O(p))$. □

Proof of Theorem 5.4. Theorem 5.4 follows directly by Algorithm 1, Lemma C.8, and Lemma C.9. □

D Omitted Proofs from Section 5.2

D.1 Proof from Theorem 5.2

The proof is almost the same as Theorem 4.2, except that we need to bound the total volume of hyperedges instead of only the number of normal edges.

Runtime. To analyze the total volume of edges and hyperedges across all subproblems within the same recursion depth, we bound normal edges and hyperedges separately.

For normal edges (rank-2 edges), the argument is the same as Theorem 4.2. after computing an induced decomposition from a good split collection, the total number of edges is increased by at most $\sum_{i=1}^{\ell} |V(G_i)| \leq 2n$ (notice that we charge the number of the newly generated edges in the last decomposed graph $(G_{\ell+1}, \mathcal{T}_{\ell+1})$ to the edges across each split). Hence, we know that at any recursion depth there are at most $m + 2n \log |\mathcal{T}|$ edges in total. Here n refers to the total number of vertices in the input graph, and m refers to the total number of vertices in the input graph.

For hyperedges of rank larger than 2, we use a potential method similar to the vertices number analysis of Theorem 4.2. Let $E' = \{e \in E, |e| \geq 3\}$ to be the set of hyperedges and $p'_G := \sum_{e \in E'} |e|$ to be the total volume of hyperedges. Define $\Phi(G) := 3p'_G - 6|E'(G)|$. By definition $p'_G \geq 3|E(G)| \geq 0$, so $\Phi(G) \geq 0$. Consider the induced decomposition $\{(G_i, \mathcal{T}_i)\}$ on a good split collection of size k. By definition of simple refinements, we know that $\Phi(G, \mathcal{T}) = \sum_{i=1}^{k+1} \Phi(G_i, \mathcal{T}_i)$. Therefore, the sum of all potentials within the same recursion depth can be upper bounded by the root problem's potential. Since the total volume of any subproblem $p'_{G'} \leq 3p'_{G'} - 6|E'(G')| = \Phi(G')$ by definition, we conclude that the total volume of hyperedges across all subproblems at any particular recursion depth (or any collection of subproblems that are not related to each other) is at most $\Phi(G) \leq 3p$. Here p refers to the total volume of hyperedges in the input graph.

Finally, we add up the runtime needed per recursion depth. Fix any recursion depth, for each subproblem $(G_j, \mathcal{T}_j)$, by Lemma 5.7 the runtime spent in Line 5 is at most $O((\log^3 n) \cdot \text{MaxFlow}(O(|V(G_j)| + |E(G_j)|), O(p_{G_j}))$, the runtime spent for Line 9 is linear in the graph size $O(|V(G_j)| + p_{G_j})$, and by Lemma 5.8 merging hypercactus takes $O(\log |\mathcal{T}_j| \cdot \text{MaxFlow}(O(|V(G_j)| + |E(G_j)|), O(p_{G_j}))$ time. Hence, combining all the subproblems together and using the bound of total volume of edges, the runtime of Algorithm 2 is $O(\log^4 n) \cdot \text{MaxFlow}(O(n+m), O(p + n \log |T|))$.

Correctness. By Lemma 5.7, with probability $1 - n^{-11}$ the returned collection is good in Line 5. By the same analysis in the proof of Theorem 4.2, we know that there are at most $4|\mathcal{T}|$ subproblems. Hence the algorithm makes at most $4|\mathcal{T}|$ invocations to Lemma 5.7. With a union bound, we know that with probability $1 - 4n^{-10}$ the collections of splits from all subproblems are good. Now, by applying the union bound again to Lemma 5.8 we know that the returned hypercactus is a $\mathcal{T}$-Steiner hypercactus of G with probability at least $1 - 4n^{-10}$. Therefore, with another union bound we know that with probability $1 - 8n^{-10}$ Algorithm 2 correctly output a $\mathcal{T}$-Steiner hypercactus. □

E Applications

E.1 Steiner Connectivity Edge Augmentation Problem

Given an unweighted undirected graph G, a terminal set $\mathcal{T}$, and a target edge connectivity τ. The goal of the *Steiner connectivity edge augmentation problem* is to find a set of edges to G whose addition makes the value of a $\mathcal{T}$-Steiner mincut to be at least τ.

In this subsection, we briefly describe how to solve the Steiner connectivity edge augmentation problem, i.e., proving the first part of Corollary 1.3.

Corollary 1.3. *There are randomized almost-linear time algorithms that can w.h.p. compute*

- *the optimal solution of the Steiner connectivity augmentation problem.*

Cen et al. [CLP22] gives an algorithm that solves the *edge connectivity augmentation problem*, which is a special case to the Steiner connectivity edge augmentation problem with $\mathcal{T} = V$. Their algorithm utilizes Isolating Cut Lemma [LP20] and constructs a special hierarchy of vertex subsets called *extreme set tree*. With the extreme set tree, the algorithm follows the framework of Benczúr and Karger [BK00] that solves the *degree-constrained edge connectivity problem* (DECA) given an extreme set tree. The framework [CLP22, Section 3] consists of 3 phases:

1. Using external augmentation, transform the degree constraints $\beta(v)$ to *tight* degree constraints $b(v)$ for all $v \in V$.
2. Repeatedly add an augmentation chain to increase connectivity to at least $\tau - 1$.
3. Add a matching defined on the $\mathcal{T}$-Steiner cactus if the connectivity does not reach τ.

Notice that the first two phases can be easily constructed in the Steiner case. The third phase requires a computation of a $\mathcal{T}$-Steiner cactus of G. By our new Steiner cactus algorithm (Theorem 1.1), we are able to obtain an almost-linear time algorithm for DECA, and hence solving the Steiner connectivity augmentation problem.

E.2 Hypergraph +1-Steiner-Connectivity Augmentation Problem

Given a hypergraph $G = (V, E)$ and a terminal set $\mathcal{T} \subset V$, the goal of the *hypergraph +1-Steiner-connectivity augmentation problem* is to compute a minimum sized set E' of rank-2 edges such that $\lambda_{G \cup E'}(\mathcal{T}) \geq \lambda_G(\mathcal{T}) + 1$. The *optimal value* is defined to be the minimum possible $|E'|$. In this subsection we briefly describe the proof to the second part of Corollary 1.3.

Corollary 1.3. *There are randomized almost-linear time algorithms that can w.h.p. compute*

- *the optimal value of the hypergraph +1-Steiner-connectivity augmentation problem.*

Cheng [Che99] provides a formula for computing the optimal value of the hypergraph +1-connectivity augmentation problem, the value can be computed in linear time once we obtain a cactus (for all global mincuts) of G. Below, we describe Cheng's result in the Steiner setting, thus solving the hypergraph +1-Steiner-connectivity augmentation problem:

Theorem E.1 ([Che99])**.** *Given G be a hypergraph and $\mathcal{T}$ be a terminal sets, let H be the irredundant $\mathcal{T}$-Steiner hypercactus of G. Let $\alpha_G(\mathcal{T})$ be the size of the largest hyperedge in H, and $\beta_G(\mathcal{T})$ be the number of degree 1 nodes in H. Then the optimal value of +1-Steiner-connectivity augmentation is*

$$\max\left\{\alpha_G(\mathcal{T}) - 1, \left\lceil \frac{\beta_G(\mathcal{T})}{2} \right\rceil \right\} .$$

Therefore, with our new Steiner hypercactus algorithm (Theorem 1.2), the optimal value of the hypergraph +1-Steiner-connecitivty augmentation problem can be solved in almost-linear time.

E.3 Incremental Algorithm for Hypergraph Mincuts

Given a sequence of (unweighted) hyperedges inserting to an initially empty hypergraph G, the *incremental hypergraph mincut problem* requires the data structure to correctly maintain the $\lambda(G)$ subject to this sequence of insertions.

Gupta and Karmakar [GK19] gives an algorithm that solves the incremental hypergraph mincut problem in $O(\lambda n)$ amortized update time. In this subsection, we show that our hypergraph cactus algorithm (Theorem 1.2) can be used for substituting the cactus construction step and thus improving the amortized update time to the algorithm.

Corollary 1.4. *There is an algorithm that, given an unweighted hypergraph $G = (V, E)$ undergoing hyperedge insertions, maintains a mincut in time $\widehat{O}(\lambda)$ amortized update time where λ denotes the mincut value at the end of the updates.*

The algorithm by Gupta and Karmakar [GK19] proceeds in phases. A new phase starts whenever $\lambda(G)$ increases. Within a phase, they spend $T_{\text{cactus}} + \tilde{O}(n + \sum_e |e|)$ time where the sum is over all edges inserted in this phase. By using previous hypercactus construction of Chekuri and Xu's algorithm [CX17], they obtained an $\tilde{O}(\lambda(G)n)$ amortized update time where $\lambda(G)$ denotes the mincut value after all the updates. Now, if we use our almost-linear time construction, the time per phase is $\widehat{O}(p)$ where p is the total size of the hypergraph at the end of the phase. After $\lambda(G)$ phases, the total update time is then $\widehat{O}(p\lambda(G))$, which implies $\widehat{O}(\lambda(G))$ amortized update time.